A DISTRIBUTED AUDITORY SCENE SIMULATION SOFTWARE SYSTEM

A Master's Thesis
Presented to the
Department of Computer and Information Science
State University of New York Institute of Technology
at Utica/Rome

In Partial Fulfillment
of the Requirements for the
Master of Science Degree

by
Matt Gentner

April 1998

SUNY INSTITUTE OF TECHNOLOGY
AT UTICA/ROME

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE OF APPROVAL

Approved and recommended for acceptance as
a thesis in partial fulfillment of the requirements
for the degree of Master of Science in
computer and information science

May 6, 1998
Date

Rosemary J. Mullick

Rosemary Mullick, Ph.D.
Advisor

[illegible]

Saumendra Sengupta, Ph.D.

[illegible]

Scott Spetka, Ph.D.

ABSTRACT

The purpose of this thesis is to research and apply supporting contemporary software development technologies to develop an auditory scene simulator. Auditory scene simulation means producing the subjective effects of attenuation and delay accurately for arbitrary spatial arrangements of sound sources. An operational simulator prototype is included and described in both stand-alone and distributed software configurations.

The simulator is object oriented, written in the Java and C++ programming languages. Application of software architectural design patterns such as Model-View-Controller, Delegate, and Factory is made within the simulator, and described in this paper. The Common Object Request Broker Architecture (CORBA) was followed while building the included auditory scene simulator. CORBA facilitates integration of the simulator's various Java and C++ software subsystems, and provides potential for distributed computing capabilities.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Chapter 1: Introduction

The Soundscape Auditory Scene Simulator is a prototypical software tool, developed for this thesis by the author. It accurately simulates both attenuation and delay of spatially oriented sound sources. The simulator renders graphical and/or binary file representations of the subjective experience one would expect at the center of an auditory scene. As an experimentalists' tool, the Soundscape Auditory Scene Simulator enables users to quickly add and configure models of:

- echoing surfaces
- periodic wave forms
- moving sound sources

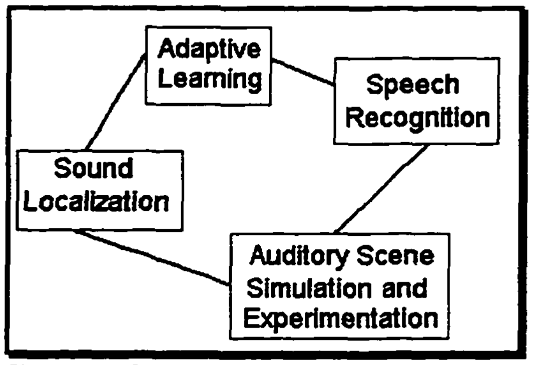


Figure 1: Related Topics Association

Auditory scene simulation is a supporting field of study to other contemporary and challenging topics in information technologies. Applications such as urban gunshot detection, ground seismic reporting, surveillance, and aid of hearing impairment may benefit from support provided by auditory scene simulation[1].

Localization of sound sources[2] is a sensation taken for granted by people (and animals) with normal hearing. Somehow, our minds and ears "lock onto" sources of sound as separate meaningful streams of information automatically. It is commonly believed that part of sound localization is achieved by phasing the input of the right and left ears to achieve an azimuth bias in the direction of an interesting sound. Yet, how the sound became interesting in the first place (without sound localization) remains an open question. The Soundscape auditory scene simulator is designed to provide experimental data which may help to explain the mysterious and complex process of sound localization.

[1] See R. Showen, J. Dunham, A.C. Phillips, S. Sandoval: "World's First Proven Gunshot Location System!" Trilon Technology (http://www.shotspotter.com); Los Altos, California 1998.
[2] See R. Duda: "Sound Localization Research" San Jose State University College of Engineering (http://www-engr.sjsu.edu/~duda/Duda.Research.html); San Jose, California 1998.

Progress in computer systems' cost-to-performance ratio make personal experimentation with auditory scenery feasible. And, very recent progress in software engineering has been applied in this thesis work to produce a reliable and accurate simulator. One which can perform efficiently on a single personal computer, or be deployed as a distributed software system over a network of computers.

Controlled auditory scene experimentation, analysis and simulation form sources of possible partial explanations to how sound localization occurs. Still, experimentation and analysis on their own provide only alternative **subjective** approaches. Simulation of an auditory scene may augment the experimentation and analysis by providing new **environmental** data not apparent from a subjective point of view. Soundscape fills in the gaps with environmental clues such as instantaneous monitoring of sound pressure across surfaces.

Context of This Writing

This research required an investigation of other topics which are popularly considered technological paradigms in the information sciences. Namely, these topics are: object-oriented programming, CORBA and distributed software development, OMT (Object Modeling Technique) and architectural software Design Patterns. In addition, the Java and C++ languages and practical uses made of other exciting technologies were explored.

Typical Operational Steps

To simulate an auditory scene there are usually three operational tasks:

1. plan and diagram the objects in the simulation (During this task, all attributes which need non-default values must be identified.)
2. start the simulator, add and configure all the objects involved in the simulation plan, and set the simulation's time constraints
3. name an output file, run the simulation, perform single steps as necessary, and be sure to get screen shots for a simulation report

During familiarization with the simulator, task #1 (above) could be skipped. For more specific operational details, see the included description of the user interface (chapter two) and demonstrative use cases (chapter three).

Simulator Operation

During simulation, queues of **Signal** objects are maintained between each sound source model and sound listening model pair. The Signal object is initialized with references to both the originating sound source model and the target sound listening model, and a current time stamp. With it's given parameters, a Signal object may calculate it's own attenuation and time of effect (on the sound listening model).

For each simulation instant, all required Signal objects are made and managed by the software's signal queue managers. Then, the head of each Signal queue is checked for ready Signal objects. Since the Signal objects are enqueued in order, there is no sorting or searching for Signal objects which have met their time of effect. When Signal objects are found which have met their time of effect, they are dequeued and then used to "rattle" their target sound listener model.

Origin of Concept

Big surprises were encountered while the work for this thesis was performed. As may often be the case, a general interest kicked things off, a niche was found and the work gained focus from there. For this thesis, computerized sound localization is and was the general interest. Computerized auditory scene simulation became the niche. How this thesis topic narrowed from sound localization to auditory scene simulation is best explained by contrasting what was planned to what actually occurred.

<u>What was planned</u>: This thesis work began as a two-part investigation of applied sound localization. The first quarter or first half of the work was originally planned to involve a small computer program (simulator) which would allow easy composition of sound sources in space, and render multiple output streams of pulse code modulated audio data. The remaining time was planned to be spent to develop a second computer program (listening azimuth selector), which would accept a pair of the output streams from the proprietary simulator. It was thought that a user might input a "listening azimuth" and/or "listening elevation" and the listening azimuth selector would render a single corresponding auditory stream.

<u>What actually occurred</u>: When the time allotted for this simulator prototype had passed, an accurate and capable proprietary software system had been accomplished. Yet, the performance of the simulator was disappointing. Instead of orderly queuing Signal objects as described above in "Simulator Operation," the preliminary simulator consolidated all Signal objects into a single unordered container. Consequently, it was more work during each simulation instant, to search the entire Signal container for "ready" Signal objects. In short, the simulator's simplicity hampered it's performance and potential.

Although the preliminary simulator was adequate, it was clear that many improvements could be made. A decision was made to compromise less on the depth of this investigation, and perhaps compromise more on it's width. The goal of developing the listening azimuth selector computer program was relinquished, and full-time development/refinement of the auditory scene simulator ensued.

Scale of the Soundscape Software Project

Time spent developing the Soundscape auditory scene simulator is roughly four hundred and fifty hours over a sixteen month period. The net source code size (using the semicolon-count method) is 18,136 lines. Two hundred sixty-eight C++ class definitions and one hundred twelve Java class/interface definitions are involved. About 25% of the source code was handwritten. The majority and remainder of the source code was generated by IDL compilers from handwritten interface definitions. All of the simulator's source code is original, and was produced by this thesis author.

Test System Configuration

The Soundscape simulator was developed and tested on an Intel Pentium 200MHZ PC with 64MB of RAM. The project occupies approximately 40MB of disk space. SoundscapeD was developed and tested using NT4.0, and SoundscapeSA was developed and tested on both NT4.0 and Win95. JDK version 1.1.5 was used for final development and testing. Visigenic's Visibroker for Java (v3.2) and Visibroker for C++ (v3.2) were used in the development and testing of all Java and C++ distributed software subsystems.

Chapter 2: User Interface Description

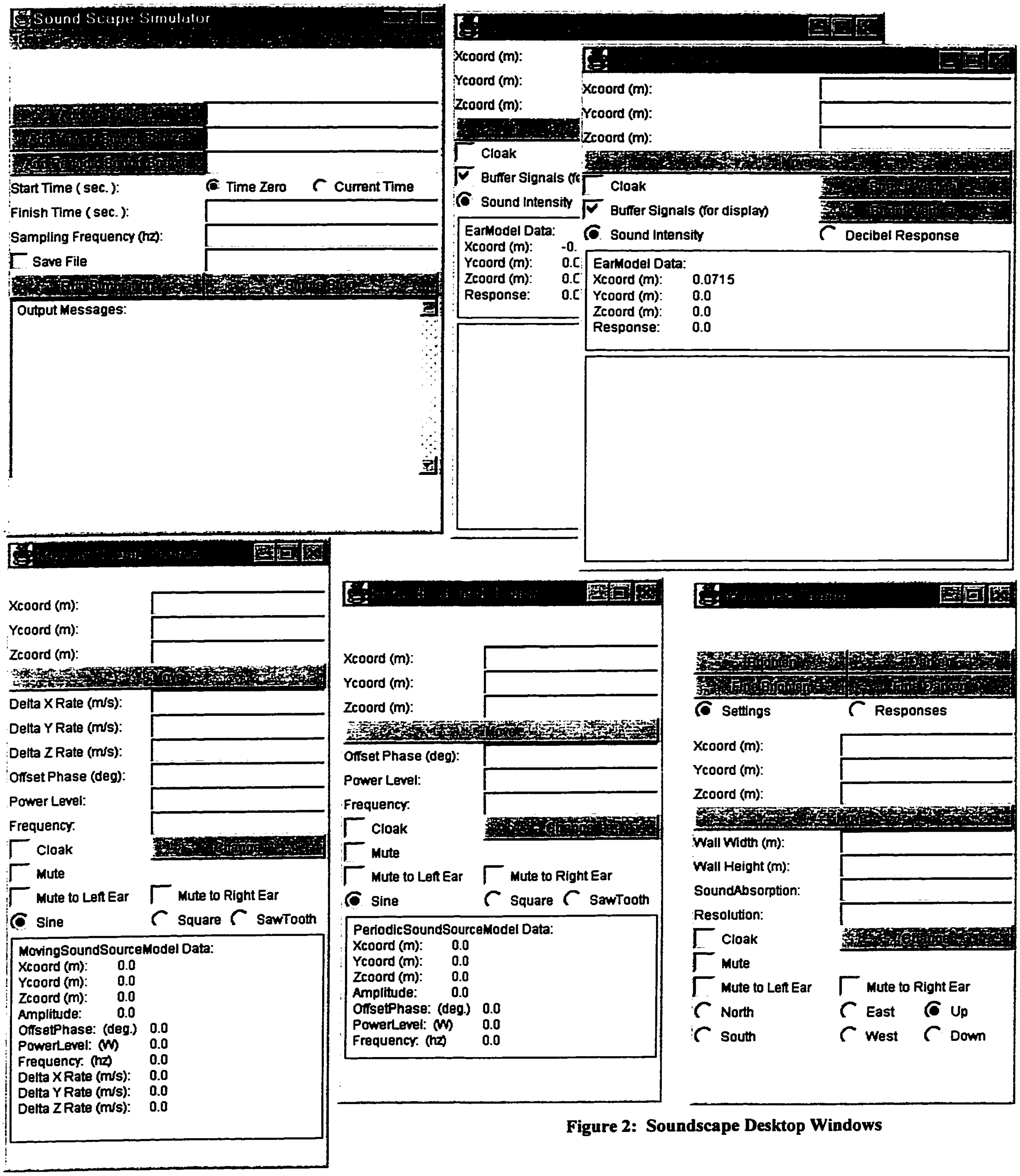
Soundscape's user-interface has been written completely in Java source code because of Java's unsurpassed portability. Whether the simulator is run in either it's stand-alone or distributed configurations, locally or through a web browser across the Internet, the user interface is equally operational on any Java-enabled computer system, regardless of platform. Soundscape's user-interface implements the Java Development Kit (JDK) 1.1.x abstract window toolkit API and has no deprecated commands as of the JDK version 1.1.5, released in January 1998.

The Soundscape simulator desktop consists of view and control panels for all modeled abstractions incorporated into a given simulation. Also, the desktop has a view and control panel for the simulator itself. Most of the view and control panels are single-screen, having simple windowed control components such as push buttons, text entry fields, and check boxes. The view and control panels may be resized or *iconified*[3]. This allows a user to quickly configure each simulation model and in turn stow each view, conserving desktop space.

As the user builds a simulation, view and control panels for the selected simulation models will pop up. Each new panel is briefly parented by the SimulatorController (the control section of the SimulatorView) and immediately bound to a new instance of it's matching simulation model. For instance, when the simulator is started two EarModel instances are created (left and right.) The view/control panels for the new models are then bound and made visible immediately. Once bound, each Model-View-Controller trinity is largely on it's own, only logging occasional text into the SimulatorView's integrated text log. An example of what a simulator user's desktop might look like is shown on the following page.

[3] meaning made to be represented as icon(s)

The Soundscape Simulator Desktop



Start Time (sec.): [✓ Time Zero] Current Time
Finish Time (sec.):
Sampling Frequency (hz):
Save File

[✓ Buffer Signals (for display)]
[✓ Sound Intensity] Decibel Response

EarModel Data:
Xcoord (m): 0.0715
Ycoord (m): 0.0
Zcoord (m): 0.0
Response: 0.0

[✓ Sine] Square SawTooth

MovingSoundSourceModel Data:
Xcoord (m): 0.0
Ycoord (m): 0.0
Zcoord (m): 0.0
Amplitude: 0.0
OffsetPhase: (deg.) 0.0
PowerLevel: (W) 0.0
Frequency: (hz) 0.0
Delta X Rate (m/s): 0.0
Delta Y Rate (m/s): 0.0
Delta Z Rate (m/s): 0.0

[✓ Sine] Square SawTooth

PeriodicSoundSourceModel Data:
Xcoord (m): 0.0
Ycoord (m): 0.0
Zcoord (m): 0.0
Amplitude: 0.0
OffsetPhase: (deg.) 0.0
PowerLevel: (W) 0.0
Frequency: (hz) 0.0

[✓ Settings] Responses

North East [✓ Up]
South West Down

Figure 2: Soundscape Desktop Windows

Description of the SimulatorView

Soundscape's SimulatorView allows the user to control the SimulatorModel, add new Modeled instances into the simulation, and specify a file name for saving the simulation results. The Soundscape-D version's SimulatorView differs from the Soundscape-SA SimulatorView in that there are text fields for entering the object names of new modeled instances. With it's integrated runtime text log, the SimulatorView reports textual information to the user from all modeled abstractions and the simulator core.

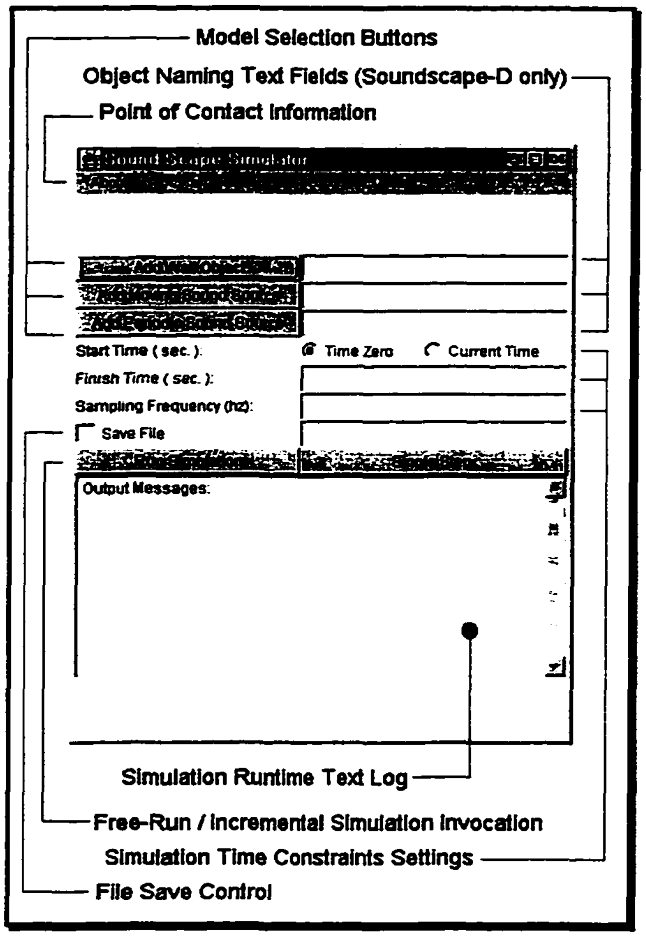

Model Selection Buttons
Object Naming Text Fields (Soundscape-D only)
Point of Contact Information
Sound Scape Simulator
Start Time (sec.):
Time Zero
Current Time
Finish Time (sec.):
Sampling Frequency (hz):
Save File
Output Messages:
Simulation Runtime Text Log
Free-Run / Incremental Simulation Invocation
Simulation Time Constraints Settings
File Save Control

Figure 3: SimulatorView

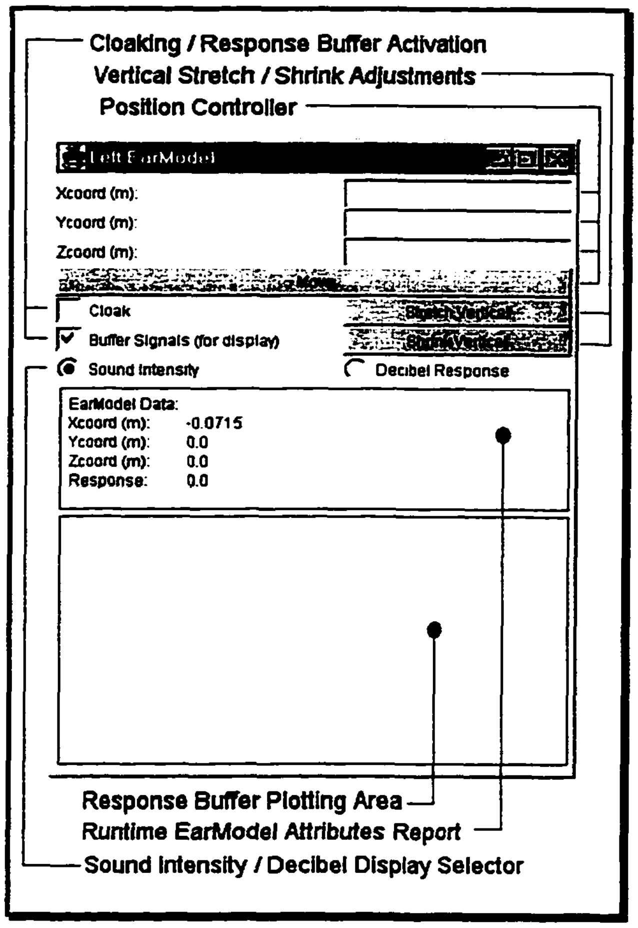
Components of the SimulatorView are arranged in a logical order, from top to bottom. To run a simulation, a user would begin by naming and selecting each model in their simulation. Object naming is achieved in the Soundscape-D configuration by first typing the object name into the text field beside the model-selection button and then pressing the model-selection button. A named instance of the model-type is then immediately created and it's view/controls made visible.

After all desired models are added to the simulator, the user should type in the desired finish time (in seconds) and sampling frequency (in hertz). If a saved output file is desired, the 'Save File' check box should be toggled after a filename is entered in the space beside the 'Save File' checkbox. Each time the 'Save File' checkbox is toggled **on**, a file is opened with the name specified. Each time the 'Save File' checkbox is toggled **off**, the currently open output file (if any) is closed. The output file simply contains an interleaved stream of IEEE double precision floating point values, which are the pulse code modulated responses from the simulator's right and left EarModels.

Typically, a user will want to cloak all simulation models prior to running a simulation, in order to boost the simulator's performance. Cloaking a model eliminates the work involved in updating it's user-interface, so the system spends more time rendering an auditory scene. To run the simulation, the 'Run Simulation' button is then pressed. During the simulation, various information will be written to the text log. Once the simulation is run to the specified completion time, a completion message will appear in the text log. The user may then close the output file and/or inspect the simulation progress in single-step mode as desired.

Description of the EarView

The EarView is the simplest view and controller in the Soundscape simulator. By default, the left and right EarModels are centered +/- 7.15 centimeters on either side of the X-axis. An auditory scene experimentalist may be interested in repositioning the EarModels, and may do so easily with the EarView's position controller. The new coordinates should be entered into the appropriate text fields, and then the 'Move' button pressed.


Cloaking / Response Buffer Activation
Vertical Stretch / Shrink Adjustments
Position Controller
Left EarModel
Move
StretchVertical
ShrinkVertical
Response Buffer Plotting Area
Runtime EarModel Attributes Report
Sound Intensity / Decibel Display Selector

Xcoord (m):
Ycoord (m):
Zcoord (m):
Cloak
Buffer Signals (for display)
Sound Intensity
Decibel Response

EarModel Data:
Xcoord (m): -0.0715
Ycoord (m): 0.0
Zcoord (m): 0.0
Response: 0.0

Figure 4: EarView User Interface

Any repositioning of the EarModels should be done prior to running the simulation. Invalidation of simulation results will occur if an EarModel is moved after a simulation has begun.

As seen to the right, the EarView has toggles to activate cloaking and/or buffering of response values. Cloaking a simulation model will typically boost the simulator's performance, although at times it is interesting to watch a simulation's progression by not cloaking all models. The 'Buffer Signal' switch should be toggled on whenever a graphical representation of the latest EarModel responses is of interest.

Maintaining a buffer of response values has a slight computational cost, so for long simulation runs a user may prefer to run the majority of the simulation without buffering enabled, and then enable buffering for the last 600 simulation instants. The EarView plots the latest six hundred responses, with the most recent responses being at the right extreme of the screen. Until six hundred responses have been buffered (and the response buffer is full) the results are not plotted. More or less of the response buffer may be visible, depending on how EarView window is stretched horizontally. Also, the responses may be viewed in sound intensity or decibel format by user preference.

Often, cloaking a simulation model results in expired data being displayed persistently on the various Soundscape simulator views. A user may refresh the data by simply mouse clicking on the view panel. In most cases, this will cause the view to be repainted with updated values.

Description of the PeriodicSoundSourceView

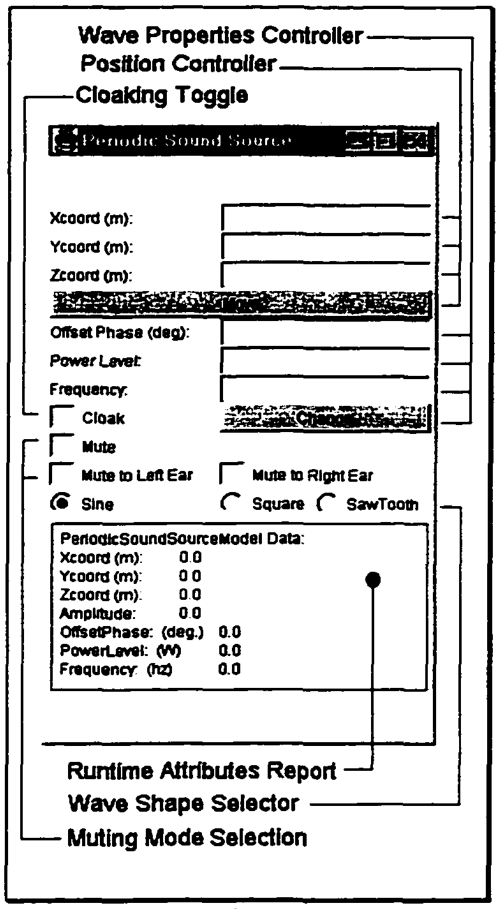
The PeriodicSoundSourceView has position controller and cloaking toggle components which are identical in operational behavior as described for the EarView. Unlike the EarModels, repositioning a sound source at any time will not result in invalidation of simulation results. Movement of a sound source will still result in an accurate simulation.

In addition, there are controls for wave properties and various muting mode selection. To adjust the initial offset phase, power level, and or frequency, the new value(s) should be entered in the appropriate text field. Changes are committed when the 'Change' button is pressed. In general, if a text field is left blank then no change will occur for the matching attribute.

Muting mode selections are:

- muted completely (to ALL sound listening objects)
- muted to the left ear model
- muted to the right ear model

Complete muting includes both ear models. The option of muting a sound source to one or both ear models is offered so that users may observe only echoed sound if desired.



```
PeriodicSoundSourceModel Data:
Xcoord (m):      0.0
Ycoord (m):      0.0
Zcoord (m):      0.0
Amplitude:       0.0
OffsetPhase: (deg.)  0.0
PowerLevel: (W)      0.0
Frequency: (hz)      0.0
```

Figure 5: PeriodicSoundSourceView

Wave shape may be selected by toggling the desired option. Note that the square and sawtooth wave shapes are scaled with an RMS[4] factor so that their wave integral is equal to a sinusoid wave of the same amplitude.

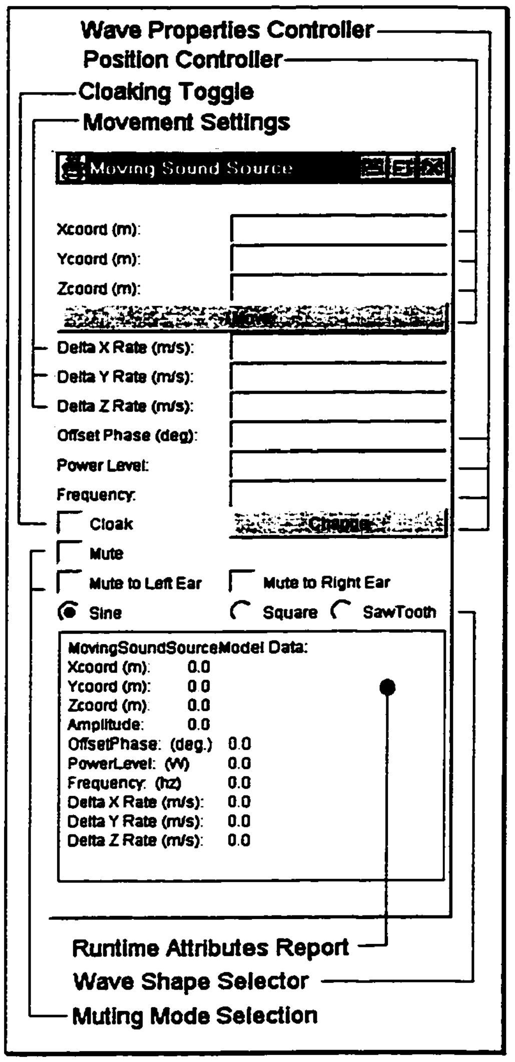
[4] RMS here stands for root-mean-square, a method for integrating functions. Simple wave shapes such as square, sinusoid and sawtooth have fixed scaling factors by which their amplitude and wave energy may be related.

Description of the MovingSoundSourceView

The MovingSoundSourceView has position controller, cloaking toggle, muting mode, wave properties, and wave shape selection components which are identical in operational behavior as described for the PeriodicSoundSourceView. The position settings made for a moving sound source model become the **initial** coordinates.

In addition, there are controls for the programmed movement of the MovingSoundSourceModel during the simulation. Linear rates of movement in each coordinate may be entered in the same way as the wave properties of the moving sound source model.

Of course, if a moving sound source model has no movement, it will effect the simulation exactly as a stationary sound source with similar wave property settings.



Xcoord (m):
Ycoord (m):
Zcoord (m):
Delta X Rate (m/s):
Delta Y Rate (m/s):
Delta Z Rate (m/s):
Offset Phase (deg):
Power Level:
Frequency:
Cloak
Mute
Mute to Left Ear
Mute to Right Ear
[✓ Sine] Square SawTooth

```
MovingSoundSourceModel Data:
Xcoord (m):        0.0
Ycoord (m):        0.0
Zcoord (m):        0.0
Amplitude:         0.0
OffsetPhase: (deg.)  0.0
PowerLevel: (W)      0.0
Frequency: (hz)      0.0
Delta X Rate (m/s):  0.0
Delta Y Rate (m/s):  0.0
Delta Z Rate (m/s):  0.0
```

Figure 6: MovingSoundSourceView

Description of the WallView

The WallView has position controller, cloaking toggle and muting mode controls which are identical in operational behavior as described for the PeriodicSoundSourceView. The position settings made for a wall model become the coordinates of it's upper left corner.

In addition, there are controls for the height, width, sound absorption, and resolution of the wall model. Height and width should be entered in meters. Sound absorption should be a positive decimal value between zero and one. For example, if a wall model should absorb 25% of the sound energy it is met with, a value of .25 should be entered. In this way, a user may control the amount of echoed sound during auditory scene simulation.

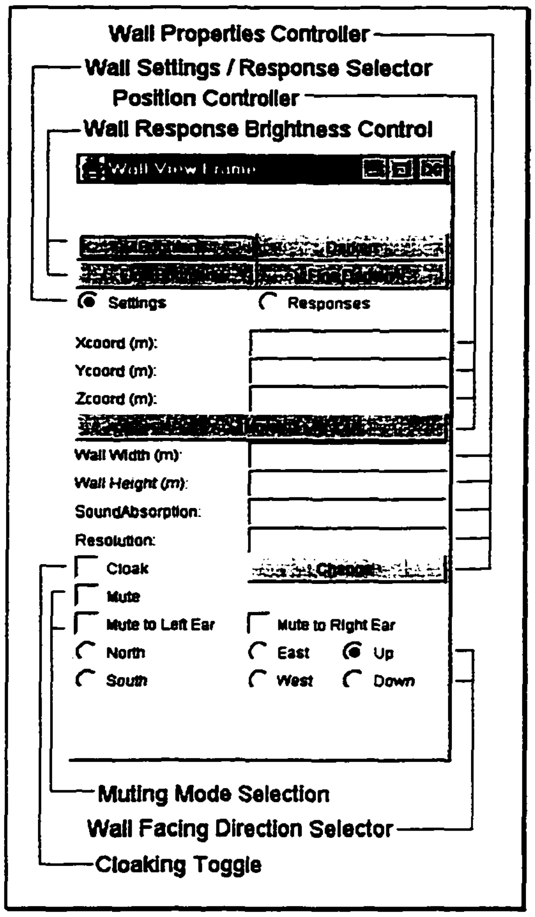


Figure 7: WallView

Resolution is the distance (in meters) between the modeled echo points on the wall model's surface. Care should be taken when deciding upon the resolution of a wall model. The sampling frequency of a simulation is a limiting factor in how close objects may be positioned before simulation errors may occur.

Dividing the simulation's speed of sound (344 m/s) by the simulation's (user-specified) sampling frequency will give this minimum spacing of simulation objects, and the minimum safe resolution for a wall model. For instance, if a user wants a simulation sampling rate of 44,100HZ then the minimum spacing is (344 / 44100) or 7.8 millimeters. Objects closer than this distance during the simulation will have responses which reflect erratic "piling up" of simulated sound pressure.

At the bottom of the WallView is a selection group for the wall model's facing direction. The directions 'Up' and 'Down' might be chosen when simulating floor and/or ceiling surfaces. A graphical representation of the instantaneous sound pressure at the wall model's surface may be seen by toggling the 'Responses' selector. After carefully adjusting the brightness controls, varying isobars may be visible in the wall response viewing panel. To return to the wall model settings panel, the 'Settings' selector should be clicked.

Chapter 3 : Demonstrative Use Cases

The use cases in this chapter were carefully planned and written to provide the reader with an illustration of the Soundscape simulator's features and operation. Each use case was kept simple, in order to reduce confusion. Often, settings such as one whole second, (for simulation duration) one whole meter, and 44.1kHZ (CD-quality sampling frequency) were used when selection of values was arbitrary. The use case examples all share a common format which consists of:

1. a brief description
2. a walk-through of exactly how the use case was performed
3. illustrative screen shots of the simulator's controls
4. observations about the use case

In addition to providing familiarity with the Soundscape simulator, the use cases show that simulation results match closely the attenuation, delay and reflection of real sound waves. While preparing this chapter, it was thought that if elementary wave-mechanical properties could be made apparent in example simulations, then the reliability of the simulator's results would be established. The use cases presented in this chapter challenge the simulator to properly imitate both attenuation and delay of propagating sound in five of such example simulations.

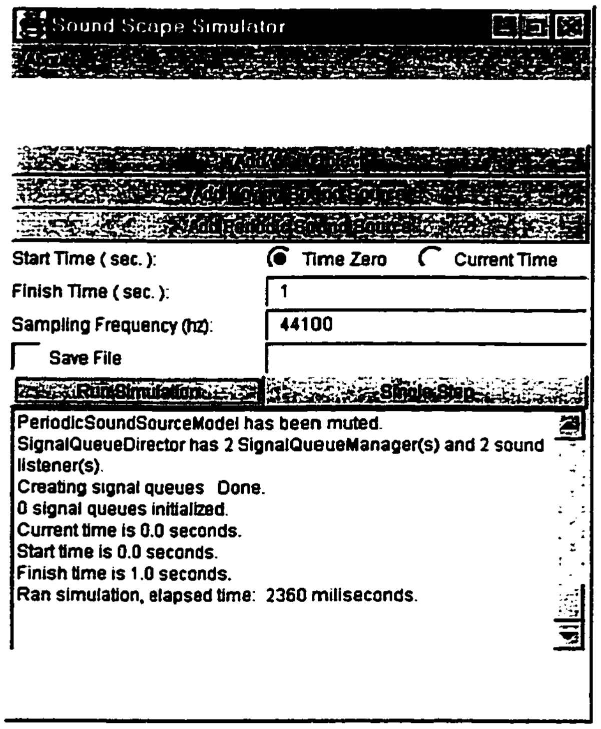
Quality of the simulator's realism is demonstrated numerically in both the wave cancellation and the wave construction use cases. In the complementary wave use case, the Soundscape simulator's integrity is further authenticated by a graphical representation of a resultant wave form. Lastly, in the Doppler effect and wall echo use cases, the resulting graphical output of the simulator implies that it's operation is consistent with analogous situations in the physical world.

Wave Cancellation

This first use case demonstrates wave energy conservation, using the Soundscape simulator. In this scenario, there are two periodic sound sources arranged forward of and centered to the subject. The "upper" sound source is positioned one meter above ear level. The "lower" sound source is one meter below ear level. So, both sound sources are equal distances from the subject's right and left ear models. Also, the lower sound source's initial offset phase is set to 180 degrees, so that it is anti-noise to the upper sound source.

The following setup steps were taken:

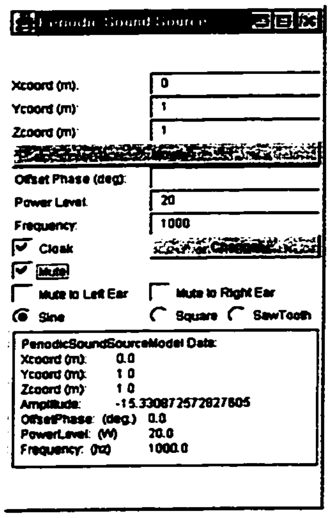
1. The simulator was started, and 'Time Zero' was selected as the simulation start time.
2. A finish time of 1 second was specified.
3. The simulation sampling frequency was set to 44100HZ.



Start Time (sec.): [✓ Time Zero] Current Time
Finish Time (sec.): 1
Sampling Frequency (hz): 44100
Save File

```
PeriodicSoundSourceModel has been muted.
SignalQueueDirector has 2 SignalQueueManager(s) and 2 sound listener(s).
Creating signal queues  Done.
0 signal queues initialized.
Current time is 0.0 seconds.
Start time is 0.0 seconds.
Finish time is 1.0 seconds.
Ran simulation, elapsed time: 2360 milliseconds.
```

Figure 8: Cancellation Steps 1-3



Xcoord (m): 0
Ycoord (m): 1
Zcoord (m): 1
Offset Phase (deg):
Power Level: 20
Frequency: 1000
[✓ Cloak]
[✓ Mute]
Mute to Left Ear
Mute to Right Ear
[✓ Sine] Square SawTooth

```
PeriodicSoundSourceModel Data:
Xcoord (m):        0.0
Ycoord (m):        1 0
Zcoord (m):        1 0
Amplitude:         -15.330872572827605
OffsetPhase: (deg.)   0.0
PowerLevel: (W)       20.0
Frequency: (hz)       1000.0
```

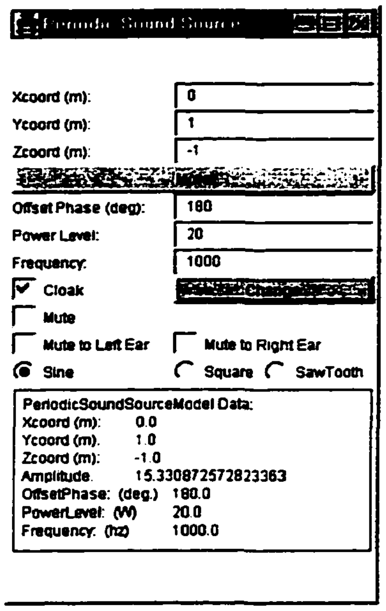
Figure 9:

Cancellation Steps 4-7

4. A periodic sound source was added to the simulation.
5. The new sound source was positioned at coordinates (0, 1, 1).
6. Power level and driving frequency of the sound source were set to 20W and 1000HZ.
7. The sound source wave shape was observed to be set to SINE by default.

(continued on the following pages)

8. A second periodic sound source was added to the simulation.
9. The second (lower) sound source was positioned at coordinates (0, 1, -1).
10. Power level and driving frequency of the lower sound source were set to 20W and 1000HZ.
11. The sound source wave shape was observed to be set to SINE by default.
12. Offset phase of the lower sound source was set to 180 degrees.
13. Both ear models and sound sources were cloaked, and the 'Run Simulation' button was pressed.

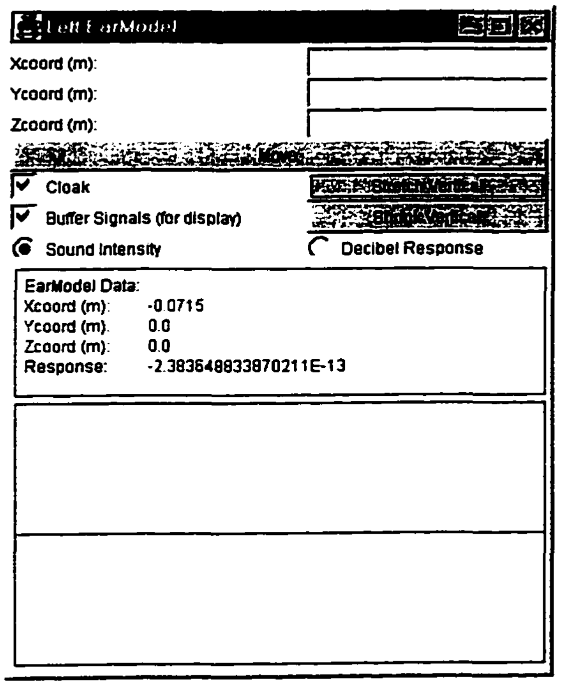


Xcoord (m): 0
Ycoord (m): 1
Zcoord (m): -1
Offset Phase (deg): 180
Power Level: 20
Frequency: 1000
[✓ Cloak]
Mute
Mute to Left Ear   Mute to Right Ear
[✓ Sine]   Square   SawTooth

```
PeriodicSoundSourceModel Data:
Xcoord (m):        0.0
Ycoord (m).        1.0
Zcoord (m):        -1.0
Amplitude.         15.330872572823363
OffsetPhase: (deg.)   180.0
PowerLevel: (W)       20.0
Frequency: (hz)       1000.0
```

Figure 10:

Cancellation Steps 8-12



Xcoord (m):
Ycoord (m):
Zcoord (m):
[✓ Cloak]
[✓ Buffer Signals (for display)]
[✓ Sound Intensity]   Decibel Response

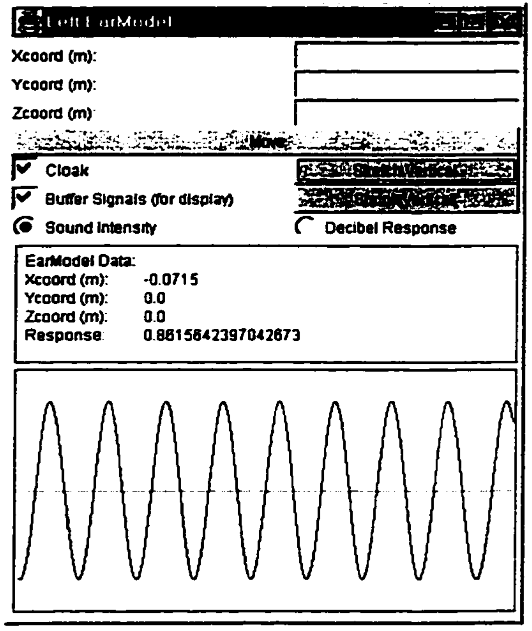
```
EarModel Data:
Xcoord (m):     -0.0715
Ycoord (m).     0.0
Zcoord (m):     0.0
Response:       -2.383648833870211E-13
```

Figure 11: Canceled Response

After the simulation completed its run, the responses of the left and right ear models were observed to be identical. Also, the responses were observed to be nearly zero. The lack of any appreciable responses implies that the two sound sources were accurately simulated as anti-noise to each other.

To contrast the results of the anti-noise scenario, the simulation was run again with the lower sound source muted (i.e. without anti-noise applied.)

The response without anti-noise applied was observed to be significantly (thirteen orders of magnitude) more than after the previous simulation run. A quick division of the anti-noise result with the un-cancelled result shows that the anti-noise does effectively cancel 100% of the upper sound source's sound. The anti-noise result is about one-third of one-*trillionth* of the result without wave cancellation.



Xcoord (m):
Ycoord (m):
Zcoord (m):

[✓] Cloak
[✓] Buffer Signals (for display)
● Sound Intensity
○ Decibel Response

```
EarModel Data:
Xcoord (m):    -0.0715
Ycoord (m):    0.0
Zcoord (m):    0.0
Response       0.8615642397042673
```

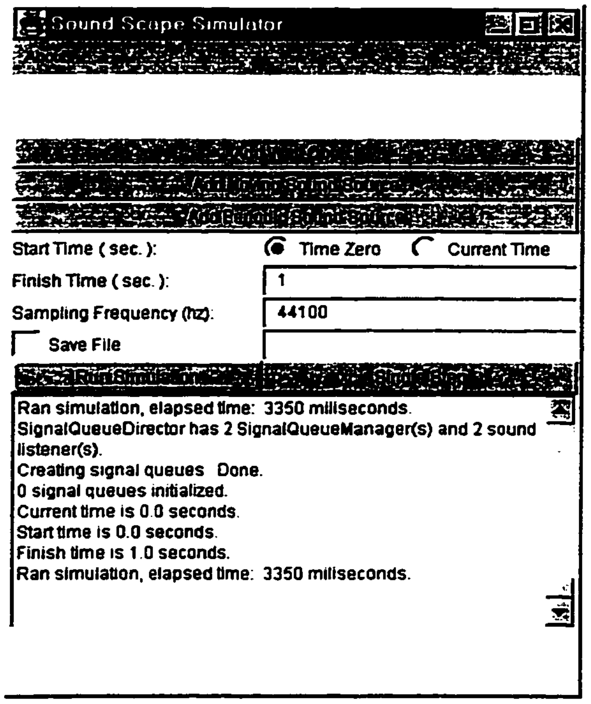
Figure 12: Un-canceled Response

Complementary Wave Test

This second case demonstrates how elementary wave shapes can complement each other and result in more complex shapes in the Soundscape simulator. As in the wave cancellation use case, this use case has two periodic sound sources arranged forward of and centered to the subject. The "upper" sound source is positioned one meter above ear level. The "lower" sound source is one meter below ear level. So, both sound sources are equal distances from the subject's right and left ear models. The upper sound source's wave shape is left in the (default) sinusoid selection. The lower sound source's wave shape is set to square and it's initial offset phase is set to 180 degrees.

The following setup steps were taken:

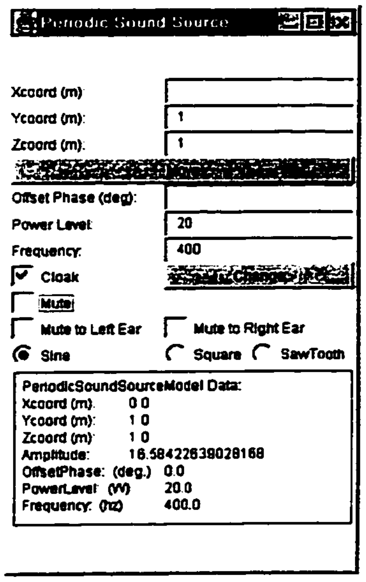
1. The simulator was started, and 'Time Zero' was selected as the simulation start time.
2. A finish time of 1 second was specified.
3. The simulation sampling frequency was set to 44100HZ.



Start Time (sec.): [✓ Time Zero] Current Time
Finish Time (sec.): 1
Sampling Frequency (hz): 44100
Save File

```
Ran simulation, elapsed time:  3350 milliseconds.
SignalQueueDirector has 2 SignalQueueManager(s) and 2 sound
listener(s).
Creating signal queues   Done.
0 signal queues initialized.
Current time is 0.0 seconds.
Start time is 0.0 seconds.
Finish time is 1.0 seconds.
Ran simulation, elapsed time:  3350 milliseconds.
```

Figure 13: Complementary Steps 1-3



Xcoord (m):
Ycoord (m): 1
Zcoord (m): 1
Offset Phase (deg):
Power Level: 20
Frequency: 400
[✓ Cloak]
Mute
Mute to Left Ear
Mute to Right Ear
[✓ Sine] Square SawTooth

```
PeriodicSoundSourceModel Data:
Xcoord (m).        0 0
Ycoord (m):        1 0
Zcoord (m)         1 0
Amplitude:         16.58422639028168
OffsetPhase:  (deg.)  0.0
PowerLevel  (W)       20.0
Frequency:  (hz)      400.0
```

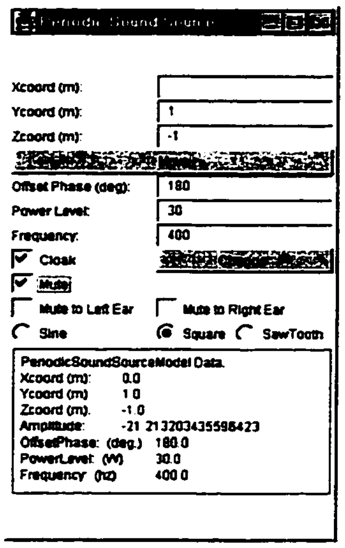
Figure 14:

Complementary Steps 4-7

4. A periodic sound source was added to the simulation.
5. The new sound source was positioned at coordinates (0, 1, 1).
6. Power level and driving frequency of the sound source were set to 20W and 1000HZ.
7. The sound source wave shape was observed to be set to 'Sine' by default.

(continued on the following pages)

8. A second periodic sound source was added to the simulation.
9. The second (lower) sound source was positioned at coordinates (0, 1, -1).
10. Power level and driving frequency of the lower sound source were set to 20W and 1000HZ.
11. The sound source wave shape was be set to 'Square'.
12. Offset phase of the lower sound source was set to 180 degrees.
13. The lower sound source was muted for the first trial run.

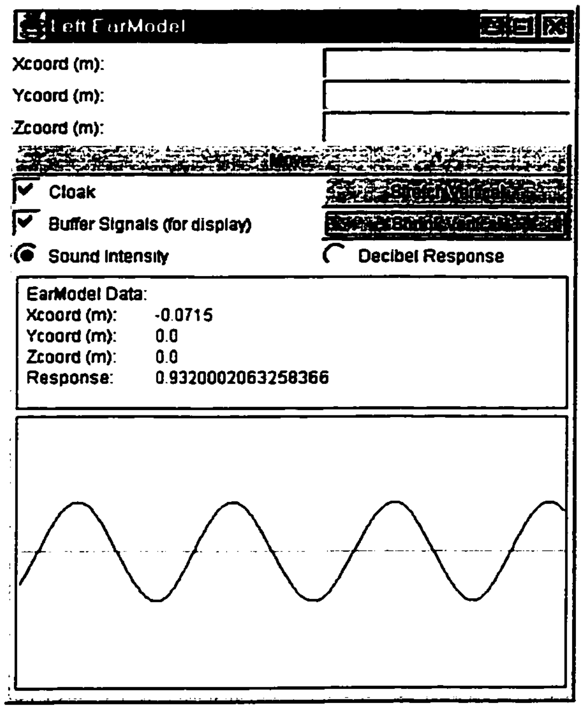


```
Xcoord (m):
Ycoord (m):              1
Zcoord (m):              -1
Offset Phase (deg):      180
Power Level:             30
Frequency:               400
[✓] Cloak
[✓] Mute
[ ] Mute to Left Ear     [ ] Mute to Right Ear
( ) Sine                 (●) Square   ( ) SawTooth

PeriodicSoundSourceModel Data.
Xcoord (m):        0.0
Ycoord (m)         1 0
Zcoord (m).        -1.0
Amplitude:         -21 21 3203435598423
OffsetPhase: (deg.)   180.0
PowerLevel: (W)       30.0
Frequency (hz)        400 0
```

Figure 15:

Complementary Steps 8-13



```
Xcoord (m):
Ycoord (m):
Zcoord (m):
[✓] Cloak
[✓] Buffer Signals (for display)
(●) Sound Intensity        ( ) Decibel Response

EarModel Data:
Xcoord (m):    -0.0715
Ycoord (m):    0.0
Zcoord (m):    0.0
Response:      0.93200020632583666
```

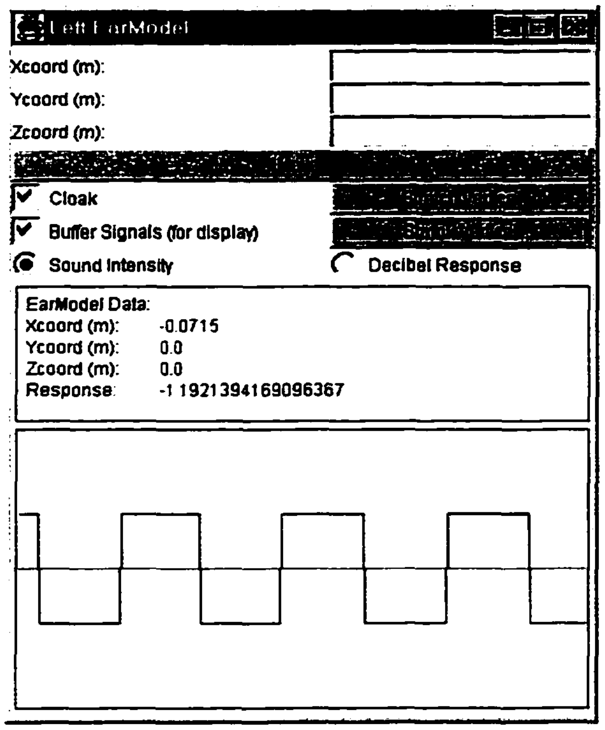
Figure 16: Sinusoid Contribution

14. Both ear models and sound sources were cloaked, and the 'Run Simulation' button was pressed.

A quick trial simulation was run, with just the sinusoid sound source activated, and the square wave sound source muted. Figure 3.2.4 shows a graph of the sinusoid sound source's wave shape.

Another quick trial was run with the sinusoid sound source muted and the square wave sound source active. Figure 17 shows a graph of the square-wave sound source's wave shape.

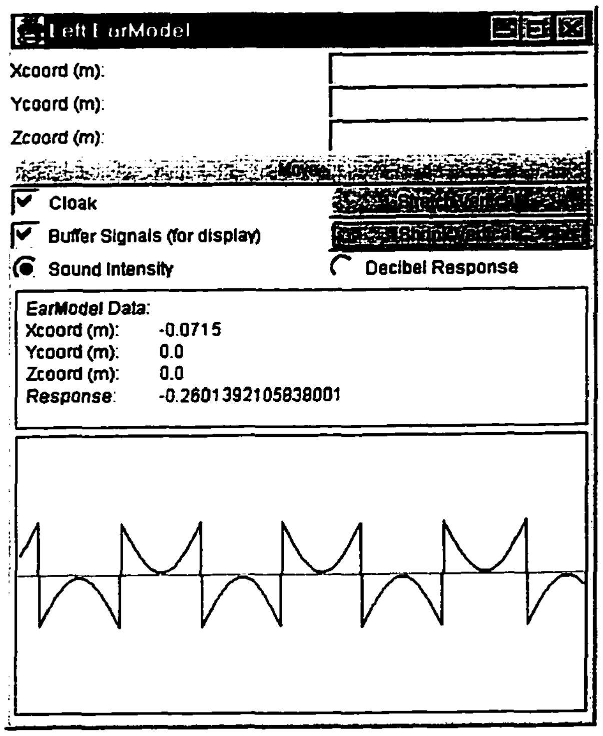
Note that it is not necessary to rebuild the simulation from scratch. In this case, the muting check boxes of the sound sources were simply adjusted, and the 'Run Simulation' button pressed for another trial run. Thus, simulations may be re-run in some cases with a couple of quick mouse-clicks.



```
EarModel Data:
Xcoord (m):    -0.0715
Ycoord (m):    0.0
Zcoord (m):    0.0
Response:      -1 1921394169096367
```

Figure 17: Square Contribution

Finally, a third simulation was run with both sound sources activated, resulting in the complementary wave shape in Figure 18. The graph shows that the two types of sound met and formed an appropriate wave shape. It also shows that the arrival of the two types of sound remained properly synchronized to the end of the simulation.



```
EarModel Data:
Xcoord (m):    -0.0715
Ycoord (m):    0.0
Zcoord (m):    0.0
Response:      -0.2601392105838001
```

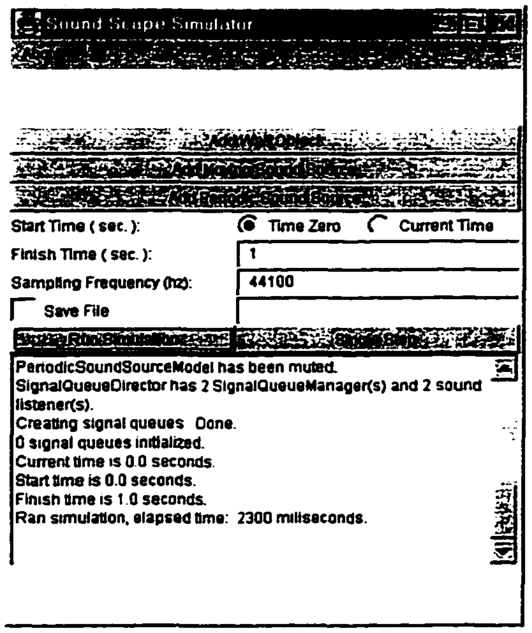
Figure 18: Complementary Responses

Wave Construction

This third use case is similar to the wave cancellation use case and was also contrived to establish the integrity of the Soundscape simulator. In this scenario, there are two periodic sound sources arranged forward of and centered to the subject. The "upper" sound source is positioned one meter above ear level. The "lower" sound source is one meter below ear level. So, both sound sources are equal distances from the subject's right and left ear models. Both sound sources have their wave shape set to 'Square'.

The following setup steps were taken:

1. The simulator was started, and 'Time Zero' was selected as the simulation start time.
2. A finish time of 1 second was specified.
3. The simulation sampling frequency was set to 44100HZ.

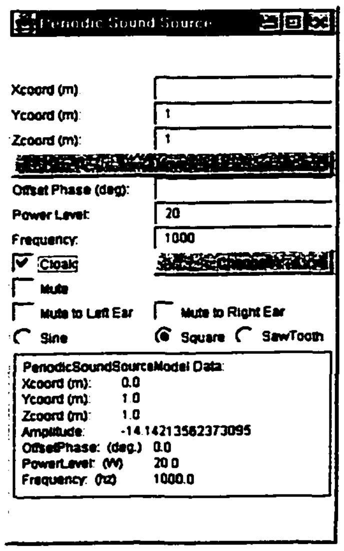


Start Time (sec.): [✓ Time Zero] Current Time
Finish Time (sec.): 1
Sampling Frequency (hz): 44100
Save File

```
PeriodicSoundSourceModel has been muted.
SignalQueueDirector has 2 SignalQueueManager(s) and 2 sound
listener(s).
Creating signal queues  Done.
0 signal queues initialized.
Current time is 0.0 seconds.
Start time is 0.0 seconds.
Finish time is 1.0 seconds.
Ran simulation, elapsed time:  2300 miliseconds.
```

Figure 19: Construction Steps 1-3

4. A periodic sound source was added to the simulation.
5. The new sound source was positioned at coordinates (0, 1, 1).
6. Power level and driving frequency of the sound source were set to 20W and 1000HZ.
7. The sound source wave shape was set to 'Square'.



Xcoord (m):
Ycoord (m): 1
Zcoord (m): 1
Offset Phase (deg):
Power Level: 20
Frequency: 1000
[✓ Cloak]
Mute
Mute to Left Ear
Mute to Right Ear
Sine [✓ Square] SawTooth

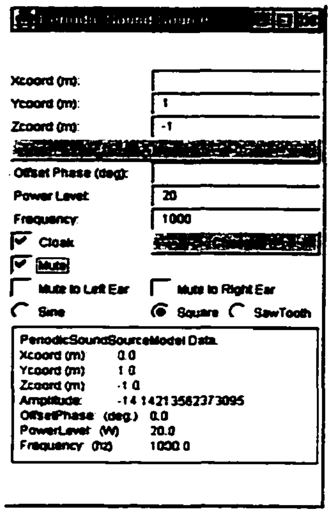
```
PeriodicSoundSourceModel Data:
Xcoord (m):        0.0
Ycoord (m):        1.0
Zcoord (m):        1.0
Amplitude:         -14.142135623730 95
OffsetPhase: (deg.)  0.0
PowerLevel: (W)      20.0
Frequency: (hz)      1000.0
```

Figure 20:

Construction Steps 4-7

(continued on the following pages)

8. A second periodic sound source was added to the simulation.
9. The second (lower) sound source was positioned at coordinates (0, 1, -1).
10. Power level and driving frequency of the lower sound source were set to 20W and 1000HZ.
11. The sound source wave shape was set to 'Square'.
12. For the first trial run, the lower sound source was muted.
13. Both ear models and sound sources were cloaked, and the 'Run Simulation' button was pressed.

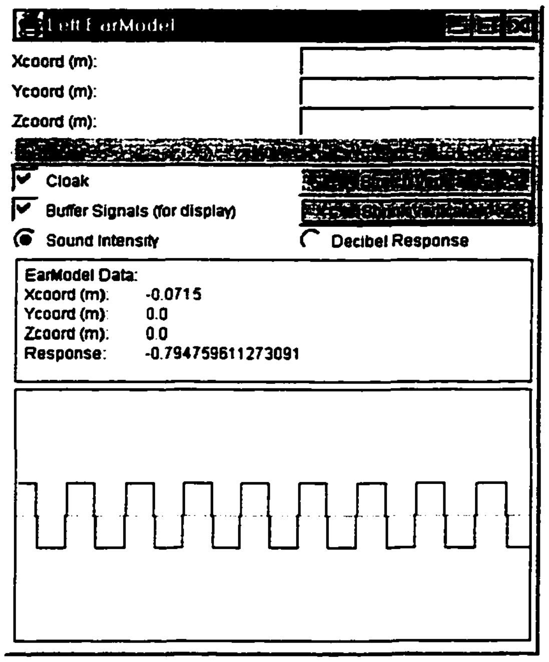


Xcoord (m):
Ycoord (m): 1
Zcoord (m): -1
Offset Phase (deg):
Power Level: 20
Frequency: 1000
[✓] Cloak
[✓] Mute
[ ] Mute to Left Ear [ ] Mute to Right Ear
( ) Sine (●) Square ( ) SawTooth

```
PeriodicSoundSourceModel Data
Xcoord (m)          0.0
Ycoord (m)          1.0
Zcoord (m)          -1.0
Amplitude           -14.142135623730951
OffsetPhase (deg.)  0.0
PowerLevel (W)      20.0
Frequency (Hz)      1000.0
```

Figure 21:

Construction Steps 8-12



Xcoord (m):
Ycoord (m):
Zcoord (m):
[✓] Cloak
[✓] Buffer Signals (for display)
(●) Sound Intensity ( ) Decibel Response

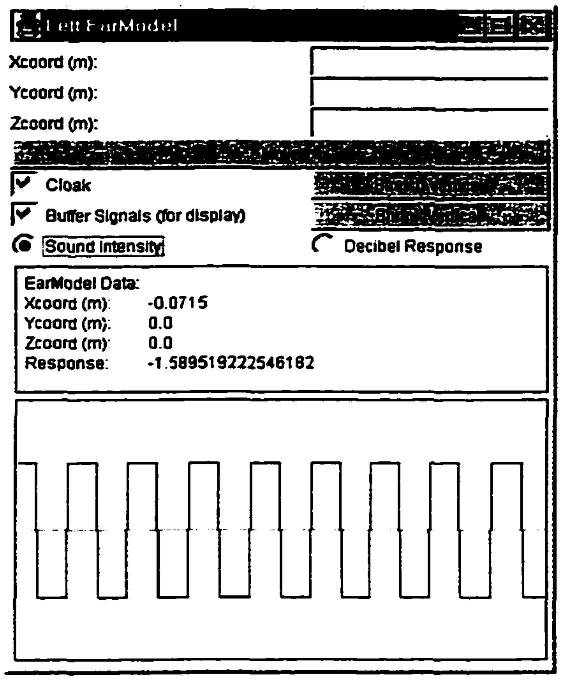
```
EarModel Data:
Xcoord (m):   -0.0715
Ycoord (m):   0.0
Zcoord (m):   0.0
Response:     -0.7947598611273091
```

Figure 22: Single Wave Responses

After the simulation completed its run, the responses of the left and right ear models were observed to be identical. The most recent response values of the ear models were observed to be about -0.8.

To observe effects of wave construction (if any), the simulation was run again with the lower sound source un-muted.

The most recent ear model responses with both sound sources activated was observed to be about -1.59. A quick division of the anti-noise result with the single sound source result shows that the constructive wave (from the lower sound source) effectively doubles the subjective sound perceived by the ear models.



```
EarModel Data:
Xcoord (m):    -0.0715
Ycoord (m):    0.0
Zcoord (m):    0.0
Response:      -1.5895192225461 82
```

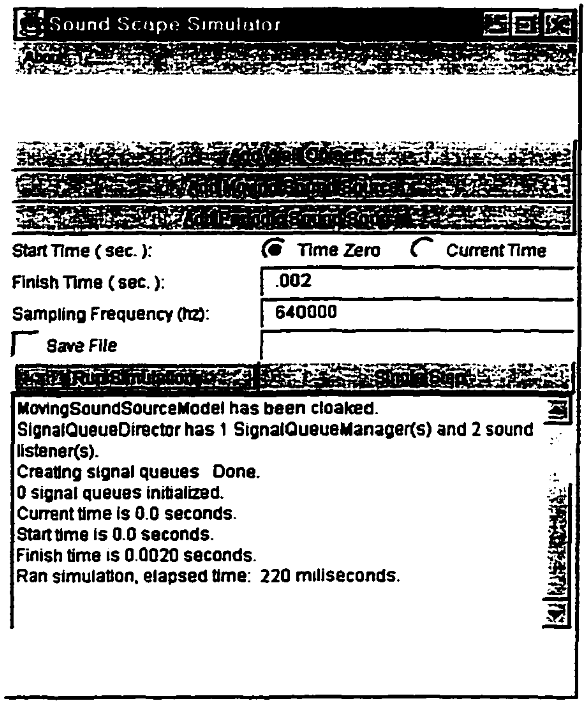
Figure 23: Dual Wave Construction

Doppler Effect of an Approaching Sound Source

This forth use case demonstrates the use and results of a moving sound source in the Soundscape simulator. In this scenario, there is a single moving sound source 3 centimeters in front of the right ear model. First, a trial simulation is run without any runtime movement by the moving sound source. For contrast, in the second trial simulation the moving sound source is set to approach the right ear model at a constant speed of 100 meters per second.

The following setup steps were taken:

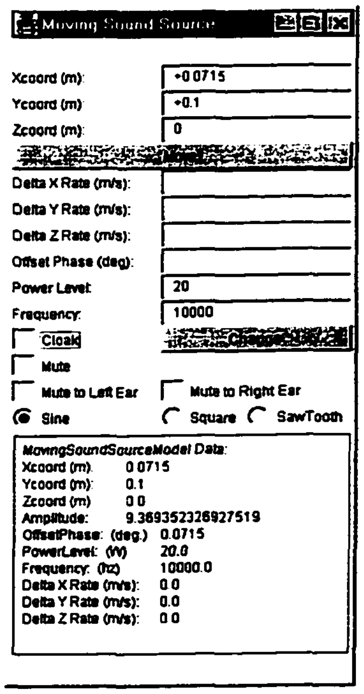
1. The simulator was started, and 'Time Zero' was selected as the simulation start time.
2. A finish time of 2 milliseconds was specified.
3. The simulation sampling frequency was set to 640,000HZ.



Start Time (sec.): [✓ Time Zero] Current Time
Finish Time (sec.): .002
Sampling Frequency (hz): 640000
Save File

```
MovingSoundSourceModel has been cloaked.
SignalQueueDirector has 1 SignalQueueManager(s) and 2 sound
listener(s).
Creating signal queues   Done.
0 signal queues initialized.
Current time is 0.0 seconds.
Start time is 0.0 seconds.
Finish time is 0.0020 seconds.
Ran simulation, elapsed time:  220 miliseconds.
```

Figure 24: Doppler Steps 1-3



Xcoord (m): +0.0715
Ycoord (m): +0.1
Zcoord (m): 0
Delta X Rate (m/s):
Delta Y Rate (m/s):
Delta Z Rate (m/s):
Offset Phase (deg):
Power Level: 20
Frequency: 10000
Cloak
Mute
Mute to Left Ear | Mute to Right Ear
[✓ Sine] Square SawTooth

```
MovingSoundSourceModel Data:
Xcoord (m):        0.0715
Ycoord (m):        0.1
Zcoord (m)         0.0
Amplitude:         9.369352326927519
OffsetPhase: (deg.)  0.0715
PowerLevel: (W)      20.0
Frequency: (hz)      10000.0
Delta X Rate (m/s):  0.0
Delta Y Rate (m/s):  0.0
Delta Z Rate (m/s):  0.0
```

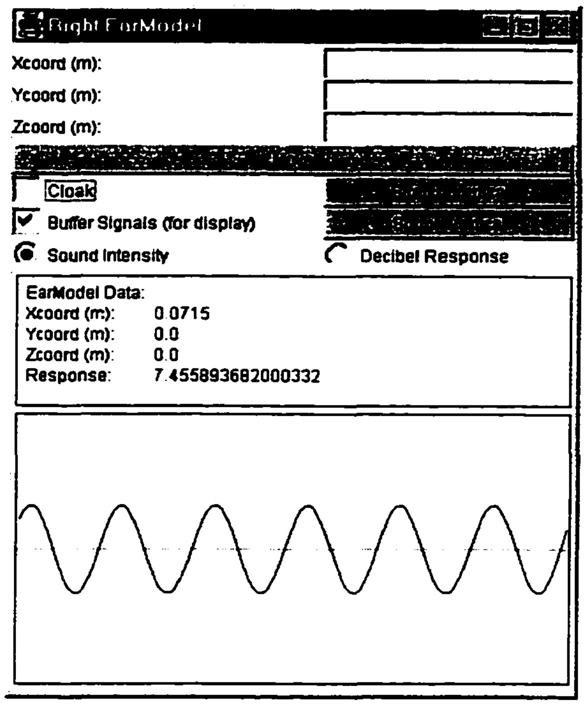
Figure 25: Doppler Steps 4-7

4. A moving sound source was added to the simulation.
5. The new sound source was positioned 1 centimeter. directly in front of the right ear model, at coordinates (0.0715, 0.1, 0.0).
6. Power level and driving frequency of the sound source were set to 20W and 10,000HZ.
7. The sound source wave shape was observed to be set to 'Sine' by default.

(continued on the following pages)

8. Both ear models and the moving sound source was cloaked, and the 'Run Simulation' button was pressed.

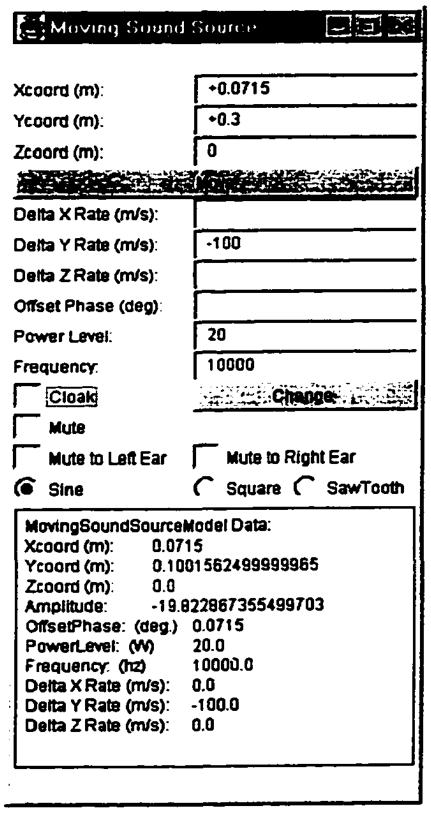
As in Figure 26, since the moving sound source had no runtime movement, the most recent responses of the right ear model appeared to be from a stationary sound source. There weren't any noticeable increases or decreases in the signal strength from peak to peak, nor any apparent change in the wavelength.



Xcoord (m):
Ycoord (m):
Zcoord (m):
[ ] Cloak
[✓] Buffer Signals (for display)
[● Sound Intensity] ○ Decibel Response

```
EarModel Data:
Xcoord (m):    0.0715
Ycoord (m):    0.0
Zcoord (m):    0.0
Response:      7.455893682000332
```

Figure 26: Stationary Responses



Xcoord (m): +0.0715
Ycoord (m): +0.3
Zcoord (m): 0
Delta X Rate (m/s):
Delta Y Rate (m/s): -100
Delta Z Rate (m/s):
Offset Phase (deg):
Power Level: 20
Frequency: 10000
[ ] Cloak
[ ] Mute
[ ] Mute to Left Ear [ ] Mute to Right Ear
[● Sine] ○ Square ○ SawTooth

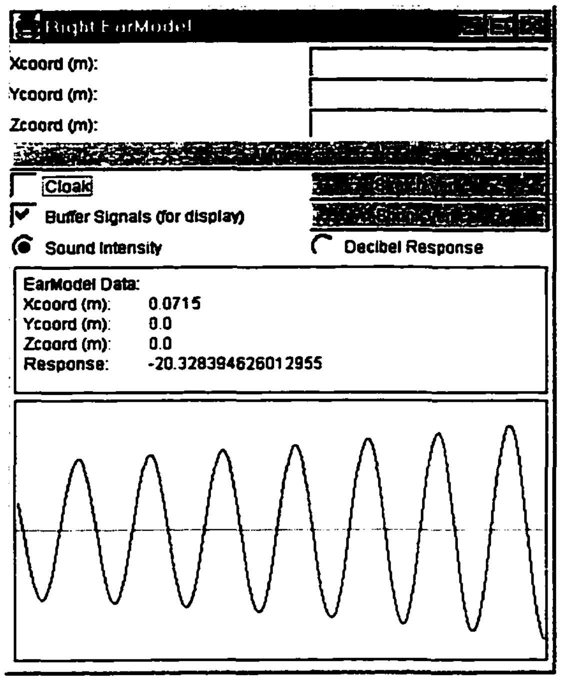
```
MovingSoundSourceModel Data:
Xcoord (m):           0.0715
Ycoord (m):           0.1001562499999965
Zcoord (m):           0.0
Amplitude:            -19.822867355499703
OffsetPhase: (deg.)   0.0715
PowerLevel: (W)       20.0
Frequency: (hz)       10000.0
Delta X Rate (m/s):   0.0
Delta Y Rate (m/s):   -100.0
Delta Z Rate (m/s):   0.0
```

Figure 27: Movement Engaged

For the second trial, the moving sound source was repositioned 3 centimeters directly in front of the right ear model. Also, a Y-coordinate rate of change of -100 meters per second was entered. Then, the 'Run Simulation' button on the SimulatorView panel was again pressed. The initial position of 1 centimeter was used in the first (stationary) trial because after 2 milliseconds, the moving sound source would be positioned about 1 centimeter in front of the right ear model.

In Figure 28, an increase in signal strength from the approaching sound source is clearly apparent. Note also that the frequency is higher than when the sound source was stationary in the first trial. There are about eight cycles in this condition, compared to approximately six in the condition using a stationary sound source.

Although there are no units on the display in this version of the Soundscape simulator, the rise in signal strength and higher frequency are appropriate Doppler effects of an approaching sound source[5].



Xcoord (m):
Ycoord (m):
Zcoord (m):

[ ] Cloak
[✓] Buffer Signals (for display)
[● Sound Intensity] ◯ Decibel Response

```
EarModel Data:
Xcoord (m):     0.0715
Ycoord (m):     0.0
Zcoord (m):     0.0
Response:       -20.328394626012955
```

Figure 28: Dopplered Responses

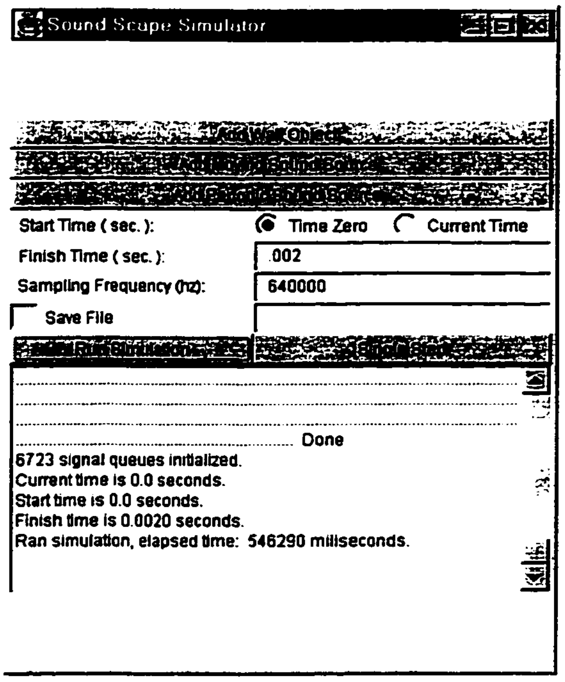
[5] See R.T. Weidner: Physics, Allyn & Bacon 1989.

Wall Echo of Sound's Oblique Approach

This fifth and last use case demonstrates the incorporation of a small echoing surface in the Soundscape simulator. In this scenario, there is a small square and flat echoing surface positioned in front and below the left ear model. There is also a periodic sound source positioned at the origin, driven at 10,000HZ. In order to ignore the ear model's responses to **direct** sound from the periodic sound source, it is muted to both the right and left ear models. Since the sound source is not muted to the wall model, the echoed sounds from the wall model are all that effect the ear models during the simulation.

The following setup steps were taken:

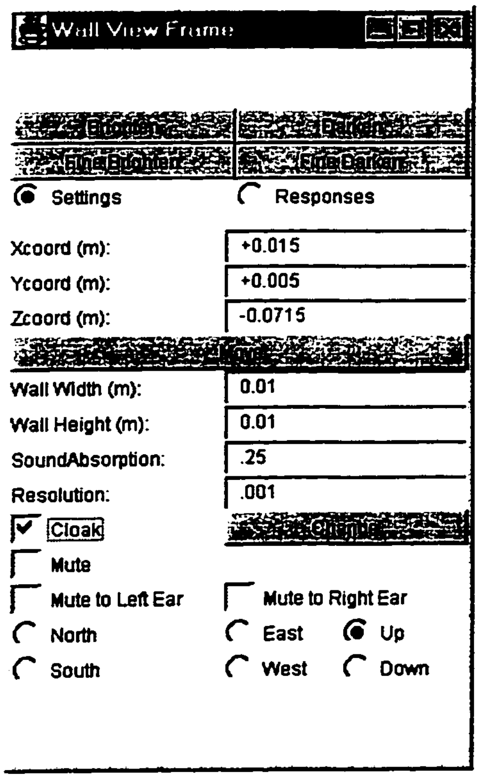
1. The simulator was started, and 'Time Zero' was selected as the simulation start time.
2. A finish time of 2 milliseconds was specified.
3. The simulation sampling frequency was set to 640,000HZ.



Start Time (sec.):	[✓ Time Zero] Current Time
Finish Time (sec.):	.002
Sampling Frequency (hz):	640000
Save File	

```
........................................ Done
6723 signal queues initialized.
Current time is 0.0 seconds.
Start time is 0.0 seconds.
Finish time is 0.0020 seconds.
Ran simulation, elapsed time: 546290 milliseconds.
```

Figure 29: Echo Steps 1-3



[✓ Settings] Responses

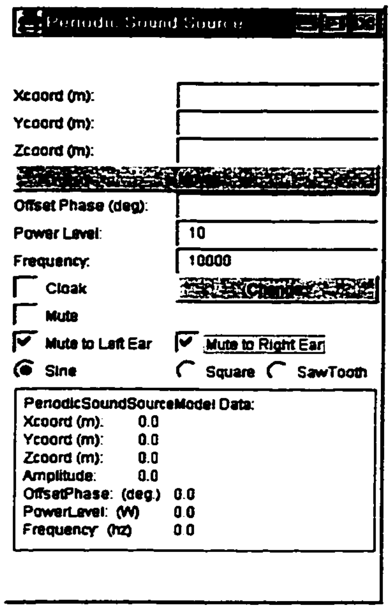
Xcoord (m):	+0.015
Ycoord (m):	+0.005
Zcoord (m):	-0.0715
Wall Width (m):	0.01
Wall Height (m):	0.01
SoundAbsorption:	.25
Resolution:	.001

[✓ Cloak]
Mute
Mute to Left Ear — Mute to Right Ear
North — East — [✓ Up]
South — West — Down

Figure 30: Echo Steps 4-8

4. A wall object was added to the simulation.
5. The new wall object was positioned at coordinates (0.015, 0.005, -0.0715).
6. The wall width and wall height were both set to 1 centimeter.
7. The wall resolution was set to 1 millimeter. The resolution of a wall is the spacing between it's echo points. Since the echo points of the wall are centered, this wall configuration created a 9 x 9 array of echo points.
8. The wall sound absorption was set to .25

9. A periodic sound source was added to the simulation.
10. The sound source was left at it's default position of (0, 0, 0).
11. Power level and driving frequency of the lower sound source were set to 10W and 10,000HZ.
12. The sound source wave shape was observed to be set to 'Sine' by default.
13. Both ear models, the wall model and the sound source were cloaked, and the 'Run Simulation' button was pressed.



Xcoord (m):
Ycoord (m):
Zcoord (m):
Offset Phase (deg):
Power Level: 10
Frequency: 10000
Cloak
Mute
[✓] Mute to Left Ear [✓] Mute to Right Ear
[● Sine] Square SawTooth

```
PeriodicSoundSourceModel Data:
Xcoord (m):       0.0
Ycoord (m):       0.0
Zcoord (m):       0.0
Amplitude:        0.0
OffsetPhase: (deg.)  0.0
PowerLevel: (W)      0.0
Frequency (hz)       0.0
```

Figure 31: Echo Steps 9-12

After the simulation completed it's run, the wall model was un-cloaked. The 'Responses' toggle was clicked to see the wall response view canvas. As shown in Figure 31, the varying isobars of instantaneous sound pressure are shaded according to their relative intensities.

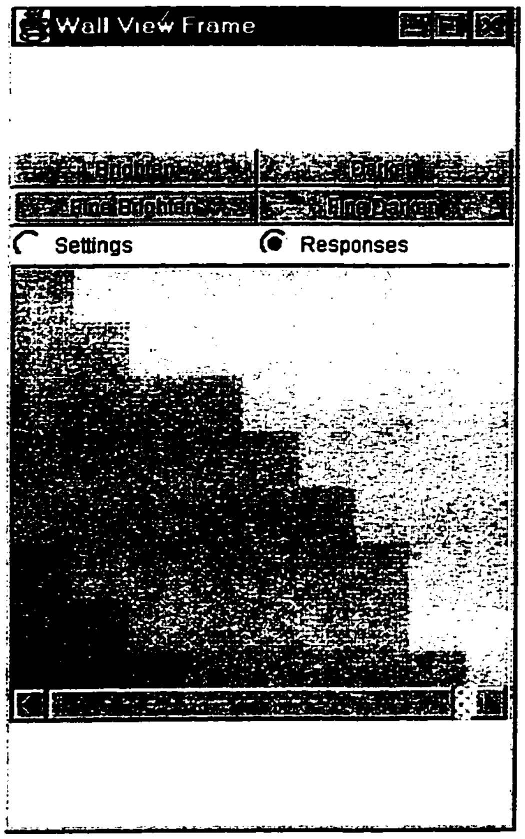


Figure 32: Wall Response View

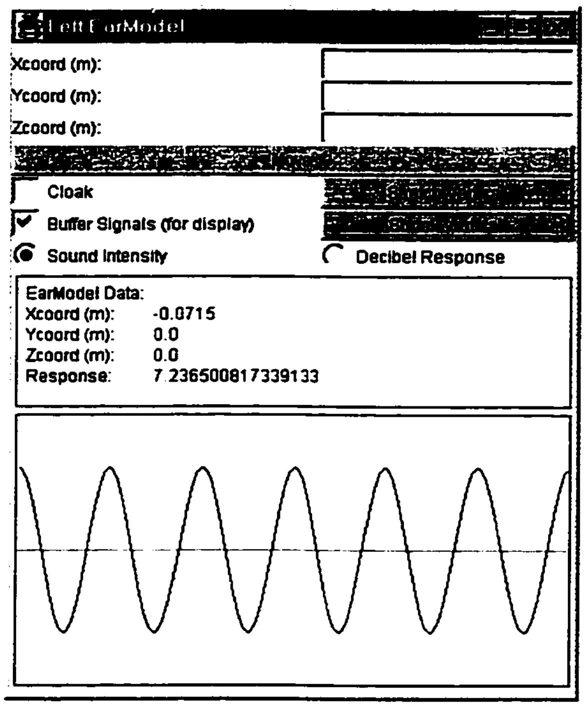
The view in Figure 32 was attained by pressing the 'Brighten' button five times, and then pressing the 'Single Step' button on the Soundscape simulator view panel six times.

In practice, the brightness of the wall response view is difficult to set for best visibility. This is partly due to the fact that when the sound wave shape is in it's trough, the signal effect is negative. So, the 'Darken' button may actually brighten the image.

To help resolve some of this confusion, when the brightness adjustment buttons are pressed on the wall response view panel, an average instantaneous sound intensity for the wall model is logged in the simulator's text log. If the average is a negative value, then the brightness will have opposite effects for that simulation instant.

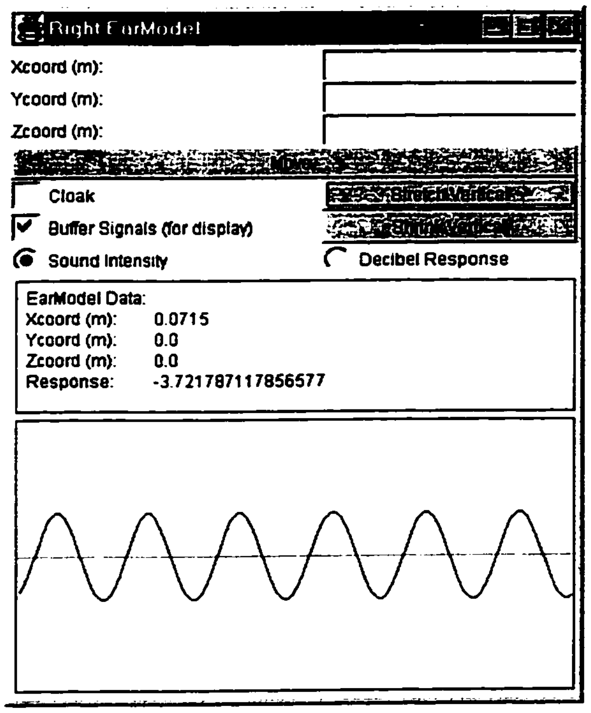
Here are the right and left ear model response views (Figures 3.5.5 and 3.5.6 respectively) after the simulation was run. The echoed sinusoid wave shape still looks sinusoid and the apparent frequency is equal, when compared with simulations involving direct reception of a 10,000HZ sound source.

The response of the left ear model were about twice as strong as the responses of the right ear model. This is because the wall model was positioned closer to the left ear model.



EarModel Data:
Xcoord (m): -0.0715
Ycoord (m): 0.0
Zcoord (m): 0.0
Response: 7.236500817339133

Figure 33: Left Ear Echo Responses



EarModel Data:
Xcoord (m): 0.0715
Ycoord (m): 0.0
Zcoord (m): 0.0
Response: -3.721787117856577

Figure 34: Right Ear Echo Responses

The responses of the right ear model appear to form a more flat wave shape than the responses of the left ear model. This might be because the (more distant and sideways) projection of the wall model is different for the right ear than the left ear model's (more square) projection.

A wall model oriented in an orthogonal way to both a sound source and ear model would echo exactly the same originating wave shape from the sound source. But, as the approach and departure of the sound become oblique, it is easy to speculate about how the breadth of the echoing surface would have a smoothing effect on the resultant echoed sound.

Chapter 4 - Building from Design Patterns

In the last few years, the software development trade has seen a steadily building emphasis upon use of design patterns for both the maintenance of legacy systems and the creation of new software projects. The term "design patterns" has a variety of muddled definitions. Design patterns might be thought of as recipes[6] for composing classes of objects into working software subsystems. Just like recipes (which have their place in a meal), design patterns have different scope of applicability in a software project. Recipes might be for an appetizer, soup, beverage, entree, or dessert. Some design patterns can be demonstrated with a few lines of code, and others are meant to orchestrate entire libraries of classes.

Cookbooks of design patterns have been used to investigate (and *practice*) application of design patterns into this thesis' software. They were referenced to capitalize on expert solutions and to understand distributed systems integration, since CORBA is packed with design patterns[7]. This chapter presents three cases where design patterns were successfully facilitated in the development of the Soundscape auditory scene simulator.

In this chapter, it is shown how the **Model-View-Controller**, **Delegate**, and **Factory Method** design patterns were followed while building the Soundscape simulator. Other design patterns were also followed, namely:

- the **Flyweight** pattern, where the simulator core's SignalQueueManager derivatives all share a polymorphic pool of SoundListener instances whose attributes are externalized
- the **Strategy** pattern, where the SignalQueueDirector employs specially tasked SignalQueueManager derivatives on the fly while a simulation is initialized

There are two major reasons for why these particular patterns were chosen instead of others. **First**, at the time of writing this author has had very limited experience in building software from design patterns. So, many more design patterns may have been applied if an expert pattern architect had written the simulator software. **Second**, in the pattern selection rationale is, of course, that the patterns fit well to build parts of the simulator.

[6] Thanks to Professor Sam "Try thinking of food if you get confused." Sengupta.
[7] See R. Malveau, T.J. Mowbray: CORBA Design Patterns, Wiley 1997.

Application of the Model-View-Controller Pattern

The **Model-View-Controller** (MVC) design pattern is applied in five different collaborations in the Soundscape simulator:

1. as the EarModel, EarView, and EarController collaboration
2. as the PeriodicSoundSourceModel, PeriodicSoundSourceView, and PeriodicSoundSourceController collaboration
3. as the MovingSoundSourceModel, MovingSoundSourceView, and MovingSoundSourceController collaboration
4. as the WallModel, WallView, and WallController collaboration
5. and, as the SimulatorModel, SimulatorView, and SimulatorController collaboration

The Soundscape simulator benefits greatly from several of the MVC design pattern's intrinsic benefits[8]. First and foremost, MVC promotes clear separation of the user-interface **display**, the user-interface **event handling**, and the software **modeling abstraction**. These are three quite separate types of business, and as software projects grow in sophistication most developers would tend to divide them even without application of MVC. Design pattern's promise of eased reuse was exemplified in several instances when MVC was applied in the Soundscape project.

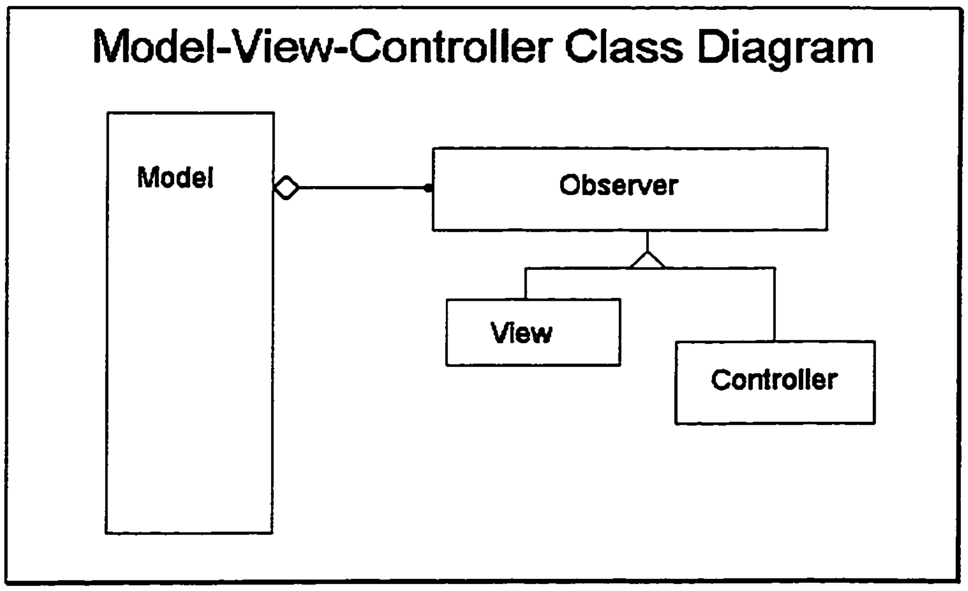


Diagram 1: Model-View-Controller

[8] See F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, M. Stal: Pattern-Oriented Architecture, A System of Patterns, Wiley 1996.

An inviting characteristic of MVC is it's simplicity, as shown above. Model types maintain a list of Observer types. The View and Controller types are derivatives of their Observer base type. The Observer abstract class of objects has one behavior, a method called **update()**. So, when the Model's state changes it may notify any "observing" View and Controller types to update themselves.

In the Soundscape project, Controller types usually only implement their **update()** notification by re-matching their toggle controls (if any) with the (potentially changed) appropriate attribute in the Model they observe. For instance, the EarController will ask the EarModel if it is cloaked, and if so put a check-mark next to it's toggle box during an update. The View typed classes of objects have a lot more to do during an **update()**. For instance, the EarView must check the EarModel's three coordinates, response, and response buffer and then typically display all the new data.

The Model types have the easiest role in the collaboration, because they treat the View and Controllers polymorphically. So, as Observer types come and go during a Model object's life, it does not know them to be Views or Controllers. The Model simply iterates through it's list of Observers and notifies each to **update()** when necessary. In the Soundscape project, Model types also have the capability of being cloaked from their Observers. Internally, the Model always skips **update()** notification while it is cloaked.

One of MVC's greatest advantages is opportunity for reuse, which often reveals itself as software projects grow. In Soundscape's case, the EarController, PeriodicSoundSourceController, MovingSoundSourceController, and WallController classes of objects all reuse code from the PositionHolderController, as seen in Figure-35. A class diagram of this inheritance is shown as Diagram 2, below.

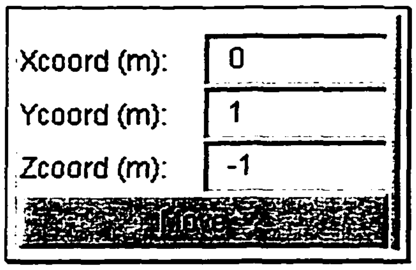

Xcoord (m): 0
Ycoord (m): 1
Zcoord (m): -1

Figure 35: Position Controller

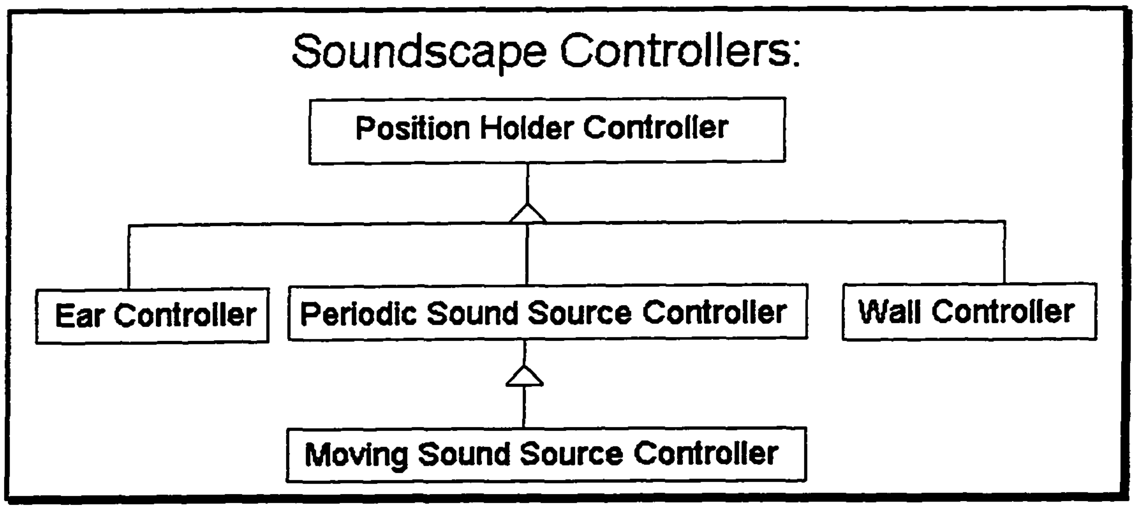

Soundscape Controllers:
Position Holder Controller
Ear Controller
Periodic Sound Source Controller
Wall Controller
Moving Sound Source Controller

Diagram 2: Code Reuse by Controllers

Application of the Delegate Pattern

The **Delegate** design pattern is applied to each of Soundscape's View classes to favor aggregation over inheritance in class relationships[9]. Interface Definition Language (IDL) compilers produce special class definitions called TIE classes in order to facilitate use of the Delegate pattern. As shown below, the EarModel class of objects inherits from an IDL compiler generated class called _IEarModelImplBase.

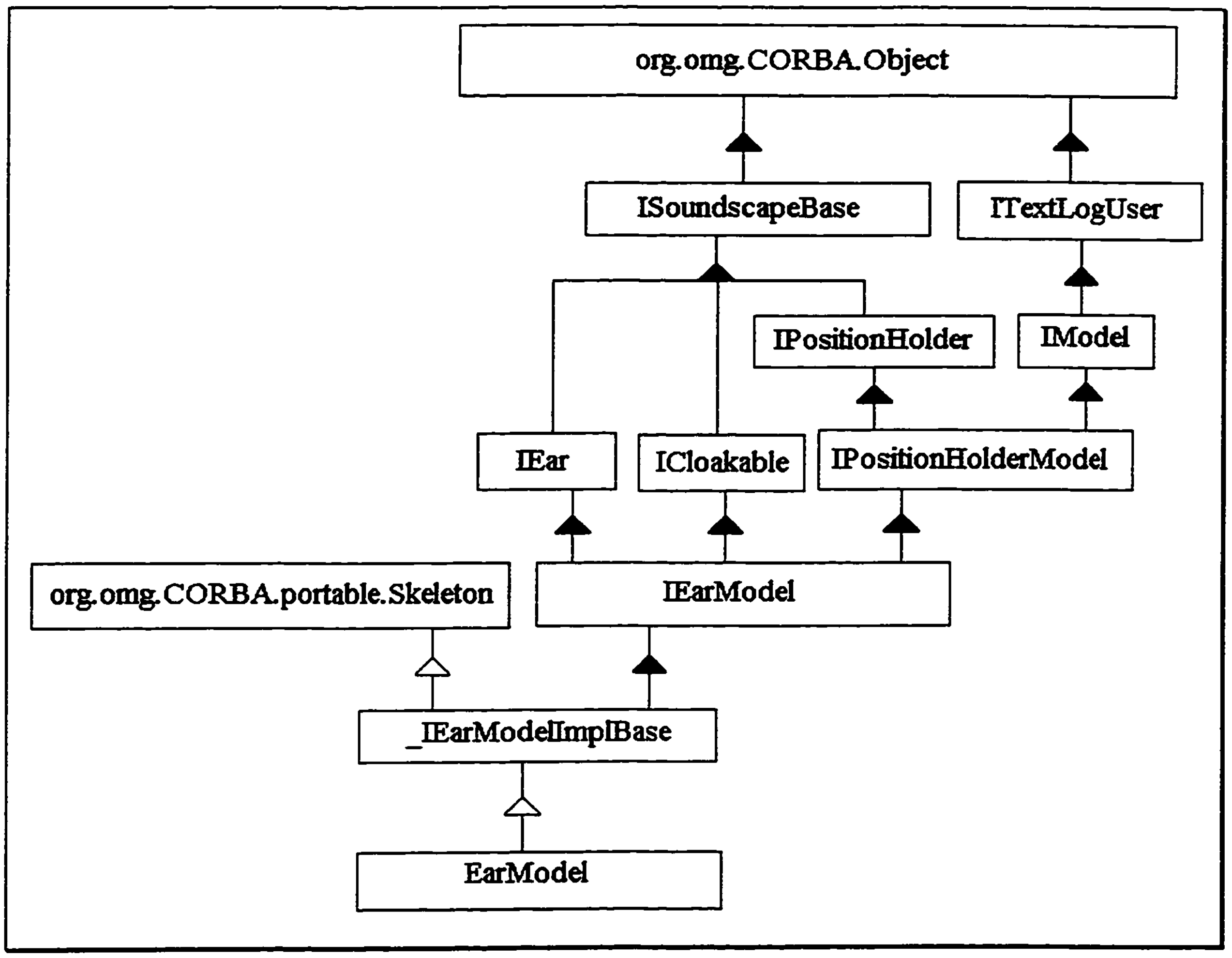


Diagram 3: EarModel Class Inheritance

This is the conventional way in which application implementation code gains the distributed behaviors encompassed by CORBA. Above the EarModel class is it's base class _IEarModelImplBase, which in turn has the Object Request Broker's (ORB) Skeleton class as it's base. On the right half of the diagram there is a hierarchy of interface definitions which are parts of the SoundscapeD IDL module which the EarModel implements. Note though that the right side of the hierarchy is populated with Object Modeling Technique

[9] See D. Harkey, R. Orfali: Client/Server Programming with Java and CORBA, Wiley 1997.

(OMT) black arrow tips, for **virtual** inheritance. The EarModel class (and all Java class definitions) may only inherit from one base class in a non-virtual manner.

When application implementation class definitions already inherit from their single allotted base class, CORBA's TIE mechanism saves them from exclude from CORBA's distributed benefits. As shown in Diagram 4, the application implementation class EarViewFrame inherits from the JDK's Frame class.

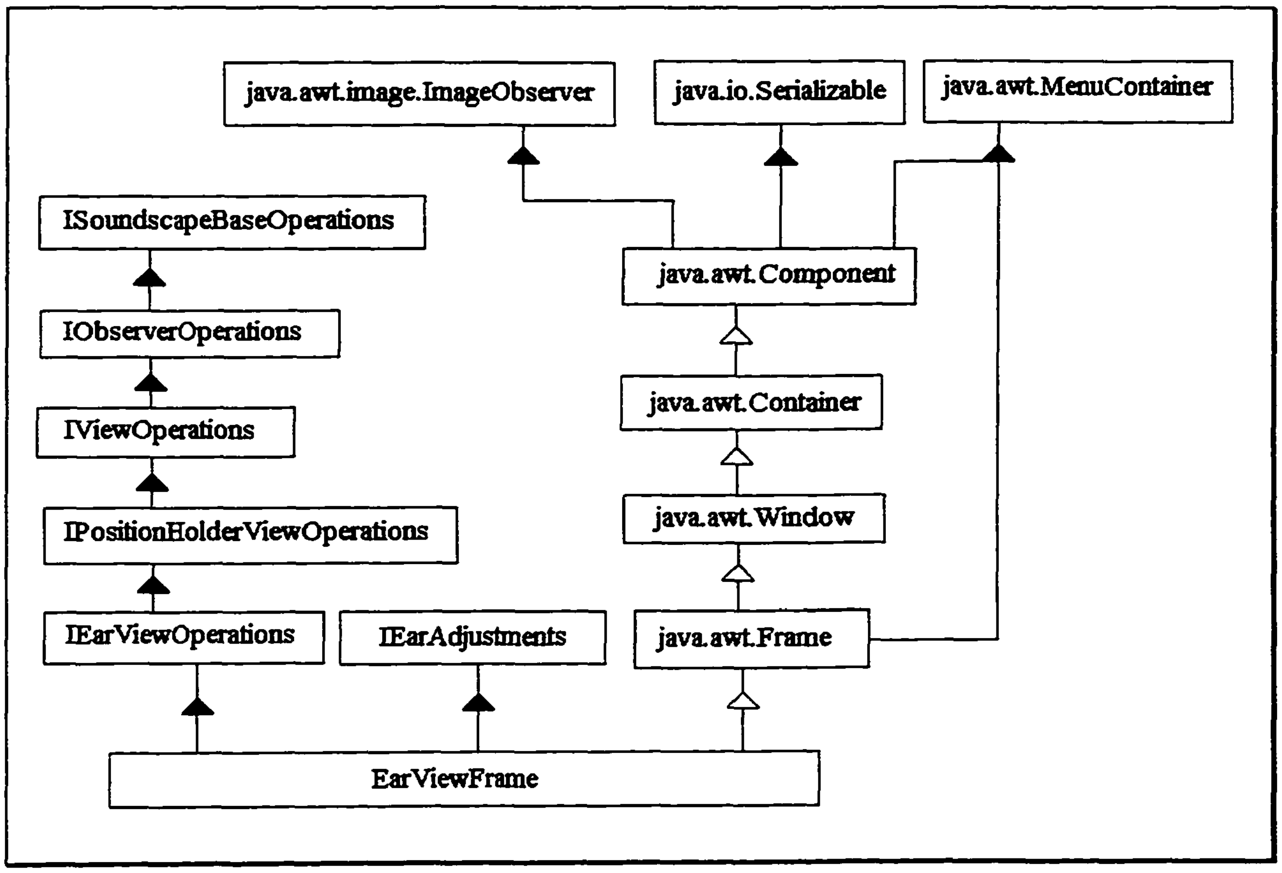


Diagram 4: EarViewFrame Inheritance

The EarViewFrame also implements an IDL compiler-generated interface called IEarViewOperations. This offered as a back door to the distributed world of objects, and is the key to tying the EarViewFrame in with the ORB so that it may meet and communicate with the EarModel class of objects during runtime. The set of behaviors bundled together as IEarViewOperations is a set of actions special to the EarViewFrame. Above the IEarViewOperations interface are other interfaces from which the EarViewFrame accumulates responsibilities. The important thing to note from this diagram is that there is no mention of any CORBA or distributed responsibilities.

In Diagram 5, at the lower left corner we see an IDL compiler-generated class called _tie_IEarView. Above _tie_IEarView, it is apparent from the hierarchy that it is well stitched into the greener grasslands of distributed-ness.

A class called _IEarViewImplBase is shown in the diagram as a base class to _tie_IEarView. Recall that in the earlier diagram of EarModel's hierarchy, the _IEarModelImplBase was the direct parent to the EarModel class. Class definitions following the _*ImplBase naming convention are the usual way to integrating application code in with the IDL compiler-generated classes.

Also, just to the right of the _tie_IEarView class we see that it aggregates something of the type IEarViewOperations.

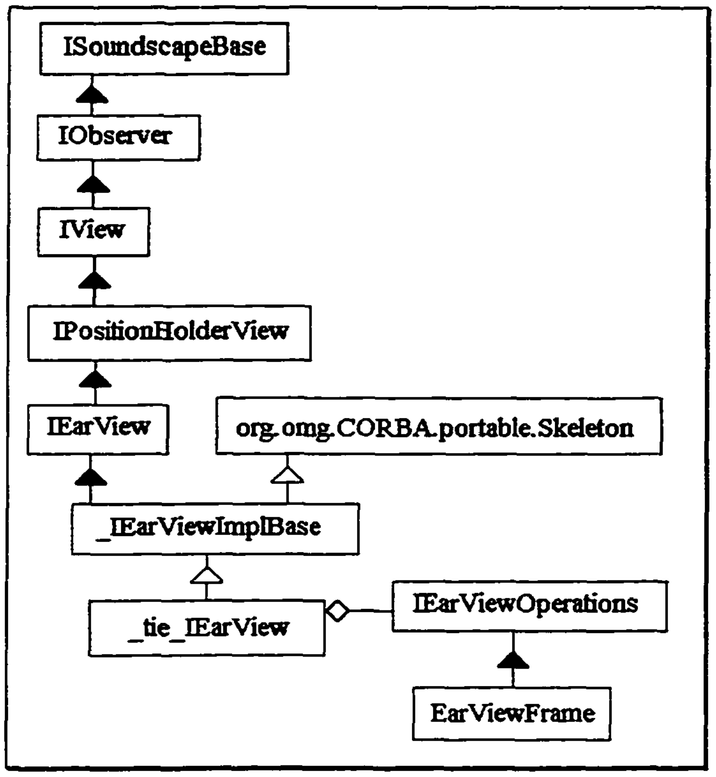


Diagram 5: _tie_IEarView Composition

That "something" is an instance of our EarViewFrame class, which is used by the _tie_IEarView type as a **delegate**. During runtime, if a _tie_IEarView instance is called upon to perform any EarView operations, it delegates them to it's aggregate. Otherwise, if calls are made to a _tie_IEarView instance for CORBA related tasks, it can perform them without delegation.

The only trick left is how to keep references during runtime on both instances, so that we have an object that is distributed, and also part of JDK's abstract window toolkit. The solution has three steps:

1. Create an EarViewFrame object with it's constructor (as usual).
2. Pass the EarViewFrame object into the _tie_IEarView constructor to create a _tie_IEarViewFrame object.
3. Use the _tie_IEarViewFrame object furthermore within the application as an IEarViewOperations reference with distributed capabilities.

The EarViewFrame reference from step 1 should be discarded once step 2 is complete. The _tie_IEarViewFrame reference is the full featured reference which ought to be retained and used.

Application of the Factory Method Pattern

The **Factory Method** design pattern is applied in the SimulatorController and SimulatorModel class definitions of both SoundscapeSA and SoundscapeD. Factory methods and/or classes are used when the assembly of an object and it's assimilation into an appropriate software subsystem is non-trivial[10]. On the following pages are two excerpts of source code from the SimulatorModel and SimulatorController class definitions.

As is usually the case when applying design patterns, there were valuable reuse opportunities which were revealed while putting the Factory Method pattern in place. Specifically, the factory method implementation for one simulation model was easily cannibalized to clone other factory methods for other simulation models. The sequence of steps involved in activating a new simulation model is strict and lengthy, but largely similar for the various simulation models.

[10] See E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley 1995.

Example Model Factory Method

The C++ source code excerpt below is from the SimulatorModel class definition.

```
// ----------------------------------------------------------------
// -    CREATE, REGISTER, AND INTEGRATE INTO THE SIMULATION
// -    ONE NAMED MovingSoundSourceModel.
// ----------------------------------------------------------------
void SimulatorModel::newMovingSoundSourceModel( const char* IN_strInstanceName )
{
    char* strInstanceName = NULL;
    CORBA::ORB_ptr pORB = NULL;
    CORBA::BOA_ptr pBOA = NULL;
    MovingSoundSourceModel* pNewMovingSoundSourceModel = NULL;
    IMovingSoundSourceModel_ptr pNewMovingSoundSourceModel_ptr = NULL;

    try
    {
        // Make concrete copy of instance name
        strInstanceName = new char[ strlen( IN_strInstanceName ) + 1 ];
        strInstanceName = strcpy( strInstanceName, IN_strInstanceName );

        // Initialize the ORB and BOA.
        pORB = CORBA::ORB_init();
        pBOA = pORB->BOA_init();

        // Create the MovingSoundSourceModel object.
        getTextLog()->logText( "Creating a MovingSoundSourceModel object named '" );
        getTextLog()->logText( strInstanceName );
        getTextLog()->logText( "' \n" );
        pNewMovingSoundSourceModel = new MovingSoundSourceModel( pszMyViewInstanceName, strInstanceName );
        pNewMovingSoundSourceModel_ptr = pNewMovingSoundSourceModel;

        // Export to the ORB newly created object.
        getTextLog()->logText( "Exporting the test MovingSoundSourceModel object.." );
        pBOA->obj_is_ready( pNewMovingSoundSourceModel_ptr );

        // Notify the SignalQueueDirector to assimilate
        // the new MovingSoundSource into the simulation.
        pSignalQueueDirector->addMovingSoundSource( *pNewMovingSoundSourceModel );

        pORB = NULL;
        pBOA = NULL;
        pNewMovingSoundSourceModel = NULL;
        pNewMovingSoundSourceModel_ptr = NULL;
        delete strInstanceName, strInstanceName = NULL;
    }
    catch( const CORBA::Exception& e )
    {
        cerr << "In SimulatorModel::newMovingSoundSourceModel, a CORBA::Exception has occured: " << e << endl;
    }
}
```

After some variable declarations, just inside the **try** block are the steps encapsulated by this factory method:

1. Concrete copy is made of the MovingSoundSourceModel's instance name.
2. The ORB and Basic Object Adapter (BOA) are initialized.
3. The new MovingSoundSourceModel is created.
4. The named MovingSoundSourceModel is registered with the BOA.
5. Assimilation of the new MovingSoundSourceModel is made by the SignalQueueDirector.

Also necessary are the nullification and memory deallocation of the method's local variables. The **try-catch** block is required for initialization of the ORB and BOA.

Example View Factory Method

The Java source code excerpt below is from the SimulatorController class definition.

```
protected void NewMovingSoundSource()
{
    ORB myORB = null;
    BOA myBOA = null;
    String strInstanceName = tfNewMovingSoundSourceName.getText();

    try
    {
        // Initialize the ORB and BOA
        System.out.println("Initializing the ORB");
        myORB = ORB.init();
        myBOA = myORB.BOA_init();

        // Attempt bind to the Test MovingSoundSourceModel Object
        System.out.println( "Building MovingSoundSourceModel Object named '"
                                                  + strInstanceName + "'" );
        ( mySimulatorView.getSimulatorModel() ).newMovingSoundSourceModel( strInstanceName );

        // Create a MovingSoundSourceViewFrame Object
        MovingSoundSourceViewFrame NewMovingSoundSourceView =
            new MovingSoundSourceViewFrame( strInstanceName );

        // Seed the TIE object with the MovingSoundSourceViewFrame reference
        System.out.println( "Created MovingSoundSourceViewFrame object,
                                  tying MovingSoundSourceViewFrame in with the ORB.." );
        _tie_IMovingSoundSourceView NewTIEMovingSoundSourceView =
            new _tie_IMovingSoundSourceView( NewMovingSoundSourceView, ( strInstanceName + "View") );
        NewTIEMovingSoundSourceView.initializeGUI();

        // Export to the ORB newly created object.
        myBOA.obj_is_ready( NewTIEMovingSoundSourceView );
    }
    catch( SystemException e )
    {
        System.err.println( "System Exception" );
        System.err.println( e );
    }
}
```

This method accompanies the previous page's method in creating a new MovingSoundSource Model-View pair. The MovingSoundSourceController is created and maintained by the MovingSoundSourceView. After some variable declarations, just inside the **try** block are the steps encapsulated by this factory method:

1. The ORB and Basic Object Adapter (BOA) are initialized.
2. A call to the SimulatorModel is made for a new MovingSoundSourceModel to be named and bound with the local BOA.
3. A named TIE MovingSoundSourceView is created and initialized.
4. The new TIE MovingSoundSource is registered with the local BOA.

Chapter 5: Gateway

Why a Distributed Simulator Deployment?

The computational costs, in time and resources, were disappointing when development of the Soundscape simulator first began. With an average desktop PC, only very simple auditory scene simulations were feasible. In hopes of overcoming these limitations, a distributed version of the Soundscape simulator was written. An intrinsic benefit of distributed systems is their heightened potential for computing power, which is exactly what was desired for the Soundscape simulator.

Along with the increases in potential, Soundscape-D exercises CORBA's programming language neutrality. So, pieces of the simulator which were known to be computationally intensive were rewritten in (more efficient) C++ source code[11]. Sacrifices in terms of portability were avoided because the user interface remained written in Java, and retains the same platform independence[12] as the stand-alone version's user interface.

Thanks to design patterns and MVC, the dichotomy of the simulator (user interface versus modeling abstractions) was made clear. Therefore, the re-development of the simulator's modeling abstractions and back-end code in C++ was simple for three reasons:

1. Identification of components which needed to be redeveloped was straight forward.
2. Since the stand-alone version (Soundscape-SA) was already operational, much of the development planning was already complete.
3. Because of the syntactic similarities between Java and C++, the transcription to target "peer class" definitions was largely a cut and paste operation.

[11] See M.R. Headington, D.D. Riley: Data Abstractions and Structures Using C++, Heath 1994.
[12] See K. Arnold, J. Gosling: The Java Programming Language 2nd Edition, Addison-Wesley 1996.

It is hoped that a distributed auditory scene simulator will open new opportunities for growth of auditory scene simulation. For instance, CORBA's specifications on object persistence might be used to more easily architect very large and detailed auditory scenes from a public accumulation of "stock" simulation models. Also, simultaneous multiple user participation in a simulation may be made possible. Lastly (for now) CORBA specifies dynamic revision control ("trader") of applications themselves[13], allowing users to have components of the simulator itself to be automatically updated when this CORBA service becomes available.

[13] See T.J. Mowbray, R. Zahavi: The Essential CORBA: Systems Integration Using Distributed Objects, Wiley 1995.

Overview of the Soundscape-D Interface Definition Module

The Interface Definition Language (IDL) module for Soundscape-D is written (as usual for IDL files) to define sets of services for each class of distributed objects in the application. A copy of the SoundscapeD.idl is included in Appendix A for more concrete understanding of how the simulator's sets of services are defined. As mentioned earlier, the modeling abstractions of the simulator and all computationally intensive components of the simulator were identified to be rewritten in C++. The user interface remains sourced in Java. The two software subsystems communicate at runtime through their language-specific ORBs which in turn communicate via CORBA's Internet Inter-ORB Protocol (IIOP) as sketched below.

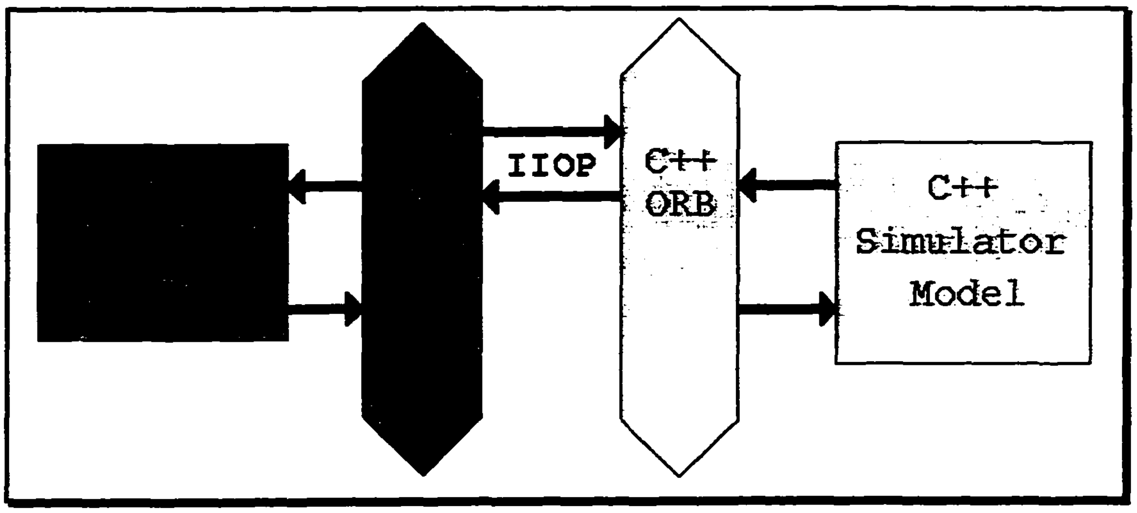


Figure 36: IIOP Binds Java and C++ Sourced Components at Runtime

From a developmental perspective, this mechanism is transparent. The only decisions to be made are, which classes to write in C++ and which to leave in Java source code. When the partial migration to C++ was planned for the Soundscape simulator, *everything* which wasn't part of the user-interface was re-implemented in C++.

Diagram 6 shows an interface hierarchy of all the user-interface related interfaces included in the Soundscape-D IDL module. These are implementations to be left in Java. Note that the interfaces as defined in the IDL module all begin with a capital 'I' to denote them as interfaces. For diagramming purposes, the 'I' is left off, as this diagram (and the next) only show interface definitions.

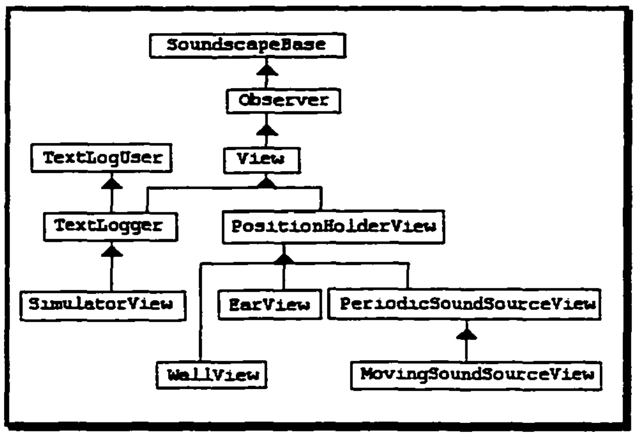


Diagram 6: User Interface Components

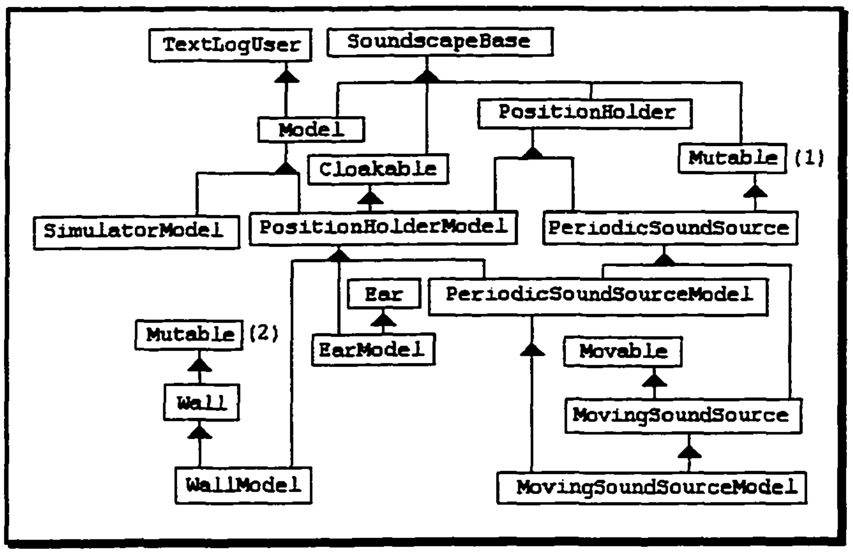


Diagram 7: Abstracted Modeling Components

Above, in Diagram 7, a diagram showing the hierarchy of modeled abstractions displays all the interfaces which were re-implemented in C++ source code. Note that the Mutable interface appears twice on the diagram. There is only one Mutable interface, but because of this particular diagram's layout it appears twice to avoid crossing lines of inheritance.

Costs / Gains Comparison of Soundscape-D to Soundscape-SA

For a comparison of computing costs, the "Doppler Effect of an Approaching Sound Source" use case of Chapter 3 was performed. The use case steps were all followed exactly, except that the simulation's finish time was set longer to one full second. The results presented are from the second half of the use case, with the movement of the sound source engaged.

Although designed for network deployment, Soundscape-D had not been tested as such at the time of this writing. So, this comparison was performed with both configurations running on a single machine. Java's (v1.1.5) just-in-time compilation (JIT) was activated for the example, boosting the Java portions of performance approximately 25%.

	Soundscape-SA	Soundscape-D	% change
Class/Interface Definitions	80	379	+473
Compiled Size	135KB	557KB	+413
Runtime Memory	5,052KB	11,760KB	+233
Elapsed Time	32.807 sec.	22.432 sec.	-31.6
Millions of Instructions Performed	12,795	8,748	-31.6

Table 1: Results of a Soundscape-SA/Soundscape-D Comparison

Development Costs: The first row of Table 5.4 shows the most remarkable change. Development of the Soundscape-D version required more than four times the code that was required for Soundscape-SA. In the second row of Table 5.4 there is a roughly matching increase in the compiled size of the Soundscape-D product- a bit more than four times the compiled size of the compiled Soundscape-SA product.

Memory Cost: Taking up well over twice as much memory as the stand-alone version, Soundscape-D required about eleven megabytes (third row of Table 5.4). This number is the sum of both ORBs' object-server agents (2,144KB), the C++ sourced SimulatorModel executable (3,020KB), and the Java interpreter(6,608KB), which was running the SimulatorView.

Performance Gain: The good news is that, even with the overhead of running two ORBs on a single machine, Soundscape-D shaved 31% off of the stand-alone version's time consumption.

Future Improvements

Certainly, there is much room for growth of the Soundscape auditory scene simulator. Feasible priority options are:

1. enhanced file I/O capabilities, including some sort of viewer for the simulator's output files
2. refinements in the user-interface, including scale of unit measurement along the graphical displays
3. a better populated library of simulation objects, perhaps including oscillating strings and surfaces
4. availability of more verbose logs of simulations

Re-prioritization of these and other possible improvements may be influenced by feedback from auditory scene experimentalists.

Conclusion

This thesis work has provided the Soundscape simulator as a working prototype tool with which to facilitate auditory scene simulations. It stands as a lightly-tested ground work for further development of auditory scene simulation systems which may support studies of human audition. A basic operational protocol was established for experimenting with simulated auditory scenes, the principles of the Soundscape simulator were described, and computer system requirements of the simulator were specified. Detailed examples were presented as proof that the Soundscape simulator accurately emulates the effects of attenuation, delay and echoed sound during it's simulation of simple digital auditory scenes.

Although limited at this early stage of software development, the simulator's infrastructure was shown to be based on design patterns as solid, extensible and reusable expert architectural software components. As hard evidence of the simulator's extensibility, a CORBA distributed version was developed and shown to be a successful endeavor. Once distributed, the Soundscape auditory scene simulator was shown to have grown in both speed and size, straining the limits of today's information technology for elusive answers about how living beings listen.

Bibliography

E. Anuff: The Java Sourcebook, Wiley 1996.

K. Arnold, J. Gosling: The Java Programming Language 2nd Edition, Addison-Wesley 1996.

J.R. Jackson, A.L. McClellan: Java by Example, SunSoft Press of Prentice Hall 1996.

D.J. Berg, J.S. Fritzinger: Advanced Techniques for Java Developers, Wiley 1997.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, M. Stal: Pattern-Oriented Architecture, A System of Patterns, Wiley 1996.

M.C. Daconta: Java for C/C++ Programmers, Wiley 1996.

H.M. Deitel, P.J. Deitel: C++ How to Program, Prentice Hall 1994.

H.M. Deitel, P.J. Deitel: Java How to Program 2nd Edition, Prentice Hall 1998.

R. Duda: "Sound Localization Research" San Jose State University College of Engineering (http://www-engr.sjsu.edu/~duda/Duda.Research.html); San Jose, California 1998.

K. Duddy, A. Vogel: Java Programming with CORBA, Wiley 1997.

E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley 1995.

D. Harkey, R. Orfali: Client/Server Programming with Java and CORBA, Wiley 1997.

M.R. Headington, D.D. Riley: Data Abstractions and Structures Using C++, Heath 1994.

C.S. Horstmann: Mastering Object-Oriented Design in C++, Wiley 1995.

R. Malveau, T.J. Mowbray: CORBA Design Patterns, Wiley 1997.

T.J. Mowbray, R. Zahavi: The Essential CORBA: Systems Integration Using Distributed Objects, Wiley 1995.

R. Showen, J. Dunham, A.C. Phillips, S. Sandoval: "World's First Proven Gunshot Location System!" Trilon Technology (http://www.shotspotter.com); Los Altos, California 1998.

R.T. Weidner: Physics, Allyn & Bacon 1989.

Appendix A: Essay: Java as a Modeling Language for C++ Projects

"Java as a Modeling Language for C++ Projects"

Expense and Schedule of Medium Sized Software Development Projects

What is meant by a medium-sized software project in this context is a full-featured and contemporary first-revision software application. One of roughly 100,000 lines of net, high level language source code. In order to appreciate the significance of such an undertaking, consider the development costs in dollars and man-years of labor:

Of the members of a hypothetical development team, the gross average cost in compensation to the company is $50,000 per year. Each year, the developers work 240 full days, producing 12 lines of source code per day. So, each man-year translates into 2,880 lines of shippable source code. At this rate, there are 34.72 man-years of labor involved in the **development** of the product's first run. Now we're talking over $1.7 million in costs, to develop a working software package.

Importance of A Modeling Phase

The problem with the above situation is that the company has staffed a dozen full time employees, waited three years, and poured a couple million dollars into a product before it's marketing department has had a chance to try it. Ideally, since the specification probably came from Marketing, the product should be exactly as expected.

However, in the real world (and especially in the software industry) the Engineering department must be given plenty of room for interpretation, and surprises often occur. Without some way to monitor (and respond to) the shaping of the product during development, the surprises could be numerous and (at $17 per line of code) expensive. An early product **model**, updated frequently throughout the evolution of the software product, would surely ease some potential shock and surprise.

Most successful contemporary software development projects **are** phased in industry. Phasing the project schedule gives the development group a unified spirit in what corporate expectations are on a quarterly basis, and provides a means for checkpoints in development progress. A modeling phase, scheduled as an ordinary development phase but toward the beginning of a project life, would provide a working and explicit example of how the software product is evolving on a weekly basis. By furnishing a working model, the burden of revising specifications and exchanging other technical corporate information is lightened. In many cases, members will be enabled to **show** what they mean, rather than writing lengthy technical descriptions.

Java as a Modeling Language Candidate

Especially in cases where the target software package is known to be sourced in C++, the Java programming language is best fit as a prototypical source language option. The strong reasons for Java's candidacy fall into three main categories: Low Code Liability, Platform Independence, and Eased Learning Curve.

Low Code Liability

By code liability, I mean the expense in time and money caused by dealing with large volumes of source code. Code liability slows the development process because even in the best modular designs, a significant revision in one place means a survey of that code area and other appropriate changes in neighboring modules. Comments embedded in source code must be revisited to properly annotate effected sections. Test programs and other supporting work must also be revised.

More source code means more debugging, more comments, and more delay involved in implementing software design revisions. Java has one source file per class definition, as opposed to two or three files per class definition for C++ (for the header, resource header, and implementation files). Rudimentary and time-consuming tasks such as revision control, make file maintenance, naming standardization, and code navigation are eased simply by the lack of unneeded code in Java-sourced implementations.

Platform Independence

Developers modeling a project in Java may easily work concurrently on their platform of choice, be they Macintosh, PC, or Unix workstations. The Java virtual machine lies a common ground on which platform-specific development sub-teams can meet and communicate. It is a working and available model for a hypothetical platform, easing cross platform migration. As necessary, non development personnel involved would be allowed to purview a web-demonstration through an Internet browser (without having a development workstation or licensed development software).

Separation of application services from the user interface is persuaded by Java's elegant event model, encouraging reusable back-end architectures. Prudent design patterns are encouraged through Java's syntax (distinguished interfaces and single inheritance) and type strength. Performance bottlenecks common to all platforms may be discovered and addressed earlier in development through use of the Java interpreter. Finally, several CORBA implementations are available, enabling multi-language development and fielding of software systems. This means that the components of a Java software model may be transcribed into native-compilable in piece-wise progression.

Eased Learning Curve

The Java language is meant to improve on the strengths of the C++ language, and diminish it's weaknesses while changing as little as possible. Details such as pointer syntax and garbage collection may be foregone while modeling in Java. Java's streamlined syntax provides a stirrup for trade veterans to get a leg up on object-oriented design and implementation. And, for experienced object-oriented developers, Java provides another way to create object-oriented systems, strengthening understanding and perhaps precipitating new architectural concepts. Teams related to product deployment, but outside of the core development staff might use a Web demonstration for familiarization, preliminary testing and documentation.

Concerns of Modeling a Project With Java

1.) The Java Development Kit is evolving quickly. One version should be picked, and a concerted team effort will probably be required to improve existing Java modeling source code to updated revisions of the JDK.

2.) Revision control of Java source code might not be available for concurrent development on multiple platforms.

3.) Native platform implementations and ActiveX controls compatible code should be rewritten in pure Java source code if multiple platform deployment of a software project is targeted.

4.) Detailed user-interface implementations should be avoided (and may be impossible) during Java modeling.

Summary

Today's software development projects are expensive and time consuming, usually involving multiple development phases. For software aimed at multiple platform markets, Java is an ideal modeling language. The Java virtual machine lies as a perfect common ground and a hypothetical platform. By using it during early development, talented software engineers may address a majority of their architectural and implementation eventualities up front.

References:

Anuff, Ed The Java Sourcebook. Wiley Computer Publishing, 1996.

Davis, Stephen R. Learn Java Now. Microsoft Press, 1996.

Deitel H.M. and Deitel P.J. Java How to Program. 2nd edition Prentice Hall, 1997.

Duddy, Keith and Vogel, Andreas Java Programming with CORBA. Wiley Computer Publishing, 1997.

Jackson, Jerry R., and McClellan, Alan L. Java by Example. SunSoft Press, 1996.

Appendix B: Java Documentation for Soundscape-SA Source Code

Class Hierarchy

- class java.lang.Object
  - class java.awt.CheckboxGroup (implements java.io.Serializable)
    - class BufferViewCardSelector
    - class DirectionSelectionGroup
    - class StartTimeSelector
    - class WallViewCardSelector
    - class WaveSelectionGroup
  - class CircularDoubleArray (implements ICircularDoubleArray)
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Canvas
      - class ViewCanvas
        - class BufferViewCanvas
          - class DecibelResponseViewCanvas
          - class SoundIntensityViewCanvas
        - class PositionHolderViewCanvas
          - class EarViewCanvas
          - class PeriodicSoundSourceViewCanvas
            - class MovingSoundSourceViewCanvas
    - class java.awt.Container
      - class java.awt.Panel
        - class java.applet.Applet
          - class SoundscapeApplet (implements java.awt.event.ActionListener)
        - class PositionHolderController (implements java.awt.event.ActionListener, IController)
          - class EarController (implements java.awt.event.ActionListener, java.awt.event.ItemListener, IController)
          - class PeriodicSoundSourceController (implements java.awt.event.ActionListener, java.awt.event.ItemListener, IController)
            - class MovingSoundSourceController (implements java.awt.event.ActionListener, IController)
          - class WallController (implements java.awt.event.ActionListener, java.awt.event.ItemListener, IController, ITextLogUser)
        - class ResponseBufferPanel
        - class SimulatorController (implements java.awt.event.ActionListener, IController, java.awt.event.ItemListener, ITextLogUser< /a>)
        - class WallViewPanel
      - class java.awt.ScrollPane
        - class WallResponsesViewCanvas
      - class java.awt.Window
        - class java.awt.Dialog
          - class POCPopUp (implements java.awt.event.ActionListener)
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class EarViewFrame (implements IEarView)
          - class MovingSoundSourceViewFrame (implements IView, IObserver, ITextLogUser)
          - class PeriodicSoundSourceViewFrame (implements IPeriodicSoundSourceView)
          - class SimulatorView (implements ISimulatorView)
          - class WallViewFrame (implements IWallView, java.awt.event.ItemListener)
  - class Direction (implements IDirection)

- class EarModel (implements IEarModel, ICloakable, ITextLogUser)
- class EchoPoint (implements IEchoPoint)
- interface ICircularDoubleArray
- interface ICloakable (extends ISoundscapeBase)
- interface IController
- interface IDirection
- interface IEar (extends ISoundListener)
- interface IEarModel (extends IEar, ICloakable, IPositionHolderModel)
- interface IEarView (extends IPositionHolderView, IEarViewAdjustments)
- interface IEarViewAdjustments
- interface IEchoPoint (extends ISoundListener, ISoundSource)
- interface IModel (extends ISoundscapeBase, ITextLogUser)
- interface IMovable
- interface IMovingSoundSource (extends IPeriodicSoundSource, IMovable)
- interface IMovingSoundSourceModel (extends IPeriodicSoundSourceModel, IMovingSoundSource)
- interface IMutable (extends ISoundscapeBase)
- interface IObserver (extends ISoundscapeBase)
- interface IPeriodicSoundSource (extends ISoundSource)
- interface IPeriodicSoundSourceModel (extends IPositionHolderModel, IPeriodicSoundSource, ICloakable)
- interface IPeriodicSoundSourceView (extends IPositionHolderView)
- interface IPositionHolder (extends ISoundscapeBase)
- interface IPositionHolderModel (extends IModel, IPositionHolder)
- interface IPositionHolderView (extends IView)
- interface ISignalQueueMgr
- interface ISimulatorModel (extends IModel)
- interface ISimulatorView (extends IView, ITextLogger)
- interface ISoundListener (extends IPositionHolder)
- interface ISoundSource (extends IMutable, IPositionHolder)
- interface ISoundscapeBase
- interface ITextLogUser
- interface ITextLogger (extends IView, ITextLogUser)
- interface IThreeDimensionalPoint (extends IPositionHolder, ISoundscapeBase)
- interface IView (extends IObserver)
- interface IWall (extends IMutable)
- interface IWallModel (extends IWall, ICloakable, IPositionHolderModel)
- interface IWallView (extends IPositionHolderView)
- class PeriodicSoundSourceModel (implements IPeriodicSoundSourceModel)
  - class MovingSoundSourceModel (implements IMovingSoundSourceModel)
- class Signal
- class SignalQueue
- class SignalQueueDirector (implements ITextLogUser)
- class SignalQueueElement
- class SignalQueueMgr (implements ISignalQueueMgr)
  - class SoundSourceSignalQueueMgr (implements ISignalQueueMgr)
    - class MovingSoundSourceSignalQueueMgr (implements ISignalQueueMgr)
  - class WallSignalQueueMgr (implements ISignalQueueMgr)
- class SimulatorModel (implements ISimulatorModel, ITextLogUser)
- class ThreeDimensionalPoint (implements IThreeDimensionalPoint)
- class WallModel (implements IWallModel, ITextLogUser)

Index of all Fields and Methods

dElevation_InRadians. Variable in class Direction
dequeueSignal(). Method in class SignalQueue
deriveCornerPoints(). Method in class WallModel
detach(IObserver). Method in class EarModel
detach(IObserver). Method in interface IModel
THIS MODIFIER UN-REGISTERS AN Observer (EITHER AN USER-INTERFACE View OR A USER-INTERFACE Controller) FROM IT'S Model.
detach(IObserver). Method in class PeriodicSoundSourceModel
detach(IObserver). Method in class SimulatorModel
detach(IObserver). Method in class WallModel
dFinishTime. Variable in class SimulatorController
dFinishTime. Variable in class SimulatorModel
dFrequency. Variable in class PeriodicSoundSourceModel
dInitialXCoord. Variable in class MovingSoundSourceModel
dInitialYCoord. Variable in class MovingSoundSourceModel
dInitialZCoord. Variable in class MovingSoundSourceModel
Direction(double, double). Constructor for class Direction
Direction(double, double, String). Constructor for class Direction
disengageResponseBuffer(). Method in class EarViewFrame
disengageResponseBuffer(). Method in interface IEarViewAdjustments
THIS MODIFIER DISENGAGES THE EarViewFrame's BUFFERED ARRAYS OF RESPONSES, FOR USER DISPLAY
display(). Method in class EarViewFrame
display(). Method in interface IThreeDimensionalPoint
This method displays the coordinates of an IThreeDimensionalPoint instance at the system's standard output.
display(). Method in class ResponseBufferPanel
display(). Method in class ThreeDimensionalPoint
display(). Method in class WallViewFrame
display(). Method in class WallViewPanel
displayAvgResponse(). Method in interface IWall
THIS METHOD CALCULATES THE AVERAGE RESPONSE VALUES FOR ALL THE EchoPoints THE WALL HAS.
displayAvgResponse(). Method in class WallModel
displayCoords(). Method in class EchoPoint
displayTimeConstraints(). Method in class SimulatorController
distanceTo(IThreeDimensionalPoint). Method in interface IThreeDimensionalPoint
This method returns the distance in meters to the given IThreeDimensionalPoint instance.
distanceTo(IThreeDimensionalPoint). Method in class ThreeDimensionalPoint
distanceToTarget(). Method in class Signal
dNegligibleSoundIntensityFloor. Static variable in class SignalQueueMgr
dOffsetPhase. Variable in class PeriodicSoundSourceModel
dOffsetPhase_InRadians. Variable in class PeriodicSoundSourceModel
dosOutputFile. Variable in class SimulatorModel
DOWN. Static variable in class Direction
dPowerLevel. Variable in class PeriodicSoundSourceModel
dRadialFrequency. Variable in class PeriodicSoundSourceModel
drawBorder(Graphics). Method in class ViewCanvas
drawEarText(Graphics). Method in class EarViewCanvas
drawHeader(Graphics). Method in class ViewCanvas
drawMovingSoundSourceText(Graphics). Method in class MovingSoundSourceViewCanvas
drawPeriodicSoundSourceText(Graphics). Method in class PeriodicSoundSourceViewCanvas
drawPositonText(Graphics). Method in class PositionHolderViewCanvas
drawTextRow(Graphics, String, String). Method in class ViewCanvas
dResolution. Variable in class WallModel
dResponse. Variable in class EchoPoint
dResponseBuffer. Variable in class EarViewCanvas

getPeriodicSoundSourceModel(). Method in class MovingSoundSourceViewFrame
getPeriodicSoundSourceModel(). Method in class PeriodicSoundSourceViewFrame
getPositionHolderModel(). Method in class EarViewFrame
getPositionHolderModel(). Method in interface IPositionHolderView
This accessor returns a reference to the PositionHolderView object's associated PositionHolder, and is often called by the PositionHolderController.
getPositionHolderModel(). Method in class MovingSoundSourceViewFrame
getPositionHolderModel(). Method in class PeriodicSoundSourceViewFrame
getPositionHolderModel(). Method in class WallViewFrame
getPowerLevel(). Method in class EchoPoint
getPowerLevel(). Method in interface ISoundSource
THIS ACCESSOR RETURNS THE SoundSource's RUNTIME POWER LEVEL.
getPowerLevel(). Method in class PeriodicSoundSourceModel
getPowerLevelTextFieldValue(). Method in class PeriodicSoundSourceController
getResolution(). Method in interface IWall
THIS ACCESSOR RETURNS THE DISTANCE BETWEEN THE WALL'S EchoPoints.
getResolution(). Method in class WallModel
getResolutionTextFieldValue(). Method in class WallController
getResponse(). Method in class EarModel
getResponse(). Method in class EchoPoint
getResponse(). Method in interface ISoundListener
THIS ACCESSOR GIVES THE SoundListener's RUNTIME RESPONSE IN SOUND INTENSITY.
getResponseBufferView(). Method in class EarViewFrame
getResponseBufferView(). Method in interface IEarViewAdjustments
THIS ACCESSOR RETURNS THE EarViewFrame's RESPONSE BUFFER VIEW(S), SO THAT THE EarController MAY ACT DIRECTLY ON THEIR REFERENCES AT RUNTIME.
getResponseByIndex(int). Method in interface IWall
THIS ACCESSOR RETURNS THE Wall's RUNTIME RESPONSE VALUES, BY THE INDEX NUMBER OF EACH EchoPoint THIS IS CALLED BY THE Wall's View FOR GRAPHICAL DISPLAY.
getResponseByIndex(int). Method in class WallModel
getResponsesCheckbox(). Method in class WallViewCardSelector
getRightEarModel(). Method in interface ISimulatorModel
THIS FACTORY METHOD CREATES AND STORES AN EarModel INSTANCE, AND THEN SERVES IT'S IEarModel REFERENCE TO THE CALLING SoundscapeClient.
getRightEarModel(). Method in class SimulatorModel
getSawToothWaveCheckbox(). Method in class WaveSelectionGroup
getSettingsCheckbox(). Method in class WallViewCardSelector
getSignalStrength(). Method in class Signal
getSimulatorModel(). Method in interface ISimulatorView
This accessor returns a reference to the SimulatorView object's associated SimulatorModel, and is often called by the SimulatorController.
getSimulatorModel(). Method in class SimulatorView
getSineWaveCheckbox(). Method in class WaveSelectionGroup
getSoundAbsorption(). Method in class EchoPoint
getSoundAbsorption(). Method in interface IEchoPoint
This accessor returns a double value for the current sound absorption setting of an EchoPoint object.
getSoundAbsorption(). Method in interface IWall
THIS ACCESSOR GIVES THE Wall's RUNTIME (USER-SPECIFIED) SOUND ABSORPTION.
getSoundAbsorption(). Method in class WallModel
getSoundAbsorptionTextFieldValue(). Method in class WallController
getSoundIntensityBuffer(). Method in class EarModel
getSoundIntensityBuffer(). Method in interface IEar
THIS ACCESSOR RETURNS A BUFFER OF THE Ear's LATEST SOUND INTENSITY

RESPONSES.
getSoundIntensityCheckbox(). Method in class BufferViewCardSelector
getSouthCheckbox(). Method in class DirectionSelectionGroup
getSquareWaveCheckbox(). Method in class WaveSelectionGroup
getTextLog(). Method in class EarModel
getTextLog(). Method in class EarViewFrame
getTextLog(). Method in interface ITextLogUser
This accessor shares a reference to an ITextLogUser instance's ITextLogger implementation.
getTextLog(). Method in class MovingSoundSourceViewFrame
getTextLog(). Method in class PeriodicSoundSourceModel
getTextLog(). Method in class PeriodicSoundSourceViewFrame
getTextLog(). Method in class SignalQueueDirector
getTextLog(). Method in class SimulatorController
getTextLog(). Method in class SimulatorModel
getTextLog(). Method in class SimulatorView
getTextLog(). Method in class WallController
getTextLog(). Method in class WallModel
getTextLog(). Method in class WallViewFrame
getTextLog(). Method in class WallViewPanel
getTimeEffect(). Method in class Signal
getTimeZeroCheckbox(). Method in class StartTimeSelector
getUpCheckbox(). Method in class DirectionSelectionGroup
getUpperLeftHandCorner(). Method in interface IWall
THIS ACCESSOR RETURNS A REFERENCE TO THE Wall INSTANCE'S UPPER LEFT-HAND CORNER.
getUpperLeftHandCorner(). Method in class WallModel
getUpperRightHandCorner(). Method in interface IWall
THIS ACCESSOR RETURNS A REFERENCE TO THE Wall INSTANCE'S UPPER RIGHT-HAND CORNER.
getUpperRightHandCorner(). Method in class WallModel
getWallController(). Method in class WallViewPanel
getWallHeight(). Method in interface IWall
THIS ACCESSOR GIVES THE Wall's RUNTIME HEIGHT IN METERS.
getWallHeight(). Method in class WallModel
getWallHeightTextFieldValue(). Method in class WallController
getWallModel(). Method in interface IWallView
This accessor returns a reference to the WallView object's associated WallModel, and is often called by the WallController.
getWallModel(). Method in class WallViewFrame
getWallViewPanel(). Method in class WallViewFrame
getWallWidth(). Method in interface IWall
THIS ACCESSOR GIVES THE Wall's RUNTIME WIDTH IN METERS.
getWallWidth(). Method in class WallModel
getWallWidthTextFieldValue(). Method in class WallController
getWaveShape(). Method in interface IPeriodicSoundSource
THIS ACCESSOR GIVES THE PeriodicSoundSource's RUNTIME WAVESHAPE, IN TERMS OF THE CONSTANTS DEFINED ABOVE.
getWaveShape(). Method in class PeriodicSoundSourceModel
getWestCheckbox(). Method in class DirectionSelectionGroup
getWindowAdapter(). Method in class POCPopUp
getWindowAdapter(). Method in class SimulatorView
getXCoord(). Method in class EarModel
getXCoord(). Method in class EchoPoint
getXCoord(). Method in interface IPositionHolder
THIS ACCESSOR RETURNS THE PositionHolder's RUNTIME X COORDINATE VALUE, IN METERS.
getXCoord(). Method in class PeriodicSoundSourceModel

getXCoord(). Method in class ThreeDimensionalPoint
getXCoord(). Method in class WallModel
getXTextFieldValue(). Method in class PositionHolderController
getYCoord(). Method in class EarModel
getYCoord(). Method in class EchoPoint
getYCoord(). Method in interface IPositionHolder
    THIS ACCESSOR RETURNS THE PositionHolder's RUNTIME Y COORDINATE VALUE, IN METERS.
getYCoord(). Method in class PeriodicSoundSourceModel
getYCoord(). Method in class ThreeDimensionalPoint
getYCoord(). Method in class WallModel
getYTextFieldValue(). Method in class PositionHolderController
getZCoord(). Method in class EarModel
getZCoord(). Method in class EchoPoint
getZCoord(). Method in interface IPositionHolder
    THIS ACCESSOR RETURNS THE PositionHolder's RUNTIME Z COORDINATE VALUE, IN METERS.
getZCoord(). Method in class PeriodicSoundSourceModel
getZCoord(). Method in class ThreeDimensionalPoint
getZCoord(). Method in class WallModel
getZTextFieldValue(). Method in class PositionHolderController
gridbag. Variable in class EarViewFrame
gridbag. Variable in class MovingSoundSourceViewFrame
gridbag. Variable in class PeriodicSoundSourceViewFrame
gridbag. Variable in class PositionHolderController
gridbag. Variable in class SimulatorController
gridbag. Variable in class SimulatorView
gridbag. Variable in class WallViewFrame
gridbagconstraints. Variable in class EarViewFrame
gridbagconstraints. Variable in class MovingSoundSourceViewFrame
gridbagconstraints. Variable in class PeriodicSoundSourceViewFrame
gridbagconstraints. Variable in class PositionHolderController
gridbagconstraints. Variable in class SimulatorController
gridbagconstraints. Variable in class SimulatorView
gridbagconstraints. Variable in class WallViewFrame

I

iActual_Index. Variable in class CircularDoubleArray
iArrayHeight. Variable in class WallResponsesViewCanvas
iArrayWidth. Variable in class WallResponsesViewCanvas
iBase. Variable in class CircularDoubleArray
iBufferSize. Variable in class BufferViewCanvas
iDisplayHeight. Variable in class ResponseBufferPanel
iDisplayHeight. Variable in class WallResponsesViewCanvas
iDisplayHeight. Variable in class WallViewPanel
iDisplayWidth. Variable in class ResponseBufferPanel
iDisplayWidth. Variable in class WallResponsesViewCanvas
iDisplayWidth. Variable in class WallViewPanel
iFieldLabelIndent. Static variable in class ViewCanvas
iFieldValueIndent. Static variable in class ViewCanvas
iFirstTextRowYPos. Static variable in class ViewCanvas
iNew_Value_Index. Variable in class CircularDoubleArray
init(). Method in class SoundscapeApplet

initializeWallFrame(). Method in class WallController
initMenuBar(). Method in class SimulatorView
initSignalStrength(). Method in class Signal
initTimeOfEffect(double). Method in class Signal
iNumber_Additions. Variable in class CircularDoubleArray
iNumEchoPointsHigh. Variable in class WallModel
iNumEchoPointsWide. Variable in class WallModel
isBuffering(). Method in class EarModel
isBuffering(). Method in interface IEar
    THIS ACCESSOR RETURNS true IF THE Ear IS CURRENTLY BUFFERING IT'S SOUND INTENSITY AND DECIBEL RESPONSES.
isCloaked(). Method in class EarModel
isCloaked(). Method in interface ICloakable
    USED TO CHECK IF A Cloakable OBJECT IS CLOAKED AT RUNTIME.
isCloaked(). Method in class PeriodicSoundSourceModel
isCloaked(). Method in class WallModel
isEmpty(). Method in class SignalQueue
isFull(). Method in class CircularDoubleArray
isFull(). Method in interface ICircularDoubleArray
    This accessor returns `true` if the array is currently full.
iSize. Variable in class CircularDoubleArray
iSize. Variable in class SignalQueue
isMuted(). Method in class EchoPoint
isMuted(). Method in interface IMutable
    This accessor returns `true` if the `ISoundSource` is currently muted.
isMuted(). Method in class PeriodicSoundSourceModel
isMuted(). Method in class WallModel
isMutedToLeftEar(). Method in class EchoPoint
isMutedToLeftEar(). Method in interface IMutable
    This accessor returns `true` if the `ISoundSource` is currently muted to the simulation's LeftEar.
isMutedToLeftEar(). Method in class PeriodicSoundSourceModel
isMutedToLeftEar(). Method in class WallModel
isMutedToRightEar(). Method in class EchoPoint
isMutedToRightEar(). Method in interface IMutable
    This accessor returns `true` if the `ISoundSource` is currently muted to the simulation's RightEar.
isMutedToRightEar(). Method in class PeriodicSoundSourceModel
isMutedToRightEar(). Method in class WallModel
itemStateChanged(ItemEvent). Method in class EarController
itemStateChanged(ItemEvent). Method in class PeriodicSoundSourceController
itemStateChanged(ItemEvent). Method in class SimulatorController
itemStateChanged(ItemEvent). Method in class WallController
itemStateChanged(ItemEvent). Method in class WallViewFrame
iTextRowHeight. Static variable in class ViewCanvas
iTextRowYPos. Variable in class ViewCanvas
iWaveShape. Variable in class PeriodicSoundSourceModel
iZoomFactor. Variable in class WallResponsesViewCanvas

L

lblFrequency. Variable in class PeriodicSoundSourceController
lblOffsetPhase. Variable in class PeriodicSoundSourceController
lblPowerLevel. Variable in class PeriodicSoundSourceController
lblXCoord. Variable in class PositionHolderController
lblYCoord. Variable in class PositionHolderController

THIS MODIFIER ACCEPTS strength AS SOUND PRESSURE.
reinitialize(). Method in interface IWall
THIS MODIFIER MAKES THE Wall RECALCULATE ALL OF IT'S ATTRIBUTES WHICH ARE BASED ON THE (USER-SPECIFIED) UPPER-LEFTHAND CORNER AND FACING DIRECTION.
reinitialize(). Method in class WallModel
reInitialize(ISoundListener, ISoundSource, double). Method in class Signal
reinitialize(Signal). Method in class SignalQueueElement
ResponseBuffer. Variable in class BufferViewCanvas
ResponseBufferPanel(int, int, IEarModel, IController). Constructor for class ResponseBufferPanel
RestoreColor. Variable in class ViewCanvas
RightEar. Variable in class SignalQueueMgr
RightEarModel. Variable in class SignalQueueDirector
RightEarModel. Variable in class SimulatorModel
RightEarMutingCheckbox. Variable in class PeriodicSoundSourceController
RightEarMutingCheckbox. Variable in class WallController
RightEarViewFrame. Variable in class SimulatorView
rummageForSignalQueueElement(). Method in class SignalQueue
runSignals(double). Method in class SignalQueueDirector
runSimulation(). Method in interface ISimulatorModel
THIS MODIFIER INVOKES THE INITIALIZED SIMULATION, IN FREE-RUN FASHION.
runSimulation(). Method in class SimulatorController
runSimulation(). Method in class SimulatorModel

S

SAWTOOTH. Static variable in interface IPeriodicSoundSource
setAmplitude(double). Method in class EchoPoint
setAmplitude(double). Method in interface ISoundSource
THIS MODIFIER CHANGES THE SoundSource's RUNTIME AMPLITUDE.
setAmplitude(double). Method in class PeriodicSoundSourceModel
setAzimuth(double). Method in class Direction
setAzimuth(double). Method in interface IDirection
This modifier sets the current azimuth (right/left heading) of the Direction, in degrees.
setBuffering(boolean). Method in class EarModel
setBuffering(boolean). Method in interface IEar
THIS MODIFIER SIGNALS THE Ear WHETHER TO BEGIN (OR CONTINUE) BUFFERING IT'S SOUND INTENSITY AND DECIBEL RESPONSES.
setDeltaXRate(double). Method in interface IMovable
THIS MODIFIER SETS THE IMovable's RUNTIME X COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
setDeltaXRate(double). Method in class MovingSoundSourceModel
setDeltaYRate(double). Method in interface IMovable
THIS MODIFIER SETS THE IMovable's RUNTIME Y COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
setDeltaYRate(double). Method in class MovingSoundSourceModel
setDeltaZRate(double). Method in interface IMovable
THIS MODIFIER SETS THE IMovable's RUNTIME Z COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
setDeltaZRate(double). Method in class MovingSoundSourceModel
setElevation(double). Method in class Direction
setElevation(double). Method in interface IDirection
This modifier sets the current elevation (up/down vantage) of the Direction, in degrees.
setFacingDirection(Direction). Method in interface IWall

setTextLog(ITextLogger). Method in interface ITextLogUser
    This modifier accepts an ITextLogger instance and assigns an ITextLogUser instance's aggregate ITextLogger reference to the given object.
setTextLog(ITextLogger). Method in class MovingSoundSourceViewFrame
setTextLog(ITextLogger). Method in class PeriodicSoundSourceModel
setTextLog(ITextLogger). Method in class PeriodicSoundSourceViewFrame
setTextLog(ITextLogger). Method in class SignalQueueDirector
setTextLog(ITextLogger). Method in class SimulatorController
setTextLog(ITextLogger). Method in class SimulatorModel
setTextLog(ITextLogger). Method in class SimulatorView
setTextLog(ITextLogger). Method in class WallController
setTextLog(ITextLogger). Method in class WallModel
setTextLog(ITextLogger). Method in class WallViewFrame
setTextLog(ITextLogger). Method in class WallViewPanel
setTimeConstraints(). Method in class SimulatorController
setTimeConstraints(double, double, double). Method in interface ISimulatorModel
    THIS MODIFIER SETS THE START AND FINISH TIMES (IN SECONDS) OF THE SIMULATION, FOR FREE-RUN MODE.
setTimeConstraints(double, double, double). Method in class SimulatorModel
setupSAFStreams(BufferedOutputStream, BufferedOutputStream). Method in class SimulatorController
setWallHeight(double). Method in interface IWall
    THIS MODIFIER CHANGES THE RUNTIME HEIGHT (in meters) OF THE Wall.
setWallHeight(double). Method in class WallModel
setWallWidth(double). Method in interface IWall
    THIS MODIFIER CHANGES THE RUNTIME WIDTH (in meters) OF THE Wall.
setWallWidth(double). Method in class WallModel
setWaveShape(int). Method in interface IPeriodicSoundSource
    THIS MODIFIER SETS THE PeriodicSoundSource's RUNTIME WAVESHAPE, IN TERMS OF THE CONSTANTS DEFINED ABOVE.
setWaveShape(int). Method in class PeriodicSoundSourceModel
showDecibelResponse(). Method in class EarViewFrame
showDecibelResponse(). Method in interface IEarViewAdjustments
    THIS MODIFIER SELECTS THE DECIBEL RESPONSE BUFFER VIEW CARD AS VISIBLE TO THE USER.
showDecibelResponse(). Method in class ResponseBufferPanel
showResponsesCard(). Method in class WallViewPanel
showSettingsCard(). Method in class WallViewPanel
showSoundIntensity(). Method in class EarViewFrame
showSoundIntensity(). Method in interface IEarViewAdjustments
    THIS MODIFIER SELECTS THE SOUND INTENSITY BUFFER VIEW CARD AS VISIBLE TO THE USER.
showSoundIntensity(). Method in class ResponseBufferPanel
shrinkVertical(). Method in class BufferViewCanvas
shrinkVertical(). Method in class EarViewFrame
shrinkVertical(). Method in interface IEarViewAdjustments
    THIS MODIFIER HALVES THE VERTICAL STRETCHING FACTOR OF THE EarViewFrame's RESPONSE BUFFER VIEW, AND REFRESHES THE VIEW(S).
shrinkVertical(). Method in class ResponseBufferPanel
Signal(ISoundListener, ISoundSource, double). Constructor for class Signal
signalIsOK(Signal). Method in class SignalQueueMgr
SignalQueue(). Constructor for class SignalQueue
SignalQueueDirector(IEarModel, IEarModel). Constructor for class SignalQueueDirector
SignalQueueDirector(IEarModel, IEarModel, double). Constructor for class SignalQueueDirector
SignalQueueElement(Signal). Constructor for class SignalQueueElement
SignalQueueMgr(IEar, IEar, Vector). Constructor for class SignalQueueMgr
sigTempSignal. Variable in class SignalQueueMgr

[illegible]

```java
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
// Class:          BufferViewCanvas
// Author:         Matt Gentner
// Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The class BufferViewCanvas is a visible component of the EarViewFrame.
 * Derivatives of BufferViewCanvas are the DecibelResponseViewCanvas and
 * the SoundIntensityViewCanvas.  Each type of BufferViewCanvas plots a
 * graphical representation of the latest EarModel response values.
 *
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.awt.Dimension;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;

import ViewCanvas;
import CircularDoubleArray;


public class BufferViewCanvas
    extends ViewCanvas
{
    protected double dZoomFactor;
    protected int iBufferSize;
    protected IEarModel myEarModel = null;
    protected CircularDoubleArray ResponseBuffer = null;

    public BufferViewCanvas(  int iDisplayWidth_param,
                              int iDisplayHeight_param,
                              IEarModel EarModel_param )
    {
        super( iDisplayWidth_param, iDisplayHeight_param, EarModel_param );

        dZoomFactor = +1.0e0d;
        iBufferSize = 600;

        myEarModel = EarModel_param;
    }

    protected void initializeGUI()
    {
        setVisible( true );
    }

    protected void setResponseBuffer( CircularDoubleArray ResponseBuffer_param )
    {
        ResponseBuffer = ResponseBuffer_param;
    }

    public void update()
    {
        Graphics g = getGraphics();
        cleanCanvas( g );
        drawBorder( g );
        plotResponseBuffer( g );
    }

    public void paint( Graphics g )
    {
        cleanCanvas( g );
        drawBorder( g );

        if( bufferIsFull() )
            plotResponseBuffer( g );
    }

    public boolean bufferIsFull()
    {
        return( ResponseBuffer.isFull() );
    }

    public void addElement( double dResponse_param )
    {
        ResponseBuffer.addElement( dResponse_param );
    }

    public void stretchVertical()
    {
        dZoomFactor *= +2.0e0d;
        repaint();
    }

    public void shrinkVertical()
    {
        dZoomFactor /= +2.0e0d;
        repaint();
    }

    /*----------------------------------------------------------------
        READER BEWARE..
        WARNING:  THIS FUNCTION LOOKS NASTY!  IT PLOTS THE
        BUFFERED RESPONSES, DRAWING FROM RIGHT TO LEFT.  ALSO,
        SINCE THE LAST VALUES OF THE BUFFER ARE THE MOST RECENT,
        IT TRAVERSES FROM THE END OF THE ResponseBuffer TOWARD
        THE BEGINING.  YUCK.
    -------------------------------------------------------------------*/
    protected void plotResponseBuffer( Graphics g )
    {
        // System.out.println( "Plotting buffer of " + ResponseBuffer.size() + " elements: \n" );

        // USE getSize() METHOD OF THIS GRAPHICS Component
        Dimension dimCanvas = getSize();
        int frame_width = dimCanvas.width;

        // SET UP DRAWING LIMITS AND PARAMETERS
        int iResponseBufferSize = ResponseBuffer.size();
        int vertical_middle = ( dimCanvas.height / 2 );
        int horizontal_margin = 5;
        int right_horiz_limit = dimCanvas.width - horizontal_margin;
        int left_horiz_limit = ( right_horiz_limit - iResponseBufferSize );

        if( left_horiz_limit < horizontal_margin )
            left_horiz_limit = horizontal_margin;

        // DRAW A GREEN LINE ACROSS THE MIDDLE
        g.setColor( Color.green );
        g.drawLine( horizontal_margin, vertical_middle, right_horiz_limit, vertical_middle );
        g.setColor( Color.black );

        // SET UP THE FIRST PLOTTING POINT'S COORDINATES
        int x1, y1, x2, y2;

        x1 = right_horiz_limit;
        y1 = ( vertical_middle - (int)( dZoomFactor *
                ResponseBuffer.elementAt( iResponseBufferSize ) ) );

        // ITERATE THROUGH THE ARRAY, AND CONNECT EACH POINT
        for( int index = ( right_horiz_limit - 1 );
            index > left_horiz_limit;
            index-- )
        {
            x2 = index;

            y2 = vertical_middle - (int)( dZoomFactor *
                ResponseBuffer.elementAt( iResponseBufferSize -
                                    ( right_horiz_limit - index ) ) );

            g.drawLine( x1, y1, x2, y2 );

            x1 = x2;
            y1 = y2;
        }
    }

    ///////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "BufferViewCanvas" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////
```

```java
//-----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          BufferViewCardSelector
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The class BufferViewCardSelector is a CheckboxGroup with
 * selections for either 'Decibel' or 'SoundIntensity' for
 * user choice of viewing decibel or sound intensity values
 * in the EarViewFrame's BufferViewCanvas.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.event.ItemListener;

public class BufferViewCardSelector extends CheckboxGroup
{
    protected Checkbox cbSoundIntensity = null;
    protected Checkbox cbDecibelResponse = null;
    protected ItemListener myItemListener = null;

    BufferViewCardSelector( ItemListener ItemListener_param )
    {
        myItemListener = ItemListener_param;

        initializeCheckBoxes();

        setSelectedCheckbox( cbSoundIntensity );
    }

    protected void initializeCheckBoxes()
    {
        cbSoundIntensity = new Checkbox( "Sound Intensity", this, true );
        cbSoundIntensity.addItemListener( myItemListener );

        cbDecibelResponse = new Checkbox( "Decibel Response", this, false );
        cbDecibelResponse.addItemListener( myItemListener );
    }

    public Checkbox getSoundIntensityCheckbox()
    {
        return cbSoundIntensity;
    }

    public Checkbox getDecibelResponseCheckbox()
    {
        return cbDecibelResponse;
    }

}
```

```
//-------------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          CircularDoubleArray
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-------------------------------------------------------------------------
/**
 * THE CircularDoubleArray IS A CLASS WHICH IS DESIGNED TO BE USED AS
 * AN OVER-WRITABLE BUFFER OF IEEE double VALUES.  THESE VALUES MAY
 * BE REVIEWED IN GRAPHICAL DISPLAY THROUGH THE EarView AS THE
 * EarModel's RECENT SOUND INTENSITY AND DECIBEL RESPONSES.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-------------------------------------------------------------------------

import ICircularDoubleArray;


public class CircularDoubleArray
    implements ICircularDoubleArray
{
    protected boolean bArrayIsFilled;
    protected int iActual_Index;
    protected int iBase;
    protected int iNew_Value_Index;
    protected int iNumber_Additions;
    protected int iSize;
    protected double[] dArray;

    CircularDoubleArray( int size )  {
        iBase = 0;
        iNew_Value_Index = 0;
        iNumber_Additions = 0;
        bArrayIsFilled = false;
        iSize = size;
        dArray = new double[ iSize ];
        // System.out.println( "Created a CircularDoubleArray with length:  " + dArray.length );
    }

    public double elementAt( int IN_iUser_Queried_Index )
    {
        iActual_Index = ( iBase + ( IN_iUser_Queried_Index - 1 ) );

        if( iActual_Index < iSize )
            return dArray[ iActual_Index ];

        else
            return dArray[ ( iActual_Index - iSize ) ];
            // elements of the CircularArray are numbered
    }       // from ONE to SIZE, inclusive.  (not ZERO to SIZE-1)

    public void addElement( double IN_dNew_Value )
    {
        if( bArrayIsFilled )  {
            iBase++;

            if( iBase == iSize )  {
                iBase = 0;
                iNew_Value_Index = ( iSize - 1 );
            }

            else
                iNew_Value_Index = iBase - 1;

            dArray[ iNew_Value_Index ] = IN_dNew_Value;
            iNumber_Additions++;
        }

        else
        {
            dArray[ iNumber_Additions++ ] = IN_dNew_Value;

            if( iNumber_Additions == iSize )
                bArrayIsFilled = true;
        }
    }

    public boolean isFull()
    {
        return bArrayIsFilled;
    }

    public int size()
    {
        return iSize;
    }
}  // end of CircularDoubleArray class
```

```
//----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          DecibelResponseViewCanvas
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class DecibelResponseViewCanvas extends BufferViewCanvas
 * and is a visual component of the EarViewFrame which graphically
 * plots the recent decibel response values of the EarModel.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import BufferViewCanvas;


public class DecibelResponseViewCanvas
    extends BufferViewCanvas
{

    public DecibelResponseViewCanvas(  int iDisplayWidth_param,
                                       int iDisplayHeight_param,
                                       IEarModel EarModel_param )
    {
        super( iDisplayWidth_param, iDisplayHeight_param, EarModel_param );
        super.setResponseBuffer( EarModel_param.getDecibelResponseBuffer() );
    }

    public void update()
    {
        ResponseBuffer = myEarModel.getDecibelResponseBuffer();
        super.update();
    }
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          Direction
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * Direction IS A CLASS WHICH IS DESIGNED TO BE AN ATTRIBUTE OF
 * Soundscape SIMULATION OBJECTS THAT HAVE SURFACES WHICH ARE
 * ORIENTED IN A PARTICULAR DIRECTION OR DIRECTIONS.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.lang.Math;
import java.lang.String;

import IDirection;


public class Direction implements IDirection
{
    public final static Direction NORTH    = new Direction( 0.0e0d, 0.0e0d, "NORTH" );

    public final static Direction SOUTH    = new Direction( +1.80e+2d, 0.0e0d, "SOUTH" );

    public final static Direction EAST    = new Direction( -9.0e+1d, 0.0e0d, "EAST" );

    public final static Direction WEST    = new Direction( +9.0e+1d, 0.0e0d, "WEST" );

    public final static Direction UP    = new Direction( 0.0e0d, +9.0e+1d, "UP" );

    public final static Direction DOWN    = new Direction( 0.0e0d, +9.0e+1d, "DOWN" );

    protected double dAzimuth = 0.0e0d;
    protected double dAzimuth_InRadians = 0.0e0d;
    protected double dElevation = 0.0e0d;
    protected double dElevation_InRadians = 0.0e0d;
    protected String strMyName = null;

    public Direction( double dAzimuth_param,
                      double dElevation_param )
    {
        this( dAzimuth_param, dElevation_param, "Un-named" );
    }

    public Direction( double dAzimuth_param,
                      double dElevation_param,
                      String IN_strDirectionName )
    {
        dAzimuth = dAzimuth_param;
        dElevation = dElevation_param;
        strMyName = IN_strDirectionName;

        dAzimuth_InRadians = ( Math.PI * dAzimuth / +1.80e+2d );
        dElevation_InRadians = ( Math.PI * dElevation / +1.80e+2d );
    }

    public double getAzimuth()
    {
        return dAzimuth;
    }

    public double getElevation()
    {
        return dElevation;
    }

    public double getAzimuth_InRadians()
    {
        return dAzimuth_InRadians;
    }

    public double getElevation_InRadians()
    {
        return dElevation_InRadians;
    }

    public void setAzimuth( double dAzimuth_param )
    {
        dAzimuth = dAzimuth_param;
        dAzimuth_InRadians = ( Math.PI * dAzimuth / +1.80e+2d );
    }

    public void setElevation( double dElevation_param )
    {
        dElevation = dElevation_param;
        dElevation_InRadians = ( Math.PI * dElevation / +1.80e+2d );
    }

    public String toString()
    {
        return strMyName;
    }

}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          DirectionSelectionGroup
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class DirectionSelectionGroup is a CheckboxGroup
 * with six orthogonal selections to control the facing
 * direction of a WallModel.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.event.ItemListener;

public class DirectionSelectionGroup extends CheckboxGroup
{
    protected Checkbox cbNorth = null;
    protected Checkbox cbSouth = null;
    protected Checkbox cbEast = null;
    protected Checkbox cbWest = null;
    protected Checkbox cbUp = null;
    protected Checkbox cbDown = null;
    protected ItemListener myItemListener = null;

    DirectionSelectionGroup( ItemListener ItemListener_param )
    {
        myItemListener = ItemListener_param;

        initializeCheckBoxes();

        setSelectedCheckbox( cbNorth );
    }

    protected void initializeCheckBoxes()
    {
        cbNorth = new Checkbox( "North", this, true );
        cbNorth.addItemListener( myItemListener );

        cbSouth = new Checkbox( "South", this, false );
        cbSouth.addItemListener( myItemListener );

        cbEast = new Checkbox( "East", this, false );
        cbEast.addItemListener( myItemListener );

        cbWest = new Checkbox( "West", this, false );
        cbWest.addItemListener( myItemListener );

        cbUp = new Checkbox( "Up", this, false );
        cbUp.addItemListener( myItemListener );

        cbDown = new Checkbox( "Down", this, false );
        cbDown.addItemListener( myItemListener );
    }

    public Checkbox getNorthCheckbox()
    {
        return cbNorth;
    }

    public Checkbox getSouthCheckbox()
    {
        return cbSouth;
    }

    public Checkbox getEastCheckbox()
    {
        return cbEast;
    }

    public Checkbox getWestCheckbox()
    {
        return cbWest;
    }

    public Checkbox getUpCheckbox()
    {
        return cbUp;
    }

    public Checkbox getDownCheckbox()
    {
        return cbDown;
    }

}
```

```
//------------------------------------------------------------------------
//
//        Department of Computer Science, SUNY Institute of Technology
//         Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
// Class:          EarController
// Author:         Matt Gentner
// Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The class EarController accepts and handles all
 * user-interface events which come from controls
 * on the EarViewFrame.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//------------------------------------------------------------------------

import java.awt.Button;
import java.awt.Checkbox;
import java.awt.GridBagConstraints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.lang.Throwable;

import IController;
import IEarView;
import PositionHolderController;
import IEarModel;
import BufferViewCardSelector;


public class EarController
    extends PositionHolderController
    implements ActionListener, ItemListener, IController
{
    protected IEarModel myEarModel = null;
    protected IEarView myEarView = null;

    protected Button btnStretchVertical = null;
    protected Button btnShrinkVertical = null;

    protected Checkbox CloakingCheckbox = null;
    protected Checkbox BufferingCheckbox = null;

    protected BufferViewCardSelector BufferViewSelector = null;

    public EarController( int width, int height,
                           IEarModel EarModel_param,
                           IEarView EarView_param )
    {
        super( width, height, EarModel_param );

        initializeEarController( EarModel_param, EarView_param );

        initializeEarControllerAggregates();
        initializeEarControllerFrame();

        setSize( width, height );
        setVisible( true );
    }

    ///////////////////////////////////////////////////////////////////////////
    // PROTECTED INTIALIZATION METHODS:

    protected void initializeEarController(  IEarModel EarModel_param,
                                    IEarView EarView_param )
    {
        myEarModel = EarModel_param;
        myEarView = EarView_param;
    }

    protected void initializeEarControllerAggregates()
    {
        btnStretchVertical = new Button( "Stretch Vertical" );
        btnStretchVertical.addActionListener( this );
        btnShrinkVertical = new Button( "Shrink Vertical" );
        btnShrinkVertical.addActionListener( this );

        CloakingCheckbox = new Checkbox( "Cloak" );
        CloakingCheckbox.addItemListener( this );
        CloakingCheckbox.setState( myEarModel.isCloaked() );

        BufferingCheckbox = new Checkbox( "Buffer Signals (for display)" );
        BufferingCheckbox.addItemListener( this );
        BufferingCheckbox.setState( myEarModel.isBuffering() );

        BufferViewSelector = new BufferViewCardSelector( this );
    }


    protected void initializeEarControllerFrame()
    {
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( CloakingCheckbox );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnStretchVertical );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( BufferingCheckbox );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnShrinkVertical );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( BufferViewSelector.getSoundIntensityCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( BufferViewSelector.getDecibelResponseCheckbox() );
    }
    // END OF PROTECTED INTIALIZATION METHODS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IController INTERFACE IMPLEMENTATIONS:

    public void update()
    {
        super.update();
        CloakingCheckbox.setState( myEarModel.isCloaked() );
        BufferingCheckbox.setState( myEarModel.isBuffering() );

    }
    // END OF IController INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ActionListener INTERFACE IMPLEMENTATIONS:
    public void actionPerformed( ActionEvent ae )
    {
        super.actionPerformed( ae );

        if( ae.getSource() == btnStretchVertical )
            myEarView.stretchVertical();

        else if( ae.getSource() == btnShrinkVertical )
            myEarView.shrinkVertical();

        myEarModel.updateAllViews();
    }
    // END OF ActionListener INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ItemListener INTERFACE IMPLEMENTATIONS:
    public void itemStateChanged( ItemEvent ie )
    {
        if( ie.getSource() == CloakingCheckbox )
        {
            if( CloakingCheckbox.getState() )
            {
                myEarModel.cloak();
            }
            else
            {
                myEarModel.uncloak();
            }
        }
        else if( ie.getSource() == BufferingCheckbox )
        {
            myEarModel.setBuffering( BufferingCheckbox.getState() );

            if( BufferingCheckbox.getState() )
                myEarView.engageResponseBuffer();
            else
                myEarView.disengageResponseBuffer();
        }
        else if( ie.getSource() == BufferViewSelector.getDecibelResponseCheckbox() )
        {
            if( ( BufferViewSelector.getDecibelResponseCheckbox() ).getState() )
            {
                myEarView.showDecibelResponse();
            }
        }
        else if( ie.getSource() == BufferViewSelector.getSoundIntensityCheckbox() )
        {
            if( ( BufferViewSelector.getSoundIntensityCheckbox() ).getState() )
            {
                myEarView.showSoundIntensity();
            }
        }
    }
    // END OF ItemListener INTERFACE IMPLEMENTATIONS
    //////////////////////////////////////////////////////////////////////////

}
```

```java
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:           EarModel
//  Author:          Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * EarModel IS A CLASS WHICH IS DESIGNED TO BE USED AS THE MAIN
 * SUBJECTIVE COMPONENT OF THE Soundscape SIMULATOR.  OBJECTS OF
 * THE EarModel CLASS ARE TYPICALLY THE LEFT AND RIGHT EarModel
 * INSTANCES, WHICH PRODUCE THE SUBJECTIVE RESPONSES DURING
 * SIMULATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.util.Vector;
import java.lang.Double;

import ICloakable;
import IEarModel;
import ITextLogger;
import ITextLogUser;


public class EarModel
    implements IEarModel, ICloakable, ITextLogUser
{
    // INSTANCE VARIABLES:
    protected boolean bCloaked = false;
    protected boolean bBuffering = true;
    protected double x_coord = 0.0e0d;   // COORDINATES MEASURED IN METERS
    protected double y_coord = 0.0e0d;
    protected double z_coord = 0.0e0d;
    protected double dDecibelResponse = 0.0e0d;
    protected double dSoundIntensity = 0.0e0d;

    protected CircularDoubleArray DecibelResponseBuffer = null;
    protected CircularDoubleArray SoundIntensityBuffer = null;
    protected ITextLogger myTextLog = null;
    protected Vector vecObservers = null;

    // DEFAULT CONSTRUCTOR:
    public EarModel()
    {
        this( 0.0e0d, 0.0e0d, 0.0e0d );
    }

    public EarModel( double x_coord_param,
                double y_coord_param,
                double z_coord_param )
    {
        this(    x_coord_param,
                y_coord_param,
                z_coord_param,
                0.0e0d );
    }
    // PARAMETERIZED CONSTRUCTOR:
    public EarModel( double x_coord_param,
                double y_coord_param,
                double z_coord_param,
                double response_param  )
    {
        x_coord = x_coord_param;
        y_coord = y_coord_param;
        z_coord = z_coord_param;

        dDecibelResponse = response_param;

        vecObservers = new Vector( 3, 3 );
        DecibelResponseBuffer = new CircularDoubleArray( 600 );
        SoundIntensityBuffer  = new CircularDoubleArray( 600 );

    }
```

```
///////////////////////////////////////////////////////////////////////////
// IModel INTERFACE IMPLEMENTATIONS:
public void attach( IObserver SubscribingObserver_param )
{
    vecObservers.addElement( SubscribingObserver_param );
    // setTextLog( ( (ITextLogUser)SubscribingObserver_param ).getTextLog() );
    // myTextLog.logText(  "In EarModel, attached observer. \n" );
}

public void detach( IObserver UnSubscribingObserver_param )
{
    for( int i = 0; i < vecObservers.size(); i++ )
    {
        if( (IObserver)vecObservers.elementAt( i ) ==
            UnSubscribingObserver_param )
            vecObservers.removeElementAt( i );
    }
}

public void updateAllViews()
{
    for( int i = 0; i < vecObservers.size(); i++ )
        ( (IObserver)vecObservers.elementAt( i ) ).update();
}

public String getClassDefName()
{
    return( "EarModel" );
}
// END OF IModel INTERFACE IMPLEMENTATIONS
///////////////////////////////////////////////////////////////////////////


///////////////////////////////////////////////////////////////////////////
// IPositionHolderModel INTERFACE IMPLEMENTATIONS:

public double getXCoord()
{
    return x_coord;
}

public double getYCoord()
{
    return y_coord;
}

public double getZCoord()
{
    return z_coord;
}

public void moveTo( double x_coord_param,
                    double y_coord_param,
                    double z_coord_param )
{
    x_coord = x_coord_param;
    y_coord = y_coord_param;
    z_coord = z_coord_param;

    myTextLog.logText( "EarModel moved to x = " + x_coord +
                                         " y = " + y_coord +
                                         " z = " + z_coord + ". \n" );
}

// END OF IPositionHolderModel INTERFACE IMPLEMENTATIONS
///////////////////////////////////////////////////////////////////////////


///////////////////////////////////////////////////////////////////////////
// IEarModel INTERFACE IMPLEMENTATIONS:

// THIS MODIFIER ZEROES THE RESPONSE VALUE OF
// A IEarModel OBJECT.  THIS SHOULD BE DONE BETWEEN SAMPLE
// PERIODS (OTHERWISE, THE SIGNAL(S) FROM THE LAST SAMPLE
// PERIOD WILL LINGER IN THEIR INFLUENCE TO THE NEXT SAMPLE
// PERIOD.
public void dampResponse()
{
    dDecibelResponse = 0.0e0d;
    dSoundIntensity = 0.0e0d;
}


// THIS MODIFIER ENCAPSULATES ALL ACTIVITIES WHICH
```

```
    // A IEarModel OBJECT MUST PERFORM AT THE END
    // OF EACH SIMUALTION INSTANT.
    public void finalizeInstant()
    {
        convertToDecibel();

        if( bBuffering )
        {
            DecibelResponseBuffer.addElement( dDecibelResponse );
            SoundIntensityBuffer.addElement( dSoundIntensity );
        }

        if( !bCloaked )
            updateAllViews();
    }

    // THIS MODIFIER ACCEPTS dSoundIntensity_param AS SOUND PRESSURE.
    // THE IEarModel SHOULD BASE IT'S RESPONSE IN SOME
    // WAY ON THE INSTANTANEOUS AND/OR AMBIANT SOUND PRESSURE.
    public void rattle( double dSoundIntensity_param )   // strength IS SOUND PRESSURE
    {
        dSoundIntensity += dSoundIntensity_param;
        // if( dSoundIntensity_param != 0.0e0d )
        //     System.out.println( "Ear model rattled with strength:  " + dSoundIntensity_param );
    }


    protected double LogBaseTen( double x )
    {
        // THE JDK'S Math.log IS ACTUALLY A NATRUAL LOGRITHMIC FUNCTION,
        // _NOT_ LOG BASE 10.
        return ( Math.log( x ) / Math.log( +1.0e+1d ) );
    }

    protected void convertToDecibel()
    {
        double dAbsSoundIntensity = Math.abs( dSoundIntensity ); // NEED ABSOLUTE VALUE

        // SOUND LEVEL = 10 * log (  SoundIntensity / ThresholdOfHearing )
        //                      10
        if( dAbsSoundIntensity > 0.0e0d )
        {
            dDecibelResponse = ( +1.0e+1d *
                                LogBaseTen( dAbsSoundIntensity / dThreshold_Of_Hearing ) );
        }
        //else
        //    dDecibelResponse = dThreshold_Of_Hearing;

        if( dDecibelResponse == Double.NEGATIVE_INFINITY )
        {
            myTextLog.logText( "dDecibelResponse == Double.NEGATIVE_INFINITY" );
            myTextLog.logText( "dAbsSoundIntensity = " + dAbsSoundIntensity );
            myTextLog.logText( "Math.log( dAbsSoundIntensity / dThreshold_Of_Hearing ) = " + Math.log( dAbsSoundIntensi
ty / dThreshold_Of_Hearing ));

            while( true )
                ;
        }
    }



    // THIS ACCESSOR GIVES THE IEarModel's RUNTIME RESPONSE
    public double getDecibelResponse()
    {
        return dDecibelResponse;
    }

    // THIS ACCESSOR GIVES THE IEarModel's RUNTIME SoundIntensity
    public double getResponse()
    {
        return dSoundIntensity;
    }

    // END OF IEarModel INTERFACE IMPLEMENTATIONS
    //////////////////////////////////////////////////////////////////////////

    //////////////////////////////////////////////////////////////////////////
    // MODIFIER METHODS:

    // THIS ACCESSOR RETURNS true IF THE Ear IS CURRENTLY
    // BUFFERING IT'S SOUND INTENSITY AND DECIBEL RESPONSES.
    public boolean isBuffering()
    {
        return bBuffering;
    }

    // THIS MODIFIER SIGNALS THE Ear WHETHER TO BEGIN (OR CONTINUE)
    // BUFFERING IT'S SOUND INTENSITY AND DECIBEL RESPONSES.
    public void setBuffering( boolean bBuffering_param )
    {
        bBuffering = bBuffering_param;

        if( bBuffering )
            myTextLog.logText( "Ear Model has engaged buffering. \n" );
        else
            myTextLog.logText( "Ear Model has disengaged buffering. \n" );
    }
    // END OF MODIFIER METHODS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // ACCESSOR METHODS:

    public boolean bufferIsFull()
    {
        return( DecibelResponseBuffer.isFull() );
    }

    public CircularDoubleArray getDecibelResponseBuffer()
    {
        return DecibelResponseBuffer;
    }

    public CircularDoubleArray getSoundIntensityBuffer()
    {
        return SoundIntensityBuffer;
    }
    // END OF ACCESSOR METHODS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // ICloakable INTERFACE IMPLEMENTATIONS:

    // USED TO CHECK IF A Cloakable
    // OBJECT IS CLOAKED AT RUNTIME.
    public boolean isCloaked()
    {
        return bCloaked;
    }

    // USED TO CANCEL GUI REDRAWS,
    // AND SPEED PERFORMANCE DURING
    // LONG STRETCHES OF INTENSE SIMULATION.
    public void cloak()
    {
        bCloaked = true;
        myTextLog.logText( "Ear Model has been cloaked. \n" );
    }

    // (RE-)ENABLES A Cloakable OBJECT'S
    // GUI FOR GRAPHICAL UPDATES.
    public void uncloak()
    {
        bCloaked = false;
        myTextLog.logText( "Ear Model has been un-cloaked. \n" );
    }
    // END OF ICloakable INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////

} // END OF Ear CLASS DEFINITION
```

```
//---------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          EarViewCanvas
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------------
/**
 * The class EarViewCanvas extends PositionHolderViewCanvas
 * and is a visual component of the EarViewFrame which
 * displays textual attribute values of the EarMdodel
 * at runtime.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------------

import java.awt.Color;
import java.awt.Graphics;
import java.lang.Throwable;

import PositionHolderViewCanvas;
import IEarModel;


public class EarViewCanvas
    extends PositionHolderViewCanvas
    // AS SUCH, extends ViewCanvas
    // AS SUCH, implements IView
{
    protected IEarModel myEarModel = null;
    protected double dResponseBuffer = 0.0e0d;


    public EarViewCanvas(    int iDisplayWidth_param,
                                int iDisplayHeight_param,
                                IEarModel EarModel_param )
    {
        super( iDisplayWidth_param, iDisplayHeight_param, EarModel_param );

        initializeEarModel( EarModel_param );

        // BUFFER LATEST VALUE OF RESPONSE
        updateResponseBuffer();
    }

    public void paint( Graphics g )
    {
        super.paint( g );
        drawEarText( g );
    }

    protected void drawEarText( Graphics g )
    {
        RestoreColor = g.getColor();

        // BUFFER LATEST VALUE OF RESPONSE
        updateResponseBuffer();

        g.drawString( "Response:  ", 10, 80 );
        g.drawString( String.valueOf( dResponseBuffer ), 100, 80 );

        g.setColor( RestoreColor );

    }

    protected void updateResponseBuffer()
    {
        // BUFFER LATEST VALUE OF RESPONSE
        dResponseBuffer = myEarModel.getResponse();
    }

    ///////////////////////////////////////////////////////////////////////////
    // View INTERFACE IMPLEMENTATIONS:                                      //

    protected void initializeEarModel( IEarModel EarModel_param )
    {
        myEarModel = EarModel_param;
    }

    public void update()
    {
        repaint();
        /*
        RestoreColor = CurrentGraphicsContext.getColor();

        // ERASE OLD VALUE OF RESPONSE
        CurrentGraphicsContext.setColor( getBackground() );
        CurrentGraphicsContext.drawString( String.valueOf( dResponseBuffer ), 100, 20 );

        // BUFFER LATEST VALUE OF RESPONSE
        updateResponseBuffer();

        // DRAW LATEST VALUE OF RESPONSE
        CurrentGraphicsContext.setColor( TextColor );
        CurrentGraphicsContext.drawString( String.valueOf( dResponseBuffer ), 100, 20 );

        CurrentGraphicsContext.setColor( RestoreColor );
        */
    }

    // END OF View INTERFACE IMPLEMENTATIONS                              //
    ////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "EarViewCanvas" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////


}  // end of EarViewCanvas class
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          EarViewFrame
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The class EarViewFrame is the main visual
 * user interface component for an EarModel.
 * It holds display components for EarModel
 * attributes and controls which allow the user
 * to configure the EarModel values.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//------------------------------------------------------------------------

import java.awt.Component;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.lang.String;
import java.lang.Throwable;

import EarController;
import EarViewCanvas;
import IEarModel;
import IEarView;
import IObserver;
import ITextLogger;
import ITextLogUser;
import ResponseBufferPanel;


public class EarViewFrame extends Frame
    implements IEarView
    // AS SUCH, implements IEarViewAdjustments,
    // IView, IObserver, ITextLogUser
{
    // INSTANCE VARIABLES:
    protected boolean bBuffering = true;
    protected IEarModel myEarModel = null;

    protected GridBagLayout gridbag = null;
    protected GridBagConstraints gridbagconstraints = null;

    protected ITextLogger myTextLog = null;
    protected EarController myEarController = null;
    protected EarViewCanvas myEarView = null;
    protected ResponseBufferPanel myResponseBufferView = null;

    // GUI CONSTRUCTOR:
    public EarViewFrame( IEarModel IEarModel_param, ITextLogger IN_TextLogger )
    {
        this( IEarModel_param, "EarViewFrame", IN_TextLogger );
    }

    public EarViewFrame( IEarModel IEarModel_param, String name_param, ITextLogger IN_TextLogger )
    {
        super( name_param );

        myEarModel = IEarModel_param;
        setTextLog( IN_TextLogger );
        myEarModel.attach( this );
        myEarModel.setTextLog( IN_TextLogger );

        initializeAggegates();
        initializeGUI();
    }

    ///////////////////////////////////////////////////////////////////////////
    // INITIALIZATION METHODS:
    protected void initializeAggegates()
    {
        myEarView = new EarViewCanvas( 400, 100, myEarModel );
        myResponseBufferView = new ResponseBufferPanel( 400, 200, myEarModel, myEarController );
        myResponseBufferView.showSoundIntensity();

        makeController();
    }

    protected void addComponent( Component c )
    {
        gridbag.setConstraints( c, gridbagconstraints );
        add( c );
    }

    public void initializeGUI()
    {
        setLocation( 10, 10 );
        setSize( 400, 500 );

        gridbag = new GridBagLayout();
        gridbagconstraints = new GridBagConstraints();
        setLayout( gridbag );

        gridbagconstraints.weightx = 1.0;
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        gridbagconstraints.fill = GridBagConstraints.BOTH;
        gridbagconstraints.gridheight = 12;
        gridbagconstraints.gridheight = 1;
        gridbagconstraints.weighty = 0.0;           //reset to the default

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( myEarController );
        addComponent( myEarView );
        addComponent( myResponseBufferView );

        setVisible( true );
    }
    // END OF INITIALIZATION METHODS
    ///////////////////////////////////////////////////////////////////////////


    ///////////////////////////////////////////////////////////////////////////
    // Frame CLASS METHOD OVER-RIDES:
    public void paint( Graphics g )
    {
        myEarController.repaint();

        myEarView.repaint();

        myResponseBufferView.repaint();
    }

    // END OF Frame CLASS METHOD OVER-RIDES
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IObserver INTERFACE IMPLEMENTATIONS:
    public void update()
    {
        myEarController.update();
        myEarView.update();
        myResponseBufferView.update();
    }

    // END OF IObserver INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IView INTERFACE IMPLEMENTATIONS:

    public void makeController()
    {
        myEarController = new EarController( 300, 170, myEarModel, this );
    }

    public void display()
    {
        repaint();
    }

    public void finalize() throws Throwable
    {
        myEarModel.detach( this );
        super.finalize();
    }
    // END OF IView INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
// IEarView INTERFACE IMPLEMENTATIONS:

public void engageResponseBuffer()
{
    bBuffering = true;
}

public void disengageResponseBuffer()
{
    bBuffering = false;
}

public EarViewCanvas getEarViewCanvas()
{
    return myEarView;
}

public ResponseBufferPanel getResponseBufferView()
{
    return myResponseBufferView;
}

public void stretchVertical()
{
    myResponseBufferView.stretchVertical();
}

public void shrinkVertical()
{
    myResponseBufferView.shrinkVertical();
}

public void showDecibelResponse()
{
    myResponseBufferView.showDecibelResponse();
}

public void showSoundIntensity()
{
    myResponseBufferView.showSoundIntensity();
}
// END OF IEarView INTERFACE IMPLEMENTATIONS
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// Model ACCESSOR METHODS:

public IEarModel getEarModel()
{
    return myEarModel;
}

public IPositionHolderModel getPositionHolderModel()
{
    return myEarModel;
}
// END OF ModelACCESSOR METHODS
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// ITextLogUser INTERFACE IMPLEMENTATIONS:

public ITextLogger getTextLog()
{
    return myTextLog;
}

public void setTextLog( ITextLogger IN_TextLog )
{
    myTextLog = ( IN_TextLog );
}
// END OF ITextLogUser INTERFACE IMPLEMENTATIONS
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// ISoundscapeBase INTERFACE IMPLEMENTATIONS:

public String getClassDefName()
{
    return( "EarViewFrame" );
}
// END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
////////////////////////////////////////////////////////////////////////////
```

```
}  // end of EarViewFrame class
```

```
//------------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:           EchoPoint
//  Author:          Matt Gentner
//  Original Date:   January, 1998
//------------------------------------------------------------------------
/**
 * THE EchoPoint IS A CLASS OF OBJECTS WHICH ARE MEANT TO SEEM EMBEDDED
 * EXACTLY AT THE SURFACE OF A WallModel, AT ZERO DEPTH.  EchoPoint
 * OBJECTS HAVE BOTH ISoundListener AND ISoundSource PROPERTIES IN THAT
 * THEIR RESPONSES TO AMBIANT SOUND PRESSURE MAY BE VIEWED;  AND THEIR
 * ATTENUATED RESPONSES ARE ECHOES OR SOURCES OF SOUND TO OTHER
 * ISoundListeners DURING SIMULATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//------------------------------------------------------------------------

import java.lang.Throwable;
import java.util.Vector;

import ICloakable;
import IEchoPoint;
import ISoundListener;


public class EchoPoint
    implements IEchoPoint
{
    // INSTANCE VARIABLES:
    protected boolean bMuted = false;
    protected boolean bMutedToLeftEar = false;
    protected boolean bMutedToRightEar = false;
    protected double x_coord = 0.0e0d;
    protected double y_coord = 0.0e0d;
    protected double z_coord = 0.0e0d;
    protected double dAmplitude = 0.0e0d;
    protected double dArea = 0.0e0d;
    protected double dResponse = 0.0e0d;
    protected double dSoundAbsorption = 0.0e0d;

    // PARAMETERIZED CONSTRUCTOR:
    public EchoPoint( double x_coord_param,
                double y_coord_param,
                double z_coord_param,
                double dArea_param,
                double dSoundAbsorption_param )
    {
        x_coord = x_coord_param;
        y_coord = y_coord_param;
        z_coord = z_coord_param;

        dArea = dArea_param;
        dSoundAbsorption = dSoundAbsorption_param;
    }

    public String getClassDefName()
    {
        return( "EchoPoint" );
    }

    ///////////////////////////////////////////////////////////////////////////
    // IMutable INTERFACE IMPLEMENTATIONS:

    public boolean isMuted()
    {
        return bMuted;
    }

    public void setMuted( boolean bMuted_param )
    {
        bMuted = bMuted_param;
    }
    public boolean isMutedToLeftEar()
    {
        return bMutedToLeftEar;
    }

    public void setMutedToLeftEar( boolean IN_bMutedToLeftEar )
    {
        bMutedToLeftEar = IN_bMutedToLeftEar;
    }

    public boolean isMutedToRightEar()
    {
        return bMutedToRightEar;
    }

    public void setMutedToRightEar( boolean IN_bMutedToRightEar )
    {
        bMutedToRightEar = IN_bMutedToRightEar;
    }
    // END OF IMutable INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IPositionHolder INTERFACE IMPLEMENTATIONS:

    public double getXCoord()
    {
        return x_coord;
    }

    public double getYCoord()
    {
        return y_coord;
    }

    public double getZCoord()
    {
        return z_coord;
    }

    public void moveTo( double x, double y, double z )
    {
        x_coord = x;
        y_coord = y;
        z_coord = z;
    }
    // END OF IPositionHolder INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    public void dampResponse()
    {
        dResponse = 0.0e0d;
    }

    public void rattle( double strength )
    {
        // dSoundAbsorption OF EchoPoint AFFECTS RESPONSE
        strength *= ( +1.0e0d - dSoundAbsorption );
        dResponse += strength;
        dAmplitude = dResponse;
    }


    // THIS ACCESSOR GIVES THE ISoundListener's RUNTIME RESPONSE
    public double getResponse()
    {
        return dResponse;
    }

    public void finalizeInstant()
    {
        if( bMuted )
            dAmplitude = 0.0e0d;
    }

    ///////////////////////////////////////////////////////////////////////////
    // ISoundSource INTERFACE IMPLEMENTATIONS:
    public double getAmplitude()
    {
        return dAmplitude;
    }

    public void setAmplitude( double dAmplitude_param )
    {
    }

    public double getPowerLevel()
    {
        // EchoPoints ARE NOT POWERED
        return 0.0e0d;
    }

    public void setPowerLevel( double dPowerLevel_param )
    {
        // EchoPoints ARE NOT POWERED
    }

    public void makeSound( double time )
    {
        // EchoPoints DO NOT MAKE SOUNDS OF THEIR OWN
    }
    // END OF ISoundSource INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IEchoPoint INTERFACE IMPLEMENTATIONS:
    public void displayCoords()
    {
        System.out.println( " x_coord = " + x_coord +
                            " y_coord = " + y_coord +
                            " z_coord = " + z_coord );
    }

    public double getSoundAbsorption()
    {
        return dSoundAbsorption;
    }

    public void setSoundAbsorption( double dSoundAbsorption_param )
    {
        dSoundAbsorption = dSoundAbsorption_param;
    }
    // END OF IEchoPoint INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

} // END OF EchoPoint CLASS DEFINITION
```

```
//---------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ICircularDoubleArray
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------
/**
 * The interface ICircularDoubleArray is
 * used to narrow and discipline the use
 * of the CircularDoubleArray class of
 * objects.  The CircularDoubleArray is
 * an array which, when full, adjusts
 * it's indexes so that for FIFO usage
 * the oldest value is overwritten to
 * accomodate a new value.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------


public interface ICircularDoubleArray
{
    /**
    * This accessor returns a double value
    * at the specified (apparent) index.
    *
    * @see     CircularDoubleArray#elementAt( int IN_iUser_Queried_Index )
    * @since   SoundscapeSimulation1.0
    */
    public double elementAt( int IN_iUser_Queried_Index );

    /**
    * This modifier inserts the given value into
    * the first empty slot, if the array is not full.
    * Otherwise, it inserts the new value in the
    * array's oldest slot and an offset is made to
    * correct further indexed queries.
    *
    * @see     CircularDoubleArray#addElement( double IN_dNew_Value )
    * @since   SoundscapeSimulation1.0
    */
    public void addElement( double IN_dNew_Value );

    /**
    * This accessor returns <code>true</code>
    * if the array is currently full.
    *
    * @see     CircularDoubleArray#isFull()
    * @since   SoundscapeSimulation1.0
    */
    public boolean isFull();

    /**
    * This accessor returns an integer number
    * of the non-empty slots in the array.
    *
    * @see     CircularDoubleArray#size()
    * @since   SoundscapeSimulation1.0
    */
    public int size();
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Class:          ICloakable
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>ICloakable</code> interface should be implemented by
 * simulation models.  The models can then determine for themselves
 * if they are cloaked, and skip updates to their IView implementations.
 * <p>
 * Examples:  EarModel and PeriodicSoundSourceModel
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarModel
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------

import ISoundscapeBase;

public interface ICloakable
         extends ISoundscapeBase
{
    /**
    * USED TO CHECK IF A Cloakable
    * OBJECT IS CLOAKED AT RUNTIME.
    *
    * @see     EarModel#isCloaked()
    * @since   SoundscapeSimulation1.0
    */
    public boolean isCloaked();

    /**
    * USED TO CANCEL GUI REDRAWS,
    * AND SPEED PERFORMANCE DURING
    * LONG STRETCHES OF INTENSE SIMULATION.
    *
    * @see     EarModel#cloak()
    * @since   SoundscapeSimulation1.0
    */
    public void cloak();

    /**
    * (RE-)ENABLES A Cloakable OBJECT'S
    * GUI FOR GRAPHICAL UPDATES.
    *
    * @see     EarModel#uncloak()
    * @since   SoundscapeSimulation1.0
    */
    public void uncloak();
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Class:          IController
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>IController</code> interface is meant to belong
 * to a Model-View-Controller software design pattern.  Therefore,
 * for each Controller class definition in the simulation, there is
 * usually one or more corresponding View classes.
 * <p>
 * IController implementations are the event handlers which
 * interpret user-interface actions, and act on Model
 * classes appropriately.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarController, PeriodicSoundSourceController, and WallController
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------


public interface IController
{
    /**
     * THIS MODIFIER CHANGES ANY GUI COMPONENTS
     * WHICH DISPLAY THE STATUS OF THE ASSIGNED Model.
     * OFTEN, Checkbox(es) MUST BE UPDATED TO REFLECT
     * THE Model's CURRENT STATE.
     *
     * @see     EarController#update()
     * @since   SoundscapeSimulation1.0
     */
    public void update();
}
```

```
//--------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Class:          IDirection
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//--------------------------------------------------------------------------
/**
 * IDirection IS AN INTERFACE FOR CLASSES OF
 * Soundscape SIMULATION OBJECTS WHICH ARE ORIENTED
 * BY BOTH AZIMUTH AND ELEVATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//--------------------------------------------------------------------------

import java.lang.String;


public interface IDirection
{
    /**
     * This accessor returns the current azimuth
     * (right/left heading) of the Direction, in
     * degrees.
     *
     * @see     Direction#getAzimuth()
     * @since   SoundscapeSimulation1.0
     */
    public double getAzimuth();

    /**
     * This accessor returns the current elevation
     * (up/down vantage) of the Direction, in
     * degrees.
     *
     * @see     Direction#getElevation()
     * @since   SoundscapeSimulation1.0
     */
    public double getElevation();

    /**
     * This accessor returns the current azimuth
     * (right/left heading) of the Direction, in
     * radians.
     *
     * @see     Direction#getAzimuth_InRadians()
     * @since   SoundscapeSimulation1.0
     */
    public double getAzimuth_InRadians();

    /**
     * This accessor returns the current elevation
     * (up/down vantage) of the Direction, in
     * radians.
     *
     * @see     Direction#getElevation_InRadians()
     * @since   SoundscapeSimulation1.0
     */
    public double getElevation_InRadians();

    /**
     * This modifier sets the current azimuth
     * (right/left heading) of the Direction, in
     * degrees.
     *
     * @see     Direction#setAzimuth( double IN_dDegreesAzimuth )
     * @since   SoundscapeSimulation1.0
     */
    public void setAzimuth( double IN_dDegreesAzimuth );

    /**
     * This modifier sets the current elevation
     * (up/down vantage) of the Direction, in
     * degrees.
     *
     * @see     Direction#setElevation( double IN_dDegreesElevation )
     * @since   SoundscapeSimulation1.0
     */
    public void setElevation( double IN_dDegreesElevation );

    /**
     * This accessor returns the instance name
     * of a Direction object.
     *
     * @see     Direction#toString()
     * @since   SoundscapeSimulation1.0
     */
    public String toString();
}
```

```
//-----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IEar
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The interface <code>IEar</code> was written specifically to define
 * a clean public interface for the EarModel class of objects.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarModel
 * @since   SoundscapeSimulator1.0
 */
//-----------------------------------------------------------------------

import CircularDoubleArray;
import ISoundListener;


public interface IEar
    extends ISoundListener
{
    /**
    * THIS ACCESSOR RETURNS true IF THE Ear IS CURRENTLY
    * BUFFERING IT'S SOUND INTENSITY AND DECIBEL RESPONSES.
    *
    * @see     EarModel#isBuffering()
    * @since   SoundscapeSimulation1.0
    */
    public boolean isBuffering();

    /**
    * THIS MODIFIER SIGNALS THE Ear WHETHER TO BEGIN (OR CONTINUE)
    * BUFFERING IT'S SOUND INTENSITY AND DECIBEL RESPONSES.
    *
    * @see     EarModel#setBuffering( boolean bBuffering_param )
    * @since   SoundscapeSimulation1.0
    */
    public void setBuffering( boolean IN_bBuffering );

    /**
    * THIS ACCESSOR GIVES THE Ear's RUNTIME
    * RESPONSE IN DECIBEL UNITS.  THIS IS CALLED
    * BY THE Ear's View.
    *
    * @see     EarModel#getDecibelResponse()
    * @since   SoundscapeSimulation1.0
    */
    public double getDecibelResponse();

    /**
    * THIS ACCESSOR RETURNS A BUFFER OF THE Ear's
    * LATEST DECIBEL RESPONSES.  THIS IS
    * CALLED BY THE Ear's View.
    *
    * @see     EarModel#getDecibelResponseBuffer()
    * @since   SoundscapeSimulation1.0
    */
    public CircularDoubleArray getDecibelResponseBuffer();

    /**
    * THIS ACCESSOR RETURNS A BUFFER OF THE Ear's
    * LATEST SOUND INTENSITY RESPONSES.  THIS IS
    * CALLED BY THE Ear's View.
    *
    * @see     EarModel#getSoundIntensityBuffer()
    * @since   SoundscapeSimulation1.0
    */
    public CircularDoubleArray getSoundIntensityBuffer();
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Class:          IEarModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>IEarModel</code> was written specifically for
 * the EarModel class, so that collaborating classes could use
 * a lightweight reference of the avaiable public methods of
 * an EarModel reference at runtime.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarModel
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------

import ICloakable;
import IPositionHolderModel;


public interface IEarModel
    extends IEar, ICloakable, IPositionHolderModel
    // AS SUCH, extends IModel, ISoundListener,
    // AND IPositionHolder
{
    /**
     * THIS MODIFIER MAKES THE Ear CALCULATE
     * IT'S RUNTIME DECIBEL RESPONSE, AND THEN
     * BUFFER BOTH IT'S DECIBEL RESPONSE AND
     * IT'S SOUND INTENSITY RESPONSE.
     *
     * @see     EarModel#finalizeInstant()
     * @since   SoundscapeSimulation1.0
     */
    public void finalizeInstant();

}
```

```
//------------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//            Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IEarView
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The interface <code>IEarView</code> interface adds a single
 * accessor to the IPositionHolderView interface, for getting
 * a reference to the IEarView instance's IEarModel.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarViewFrame
 * @since   SoundscapeSimulator1.0
 */
//------------------------------------------------------------------------

import IEarModel;
import IEarViewAdjustments;
import IPositionHolderView;


public interface IEarView
    extends IPositionHolderView, IEarViewAdjustments
{
    /**
    * This accessor returns a reference
    * to the EarView object's associated
    * EarModel, and is often called by
    * the EarController.
    *
    * @see     EarViewFrame#getEarModel()
    * @since   SoundscapeSimulation1.0
    */
    IEarModel getEarModel();
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IEarViewAdjustments
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>IEarViewAdjustments</code> interface is a set of methods
 * which are paricular to the EarView class implementation.  This
 * interface is a collection of methods which are required by the
 * EarController to act out user-invoked commands which arrive
 * as GUI events.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarViewFrame
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------

import ResponseBufferPanel;


public interface IEarViewAdjustments
{
    /**
    * THIS MODIFIER ENGAGES THE EarView's BUFFERED
    * ARRAYS OF RESPONSES, FOR USER DISPLAY
    *
    * @see     EarViewFrame#engageResponseBuffer()
    * @since   SoundscapeSimulation1.0
    */
    public void engageResponseBuffer();

    /**
    * THIS MODIFIER DISENGAGES THE EarView's BUFFERED
    * ARRAYS OF RESPONSES, FOR USER DISPLAY
    *
    * @see     EarViewFrame#disengageResponseBuffer()
    * @since   SoundscapeSimulation1.0
    */
    public void disengageResponseBuffer();

    /**
    * THIS ACCESSOR RETURNS THE EarView's RESPONSE BUFFER
    * VIEW(S), SO THAT THE EarController MAY ACT DIRECTLY
    * ON THEIR REFERENCES AT RUNTIME.
    *
    * @see     EarViewFrame#getResponseBufferView()
    * @since   SoundscapeSimulation1.0
    */
    public ResponseBufferPanel getResponseBufferView();

    /**
    * THIS MODIFIER DOUBLES THE VERTICAL STRETCHING FACTOR
    * OF THE EarView's RESPONSE BUFFER VIEW, AND REFRESHES
    * THE VIEW(S).
    *
    * @see     EarViewFrame#stretchVertical()
    * @since   SoundscapeSimulation1.0
    */
    public void stretchVertical();

    /**
    * THIS MODIFIER HALVES THE VERTICAL STRETCHING FACTOR
    * OF THE EarView's RESPONSE BUFFER VIEW, AND REFRESHES
    * THE VIEW(S).
    *
    * @see     EarViewFrame#shrinkVertical()
    * @since   SoundscapeSimulation1.0
    */
    public void shrinkVertical();

    /**
    * THIS MODIFIER SELECTS THE DECIBEL RESPONSE BUFFER
    * VIEW CARD AS VISIBLE TO THE USER.
    *
    * @see     EarViewFrame#showDecibelResponse()
    * @since   SoundscapeSimulation1.0
    */
    public void showDecibelResponse();
    /**
    * THIS MODIFIER SELECTS THE SOUND INTENSITY BUFFER
    * VIEW CARD AS VISIBLE TO THE USER.
    *
    * @see      EarViewFrame#showSoundIntensity()
    * @since    SoundscapeSimulation1.0
    */
    public void showSoundIntensity();
}
```

```
//---------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IEchoPoint
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------
/**
 * The interface IEchoPoint is
 * used to narrow and discipline the use
 * of the EchoPoint class of
 * objects.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------


public interface IEchoPoint
    extends ISoundListener, ISoundSource
{
    /**
    * This accessor returns a double value
    * for the current sound absorption setting
    * of an EchoPoint object.
    *
    * @see     EchoPoint#getSoundAbsorption()
    * @since   SoundscapeSimulation1.0
    */
    public double getSoundAbsorption();

    /**
    * This modifier sets the current sound absorption
    * for an EchoPoint object. the parameter IN_dSoundAbsorption
    * should be a value between 0.0 and +1.0 inclusive.
    * It is a portion of the sound an EchoPoint instance will
    * NOT reflect, for example a sound absorption value of
    * .25 means that an EchoPoint will reflect (echo) 75%
    * of the sound which strikes it.
    *
    * @see     EchoPoint#setSoundAbsorption( double IN_dSoundAbsorption )
    * @since   SoundscapeSimulation1.0
    */
    public void setSoundAbsorption( double IN_dSoundAbsorption );

    /**
    * This method notifies and EchoPoint that the
    * end of the current simulation instant has arrived.
    * Actions such as buffering the current response
    * of the EchoPoint must be made.
    *
    * @see     EchoPoint#finalizeInstant()
    * @since   SoundscapeSimulation1.0
    */
    public void finalizeInstant();
}
```

```
//---------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------
/**
 * The interface <code>IModel</code> interface is meant to belong
 * to a Model-View-Controller software design pattern.  Therefore,
 * for each Model class definition in the simulation, there are
 * usually corresponding View and Controller classes.
 * <p>
 * IModel implementations are the core abstractions of the
 * Model-View-Controller trinity.  They are initially set
 * by the user through the user interface IController.
 * During the simulation, other models and the simulation
 * time affect the IModel implementations.
 * <p>
 * Examples:  EarModel, PeriodicSoundSourceModel, WallModel
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     WallModel
 * @since   SoundscapeSimulator1.0
 */
//---------------------------------------------------------------------

import java.lang.Object;
import java.lang.String;

import IObserver;
import ISoundscapeBase;
import ITextLogUser;


public interface IModel
        extends ISoundscapeBase, ITextLogUser
{
    /**
    * THIS MODIFIER REGISTERS AN Observer (EITHER AN
    * USER-INTERFACE View OR A USER-INTERFACE Controller)
    * WITH A Model.  IT IS TYPICALLY CALLED ONCE BY
    * THE Model's PRIMARY View, SINCE THE View(s)
    * CREATE, AGGREGATE AND DESTROY THEIR Controller.
    * ONCE AN Observer ATTACHES TO IT'S Model, IT
    * SUBSCRIBES TO (AND CAN MAKE CHANGES TO) THAT
    * Model.
    *
    * @see     EarModel#attach( IObserver SubscribingObserver_param )
    * @since   SoundscapeSimulation1.0
    */
    public void attach( IObserver IN_SubscribingObserver );

    /**
    * THIS MODIFIER UN-REGISTERS AN Observer (EITHER AN
    * USER-INTERFACE View OR A USER-INTERFACE Controller)
    * FROM IT'S Model.  THIS IS TYPICALLY DONE JUST PRIOR
    * TO THE PRIMARY View's DESTRUCTION.
    *
    * @see     EarModel#detach( IObserver SubscribingObserver_param )
    * @since   SoundscapeSimulation1.0
    */
    public void detach( IObserver IN_UnSubscribingObserver );

    /**
    * THIS METHOD IS ANALOGOUS TO THE MODEL-VIEW-CONTROLLER
    * DESIGN PATTERN'S notify() METHOD.  YET, THE
    * Java Development Kit ALREADY HAS A notify() METHOD
    * DEEPLY INGRAINED:
    * Error:  Cannot override final method 'void Object.notify()'
    * THEREFORE, THIS METHOD IS RENAMED TO updateAllViews
    *
    * @see     EarModel#updateAllViews()
    * @since   SoundscapeSimulation1.0
    */
    public void updateAllViews();

}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IMovable
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>IMovable</code> interface should be implemented by any
 * SoundscapeSimulator class whose instances' position is changing and
 * time-dependent.
 * <p>
 * Examples:  MovingSoundSourceModel
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     MovingSoundSourceModel
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------


public interface IMovable
{
    /**
     * THIS ACCESSOR RETURNS THE IMovable's INITIAL
     * X COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE X-AXIS LIES IN A EASTERN DIRECTION.
     *
     * @see     MovingSoundSourceModel#getInitialXCoord()
     * @since   SoundscapeSimulation1.0
     */
    public double getInitialXCoord();

    /**
     * THIS ACCESSOR RETURNS THE IMovable's INITIAL
     * Y COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE Y-AXIS LIES IN A NORTHERN DIRECTION.
     *
     * @see     MovingSoundSourceModel#getInitialYCoord()
     * @since   SoundscapeSimulation1.0
     */
    public double getInitialYCoord();

    /**
     * THIS ACCESSOR RETURNS THE IMovable's INITIAL
     * Z COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE Z-AXIS LIES IN A UPWARD DIRECTION.
     *
     * @see     MovingSoundSourceModel#getInitialZCoord()
     * @since   SoundscapeSimulation1.0
     */
    public double getInitialZCoord();

    /**
     * THIS MODIFIER SETS THE IMovable's INITIAL
     * X COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE X-AXIS LIES IN A EASTERN DIRECTION.
     *
     * @see     MovingSoundSourceModel#setInitialXCoord( double IN_dInitialXCoord )
     * @since   SoundscapeSimulation1.0
     */
    public void setInitialXCoord( double IN_dInitialXCoord );

    /**
     * THIS MODIFIER SETS THE IMovable's INITIAL
     * Y COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE Y-AXIS LIES IN A NORTHERN DIRECTION.
     *
     * @see     MovingSoundSourceModel#setInitialYCoord( double IN_dInitialYCoord )
     * @since   SoundscapeSimulation1.0
     */
    public void setInitialYCoord( double IN_dInitialYCoord );

    /**
     * THIS MODIFIER SETS THE IMovable's INITIAL
     * Z COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE Z-AXIS LIES IN A UPWARD DIRECTION.
     *
     * @see     MovingSoundSourceModel#setInitialZCoord( double IN_dInitialZCoord )
     * @since   SoundscapeSimulation1.0
     */
```

```
public void setInitialZCoord( double IN_dInitialZCoord );

/**
 * THIS ACCESSOR RETURNS THE IMovable's RUNTIME
 * X COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
 * IN THE SIMULATION, THE POSITIVE X-AXIS
 * LIES IN A EASTERN DIRECTION.  SO, A NEGATIVE
 * DeltaXRate VALUE MEANS THAT THE IMovable
 * OBJECT IS MOVING WEST.
 *
 * @see      MovingSoundSourceModel#getDeltaXRate()
 * @since    SoundscapeSimulation1.0
 */
public double getDeltaXRate();

/**
 * THIS ACCESSOR RETURNS THE IMovable's RUNTIME
 * Y COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
 * IN THE SIMULATION, THE POSITIVE Y-AXIS
 * LIES IN A NORTHERN DIRECTION.  SO, A NEGATIVE
 * DeltaYRate VALUE MEANS THAT THE IMovable
 * OBJECT IS MOVING SOUTH.
 *
 * @see      MovingSoundSourceModel#getDeltaYRate()
 * @since    SoundscapeSimulation1.0
 */
public double getDeltaYRate();

/**
 * THIS ACCESSOR RETURNS THE IMovable's RUNTIME
 * Z COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
 * IN THE SIMULATION, THE POSITIVE Z-AXIS
 * LIES IN A UPWARD DIRECTION.  SO, A NEGATIVE
 * DeltaZRate VALUE MEANS THAT THE IMovable
 * OBJECT IS MOVING DOWNWARD.
 *
 * @see      MovingSoundSourceModel#getDeltaZRate()
 * @since    SoundscapeSimulation1.0
 */
public double getDeltaZRate();

/**
 * THIS MODIFIER SETS THE IMovable's RUNTIME
 * X COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
 * IN THE SIMULATION, THE POSITIVE X-AXIS
 * LIES IN A EASTERN DIRECTION.  SO, A NEGATIVE
 * DeltaXRate VALUE MEANS THAT THE IMovable
 * OBJECT IS MOVING WEST.
 *
 * @see      MovingSoundSourceModel#setDeltaXRate( double IN_dDeltaXRate )
 * @since    SoundscapeSimulation1.0
 */
public void setDeltaXRate( double IN_dDeltaXRate );

/**
 * THIS MODIFIER SETS THE IMovable's RUNTIME
 * Y COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
 * IN THE SIMULATION, THE POSITIVE Y-AXIS
 * LIES IN A NORTHERN DIRECTION.  SO, A NEGATIVE
 * DeltaYRate VALUE MEANS THAT THE IMovable
 * OBJECT IS MOVING SOUTH.
 *
 * @see      MovingSoundSourceModel#setDeltaYRate( double IN_dDeltaYRate )
 * @since    SoundscapeSimulation1.0
 */
public void setDeltaYRate( double IN_dDeltaYRate );

/**
 * THIS MODIFIER SETS THE IMovable's RUNTIME
 * Z COORDINATE (SIGNED) SPEED, IN METERS PER SECOND.
 * IN THE SIMULATION, THE POSITIVE Z-AXIS
 * LIES IN A UPWARD DIRECTION.  SO, A NEGATIVE
 * DeltaZRate VALUE MEANS THAT THE IMovable
 * OBJECT IS MOVING DOWNWARD.
 *
 * @see      MovingSoundSourceModel#setDeltaZRate( double IN_dDeltaZRate )
 * @since    SoundscapeSimulation1.0
 */
public void setDeltaZRate( double IN_dDeltaZRate );

/**
 * THIS MODIFIER NOTIFIES THE IMovable OBJECT
 * TO POSITION ITSELF PROPERLY, FOR THE SPECIFIED
 * TIME (IN SECONDS.)
 *
```

```
     * @param   IN_dTime Should be the time, in seconds.
     *
     * @see     MovingSoundSourceModel#moveToTime( double IN_dTime )
     * @since   SoundscapeSimulation1.0
     */
    public void moveToTime( double IN_dTime );
}
```

```
//-------------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IMovingSoundSource
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-------------------------------------------------------------------------
/**
 * THIS TAGGING INTERFACE IS A UNION OF THE COMBINED
 * BEHAVIORS OF THE IMovable AND ISoundSource INTERFACES.
 * <p>
 * SO FAR, THIS INTERFACE HAS NO UNIQUE BEHAVIORS
 * OF IT'S OWN.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarModel
 * @since   SoundscapeSimulator1.0
 */
//-------------------------------------------------------------------------

import IMovable;
import IPeriodicSoundSource;


public interface IMovingSoundSource
    extends IPeriodicSoundSource, IMovable
{

}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IMovingSoundSourceModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>IMovingSoundSourceModel</code> was written
 * specifically for the MovingSoundSourceModel class, so that
 * collaborating classes could use a lightweight reference of the
 * avaiable public methods of an MovingSoundSourceModel reference
 * at runtime.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     MovingSoundSourceModel
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------

import IMovingSoundSource;
import IPeriodicSoundSourceModel;


public interface IMovingSoundSourceModel
    extends IPeriodicSoundSourceModel, IMovingSoundSource
    // AS SUCH, extends IPeriodicSoundSource, IModel, IMutable,
    // ISoundSource, IMovable AND IPositionHolder
{

}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IMutable
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>IMutable</code> interface should be implemented by any
 * SoundscapeSimulator class whose instances are <code>ISoundSource</code>
 * implementors, or otherwise producers of sound'during simulation
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     PointSoundSourceModel
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------

import ISoundscapeBase;


public interface IMutable
        extends ISoundscapeBase
{
    /**
     * This accessor returns <code>true</code> if the
     * <code>ISoundSource</code> is currently muted.
     *
     * @see     PointSoundSourceModel#isMuted()
     * @since   SoundscapeSimulation1.0
     */
    public boolean isMuted();

    /**
     * This modifer changes the <code>ISoundSource</code>'s
     * muted state.  This state includes muting to both of
     * the simulation's ears, and all other SoundListeners
     * in the simulation.
     *
     * @param   bMuted_param Should be  <code>true</code> to
     *            mute the object, or <code>false</code> to set
     *            the object as not muted.
     *
     * @see     PointSoundSourceModel#setMuted( boolean bMuted_param )
     * @since   SoundscapeSimulation1.0
     */
    public void setMuted( boolean IN_bMuted );

    /**
     * This accessor returns <code>true</code> if the
     * <code>ISoundSource</code> is currently muted
     * to the simulation's LeftEar.
     *
     * @see     PointSoundSourceModel#isMutedToLeftEar()
     * @since   SoundscapeSimulation1.0
     */
    public boolean isMutedToLeftEar();

    /**
     * This modifer changes the <code>ISoundSource</code>'s
     * muted state, with respect to the LeftEar only.  So,
     * muting a <code>ISoundSource</code> to the simulation's
     * LeftEar will not mute the <code>ISoundSource</code> to
     * other <code>ISoundListener</code>s such as EchoPoints,
     * nor the simulation's RightEar.
     *
     * @param   IN_bMutedToLeftEar Should be <code>true</code> to
     *            mute the object, or <code>false</code> to set
     *            the object as not muted.
     *
     * @see     PointSoundSourceModel#setMutedToLeftEar( boolean IN_bMutedToLeftEar )
     * @since   SoundscapeSimulation1.0
     */
    public void setMutedToLeftEar( boolean IN_bMutedToLeftEar );

    /**
     * This accessor returns <code>true</code> if the
     * <code>ISoundSource</code> is currently muted
     * to the simulation's RightEar.
     *
     * @see     PointSoundSourceModel#isMutedToRightEar()
     * @since   SoundscapeSimulation1.0
     */
    public boolean isMutedToRightEar();

    /**
     * This modifer changes the <code>ISoundSource</code>'s
     * muted state, with respect to the RightEar only.  So,
     * muting a <code>ISoundSource</code> to the simulation's
     * RightEar will not mute the <code>ISoundSource</code> to
     * other <code>ISoundListener</code>s such as EchoPoints,
     * nor the simulation's LeftEar.
     *
     * @param   IN_bMutedToRightEar Should be <code>true</code> to
     *            mute the object, or <code>false</code> to set
     *            the object as not muted.
     *
     * @see     PointSoundSourceModel#setMutedToRightEar( boolean IN_bMutedToRightEar )
     * @since   SoundscapeSimulation1.0
     */
    public void setMutedToRightEar( boolean IN_bMutedToRightEar );
}
```

```
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IObserver
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The interface <code>IObserver</code> interface is meant to belong
 * to a Model-View-Controller software design pattern.  Therefore,
 * for each IObserver class definition in the simulation, there MUST
 * be a corresponding View and Controller classes.
 * <p>
 * The IObserver interface allows IView and IController
 * class implementations to be treated polymorphically,
 * since both interfaces extend IObserver.  Specifically,
 * both the View and Controller class definitions have
 * their own custom update() implementaion.
 * <p>
 * Examples:  IView, IController, PeriodicSoundSourceController
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     PeriodicSoundSourceController
 * @since   SoundscapeSimulator1.0
 */
//-----------------------------------------------------------------------

import ISoundscapeBase;


public interface IObserver
         extends ISoundscapeBase
{
    /**
    * THIS MODIFIER NOTIFIES THE View OR Conroller
    * THAT THE Model's STATE HAS CHANGED, AND
    * APPROPRIATE REDISPLAY IS LIKELY.
    *
    * @see     PeriodicSoundSourceController#update()
    * @since   SoundscapeSimulation1.0
    */
    public void update();
}
```

```java
//--------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IPeriodicSoundSource
//· Author:         Matt Gentner
//  Original Date:  January, 1998
//--------------------------------------------------------------------
/**
 * The interface <code>IPeriodicSoundSource</code> interface should be implemented by
 * ISoundSource implementing class definitions which produce periodic wave shapes.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     PeriodicSoundSourceModel
 * @since   SoundscapeSimulator1.0
 */
//--------------------------------------------------------------------

public interface IPeriodicSoundSource
    extends ISoundSource
    // AS SUCH, extends IMutable AND IPositionHolder
{
    // WAVE SHAPE CONSTANTS
    public final static int SINE = 1;

    public final static int SQUARE = 2;

    public final static int SAWTOOTH = 3;

    /**
    * THIS ACCESSOR GIVES THE PeriodicSoundSource's RUNTIME
    * OFFSET PHASE, IN DEGREES.
    *
    * @see     PeriodicSoundSourceModel#getOffsetPhase()
    * @since   SoundscapeSimulation1.0
    */
    public double getOffsetPhase();

    /**
    * THIS MODIFIER SETS THE PeriodicSoundSource's RUNTIME
    * OFFSET PHASE, IN DEGREES.
    *
    * @see     PeriodicSoundSourceModel#setOffsetPhase( double IN_dOffsetPhase )
    * @since   SoundscapeSimulation1.0
    */
    public void setOffsetPhase( double IN_dOffsetPhase );

    /**
    * THIS ACCESSOR GIVES THE PeriodicSoundSource's RUNTIME
    * FREQUENCY, IN HERTZ.
    *
    * @see     PeriodicSoundSourceModel#getFrequency()
    * @since   SoundscapeSimulation1.0
    */
    public double getFrequency();

    /**
    * THIS MODIFIER SETS THE PeriodicSoundSource's RUNTIME
    * FREQUENCY, IN HERTZ.
    *
    * @see     PeriodicSoundSourceModel#setFrequency( double IN_dFrequency )
    * @since   SoundscapeSimulation1.0
    */
    public void setFrequency( double IN_dFrequency );

    /**
    * THIS ACCESSOR GIVES THE PeriodicSoundSource's RUNTIME
    * WAVESHAPE, IN TERMS OF THE CONSTANTS DEFINED ABOVE.
    *
    * @see     PeriodicSoundSourceModel#getWaveShape()
    * @since   SoundscapeSimulation1.0
    */
    public int getWaveShape();

    /**
    * THIS MODIFIER SETS THE PeriodicSoundSource's RUNTIME
    * WAVESHAPE, IN TERMS OF THE CONSTANTS DEFINED ABOVE.
    *
    * @see     PeriodicSoundSourceModel#setWaveShape( int IN_iWaveShape )
    * @since   SoundscapeSimulation1.0
    */
    public void setWaveShape( int IN_iWaveShape );
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IPeriodicSoundSourceModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The interface <code>IPeriodicSoundSourceModel</code> was written
 * specifically for the PeriodicSoundSourceModel class, so that
 * collaborating classes could use a lightweight reference of the
 * avaiable public methods of an PeriodicSoundSourceModel reference
 * at runtime.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     PeriodicSoundSourceModel
 * @since   SoundscapeSimulator1.0
 */
//------------------------------------------------------------------------

import ICloakable;
import IPeriodicSoundSource;
import IPositionHolderModel;


public interface IPeriodicSoundSourceModel
    extends IPositionHolderModel, IPeriodicSoundSource, ICloakable
    // AS SUCH, extends IModel, IMutable,
    // ISoundSource, AND IPositionHolder
{

}
```

```
//--------------------------------------------------------------------
//
//        Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IPeriodicSoundSourceView
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//--------------------------------------------------------------------
/**
 * The interface <code>IPeriodicSoundSourceView</code> interface adds a single
 * accessor to the IPositionHolderView interface, for getting
 * a reference to the IPeriodicSoundSourceView instance's IPeriodicSoundSourceModel.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     PeriodicSoundSourceViewFrame
 * @since   SoundscapeSimulator1.0
 */
//--------------------------------------------------------------------

import IPeriodicSoundSourceModel;
import IPositionHolderView;


public interface IPeriodicSoundSourceView
    extends IPositionHolderView
{
    /**
    * This accessor returns a reference
    * to the PeriodicSoundSourceView object's associated
    * PeriodicSoundSourceModel, and is often called by
    * the PeriodicSoundSourceController.
    *
    * @see     PeriodicSoundSourceViewFrame#getPeriodicSoundSourceModel()
    * @since   SoundscapeSimulation1.0
    */
    IPeriodicSoundSourceModel getPeriodicSoundSourceModel();
}
```

```
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IPositionHolder
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The interface <code>IPositionHolder</code> interface should be implemented by any
 * SoundscapeSimulator class whose instances are either <code>ISoundSource</code>
 * or <code>ISoundListener</code> implementors.  <code>IPositionHolder</code>
 * classes are simply SoundscapeSimulator objects which have three dimensional
 * cartesian coordinates.
 * <p>
 * Examples are:  the WallModel and the PeriodicSoundSource class definitions.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     PointSoundSourceModel
 * @since   SoundscapeSimulator1.0
 */
//-----------------------------------------------------------------------

import ISoundscapeBase;

public interface IPositionHolder
         extends ISoundscapeBase
{
    /**
     * THIS ACCESSOR RETURNS THE PositionHolder's RUNTIME
     * X COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE X-AXIS LIES IN A EASTERN DIRECTION.
     *
     * @see     PositionHolderModel#getXCoord()
     * @since   SoundscapeSimulation1.0
     */
    public double getXCoord();

    /**
     * THIS ACCESSOR RETURNS THE PositionHolder's RUNTIME
     * Y COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE Y-AXIS LIES IN A NORTHERN DIRECTION.
     *
     * @see     PositionHolderModel#getYCoord()
     * @since   SoundscapeSimulation1.0
     */
    public double getYCoord();

    /**
     * THIS ACCESSOR RETURNS THE PositionHolder's RUNTIME
     * Z COORDINATE VALUE, IN METERS.  IN THE SIMULATION,
     * THE POSITIVE Z-AXIS LIES IN A UPWARD DIRECTION.
     *
     * @see     PositionHolderModel#getZCoord()
     * @since   SoundscapeSimulation1.0
     */
    public double getZCoord();

    /**
     * THIS MODIFIER MOVES THE PositionHolder's RUNTIME
     * COORDINATE VARIABLES TO THE SPECIFIED CARTESIAN
     * PARAMETERS.  OFTEN, THERE ARE OTHER ACTIVITIES
     * ASSOCIATED WITH MOVEMENT, SUCH AS UPDATING
     * CLASS-SPECIFIC DATA STRUCTURES AND/OR RECALCULATING
     * ATTRIBUTES WHICH ARE DEPENDENT ON POSITION.
     *
     * @see     PositionHolderModel#moveTo( double x, double y, double z )
     * @since   SoundscapeSimulation1.0
     */
   public void moveTo( double x, double y, double z );
}
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IPositionHolderModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The interface <code>IPositionHolderModel</code> interface combines
 * the behaviors of the IModel and ICloakable interfaces.  This
 * becomes a set of methods which are common to most SoundscapeSimulator
 * modeled classes of objects.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     WallModel
 * @since   SoundscapeSimulator1.0
 */
//------------------------------------------------------------------------

import IModel;
import IPositionHolder;


public interface IPositionHolderModel
    extends IModel, IPositionHolder
{

}
```

```
//-----------------------------------------------------------------------
//
//        Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IPositionHolderView
//  Author:             Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The interface IPositionHolderView derives from IView
 * and adds a single method to access an object's
 * IPositionHolderModel instance at runtime.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import IPositionHolderModel;
import IView;


public interface IPositionHolderView
    extends IView
{
    /**
    * This accessor returns a reference
    * to the PositionHolderView object's associated
    * PositionHolder, and is often called by
    * the PositionHolderController.
    *
    * @see     PositionHolderViewFrame#getPositionHolderModel()
    * @since   SoundscapeSimulation1.0
    */
    IPositionHolderModel getPositionHolderModel();
}
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ISignalQueueMgr
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The interface ISignalQueueMgr is a common set of
 * methods which all of the SignalQueueMgr derivatives
 * implement, and allows the SignalQueueDirector to
 * manage it's SignalQueueMgr objects polymorphically
 * at runtime.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//------------------------------------------------------------------------

import java.lang.String;


public interface ISignalQueueMgr
{
    /**
    * This initializer notifies a SignalQueueMgr to
    * set up all necessary SignalQueues.  A string
    * of dots is returned, having one dot for each
    * SignalQueue initialized.
    *
    * @see     WallSignalQueueMgr#initializeSignalQueues()
    * @since   SoundscapeSimulation1.0
    */
    public String initializeSignalQueues();

    /**
    * This modifier sets the sample period of
    * the SignalQueueMgr, in seconds.
    *
    * @see     SignalQueueMgr#setSamplePeriod( double IN_dSamplePeriod )
    * @since   SoundscapeSimulation1.0
    */
    public void setSamplePeriod( double IN_dSamplePeriod );

    /**
    * This method notifies the SignalQueueMgr
    * to create and queue all necessary Signal
    * objects for the specified simulation instant.
    *
    * @see     SignalQueueMgr#makeSignals( double IN_dCurrentTime )
    * @since   SoundscapeSimulation1.0
    */
    public void makeSignals( double IN_dCurrentTime );

    /**
    * This method notifies the SignalQueueMgr
    * to identify and dequeue all ready Signal
    * objects for the specified simulation instant.
    *
    * @see     SignalQueueMgr#effectSignals( double IN_dCurrentTime )
    * @since   SoundscapeSimulation1.0
    */
    public void effectSignals( double IN_dCurrentTime );

    /**
    * This method notifies the SignalQueueMgr
    * to (in turn) notify associated ISoundListener
    * objects to end finalize the current simulation
    * instant.
    *
    * @see     SignalQueueMgr#finalizeInstant()
    * @since   SoundscapeSimulation1.0
    */
    public void finalizeInstant();
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ISimulatorModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>ISimulatorModel</code> interface is written
 * specifically as a clean public interface for the SoundscapeServer.
 * This interface is a contract of obligated services to the SoundscapeServer,
 * and a promise of available services to the SoundscapeClient.
 * <p>
 *
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     WallModel
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------

import java.io.BufferedOutputStream;

import IEarModel;
import IModel;
import IPeriodicSoundSourceModel;
import ITextLogger;
import IWallModel;


public interface ISimulatorModel
    extends IModel
{
    /**
    * THIS FACTORY METHOD CREATES AND STORES
    * AN EarModel INSTANCE, AND THEN SERVES
    * IT'S IEarModel REFERENCE TO THE CALLING
    * SoundscapeClient.
    *
    * @see     SoundscapeServer#getLeftEarModel()
    * @since   SoundscapeSimulation1.0
    */
    public IEarModel getLeftEarModel();

    /**
    * THIS FACTORY METHOD CREATES AND STORES
    * AN EarModel INSTANCE, AND THEN SERVES
    * IT'S IEarModel REFERENCE TO THE CALLING
    * SoundscapeClient.
    *
    * @see     SoundscapeServer#getRightEarModel()
    * @since   SoundscapeSimulation1.0
    */
    public IEarModel getRightEarModel();

    /**
    * THIS FACTORY METHOD CREATES AND STORES
    * A PeriodicSoundSourceModel INSTANCE, AND THEN SERVES
    * IT'S IPeriodicSoundSourceModel REFERENCE TO THE CALLING
    * SoundscapeClient.
    *
    * @see     SoundscapeServer#newPeriodicSoundSourceModel()
    * @since   SoundscapeSimulation1.0
    */
    public IPeriodicSoundSourceModel newPeriodicSoundSourceModel();

    /**
    * THIS FACTORY METHOD CREATES AND STORES
    * A MovingSoundSourceModel INSTANCE, AND THEN SERVES
    * IT'S IMovingSoundSourceModel REFERENCE TO THE CALLING
    * SoundscapeClient.
    *
    * @see     SoundscapeServer#newMovingSoundSourceModel()
    * @since   SoundscapeSimulation1.0
    */
    public IMovingSoundSourceModel newMovingSoundSourceModel();
    /**
    * THIS FACTORY METHOD CREATES AND STORES
    * A WallModel INSTANCE, AND THEN SERVES
    * IT'S IWallModel REFERENCE TO THE CALLING
    * SoundscapeClient.
    *
    * @see     SoundscapeServer#newWallModel()
    * @since   SoundscapeSimulation1.0
    */
    public IWallModel newWallModel();

    /**
    * THIS MODIFIER SETS THE START AND FINISH TIMES
    * (IN SECONDS) OF THE SIMULATION, FOR FREE-RUN MODE.
    * ALSO, THIS METHOD SETS THE SIMULATION'S
    * RESOLUTION (IN METERS.)
    *
    * @see     SoundscapeServer#setTimeConstraints( double dStartTime,
                                                    double dFinishTime,
                                                    double dResolution )
    * @since   SoundscapeSimulation1.0
    */
    public void setTimeConstraints( double IN_dStartTime,
                                    double IN_dFinishTime,
                                    double IN_dResolution );
    /**
    * THIS MODIFIER CLOAKS ALL ICloakable INSTANCES
    * IT HAS.
    *
    * @see     SoundscapeServer#cloakAllObjects()
    * @since   SoundscapeSimulation1.0
    */
    public void cloakAllObjects();

    /**
    * THIS MODIFIER INVOKES THE INITIALIZED SIMULATION,
    * IN FREE-RUN FASHION.
    *
    * @see     SoundscapeServer#runSimulation()
    * @since   SoundscapeSimulation1.0
    */
    public void runSimulation();

    /**
    * THIS MODIFIER UNCLOAKS ALL ICloakable INSTANCES
    * IT HAS.
    *
    * @see     SoundscapeServer#uncloakAllObjects()
    * @since   SoundscapeSimulation1.0
    */
    public void uncloakAllObjects();

    /**
    * THIS MODIFIER INVOKES THE INITIALIZED SIMULATION,
    * IN SINGLE STEP FASHION.
    *
    * @see     SoundscapeServer#singleStep()
    * @since   SoundscapeSimulation1.0
    */
    public void singleStep();

    /**
    * THIS MODIFIER INITIALIZES THE SoundscapeServer'S
    * TEXT LOG, TO BE SHARED WITH THE SoundscapeClient.
    *
    * @see     SoundscapeServer#setTextLog( ITextLogger IN_TextLog )
    * @since   SoundscapeSimulation1.0
    */
    public void setTextLog( ITextLogger IN_TextLog );

    /**
    * THIS MODIFIER OPENS AN OUTPUT FILE FOR
    * SAVING THE SEQUENTIAL EAR RESPONSE VALUES.
    *
    * @see     SoundscapeServer#openOutputFile( String IN_strFileName )
    * @since   SoundscapeSimulation1.0
    */
    public void openOutputFile( String IN_strFileName );

    /**
    * THIS MODIFIER CLOSES THE CURRENT OUTPUT FILE.
    *
    * @see     SoundscapeServer#closeOutputFile()
    * @since   SoundscapeSimulation1.0
    */
    public void closeOutputFile();
}
```

```java
//--------------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ISimulatorView
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//--------------------------------------------------------------------------
/**
 * The interface <code>ISimulatorView</code> has just a
 * pair of <code>public</code> methods which are often
 * called by the SimulatorController or client code used
 * to deploy the simulator.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//--------------------------------------------------------------------------

import ISimulatorModel;
import ITextLogger;
import IView;


public interface ISimulatorView
    extends IView, ITextLogger
{
    /**
    * This modifier method is called by a client
    * SimulatorView factory mechanism, to complete
    * the GUI initialization of a SimulatorView
    * object.  This lazy initialization is done
    * because the SimulatorView must be (1)created,
    * (2)attached to it's model and then (3) made
    * visible and allowed to display it's model's
    * attribute values.
    *
    * @see     SimulatorView#initialize()
    * @since   SoundscapeSimulation1.0
    */
    public void initialize();

    /**
    * This accessor returns a reference
    * to the SimulatorView object's associated
    * SimulatorModel, and is often called by
    * the SimulatorController.
    *
    * @see     SimulatorView#getSimulatorModel()
    * @since   SoundscapeSimulation1.0
    */
    public ISimulatorModel getSimulatorModel();
}
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ISoundListener
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The interface <code>ISoundListener</code> interface should be implemented by any
 * SoundscapeSimulator class whose instances are targets of the Signal class of
 * objects.
 * <p>
 * Examples are:  the EchoPoint and the EarModel class definitions.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarModel
 * @since   SoundscapeSimulator1.0
 */
//------------------------------------------------------------------------

import IPositionHolder;


public interface ISoundListener
    extends IPositionHolder
{
    final double dThreshold_Of_Hearing = ( +1.0e-13d ); // 0db
    final double dThreshold_Of_Negligence = ( +1.0e-38d ); // -25db
    final double dSpeed_Of_Sound = ( +3.44e+2d ); // MEASURED IN METERS PER SECOND

    /**
     * THIS MODIFIER ZEROES THE RESPONSE VALUE OF
     * A SoundListener OBJECT.  THIS SHOULD BE DONE BETWEEN SAMPLE
     * PERIODS (OTHERWISE, THE SIGNAL(S) FROM THE LAST SAMPLE
     * PERIOD WILL LINGER IN THEIR INFLUENCE TO THE NEXT SAMPLE
     * PERIOD.)
     *
     * @see     EarModel#dampResponse()
     * @since   SoundscapeSimulation1.0
     */
    public void dampResponse();

    /**
     * THIS MODIFIER ACCEPTS strength AS SOUND PRESSURE.
     * THE SoundListener SHOULD BASE IT'S RESPONSE IN SOME
     * WAY ON THE INSTANTANEOUS AND/OR AMBIANT SOUND PRESSURE.
     * THE ONLY CLASS WHICH CALLS THIS METHOD IS THE
     * Signal CLASS OF OBJECTS, WHICH 'RATTLE' THEIR
     * TARGETS AT THEIR CALUCULATED TIME OF EFFECT.
     *
     * @param   strength Should be in terms of the sound (watts / sqr. meter)
     *          intensity exerted on a ISoundListener implementing
     *          simulation model.
     *
     * @see     EarModel#rattle( double strength )
     * @since   SoundscapeSimulation1.0
     */
    public void rattle( double strength );

    /**
     * THIS ACCESSOR GIVES THE SoundListener's RUNTIME
     * RESPONSE IN SOUND INTENSITY.  THIS IS CALLED BY
     * BOTH THE SoundListener's View(s) AND IT'S
     * USER-INTERFACE Controller.
     *
     * @see     EarModel#getResponse()
     * @since   SoundscapeSimulation1.0
     */
    public double getResponse();
}
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//       Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ISoundscapeBase
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The interface ISoundscapeBase simply
 * offers an accessor to query the name
 * of a class an object belongs to at
 * runtime.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//------------------------------------------------------------------------

public interface ISoundscapeBase
{
    /**
    * THIS ACCESSOR RETURNS THE SIMULATION OBJECT'S
    * SPECIFIC CLASS NAME.  IT IS TYPICALLY USED IN
    * THE USER-INTERFACE ViewCanvases, TO DISPLAY
    * A SMALL HEADER IN THEIR TEXTUAL REPORT.
    *
    * @see     EarModel#getClassDefName()
    * @since   SoundscapeSimulation1.0
    */
    public String getClassDefName();
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ISoundSource
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface <code>ISoundSource</code> interface should be implemented by any
 * SoundscapeSimulator class whose instances produce sound during the simulation.
 * <p>
 * Examples of such classes are the PeriodicSoundSourceModel and the EchoPoint
 * class definitions.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     PointSoundSourceModel
 * @since   SoundscapeSimulator1.0
 */
//----------------------------------------------------------------------

import IMutable;
import IPositionHolder;


public interface ISoundSource
    extends IMutable, IPositionHolder
{
    /**
     * THIS ACCESSOR GIVES THE SoundSource's RUNTIME
     * AMPLITUDE.
     *
     * @see     PeriodicSoundSourceModel#getAmplitude()
     * @since   SoundscapeSimulation1.0
     */
    public double getAmplitude();

    /**
     * THIS MODIFIER CHANGES THE SoundSource's RUNTIME
     * AMPLITUDE.
     *
     * @param   dAmplitude_param Should be used to set
     *            set the initial amplitude of an ISoundSource
     *            implementing class.
     *
     * @see     PeriodicSoundSourceModel#setAmplitude( double IN_dAmplitude )
     * @since   SoundscapeSimulation1.0
     */
    public void setAmplitude( double IN_dAmplitude );

    /**
     * THIS ACCESSOR RETURNS THE SoundSource's RUNTIME
     * POWER LEVEL.
     *
     * @see     PeriodicSoundSourceModel#getPowerLevel()
     * @since   SoundscapeSimulation1.0
     */
    public double getPowerLevel();

    /**
     * THIS MODIFIER SETS THE SoundSource's RUNTIME
     * POWER LEVEL.
     *
     * @param   dPowerLevel_param Should Should be used to set
     *            set the power level (in watts) of an ISoundSource
     *            implementing class.
     *
     * @see     PeriodicSoundSourceModel#setPowerLevel( double IN_dPowerLevel )
     * @since   SoundscapeSimulation1.0
     */
    public void setPowerLevel( double IN_dPowerLevel );

    /**
     * THIS MODIFIER ACCEPTS time ( IN SECONDS ).
     * EACH PARTICULAR SoundSource MAY THEN BASE
     * IT'S amplitude IN SOME WAY ON THE GIVEN
     * TIME.
     *
     * @param   time Should be used (in seconds)to get the instantaneous
     *            sound intensity calculated for a particular time
     *            during the simulation.
     *
     * @see     PeriodicSoundSourceModel#makeSound( double IN_dTime )
     * @since   SoundscapeSimulation1.0
     */
    public void makeSound( double IN_dTime );
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ITextLogger
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The interface ITextLogger designates class
 * implementations which may accept and log
 * stings of text during simulation.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import ITextLogUser;
import IView;


public interface ITextLogger
    extends IView, ITextLogUser
{
    /**
     * This method submits a string of text
     * to an ITextLogger implementation, for
     * logging purposes.
     *
     * @see     SimulatorView#logText( String IN_strLogText )
     * @since   SoundscapeSimulation1.0
     */
    public void logText( String IN_strLogText );
}
```

```
//-----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: ITextLogUser
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The ITextLogUser interface has a pair of methods
 * which are commonly used by simulation objects to
 * enable integrated logging of their textual output.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import ITextLogger;


public interface ITextLogUser
{
    /**
    * This accessor shares a reference
    * to an ITextLogUser instance's
    * ITextLogger implementation.
    *
    * @see     EarModel#getTextLog()
    * @since   SoundscapeSimulation1.0
    */
    public ITextLogger getTextLog();

    /**
    * This modifier accepts an ITextLogger
    * instance and assigns an ITextLogUser
    * instance's aggregate ITextLogger reference
    * to the given object.
    *
    * @see     EarViewFrame#getEarModel()
    * @since   SoundscapeSimulation1.0
    */
    public void setTextLog( ITextLogger IN_TextLog );
}
```

```java
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IThreeDimensionalPoint
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The interface IThreeDimensionalPoint offers a pair of
 * utility methods coomunly used when dealing with the
 * the ThreeDimensionalPoint class of objects.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------


public interface IThreeDimensionalPoint
    extends IPositionHolder, ISoundscapeBase
{
    /**
    * This method displays the coordinates
    * of an IThreeDimensionalPoint instance
    * at the system's standard output.  This
    * helps sometimes in testing usage of
    * IThreeDimensionalPoint classes.
    *
    * @see     ThreeDimensionalPoint#display()
    * @since   SoundscapeSimulation1.0
    *
    */
    public void display();

    /**
    * This method returns the distance in
    * meters to the given IThreeDimensionalPoint
    * instance.
    *
    * @see     ThreeDimensionalPoint#distanceTo( IThreeDimensionalPoint IN_OtherThreeDimensionalPoint )
    * @since   SoundscapeSimulation1.0
    *
    */
    public double distanceTo( IThreeDimensionalPoint IN_OtherThreeDimensionalPoint );
}
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IView
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The interface <code>IView</code> interface is meant to belong
 * to a Model-View-Controller software design pattern.  Therefore,
 * for each View class definition in the simulation, there are
 * usually corresponding Model and Controller classes.
 * <p>
 * IView implementations are the aesthetic display-type
 * user interface componenets of the Model-View-Controller (MVC)
 * trinity.
 * <p>
 * Examples:  EarView, PeriodicSoundSourceView, WallView
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     EarView
 * @since   SoundscapeSimulator1.0
 */
//------------------------------------------------------------------------

import IObserver;


public interface IView
    extends IObserver
{
    /**
     * THIS MODIFIER ALLOWS THE CREATOR OF AN
     * IView INSTANCE TO POSTPONE INITIALIZTION
     * OF THE IView'S GUI.  SO, AN IView INSTANCE
     * MAY BE ATTACHED TO IT'S MODEL PRIOR TO
     * FULL INITIALIZATION OF IT'S USER INTERFACE.
     *
     * @see     EarView#initializeGUI()
     * @since   SoundscapeSimulation1.0
     *
     */
    public void initializeGUI();
}
```

```
//-----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IWall
//  Author:          Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The interface <code>IWall</code> interface should be implemented by
 * planar and rectangular collections of ISoundListener and/or ISoundSource
 * implementing class definitions.  This interface is not an extension to
 * the ICloakable, and so class implementations should not require frequent
 * updates during simulations.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     WallModel
 * @since   SoundscapeSimulator1.0
 */
//-----------------------------------------------------------------------

import java.util.Vector;

import Direction;
import IEchoPoint;
import IMutable;
import IThreeDimensionalPoint;


public interface IWall
    extends IMutable
{
    /**
    * THIS ACCESSOR RETURNS THE Wall's RUNTIME RESPONSE
    * VALUES, BY THE INDEX NUMBER OF EACH EchoPoint
    * THIS IS CALLED BY THE Wall's View FOR GRAPHICAL
    * DISPLAY.
    *
    * @see     WallModel#getResponseArray()
    * @since   SoundscapeSimulation1.0
    */
    public double getResponseByIndex( int IN_iEchoPointIndex );

    /**
    * THIS MODIFIER CHANGES THE FACING DIRECTION
    * OF THE Wall.  THIS IS CALLED BY THE Wall's
    * USER-INTERFACE Controller.
    *
    * @see     WallModel#setFacingDirection( Direction FacingDirection_param )
    * @since   SoundscapeSimulation1.0
    */
    public void setFacingDirection( Direction FacingDirection_param );

    /**
    * THIS ACCESSOR SIMPLY RETURNS A REFERENCE TO
    * THE CURRENT FACING Direction OF THE WallModel.
    * THIS IS USED BY THE WallController TO UPDATE
    * IT'S DIRECTION SELECTOR CONTROL.
    *
    * @see     WallModel#getFacingDirection()
    * @since   SoundscapeSimulation1.0
    */
    public Direction getFacingDirection();

    /**
    * THIS MODIFIER MAKES THE Wall RECALCULATE ALL OF
    * IT'S ATTRIBUTES WHICH ARE BASED ON THE
    * (USER-SPECIFIED) UPPER-LEFTHAND CORNER AND
    * FACING DIRECTION.
    *
    * @see     WallModel#reinitialize()
    * @since   SoundscapeSimulation1.0
    */
    public void reinitialize();

    /**
    * THIS ACCESSOR GIVES THE Wall's RUNTIME WIDTH
    * IN METERS.  THIS IS CALLED BY BOTH THE Wall's
    * View(s), AND IT'S USER-INTERFACE Controller.
    *
    * @see     WallModel#getWallWidth()
    * @since   SoundscapeSimulation1.0
    */
    public double getWallWidth();

    /**
    * THIS MODIFIER CHANGES THE RUNTIME WIDTH (in meters)
    * OF THE Wall.  THIS IS CALLED BY THE Wall's
    * USER-INTERFACE Controller.
    *
    * @see     WallModel#setWallWidth( double dWallWidth_param )
    * @since   SoundscapeSimulation1.0´
    */
    public void setWallWidth( double dWallWidth_param );

    /**
    * THIS ACCESSOR GIVES THE Wall's RUNTIME HEIGHT
    * IN METERS.  THIS IS CALLED BY BOTH THE Wall's
    * View(s), AND IT'S USER-INTERFACE Controller.
    *
    * @see     WallModel#getWallHeight()
    * @since   SoundscapeSimulation1.0
    */
    public double getWallHeight();

    /**
    * THIS MODIFIER CHANGES THE RUNTIME HEIGHT (in meters)
    * OF THE Wall.  THIS IS CALLED BY THE Wall's
    * USER-INTERFACE Controller.
    *
    * @see     WallModel#setWallHeight( double dWallHeight_param )
    * @since   SoundscapeSimulation1.0
    */
    public void setWallHeight( double dWallHeight_param );

    /**
    * THIS ACCESSOR GIVES THE Wall's RUNTIME
    * (USER-SPECIFIED) SOUND ABSORPTION.
    *
    * THIS ACCESSOR GIVES THE SoundSource's RUNTIME
    * AMPLITUDE.
    *
    * @see     WallModel#getSoundAbsorption()
    * @since   SoundscapeSimulation1.0
    */
    public double getSoundAbsorption();

    /**
    * THIS MODIFIER CHANGES THE  SOUND ABSORPTION
    * OF THE Wall.  VALUES SHOULD BE BETWEEN 0.0 AND
    * +1.0 INCLUSIVE, +1.0 MEANING THAT THW WALL ABSORBS
    * ALL SOUND IT MEETS, AND ECHOES NONE.  THIS IS
    * CALLED BY THE Wall's USER-INTERFACE Controller.
    *
    * @see     WallModel#setSoundAbsorption( double dSoundAbsorption_param )
    * @since   SoundscapeSimulation1.0
    */
    public void setSoundAbsorption( double dSoundAbsorption_param );

    /**
    * THIS ACCESSOR RETURNS THE Wall's RUNTIME EchoPoints
    * Vector, WHICH IS A CONVENIENT CONTAINER OF IT'S
    * EchoPoints' REFERENCES.  IN GENERAL, THIS IS CALLED
    * BY THE SimulatorCore TO ADD THE EchoPoints THEMSELVES
    * DIRECTLY INTO THE SIMULATION (RATHER THAN HAVING
    * THE Wall AS A MIDDLE MAN.)
    *
    * THIS ACCESSOR GIVES THE SoundSource's RUNTIME
    * AMPLITUDE.
    *
    * @see     WallModel#getEchoPoints()
    * @since   SoundscapeSimulation1.0
    */
    public IEchoPoint getEchoPointByIndex( int IN_iEchoPointIndex );

    /**
     * THIS METHOD CALCULATES THE AVERAGE RESPONSE
     * VALUES FOR ALL THE EchoPoints THE WALL
     * HAS.  SOMETIMES, THIS IS USEFUL FOR
     * TROUBLE-SHOOTING PURPOSES.  ALSO, HELPS
     * WHEN ONE IS TRYING TO ADJUST THE BRIGHTNESS
     * IN THE WALL-RESPONSES VIEW!
     *
     * @see     WallModel#displayAvgResponse()
     * @since   SoundscapeSimulation1.0
     */
    public void displayAvgResponse();
```

```
/**
 * THIS ACCESSOR RETURNS THE TOTAL NUMBER
 * EchoPoints THE WALL HAS.  SOMETIMES, THIS
 * IS USEFUL FOR TROUBLE-SHOOTING PURPOSES.
 *
 * @see     WallModel#getNumEchoPoints()
 * @since   SoundscapeSimulation1.0
 */
public int getNumEchoPoints();

/**
 * THIS ACCESSOR RETURNS THE TOTAL NUMBER
 * EchoPoints THE WALL HAS PER ROW.
 * SOMETIMES, THIS
 * IS USEFUL FOR TROUBLE-SHOOTING PURPOSES.
 *
 * @see     WallModel#getNumEchoPointsWide()
 * @since   SoundscapeSimulation1.0
 */
public int getNumEchoPointsWide();

/**
 * THIS ACCESSOR RETURNS THE TOTAL NUMBER
 * EchoPoints THE WALL HAS PER COLUMN.
 * SOMETIMES, THIS
 * IS USEFUL FOR TROUBLE-SHOOTING PURPOSES.
 *
 * @see     WallModel#getNumEchoPointsHigh()
 * @since   SoundscapeSimulation1.0
 */
public int getNumEchoPointsHigh();
/**
 * THIS ACCESSOR RETURNS A REFERENCE TO THE Wall
 * INSTANCE'S UPPER LEFT-HAND CORNER.  SOMETIMES,
 * THIS IS USEFUL FOR TROUBLE-SHOOTING PURPOSES.
 *
 * @see     WallModel#getUpperLeftHandCorner()
 * @since   SoundscapeSimulation1.0
 */
public IThreeDimensionalPoint getUpperLeftHandCorner();

/**
 * THIS ACCESSOR RETURNS A REFERENCE TO THE Wall
 * INSTANCE'S UPPER RIGHT-HAND CORNER.  SOMETIMES,
 * THIS IS USEFUL FOR TROUBLE-SHOOTING PURPOSES.
 *
 * @see     WallModel#getUpperRightHandCorner()
 * @since   SoundscapeSimulation1.0
 */
public IThreeDimensionalPoint getUpperRightHandCorner();

/**
 * THIS ACCESSOR RETURNS A REFERENCE TO THE Wall
 * INSTANCE'S LOWER LEFT-HAND CORNER.  SOMETIMES,
 * THIS IS USEFUL FOR TROUBLE-SHOOTING PURPOSES.
 *
 * @see     WallModel#getLowerLeftHandCorner()
 * @since   SoundscapeSimulation1.0
 */
public IThreeDimensionalPoint getLowerLeftHandCorner();

/**
 * THIS ACCESSOR RETURNS A REFERENCE TO THE Wall
 * INSTANCE'S LOWER RIGHT-HAND CORNER.  SOMETIMES,
 * THIS IS USEFUL FOR TROUBLE-SHOOTING PURPOSES.
 *
 * @see     WallModel#getLowerRightHandCorner()
 * @since   SoundscapeSimulation1.0
 */
public IThreeDimensionalPoint getLowerRightHandCorner();


/**
 * THIS ACCESSOR RETURNS THE DISTANCE BETWEEN
 * THE WALL'S EchoPoints.
 *
 * @see     WallModel#getResolution()
 * @since   SoundscapeSimulation1.0
 */
public double getResolution();

/**
 * THIS MODIFIER SETS THE WallModel's CURRENT
 * EchoPoint RESOLUTION.  THE VALUE IN_dResolution
```

```
     * SHOULD BE GIVEN IN METERS.
     *
     * @see     WallModel#setResolution( double IN_dResolution )
     * @since   SoundscapeSimulation1.0
     */
    public void setResolution( double IN_dResolution );
}
```

```
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IWallModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The interface <code>IWallModel</code> was written specifically for
 * the WallModel class, so that collaborating classes could use
 * a lightweight reference of the avaiable public methods of
 * an WallModel reference at runtime.
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see     WallModel
 * @since   SoundscapeSimulator1.0
 */
//-----------------------------------------------------------------------

import ICloakable;
import IPositionHolderModel;
import IWall;


public interface IWallModel
    extends IWall, ICloakable, IPositionHolderModel
    // AS SUCH, extends IModel, IMutable, AND IPositionHolder
{
    /**
     * THIS MODIFIER ITERATES THROUGH ALL OF
     * A Wall OBJECT'S EchoPoints, AND COPIES
     * THEIR RESPONSES INTO THE RESPONSE ARRAY
     *
     * @see     WallModel#finalizeInstant()
     * @since   SoundscapeSimulation1.0
     */
    public void finalizeInstant();
}
```

```
//-----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//            Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Interface Definition File, (c) 1998 Matt Gentner
//
//  Interface Name: IWallView
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The interface <code>IWallView</code> interface adds a single
 * accessor to the IPositionHolderView interface, for getting
 * a reference to the IWallView instance's IWallModel.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import IView;
import IWallModel;


public interface IWallView
    extends IPositionHolderView
{
    /**
    * This method is used to brighten the
    * shaded display of a WallModel's
    * EchoPoint response values.
    *
    * @see     WallResponsesViewCanvas#brighten()
    * @since   SoundscapeSimulation1.0
    */
    public void brighten();

    /**
    * This method is used to darken the
    * shaded display of a WallModel's
    * EchoPoint response values.
    *
    * @see     WallResponsesViewCanvas#darken()
    * @since   SoundscapeSimulation1.0
    */
    public void darken();

    /**
    * This method is used to fine-brighten the
    * shaded display of a WallModel's
    * EchoPoint response values.
    *
    * @see     WallResponsesViewCanvas#fineBrighten()
    * @since   SoundscapeSimulation1.0
    */
    public void fineBrighten();

    /**
    * This method is used to fine-darken the
    * shaded display of a WallModel's
    * EchoPoint response values.
    *
    * @see     WallResponsesViewCanvas#fineDarken()
    * @since   SoundscapeSimulation1.0
    */
    public void fineDarken();

    /**
    * This accessor returns a reference
    * to the WallView object's associated
    * WallModel, and is often called by
    * the WallController.
    *
    * @see     WallViewFrame#getWallModel()
    * @since   SoundscapeSimulation1.0
    */
    public IWallModel getWallModel();
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          MovingSoundSourceController
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class MovingSoundSourceController accepts and
 * handles all user-interface events which come from
 * controls on the MovingSoundSourceViewFrame.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.Throwable;
import java.lang.Double;
import java.lang.NumberFormatException;

import IModel;
import IView;
import IController;
import PeriodicSoundSourceController;
import IMovingSoundSourceModel;


public class MovingSoundSourceController
    extends PeriodicSoundSourceController
    implements ActionListener, IController
{
    protected IMovingSoundSourceModel myMovingSoundSourceModel = null;
    protected IView myMovingSoundSourceView = null;

    static Label lblDeltaXRate = new Label( "Delta X Rate (m/s):  " );
    static Label lblDeltaYRate = new Label( "Delta Y Rate (m/s):  " );
    static Label lblDeltaZRate = new Label( "Delta Z Rate (m/s):  " );

    protected TextField tfDeltaXRate = null;
    protected TextField tfDeltaYRate = null;
    protected TextField tfDeltaZRate = null;

    public MovingSoundSourceController( int width, int height,
                            IMovingSoundSourceModel MovingSoundSourceModel_param,
                            IView MovingSoundSourceView_param )
    {
        myMovingSoundSourceModel = MovingSoundSourceModel_param;
        myMovingSoundSourceView = MovingSoundSourceView_param;

        setSize( width, height );

        initializeAllAggregates();
        initializeAsPositionHolderController();
        initializeAsMovingSoundSourceController();
        initializeAsPeriodicSoundSourceController();

        setVisible( true );
    }

    protected void initializeAllAggregates()
    {
        myPositionHolderModel = myMovingSoundSourceModel;

        initializeAggegates();
        initializePeriodicSoundSource( (IPeriodicSoundSourceModel)myMovingSoundSourceModel,
                                       (IView)myMovingSoundSourceView );

        initializePeriodicSoundSourceAggegates();
        initializeMovingSoundSourceAggegates();
    }
```

```
protected void initializeAsPositionHolderController()
{
    initializeFrame();
}

protected void initializeAsMovingSoundSourceController()
{
    initializeMovingSoundSourceFrame();
}

protected void initializeAsPeriodicSoundSourceController()
{
    initializePeriodicSoundSourceFrame();
}

protected void initializeMovingSoundSourceAggegates()
{
    btnChange.addActionListener( this );

    tfDeltaXRate = new TextField( 10 );
    tfDeltaYRate = new TextField( 10 );
    tfDeltaZRate = new TextField( 10 );
}


protected void initializeMovingSoundSourceFrame()
{
    gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
    addComponent( lblDeltaXRate );
    gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
    addComponent( tfDeltaXRate );

    gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
    addComponent( lblDeltaYRate );
    gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
    addComponent( tfDeltaYRate );

    gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
    addComponent( lblDeltaZRate );
    gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
    addComponent( tfDeltaZRate );
}

public double getDeltaXRateTextFieldValue()
{
    Double dblX_Temp= new Double( myMovingSoundSourceModel.getDeltaXRate() );
    try {
        if ( ( tfDeltaXRate.getText() ).length() > 0 )
            dblX_Temp = Double.valueOf( tfDeltaXRate.getText() );

    } catch( NumberFormatException nfe )  {
        System.out.println( nfe.toString() );
    }

    return( dblX_Temp.doubleValue() );
}

public double getDeltaYRateTextFieldValue()
{
    Double dblY_Temp= new Double( myMovingSoundSourceModel.getDeltaYRate() );
    try {
        if ( ( tfDeltaYRate.getText() ).length() > 0 )
            dblY_Temp = Double.valueOf( tfDeltaYRate.getText() );

    } catch( NumberFormatException nfe )  {
        System.out.println( nfe.toString() );
    }

    return( dblY_Temp.doubleValue() );
}

public double getDeltaZRateTextFieldValue()
{
    Double dblZ_Temp= new Double( myMovingSoundSourceModel.getDeltaZRate() );
    try {
        if ( ( tfDeltaZRate.getText() ).length() > 0 )
            dblZ_Temp = Double.valueOf( tfDeltaZRate.getText() );

    } catch( NumberFormatException nfe )  {
        System.out.println( nfe.toString() );
    }

    return( dblZ_Temp.doubleValue() );
}
```

```
    ////////////////////////////////////////////////////////////////////////////
    // IController INTERFACE IMPLEMENTATIONS:

    public void update()
    {
        super.update();
    }
    // END OF IController INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // ActionListener INTERFACE IMPLEMENTATIONS:
    public void actionPerformed( ActionEvent ae )
    {
        super.actionPerformed( ae );

        if( ae.getSource() == btnChange )
        {
            myMovingSoundSourceModel.setDeltaXRate( getDeltaXRateTextFieldValue() );
            myMovingSoundSourceModel.setDeltaYRate( getDeltaYRateTextFieldValue() );
            myMovingSoundSourceModel.setDeltaZRate( getDeltaZRateTextFieldValue() );
        }
        else if( ae.getSource() == btnMove )
        {
            myMovingSoundSourceModel.setInitialXCoord( getXTextFieldValue() );
            myMovingSoundSourceModel.setInitialYCoord( getYTextFieldValue() );
            myMovingSoundSourceModel.setInitialZCoord( getZTextFieldValue() );
        }

        myMovingSoundSourceModel.updateAllViews();
    }
    // END OF ActionListener INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////
}
```

```
//-----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          MovingSoundSourceModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 *
 * THE MovingSoundSourceModel CLASS OF OBJECTS DERIVES ALL BEHAVIORS OF
 * THE PeriodicSoundSourceModel CLASS, AND ADDS BEHAVIORS RELATED TO
 * TIME-DEPENDENT MOVEMENT DURING SIMULATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import IMovingSoundSourceModel;
import PeriodicSoundSourceModel;


public class MovingSoundSourceModel
    extends PeriodicSoundSourceModel
    implements IMovingSoundSourceModel
    // AS SUCH, extends IModel, IMovable, IMutable,
    // ISoundSource, AND IPositionHolderModel, IPositionHolder
{
    protected double dInitialXCoord = 0.0e0d;
    protected double dInitialYCoord = 0.0e0d;
    protected double dInitialZCoord = 0.0e0d;

    protected double dDeltaXRate = 0.0e0d;
    protected double dDeltaYRate = 0.0e0d;
    protected double dDeltaZRate = 0.0e0d;

    // DEFAULT CONSTRUCTOR:
    public MovingSoundSourceModel()
    {
        this( 0.0e0d, 0.0e0d, 0.0e0d );
    }

    public MovingSoundSourceModel( double x_coord_param,
                double y_coord_param,
                double z_coord_param )
    {
        this(    x_coord_param,
                y_coord_param,
                z_coord_param,
                0.0e0d );
    }
    // PARAMETERIZED CONSTRUCTOR:
    public MovingSoundSourceModel( double x_coord_param,
                double y_coord_param,
                double z_coord_param,
                double dPowerLevel_param )
    {
        super(    x_coord_param,
                y_coord_param,
                z_coord_param,
                dPowerLevel_param );
    }

    ///////////////////////////////////////////////////////////////////////
    // IModel INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "MovingSoundSourceModel" );
    }
    // END OF IModel INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////
    // IMovable INTERFACE IMPLEMENTATIONS:

    public double getInitialXCoord()
    {
        return dInitialXCoord;
    }

    public double getInitialYCoord()
    {
        return dInitialYCoord;
    }

    public double getInitialZCoord()
    {
        return dInitialZCoord;
    }

    public void setInitialXCoord( double IN_dInitialXCoord )
    {
        dInitialXCoord = IN_dInitialXCoord;
    }

    public void setInitialYCoord( double IN_dInitialYCoord )
    {
        dInitialYCoord = IN_dInitialYCoord;
    }

    public void setInitialZCoord( double IN_dInitialZCoord )
    {
        dInitialZCoord = IN_dInitialZCoord;
    }

    public double getDeltaXRate()
    {
        return dDeltaXRate;
    }

    public double getDeltaYRate()
    {
        return dDeltaYRate;
    }

    public double getDeltaZRate()
    {
        return dDeltaZRate;
    }

    public void setDeltaXRate( double IN_dDeltaXRate )
    {
        dDeltaXRate = IN_dDeltaXRate;
    }

    public void setDeltaYRate( double IN_dDeltaYRate )
    {
        dDeltaYRate = IN_dDeltaYRate;
    }

    public void setDeltaZRate( double IN_dDeltaZRate )
    {
        dDeltaZRate = IN_dDeltaZRate;
    }

    public void moveToTime( double IN_dTime )
    {
        moveTo( ( dInitialXCoord + ( IN_dTime * dDeltaXRate ) ),
                ( dInitialYCoord + ( IN_dTime * dDeltaYRate ) ),
                ( dInitialZCoord + ( IN_dTime * dDeltaZRate ) ) );
    }
    // END OF IMovable INTERFACE IMPLEMENTATIONS
    /////////////////////////////////////////////////////////////////////////

} // END OF MovingSoundSourceModel CLASS DEFINITION
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
// Class:          MovingSoundSourceSignalQueueMgr
// Author:         Matt Gentner
// Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 *
 * THE MovingSoundSourceSignalQueueMgr IS A SignalQueueMgr DERIVATIVE
 * WHICH HAS IMPLEMENTATION SPECIALIZED FOR MANAGING THE EFFECTS OF
 * A MovingSoundSourceModel ON ISoundListeners DURING SIMULATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.lang.String;
import java.util.Vector;

import IEar;
import IMovingSoundSource;
import ISoundListener;
import ISoundSource;
import SignalQueueMgr;


public class MovingSoundSourceSignalQueueMgr extends SoundSourceSignalQueueMgr
        implements ISignalQueueMgr
{
    protected IMovingSoundSource myMovingSoundSource = null;


    public MovingSoundSourceSignalQueueMgr( IEar IN_LeftEar,
                                IEar IN_RightEar,
                                IMovingSoundSource IN_myMovingSoundSource,
                                Vector IN_vecSharedSoundListeners )
    {
        super( IN_LeftEar,
               IN_RightEar,
               IN_myMovingSoundSource,
               IN_vecSharedSoundListeners );

        myMovingSoundSource = IN_myMovingSoundSource;
    }

    ////////////////////////////////////////////////////////////////////////
    // ISignalQueueMgr INTERFACE IMPLEMENTATIONS:

    public void effectSignals( double IN_dCurrentTime )
    {
        if( !myMovingSoundSource.isMuted() )
        {
            double dSignalStrength = 0.0e0d;
            double dSoundSourceTime = 0.0e0d;
            double dAttenuatedIntensity = 0.0e0d;
            double dDistanceToListener = 0.0e0d;
            ISoundListener tempSoundListener = null;

            effectEarSignals( IN_dCurrentTime );

            for( int i = 0; i < vecSharedSoundListeners.size(); i++ )
            {
                tempSoundListener = (ISoundListener)vecSharedSoundListeners.elementAt( i );
                dDistanceToListener = distanceToListener( tempSoundListener, IN_dCurrentTime );
                dSoundSourceTime = ( IN_dCurrentTime - ( dDistanceToListener / dSpeed_Of_Sound ) );
                myMovingSoundSource.makeSound( dSoundSourceTime );

                dAttenuatedIntensity = (    myMovingSoundSource.getAmplitude() /
                                        ( +4.0e0d * Math.PI * dDistanceToListener ) );

                tempSoundListener.rattle( dAttenuatedIntensity );
            }
        }

    }
    // END OF ISignalQueueMgr INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////

    private void effectEarSignals( double IN_dCurrentTime )
    {
        double dSignalStrength = 0.0e0d;
        double dSoundSourceTime = 0.0e0d;
        double dAttenuatedIntensity = 0.0e0d;
        double dDistanceToListener = 0.0e0d;

        if( !myMovingSoundSource.isMutedToLeftEar() )
        {
            dDistanceToListener = distanceToListener( LeftEar, IN_dCurrentTime );
            dSoundSourceTime = ( IN_dCurrentTime - ( dDistanceToListener / dSpeed_Of_Sound ) );
            myMovingSoundSource.makeSound( dSoundSourceTime );

            dAttenuatedIntensity = (    myMovingSoundSource.getAmplitude() /
                                    ( +4.0e0d * Math.PI * dDistanceToListener ) );

            LeftEar.rattle( dAttenuatedIntensity );
        }

        if( !myMovingSoundSource.isMutedToRightEar() )
        {
            dDistanceToListener = distanceToListener( RightEar, IN_dCurrentTime );
            dSoundSourceTime = ( IN_dCurrentTime - ( dDistanceToListener / dSpeed_Of_Sound ) );
            myMovingSoundSource.makeSound( dSoundSourceTime );

            dAttenuatedIntensity = (    myMovingSoundSource.getAmplitude() /
                                    ( +4.0e0d * Math.PI * dDistanceToListener ) );

            RightEar.rattle( dAttenuatedIntensity );
        }
    }

    private double distanceToListener( ISoundListener IN_SoundListener,
                                       double IN_dListenerCurrentTime )
    {
        double dXDistance, dYDistance, dZDistance;
        double dDistanceToListener = 0.0e0d;

        myMovingSoundSource.moveToTime( IN_dListenerCurrentTime );

        dXDistance = ( ( IN_SoundListener.getXCoord() - myMovingSoundSource.getXCoord() ) /
                     ( +1.0e0d - ( myMovingSoundSource.getDeltaXRate() / dSpeed_Of_Sound ) ) );

        dYDistance = ( ( IN_SoundListener.getYCoord() - myMovingSoundSource.getYCoord() ) /
                     ( +1.0e0d - ( myMovingSoundSource.getDeltaYRate() / dSpeed_Of_Sound ) ) );

        dZDistance = ( ( IN_SoundListener.getZCoord() - myMovingSoundSource.getZCoord() ) /
                     ( +1.0e0d - ( myMovingSoundSource.getDeltaZRate() / dSpeed_Of_Sound ) ) );

        try
        {
            dDistanceToListener =
                Math.sqrt(
                    Math.pow( dXDistance, +2.0e0d ) +
                    Math.pow( dYDistance, +2.0e0d ) +
                    Math.pow( dZDistance, +2.0e0d )    );

        }  catch ( ArithmeticException e )
        {
                System.out.println( e.toString() );
        }
        return dDistanceToListener;
    }

}
```

```
//---------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          MovingSoundSourceViewCanvas
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------
/**
 * The class MovingSoundSourceViewCanvas extends PositionHolderViewCanvas
 * and is a visual component of the MovingSoundSourceViewFrame which
 * displays textual attribute values of the MovingSoundSourceMdodel
 * at runtime.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------

import java.awt.Color;
import java.awt.Graphics;
import java.lang.Throwable;

import PeriodicSoundSourceViewCanvas;
import IMovingSoundSourceModel;
import IPositionHolderModel;


public class MovingSoundSourceViewCanvas
    extends PeriodicSoundSourceViewCanvas
    // AS SUCH, extends ViewCanvas
{
    protected IMovingSoundSourceModel myMovingSoundSourceModel = null;

    public MovingSoundSourceViewCanvas( int iDisplayWidth_param,
                                int iDisplayHeight_param,
                                IMovingSoundSourceModel MovingSoundSourceModel_param )
    {
        super( iDisplayWidth_param, iDisplayHeight_param,
               ( (IPeriodicSoundSourceModel)MovingSoundSourceModel_param ) );

        myMovingSoundSourceModel = MovingSoundSourceModel_param;
    }

    public void paint( Graphics g )
    {
        super.paint( g );
        drawMovingSoundSourceText( g );
    }

    protected void drawMovingSoundSourceText( Graphics g )
    {
        g.drawString( "Delta X Rate (m/s):  ", 10, 140 );
        g.drawString( String.valueOf( myMovingSoundSourceModel.getDeltaXRate() ), 130, 140 );
        g.drawString( "Delta Y Rate (m/s):  ", 10, 155 );
        g.drawString( String.valueOf( myMovingSoundSourceModel.getDeltaYRate() ), 130, 155 );
        g.drawString( "Delta Z Rate (m/s):  ", 10, 170 );
        g.drawString( String.valueOf( myMovingSoundSourceModel.getDeltaZRate() ), 130, 170 );
    }

    ////////////////////////////////////////////////////////////////////////
    // View INTERFACE IMPLEMENTATIONS:

    protected void initializeMovingSoundSourceModel( IMovingSoundSourceModel MovingSoundSourceModel_param )
    {
        myMovingSoundSourceModel = MovingSoundSourceModel_param;
    }

    public void update()
    {
        repaint();
    }

    // END OF View INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          MovingSoundSourceViewFrame
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class MovingSoundSourceViewFrame is the main visual
 * user interface component for an MovingSoundSourceModel.
 * It holds display components for MovingSoundSourceModel
 * attributes and controls which allow the user
 * to configure the MovingSoundSourceModel values.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Component;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.lang.String;
import java.lang.Throwable;

import IObserver;
import IMovingSoundSourceModel;
import ITextLogger;
import ITextLogUser;
import IView;
import MovingSoundSourceController;
import MovingSoundSourceViewCanvas;


public class MovingSoundSourceViewFrame extends Frame
    implements IView, IObserver, ITextLogUser
{
    // INSTANCE VARIABLES:
    protected GridBagLayout gridbag = null;
    protected GridBagConstraints gridbagconstraints = null;

    protected IMovingSoundSourceModel myMovingSoundSourceModel = null;
    protected MovingSoundSourceController myMovingSoundSourceController = null;
    protected MovingSoundSourceViewCanvas myMovingSoundSourceView = null;
    protected ITextLogger myTextLog = null;

    // GUI CONSTRUCTOR:
    public MovingSoundSourceViewFrame( IMovingSoundSourceModel MovingSoundSourceModel_param, ITextLogger IN_TextLogger
)
    {
        this( MovingSoundSourceModel_param, "Moving Sound Source", IN_TextLogger );
    }

    public MovingSoundSourceViewFrame( IMovingSoundSourceModel MovingSoundSourceModel_param, String name_param, ITextLo
gger IN_TextLogger )
    {
        super( name_param );

        myMovingSoundSourceModel = MovingSoundSourceModel_param;
        setTextLog( IN_TextLogger );
        myMovingSoundSourceModel.attach( this );
        myMovingSoundSourceModel.setTextLog( IN_TextLogger );

        initializeAggegates();
        initializeGUI();

        setVisible( true );
    }

    ////////////////////////////////////////////////////////////////////////
    // INITIALIZATION METHODS:
    protected void initializeAggegates()
    {
        myMovingSoundSourceView = new MovingSoundSourceViewCanvas( 300, 200, myMovingSoundSourceModel );

        makeController();
    }

    protected void addComponent( Component c )
    {
        gridbag.setConstraints( c, gridbagconstraints );
        add( c );
    }

    public void initializeGUI()
    {
        setLocation( 40, 40 );
        setSize( 300, 600 );

        gridbag = new GridBagLayout();
        gridbagconstraints = new GridBagConstraints();
        setLayout( gridbag );

        gridbagconstraints.weightx = 1.0;
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        gridbagconstraints.fill = GridBagConstraints.BOTH;
        gridbagconstraints.gridheight = 12;
        gridbagconstraints.gridheight = 1;
        gridbagconstraints.weighty = 0.0;           //reset to the default

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( myMovingSoundSourceController );
        addComponent( myMovingSoundSourceView );
    }
    // END OF INITIALIZATION METHODS
    ///////////////////////////////////////////////////////////////////////////


    ///////////////////////////////////////////////////////////////////////////
    // Frame CLASS METHOD OVER-RIDES:
    public void paint( Graphics g )
    {
        myMovingSoundSourceController.repaint();

        myMovingSoundSourceView.repaint();
    }

    // END OF Frame CLASS METHOD OVER-RIDES
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IObserver INTERFACE IMPLEMENTATIONS:

    public void update()
    {
        myMovingSoundSourceController.update();
        myMovingSoundSourceView.update();
    }
    // END OF IObserver INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IView INTERFACE IMPLEMENTATIONS:

    public void makeController()
    {
        myMovingSoundSourceController = new
            MovingSoundSourceController( 300, 320, myMovingSoundSourceModel, this );
    }

    public void finalize() throws Throwable
    {
        myMovingSoundSourceModel.detach( this );
        super.finalize();
    }
    // END OF IView INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "MovingSoundSourceViewFrame" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // IPositionHolderView METHODS:

    public IPositionHolderModel getPositionHolderModel()
    {
        return( (IPositionHolderModel)myMovingSoundSourceModel );
    }
    // END OF IPositionHolderView METHODS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // IPeriodicSoundSourceView METHODS:

    public IPeriodicSoundSourceModel getPeriodicSoundSourceModel()
    {
        return( (IPeriodicSoundSourceModel)myMovingSoundSourceModel );
    }
    // END OF IPeriodicSoundSourceView METHODS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // IMovingSoundSourceView METHODS:

    public IMovingSoundSourceModel getMovingSoundSourceModel()
    {
        return myMovingSoundSourceModel;
    }
    // END OF IMovingSoundSourceView METHODS
    ////////////////////////////////////////////////////////////////////////////

}  // end of MovingSoundSourceViewFrame class
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          PeriodicSoundSourceController
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class PeriodicSoundSourceController accepts
 * and handles all user-interface events which come
 * from controls on the PeriodicSoundSourceViewFrame.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Button;
import java.awt.Checkbox;
import java.awt.GridBagConstraints;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.lang.Throwable;
import java.lang.Double;
import java.lang.NumberFormatException;

import IModel;
import IView;
import IController;
import PositionHolderController;
import IPeriodicSoundSourceModel;
import WaveSelectionGroup;


public class PeriodicSoundSourceController
    extends PositionHolderController
    implements ActionListener, ItemListener, IController
{
    protected IPeriodicSoundSourceModel myPeriodicSoundSourceModel = null;
    protected IView myPeriodicSoundSourceView = null;

    protected Label lblOffsetPhase = new Label( "Offset Phase (deg):  " );
    protected Label lblPowerLevel = new Label( "Power Level:  " );
    protected Label lblFrequency = new Label( "Frequency:  " );

    protected TextField tfOffsetPhase = null;
    protected TextField tfPowerLevel = null;
    protected TextField tfFrequency = null;

    protected Button btnChange = null;

    protected Checkbox CloakingCheckbox = null;
    protected Checkbox MutingCheckbox = null;
    protected Checkbox LeftEarMutingCheckbox = null;
    protected Checkbox RightEarMutingCheckbox = null;

    protected WaveSelectionGroup WaveSelector = null;

    public PeriodicSoundSourceController()
    {
        // WAIT FOR DERIVED CLASS TO INITIALIZE
    }

    public PeriodicSoundSourceController( int width, int height,
                            IPeriodicSoundSourceModel PeriodicSoundSourceModel_param,
                            IView PeriodicSoundSourceView_param )
    {
        super( width, height, PeriodicSoundSourceModel_param );

        initializePeriodicSoundSource( PeriodicSoundSourceModel_param, PeriodicSoundSourceView_param );

        initializePeriodicSoundSourceAggegates();
        initializePeriodicSoundSourceFrame();
        setSize( width, height );
        setVisible( true );
    }

    protected void initializePeriodicSoundSourceAggegates()
    {
        btnChange = new Button( "Change" );
        btnChange.addActionListener( this );

        CloakingCheckbox = new Checkbox( "Cloak" );
        CloakingCheckbox.addItemListener( this );
        CloakingCheckbox.setState( myPeriodicSoundSourceModel.isCloaked() );

        MutingCheckbox = new Checkbox( "Mute" );
        MutingCheckbox.addItemListener( this );
        MutingCheckbox.setState( myPeriodicSoundSourceModel.isMuted() );

        LeftEarMutingCheckbox = new Checkbox( "Mute to Left Ear" );
        LeftEarMutingCheckbox.addItemListener( this );
        LeftEarMutingCheckbox.setState( myPeriodicSoundSourceModel.isMutedToLeftEar() );

        RightEarMutingCheckbox = new Checkbox( "Mute to Right Ear" );
        RightEarMutingCheckbox.addItemListener( this );
        RightEarMutingCheckbox.setState( myPeriodicSoundSourceModel.isMutedToRightEar() );

        tfOffsetPhase = new TextField( 10 );
        tfPowerLevel = new TextField( 10 );
        tfFrequency = new TextField( 10 );

        WaveSelector = new WaveSelectionGroup( this );
    }


    protected void initializePeriodicSoundSourceFrame()
    {
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblOffsetPhase );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfOffsetPhase );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblPowerLevel );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfPowerLevel );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblFrequency );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfFrequency );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( CloakingCheckbox );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnChange );

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( MutingCheckbox );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( LeftEarMutingCheckbox );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( RightEarMutingCheckbox );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( WaveSelector.getSineWaveCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( WaveSelector.getSquareWaveCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( WaveSelector.getSawToothWaveCheckbox() );

    }

    public void initializePeriodicSoundSource(  IModel PeriodicSoundSourceModel_param,
                                IView PeriodicSoundSourceView_param )
    {
        myPeriodicSoundSourceModel = ( (IPeriodicSoundSourceModel)PeriodicSoundSourceModel_param );
        myPeriodicSoundSourceView = ( (IView)PeriodicSoundSourceView_param );
    }

    public double getOffsetPhaseTextFieldValue()
    {
        Double dblX_Temp= new Double( myPeriodicSoundSourceModel.getXCoord() );
        try {
            if ( ( tfOffsetPhase.getText() ).length() > 0 )
                dblX_Temp = Double.valueOf( tfOffsetPhase.getText() );
        } catch( NumberFormatException nfe )  {
            System.out.println( nfe.toString() );
        }

        return( dblX_Temp.doubleValue() );
    }

    public double getPowerLevelTextFieldValue()
    {
        Double dblY_Temp= new Double( myPeriodicSoundSourceModel.getYCoord() );
        try {
            if ( ( tfPowerLevel.getText() ).length() > 0 )
                dblY_Temp = Double.valueOf( tfPowerLevel.getText() );

        } catch( NumberFormatException nfe )  {
            System.out.println( nfe.toString() );
        }

        return( dblY_Temp.doubleValue() );
    }

    public double getFrequencyTextFieldValue()
    {
        Double dblZ_Temp= new Double( myPeriodicSoundSourceModel.getZCoord() );
        try {
            if ( ( tfFrequency.getText() ).length() > 0 )
                dblZ_Temp = Double.valueOf( tfFrequency.getText() );

        } catch( NumberFormatException nfe )  {
            System.out.println( nfe.toString() );
        }

        return( dblZ_Temp.doubleValue() );
    }

    ///////////////////////////////////////////////////////////////////////////
    // IController INTERFACE IMPLEMENTATIONS:

    public void update()
    {
        super.update();

        CloakingCheckbox.setState( myPeriodicSoundSourceModel.isCloaked() );
        MutingCheckbox.setState( myPeriodicSoundSourceModel.isMuted() );
    }
    // END OF IController INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ActionListener INTERFACE IMPLEMENTATIONS:
    public void actionPerformed( ActionEvent ae )
    {
        super.actionPerformed( ae );

        if( ae.getSource() == btnChange )
        {
            myPeriodicSoundSourceModel.setOffsetPhase( getOffsetPhaseTextFieldValue() );
            myPeriodicSoundSourceModel.setPowerLevel( getPowerLevelTextFieldValue() );
            myPeriodicSoundSourceModel.setFrequency( getFrequencyTextFieldValue() );
        }
        myPeriodicSoundSourceModel.updateAllViews();
    }
    // END OF ActionListener INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ItemListener INTERFACE IMPLEMENTATIONS:
    public void itemStateChanged( ItemEvent ie )
    {
        if( ie.getSource() == CloakingCheckbox )
        {
            if( CloakingCheckbox.getState() )
                myPeriodicSoundSourceModel.cloak();
            else
                myPeriodicSoundSourceModel.uncloak();
        }
        else if( ie.getSource() == MutingCheckbox )
        {
            myPeriodicSoundSourceModel.setMuted( MutingCheckbox.getState() );
        }
        else if( ie.getSource() == LeftEarMutingCheckbox )
        {
            myPeriodicSoundSourceModel.setMutedToLeftEar( LeftEarMutingCheckbox.getState() );
        }
        else if( ie.getSource() == RightEarMutingCheckbox )
        {
            myPeriodicSoundSourceModel.setMutedToRightEar( RightEarMutingCheckbox.getState() );
        }
        else if( ie.getSource() == WaveSelector.getSineWaveCheckbox() )
        {
            if( ( WaveSelector.getSineWaveCheckbox() ).getState() )
                myPeriodicSoundSourceModel.setWaveShape( IPeriodicSoundSourceModel.SINE );
        }
        else if( ie.getSource() == WaveSelector.getSquareWaveCheckbox() )
        {
            if( ( WaveSelector.getSquareWaveCheckbox() ).getState() )
                myPeriodicSoundSourceModel.setWaveShape( IPeriodicSoundSourceModel.SQUARE );
        }
        else if( ie.getSource() == WaveSelector.getSawToothWaveCheckbox() )
        {
            if( ( WaveSelector.getSawToothWaveCheckbox() ).getState() )
                myPeriodicSoundSourceModel.setWaveShape( IPeriodicSoundSourceModel.SAWTOOTH );
        }
    }
    // END OF ItemListener INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

}
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          PeriodicSoundSourceModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * THE PeriodicSoundSourceModel IS A CLASS OF OBJECTS WHICH ARE USER-
 * CONFIGURABLE ABSTRACTIONS OF REAL WORLD SOUND SOURCES.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//------------------------------------------------------------------------

import java.util.Vector;

import IObserver;
import ITextLogger;
import ITextLogUser;
import IPeriodicSoundSourceModel;
import IPeriodicSoundSourceModel;


public class PeriodicSoundSourceModel
    implements IPeriodicSoundSourceModel
    // AS SUCH, implements IModel ICloakable, IMutable,
    // ISoundSource, AND IPositionHolder
{
    // CONSTANT USED TO SCALE WAVE FORMS
    protected final static double dSineRMSFactor = Math.sin( Math.PI * ( +4.5e+1d / +1.8e+2d ) );

    // INSTANCE VARIABLES:
    protected boolean bCloaked = false;
    protected boolean bMuted = false;
    protected boolean bMutedToLeftEar = false;
    protected boolean bMutedToRightEar = false;
    protected double x_coord = 0.0e0d;
    protected double y_coord = 0.0e0d;
    protected double z_coord = 0.0e0d;
    protected double dAmplitude = 0.0e0d;
    protected double dPowerLevel = 0.0e0d;
    protected double dFrequency = 0.0e0d;
    protected double dRadialFrequency = 0.0e0d;
    protected double dOffsetPhase = 0.0e0d;
    protected double dOffsetPhase_InRadians = 0.0e0d;
    protected int iWaveShape = SINE;

    protected ITextLogger myTextLog = null;
    protected Vector vecPeriodicSoundSourceObservers = null;

    // DEFAULT CONSTRUCTOR:
    public PeriodicSoundSourceModel()
    {
        this( 0.0e0d, 0.0e0d, 0.0e0d );
    }

    public PeriodicSoundSourceModel( double x_coord_param,
                double y_coord_param,
                double z_coord_param )
    {
        this(    x_coord_param,
                y_coord_param,
                z_coord_param,
                0.0e0d );
    }
    // PARAMETERIZED CONSTRUCTOR:
    public PeriodicSoundSourceModel( double x_coord_param,
                double y_coord_param,
                double z_coord_param,
                double dPowerLevel_param )
    {
        x_coord = x_coord_param;
        y_coord = y_coord_param;
        z_coord = z_coord_param;
        dPowerLevel = dPowerLevel_param;
        dAmplitude = 0.0e0d;
        vecPeriodicSoundSourceObservers = new Vector( 3, 3 );
    }


    ///////////////////////////////////////////////////////////////////////////
    // ICloakable INTERFACE IMPLEMENTATIONS:

    // USED TO CHECK IF A Cloakable
    // OBJECT IS CLOAKED AT RUNTIME.
    public boolean isCloaked()
    {
        return bCloaked;
    }

    // USED TO CANCEL GUI REDRAWS,
    // AND SPEED PERFORMANCE DURING
    // LONG STRETCHES OF INTENSE SIMULATION.
    public void cloak()
    {
        bCloaked = true;
        myTextLog.logText( getClassDefName() + " has been cloaked. \n" );
    }

    // (RE-)ENABLES A Cloakable OBJECT'S
    // GUI FOR GRAPHICAL UPDATES.
    public void uncloak()
    {
        bCloaked = false;
        myTextLog.logText( getClassDefName() + " has been un-cloaked. \n" );
    }
    // END OF ICloakable INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IModel INTERFACE IMPLEMENTATIONS:
    public void attach( IObserver SubscribingObserver_param )
    {
        vecPeriodicSoundSourceObservers.addElement( SubscribingObserver_param );
        // setTextLog( ( (ITextLogUser)SubscribingObserver_param ).getTextLog() );
        // myTextLog.logText(  "In " + getClassDefName() + ", attached observer. \n" );
    }

    public void detach( IObserver UnSubscribingObserver_param )
    {
        for( int i = 0; i < vecPeriodicSoundSourceObservers.size(); i++ )
        {
            if( (IObserver)vecPeriodicSoundSourceObservers.elementAt( i ) ==
                UnSubscribingObserver_param )
                vecPeriodicSoundSourceObservers.removeElementAt( i );
        }
    }

    public void updateAllViews()
    {
        for( int i = 0; i < vecPeriodicSoundSourceObservers.size(); i++ )
            ( (IObserver)vecPeriodicSoundSourceObservers.elementAt( i ) ).update();
    }

    public String getClassDefName()
    {
        return( "PeriodicSoundSourceModel" );
    }
    // END OF IModel INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IMutable INTERFACE IMPLEMENTATIONS:
    public boolean isMuted()
    {
        return bMuted;
    }

    public void setMuted( boolean IN_bMuted )
    {
        bMuted = IN_bMuted;

        if( bMuted )
            myTextLog.logText( getClassDefName() + " has been muted. \n" );
        else
            myTextLog.logText( getClassDefName() + " has been un-muted. \n" );
    }

    public boolean isMutedToLeftEar()
    {
        return bMutedToLeftEar;
    }

    public void setMutedToLeftEar( boolean IN_bMutedToLeftEar )
    {
        bMutedToLeftEar = IN_bMutedToLeftEar;

        if( bMutedToLeftEar )
            myTextLog.logText( getClassDefName() + " has been muted to the left ear. \n" );
        else
            myTextLog.logText( getClassDefName() + " has been un-muted to the left ear. \n" );
    }

    public boolean isMutedToRightEar()
    {
        return bMutedToRightEar;
    }

    public void setMutedToRightEar( boolean IN_bMutedToRightEar )
    {
        bMutedToRightEar = IN_bMutedToRightEar;

        if( bMutedToRightEar )
            myTextLog.logText( getClassDefName() + " has been muted to the right ear. \n" );
        else
            myTextLog.logText( getClassDefName() + " has been un-muted to the right ear. \n" );
    }
    // END OF IMutable INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IPositionHolder INTERFACE IMPLEMENTATIONS:

    public double getXCoord()
    {
        return x_coord;
    }

    public double getYCoord()
    {
        return y_coord;
    }

    public double getZCoord()
    {
        return z_coord;
    }

    public void moveTo( double x_coord_param,
                        double y_coord_param,
                        double z_coord_param )
    {
        x_coord = x_coord_param;
        y_coord = y_coord_param;
        z_coord = z_coord_param;

        if( !bCloaked )
        {
            myTextLog.logText( getClassDefName() + " has been moved to x_coord = " +
                x_coord + ", y_coord = " + y_coord + ", z_coord = " + z_coord + " \n" );
        }
    }
    // END OF IPositionHolder INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IPeriodicSoundSourceModel INTERFACE IMPLEMENTATIONS:                          //

    // THIS ACCESSOR GIVES THE PeriodicSoundSourceModel's RUNTIME
    // OFFSET PHASE, IN DEGREES.
    public double getOffsetPhase()
    {
        return dOffsetPhase;
    }

    // THIS MODIFIER SETS THE PeriodicSoundSourceModel's RUNTIME
    // OFFSET PHASE, IN DEGREES.
    public void setOffsetPhase( double dOffsetPhase_param )
    {
        dOffsetPhase = dOffsetPhase_param;
        updateRadialMeasurements();
    }

    // THIS ACCESSOR GIVES THE PeriodicSoundSourceModel's RUNTIME
    // FREQUENCY, IN HERTZ.
```

```
public double getFrequency()
{
    return dFrequency;
}

// THIS MODIFIER SETS THE PeriodicSoundSourceModel's RUNTIME
// FREQUENCY, IN HERTZ.
public void setFrequency( double dFrequency_param )
{
    dFrequency = dFrequency_param;
    updateRadialMeasurements();
}

// THIS ACCESSOR GIVES THE PeriodicSoundSourceModel's RUNTIME
// WAVESHAPE, IN TERMS OF THE CONSTANTS DEFINED ABOVE.
public int getWaveShape()
{
    return iWaveShape;
}

// THIS MODIFIER SETS THE PeriodicSoundSourceModel's RUNTIME
// WAVESHAPE, IN TERMS OF THE CONSTANTS DEFINED ABOVE.
public void setWaveShape( int iWaveShape_param )
{
    iWaveShape = iWaveShape_param;

    if( iWaveShape == SINE )
        myTextLog.logText( getClassDefName() + "'s waveshape set to SINE. \n" );

    else if( iWaveShape == SQUARE )
        myTextLog.logText( getClassDefName() + "'s waveshape set to SQUARE. \n" );

    else if( iWaveShape == SAWTOOTH )
        myTextLog.logText( getClassDefName() + "'s waveshape set to SAWTOOTH. \n" );
}

// END OF IPeriodicSoundSourceModel INTERFACE IMPLEMENTATIONS                    //
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// ISoundSource INTERFACE IMPLEMENTATIONS:
public double getAmplitude()
{
    return dAmplitude;
}

public void setAmplitude( double dAmplitude_param )
{
    dAmplitude = dAmplitude_param;
}

public double getPowerLevel()
{
    return dPowerLevel;
}

public void setPowerLevel( double dPowerLevel_param )
{
    dPowerLevel = dPowerLevel_param;
}
// END OF ISoundSource INTERFACE IMPLEMENTATIONS
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// MODIFIER METHODS:                                                        //

protected void updateRadialMeasurements()
{
    dOffsetPhase_InRadians = ( ( dOffsetPhase / +3.600e+2d ) * ( +2.0e0d ) * Math.PI );
    dRadialFrequency = ( ( +2.0e0d ) * Math.PI * dFrequency );
}
// END OF MODIFIER METHODS                                                  //
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// IMPLEMENTATIONS FOR PARENT CLASS ABSTRACT MODIFIER METHODS:              //

public void makeSound( double dTime_Param )
{
    if( ( bMuted ) || ( dTime_Param < 0.0e0d ) )
        setAmplitude( 0.0e0d );

    else
    {
        if( iWaveShape == SINE )
```

```
                makeSineWaveSound( dTime_Param );

            else if( iWaveShape == SQUARE )
                makeSquareWaveSound( dTime_Param );

            else if( iWaveShape == SAWTOOTH )
                makeSawToothWaveSound( dTime_Param );
        }

        if( !bCloaked )
            updateAllViews();
    }

    protected void makeSineWaveSound( double dTime_Param )
    {
            setAmplitude( dPowerLevel *
                Math.sin( ( dRadialFrequency * dTime_Param )
                            + dOffsetPhase_InRadians ) );
    }

    protected void makeSquareWaveSound( double dTime_Param )
    {
        double dCurrentSineValue =
            Math.sin( ( dRadialFrequency * dTime_Param )
                            + dOffsetPhase_InRadians );

        if( dCurrentSineValue >= 0 )
            setAmplitude( dPowerLevel * dSineRMSFactor );
        else
            setAmplitude( -1.0e0d * dPowerLevel * dSineRMSFactor );
    }

    protected void makeSawToothWaveSound( double dTime_Param )
    {
        double dTheta = ( ( dRadialFrequency * dTime_Param ) + dOffsetPhase_InRadians );
        double dThetaDividedByPI = ( dTheta / Math.PI );

            setAmplitude( dPowerLevel * dSineRMSFactor *
                Math.abs( dThetaDividedByPI - Math.floor( dThetaDividedByPI ) ) );
    }

    // END OF ABSTRACT MODIFIER METHOD IMPLEMENTATIONS                  //
    //////////////////////////////////////////////////////////////////////

    //////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = IN_TextLog;
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    //////////////////////////////////////////////////////////////////////

} // END OF PeriodicSoundSourceModel CLASS DEFINITION
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          PeriodicSoundSourceViewCanvas
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class PeriodicSoundSourceViewCanvas extends PositionHolderViewCanvas
 * and is a visual component of the PeriodicSoundSourceViewFrame which
 * displays textual attribute values of the PeriodicSoundSourceMdodel
 * at runtime.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Color;
import java.awt.Graphics;
import java.lang.Throwable;

import PositionHolderViewCanvas;
import IPeriodicSoundSourceModel;
import IPositionHolderModel;


public class PeriodicSoundSourceViewCanvas
    extends PositionHolderViewCanvas
    // AS SUCH, extends ViewCanvas
{
    protected IPeriodicSoundSourceModel myPeriodicSoundSourceModel = null;

    public PeriodicSoundSourceViewCanvas(    int iDisplayWidth_param,
                            int iDisplayHeight_param,
                            IPeriodicSoundSourceModel PeriodicSoundSourceModel_param )
    {
        super( iDisplayWidth_param, iDisplayHeight_param,
                ( (IPositionHolderModel)PeriodicSoundSourceModel_param ) );

        myPeriodicSoundSourceModel = PeriodicSoundSourceModel_param;
    }

    public void paint( Graphics g )
    {
        super.paint( g );
        drawPeriodicSoundSourceText( g );
    }

    protected void drawPeriodicSoundSourceText( Graphics g )
    {
        g.drawString( "Amplitude:  ", 10, 80 );
        g.drawString( String.valueOf( myPeriodicSoundSourceModel.getAmplitude() ), 100, 80 );
        g.drawString( "OffsetPhase:  (deg.)", 10, 95 );
        g.drawString( String.valueOf( myPeriodicSoundSourceModel.getOffsetPhase() ), 130, 95 );
        g.drawString( "PowerLevel:  (W)", 10, 110 );
        g.drawString( String.valueOf( myPeriodicSoundSourceModel.getPowerLevel() ), 130, 110 );
        g.drawString( "Frequency:  (hz)", 10, 125 );
        g.drawString( String.valueOf( myPeriodicSoundSourceModel.getFrequency() ), 130, 125 );
    }

    //////////////////////////////////////////////////////////////////////
    // View INTERFACE IMPLEMENTATIONS:

    public void update()
    {
        repaint();
    }
    // END OF View INTERFACE IMPLEMENTATIONS
    //////////////////////////////////////////////////////////////////////

    //////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "PeriodicSoundSourceViewCanvas" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////////

}  // end of PeriodicSoundSourceViewCanvas class
```

```
//----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:           PeriodicSoundSourceViewFrame
//  Author:          Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class PeriodicSoundSourceViewFrame is the main visual
 * user interface component for an PeriodicSoundSourceModel.
 * It holds display components for PeriodicSoundSourceModel
 * attributes and controls which allow the user
 * to configure the PeriodicSoundSourceModel values.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Component;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.lang.String;
import java.lang.Throwable;

import IObserver;
import IPeriodicSoundSourceModel;
import ITextLogger;
import ITextLogUser;
import IView;
import PeriodicSoundSourceController;
import PeriodicSoundSourceViewCanvas;


public class PeriodicSoundSourceViewFrame extends Frame
    implements IPeriodicSoundSourceView
    // AS SUCH, implements IObserver, ITextLogUser
{
    // INSTANCE VARIABLES:
    protected GridBagLayout gridbag = null;
    protected GridBagConstraints gridbagconstraints = null;

    protected IPeriodicSoundSourceModel myPeriodicSoundSourceModel = null;
    protected PeriodicSoundSourceController myPeriodicSoundSourceController = null;
    protected PeriodicSoundSourceViewCanvas myPeriodicSoundSourceView = null;
    protected ITextLogger myTextLog = null;

    // GUI CONSTRUCTOR:
    public PeriodicSoundSourceViewFrame( IPeriodicSoundSourceModel PeriodicSoundSourceModel_param, ITextLogger IN_TextL
ogger )
    {
        this( PeriodicSoundSourceModel_param, "Periodic Sound Source", IN_TextLogger );
    }

    public PeriodicSoundSourceViewFrame( IPeriodicSoundSourceModel PeriodicSoundSourceModel_param, String name_param, I
TextLogger IN_TextLogger )
    {
        super( name_param );

        myPeriodicSoundSourceModel = PeriodicSoundSourceModel_param;
        setTextLog( IN_TextLogger );
        myPeriodicSoundSourceModel.attach( this );
        myPeriodicSoundSourceModel.setTextLog( IN_TextLogger );

        initializeAggegates();
        initializeGUI();

        setVisible( true );
    }

    //////////////////////////////////////////////////////////////////////////
    // INITIALIZATION METHODS:
    protected void initializeAggegates()
    {
        myPeriodicSoundSourceView = new PeriodicSoundSourceViewCanvas( 300, 140, myPeriodicSoundSourceModel );
        makeController();
    }

    protected void addComponent( Component c )
    {
        gridbag.setConstraints( c, gridbagconstraints );
        add( c );
    }

    public void initializeGUI()
    {
        setLocation( 20, 20 );
        setSize( 300, 500 );

        gridbag = new GridBagLayout();
        gridbagconstraints = new GridBagConstraints();
        setLayout( gridbag );

        gridbagconstraints.weightx = 1.0;
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        gridbagconstraints.fill = GridBagConstraints.BOTH;
        gridbagconstraints.gridheight = 12;
        gridbagconstraints.gridheight = 1;
        gridbagconstraints.weighty = 0.0;           //reset to the default

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( myPeriodicSoundSourceController );
        addComponent( myPeriodicSoundSourceView );
    }
    // END OF INITIALIZATION METHODS
    ///////////////////////////////////////////////////////////////////////////


    ///////////////////////////////////////////////////////////////////////////
    // Frame CLASS METHOD OVER-RIDES:
    public void paint( Graphics g )
    {
        myPeriodicSoundSourceController.repaint();

        myPeriodicSoundSourceView.repaint();
    }

    // END OF Frame CLASS METHOD OVER-RIDES
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IObserver INTERFACE IMPLEMENTATIONS:

    public void update()
    {
        myPeriodicSoundSourceController.update();
        myPeriodicSoundSourceView.update();
    }
    // END OF IObserver INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IView INTERFACE IMPLEMENTATIONS:

    public void makeController()
    {
        myPeriodicSoundSourceController = new
            PeriodicSoundSourceController( 300, 220, myPeriodicSoundSourceModel, this );
    }

    public void finalize() throws Throwable
    {
        myPeriodicSoundSourceModel.detach( this );
        super.finalize();
    }
    // END OF IView INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "PeriodicSoundSourceViewFrame" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IPositionHolderView METHODS:

    public IPositionHolderModel getPositionHolderModel()
    {
        return myPeriodicSoundSourceModel;
    }
    // END OF IPositionHolderView METHODS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IPeriodicSoundSourceView METHODS:

    public IPeriodicSoundSourceModel getPeriodicSoundSourceModel()
    {
        return myPeriodicSoundSourceModel;
    }
    // END OF IPeriodicSoundSourceView METHODS
    ///////////////////////////////////////////////////////////////////////////

}  // end of PeriodicSoundSourceViewFrame class
```

```
//---------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          POCPopUp
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------
/**
 * The class POCPopUp extends the JDK's Dialog class, and simply
 * displays point of contact information about the application.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------

import java.awt.Button;
import java.awt.FlowLayout;
import java.awt.Dialog;
import java.awt.Frame;
import java.awt.Label;
import java.awt.Panel;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;


class POCPopUp extends Dialog
        implements ActionListener
{
    protected Button btnOK = null;

    public POCPopUp( Frame parent )
    {
        super( parent, "Point of Contact Info", true );
        setSize( 500, 250 );
        Panel p1 = new Panel();
        p1.setLayout( new FlowLayout() );

        Label l1 = new Label( "Soundscape-SA Auditory Scene Simulator" );
        l1.setAlignment( Label.CENTER );
        p1.add( l1 );

        p1.add( new Label( "Department of Computer and Information Science" ) );
        p1.add( new Label( "SUNY Institute of Technology, Utica/Rome New York USA" ) );
        p1.add( new Label( "http://www.cs.sunyit.edu/~gentnem" ) );
        p1.add( new Label( "Free for non-commercial use, (c) 1998 Matt Gentner " ) );

        add( "Center", p1 );

        Panel p2 = new Panel();
        btnOK = new Button( "OK" );
        p2.add( btnOK );
        btnOK.addActionListener( this );
        add( "South", p2 );
        addWindowListener( getWindowAdapter() );
        setVisible( true );
    }

    public void actionPerformed( ActionEvent e )
    {
        if( e.getSource() == btnOK )
        {
            dispose();
        }
    }

    protected WindowAdapter getWindowAdapter()
    {
        return( new WindowAdapter()
            {
                public void windowClosing( WindowEvent e )
                {
                    dispose();
                }
            } );
    }
}
```

```java
//--------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          PositionHolderController
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//--------------------------------------------------------------------
/**
 * The class PositionHolderController accepts and
 * handles all user-interface events which come
 * from controls on an associated ViewFrame.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//--------------------------------------------------------------------

import java.awt.Button;
import java.awt.Component;
import java.awt.Label;
import java.awt.TextField;
import java.awt.Panel;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.Double;
import java.lang.NumberFormatException;
import java.lang.String;
import java.lang.Throwable;

import IModel;
import IController;


public class PositionHolderController
    extends Panel
    implements ActionListener, IController
{
    protected IPositionHolderModel myPositionHolderModel = null;

    protected Label lblXCoord = new Label( "Xcoord (m):  " );
    protected Label lblYCoord = new Label( "Ycoord (m):  " );
    protected Label lblZCoord = new Label( "Zcoord (m):  " );

    protected TextField tfXCoord = null;
    protected TextField tfYCoord = null;
    protected TextField tfZCoord = null;

    protected Button btnMove = null;

    protected GridBagLayout gridbag = null;
    protected GridBagConstraints gridbagconstraints = null;

    public PositionHolderController()
    {
        // WAIT FOR DERIVED CLASS TO INITIALIZE
    }

    public PositionHolderController( int width, int height,
                            IPositionHolderModel PositionHolderModel_param )
    {
        myPositionHolderModel = PositionHolderModel_param;

        initializeAggegates();
        initializeFrame();

        setSize( width, height );
        setVisible( true );
    }

    protected void initializeAggegates()
    {
        btnMove = new Button( "Move" );
        btnMove.addActionListener( this );

        tfXCoord = new TextField( 10 );
        tfYCoord = new TextField( 10 );
        tfZCoord = new TextField( 10 );

        gridbag = new GridBagLayout();
        gridbagconstraints = new GridBagConstraints();
    }

    protected void addComponent( Component c )
    {
        gridbag.setConstraints( c, gridbagconstraints );
        add( c );
    }

    protected void initializeFrame()
    {
        setLayout( gridbag );

        gridbagconstraints.weightx = 1.0;
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        gridbagconstraints.fill = GridBagConstraints.BOTH;
        gridbagconstraints.gridheight = 12;
        gridbagconstraints.gridheight = 1;
        gridbagconstraints.weighty = 0.0;          //reset to the default

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblXCoord );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfXCoord );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblYCoord );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfYCoord );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblZCoord );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfZCoord );

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnMove );

    }

    public double getXTextFieldValue()
    {
        Double dblX_Temp= new Double( myPositionHolderModel.getXCoord() );
        try {
            if ( ( tfXCoord.getText() ).length() > 0 )
                dblX_Temp = Double.valueOf( tfXCoord.getText() );

        } catch( NumberFormatException nfe )  {
            System.out.println( nfe.toString() );
        }

        return( dblX_Temp.doubleValue() );
    }

    public double getYTextFieldValue()
    {
        Double dblY_Temp= new Double( myPositionHolderModel.getYCoord() );
        try {
            if ( ( tfYCoord.getText() ).length() > 0 )
                dblY_Temp = Double.valueOf( tfYCoord.getText() );

        } catch( NumberFormatException nfe )  {
            System.out.println( nfe.toString() );
        }

        return( dblY_Temp.doubleValue() );
    }

    public double getZTextFieldValue()
    {
        Double dblZ_Temp= new Double( myPositionHolderModel.getZCoord() );
        try {
            if ( ( tfZCoord.getText() ).length() > 0 )
                dblZ_Temp = Double.valueOf( tfZCoord.getText() );

        } catch( NumberFormatException nfe )  {
            System.out.println( nfe.toString() );
        }

        return( dblZ_Temp.doubleValue() );
    }
    public void actionPerformed( ActionEvent ae )
    {
        if( ae.getSource() == btnMove )
        {
            myPositionHolderModel.moveTo( getXTextFieldValue(),
                                          getYTextFieldValue(),
                                          getZTextFieldValue() );
        }
    }

    public void update()
    {
    }

    ///////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "PositionHolderController" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

} // end of PositionHolderController class
```

```
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          PositionHolderViewCanvas
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The class PositionHolderViewCanvas is a visual component
 * of various simulator views.  It shows textual information
 * about the PositionHolder attributes of a simulation model.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.awt.Color;
import java.awt.Graphics;

import ViewCanvas;
import IPositionHolderModel;


public abstract class PositionHolderViewCanvas
    extends ViewCanvas
    // AS SUCH, implements IView
{
    protected IPositionHolderModel myPositionHolderModel = null;
    protected Graphics CurrentGraphicsContext = null;
    protected double x_coord_buffer = 0.0e0d;
    protected double y_coord_buffer = 0.0e0d;
    protected double z_coord_buffer = 0.0e0d;


    public PositionHolderViewCanvas(  int iDisplayWidth_param,
                            int iDisplayHeight_param,
                            IPositionHolderModel PositionHolderModel_param )
    {
        super( iDisplayWidth_param, iDisplayHeight_param,
               PositionHolderModel_param );

        initializePositionHolderModel( PositionHolderModel_param );

        // BUFFER LATEST VALUES OF COORDINATES
        updateCoordBuffers();
    }

    public void paint( Graphics g )
    {
        super.paint( g );
        drawPositonText( g );
        CurrentGraphicsContext = g;
    }

    protected void drawPositonText( Graphics g )
    {
        RestoreColor = g.getColor();

        // BUFFER LATEST VALUES OF COORDINATES
        updateCoordBuffers();

        g.setColor( TextColor );
        g.drawString( "Xcoord (m):  ", 10, 35 );
        g.drawString( String.valueOf( x_coord_buffer ), 100, 35 );
        g.drawString( "Ycoord (m):  ", 10, 50 );
        g.drawString( String.valueOf( y_coord_buffer ), 100, 50 );
        g.drawString( "Zcoord (m):  ", 10, 65 );
        g.drawString( String.valueOf( z_coord_buffer ), 100, 65 );

        g.setColor( RestoreColor );

    }

    protected void updateCoordBuffers()
    {
        // BUFFER LATEST VALUES OF COORDINATES
        x_coord_buffer = myPositionHolderModel.getXCoord();
        y_coord_buffer = myPositionHolderModel.getYCoord();
        z_coord_buffer = myPositionHolderModel.getZCoord();
    }

    ///////////////////////////////////////////////////////////////////////////
    // View INTERFACE IMPLEMENTATIONS:
    protected void initializePositionHolderModel( IPositionHolderModel PositionHolderModel_param )
    {
        myPositionHolderModel = PositionHolderModel_param;
    }

    public void update()
    {
        repaint();
        /*
        System.out.println( "In PositionHolderViewCanvas, update called." );
        RestoreColor = CurrentGraphicsContext.getColor();

        // ERASE OLD VALUES OF COORDINATES
        CurrentGraphicsContext.setColor( getBackground() );
        CurrentGraphicsContext.drawString( String.valueOf( x_coord_buffer ), 100, 20 );
        CurrentGraphicsContext.drawString( String.valueOf( y_coord_buffer ), 100, 35 );
        CurrentGraphicsContext.drawString( String.valueOf( z_coord_buffer ), 100, 50 );

        // BUFFER LATEST VALUES OF COORDINATES
        updateCoordBuffers();

        // DRAW LATEST VALUES OF COORDINATES
        CurrentGraphicsContext.setColor( TextColor );
        CurrentGraphicsContext.drawString( String.valueOf( x_coord_buffer ), 100, 20 );
        CurrentGraphicsContext.drawString( String.valueOf( y_coord_buffer ), 100, 35 );
        CurrentGraphicsContext.drawString( String.valueOf( z_coord_buffer ), 100, 50 );

        CurrentGraphicsContext.setColor( RestoreColor );
        */
    }

    // END OF View INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////
}
```

```
//-----------------------------------------------------------------------
//
//        Department of Computer Science, SUNY Institute of Technology
//         Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          ResponseBufferPanel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The class ResponseBufferPanel extends the JDK's Panel
 * class, and is used to allocate a portion of the
 * EarViewFrame's visual area for the BufferViewCanvas.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.awt.CardLayout;
import java.awt.Dimension;
import java.awt.Panel;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;

import IController;
import IEarModel;
import CircularDoubleArray;
import DecibelResponseViewCanvas;
import SoundIntensityViewCanvas;

public class ResponseBufferPanel
    extends Panel
{
    protected double dZoomFactor;
    protected IEarModel myEarModel;
    protected int iDisplayWidth = 0;
    protected int iDisplayHeight = 0;
    protected CardLayout myCardLayoutManager = null;
    protected IController myEarController = null;
    protected DecibelResponseViewCanvas myDecibelResponseViewCanvas = null;
    protected SoundIntensityViewCanvas mySoundIntensityViewCanvas = null;

    public ResponseBufferPanel(    int iDisplayWidth_param,
                            int iDisplayHeight_param,
                            IEarModel EarModel_param,
                            IController EarController_param )
    {
        iDisplayWidth = iDisplayWidth_param;
        iDisplayHeight = iDisplayHeight_param;
        myEarModel = EarModel_param;
        myEarController = EarController_param;


        myDecibelResponseViewCanvas = new DecibelResponseViewCanvas(  iDisplayWidth,
                                                            iDisplayHeight,
                                                            myEarModel );

        mySoundIntensityViewCanvas = new SoundIntensityViewCanvas(  iDisplayWidth,
                                                            iDisplayHeight,
                                                            myEarModel );

        initializePanel();
    }

    protected void initializePanel()
    {
        myCardLayoutManager = new CardLayout();
        setLayout( myCardLayoutManager );
        add( myDecibelResponseViewCanvas, "DecibelResponse" );
        add( mySoundIntensityViewCanvas, "SoundIntensity" );
        myCardLayoutManager.show( this, "SoundIntensity" );
    }

    public void showSoundIntensity()
    {
        myCardLayoutManager.show( this, "SoundIntensity" );
    }

    public void showDecibelResponse()
    {
        myCardLayoutManager.show( this, "DecibelResponse" );
    }

    public IController getEarController()
    {
        return myEarController;
    }

    public void stretchVertical()
    {
        myDecibelResponseViewCanvas.stretchVertical();
        mySoundIntensityViewCanvas.stretchVertical();
    }

    public void shrinkVertical()
    {
        myDecibelResponseViewCanvas.shrinkVertical();
        mySoundIntensityViewCanvas.shrinkVertical();
    }


    ///////////////////////////////////////////////////////////////////////////
    // IView INTERFACE IMPLEMENTATIONS:
    public void display()
    {
        repaint();
    }

    public void update()
    {
        myDecibelResponseViewCanvas.update();
        mySoundIntensityViewCanvas.update();
    }

    // END OF IView INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "ResponseBufferPanel" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

}  // end of ResponseBufferPanel class
```

```
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          Signal
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * THE Signal CLASS OF OBJECTS ARE CREATED AND QUEUED DURING
 * SIMULATION.  UPON INSTANTIATION, THEY ARE GIVEN REFERENCES TO
 * THEIR ORIGINATING ISoundSource, AND THEIR TARGET ISoundListener.  THE
 * SignalQueueMgr CLASSES OF OBJECTS ARE RESPONSIBLE FOR FINDING READY
 * Signal OBJECTS AND TRIGGERING THEIR EFFECT.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.lang.ArithmeticException;
import java.lang.Math;

import ISoundSource;
import ISoundListener;


public class Signal
{
    // SHARED, Signal CLASS VARIABLES:
    static double dSpeed_Of_Sound = ISoundListener.dSpeed_Of_Sound;

    // INSTANCE VARIABLES:
    protected double dDistance = 0.0e0d;
    protected double dTimeOfEffect = 0.0e0d;
    protected double dSignalStrength = 0.0e0d;

    protected ISoundListener myTargetSoundListener = null;
    protected ISoundSource myOrigSoundSource = null;


    // PARAMETERIZED CONSTRUCTOR:
    public Signal( ISoundListener IN_TargetSoundListener,
                   ISoundSource IN_OrigSoundSource,
                   double IN_dBaseTime )
    {
        myTargetSoundListener = IN_TargetSoundListener;
        myOrigSoundSource = IN_OrigSoundSource;
        initTimeOfEffect( IN_dBaseTime );
        initSignalStrength();
    }


    ////////////////////////////////////////////////////////////////////////
    // INITIALIZATION METHODS:
    public double distanceToTarget()
    {
        if( ( myTargetSoundListener.getXCoord() == myOrigSoundSource.getXCoord() ) &&
            ( myTargetSoundListener.getYCoord() == myOrigSoundSource.getYCoord() ) &&
            ( myTargetSoundListener.getZCoord() == myOrigSoundSource.getZCoord() ) )
        {
            dDistance = 0.0e0d;
        }

        else
        {
            try
            {
                dDistance =
                    Math.sqrt(
                        Math.pow( ( myTargetSoundListener.getXCoord() -
                                    myOrigSoundSource.getXCoord() ), ( +2.0e0d ) ) +
                        Math.pow( ( myTargetSoundListener.getYCoord() -
                                    myOrigSoundSource.getYCoord() ), ( +2.0e0d ) ) +
                        Math.pow( ( myTargetSoundListener.getZCoord() -
                                    myOrigSoundSource.getZCoord() ), ( +2.0e0d ) )
                        );

            } catch ( ArithmeticException e )
            {
                    System.out.println( e.toString() );
            }
        }

        return dDistance;
    }

    protected void initSignalStrength()
    {
        if( myOrigSoundSource.isMuted() ) // THIS Signal's ISoundSource IS MUTED.
            dSignalStrength = 0.0e0d;

        else
        {
            dSignalStrength = (    myOrigSoundSource.getAmplitude() /
                            ( ( +4.0e0d ) * Math.PI * dDistance ) );
        }
    }

    protected void initTimeOfEffect( double IN_dBaseTime )
    {
        dTimeOfEffect = IN_dBaseTime +
                            ( distanceToTarget() / dSpeed_Of_Sound );
    }
    // END OF INITIALIZATION METHODS
    /////////////////////////////////////////////////////////////////////////////


    /////////////////////////////////////////////////////////////////////////////
    // MODIFIER METHODS:
    public void reInitialize( ISoundListener IN_TargetSoundListener,
                              ISoundSource IN_OrigSoundSource,
                              double IN_dBaseTime )
    {
        // THIS IS FASTER THAN HEAP (DEALLOCATION AND) ALLOCATION
        // OF Signal INSTANCES DURING SIMULATION, AND SHOULD BE
        // CALLED BY THE SignalQueue INSTANCES AS THEY MANIPULATE
        // Signal OBJECTS.
        myTargetSoundListener = IN_TargetSoundListener;
        myOrigSoundSource = IN_OrigSoundSource;
        initTimeOfEffect( IN_dBaseTime );
        initSignalStrength();
    }

    public void speak()
    {
        myTargetSoundListener.rattle( dSignalStrength );
        // System.out.println( "Signal has spoken" );
    }
    // END OF MODIFIER METHODS
    /////////////////////////////////////////////////////////////////////////////


    /////////////////////////////////////////////////////////////////////////////
    // ACCESSOR METHODS:
    public double getTimeEffect()
    {
        return dTimeOfEffect;
    }

    public double getSignalStrength()
    {
        return dSignalStrength;
    }
    // END OF ACCESSOR METHODS
    /////////////////////////////////////////////////////////////////////////////

} // END OF Signal CLASS DEFINITION□
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          SignalQueue
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * THE SignalQueue IS A CLASS WHICH IS DESIGNED TO BE USED AS AN
 * ORDERED, FIFO AND UN-INDEXED CONTAINER OF Signal INSTANCES.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.util.Vector;

import SignalQueueElement;


public class SignalQueue
{
    protected static Vector vecSignalQueueElementTrashBin = new Vector( 1000, 100 );

    protected int iSize;
    protected SignalQueueElement qeHead;
    protected SignalQueueElement qeTail;

    public SignalQueue()
    {
        iSize = 0;
        qeHead = null;
        qeTail = null;
    }

    protected SignalQueueElement rummageForSignalQueueElement()
    {
        SignalQueueElement old_SignalQueueElement = null;
        if( vecSignalQueueElementTrashBin.size() > 0 )
        {
            old_SignalQueueElement = (
                (SignalQueueElement)vecSignalQueueElementTrashBin.elementAt( vecSignalQueueElementTrashBin.size() - 1 )
 );

            vecSignalQueueElementTrashBin.removeElementAt( vecSignalQueueElementTrashBin.size() - 1 );
        }

        return( old_SignalQueueElement );
    }

    protected void addToTrashBin( SignalQueueElement IN_OldSignalQueueElement )
    {
        IN_OldSignalQueueElement.neutralize();
        vecSignalQueueElementTrashBin.addElement( IN_OldSignalQueueElement );
    }

    public void enqueueSignal( Signal IN_NewSignal )
    {
        SignalQueueElement qeTemp = null;

        if( vecSignalQueueElementTrashBin.size() == 0 )
        {
            qeTemp = new SignalQueueElement( IN_NewSignal );
        }
        else
        {
            qeTemp = rummageForSignalQueueElement();
            qeTemp.reinitialize( IN_NewSignal );
        }

        if( iSize == 0 )
            qeHead = qeTemp;

        else
            qeTail.setNext( qeTemp );

        qeTail = qeTemp;
        iSize++;
    }

    public Signal dequeueSignal()
    {
        SignalQueueElement qeChoppedHead = null;
        Signal dequeuedSignal = null;

        if( qeHead != null )
        {
            dequeuedSignal = qeHead.getPayload();
            qeChoppedHead = qeHead;
            qeHead = ( qeHead.getNext() );
            addToTrashBin( qeChoppedHead );
            iSize--;
        }

        return( dequeuedSignal );
    }

    public Signal peekAtSignalQueue()
    {
        if( qeHead != null )
            return( qeHead.getPayload() );
        else
            return null;
    }

    public int size()
    {
        return iSize;
    }

    public boolean isEmpty()
    {
        if( iSize > 0 )
            return false;
        else
            return true;
    }
}
```

```
//-----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:            SignalQueueDirector
//  Author:           Matt Gentner
//  Original Date:    January, 1998
//-----------------------------------------------------------------------
/**
 * THE SignalQueueDirector ACTS AS A LIASON BETWEEN THE SimulatorModel
 * AND THE VARIOUS SignalQueueMgr DERIVATIVES.  FOR EACH ADDITIONAL
 * SoundSource, THE SignalQueueDirector CREATES AND COORDINATES AN
 * APPROPRIATE SignalQueueMgr.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.util.Vector;
import java.lang.Math;

import IEarModel;
import IMovingSoundSource;
import IPeriodicSoundSource;
import ITextLogUser;
import IWallModel;
import MovingSoundSourceSignalQueueMgr;
import SoundSourceSignalQueueMgr;
import WallSignalQueueMgr;


public class SignalQueueDirector
    implements ITextLogUser
{
    protected double dSamplePeriod;
    protected IEarModel LeftEarModel;
    protected IEarModel RightEarModel;
    protected ITextLogger myTextLog = null;
    protected String strProgressLog = null;
    protected Vector vecSharedSoundListeners;
    protected Vector vecSignalQueueManagers;


    public SignalQueueDirector( IEarModel IN_LeftEar,
                                IEarModel IN_RightEar )
    {
        this(   IN_LeftEar,
                IN_RightEar,
                ISoundListener.dThreshold_Of_Negligence );
    }

    public SignalQueueDirector( IEarModel IN_LeftEar,
                                IEarModel IN_RightEar,
                                double IN_dNegligibleSoundIntensityFloor )
    {
        LeftEarModel = IN_LeftEar;
        RightEarModel = IN_RightEar;

        vecSharedSoundListeners = new Vector( 1000, 100 );
        vecSignalQueueManagers = new Vector( 10, 5 );
    }

    public void setSamplePeriod( double IN_dSamplePeriod )
    {
        dSamplePeriod = IN_dSamplePeriod;
    }

    public void addPeriodicSoundSource( IPeriodicSoundSource IN_IPeriodicSoundSource )
    {
        vecSignalQueueManagers.addElement(
            new SoundSourceSignalQueueMgr( LeftEarModel,
                                            RightEarModel,
                                            IN_IPeriodicSoundSource,
                                            vecSharedSoundListeners ) );
    }

    public void addMovingSoundSource( IMovingSoundSource IN_IMovingSoundSource )
    {
        vecSignalQueueManagers.addElement(
            new MovingSoundSourceSignalQueueMgr( LeftEarModel,
                                                RightEarModel,
                                                IN_IMovingSoundSource,
                                                vecSharedSoundListeners ) );
    }

    public void addWall( IWallModel IN_IWallModel )
    {
        vecSignalQueueManagers.addElement(
            new WallSignalQueueMgr( LeftEarModel,
                                    RightEarModel,
                                    IN_IWallModel,
                                    vecSharedSoundListeners ) );
    }

    public void initializeSignalQueues()
    {
        strProgressLog = new String( "Creating signal queues " );

        myTextLog.logText( "SignalQueueDirector has " +
            vecSignalQueueManagers.size() + " SignalQueueManager(s) and " +
            ( vecSharedSoundListeners.size() + 2 ) + " sound listener(s). \n" );

        for( int i = 0; i < vecSignalQueueManagers.size(); i++ )
        {
            ( (ISignalQueueMgr)vecSignalQueueManagers.elementAt( i ) ).setSamplePeriod( dSamplePeriod );
            strProgressLog += ( (ISignalQueueMgr)vecSignalQueueManagers.elementAt( i ) ).initializeSignalQueues();
        }

        strProgressLog += ( "  Done." );
        myTextLog.logText( strProgressLog + "\n" );
        myTextLog.logText( ( strProgressLog.length() - 30 ) + " signal queues initialized. \n" );

    }

    public void makeSignals( double IN_dCurrentTime )
    {
        for( int i = 0; i < vecSignalQueueManagers.size(); i++ )
            ( (ISignalQueueMgr)vecSignalQueueManagers.elementAt( i ) ).makeSignals( IN_dCurrentTime );
    }

    protected void dampAllListeners()
    {
        // THIS METHOD DAMPS THE RESPONSE OF EACH ISoundListener TO ZERO,
        // AND SHOULD BE CALLED BETWEEN SIMULATION STEPS
        LeftEarModel.dampResponse();
        RightEarModel.dampResponse();

        for( int i = 0; i < vecSharedSoundListeners.size(); i++ )
            ( (ISoundListener)vecSharedSoundListeners.elementAt( i ) ).dampResponse();
    }


    public void effectSignals( double IN_dCurrentTime )
    {
        for( int i = 0; i < vecSignalQueueManagers.size(); i++ )
            ( (ISignalQueueMgr)vecSignalQueueManagers.elementAt( i ) ).effectSignals( IN_dCurrentTime );
    }


    public void runSignals( double IN_dCurrentTime )
    {
        dampAllListeners();

        effectSignals( IN_dCurrentTime );
    }

    public void finalizeInstants()
    {
        LeftEarModel.finalizeInstant();
        RightEarModel.finalizeInstant();

        for( int i = 0; i < vecSignalQueueManagers.size(); i++ )
            ( (ISignalQueueMgr)vecSignalQueueManagers.elementAt( i ) ).finalizeInstant();
    }

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }
    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = IN_TextLog;
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

}  // end of SignalQueueDirector class definition
```

```
//---------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          SignalQueueElement
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------
/**
 * THE SignalQueueElement CLASS OF OBJECTS ARE MEANT TO BE NODES
 * MANAGED BY THE SignalQueue CONTAINER CLASS.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------

import Signal;


public class SignalQueueElement
{
    protected Signal myPayload;
    protected SignalQueueElement nextSignalQueueElement;

    public SignalQueueElement( Signal IN_myPayload )
    {
        myPayload = IN_myPayload;
        nextSignalQueueElement = null;
    }

    public void reinitialize( Signal IN_myPayload )
    {
        myPayload = IN_myPayload;
        nextSignalQueueElement = null;
    }

    public Signal getPayload()
    {
        return myPayload;
    }

    public SignalQueueElement getNext()
    {
        return nextSignalQueueElement;
    }

    public void setNext( SignalQueueElement IN_nextSignalQueueElement )
    {
        nextSignalQueueElement = IN_nextSignalQueueElement;
    }

    public void neutralize()
    {
        myPayload = null;
        nextSignalQueueElement = null;
    }

} // END OF SignalQueueElement CLASS DEFINITION
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          SignalQueueMgr
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * THE SignalQueueMgr IS A BASE CLASS WHICH IS MEANT TO PROVIDE A COMMON
 * INTERFACE TO THE SPECIALIZED SignalQueueMgr DERIVATIVES, AND ALSO
 * IMPLEMENT COMMON UTILITY BEHAVIORS FOR THEM.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.lang.Math;
import java.lang.String;
import java.util.Vector;

import ISoundListener;
import ISoundSource;


public abstract class SignalQueueMgr
        implements ISignalQueueMgr
{
    protected static double dSamplePeriod;
    protected static double dNegligibleSoundIntensityFloor = ISoundListener.dThreshold_Of_Negligence;
    protected static Vector vecSignalTrashBin = null;

    protected IEar LeftEar = null;
    protected IEar RightEar = null;
    protected Signal sigTempSignal = null;
    protected Vector vecSharedSoundListeners = null;

    public SignalQueueMgr( IEar IN_LeftEar,
                           IEar IN_RightEar,
                           Vector IN_vecSharedSoundListeners )
    {
        LeftEar = IN_LeftEar;
        RightEar = IN_RightEar;
        vecSharedSoundListeners = IN_vecSharedSoundListeners;
    }

    ///////////////////////////////////////////////////////////////////////////
    // ISignalQueueMgr INTERFACE IMPLEMENTATIONS:

    public abstract String initializeSignalQueues();

    public void setSamplePeriod( double IN_dSamplePeriod )
    {
        dSamplePeriod = IN_dSamplePeriod;
    }

    public abstract void makeSignals( double IN_dCurrentTime );

    public abstract void effectSignals( double IN_dCurrentTime );

    public abstract void finalizeInstant();
    // END OF ISignalQueueMgr INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    protected void enlargeTrashBin( int IN_iCapacity )
    {
        if( vecSignalTrashBin == null )
            vecSignalTrashBin = new Vector( IN_iCapacity, 100 );

        else
        {
            vecSignalTrashBin.ensureCapacity( IN_iCapacity + vecSignalTrashBin.capacity() );
        }
    }

    protected boolean signalIsOK( Signal IN_Signal )
    {
        if( ( IN_Signal.distanceToTarget() > 0.0e0d ) &&
                ( Math.abs( IN_Signal.getSignalStrength() ) >= ISoundListener.dThreshold_Of_Negligence ) )
        {
            return true;
        }
        else
            return false;
    }

    protected Signal NewSignal( ISoundListener IN_ISoundListener,
                                ISoundSource IN_ISoundSource,
                                double IN_dCurrentTime )
    {
        Signal tempSignal = null;

        if( vecSignalTrashBin.size() == 0 )
            tempSignal = new Signal( IN_ISoundListener, IN_ISoundSource, IN_dCurrentTime );

        else
        {
            tempSignal = ( (Signal)vecSignalTrashBin.elementAt( vecSignalTrashBin.size() - 1 ) );
            vecSignalTrashBin.removeElementAt( vecSignalTrashBin.size() - 1 );
            tempSignal.reInitialize( IN_ISoundListener, IN_ISoundSource, IN_dCurrentTime );
        }

        return tempSignal;
    }

    protected void addToTrashBin( Signal IN_Signal )
    {
        vecSignalTrashBin.addElement( IN_Signal );
        IN_Signal = null;
    }

}  // end of SignalQueueMgr class definition
```

```
//---------------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          SimulatorController
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------------
/**
 * The class SimulatorController accepts and handles all
 * user-interface events which come from controls
 * on the SimulatorView.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------------

import java.awt.Button;
import java.awt.Component;
import java.awt.Button;
import java.awt.Checkbox;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Label;
import java.awt.Panel;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.io.BufferedOutputStream;
import java.lang.Throwable;

import POCPopUp;
import IController;
import SimulatorView;
import ITextLogger;
import ITextLogUser;
import MovingSoundSourceViewFrame;
import PeriodicSoundSourceViewFrame;
import ISimulatorModel;
import StartTimeSelector;


public class SimulatorController
    extends Panel
    implements ActionListener, IController, ItemListener, ITextLogUser
{
    protected double dCurrentTime = 0.0e0d;
    protected double dStartTime = 0.0e0d;
    protected double dFinishTime = 0.0e0d;
    protected double dSamplePeriod = 0.0e0d;

    protected GridBagLayout gridbag = null;
    protected GridBagConstraints gridbagconstraints = null;

    protected ITextLogger myTextLog = null;
    protected ISimulatorModel mySimulatorModel = null;
    protected SimulatorView mySimulatorView = null;

    protected Button btnAddWall = null;
    protected Button btnAddMovingSoundSource = null;
    protected Button btnAddPeriodicSoundSource = null;
    protected Button btnRunSimulation = null;
    protected Button btnSingleStep = null;
    protected Checkbox cbSaveFile = null;

    protected StartTimeSelector myStartTimeSelector = null;
    protected TextField tfFinish_Time = null;
    protected TextField tfSampleFreq = null;
    protected TextField tfFileName = null;

    public SimulatorController( int width, int height,
                              ISimulatorModel IN_SimulatorModel,
                              SimulatorView IN_SimulatorView )
    {
        setSize( width, height );
        mySimulatorModel = IN_SimulatorModel;
        mySimulatorView = IN_SimulatorView;
        setTextLog( IN_SimulatorView );

        initializeAggregates();
        initializeCanvas();

        setVisible( true );
    }

    ////////////////////////////////////////////////////////////////////
    // PROTECTED INTIALIZATION METHODS:

    protected void addComponent( Component c )
    {
        gridbag.setConstraints( c, gridbagconstraints );
        add( c );
    }

    protected void initializeAggregates()
    {
        gridbag = new GridBagLayout();
        gridbagconstraints = new GridBagConstraints();

        btnAddWall = new Button( "Add Wall Object" ) ;
        btnAddWall.addActionListener( this );

        btnAddMovingSoundSource = new Button( "Add Moving Sound Source" );
        btnAddMovingSoundSource.addActionListener( this );

        btnAddPeriodicSoundSource = new Button( "Add Periodic Sound Source" );
        btnAddPeriodicSoundSource.addActionListener( this );

        btnRunSimulation = new Button( "Run Simulation" );
        btnRunSimulation.addActionListener( this );

        btnSingleStep = new Button( "Single Step" );
        btnSingleStep.addActionListener( this );

        cbSaveFile = new Checkbox( "Save File" );
        cbSaveFile.addItemListener( this );

        myStartTimeSelector = new StartTimeSelector( this );
        tfFinish_Time = new TextField( 10 );
        tfSampleFreq = new TextField( 10 );
        tfFileName = new TextField( 10 );
    }


    protected void initializeCanvas()
    {
        setLayout( gridbag );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        gridbagconstraints.fill = GridBagConstraints.BOTH;
        gridbagconstraints.gridheight = 12;
        gridbagconstraints.gridheight = 1;
        gridbagconstraints.weightx = 1.0;
        gridbagconstraints.weighty = 0.0;           //reset to the default

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnAddWall );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnAddMovingSoundSource );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnAddPeriodicSoundSource );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( new Label( "Start Time ( sec. ):  " ) );
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( myStartTimeSelector.getTimeZeroCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( myStartTimeSelector.getCurrentTimeCheckbox() );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( new Label( "Finish Time ( sec. ):  " ) );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfFinish_Time );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( new Label( "Sampling Frequency (hz):  " ) );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfSampleFreq );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( cbSaveFile );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfFileName );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( btnRunSimulation );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnSingleStep );
    }
    // END OF PROTECTED INTIALIZATION METHODS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IObserver INTERFACE IMPLEMENTATIONS:

    public void update()
    {
    }
    // END OF IObserver INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ActionListener INTERFACE IMPLEMENTATIONS:

    public void actionPerformed( ActionEvent ae )
    {
        if( ae.getSource() == btnAddWall )
        {
            new WallViewFrame( mySimulatorModel.newWallModel(), getTextLog() );
        }

        else if( ae.getSource() == btnAddMovingSoundSource )
        {
            new MovingSoundSourceViewFrame( mySimulatorModel.newMovingSoundSourceModel(), getTextLog() );
        }

        else if( ae.getSource() == btnAddPeriodicSoundSource )
        {
            new PeriodicSoundSourceViewFrame( mySimulatorModel.newPeriodicSoundSourceModel(), getTextLog() );
        }

        else if( ae.getSource() == btnRunSimulation )
        {
            long lStartTime = System.currentTimeMillis();

            runSimulation();

            myTextLog.logText( ( "Ran simulation, elapsed time:  " +
                        ( System.currentTimeMillis() - lStartTime ) + " miliseconds. \n" ) );
        }

        else if( ae.getSource() == btnSingleStep )
        {
            if( mySimulatorModel != null )
                mySimulatorModel.singleStep();
        }

        else if( ae.getSource() == mySimulatorView.getAboutMenuItem() )
        {
            new POCPopUp( mySimulatorView );
        }

        mySimulatorModel.updateAllViews();
    }
    // END OF ActionListener INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ItemListener INTERFACE IMPLEMENTATIONS:
    public void itemStateChanged( ItemEvent ie )
    {
        if( ie.getSource() == myStartTimeSelector.getTimeZeroCheckbox() )
        {
            if( ( myStartTimeSelector.getTimeZeroCheckbox() ).getState() )
            {
                dStartTime = 0.0e0d;
            }
        }
        else if( ie.getSource() == myStartTimeSelector.getCurrentTimeCheckbox() )
        {
            if( ( myStartTimeSelector.getCurrentTimeCheckbox() ).getState() )
            {
                dStartTime = dCurrentTime;
            }
        }
        else if( ie.getSource() == cbSaveFile )
        {
            if( cbSaveFile.getState() )
            {
                String strFileName = tfFileName.getText();

                if( strFileName.length() == 0 )
                    strFileName = "output.dat";

                mySimulatorModel.openOutputFile( strFileName );
            }
            else
                mySimulatorModel.closeOutputFile();
        }

    }
    // END OF ItemListener INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ////////////////////////////////;//////////////////////////////////////////

    protected void displayTimeConstraints()
    {
        myTextLog.logText( ( "Current time is " + dCurrentTime + " seconds. \n" ) );
        myTextLog.logText( "Start time is " + dStartTime + " seconds. \n" );
        myTextLog.logText( "Finish time is " + dFinishTime + " seconds. \n" );
    }


    protected void setTimeConstraints()
    {
        // MAKE COPIES OF CURRENT (GOOD) VALUES, IN CASE OF BAD INPUT
        double dFinishTime_temp = dFinishTime;
        double dSamplePeriod_temp = dSamplePeriod;

        try
        {
            if ( ( tfFinish_Time.getText() ).length() > 0 )
                dFinishTime_temp = ( Double.valueOf( tfFinish_Time.getText() ) ).doubleValue();

        } catch( NumberFormatException nfe )
        {
            dFinishTime_temp = dFinishTime;
            System.out.println( nfe.toString() );
        }

        dFinishTime = dFinishTime_temp;

        try
        {
            if ( ( tfSampleFreq.getText() ).length() > 0 )
                dSamplePeriod_temp = ( Double.valueOf( tfSampleFreq.getText() ) ).doubleValue();

        } catch( NumberFormatException nfe )
        {
            dSamplePeriod_temp = dSamplePeriod;
            System.out.println( nfe.toString() );
        }

        dSamplePeriod = ( +1.0e0d / dSamplePeriod_temp );

        mySimulatorModel.setTimeConstraints( dStartTime,
                                             dFinishTime,
                                             dSamplePeriod );
    }

    protected void runSimulation()
    {
        setTimeConstraints();
        displayTimeConstraints();
        mySimulatorModel.runSimulation();
        System.gc();
    }

    protected void setupSAFStreams( BufferedOutputStream bosLeftChannel,
                                    BufferedOutputStream bosRightChannel )
    // CHECKS IF REFERENCES ARE NOT null, AND IF SO THEN WRITES TO
    // THE GIVEN STREAMS DURING THE SIMULATION.
    {
        // SimulatorController DOES NO STREAMING YET
    }

    ///////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "SimulatorController" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

}  // end of SimulatorController class
```

```
//---------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          SimulatorModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//---------------------------------------------------------------------
/**
 * THE SimulatorModel IS A CLASS WHICH IS DESIGNED TO SERVE THE
 * SimulatorView AND THE SimulatorController ALL MAJOR BACK-END
 * OPERATIONS INVOLVED IN THE Soundscape SIMULATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------

import java.io.BufferedOutputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Vector;

import EarModel;
import ICloakable;
import IObserver;
import IEarModel;
import ISimulatorModel;
import ISoundSource;
import ITextLogUser;
import MovingSoundSourceModel;
import PeriodicSoundSourceModel;
import SignalQueueDirector;
import WallModel;


public class SimulatorModel
    implements ISimulatorModel, ITextLogUser
{
    protected boolean bLeftStreamingActivated = false;
    protected boolean bRightStreamingActivated = false;
    protected boolean bSavingOutputFile = false;
    protected double dStartTime = 0.0e0d;
    protected double dFinishTime = 0.0e0d;
    protected double dDuration = 0.0e0d;
    protected double dSamplePeriod = 0.0e0d;
    protected double dCurrentTime = 0.0e0d;

    protected DataOutputStream dosOutputFile = null;
    protected SignalQueueDirector mySignalQueueDirector = null;

    protected EarModel LeftEarModel = null;
    protected EarModel RightEarModel = null;
    protected ITextLogger myTextLog = null;

    protected Vector vecObservers = null;
    protected Vector vecCloakables = null;

    public SimulatorModel()
    {
        initializeVectors();
        initializeEarModels();

        mySignalQueueDirector = new SignalQueueDirector( LeftEarModel, RightEarModel );
    }

    protected void initializeVectors()
    {
        vecCloakables = new Vector( 4, 2 );
        vecObservers = new Vector( 2, 1 );
    }

    protected void initializeEarModels()
    {
        // PLACEMENT IS -3.25" ALONG THE X-AXIS
        LeftEarModel = new EarModel( -7.15e-2d, 0.0e0d, 0.0e0d );
        vecCloakables.addElement( (ICloakable)LeftEarModel );
        // PLACEMENT IS +3.25" ALONG THE X-AXIS
        RightEarModel = new EarModel( +7.15e-2d, 0.0e0d, 0.0e0d );
        vecCloakables.addElement( (ICloakable)RightEarModel );
    }

    /////////////////////////////////////////////////////////////////////////////
    // IModel INTERFACE IMPLEMENTATIONS:
    public void attach( IObserver SubscribingObserver_param )
    {
        vecObservers.addElement( SubscribingObserver_param );
    }

    public void detach( IObserver UnSubscribingObserver_param )
    {
        for( int i = 0; i < vecObservers.size(); i++ )
        {
            if( (IObserver)vecObservers.elementAt( i ) ==
                UnSubscribingObserver_param )
                vecObservers.removeElementAt( i );
        }
    }

    public void updateAllViews()
    {
        for( int i = 0; i < vecObservers.size(); i++ )
            ( (IObserver)vecObservers.elementAt( i ) ).update();
    }

    public String getClassDefName()
    {
        return( "SimulatorModel" );
    }
    // END OF IModel INTERFACE IMPLEMENTATIONS
    /////////////////////////////////////////////////////////////////////////////

    /////////////////////////////////////////////////////////////////////////////
    // ISimulatorModel INTERFACE IMPLEMENTATIONS:

    public IEarModel getLeftEarModel()
    {
        return LeftEarModel;
    }

    public IEarModel getRightEarModel()
    {
        return RightEarModel;
    }

    public IWallModel newWallModel()
    {
        WallModel NewWallModel = new WallModel();

        vecCloakables.addElement( (ICloakable)NewWallModel );
        mySignalQueueDirector.addWall( NewWallModel );

        return ( (IWallModel)NewWallModel );
    }

    public IPeriodicSoundSourceModel newPeriodicSoundSourceModel()
    {
        PeriodicSoundSourceModel NewPeriodicSoundSourceModel = new PeriodicSoundSourceModel();

        vecCloakables.addElement( (ICloakable)NewPeriodicSoundSourceModel );
        mySignalQueueDirector.addPeriodicSoundSource( NewPeriodicSoundSourceModel );

        return ( (IPeriodicSoundSourceModel)NewPeriodicSoundSourceModel );
    }

    public IMovingSoundSourceModel newMovingSoundSourceModel()
    {
        MovingSoundSourceModel NewMovingSoundSourceModel = new MovingSoundSourceModel();

        vecCloakables.addElement( (ICloakable)NewMovingSoundSourceModel );

        if( ( (IMovingSoundSource)NewMovingSoundSourceModel ) == null )
            System.out.println( "In SimulatorModel, NewMovingSoundSourceModel is null." );

        mySignalQueueDirector.addMovingSoundSource( (IMovingSoundSource)NewMovingSoundSourceModel );

        return ( (IMovingSoundSourceModel)NewMovingSoundSourceModel );
    }

    public void setTimeConstraints( double dStartTime_param,
                                    double dFinishTime_param,
                                    double dSamplePeriod_param )
    {
        dStartTime = dStartTime_param;
        dCurrentTime = dStartTime;
        dFinishTime = dFinishTime_param;
        dDuration = ( dFinishTime_param - dStartTime_param );
        dSamplePeriod = dSamplePeriod_param;
        mySignalQueueDirector.setSamplePeriod( dSamplePeriod );
        mySignalQueueDirector.initializeSignalQueues();
    }

    public void cloakAllObjects()
    {
        for( int i = 0; i < vecCloakables.size(); i++ )
        {
            ( ( ICloakable )( vecCloakables.elementAt( i ) ) ).cloak();
        }
    }

    public void runSimulation()
    {
        while( dCurrentTime < dFinishTime )
            advance();
    }

    public void singleStep()
    {
        advance();
    }

    protected void advance()
    {
        mySignalQueueDirector.makeSignals( dCurrentTime );
        mySignalQueueDirector.runSignals( dCurrentTime );
        mySignalQueueDirector.finalizeInstants();

        if( bSavingOutputFile )
        {
            try
            {
                dosOutputFile.writeDouble( LeftEarModel.getResponse() );
                dosOutputFile.writeDouble( RightEarModel.getResponse() );
            }
            catch( IOException ioe )
            {
                System.out.println( ioe.toString() );
            }
        }

        dCurrentTime += dSamplePeriod;
    }

    public void uncloakAllObjects()
    {
        for( int i = 0; i < vecCloakables.size(); i++ )
        {
            ( ( ICloakable )( vecCloakables.elementAt( i ) ) ).uncloak();
        }
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
        mySignalQueueDirector.setTextLog( myTextLog );
    }

    public void openOutputFile( String IN_strFileName )
    {
        if( dosOutputFile != null )
            closeOutputFile();
        try
        {
            dosOutputFile = new DataOutputStream(
                                new BufferedOutputStream(
                                    new FileOutputStream( IN_strFileName )
                                ) );
        }
        catch( IOException ioe )
        {
            System.out.println( ioe.toString() );
        }
        bSavingOutputFile = true;
    }

    public void closeOutputFile()
    {
        try
        {
            dosOutputFile.flush();
            dosOutputFile.close();
        }
        catch( IOException ioe )
        {
            System.out.println( ioe.toString() );
        }
        dosOutputFile = null;
        bSavingOutputFile = false;
    }


    // END OF ISimulatorModel INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

}
```

```
//--------------------------------------------------------------------
//
//        Department of Computer Science, SUNY Institute of Technology
//         Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          SimulatorView
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//--------------------------------------------------------------------
/**
 * The class SimulatorView is the main visual
 * user interface component for an SimulatorModel.
 * It holds display components for SimulatorModel
 * attributes and controls which allow the user
 * to configure the SimulatorModel values.  Also,
 * SimulatorView acts as the designated ITextLogger
 * instance of the simulation.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//--------------------------------------------------------------------

import java.awt.Component;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Menu;
import java.awt.MenuBar;
import java.awt.MenuItem;
import java.awt.TextArea;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import EarViewFrame;
import SimulatorController;
import ISimulatorModel;
import ISimulatorView;
import ITextLogger;
import ITextLogUser;
import SimulatorModel;

public class SimulatorView
    extends Frame
    implements ISimulatorView
{
    protected EarViewFrame LeftEarViewFrame = null;
    protected EarViewFrame RightEarViewFrame = null;
    protected GridBagLayout gridbag = null;
    protected GridBagConstraints gridbagconstraints = null;
    protected MenuItem mItemAbout = null;
    protected SimulatorController mySimulatorController = null;
    protected ISimulatorModel mySimulatorModel = null;
    protected TextArea taOutputMessages = null;

    SimulatorView( ISimulatorModel IN_mySimulatorModel )
    {
        super( "Sound Scape Simulator" );

        mySimulatorModel = IN_mySimulatorModel;
        initialize();
        initializeGUI();
        mySimulatorModel.attach( this );
        mySimulatorModel.setTextLog( (ITextLogger)this );
    }

    //////////////////////////////////////////////////////////////////////
    // ISimulatorView INTERFACE IMPLEMENTATIONS:

    public ISimulatorModel getSimulatorModel()
    {
        return mySimulatorModel;
    }

    public void initialize()
    {
        mySimulatorController = new SimulatorController( 400, 300, mySimulatorModel, this );
        taOutputMessages = new TextArea( "Output Messages:  \n", 10, 80, TextArea.SCROLLBARS_VERTICAL_ONLY );
        LeftEarViewFrame = new EarViewFrame( mySimulatorModel.getLeftEarModel(),
                                             "Left EarModel", (ITextLogger)this );
        RightEarViewFrame = new EarViewFrame( mySimulatorModel.getRightEarModel(),
                                             "Right EarModel", (ITextLogger)this );
    }
    // END OF ISimulatorView INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // INITIALIZATION METHODS:

    protected void addComponent( Component c )
    {
        gridbag.setConstraints( c, gridbagconstraints );
        add( c );
    }

    protected WindowAdapter getWindowAdapter()
    {
        return( new WindowAdapter()
            {
                public void windowClosing( WindowEvent e )
                {
                        System.exit( 0 );
                }
            } );
    }

    public void initializeGUI()
    {
        setLocation( 10, 10 );
        setSize( 400, 500 );
        setMenuBar( initMenuBar() );

        gridbag = new GridBagLayout();
        gridbagconstraints = new GridBagConstraints();
        setLayout( gridbag );

        gridbagconstraints.weightx = 1.0;
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        gridbagconstraints.fill = GridBagConstraints.BOTH;
        gridbagconstraints.gridheight = 12;
        gridbagconstraints.gridheight = 1;
        gridbagconstraints.weighty = 0.0;           //reset to the default

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( mySimulatorController );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( taOutputMessages );

        addWindowListener( getWindowAdapter() );
        setVisible( true );
    }

    protected MenuBar initMenuBar()
    {
        MenuBar myMenuBar = new MenuBar();
        Menu helpMenu = new Menu( "About" );

        mItemAbout = new MenuItem( "Web URL" );
        helpMenu.add( mItemAbout );
        mItemAbout.addActionListener( mySimulatorController );

        myMenuBar.setHelpMenu( helpMenu );
        return( myMenuBar );
    }

    public MenuItem getAboutMenuItem()
    {
        return mItemAbout;
    }
    // END OF INITIALIZATION METHODS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IObserver INTERFACE IMPLEMENTATIONS:

    public void update()
    {
    }
    // END OF IObserver INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:
```

```
    public void logText( String IN_strLogText )
    {
        taOutputMessages.append( IN_strLogText );
    }

    public void setTextLog( ITextLogger IN_ITextLogger )
    {
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////
    // ITextLogger INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return( (ITextLogger)this );
    }

    // END OF ITextLogger INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "SimulatorView" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////

    public static void main( String[] args )
    {
        SimulatorView ssFrame = new SimulatorView( new SimulatorModel() );
    }

}  // end of SimulatorView class
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          SoundIntensityViewCanvas
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class SoundIntensityViewCanvas extends BufferViewCanvas
 * and is a visual component of the EarViewFrame which graphically
 * plots the recent sound intensity response values of the EarModel.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import BufferViewCanvas;


public class SoundIntensityViewCanvas
    extends BufferViewCanvas
{

    public SoundIntensityViewCanvas(  int iDisplayWidth_param,
                                int iDisplayHeight_param,
                                IEarModel EarModel_param )
    {
        super( iDisplayWidth_param, iDisplayHeight_param, EarModel_param );
        super.setResponseBuffer( EarModel_param.getSoundIntensityBuffer() );
    }

    public void update()
    {
        ResponseBuffer = myEarModel.getSoundIntensityBuffer();
        super.update();
    }
}
```

```
//---------------------------------------------------------------------
//
//        Department of Computer Science, SUNY Institute of Technology
//         Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
// Class:         SoundscapeApplet
// Author:        Matt Gentner
// Original Date: January, 1998
//---------------------------------------------------------------------
/**
 * The class SoundscapeApplet extends the JDK's Applet class and is
 * used to deploy the SoundscapeSA simulator over the network.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//---------------------------------------------------------------------

import java.applet.Applet;
import java.awt.Button;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import SimulatorModel;
import SimulatorView;


public class SoundscapeApplet extends Applet
    implements ActionListener
{
    protected Button btnTrySoundscapeSA = null;

    public void init()
    {
        btnTrySoundscapeSA = new Button( "Try SoundscapeSA" );
        btnTrySoundscapeSA.addActionListener( this );
        add( btnTrySoundscapeSA );
        setVisible( true );
    }

    public void actionPerformed( ActionEvent ae )
    {
        if( ae.getSource() == btnTrySoundscapeSA )
        {
            new SimulatorView( new SimulatorModel() );
        }
    }
}
```

```
//----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          SoundSourceSignalQueueMgr
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * THE SoundSourceSignalQueueMgr CLASS OF OBJECTS IS DERIVATIVE OF
 * THE SignalQueueMgr CLASS.  THE SoundSourceSignalQueueMgr IS SPECIALIZED
 * FOR MANAGING THE EFFECTS OF STATIONARY SoundSource ABSTRACTIONS
 * IN THE Soundscape SIMULATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.lang.String;
import java.util.Vector;

import IEar;
import ISoundListener;
import ISoundSource;
import SignalQueueMgr;


public class SoundSourceSignalQueueMgr extends SignalQueueMgr
        implements ISignalQueueMgr
{
    public final double dSpeed_Of_Sound = ISoundListener.dSpeed_Of_Sound;
    protected ISoundSource mySoundSource = null;

    public SoundSourceSignalQueueMgr( IEar IN_LeftEar,
                                IEar IN_RightEar,
                                ISoundSource IN_mySoundSource,
                                Vector IN_vecSharedSoundListeners )
    {
        super( IN_LeftEar,
               IN_RightEar,
               IN_vecSharedSoundListeners );

        mySoundSource = IN_mySoundSource;
    }

    ///////////////////////////////////////////////////////////////////////
    // ISignalQueueMgr INTERFACE IMPLEMENTATIONS:

    public String initializeSignalQueues()
    {
        String strProgressLog = new String( "" );

        return strProgressLog;
    }

    public void makeSignals( double IN_dCurrentTime )
    {
    }

    public void effectSignals( double IN_dCurrentTime )
    {
        if( !mySoundSource.isMuted() )
        {
            double dSignalStrength = 0.0e0d;
            double dSoundSourceTime = 0.0e0d;
            double dAttenuatedIntensity = 0.0e0d;
            double dDistanceToListener = 0.0e0d;
            ISoundListener tempSoundListener = null;

            effectEarSignals( IN_dCurrentTime );

            for( int i = 0; i < vecSharedSoundListeners.size(); i++ )
            {
                tempSoundListener = (ISoundListener)vecSharedSoundListeners.elementAt( i );
                dDistanceToListener = distanceToListener( tempSoundListener, IN_dCurrentTime );
                dSoundSourceTime = ( IN_dCurrentTime - ( dDistanceToListener / dSpeed_Of_Sound ) );
                mySoundSource.makeSound( dSoundSourceTime );
            dAttenuatedIntensity = (    mySoundSource.getAmplitude() /
                                    ( +4.0e0d * Math.PI * dDistanceToListener ) );

            tempSoundListener.rattle( dAttenuatedIntensity );
        }
      }
    }

    public void finalizeInstant()
    {
        // NOTHING TO DO, YET.
    }
    // END OF ISignalQueueMgr INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////

    private void effectEarSignals( double IN_dCurrentTime )
    {
        double dSignalStrength = 0.0e0d;
        double dSoundSourceTime = 0.0e0d;
        double dAttenuatedIntensity = 0.0e0d;
        double dDistanceToListener = 0.0e0d;

        if( !mySoundSource.isMutedToLeftEar() )
        {
            dDistanceToListener = distanceToListener( LeftEar, IN_dCurrentTime );
            dSoundSourceTime = ( IN_dCurrentTime - ( dDistanceToListener / dSpeed_Of_Sound ) );
            mySoundSource.makeSound( dSoundSourceTime );

            dAttenuatedIntensity = (    mySoundSource.getAmplitude() /
                                    ( +4.0e0d * Math.PI * dDistanceToListener ) );

            LeftEar.rattle( dAttenuatedIntensity );
        }

        if( !mySoundSource.isMutedToRightEar() )
        {
            dDistanceToListener = distanceToListener( RightEar, IN_dCurrentTime );
            dSoundSourceTime = ( IN_dCurrentTime - ( dDistanceToListener / dSpeed_Of_Sound ) );
            mySoundSource.makeSound( dSoundSourceTime );

            dAttenuatedIntensity = (    mySoundSource.getAmplitude() /
                                    ( +4.0e0d * Math.PI * dDistanceToListener ) );

            RightEar.rattle( dAttenuatedIntensity );
        }
    }

    private double distanceToListener( ISoundListener IN_SoundListener,
                                       double IN_dListenerCurrentTime )
    {
        double dDistanceToListener = 0.0e0d;

        try
        {
            dDistanceToListener =
                Math.sqrt(
                    Math.pow( ( IN_SoundListener.getXCoord() - mySoundSource.getXCoord() ), +2.0e0d ) +
                    Math.pow( ( IN_SoundListener.getYCoord() - mySoundSource.getYCoord() ), +2.0e0d ) +
                    Math.pow( ( IN_SoundListener.getZCoord() - mySoundSource.getZCoord() ), +2.0e0d )   );

        }  catch ( ArithmeticException e )
        {
                System.out.println( e.toString() );
        }
        return dDistanceToListener;
    }

}
```

```
//------------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          StartTimeSelector
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * The class StartTimeSelector extends the JDK's CheckboxGroup
 * and is used to allow simulator users to specify whether a
 * simulation should prgress from the current time or from
 * time-zero.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//------------------------------------------------------------------------

import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.event.ItemListener;

public class StartTimeSelector extends CheckboxGroup
{
    protected Checkbox cbTimeZero = null;
    protected Checkbox cbCurrentTime = null;
    protected ItemListener myItemListener = null;

    StartTimeSelector( ItemListener ItemListener_param )
    {
        myItemListener = ItemListener_param;

        initializeCheckBoxes();

        setSelectedCheckbox( cbTimeZero );
    }

    protected void initializeCheckBoxes()
    {
        cbTimeZero = new Checkbox( "Time Zero", this, true );
        cbTimeZero.addItemListener( myItemListener );

        cbCurrentTime = new Checkbox( "Current Time", this, false );
        cbCurrentTime.addItemListener( myItemListener );
    }

    public Checkbox getTimeZeroCheckbox()
    {
        return cbTimeZero;
    }

    public Checkbox getCurrentTimeCheckbox()
    {
        return cbCurrentTime;
    }

}
```

```
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          ThreeDimensionalPoint
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * ThreeDimensionalPoint IS A CLASS OF OBJECTS WHICH ARE USED TO
 * SET THE BOUNDS OF SURFACES IN THE Soundscape SIMULATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.lang.ArithmeticException;
import java.lang.Math;

import IThreeDimensionalPoint;


public class ThreeDimensionalPoint
    implements IThreeDimensionalPoint
    // AS SUCH, implements IPositionHolder AND ISoundscapeBase
{
    protected double x_coord = 0.0e0d;
    protected double y_coord = 0.0e0d;
    protected double z_coord = 0.0e0d;

    public ThreeDimensionalPoint( double x_coord_param,
                    double y_coord_param,
                    double z_coord_param )
    {
        x_coord = x_coord_param;
        y_coord = y_coord_param;
        z_coord = z_coord_param;
    }

    ///////////////////////////////////////////////////////////////////////////
    // IPositionHolder INTERFACE IMPLEMENTATIONS:

    public double getXCoord()
    {
        return x_coord;
    }

    public double getYCoord()
    {
        return y_coord;
    }

    public double getZCoord()
    {
        return z_coord;
    }

    public void moveTo( double x, double y, double z )
    {
        x_coord = x;
        y_coord = y;
        z_coord = z;
    }
    // END OF IPositionHolder INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "ThreeDimensionalPoint" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    public void display()
    {
        System.out.println( "ThreeDimensionalPoint at x = " + x_coord +
                            " y = " + y_coord + " z = " + z_coord );
    }

    public double distanceTo( IThreeDimensionalPoint IN_OtherThreeDimensionalPoint )
    {
        double dDistance = -1.0e0d;

        try
        {
            dDistance =
                Math.sqrt(
                    Math.pow( ( x_coord - IN_OtherThreeDimensionalPoint.getXCoord() ), ( +2.0e0d ) ) +
                    Math.pow( ( y_coord - IN_OtherThreeDimensionalPoint.getYCoord() ), ( +2.0e0d ) ) +
                    Math.pow( ( z_coord - IN_OtherThreeDimensionalPoint.getZCoord() ), ( +2.0e0d ) )
                    );

        }  catch ( ArithmeticException e )
        {
                System.out.println( e.toString() );
        }

        return dDistance;
    }

}  // end of ThreeDimensionalPoint class
```

```
//-----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
// Class:          ViewCanvas
// Author:         Matt Gentner
// Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The abstract class ViewCanvas extends the JDK's Canvas class
 * and is meant to provide a skeleton for many visual components
 * which in part comprise the simulator's ViewFrame classes.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.awt.Dimension;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.lang.Throwable;

import IModel;
import IView;


public abstract class ViewCanvas
    extends Canvas
{
    public static final int iFieldLabelIndent = 10;
    public static final int iFieldValueIndent = 100;
    public static final int iTextRowHeight = 15;
    public static final int iFirstTextRowYPos = 20;

    protected int iTextRowYPos = iFirstTextRowYPos;
    protected Color TextColor = null;
    protected Color TitleColor = null;
    protected Color BorderColor = null;
    protected Color RestoreColor = null;
    protected IModel myModel = null;


    public ViewCanvas(  int iDisplayWidth_param,
                        int iDisplayHeight_param,
                        IModel Model_param )
    {
        setSize( iDisplayWidth_param, iDisplayHeight_param );
        myModel = Model_param;

        BorderColor = ( Color.black );
        TextColor = ( Color.blue );
        TitleColor = ( Color.black );
        setVisible( true );
    }

    public void paint( Graphics g )
    {
        cleanCanvas( g );
        drawBorder( g );
        drawHeader( g );
    }

    protected void cleanCanvas( Graphics g )
    {
        RestoreColor = g.getColor();

        g.setColor( getBackground() );
        g.fillRect( 0, 0, getSize().width, getSize().height );

        g.setColor( RestoreColor );
    }

    protected void drawBorder( Graphics g )
    {
        Dimension dimCanvas = getSize();    // USE size() METHOD OF THIS GRAPHICS Component
        int iRightHandEdge = ( dimCanvas.width - 3 );
        int iLowerEdge = ( dimCanvas.height - 3 );
        RestoreColor = g.getColor();
        g.setColor( BorderColor );

        g.drawLine( 3, 3, iRightHandEdge, 3  );
        g.drawLine( iRightHandEdge, 3, iRightHandEdge, iLowerEdge );
        g.drawLine( iRightHandEdge, iLowerEdge, 3, iLowerEdge );
        g.drawLine( 3, iLowerEdge, 3, 3 );
        g.setColor( RestoreColor );
    }

    protected void drawHeader( Graphics g )
    {
        g.drawString( ( myModel.getClassDefName() ) + " Data:  ", iFieldLabelIndent, 20 );
    }

    protected void drawTextRow( Graphics g, String strFieldLabel, String strFieldValue )
    {
        g.drawString( strFieldLabel, iFieldLabelIndent, iTextRowYPos );
        g.drawString( strFieldValue, iFieldValueIndent, iTextRowYPos );
        iTextRowYPos += iTextRowHeight;
    }

    ///////////////////////////////////////////////////////////////////////////
    // IView INTERFACE IMPLEMENTATIONS:

    abstract public void update();

    // END OF IView INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          WallController
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class WallController accepts and handles all
 * user-interface events which come from controls
 * on the WallViewFrame.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Button;
import java.awt.Checkbox;
import java.awt.GridBagConstraints;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.lang.Throwable;
import java.lang.Double;
import java.lang.NumberFormatException;

import Direction;
import DirectionSelectionGroup;
import IModel;
import WallViewPanel;
import IController;
import IPositionHolderModel;
import ITextLogger;
import ITextLogUser;
import IWallModel;
import PositionHolderController;


public class WallController
    extends PositionHolderController
    implements ActionListener, ItemListener, IController, ITextLogUser
{
    protected Button btnChange = null;

    static Label lblWallWidth = new Label( "Wall Width (m):  " );
    static Label lblWallHeight = new Label( "Wall Height (m):  " );
    static Label lblSoundAbsorption = new Label( "SoundAbsorption:  " );
    static Label lblResolution = new Label( "Resolution:  " );

    protected TextField tfWallWidth = null;
    protected TextField tfWallHeight = null;
    protected TextField tfSoundAbsorption = null;
    protected TextField tfResolution = null;

    protected Checkbox CloakingCheckbox = null;
    protected Checkbox MutingCheckbox = null;
    protected Checkbox LeftEarMutingCheckbox = null;
    protected Checkbox RightEarMutingCheckbox = null;

    protected DirectionSelectionGroup myDirectionSelector = null;
    protected ITextLogger myTextLog = null;
    protected IWallModel myWallModel = null;
    protected WallViewPanel myWallView = null;

    public WallController( int width, int height,
                           IWallModel WallModel_param,
                           WallViewPanel WallView_param,
                           ITextLogger IN_TextLogger )
    {
        super( width, height, (IPositionHolderModel)WallModel_param );

        initializeWall( WallModel_param, WallView_param );
        setTextLog( IN_TextLogger );
        initializeWallAggegates();
        initializeWallFrame();

        setSize( width, height );
        setVisible( true );
    }

    protected void initializeWallAggegates()
    {
        btnChange = new Button( "Change" );
        btnChange.addActionListener( this );

        CloakingCheckbox = new Checkbox( "Cloak" );
        CloakingCheckbox.addItemListener( this );
        CloakingCheckbox.setState( myWallModel.isCloaked() );

        MutingCheckbox = new Checkbox( "Mute" );
        MutingCheckbox.addItemListener( this );
        MutingCheckbox.setState( myWallModel.isMuted() );

        LeftEarMutingCheckbox = new Checkbox( "Mute to Left Ear" );
        LeftEarMutingCheckbox.addItemListener( this );
        LeftEarMutingCheckbox.setState( myWallModel.isMutedToLeftEar() );

        RightEarMutingCheckbox = new Checkbox( "Mute to Right Ear" );
        RightEarMutingCheckbox.addItemListener( this );
        RightEarMutingCheckbox.setState( myWallModel.isMutedToRightEar() );

        tfWallWidth = new TextField( 10 );
        tfWallHeight = new TextField( 10 );
        tfSoundAbsorption = new TextField( 10 );
        tfResolution = new TextField( 10 );

        myDirectionSelector = new DirectionSelectionGroup( this );
        updateDirectionSelector();
    }


    protected void initializeWallFrame()
    {
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblWallWidth );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfWallWidth );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblWallHeight );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfWallHeight );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblSoundAbsorption );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfSoundAbsorption );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( lblResolution );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( tfResolution );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( CloakingCheckbox );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnChange );

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( MutingCheckbox );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( LeftEarMutingCheckbox );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( RightEarMutingCheckbox );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( myDirectionSelector.getNorthCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( myDirectionSelector.getEastCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( myDirectionSelector.getUpCheckbox() );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( myDirectionSelector.getSouthCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( myDirectionSelector.getWestCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( myDirectionSelector.getDownCheckbox() );
    }

    public double getWallWidthTextFieldValue()
    {
        Double dblWallWidthTemp= new Double( myWallModel.getWallWidth() );
        try
        {
            if ( ( tfWallWidth.getText() ).length() > 0 )
                dblWallWidthTemp = Double.valueOf( tfWallWidth.getText() );

        } catch( NumberFormatException nfe )
        {
            System.out.println( nfe.toString() );
        }

        return( dblWallWidthTemp.doubleValue() );
    }

    public double getWallHeightTextFieldValue()
    {
        Double dblWallHeightTemp= new Double( myWallModel.getWallHeight() );
        try
        {
            if ( ( tfWallHeight.getText() ).length() > 0 )
                dblWallHeightTemp = Double.valueOf( tfWallHeight.getText() );

        } catch( NumberFormatException nfe )
        {
            System.out.println( nfe.toString() );
        }

        return( dblWallHeightTemp.doubleValue() );
    }

    public double getSoundAbsorptionTextFieldValue()
    {
        Double dblSoundAbsorptionTemp= new Double( myWallModel.getSoundAbsorption() );
        try
        {
            if ( ( tfSoundAbsorption.getText() ).length() > 0 )
                dblSoundAbsorptionTemp = Double.valueOf( tfSoundAbsorption.getText() );

        } catch( NumberFormatException nfe )
        {
            System.out.println( nfe.toString() );
        }

        return( dblSoundAbsorptionTemp.doubleValue() );
    }

    public double getResolutionTextFieldValue()
    {
        Double dblResolutionTemp= new Double( myWallModel.getResolution() );
        try
        {
            if ( ( tfResolution.getText() ).length() > 0 )
                dblResolutionTemp = Double.valueOf( tfResolution.getText() );

        } catch( NumberFormatException nfe )
        {
            System.out.println( nfe.toString() );
        }

        return( dblResolutionTemp.doubleValue() );
    }

    ///////////////////////////////////////////////////////////////////////////
    // IController INTERFACE IMPLEMENTATIONS:
    public void initializeWall(  IModel WallModel_param,
                                 WallViewPanel WallView_param )
    {
        myWallModel = ( (IWallModel)WallModel_param );
        myWallView = ( (WallViewPanel)WallView_param );
    }

    public void update()
    {
        super.update();

        CloakingCheckbox.setState( myWallModel.isCloaked() );
        MutingCheckbox.setState( myWallModel.isMuted() );
        LeftEarMutingCheckbox.setState( myWallModel.isMutedToLeftEar() );
        RightEarMutingCheckbox.setState( myWallModel.isMutedToRightEar() );
        updateDirectionSelector();
    }
    // END OF IController INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////

    protected void updateDirectionSelector()
    {
        String strCurrentDirectionName = ( myWallModel.getFacingDirection() ).toString();
        if( strCurrentDirectionName.equals( "NORTH" ) )
        {
            ( myDirectionSelector.getNorthCheckbox() ).setState( true );
        }
        else if( strCurrentDirectionName.equals( "SOUTH" ) )
        {
            ( myDirectionSelector.getSouthCheckbox() ).setState( true );
        }
        else if( strCurrentDirectionName.equals( "EAST" ) )
        {
            ( myDirectionSelector.getEastCheckbox() ).setState( true );
        }
        else if( strCurrentDirectionName.equals( "WEST" ) )
        {
            ( myDirectionSelector.getWestCheckbox() ).setState( true );
        }
        else if( strCurrentDirectionName.equals( "UP" ) )
        {
            ( myDirectionSelector.getUpCheckbox() ).setState( true );
        }
        else if( strCurrentDirectionName.equals( "DOWN" ) )
        {
            ( myDirectionSelector.getDownCheckbox() ).setState( true );
        }
    }

    ////////////////////////////////////////////////////////////////////////////
    // ActionListener INTERFACE IMPLEMENTATIONS:
    public void actionPerformed( ActionEvent ae )
    {
        super.actionPerformed( ae );

        if( ae.getSource() == btnChange )
        {
            myWallModel.setWallWidth( getWallWidthTextFieldValue() );
            myWallModel.setWallHeight( getWallHeightTextFieldValue() );
            myWallModel.setSoundAbsorption( getSoundAbsorptionTextFieldValue() );
            myWallModel.setResolution( getResolutionTextFieldValue() );

            myWallModel.reinitialize();
        }
        else if ( ae.getActionCommand() == "Brighten" )
        {
            myWallView.brighten();
            myWallModel.displayAvgResponse();
        }
        else if ( ae.getActionCommand() == "Darken" )
        {
            myWallView.darken();
            myWallModel.displayAvgResponse();
        }
        else if ( ae.getActionCommand() == "Fine Brighten" )
        {
            myWallView.fineBrighten();
            myWallModel.displayAvgResponse();
        }
        else if ( ae.getActionCommand() == "Fine Darken" )
        {
            myWallView.fineDarken();
            myWallModel.displayAvgResponse();
        }

    }
    // END OF ActionListener INTERFACE IMPLEMENTATIONS
    ////////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////////////////////////////////
    // ItemListener INTERFACE IMPLEMENTATIONS:
    public void itemStateChanged( ItemEvent ie )
    {
        boolean bDirectionChanged = false;
        if( ie.getSource() == CloakingCheckbox )
        {
            if( CloakingCheckbox.getState() )
                myWallModel.cloak();
            else
                myWallModel.uncloak();
        }
        else if( ie.getSource() == MutingCheckbox )
        {
            myWallModel.setMuted( MutingCheckbox.getState() );
        }
        else if( ie.getSource() == LeftEarMutingCheckbox )
        {
            myWallModel.setMutedToLeftEar( LeftEarMutingCheckbox.getState() );
        }
        else if( ie.getSource() == RightEarMutingCheckbox )
        {
            myWallModel.setMutedToRightEar( RightEarMutingCheckbox.getState() );
        }
        else if( ie.getSource() == myDirectionSelector.getNorthCheckbox() )
        {
            if( ( myDirectionSelector.getNorthCheckbox() ).getState() )
            {
                myWallModel.setFacingDirection( Direction.NORTH );
                bDirectionChanged = true;
            }
        }
        else if( ie.getSource() == myDirectionSelector.getSouthCheckbox() )
        {
            if( ( myDirectionSelector.getSouthCheckbox() ).getState() )
            {
                myWallModel.setFacingDirection( Direction.SOUTH );
                bDirectionChanged = true;
            }
        }
        else if( ie.getSource() == myDirectionSelector.getEastCheckbox() )
        {
            if( ( myDirectionSelector.getEastCheckbox() ).getState() )
            {
                myWallModel.setFacingDirection( Direction.EAST );
                bDirectionChanged = true;
            }
        }
        else if( ie.getSource() == myDirectionSelector.getWestCheckbox() )
        {
            if( ( myDirectionSelector.getWestCheckbox() ).getState() )
            {
                myWallModel.setFacingDirection( Direction.WEST );
                bDirectionChanged = true;
            }
        }
        else if( ie.getSource() == myDirectionSelector.getUpCheckbox() )
        {
            if( ( myDirectionSelector.getUpCheckbox() ).getState() )
            {
                myWallModel.setFacingDirection( Direction.UP );
                bDirectionChanged = true;
            }
        }
        else if( ie.getSource() == myDirectionSelector.getDownCheckbox() )
        {
            if( ( myDirectionSelector.getDownCheckbox() ).getState() )
            {
                myWallModel.setFacingDirection( Direction.DOWN );
                bDirectionChanged = true;
            }
        }

        if( bDirectionChanged )
            myWallModel.reinitialize();

    }
    // END OF ItemListener INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          WallModel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 *
 * THE WallModel IS AN ABSTRACTION OF AN ECHOING, PLANAR AND
 * RECTANGULAR SURFACE.  IT IS ESSENTIALLY A COLLECTION OF
 * EchoPoints.  THE WallModel IS BOUNDED BY FOUR
 * ThreeDimensionalPoint AGGREGATES, IT'S FACING ELEVATION AND
 * FACING AZIMUTH ARE ENCAPSULATED IN IT'S pMyFacingDirection
 * ATTRIBUTE.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.util.Vector;

import Direction;
import EchoPoint;
import IPositionHolderModel;
import ISoundListener;
import IEchoPoint;
import ITextLogger;
import ITextLogUser;
import ThreeDimensionalPoint;


public class WallModel
    implements IWallModel, ITextLogUser
    // AS SUCH, extends IModel, IMutable, ICloakable,
    // IPositionHolder, IPositionHolderModel AND IWall
{
    // INSTANCE VARIABLES:
    protected boolean bCloaked = false;
    protected boolean bMuted = false;
    protected boolean bMutedToLeftEar = false;
    protected boolean bMutedToRightEar = false;
    protected double dWallWidth = 0.0e0d;
    protected double dWallHeight = 0.0e0d;
    protected double dResolution = 0.0e0d;
    protected double dSoundAbsorption = 0.0e0d;
    protected int iNumEchoPointsWide = 0;
    protected int iNumEchoPointsHigh = 0;

    protected Direction myFacingDirection = null;
    protected ITextLogger myTextLog = null;
    protected ThreeDimensionalPoint tdpUpperLeftHandCorner = null;
    protected ThreeDimensionalPoint tdpUpperRightHandCorner = null;
    protected ThreeDimensionalPoint tdpLowerLeftHandCorner = null;
    protected ThreeDimensionalPoint tdpLowerRightHandCorner = null;
    protected Vector vecObservers = null;
    protected Vector vecEchoPoints = null;


    // DEFAULT CONSTRUCTOR:
    public WallModel()
    {
        // THE DEFAULT WallModel IS LIKE A FLOOR TILE, IN FRONT
        // BELOW AND TO THE LEFT OF THE ORIGIN. IT IS 1cm * 1cm.
        // IT'S RESOLUTION IS 1mm, AND SOUND ABSORBTION IS 25%.
        this( ( new ThreeDimensionalPoint( -1.5e-2d, +5.0e-3d, -7.15e-2d ) ),
                Direction.UP,
                ( +1.0e-2d ),
                ( +1.0e-2d ),
                ( +1.0e-3d ),
                ( +2.5e-1d ) );
    }

    // PARAMETERIZED CONSTRUCTOR:
    public WallModel( ThreeDimensionalPoint tdpUpperLeftHandCorner_param,
                      Direction FacingDirection_param,
                      double dWallWidth_param,
                    double dWallHeight_param,
                    double dResolution_param,
                    double dSoundAbsorption_param )
{
    myFacingDirection = FacingDirection_param;
    dWallWidth = dWallWidth_param;
    dWallHeight = dWallHeight_param;
    dResolution = dResolution_param;
    dSoundAbsorption = dSoundAbsorption_param;

    tdpUpperLeftHandCorner = tdpUpperLeftHandCorner_param;

    // INITIALIZE THE OTHER THREE CORNERS, BASED ON THE
    // UPPER-LEFTHAND CORNER, FACING DIRECTION,
    // WIDTH AND HEIGHT.
    deriveCornerPoints();

    initializeEchoPoints();
    vecObservers = new Vector( 3, 3 );
}

///////////////////////////////////////////////////////////////////////////
// IMutable INTERFACE IMPLEMENTATIONS:

public boolean isMuted()
{
    return bMuted;
}

public void setMuted( boolean bMuted_param )
{
    bMuted = bMuted_param;

    for( int i = 0; i < vecEchoPoints.size(); i++ )
    {
        ( (IEchoPoint)vecEchoPoints.elementAt( i ) ).setMuted( bMuted );
    }

    if( bMuted )
        myTextLog.logText( getClassDefName() + " has been muted. \n" );
    else
        myTextLog.logText( getClassDefName() + " has been un-muted. \n" );
}

public boolean isMutedToLeftEar()
{
    return bMutedToLeftEar;
}

public void setMutedToLeftEar( boolean IN_bMutedToLeftEar )
{
    bMutedToLeftEar = IN_bMutedToLeftEar;

    for( int i = 0; i < vecEchoPoints.size(); i++ )
    {
        ( (IEchoPoint)vecEchoPoints.elementAt( i ) ).setMutedToLeftEar( bMutedToLeftEar );
    }

    if( bMutedToLeftEar )
        myTextLog.logText( getClassDefName() + " has been muted to the left ear. \n" );
    else
        myTextLog.logText( getClassDefName() + " has been un-muted to the left ear. \n" );
}

public boolean isMutedToRightEar()
{
    return bMutedToRightEar;
}

public void setMutedToRightEar( boolean IN_bMutedToRightEar )
{
    bMutedToRightEar = IN_bMutedToRightEar;

    for( int i = 0; i < vecEchoPoints.size(); i++ )
    {
        ( (IEchoPoint)vecEchoPoints.elementAt( i ) ).setMutedToRightEar( bMutedToRightEar );
    }

    if( bMutedToRightEar )
        myTextLog.logText( getClassDefName() + " has been muted to the right ear. \n" );
    else
        myTextLog.logText( getClassDefName() + " has been un-muted to the right ear. \n" );
}
// END OF IMutable INTERFACE IMPLEMENTATIONS
///////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
// ICloakable INTERFACE IMPLEMENTATIONS:

public boolean isCloaked()
{
    return bCloaked;
}

public void cloak()
{
    bCloaked = true;
    myTextLog.logText( "Wall Model has been cloaked. \n" );
}

public void uncloak()
{
    bCloaked = false;
    myTextLog.logText( "Wall Model has been un-cloaked. \n" );
}
// END OF ICloakable INTERFACE IMPLEMENTATIONS
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// IModel INTERFACE IMPLEMENTATIONS:

public void attach( IObserver SubscribingObserver_param )
{
    vecObservers.addElement( SubscribingObserver_param );
    // setTextLog( ( (ITextLogUser)SubscribingObserver_param ).getTextLog() );
    // myTextLog.logText(  "In WallModel, attached observer. \n" );
}

public void detach( IObserver UnSubscribingObserver_param )
{
    for( int i = 0; i < vecObservers.size(); i++ )
    {
        if( (IObserver)vecObservers.elementAt( i ) == UnSubscribingObserver_param )
            vecObservers.removeElementAt( i );
    }
}

public void updateAllViews()
{
    for( int i = 0; i < vecObservers.size(); i++ )
        ( (IObserver)vecObservers.elementAt( i ) ).update();
}

public String getClassDefName()
{
    return( "WallModel" );
}
// END OF IModel INTERFACE IMPLEMENTATIONS
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// IPositionHolder INTERFACE IMPLEMENTATIONS:

public void moveTo( double x, double y, double z )
{
    tdpUpperLeftHandCorner.moveTo( x, y, z );
    reinitialize();
    myTextLog.logText( "Wall Model's UpperLeftHandCorner has moved to:" );
    tdpUpperLeftHandCorner.display();
}

public double getXCoord()
{
    return tdpUpperLeftHandCorner.getXCoord();
}

public double getYCoord()
{
    return tdpUpperLeftHandCorner.getYCoord();
}

public double getZCoord()
{
    return tdpUpperLeftHandCorner.getZCoord();
}
// END OF IPositionHolder INTERFACE IMPLEMENTATIONS
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
// IWall INTERFACE IMPLEMENTATIONS:
```

```
public void displayAvgResponse()
{
    double dSum = 0.0e0d;
    double dAvg = 0.0e0d;
    int i, j;

    for( j = 0; j < iNumEchoPointsHigh; j++ )
    {
        for( i = 0; i < iNumEchoPointsWide; i++ )
        {
            dSum += ( (IEchoPoint)vecEchoPoints.elementAt( j * i + i ) ).getResponse();
        }
    }

    dAvg = ( dSum / (double)( vecEchoPoints.size() ) );
    myTextLog.logText( "Average response of EchoPoints is:  " + dAvg + "\n" );
}

public IEchoPoint getEchoPointByIndex( int IN_iEchoPointIndex )
{
    return( (IEchoPoint)vecEchoPoints.elementAt( IN_iEchoPointIndex ) );
}

public double getResponseByIndex( int IN_iEchoPointIndex )
{
    return( (IEchoPoint)vecEchoPoints.elementAt( IN_iEchoPointIndex ) ).getResponse();
}

public double getResolution()
{
    return dResolution;
}

public double getSoundAbsorption()
{
    return dSoundAbsorption;
}

public double getWallHeight()
{
    return dWallHeight;
}

public double getWallWidth()
{
    return dWallWidth;
}

public void reinitialize()
{
    deriveCornerPoints();
    updateAllViews();
    System.gc();
}

public void setFacingDirection( Direction Direction_param )
{
    myFacingDirection = Direction_param;
}

public Direction getFacingDirection()
{
    return myFacingDirection;
}

public void setResolution( double IN_dResolution )
{
    dResolution = IN_dResolution;
}

public void setSoundAbsorption( double dSoundAbsorption_param )
{
    dSoundAbsorption = dSoundAbsorption_param;

    for( int i = 0; i < vecEchoPoints.size(); i++ )
    {
        ( (IEchoPoint)vecEchoPoints.elementAt( i ) ).setSoundAbsorption( dSoundAbsorption );
    }
}

public void setWallWidth( double IN_dWallWidth )
{
    dWallWidth = IN_dWallWidth;
}
```

```
public void setWallHeight( double IN_dWallHeight )
{
    dWallHeight = IN_dWallHeight;
}

public int getNumEchoPoints()
{
    return vecEchoPoints.size();
}

public int getNumEchoPointsWide()
{
    return iNumEchoPointsWide;
}

public int getNumEchoPointsHigh()
{
    return iNumEchoPointsHigh;
}

public IThreeDimensionalPoint getUpperLeftHandCorner()
{
    return tdpUpperLeftHandCorner;
}

public IThreeDimensionalPoint getUpperRightHandCorner()
{
    return tdpUpperRightHandCorner;
}

public IThreeDimensionalPoint getLowerLeftHandCorner()
{
    return tdpLowerLeftHandCorner;
}

public IThreeDimensionalPoint getLowerRightHandCorner()
{
    return tdpLowerRightHandCorner;
}

// END OF IWall INTERFACE IMPLEMENTATIONS
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
// IWallModel INTERFACE IMPLEMENTATIONS:

public void finalizeInstant()
{
    if( !bCloaked )
    {
        for( int i = 0; i < vecEchoPoints.size(); i++ )
            ( (IEchoPoint)vecEchoPoints.elementAt( i ) ).finalizeInstant();

        updateAllViews();
    }
}
// END OF IWallModel INTERFACE IMPLEMENTATIONS
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
// ISoundListener INTERFACE IMPLEMENTATIONS:

public void dampResponse()
{
    for( int i = 0; i < vecEchoPoints.size(); i++ )
    {
        ( (IEchoPoint)vecEchoPoints.elementAt( i ) ).dampResponse();
    }
}

// END OF ISoundListener INTERFACE IMPLEMENTATIONS
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
// PROTECTED MODIFIER METHODS:
protected void initializeNumEchoPoints()
{
    double dCenteringDrift = ( dResolution / +2.0e0d );

    // CALCULATE SIGNED, RAW NUMBER OF EchoPoints IN WIDTH
    iNumEchoPointsWide = (int)Math.floor( dWallWidth / dResolution );
    iNumEchoPointsHigh = (int)Math.floor( dWallHeight / dResolution );

    // KNOCK OF THE LAST EchoPoint, IF IT DRIFTS BEYOND THE WALL'S BOUNDARY
```

```
        if( ( dResolution * (double)iNumEchoPointsWide + dCenteringDrift ) >= dWallWidth )
            iNumEchoPointsWide--;

        if( ( dResolution * (double)iNumEchoPointsHigh + dCenteringDrift ) >= dWallHeight )
            iNumEchoPointsHigh--;

        vecEchoPoints = null;
        System.gc();
        vecEchoPoints = new Vector( ( iNumEchoPointsHigh * iNumEchoPointsWide ), iNumEchoPointsWide );
    }

    protected void initializeEchoPoints()
    {
        int i, j;
        double dCenteringDrift = ( dResolution / +2.0e0d );
        double dEchoPointArea = Math.pow( dResolution, +2.0e0d );

        initializeNumEchoPoints();

        if( ( myFacingDirection == Direction.NORTH ) || ( myFacingDirection == Direction.SOUTH ) )
        {
            double x_coord_offset = 0.0e0d;
            double z_coord_offset = 0.0e0d;
            double dX_CenteringDrift = 0.0e0d;
            double dZ_CenteringDrift = 0.0e0d;
            double dTotal_X_Displacement = ( tdpLowerRightHandCorner.getXCoord() -
                                             tdpUpperLeftHandCorner.getXCoord() );
            double dTotal_Z_Displacement = ( tdpLowerRightHandCorner.getZCoord() -
                                             tdpUpperLeftHandCorner.getZCoord() );
            double dX_Displacement_PerEchoPoint = ( dTotal_X_Displacement /
                                                    ( (double)iNumEchoPointsWide ) );
            double dZ_Displacement_PerEchoPoint = ( dTotal_Z_Displacement /
                                                    ( (double)iNumEchoPointsHigh ) );

            // SIGN THE CENTERING DRIFTS, BASED ON FACING DIRECTION
            if( myFacingDirection == Direction.NORTH )
                dX_CenteringDrift = dCenteringDrift;
            else
                dX_CenteringDrift = ( dCenteringDrift * -1.0e0d );

            dZ_CenteringDrift = ( dCenteringDrift * -1.0e0d );

            for( j = 0; j < iNumEchoPointsHigh; j++ )
            {
                z_coord_offset = ( dZ_Displacement_PerEchoPoint * ( (double)j ) );
                z_coord_offset += dZ_CenteringDrift;

                for( i = 0; i < iNumEchoPointsWide; i++ )
                {
                    x_coord_offset = ( dX_Displacement_PerEchoPoint * ( (double)i ) );
                    x_coord_offset += dX_CenteringDrift;

                    vecEchoPoints.addElement( new EchoPoint(
                                                  ( tdpUpperLeftHandCorner.getXCoord() + x_coord_offset ),
                                                  ( tdpUpperLeftHandCorner.getYCoord() ),
                                                  ( tdpUpperLeftHandCorner.getZCoord() + z_coord_offset ),
                                                    dEchoPointArea,
                                                    dSoundAbsorption ) );
                }
            }
        }
        else if( ( myFacingDirection == Direction.EAST ) || ( myFacingDirection == Direction.WEST ) )
        {
            double z_coord_offset = 0.0e0d;
            double y_coord_offset = 0.0e0d;
            double dY_CenteringDrift = 0.0e0d;
            double dZ_CenteringDrift = 0.0e0d;
            double dTotal_Y_Displacement = ( tdpLowerRightHandCorner.getYCoord() -
                                             tdpUpperLeftHandCorner.getYCoord() );
            double dTotal_Z_Displacement = ( tdpLowerRightHandCorner.getZCoord() -
                                             tdpUpperLeftHandCorner.getZCoord() );
            double dY_Displacement_PerEchoPoint = ( dTotal_Y_Displacement /
                                                    ( (double)iNumEchoPointsWide ) );
            double dZ_Displacement_PerEchoPoint = ( dTotal_Z_Displacement /
                                                    ( (double)iNumEchoPointsHigh ) );

            // SIGN THE CENTERING DRIFTS, BASED ON FACING DIRECTION
            if( myFacingDirection == Direction.WEST )
                dY_CenteringDrift = dCenteringDrift;
            else
                dY_CenteringDrift = ( dCenteringDrift * -1.0e0d );

            dZ_CenteringDrift = ( dCenteringDrift * -1.0e0d );

            for( j = 0; j < iNumEchoPointsHigh; j++ )
        {
            z_coord_offset = ( dZ_Displacement_PerEchoPoint * ( (double)j ) );
            z_coord_offset += dZ_CenteringDrift;

            for( i = 0; i < iNumEchoPointsWide; i++ )
            {
                y_coord_offset = ( dY_Displacement_PerEchoPoint * ( (double)i ) );
                y_coord_offset += dY_CenteringDrift;

                vecEchoPoints.addElement( new EchoPoint(
                                                  ( tdpUpperLeftHandCorner.getXCoord() ),
                                                  ( tdpUpperLeftHandCorner.getYCoord() + y_coord_offset ),
                                                  (    tdpUpperLeftHandCorner.getZCoord() + z_coord_offset ),
                                                    dEchoPointArea,
                                                    dSoundAbsorption ) );
            }
        }
    }
    else if( ( myFacingDirection == Direction.UP ) || ( myFacingDirection == Direction.DOWN ) )
    {
        double x_coord_offset = 0.0e0d;
        double y_coord_offset = 0.0e0d;
        double dX_CenteringDrift = 0.0e0d;
        double dY_CenteringDrift = 0.0e0d;
        double dTotal_X_Displacement = ( tdpLowerRightHandCorner.getXCoord() -
                                         tdpUpperLeftHandCorner.getXCoord() );
        double dTotal_Y_Displacement = ( tdpLowerRightHandCorner.getYCoord() -
                                         tdpUpperLeftHandCorner.getYCoord() );
        double dX_Displacement_PerEchoPoint = ( dTotal_X_Displacement / ( (double)iNumEchoPointsWide ) );
        double dY_Displacement_PerEchoPoint = ( dTotal_Y_Displacement / ( (double)iNumEchoPointsHigh ) );

        // SIGN THE CENTERING DRIFTS, BASED ON FACING DIRECTION
        if( myFacingDirection == Direction.UP )
            dY_CenteringDrift = dCenteringDrift;
        else
            dY_CenteringDrift = ( dCenteringDrift * -1.0e0d );

        dX_CenteringDrift = dCenteringDrift;


        for( j = 0; j < iNumEchoPointsHigh; j++ )
        {
            y_coord_offset = ( dY_Displacement_PerEchoPoint * ( (double)j ) );
            y_coord_offset += dY_CenteringDrift;

            for( i = 0; i < iNumEchoPointsWide; i++ )
            {
                x_coord_offset = ( dX_Displacement_PerEchoPoint * ( (double)i ) );
                x_coord_offset += dX_CenteringDrift;

                vecEchoPoints.addElement( new EchoPoint(
                                                  ( tdpUpperLeftHandCorner.getXCoord() + x_coord_offset ),
                                                  ( tdpUpperLeftHandCorner.getYCoord() + y_coord_offset ),
                                                    tdpUpperLeftHandCorner.getZCoord(),
                                                    dEchoPointArea,
                                                    dSoundAbsorption ) );
            }
        }
    }

    else
        myTextLog.logText( "Wall Model failed to initialize," +
                           " walls MUST face North, South, East, West, Up or Down" );

    /*
    myTextLog.logText( "iNumEchoPointsWide = " + iNumEchoPointsWide +
                       ", iNumEchoPointsHigh = " + iNumEchoPointsHigh );
    myTextLog.logText( "EchoPoints[0][0] is positioned at:  \n" );
    EchoPoints[0][0].displayCoords();
    myTextLog.logText( "EchoPoints[ ( iNumEchoPointsWide - 1 ) ][0] is positioned at:  \n" );
    EchoPoints[ ( iNumEchoPointsWide - 1 ) ][0].displayCoords();
    myTextLog.logText( "EchoPoints[ ( iNumEchoPointsWide - 1 ) ][ ( iNumEchoPointsHigh - 1 ) ] " +
                       " is positioned at:  \n" );
    EchoPoints[ ( iNumEchoPointsWide - 1 ) ][ ( iNumEchoPointsHigh - 1 ) ].displayCoords();
    myTextLog.logText( "EchoPoints[0][ ( iNumEchoPointsHigh - 1 ) ] is positioned at:  \n" );
    EchoPoints[0][ ( iNumEchoPointsHigh - 1 ) ].displayCoords();
    myTextLog.logText( " \n" );
    */

}

protected void deriveCornerPoints()
{
    double x_coord_offset = 0.0e0d;
    double y_coord_offset = 0.0e0d;
double z_coord_offset = 0.0e0d;

// SIGN THE COORDINATE OFFSETS
if( myFacingDirection == Direction.NORTH )
{
    x_coord_offset = dWallWidth;
    z_coord_offset = ( dWallHeight * -1.0e0d );
    /*
    myTextLog.logText( "In WallModel::deriveCornerPoints, Facing Direction was found to be NORTH."  );
    myTextLog.logText( "x_coord_offset = " + x_coord_offset + " z_coord_offset = " + z_coord_offset  );
    */
}
else if( myFacingDirection == Direction.SOUTH )
{
    x_coord_offset = ( dWallWidth * -1.0e0d );
    z_coord_offset = ( dWallHeight * -1.0e0d );
}
else if( myFacingDirection == Direction.EAST )
{
    y_coord_offset = ( dWallWidth * -1.0e0d );
    z_coord_offset = ( dWallHeight * -1.0e0d );
}
else if( myFacingDirection == Direction.WEST )
{
    y_coord_offset = dWallWidth;
    z_coord_offset = ( dWallHeight * -1.0e0d );
}
else if( myFacingDirection == Direction.UP )
{
    x_coord_offset = dWallWidth;
    y_coord_offset = dWallHeight;
}
else if( myFacingDirection == Direction.DOWN )
{
    x_coord_offset = dWallWidth;
    y_coord_offset = ( dWallHeight * -1.0e0d );
}

// DERIVE AND CREATE THE OTHER THREE CORNER POINTS, USING THE ABOVE SIGNED OFFSETS
if( ( myFacingDirection == Direction.NORTH )  || ( myFacingDirection == Direction.SOUTH ) )
{
    tdpUpperRightHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() + x_coord_offset ),
                                   ( tdpUpperLeftHandCorner.getYCoord() ),
                                   ( tdpUpperLeftHandCorner.getZCoord() ) );
    tdpLowerLeftHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() ),
                                   ( tdpUpperLeftHandCorner.getYCoord() ),
                                   ( tdpUpperLeftHandCorner.getZCoord() + z_coord_offset ) );
    tdpLowerRightHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() + x_coord_offset ),
                                   ( tdpUpperLeftHandCorner.getYCoord() ),
                                   ( tdpUpperLeftHandCorner.getZCoord() + z_coord_offset ) );
}
else if( ( myFacingDirection == Direction.EAST )  || ( myFacingDirection == Direction.WEST ) )
{
    tdpUpperRightHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() ),
                                   ( tdpUpperLeftHandCorner.getYCoord() + y_coord_offset ),
                                   ( tdpUpperLeftHandCorner.getZCoord() ) );
    tdpLowerLeftHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() ),
                                   ( tdpUpperLeftHandCorner.getYCoord() ),
                                   ( tdpUpperLeftHandCorner.getZCoord() + z_coord_offset ) );
    tdpLowerRightHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() ),
                                   ( tdpUpperLeftHandCorner.getYCoord() + y_coord_offset ),
                                   ( tdpUpperLeftHandCorner.getZCoord() + z_coord_offset ) );
}
else if( ( myFacingDirection == Direction.UP )  || ( myFacingDirection == Direction.DOWN ) )
{
    tdpUpperRightHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() + x_coord_offset ),
                                   ( tdpUpperLeftHandCorner.getYCoord() ),
                                   ( tdpUpperLeftHandCorner.getZCoord() ) );
    tdpLowerLeftHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() ),
                                   ( tdpUpperLeftHandCorner.getYCoord() + y_coord_offset ),
                                   ( tdpUpperLeftHandCorner.getZCoord() ) );
    tdpLowerRightHandCorner =
        new ThreeDimensionalPoint( ( tdpUpperLeftHandCorner.getXCoord() + x_coord_offset ),
                                   ( tdpUpperLeftHandCorner.getYCoord() + y_coord_offset ),
                                   ( tdpUpperLeftHandCorner.getZCoord() ) );
}
else
```

```
            myTextLog.logText( "Wall Model failed to initialize,"
                               + " walls MUST face North, South, East, West, Up or Down" );

        /*
        myTextLog.logText( "In WallModel::deriveCornerPoints, the UpperLeftHandCorner is at:  \n" );
        tdpUpperLeftHandCorner.display();
        myTextLog.logText( "In WallModel::deriveCornerPoints, the UpperRightHandCorner is at:  \n" );
        tdpUpperRightHandCorner.display();
        myTextLog.logText( "In WallModel::deriveCornerPoints, the LowerLeftHandCorner is at:  \n" );
        tdpLowerLeftHandCorner.display();
        myTextLog.logText( "In WallModel::deriveCornerPoints, the LowerRightHandCorner is at:  \n" );
        tdpLowerRightHandCorner.display();
        */
    }
    // END OF PROTECTED MODIFIER METHODS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

} // END OF WallModel CLASS DEFINITION
```

```
//-----------------------------------------------------------------------
//
//        Department of Computer Science, SUNY Institute of Technology
//         Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          WallResponsesViewCanvas
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-----------------------------------------------------------------------
/**
 * The class WallResponsesViewCanvas is a visual componenet
 * of the WallViewFrame which graphically displays the
 * runtime responses of a WallModel's EchoPoint aggregates.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-----------------------------------------------------------------------

import java.awt.Dimension;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.ScrollPane;

import IWallModel;


public class WallResponsesViewCanvas
    extends ScrollPane
{
    protected double dBrightnessFactor = +1.0e0d;
    protected int iDisplayWidth = 0;
    protected int iDisplayHeight = 0;
    protected int iArrayWidth = 0;
    protected int iArrayHeight = 0;
    protected int iZoomFactor = 0;

    protected Canvas myDrawingCanvas = null;
    protected IWallModel myWallModel = null;
    protected double[][] myResponsesArray;

    public WallResponsesViewCanvas(  int iDisplayWidth_param,
                                     int iDisplayHeight_param,
                                     IWallModel WallModel_param )
    {
        super( ScrollPane.SCROLLBARS_AS_NEEDED );

        iDisplayWidth = iDisplayWidth_param;
        iDisplayHeight = iDisplayHeight_param;
        myWallModel = WallModel_param;

        updateResponsesArray();
        initializeGUI();
    }

    protected void updateResponsesArray()
    {
        int i, j;
        int iNumEchoPoints = myWallModel.getNumEchoPoints();
        iArrayHeight = myWallModel.getNumEchoPointsHigh();
        iArrayWidth = myWallModel.getNumEchoPointsWide();
        myResponsesArray = new double[ iArrayHeight ][ iArrayWidth ];

        for( j = 0; j < iArrayHeight; j++ )
        {
            for( i = 0; i < iArrayWidth; i++ )
                myResponsesArray[ j ][ i ] = myWallModel.getResponseByIndex( j * iArrayWidth + i );
        }
    }


    protected void initializeGUI()
    {
        setSize( iDisplayWidth, iDisplayHeight );

        myDrawingCanvas = new Canvas();
        myDrawingCanvas.setSize( iDisplayWidth, iDisplayHeight );
        add( myDrawingCanvas );
    setVisible( true );
}

public void brighten()
{
    dBrightnessFactor *= +2.0e0d;
    plotResponseBuffer();
    System.out.println( "brighten() called, dBrightnessFactor = " + dBrightnessFactor );
}

public void darken()
{
    dBrightnessFactor /= +2.0e0d;
    plotResponseBuffer();
    System.out.println( "darken() called, dBrightnessFactor = " + dBrightnessFactor );
}

public void fineBrighten()
{
    dBrightnessFactor *= +1.1e0d;
    plotResponseBuffer();
    System.out.println( "fineBrighten() called, dBrightnessFactor = " + dBrightnessFactor );
}

public void fineDarken()
{
    dBrightnessFactor /= +1.1e0d;
    plotResponseBuffer();
    System.out.println( "fineDarken() called, dBrightnessFactor = " + dBrightnessFactor );
}

public void update()
{
    myResponsesArray = null;
    updateResponsesArray();
    plotResponseBuffer();
}

public void cleanCanvas()
{
    Graphics g = myDrawingCanvas.getGraphics();
    g.setColor( myDrawingCanvas.getBackground() );
    g.fillRect( 0, 0, iDisplayWidth, iDisplayHeight );
}

protected void autoZoom()
{
    iArrayWidth = myResponsesArray.length;
    iArrayHeight = myResponsesArray[0].length;
    Dimension dimDrawingCanvas = myDrawingCanvas.getSize();

    iZoomFactor = (int)( (double)dimDrawingCanvas.width / (double)iArrayWidth );

    if( iZoomFactor < 1 )
        iZoomFactor = 1;
}

public void plotResponseBuffer()
{
    int i, j;
    int iShade;
    Color DotShade;

    cleanCanvas();
    Graphics g = myDrawingCanvas.getGraphics();
    autoZoom();

    for( j = 0; j < iArrayHeight; j++ )
    {
        for( i = 0; i < iArrayWidth; i++ )
        {
            iShade = (int)( myResponsesArray[i][j] * dBrightnessFactor + +1.27e+2d );

            if( iShade > 254 )            // TOO BRIGHT
                DotShade = Color.white;
            else if( iShade < 1 )        // TOO DARK
                DotShade = Color.black;
            else                         // SOMEWHERE IN THE GRAYS
                DotShade = new Color( iShade, iShade, iShade );

            g.setColor( DotShade );
            g.fillRect( ( i * iZoomFactor ), ( j * iZoomFactor ), iZoomFactor, iZoomFactor );
        }
    }
}
```

```
}
```

```
//------------------------------------------------------------------------
//
//        Department of Computer Science, SUNY Institute of Technology
//         Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//        Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
// Class:          WallSignalQueueMgr
// Author:         Matt Gentner
// Original Date:  January, 1998
//------------------------------------------------------------------------
/**
 * THE WallSignalQueueMgr CLASS OF OBJECTS IS DERIVATIVE OF
 * THE SignalQueueMgr CLASS.  THE WallSignalQueueMgr IS SPECIALIZED
 * FOR MANAGING THE EFFECTS OF ECHOING ABSTRACTIONS IN THE Soundscape
 * SIMULATION.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//------------------------------------------------------------------------

import java.lang.String;
import java.util.Vector;

import IEar;
import ISoundListener;
import ISoundSource;
import IWallModel;
import SignalQueue;
import SignalQueueMgr;


public class WallSignalQueueMgr extends SignalQueueMgr
        implements ISignalQueueMgr
{
    protected IWallModel myWallModel = null;
    protected Vector vecLeftEarSignalQueues = null;
    protected Vector vecRightEarSignalQueues = null;
    protected Vector vecSoundListenerSignalQueues = null;
    protected Vector vecEchoPoints = null;


    public WallSignalQueueMgr( IEar IN_LeftEar,
                               IEar IN_RightEar,
                               IWallModel IN_myWallModel,
                               Vector IN_vecSharedSoundListeners )
    {
        super( IN_LeftEar,
               IN_RightEar,
               IN_vecSharedSoundListeners );

        myWallModel = IN_myWallModel;

        solicitEchoPoints();
        enlargeTrashBin( ( 2 * vecEchoPoints.size() ) );
    }

    ///////////////////////////////////////////////////////////////////////
    // ISignalQueueMgr INTERFACE IMPLEMENTATIONS:

    public String initializeSignalQueues()
    {
        System.out.println( "In WallSignalQueueMgr, initializing SignalQueues.." );
        int i;
        int iNumEchoPoints = vecEchoPoints.size();
        int iNumSoundListeners = vecSharedSoundListeners.size();
        int iNumSoundListenerSignalQueues = ( vecEchoPoints.size() * iNumSoundListeners );
        String strProgressLog = new String( "" );
        vecLeftEarSignalQueues = new Vector( vecEchoPoints.size(), 1 );
        vecRightEarSignalQueues = new Vector( vecEchoPoints.size(), 1 );
        vecSoundListenerSignalQueues = new Vector( iNumSoundListenerSignalQueues, 1 );

        for( i = 0; i < iNumEchoPoints; i++ )
        {
            vecLeftEarSignalQueues.addElement( new SignalQueue() );
            vecRightEarSignalQueues.addElement( new SignalQueue() );
            strProgressLog += "..";
        }

        for( i = 0; i < iNumSoundListenerSignalQueues; i++ )
        {
            vecSoundListenerSignalQueues.addElement( new SignalQueue() );
            strProgressLog += ".";
        }

        System.out.println( "vecLeftEarSignalQueues.size() = " +
                            vecLeftEarSignalQueues.size() );
        System.out.println( "vecRightEarSignalQueues.size() = " +
                            vecRightEarSignalQueues.size() );
        System.out.println( "vecSoundListenerSignalQueues.size() = " +
                            vecSoundListenerSignalQueues.size() );

        return strProgressLog;
    }

    public Vector solicitSharedSoundListeners()
    {
        return vecEchoPoints;
    }

    public void makeSignals( double IN_dCurrentTime )
    {
        if( !myWallModel.isMuted() )
        {
            makeEarSignals( IN_dCurrentTime );
            makeSoundListenerSignals( IN_dCurrentTime );
        }
    }

    protected void makeEarSignals( double IN_dCurrentTime )
    {
        int i;
        int iNumEarSignalQueues = vecLeftEarSignalQueues.size();
        int iNumEchoPoints = vecEchoPoints.size();

        if( iNumEchoPoints != iNumEarSignalQueues )
        {
            System.out.println( "In WallSignalQueueMgr, iNumEchoPoints != iNumEarSignalQueues" );
            System.out.println( "iNumEchoPoints = " + iNumEchoPoints );
            System.out.println( "iNumEarSignalQueues = " + iNumEarSignalQueues );
        }

        if( !myWallModel.isMutedToLeftEar() )
        {
            for( i = 0; i < iNumEchoPoints; i++ )
            {
                // System.out.println( "Creating a test Signal for the Left Ear.." );
                sigTempSignal = NewSignal( LeftEar,
                                (ISoundSource)vecEchoPoints.elementAt( i ),
                                IN_dCurrentTime );

                // System.out.println( "Test LeftEar signal strength:  "
                //                      + sigTempSignal.getSignalStrength() );

                if( signalIsOK( sigTempSignal ) )
                {
                    ( (SignalQueue)vecLeftEarSignalQueues.elementAt( i ) ).enqueueSignal( sigTempSignal );
                    // System.out.println( "Queued LeftEar signal with strength:  "
                    //                      + sigTempSignal.getSignalStrength() );
                }
            }
        }

        if( !myWallModel.isMutedToRightEar() )
        {
            for( i = 0; i < iNumEchoPoints; i++ )
            {
                sigTempSignal = NewSignal( RightEar,
                                (ISoundSource)vecEchoPoints.elementAt( i ),
                                IN_dCurrentTime );

                if( signalIsOK( sigTempSignal ) )
                    ( (SignalQueue)vecRightEarSignalQueues.elementAt( i ) ).enqueueSignal( sigTempSignal );
            }
        }
    }

    protected void makeSoundListenerSignals( double IN_dCurrentTime )
    {
        int i, j;
        int iNumEchoPoints = vecEchoPoints.size();
        int iNumSoundListeners = vecSharedSoundListeners.size();
        ISoundSource tempEchoPoint = null;

        for( j = 0; j < iNumEchoPoints; j++ )
        {
```

```
        tempEchoPoint = ( (ISoundSource)vecEchoPoints.elementAt( j ) );
        for( i = 0; i < iNumSoundListeners; i++ )
        {
            sigTempSignal = NewSignal( (ISoundListener)vecSharedSoundListeners.elementAt( i ),
                                        tempEchoPoint, IN_dCurrentTime );

            if( signalIsOK( sigTempSignal ) )
                ( (SignalQueue)vecSoundListenerSignalQueues.elementAt( j + i ) ).enqueueSignal( sigTempSignal );
        }
    }
}

// THIS METHOD CHECKS FOR Signal OBJECTS WHICH HAVE MET THEIR
// TIME OF EFFECT, LETS THEM SPEAK TO THEIR TARGETS, AND
// THEN REMOVES THEM FROM THEIR SignalQueue.
public void effectSignals( double IN_dCurrentTime )
{
    int iNumSoundListeners = vecSharedSoundListeners.size();

    effectEarSignals( IN_dCurrentTime );

    for( int i = 0; i < iNumSoundListeners; i++ )
    {
        effectSoundListenerSignals( IN_dCurrentTime,
            (ISoundListener)vecSharedSoundListeners.elementAt( i ), i );
    }
}

protected void effectEarSignals( double IN_dCurrentTime )
{
    boolean bFoundReadySignal = true;
    int i;
    int iNumReadySignals = 0;
    int iNumLeftEarSignalQueues = vecLeftEarSignalQueues.size();
    int iNumRightEarSignalQueues = vecRightEarSignalQueues.size();
    double dCumulativeSignalStrength = 0.0e0d;
    double dAvgSignalStrength = 0.0e0d;
    sigTempSignal = null;

    for( i = 0; i < iNumLeftEarSignalQueues; i++ )
    {
        while( bFoundReadySignal )  // WHILE READY Signals ARE FOUND
        {
            bFoundReadySignal = false;
            sigTempSignal = ((SignalQueue)vecLeftEarSignalQueues.elementAt( i )).peekAtSignalQueue();

            if( ( sigTempSignal != null ) && ( IN_dCurrentTime >= sigTempSignal.getTimeEffect() ) )
            {
                bFoundReadySignal = true;
                sigTempSignal = ((SignalQueue)vecLeftEarSignalQueues.elementAt( i )).dequeueSignal();
                dCumulativeSignalStrength += sigTempSignal.getSignalStrength();
                iNumReadySignals++;
                addToTrashBin( sigTempSignal );
            }
        }
        bFoundReadySignal = true;
    }

    if( iNumReadySignals > 0 )
    {
        dAvgSignalStrength = ( dCumulativeSignalStrength / ( (double)iNumReadySignals ) );
        LeftEar.rattle( dAvgSignalStrength );
    }

    bFoundReadySignal = true;
    iNumReadySignals = 0;
    dCumulativeSignalStrength = 0.0e0d;
    dAvgSignalStrength = 0.0e0d;
    sigTempSignal = null;
    for( i = 0; i < iNumRightEarSignalQueues; i++ )
    {
        while( bFoundReadySignal )  // WHILE READY Signals ARE FOUND
        {
            bFoundReadySignal = false;
            sigTempSignal = ((SignalQueue)vecRightEarSignalQueues.elementAt( i )).peekAtSignalQueue();

            if( ( sigTempSignal != null ) && ( IN_dCurrentTime >= sigTempSignal.getTimeEffect() ) )
            {
                bFoundReadySignal = true;
                sigTempSignal = ((SignalQueue)vecRightEarSignalQueues.elementAt( i )).dequeueSignal();
                dCumulativeSignalStrength += sigTempSignal.getSignalStrength();
                iNumReadySignals++;
                addToTrashBin( sigTempSignal );
            }
        }
```

```
                bFoundReadySignal = true;
            }

            if( iNumReadySignals > 0 )
            {
                dAvgSignalStrength = ( dCumulativeSignalStrength / ( (double)iNumReadySignals ) );
                RightEar.rattle( dAvgSignalStrength );
            }
        }

        protected void effectSoundListenerSignals( double IN_dCurrentTime,
                                                   ISoundListener IN_SoundListener,
                                                   int IN_iSoundListenerNum )
        {
            boolean bFoundReadySignal = true;
            int i;
            int iNumReadySignals = 0;
            int iNumEchoPoints = vecEchoPoints.size();
            int iFirstSignalQueue = ( IN_iSoundListenerNum * iNumEchoPoints );
            int iLastSignalQueue = ( iFirstSignalQueue + iNumEchoPoints );
            double dCumulativeSignalStrength = 0.0e0d;
            double dAvgSignalStrength = 0.0e0d;
            sigTempSignal = null;

            for( i = iFirstSignalQueue; i < iLastSignalQueue; i++ )
            {
                while( bFoundReadySignal )  // WHILE READY Signals ARE FOUND
                {
                    bFoundReadySignal = false;
                    sigTempSignal = ((SignalQueue)vecSoundListenerSignalQueues.elementAt( i )).peekAtSignalQueue();

                    if( ( sigTempSignal != null ) && ( IN_dCurrentTime >= sigTempSignal.getTimeEffect() ) )
                    {
                        bFoundReadySignal = true;
                        sigTempSignal = ((SignalQueue)vecSoundListenerSignalQueues.elementAt( i )).dequeueSignal();
                        dCumulativeSignalStrength += sigTempSignal.getSignalStrength();
                        iNumReadySignals++;
                        addToTrashBin( sigTempSignal );
                    }
                }
                bFoundReadySignal = true;
            }

            if( iNumReadySignals > 0 )
            {
                dAvgSignalStrength = ( dCumulativeSignalStrength / ( (double)iNumReadySignals ) );
                IN_SoundListener.rattle( dAvgSignalStrength );
            }
        }

        public void finalizeInstant()
        {
            myWallModel.finalizeInstant();
        }
        // END OF ISignalQueueMgr INTERFACE IMPLEMENTATIONS
        ///////////////////////////////////////////////////////////////////////////

        protected void solicitEchoPoints()
        {
            vecEchoPoints = null;
            System.gc();
            vecEchoPoints =  new Vector( myWallModel.getNumEchoPoints(), 1 );

            for( int i = 0; i < myWallModel.getNumEchoPoints(); i++ )
            {
                vecSharedSoundListeners.addElement( (ISoundListener)myWallModel.getEchoPointByIndex( i ) );
                vecEchoPoints.addElement( (IEchoPoint)myWallModel.getEchoPointByIndex( i ) );
            }
        }

    }
```

```
//----------------------------------------------------------------------
//
//          Department of Computer Science, SUNY Institute of Technology
//           Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//          Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          WallViewCardSelector
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class WallViewCardSelector extends the JDK's CheckboxGroup
 * class, and allows simulator users to select either the
 * WallControls or the WallResponses to be displayed.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.event.ItemListener;

public class WallViewCardSelector extends CheckboxGroup
{
    protected Checkbox cbSettings = null;
    protected Checkbox cbResponses = null;
    protected ItemListener myItemListener = null;

    WallViewCardSelector( ItemListener ItemListener_param )
    {
        myItemListener = ItemListener_param;

        initializeCheckBoxes();

        setSelectedCheckbox( cbSettings );
    }

    protected void initializeCheckBoxes()
    {
        cbSettings = new Checkbox( "Settings", this, true );
        cbSettings.addItemListener( myItemListener );

        cbResponses = new Checkbox( "Responses", this, false );
        cbResponses.addItemListener( myItemListener );
    }

    public Checkbox getSettingsCheckbox()
    {
        return cbSettings;
    }

    public Checkbox getResponsesCheckbox()
    {
        return cbResponses;
    }

}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          WallViewFrame
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class WallViewFrame is the main visual
 * user interface component for an WallModel.
 * It holds display components for WallModel
 * attributes and controls which allow the user
 * to configure the WallModel values.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Button;
import java.awt.Component;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.lang.String;
import java.lang.Throwable;

import IObserver;
import ITextLogger;
import ITextLogUser;
import IWallModel;
import IWallView;
import WallController;
import WallViewPanel;
import WallViewCardSelector;

public class WallViewFrame extends Frame
    implements    IWallView, ItemListener
    // AS SUCH, implements IObserver, IView, ITextLogUser
{
    // INSTANCE VARIABLES:
    protected Button btnBrighten = null;
    protected Button btnDarken = null;
    protected Button btnFineBrighten = null;
    protected Button btnFineDarken = null;

    protected GridBagLayout gridbag = null;
    protected GridBagConstraints gridbagconstraints = null;

    protected ITextLogger myTextLog = null;
    protected IWallModel myWallModel = null;
    protected WallController myWallController = null;
    protected WallViewPanel myWallViewPanel = null;
    protected WallViewCardSelector myWallViewPanelCardSelector = null;

    // GUI CONSTRUCTOR:
    public WallViewFrame( IWallModel WallModel_param, ITextLogger IN_TextLogger )
    {
        this( WallModel_param, "Wall View Frame", IN_TextLogger );
    }

    public WallViewFrame( IWallModel WallModel_param,
                          String name_param,
                          ITextLogger IN_TextLogger )
    {
        super( name_param );

        myWallModel = WallModel_param;
        setTextLog( IN_TextLogger );
        myWallModel.attach( this );
        myWallModel.setTextLog( IN_TextLogger );

        initializeAggegates();
        initializeGUI();
        setVisible( true );
    }

    ///////////////////////////////////////////////////////////////////////////
    // INITIALIZATION METHODS:
    protected void initializeAggegates()
    {
        myWallViewPanel = new WallViewPanel( 300, 320, myWallModel, getTextLog() );
        myWallViewPanelCardSelector = new WallViewCardSelector( this );
        makeController();

        btnBrighten = new Button( "Brighten" );
        btnBrighten.addActionListener( myWallController );
        btnDarken = new Button( "Darken" );
        btnDarken.addActionListener( myWallController );

        btnFineBrighten = new Button( "Fine Brighten" );
        btnFineBrighten.addActionListener( myWallController );
        btnFineDarken = new Button( "Fine Darken" );
        btnFineDarken.addActionListener( myWallController );
    }

    protected void addComponent( Component c )
    {
        gridbag.setConstraints( c, gridbagconstraints );
        add( c );
    }

    public void initializeGUI()
    {
        setLocation( 30, 30 );
        setSize( 300, 500 );

        gridbag = new GridBagLayout();
        gridbagconstraints = new GridBagConstraints();
        setLayout( gridbag );

        gridbagconstraints.weightx = 1.0;
        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        gridbagconstraints.fill = GridBagConstraints.BOTH;
        gridbagconstraints.gridheight = 12;
        gridbagconstraints.gridheight = 1;
        gridbagconstraints.weighty = 0.0;          //reset to the default

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( btnBrighten );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnDarken );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( btnFineBrighten );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( btnFineDarken );

        gridbagconstraints.gridwidth = GridBagConstraints.BOTH;
        addComponent( myWallViewPanelCardSelector.getSettingsCheckbox() );
        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( myWallViewPanelCardSelector.getResponsesCheckbox() );

        gridbagconstraints.gridwidth = GridBagConstraints.REMAINDER; //end row
        addComponent( myWallViewPanel );

    }
    // END OF INITIALIZATION METHODS
    ///////////////////////////////////////////////////////////////////////////


    ///////////////////////////////////////////////////////////////////////////
    // Frame CLASS METHOD OVER-RIDES:
    public void paint( Graphics g )
    {
        myWallController.repaint();

        myWallViewPanel.repaint();
    }

    // END OF Frame CLASS METHOD OVER-RIDES
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IObserver INTERFACE IMPLEMENTATIONS:

    public void update()
    {
        myWallController.update();
    myWallViewPanel.update();
}
// END OF IObserver INTERFACE IMPLEMENTATIONS
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
// IWallView INTERFACE IMPLEMENTATIONS:

public void makeController()
{
    myWallController = myWallViewPanel.getWallController();
}

public void display()
{
    repaint();
}

public void finalize() throws Throwable
{
    myWallModel.detach( this );
    super.finalize();
}

public IPositionHolderModel getPositionHolderModel()
{
    return myWallModel;
}
// END OF IWallView INTERFACE IMPLEMENTATIONS
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
// IWallView INTERFACE IMPLEMENTATIONS:
public void brighten()
{
    myWallViewPanel.brighten();
}

public void darken()
{
    myWallViewPanel.darken();
}

public void fineBrighten()
{
    myWallViewPanel.fineBrighten();
}

public void fineDarken()
{
    myWallViewPanel.fineDarken();
}

// END OF IWallView INTERFACE IMPLEMENTATIONS
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
// ACCESSOR METHODS:
public WallViewPanel getWallViewPanel()
{
    return myWallViewPanel;
}

public IWallModel getWallModel()
{
    return myWallModel;
}
// END OF ACCESSOR METHODS
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
// ItemListener INTERFACE IMPLEMENTATION:
public void itemStateChanged( ItemEvent ie )
{
    if( ie.getSource() == myWallViewPanelCardSelector.getSettingsCheckbox() )
    {
        if( ( myWallViewPanelCardSelector.getSettingsCheckbox() ).getState() )
            myWallViewPanel.showSettingsCard();
    }
    else if( ie.getSource() == myWallViewPanelCardSelector.getResponsesCheckbox() )
    {
        if( ( myWallViewPanelCardSelector.getResponsesCheckbox() ).getState() )
            myWallViewPanel.showResponsesCard();
    }
}
```

```
    // END OF ItemListener INTERFACE IMPLEMENTATION
    /////////////////////////////////////////////////////////////////////////////

    /////////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    /////////////////////////////////////////////////////////////////////////////

    /////////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "WallViewFrame" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    /////////////////////////////////////////////////////////////////////////////

}  // end of WallViewFrame class
```

```
//-------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
//  Class:          WallViewPanel
//  Author:         Matt Gentner
//  Original Date:  January, 1998
//-------------------------------------------------------------------
/**
 * The class WallViewPanel extends the JDK's Panel
 * class, and is used to allocate a portion of the
 * WallViewFrame's visual area for the WallResponsesViewCanvas.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//-------------------------------------------------------------------

import java.awt.Panel;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.lang.Throwable;

import ITextLogger;
import ITextLogUser;
import IWallModel;
import WallController;
import WallResponsesViewCanvas;


public class WallViewPanel
    extends Panel
    // AS SUCH, extends PositonHolderViewPanel AND ViewPanel
    // AS SUCH, implements IWallView
{
    protected int iDisplayWidth = 0;
    protected int iDisplayHeight = 0;

    protected CardLayout myCardLayoutManager = null;
    protected IWallModel myWallModel = null;
    protected ITextLogger myTextLog = null;
    protected WallController myWallController = null;
    protected WallResponsesViewCanvas myWallResponsesViewCanvas = null;

    public WallViewPanel(    int iDisplayWidth_param,
                             int iDisplayHeight_param,
                             IWallModel WallModel_param,
                             ITextLogger IN_TextLogger )
    {
        iDisplayWidth = iDisplayWidth_param;
        iDisplayHeight = iDisplayHeight_param;

        myWallModel = WallModel_param;
        setTextLog( IN_TextLogger );

        makeController();

        myWallResponsesViewCanvas = new WallResponsesViewCanvas(  iDisplayWidth,
                                                                  iDisplayHeight,
                                                                 myWallModel );
        initializeGUI();
    }

    public void initializeGUI()
    {
        myCardLayoutManager = new CardLayout();
        setLayout( myCardLayoutManager );
        add( myWallResponsesViewCanvas, "Responses" );
        add( myWallController, "Settings" );
        myCardLayoutManager.show( this, "Settings" );
    }

    public void showSettingsCard()
    {
        myCardLayoutManager.show( this, "Settings" );
    }

    public void showResponsesCard()
    {
        myCardLayoutManager.show( this, "Responses" );
    }

    public WallController getWallController()
    {
        return myWallController;
    }

    ///////////////////////////////////////////////////////////////////////////
    // PUBLIC MODIFER METHODS:

    public void brighten()
    {
        myWallResponsesViewCanvas.brighten();
    }

    public void darken()
    {
        myWallResponsesViewCanvas.darken();
    }

    public void fineBrighten()
    {
        myWallResponsesViewCanvas.fineBrighten();
    }

    public void fineDarken()
    {
        myWallResponsesViewCanvas.fineDarken();
    }
    // END OF PUBLIC MODIFIER METHODS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // IWallView INTERFACE IMPLEMENTATIONS:

    public void makeController()
    {
        myWallController = new WallController(  iDisplayWidth,
                                                iDisplayHeight,
                                                 myWallModel,
                                                 this,
                                                 getTextLog() );
    }

    public void display()
    {
        repaint();
    }

    public void update()
    {
        myWallResponsesViewCanvas.update();
    }

    // END OF IWallView INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ITextLogUser INTERFACE IMPLEMENTATIONS:

    public ITextLogger getTextLog()
    {
        return myTextLog;
    }

    public void setTextLog( ITextLogger IN_TextLog )
    {
        myTextLog = ( IN_TextLog );
    }
    // END OF ITextLogUser INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////

    ///////////////////////////////////////////////////////////////////////////
    // ISoundscapeBase INTERFACE IMPLEMENTATIONS:

    public String getClassDefName()
    {
        return( "WallViewPanel" );
    }
    // END OF ISoundscapeBase INTERFACE IMPLEMENTATIONS
    ///////////////////////////////////////////////////////////////////////////
}
```

```
//----------------------------------------------------------------------
//
//         Department of Computer Science, SUNY Institute of Technology
//          Project Soundscape-SA, Stand Alone Auditory Scene Simulator
//         Java Source Code Class Definition File, (c) 1998 Matt Gentner
//
// Class:          WaveSelectionGroup
// Author:         Matt Gentner
// Original Date:  January, 1998
//----------------------------------------------------------------------
/**
 * The class WaveSelectionGroup extends the JDK's CheckboxGroup
 * class, and allows the user to select wave shapes of periodic
 * sound sources in the simulator.
 * <p>
 * Examples:
 * <p>
 * @author  Matt Gentner
 * @version 1.0, 01/13/97
 * @see
 * @since   SoundscapeSA1.0
 */
//----------------------------------------------------------------------

import java.awt.Checkbox;
import java.awt.CheckboxGroup;
import java.awt.event.ItemListener;

public class WaveSelectionGroup extends CheckboxGroup
{
    protected Checkbox cbSine = null;
    protected Checkbox cbSquare = null;
    protected Checkbox cbSawTooth = null;
    protected ItemListener myItemListener = null;

    WaveSelectionGroup( ItemListener ItemListener_param )
    {
        myItemListener = ItemListener_param;

        initializeCheckBoxes();

        setSelectedCheckbox( cbSine );
    }

    protected void initializeCheckBoxes()
    {
        cbSine = new Checkbox( "Sine", this, true );
        cbSine.addItemListener( myItemListener );

        cbSquare = new Checkbox( "Square", this, false );
        cbSquare.addItemListener( myItemListener );

        cbSawTooth = new Checkbox( "SawTooth", this, false );
        cbSawTooth.addItemListener( myItemListener );
    }

    public Checkbox getSineWaveCheckbox()
    {
        return cbSine;
    }

    public Checkbox getSquareWaveCheckbox()
    {
        return cbSquare;
    }

    public Checkbox getSawToothWaveCheckbox()
    {
        return cbSawTooth;
    }

}
```

IMAGE EVALUATION TEST TARGET (QA-3)