Chapter 1

Introduction

1.1 Overview

Nowadays, the business logic layers of the modern applications are mostly constructed using either object-oriented model or relational model. The object oriented model is based on software engineering principles such as coupling, inheritance, cohesion, and encapsulation, whereas the relational model is based on predicate logic and set theory principles [1]. The object oriented model chains the building of applications within objects that have both data and behavior.

Relational model support the storage of data in tables and treatment of that data through data manipulation language (DML) inside within the database through stored procedures and externally through SQL (Structured Query Language). The relational model is in use by many database systems [1].

Object oriented technology is also commonly used in database application development. The differences between the two technologies was called the object-relational impedance mismatch [17][11]. In particular, when objects need to be stored in relational database. Object-relational mapping (ORM) is appearing to play an important role to overcome the problem of impedance mismatch. ORM is a new technology that allows applications to access relational data in an object-oriented manner. With the wide-spread use of ORM technology, domain object are built as objects and the application logic manipulates these objects in a pure object oriented manner.

The critical issue that arises is that whether such an object oriented model for business logic layer is a good choice in general. Proponents of object oriented approach tend to assume that an object oriented business model will made the system easier to maintain, easier to extend, and easier to re-use.

The aim of this paper is to critically examine this assumption and to carefully compare the applicability and flexibility of the object oriented system in comparison to relational system. The findings from this project will be significant for practical application where the business logic layer is implemented in an object oriented fashion as the present trend in enterprise computing seems to be heading.

1.2 Outline of the thesis

This thesis is structured as follows. Chapter 2, provides a theoretical background to the thesis which includes basic concept regarding relational model and an object oriented programming. The second half of the chapter is dedicated to introduce the problem of impedance mismatch and the basic concept of ORM. Chapter 3, will introduce and discuss some researches and papers regarding the comparison between the two paradigms. In addition , it will present the research proposal. Chapter 4, will present a simple case study for both technological approaches to make an initial comparison regarding the different of effects involved in implementing them and change them with respect to requirements change. Chapter 5, introduces a complicated case study regarding to change requirements in business policy. Chapter 6, discusses issue of relational representation. The final chapter, introduce a realistic case study to produce much more statistical data to compare the two approaches. Moreover, it will also highlight the issue of relationship representation. The thesis concludes with a very brief discussion of the achieved results and gives some suggestions of future work.

Chapter 2

Background

2.1 The Relational model

As already mentioned, the relational model is based on the predicate logic and set theory, developed by E.F. Code in 1970 [1]. It is built around data normalization, which simplifies maintenance and data storage by minimizing the duplication of information in turn to enforce data consistency. It used the basic concept of a relation or table. Each table has a number of columns with unique names. The columns in the table identify the attributes such as name, age, and so on. A row contains all the data of a single instance of the table such as a person named Michael. Moreover ,In the relational model, each row have to have a unique identification or key based on the data. For example , in figure 1, a social security account number (SSAN) is the key that uniquely identifies each row in the table. Normally, keys are used to join data from two or more tables based on same identification. The relational approach also includes concepts such as foreign keys , which are primary keys in one table that kept in another table to allow for the joining of data. For example of foreign keys is storing father's SSAN and your mother's in the row that represent person. Thus , parents' SSANs are keys for the rows that represent them and they are foreign keys in the row that represents person.

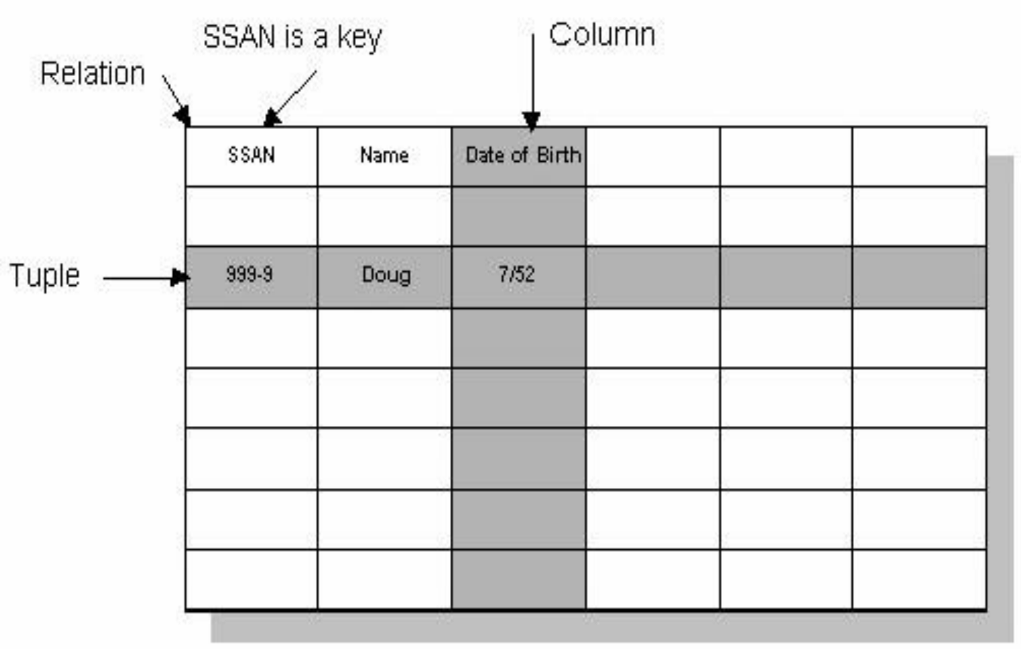


Figure 1: A table is a data structure used to organize information.

Now the relational model is the dominant model for commercial data processing applications such as A Relational database management system (RDBMS) . RDBMSs use Structured Query Language (SQL) as the data definition language (DDL) and the data manipulation language (DML).

Structured Query Language (SQL) was developed at IBM in the early 1970s. This version, firstly called SEQUEL, was considered to manipulate and retrieve data stored in IBM's original relational database product.SQL provides the capability to create and define relational database tables. After these tables are defined, the language permits one to add data to these tables. Once data has been added, one can modify, retrieve, or remove that data. SQL provides the ability of defining what type of authority one might have when accessing the data. Indeed , SQL Statements play an important role when developing database application.

Data definition language (DDL) describes the section of SQL that allows you to create, alter, and drop database objects. These database objects include schemas, , sequences views, tables, indexes ,catalogs and aliases .Currently , it refers to any formal language for describing data or information structures, like XML schemas.

The Data Manipulation Language (DML) is a language which enables users to access and manipulate data. All processing is based on values in fields of records. A SQL data manipulation statement (DML) provides four commands: Insert, Select, Update, and Delete. Database users used these commands during the routine operation of the database. Moreover, it is flexible, powerful and easy to learn. The types of queries vary from complicated multi-table queries to simple single-table queries to involving joins, nesting, set union/differences, and others. Examples of RDBMSs include Microsoft Access, developed by Microsoft and Oracle, developed by Oracle Corporation.

Relational Databases inability to handle application areas like spatial databases , applications involving special types databases , images and other applications that involve complex interrelationships of data. Furthermore , Relational databases do not natively support inheritance (It will be describe later on).[1] for further information about relational model.

2.2The Object oriented programming

Object-Oriented Programming (OOP) is a programming paradigm that enable programmer to divide a program in building blocks known as objects. Programming techniques contain features as coupling, inheritance, cohesion, and encapsulation. However, there are fundamental concepts found in the strong majority of definition of OOP. The definition of OOP is inspired by the paper of Deborah J. Armstrong.
They are the following:

2.2.1Class : In OOP a class is a template definition for a particular object or groups of objects that will always have the same features. The object created using the class is called instances Instance contains real values instead of variables. For instance, the class dog consist of traits shared by all dogs , such as fur and breed color as well as the ability to sit and bark. However , a class can have subclasses that can inherit all or some of the characteristics of the class. As a result this class become the super class for all subclasses.

2.2.2 Objects: are the building blocks of OOP and are usually defined as variables or data structures that encapsulate behavior and data in a complete unit. Objects are items that can be independently created, represent, and manipulated real world things in an abstract way. Figure 2 illustrations is a common visual representation of a software object.

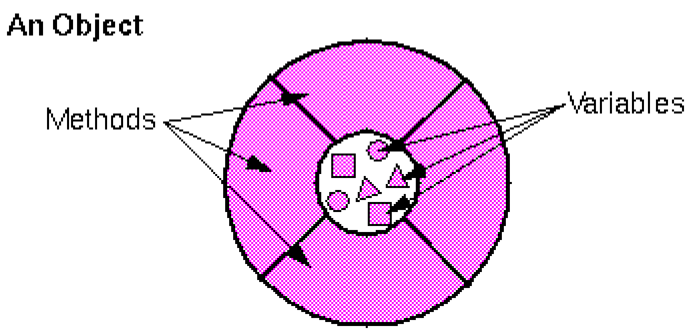


Figure 2 illustrations is a common visual representation of a software object.

2.2.3 Inheritance : In OOP is a mechanism that enable classes to inherit usually state and behavior form other classes. For example , in figure 3 , there are 3 classes: Mammal , Cat and Dog. The Dog and Cat classes both inherit from Mammal. Thus , an object of Cat or Dog can implement the method getEyeColor from the class Mammal . However , the main advantage of inheritance is that intended to assist reuse existing code with little or no modification.

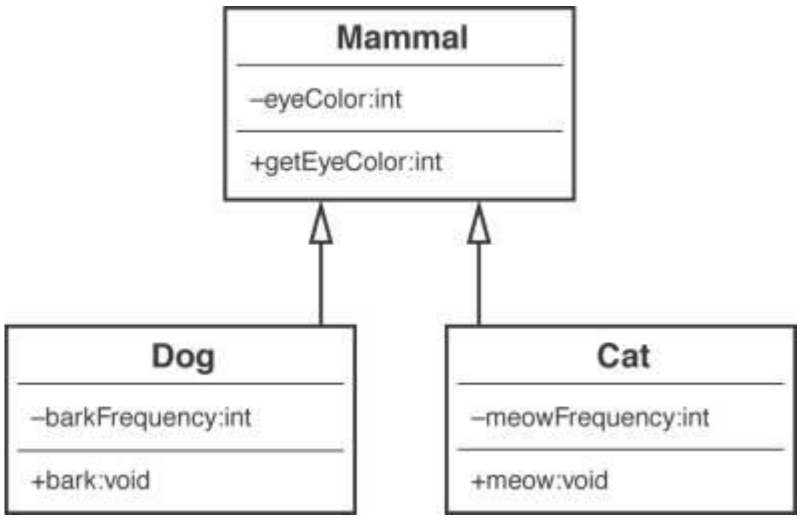


Figure 3:The Dog and Cat classes both inherit from Mammal.

2.2.4 Abstraction: involves the formulation of representations by focusing on differences and similarities with a set of entities to extract intrinsic essential characteristics (relevant common features) and avoid extrinsic incidental characteristics (irrelevant distinguishing features) with the purpose of define a single representation having those characteristics that are related to defining every element in the set.

2.2.5 Encapsulation: makes the methods and data private within an object. The gale of encapsulation is to hiding details of the internal mechanisms and data structures in turn to facilitate abstraction. Therefore, this allows the implementation to be changed without impacting the users of the interface.

2.2.6 Polymorphism: in different ways , the different objects have ability to receive the seam message and behave. Developers only need to understand the interface to use the object .As a result, this can lead to improve code readability.

2.2.7 Coupling: describes or measures dependant one object is on another object. Objects that are tightly coupled can be changed quite radically without impacting each other. The least change to objects that are tightly coupled can cause a mass of problems.

2.2.8 Cohesion : An object with high cohesion is defined for one reason and it only performs that purpose. An object with low cohesion able to do a lot of different things.

2.3 Impedance mismatch

As stated in introduction of this paper, the object oriented technology chains the building of applications within objects that have both data and behavior. Relational technology support the storage of data in tables and treatment of that data through (DML) inside within the database through stored procedures and externally through SQL .Due to the difference technologies in underlying paradigms they are do not work together seamlessly. They are have different type language. For example , java as object oriented language has string whereas Oracle as database has a varchar. Moreover , the majority of relational databases do not support schema inheritance. However , today object oriented considered the fashionable programming languages at the same time the most used database system are relational database. Indeed , the problem came out when objects need to be stored in relational database. There has been a lot of efforts have been made to overcome this. One of the best solutions so far to bridge the gap between these two systems was Object Relational Mapping (ORM or also known as O-R)[15][11][24].

2.4 Object Relational Mapping

Object Relational Mapping (ORM) is a programming technique work as bridge to convert data between incompatible type systems in object-oriented programming and relational databases [11][22]. ORM is a specialization of the general concept of object persistence. It helps making the software more robust via reduces most often the lines of code programmed. The ORM support database-access technology such as separation of concerns , inheritance , information hiding , change detection and database independence. Separation of concerns is the process to divide a program into logical parts in order to prevent overlap or at least reduce it. There are several part must be separated in database programming such as business methods that find domain objects. Information hiding is an implementation strategy that reduces cost and complexity via defining behavior in terms of interface that are implemented via certain classes. Thus ,the behavior of the subclasses is separate from the behavior of the callers. On the other words , changing on one side does not effect to change the other side. Inheritance allows to reuse of code which can help programmers to spend less time rewriting the same code. Change detection is tracking changes to domain object that are used in a database transaction so that at the end of the transaction, will ultimately reflect those changes to the database. Database independence allows applications to work with various databases without the need for change in the application for the purpose of compatibility with the new database.

2.4.1 The modus operandi of ORM

After mapping domain object model classes to relational tables, developers can access them through an API , which is implemented by a persistence provider. In the application layer, data stored in relational databases is represented as objects in the native object programming language. While the application navigates among objects, the data store transparently maps the objects into the cache [14]. Also , the database manages objects in the cache, tracking changed objects and automatically writing them back to the database for a transaction. Moreover , queries against the database are expressed in terms of the domain object model. SQL statements generates directly from the of the domain object model by the provider.

2.4.2 Mapping classes to tables

Inheritance in the domain object model is a relationship between two or more in which one class is specialization of another. There are many ways to map domain object model to relational database. For implementing Inheritance structures to a relational database there are three strategies.

2.4.3 A single table inheritance approach maps all classes in the inheritance hierarchy to a single table that contains a column for each of the fields in any of the class. We present an example to illustrate the issues related to Inheritance mapping. another. Figure 2 shows a animal relations object model in which Cat and Dog are specializations of Mammal. Figure $ depicts the persistence model for the class hierarchy of Figure 2 when this approach is taken. Notice that a MammaOID column was introduced for the primary key of the table.[11][22][13] [24]

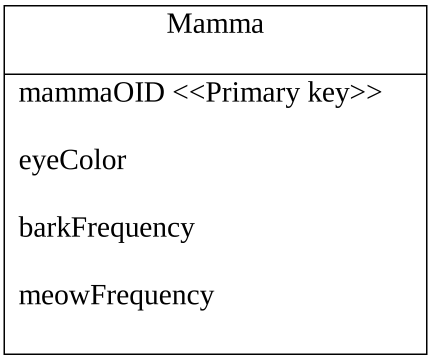


Figure 4: Mapping all classes to a single table

The advantages of A single table inheritance approach are that it is simple, that polymorphism is supported when a person changes roles, and that ad hoc reporting (reporting performed for the specific purposes of a small group of users, who commonly write the reports themselves) is also very easy with this approach because all of the personal data you need is found in one table. The disadvantages are that when a new attribute is added anywhere in the class hierarchy a new attribute should be added to the table[11][22] [24]

2.4.4 A table-per-concrete-class inheritance strategy each data entity includes both the attributes and the inherited attributes of the class that it represents. Figure 5 depicts the persistence model for the class hierarchy of Figure 2 when this approach is taken. There are data entities corresponding to both the Cat class and the Dog class because they are concrete, but not the Mammal class because it is abstract (indicated by the fact that its name is depicted in italics). Each of the data entities was assigned its own primary key, dogOID and catOID respectively. The main advantage to A table-per-class inheritance approach is that it is still fairly easy to perform ad hoc reporting, given that all the data you need about a single class is stored in only one table. There are a number of disadvantages, however. One of them is that when you change a class you must change its table and the table of any of its subclasses. For example, if you were to add weight and height to the Mammal class, you would need to update both tables, which is a lot of work. In addition third, it is difficult to support multiple roles and still maintain data integrity.[11][22][24]

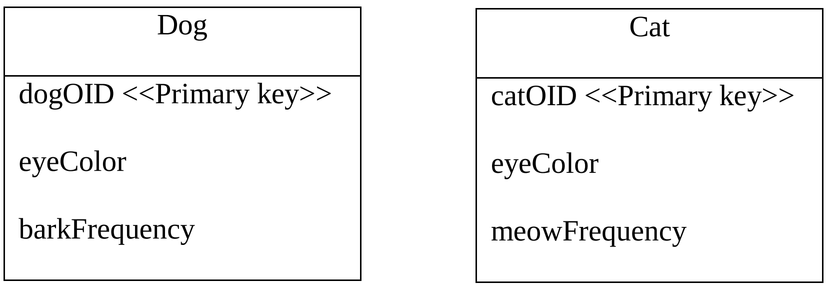


Figure 5. A table-per-concrete-class.

2.4.5 A table-per-class inheritance strategy you create one table per class, the attributes of which are the OID and the attributes that are specific to that class. Figure 6 depicts the persistence model for the class hierarchy of Figure 6 when A table-per-class inheritance approach is taken. Notice that mammalOID is used as the primary key for all three tables. The major advantage of this approach is that it conforms best to object-oriented concepts [11]. It supports polymorphism very well as you just have records in the appropriate tables for each role that an object might have. Moreover , extremely simple to modify superclasses and add new subclasses because you merely need to modify or add one table. On the other hand ,there are a number of disadvantages to this approach. First , there are many tables in the database -- one for every class, actually (plus tables to maintain relationships). Second, it takes longer to write and read data using this technique because you must access multiple tables. Third, ad hoc reporting on your database is difficult, except you add views to simulate the desired tables[11][22].

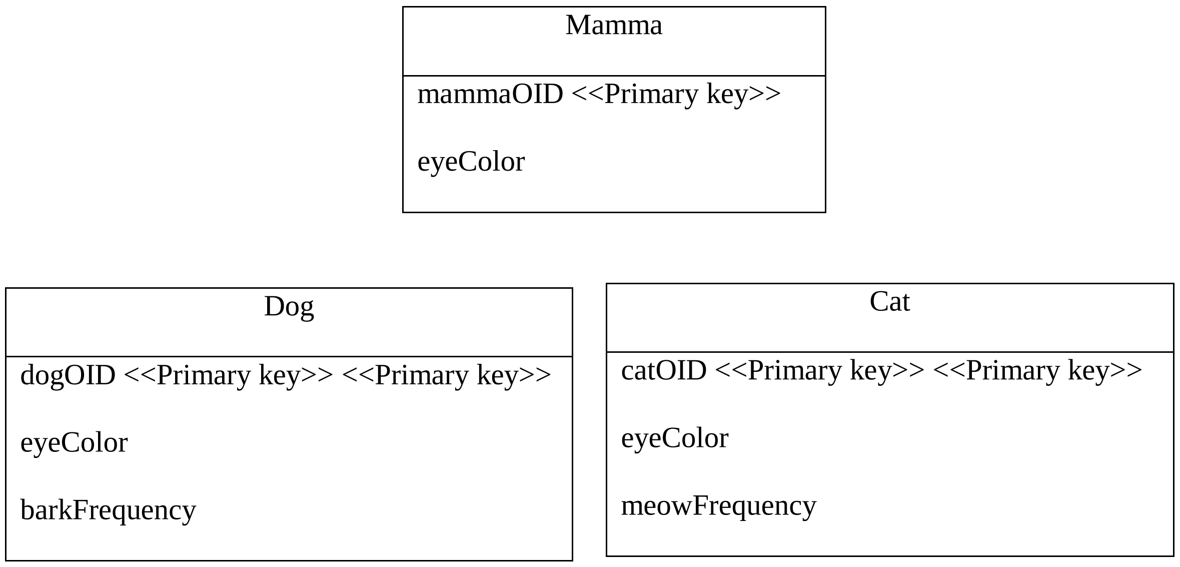


Figure 6. Mapping all classes in to its own table .

2 .5 Competition between the two paradigms

The important issue is that which paradigm is more appropriate in terms of flexibility with respect to requirements change. In the next chapter, we will discuss this point in more detail .

3. Need to Study

Today, changing requirements have become an accepted fact of life for software developer’s .Therefore; we looking for flexible approach can deal with requirements change. However, ORM is very popular in use. According to [11], most of the modern applications are built using ORM technology to access data stored in relational database rather than traditional approach. In addition, it is also claiming that using ORM tools can help reduce cost of a project. Moreover, proponents of object oriented approach tend to assume that an object oriented business model will made the system easier to maintain, easier to extend, and easier to re-use. On the other hand, proponents of traditional fashion argue that not all the world must be handled in objects. In addition, they are arguing that there is some native incompatibility between ORM code and databases. For example, Kenneth argues that you can write efficient and clean code if you know how databases work on their own terms. Moreover , Object-oriented development promises to reduce the maintenance effort, but, these promises are not based on reliable experimentation[12].Indeed , there is a significant lack of research to answer the question of whether object-oriented approach are better than traditional approach in terms of flexibility to requirements change.

The aim of this paper is to critically examine this assumption and to carefully compare the applicability and flexibility of the object oriented system in comparison to relational system. The findings from this project will be significant for practical application where the business logic layer is implemented in an object oriented fashion as the present trend in enterprise computing seems to be heading.

Chapter 3

Related work and Thesis's proposal

In this chapter, we review further some papers that are highly relevant to the issue of addressed by this thesis.

4.1 Previous works

There are few complete experimental results comparing the effectiveness of object oriented design methods. The results are available in [16].This experiment compared the effectiveness of subjects when maintaining both object oriented and procedural designs. Generally, they found that object-oriented designs were more difficult to maintain. Particularly, they found strong results that bad object-oriented designs were more difficult to maintain than procedural designs. In addition, they also found statistically weaker results suggesting that good procedural designs were easier to modify then good object oriented designs. This study was based on a small sample size of 20 students. The conclusion is that, the shift to object-oriented methods will cause a substantial drop in productivity, especially if developers are experienced with procedural methods. Moreover , the study conducted by [15] mention that traditional and some object oriented business process modeling techniques lead to a procedural process model which causes mismatch problems and impedes implementation when they have to be coded in an object based GUI language

The study conducted by [17] was looking at a smaller subset of object-oriented maintainability characteristics. in order to enhanced control the experiment, they looked only at whether inheritance, a fundamental object oriented construct, made programs more difficult or less difficult to maintain. They studied two groups about 57 students. The result was that for complex programs, inheritance makes maintenance more difficult, not less.

The study described in [18] evaluates the comprehension of both object-oriented and procedural code. This research used 30 professional developers in two phases that were at least a week apart. Comprehension was considered as the number of files each subject opened. The results of this research showed that procedural group had larger scope of comprehension. As a result, this means that those subjects maintaining a procedural project needed to open a larger percentage of the source files than subjects maintaining an object-oriented project. However, the largest gap in this study is regarding the definition of comprehension. The number of files for each version of the system is not clear from the experimental design. However, if the number of files in the object-oriented version was larger, then defining comprehension as a percentage of the files that was viewed does not seem to be an accurate measure.

In the contrast, this paper [22] argues against traditional approach. They stated that building application using triggers, views, and stored procedures is problematic for a number of reasons. First, unions or views containing joins are usually not updatable. Second, triggers and defining custom database views for every application accessing mission-critical enterprise data, is rarely acceptable due to security and manageability risks. Furthermore, object-relational features, SQL dialects, and procedural extensions vary significantly from one DBMS to the next.

Moreover, several examples in the literature find that object – oriented approach has many advantages over traditional approach. The study described in [14] compares the implementation of a GIS (Geographic Information System) database using RDBMS and OODBMS developed. They stated that, the most important advantage of using objects as the basis of database design is the ease of using them for both user verification and design. They also found, the object model can be directly implemented in a relational form, particularly the *has a* relationship or aggregation. In addition, they mention that , although normalization is a well developed methodology but , requires training and is difficult for users to understand. In contrast, Object modeling technique uses user information in the form of uses cases, which becomes the basis for the class structure. As a result, this makes the structure and the naming of objects in the structure easier for users to understand and hence verify. The conclusion is that, object models appear to be a more useful way of communicating analysis and design concepts to users.

The study conducted by [20] was based on using Hibernate as an object-relational mapping framework for eHealth systems. They found that Hibernate provokes an outstanding effect on an application development. They also stated that , the first remarkable characteristic consists of a more natural object oriented programming model. The conclusion is that, the need for string query construction, based on variable parameters, is no longer needed. Instead, direct instance storage and retrieval is enabled through the Hibernate interface. However, there are also several examples in the literature find that some advantages with respect of maintainability. Most of them, such as [22] , do not use investigational data to back this claim.

As far as we know, no one really compare the two paradigms in terms of flexibility in responding to changes of requirements. This provides the motivation for our project.

4.2. Proposal

The proposal of this research is to critically examine the assumption that an object oriented business model will made the system easier to maintain, easier to extend , and easier to re-use. In addition, to carefully compare the applicability and flexibility of the object oriented system in comparison to relational system. In other words, answer the question of whether object-oriented approach is better than traditional approach in terms of flexibility in responding to requirements change.

In our proposed approach, in order to compare between an object oriented system and relational system, we will divide our approach to two stages.

First stage, we will use simple case study for both approaches, after the implementation we will add new requirements than we redesign and implement changes in both approaches and compare, in order to determine which approach is more flexible to deal with changing requirements.

This stage is essentially an exploiting stage to establish the techniques and to gain initial understanding. In the next stage, which will be the main stage , we will use realistic case study for both approaches in order to determine which approach can made the system easier to maintain. Again, we will inform numerous changes of requirements of business rules in order to thoroughly investigate the issue. The result of such parallel implementations will forms available source of information for our critical comparison of the two paradigms.

We will use Java Database Connectivity (JDBC), representative of relational system, and Hibernate, representative of the ORM and we will use mySQL as relational database. Both of these are popular open source products.

The findings from this project will be significant for practical application where the business logic layer is implemented in an object oriented fashion as the present trend in enterprise computing seems to be heading. In addition, may be this thesis come up with recommendations that can help to make

العنوان:	Design and Implementation of Business Logic Layer Object-Oriented Design vs. Relational Design
المؤلف الرئيسي:	Al Harthy, Ali Ahmed
مؤلفين آخرين:	Nguyen, Kinh(Super)
التاريخ الميلادي:	2009
موقع:	بندورا ، فيكتوريا
الصفحات:	1 - 82
رقم MD:	615558
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	La Trobe University
الكلية:	Faculty of Science, Technology and Engineering
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	الحاسبات الالكترونية، هندسة البرمجيات، تكنولوجيا المعلومات
رابط:	https://search.mandumah.com/Record/615558

Chapter 4

First Case Study

We will use a simple case study for both technological approaches to make an initial comparison regarding the different of effects involved in implementing them and change them with respect to requirements change.

5.1. Problem Statement

A company requires a Car Park application to maintain information about employees and their parking permits. The car park has a number of parking spots, which are divided into three areas A, B, and C. Employees who want a permit have to pay a fee on a quarterly basis, which will be automatically deducted from their salary.

The purpose of the Car Park application is to help the car park manager to process the employees' applications for parking permits. Each employee has an id, a name , and a phone extension. Each permit has a permit number, the car's registration number, and the section where the car can be parked. An employee can have at most two permits. An employee may change his/her extension in the course of the employment.

When an employee gets a new car and wants to use the new car instead of the old one, he/she has to discontinue the current permit and apply for a new one.

5.2. Use Case Description

The system can be implemented as the following case.

1. Add an new employee
2. Change an employee's phone number
3. Add a permit
4. Delete an employee
5. Terminate a permit
6. Replace a permit
7. Given a permit number , get the details of the employee who has the permit.

Discontinue

This use case described is shown below.

Use Case Name: Add an Employee
Trigger/Goal: To enter the details of a new employee
Actors: Operator (or car park manager, depending on assumption)
Main Flow:

1. Operator enters employee's ID.
2. System validates that ID is new.
3. Operator enters name and phone number.
4. System saves the details.

Extensions:

2a. ID is not new:
2a1. System notifies Operator, and terminates the use case

1.

Use Case Name: Update an Employee's Phone Extension
Trigger/Goal: An employee has a new phone extension, and the operator wants to update his/her phone extension.
Actors: Operator.
Main Flow:

1. Operator enters employee's ID.
2. System validates that ID is exist.
3. Operator enters the new phone number .
4. System saves the new phone number.

Extensions:

2a. ID does not exist:
2a1. System notifies Operator, and terminates the use case

2.

Use Case Name: Add Permit
Trigger/Goal: To enter details of a new permit.
Actors: Operator.
Main Flow:

1. Operator enters employee's ID.
2. System validates that the employee with that ID exists and currently has less than 2 permits.
3. Operator enters the new permit number.
4. System validates the car's registration number.
5. System validates that the registration number is new.
6. System saves the new permit's details.

Extensions:

2a. ID does not exist:
2a1. System notifies Operator, and terminates the use case
2b. Employee already has 2 permits:
2b1. System notifies Operator, and terminates the use case
4a. Permit number is not new:

4a1. System notifies Operator, and terminates the use case

6a. Registration number is not new:

6a1. System notifies Operator, and terminates the use case

3.

Use Case Name: Terminate Permit

Trigger/Goal: To delete details of a permit that has been terminated.

Actors: Operator.

Main Flow:

1. Operator enters the permit number.
2. System validates that the permit exists.
3. System deletes the permits.

Extensions:

2a. Permit number does not exist:

2a1. System notifies Operator, and terminates the use case.

4.

Use Case Name: Replace Permit

Trigger/Goal: To replace a permit for an employee who gets a new car and switches the permit to the new car.

Actors: Operator.

Main Flow:

1. Operator enters the old permit number and the new permit number.
2. The system validates that the old permit number exists and the new permit number is actually new.
3. Operator enters the new car registration number.
4. System deletes the old permit and creates and saves the details of the new permit.

Extensions:

2a. The old permit number does not exist:

2a1. System notifies Operator, and terminates the use case.

2b. The new permit number already exist

2b1. System notifies Operator, and terminates the use case

5.

Use Case Name: Delete Employee

Trigger/Goal: To delete details of an employee who leaves the company.

Actors: Operator.

Main Flow:

1. Operator enters employee's ID.
2. System validates that the employee with that ID exists.
3. System deletes the employee's details and all the employee's permits.

Extensions:

2a. ID does not exist:

2a1. System notifies Operator, **and** terminates the use case.

6.

Use Case Name: Retrieve the Details of the Owner of a Permit.
Trigger/Goal: The operator wants the details of a permit , given the permit number.
Actors: Operator.
Main Flow:

1. Operator enters the permit number.
2. The system validates that the permit number exists.
3. System retrieves and displays the permit's owners' id , name and phone number.
4. System saves the new phone number.

Extensions:

2a. ID does not exist:

2a1. System notifies Operator, and terminates the use case.

5.3. Implementation

In order to measure the overall implementation effort due to new/changed requirements with JDBC and Hibernate, we will use:

- the size of code that is produced to complete a task. Size is measured in lines of code and takes into account new lines added or modified and deleted

- Estimates the time required to complete a task.

I will start implementation with a Hibernate and then JDBC, and I will follow these steps to construct the car park system.

Steps to build a Hibernate application

- Define the domain model (we can use it for JDBC as well).
- Create a database and tables.
- Create Plain Old Java Object (POJO) that needs to be persisted.
- Create a mapping file so Hibernate knows how to persist the class' properties.
- Create a configuration file for Hibernate to know how to connect to a database
- Create the HibernateUtil Helper Class.
- Create a Class to use Persistence Objects.

Steps to build a JDBC application

- Create a database and tables.
- Create JDBC application.

5.3.1.Building a Hibernate application

5.3.1.1.Step 1: Define the domain model

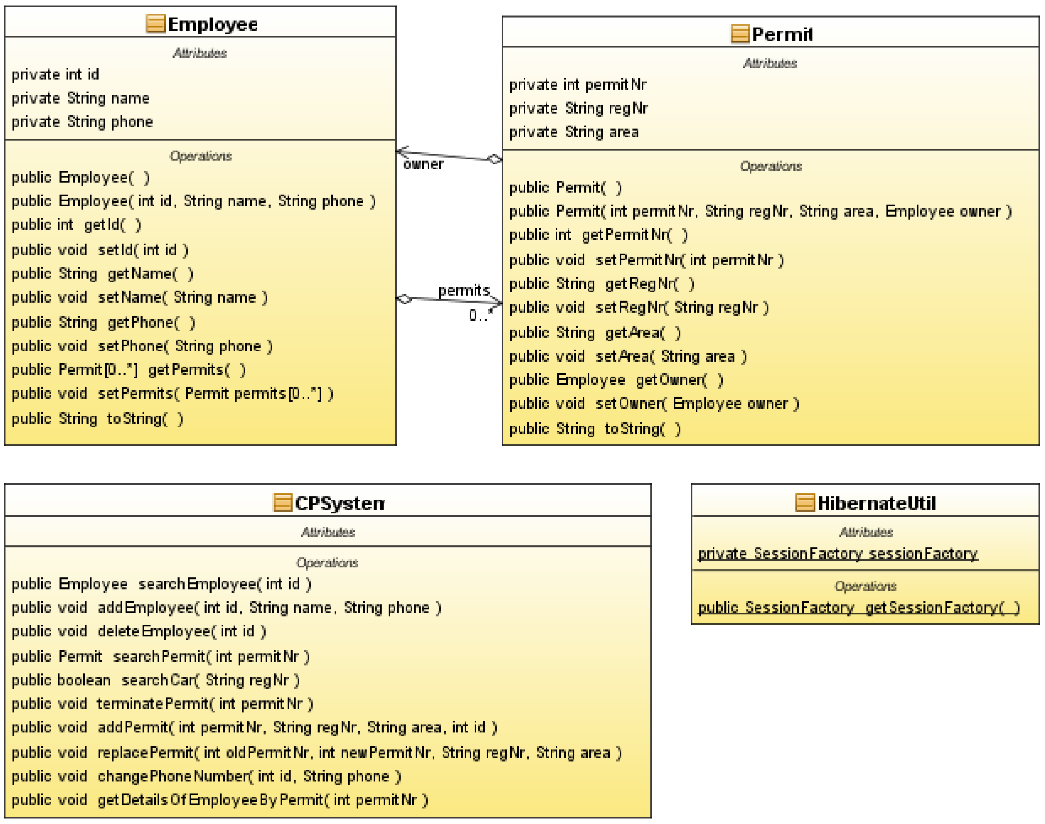
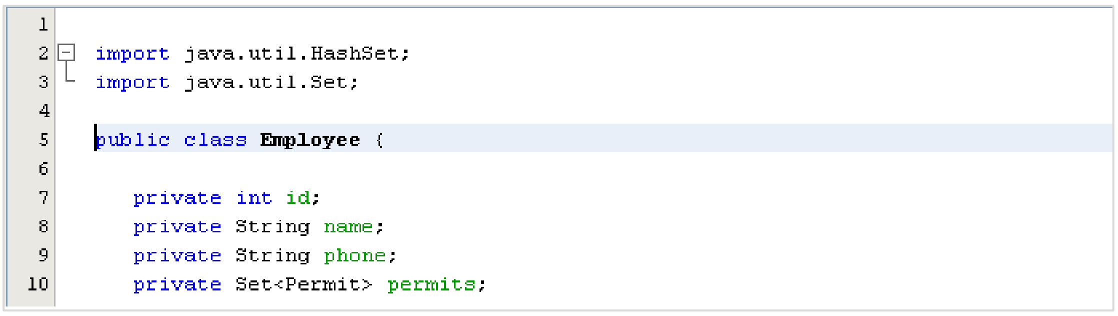

Employee
Attributes
private int id
private String name
private String phone
Operations
public Employee( )
public Employee(int id, String name, String phone)
public int getId( )
public void setId(int id)
public String getName( )
public void setName(String name)
public String getPhone( )
public void setPhone(String phone)
public Permit[0..*] getPermits( )
public void setPermits(Permit permits[0..*])
public String toString( )
owner
permits
0..*
Permit
Attributes
private int permitNr
private String regNr
private String area
Operations
public Permit( )
public Permit(int permitNr, String regNr, String area, Employee owner)
public int getPermitNr( )
public void setPermitNr(int permitNr)
public String getRegNr( )
public void setRegNr(String regNr)
public String getArea( )
public void setArea(String area)
public Employee getOwner( )
public void setOwner(Employee owner)
public String toString( )
CPSystem
Attributes
Operations
public Employee searchEmployee(int id)
public void addEmployee(int id, String name, String phone)
public void deleteEmployee(int id)
public Permit searchPermit(int permitNr)
public boolean searchCar(String regNr)
public void terminatePermit(int permitNr)
public void addPermit(int permitNr, String regNr, String area, int id)
public void replacePermit(int oldPermitNr, int newPermitNr, String regNr, String area)
public void changePhoneNumber(int id, String phone)
public void getDetailsOfEmployeeByPermit(int permitNr)
HibernateUtil
Attributes
private SessionFactory sessionFactory
Operations
public SessionFactory getSessionFactory( )

Figure 7. UML class diagram of the Car Park System

5.3.1.2.Step 2: Create POJO classes.

In this step, we will create the classes that needs to be persisted ,which are Employee and Permit class. The following code fragment show Employee class. For the rest of the source code , please , refer to the appendix.



```

import java.util.HashSet;
import java.util.Set;

public class Employee {

    private int id;
    private String name;
    private String phone;
    private Set<Permit> permits;
```

Figure 8. Code fragment from an Employee class .

5.3.1.3.Step 3: Create a data

In this step, we will use MySQL Database to store information about Car park System. So , we need to create *Employees* and *Permits* tables to refer to the classes that needs to be persisted ,which are *Employee* and *Permit* class.

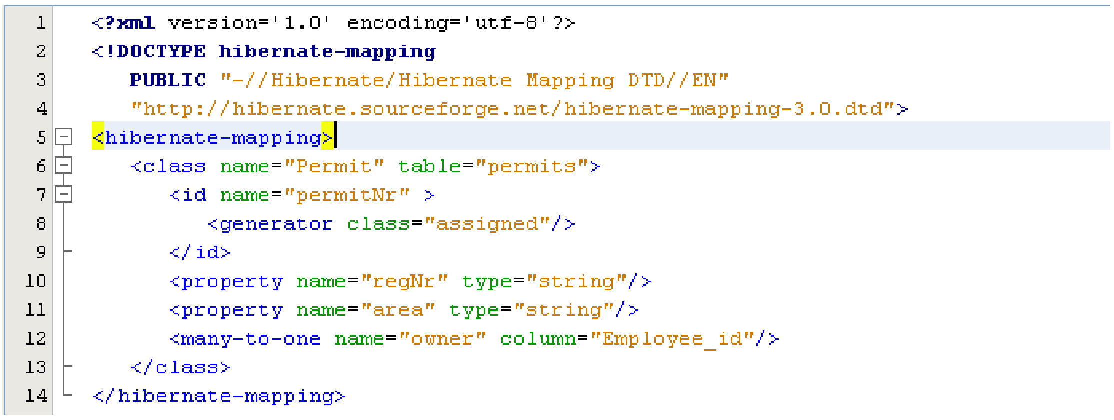
```
CREATE TABLE `employees` (
  `Employee_id` int(11) NOT NULL,
  `Employee_name` varchar(50) NOT NULL,
  `phone` varchar(50) NOT NULL,
  PRIMARY KEY (`Employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `permits` (
  `permitNr` int(11) NOT NULL,
  `regNr` varchar(50) NOT NULL,
  `area` varchar(50) NOT NULL,
  `Employee_id` int(11) NOT NULL,
  PRIMARY KEY (`permitNr`),
  KEY `Employee_id` (`Employee_id`),
  CONSTRAINT `permits_ibfk_1` FOREIGN KEY (`Employee_id`) REFERENCES
`employees` (`Employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figure 9 .Create employees and permits table.

5.3.1.4. Step 4: Writing the mapping file

For an object to be persisted to a database, Hibernate needs a mapping file for all the objects that are to be persisted. This XML file lets Hibernate know how to load and store objects, what table in the database it has to access, and what columns in the table it should use. The following is an example mapping file from the car park system application.



```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Permit" table="permits">
      <id name="permitNr" >
         <generator class="assigned"/>
      </id>
      <property name="regNr" type="string"/>
      <property name="area" type="string"/>
      <many-to-one name="owner" column="Employee_id"/>
   </class>
</hibernate-mapping>
```

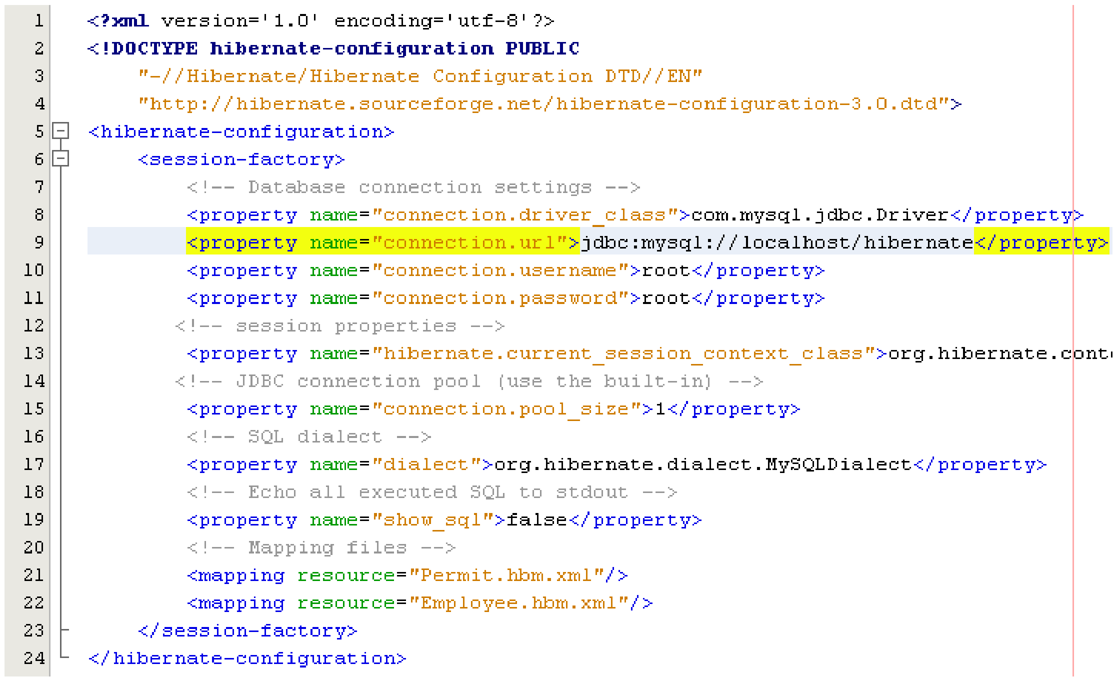
Figure 10. Permit.hbm.xml

The Hibernate-mapping element is the root element. The class element is used to map the Java class with the database table. The Java class name is specified using the name attribute of the class element and the database table name is specified using the table attribute of the class element. The meta element is used to create the class description.

The id element is used to specify the primary key. The name attribute of the id element refers to the property in the Permit class and the column attribute refers to the column in the Permits table. The type attribute holds the Hibernate mapping type. This mapping type will convert from Java to SQL data type and vice versa. The property element is used to link a property in the Java class to a column in the database table.

5.3.1.5.Step 4: Creating a configuration file for a database

Once the mapping files for the persistent objects have been completed, it is now time to configure Hibernate. This will have all the information to connect Hibernate to a database. To configure Hibernate there are a few ways in which this can be done. Firstly, we can use a simple Hibernate.properties file, a more sophisticated Hibernate.cfg.xml file, or a complete programmatic setup. For our application we will use the XML configuration file.



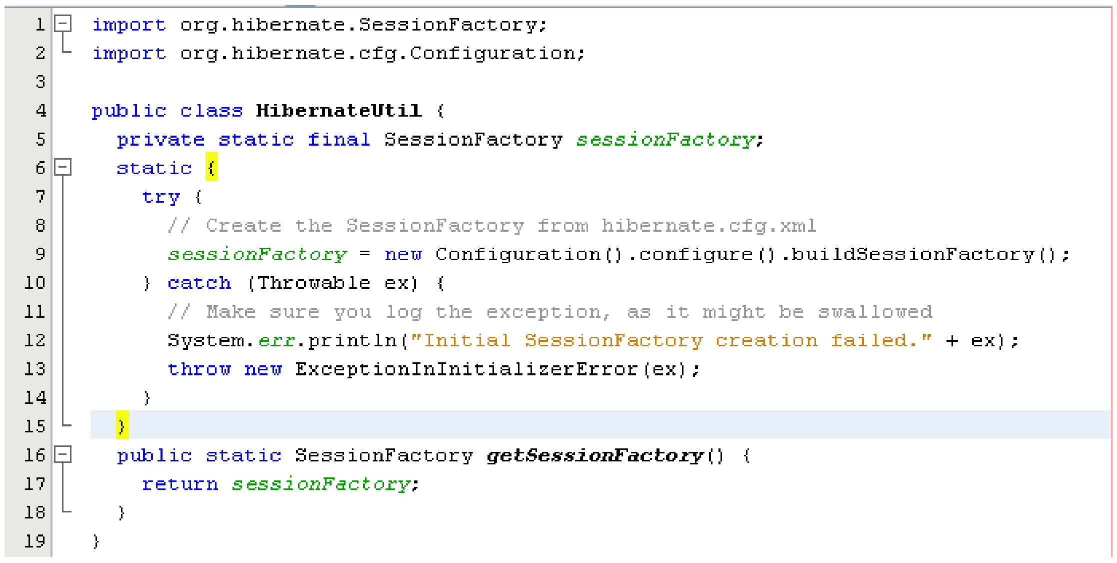
```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost/hibernate</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
       <!-- session properties -->
        <property name="hibernate.current_session_context_class">org.hibernate.cont
       <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>
        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">false</property>
        <!-- Mapping files -->
        <mapping resource="Permit.hbm.xml"/>
        <mapping resource="Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Figure 11. Hibernate.cfg.xml

Note that this configuration uses a different DTD (Document Type Definition) than the mapping files. All of the properties are inside the session-factory tags. The first four property elements contain the configuration for the MySQL connection. The dialect property element specifies the particular SQL variant Hibernate generates. The mapping files are included in the configuration at the end

5.3.1.6.Step 5: Create the HibernateUtil Helper Class

Now a helper class is needed to get Hibernate up and running. This class creates a SessionFactory object which in turn can open up new session's. A session is a single-threaded unit of work, and the SessionFactory is a thread-safe global object instantiated once. For our application the HibernateUtil class is implemented as shown below:



```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
  private static final SessionFactory sessionFactory;
  static {
    try {
      // Create the SessionFactory from hibernate.cfg.xml
      sessionFactory = new Configuration().configure().buildSessionFactory();
    } catch (Throwable ex) {
      // Make sure you log the exception, as it might be swallowed
      System.err.println("Initial SessionFactory creation failed." + ex);
      throw new ExceptionInInitializerError(ex);
    }
  }
  public static SessionFactory getSessionFactory() {
    return sessionFactory;
  }
}
```

Figure 12. HibernateUtil.java

5.3.1.7. Step 6: Create a Class to use Persistent Objects

Now that the mapping and configuration files have been written it is time to use the persistent objects in our application.Using Hibernate it is possible to add, edit, or delete an object to and from a database. In our application we use one class that facilitates all these functionalities called CPSystem.java.

5.3.1.7.1. Implement Add Employee use case

To implement Add Employee use case , we can reason what we need for each class to support it.

First, for CPSystem class we need a method to add a new employee. This method “AddEmployee” needs to check the precondition that the employee's id is new. Therefore, we need a search method, which will search for an employee with that id and return the Employee object if it exists or a null object otherwise.

Next, the addEmployee methods need to create an Employee object and save it. Therefore, we need to add a constructor method for the Employee class, which has been created in step 2.

The following code portion from the CPSystem and Employee class shows how we implement the Add Employee use case.

```
import java.util.List;
import org.Hibernate.*;
public class CPSystem
{           // attribute is omitted
    public Employee searchEmployee(int id){
            Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Employee employee = (Employee)session.get(Employee.class,id);
        return employee;
    }

   public void addEmployee(int id, String name, String phone)
   {
      //check precondition
       Employee employee = searchEmployee(id);
       boolean pre1 = (employee == null);

      if(!pre1)
      {
         System.out.println("ERROR: The id already exists!");
         return;
      }
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        employee = new Employee(id,name,phone);
        session.save(employee);
        session.getTransaction().commit();
}

}
public class Employee {
   private int id;
   private String name;
   private String phone;
   private Set<Permit> permits;

    public Employee(){

    }
    public Employee(int id, String name, String phone) {
        this.id = id;
        this.name = name;
        this.phone = phone;
        this.permits = new HashSet<Permit>();
    }

public String toString(){. . .}
. . .
}
```

Figure 13. Code portion from the CPSystem class

First , we create a new Session instance which is returned from our helper class HibernateUtil. By obtaining it from the helper class we can be sure that only one thread is running at one time .In our addEmployee() method we use the ssion.save(employee) method. This actually creates an SQL command to insert the new employee object into the database. After the object is saved, the Session is then committed.

5.3.1.7.2. Other Use Cases

We can repeat the same process for the rest of the other use cases.

5.3.2. Builing a JDBC application

5.3.2.1.Step 1: Create a database and tables.

```
CREATE TABLE `employees` (
  `Employee_id` int(11) NOT NULL,
  `Employee_name` varchar(50) NOT NULL,
  `phone` varchar(50) NOT NULL,
  PRIMARY KEY (`Employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `permits` (
  `permitNr` int(11) NOT NULL,
  `regNr` varchar(50) NOT NULL,
  `area` varchar(50) NOT NULL,
  `Employee_id` int(11) NOT NULL,
  PRIMARY KEY (`permitNr`),
  KEY `Employee_id` (`Employee_id`),
  CONSTRAINT `permits_ibfk_1` FOREIGN KEY (`Employee_id`) REFERENCES
`employees` (`Employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figure 14. Create employees and permits table.

Note: this step is the same as for the Hibernate application.

5.3.2.2.Step 2: Create Java JDBC application.

In the java JDBC application, we use one class that facilitates all the functionality called CPSystem.java. In order, to implement Add Employee use case , we need a method to add a new employee. This method "AddEmployee" needs to check the precondition that the employee's id is new. Therefore, we need a search method, which will search for an employee with that id and return the Employee object if it exists and a null object otherwise. The following code portion from the CPSystem class shows how to implement the Add Employee use case.

```
import java.sql.*;

public class CPSystem
{

// Establishing a Connection is omitted
. . .
   public boolean searchEmployee(int id){

     String sqlStmt = "SELECT Employee_id FROM Employees WHERE Employee_id = ?";
     boolean isExist = true;
     try     {
          PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
          pstmt.setInt(1,id);
          ResultSet rs = pstmt.executeQuery();
          isExist = rs.next();
      }
      catch( SQLException e ) {
          System.out.println(e.getMessage());
          e.printStackTrace();
       }

       return isExist;
    }

   public void addEmployee(int id, String name, String phone)
   {
      //check precondition

       boolean pre1 = searchEmployee(id);

      if(pre1)
      {
         System.out.println("ERROR: The id already exists!");
         return;
      }
      try {
       String sqlStmt="INSERT into Employees (Employee_id,Employee_name,phone) VALUES(?,?,?)";
       PreparedStatement pStmt = con.prepareStatement(sqlStmt);
       pStmt.setInt(1, id);
       pStmt.setString(2, name);
       pStmt.setString(3, phone);
       pStmt.executeUpdate();
      } catch (SQLException e) {
              System.out.println(e.getMessage());
             e.printStackTrace();
    }
}
. . .
}
```

Figure 15. Code portion from JDBC application

A PreparedStatement is used for SQL statements that are executed multiple times with different values. For instance, you might want to insert several values into a table, one after another. Notice that the first parameter passed to a set method indicates the specific placeholder parameter (the question marks) that you are setting. A value of 1 corresponds to the first question mark, a value of 2 corresponds to the second, and so on.

5.3.2.2.1. Other use cases

We can repeat the same process for the rest of the other use cases.

5.4. Comparison of the two approaches

Table 1. Summary of the findings of the Initial Construction

Program	File	SLOC	Total/Lines	Estimated Time
Hibernate	CPSystem.java	141	251	4 h 30 m
	Permit.java	47		
	Employee.java	47		
	HibernateUtil.java	16		
	Employee.hbm.xml	17	49	
	Permit.hbm.xml	14		
	Hibernate.cfg.xml	18		
	Total	300		
JDBC	CPSystem.java	283	283	3 h

The table above summarizes the findings of the initial construction of the car park system with two approaches. The table shows that, even though there are no significant differences between the two approaches according to the effort measured by size of source code, the approach with Hibernate took more time than using JDBC. Actually, with Hibernate we had to deal with six files whereas we had to deal with one file with JDBC. Therefore, the approach with Hibernate took about 4.30 hour compared to 3 hour using JDBC.

5.5. Impact of requirements change on the two approaches

In practice, the requirements change frequently. Therefore, it is useful to see how different approaches cope with requirements changes. As the initial investigation regarding changes of requirements, we make the following change: in the Terminate permit use case , instead of deleting the permit (as we did before) , we will make the permit as being terminated.

5.5.1.The use cases need to be modified

Use Case Name: Terminate Permit
Trigger/Goal: **To mark it as being terminated.**
 Actors: Operator.
Main Flow:

1. Operator enters the permit number.
2. System validates that the permit exists.
3. **System makes the permit as being terminated.**

Extensions:

2a. Permit number does not exist:

2a1. System notifies Operator, and terminates the use case.

5.5.2.Making changes using Hibernate to apply this modification :

1. Add a new column to permit table.
2. Add a new attribute to permit class and define getter and setter methods to it (8 lines).
3. Modify the mapping file of permit to map a new attribute(1lines).
4. Modify CPSystem.java class (16 lines) :
   a. Modify terminatePermit method.
   b. Add new method getCoutnOfPermits to check the owner currently has less than 2 permits (size of set is no longer useful).

5.5.3.Making changes using JDBC to apply this modification :

1. Add new column to permit table.
2. Modify CPSystem.java class(3 lines). :
   a. Modify terminatePermit method (change the SQL statement to be update inserted of delete) .
   b. Modify countOfPermits method (add new parameter to WHERE clause).

5.5.4.Comparison of the two approaches

Table 2. Summary of the findings after the first change requirement.

Program	File	V(1)	V(2)	Added	Modified	Deleted	A+M+D	Estimated Time
Hibernate	CPSystem.java	141	149	**10**	**4**	**2**	**16**	**40** minutes
	Permit.java	47	65	**8**	**0**	**0**	**8**	
	Employee.java	47	47	0	0	0	0	
	Employee.hbm.xml	17	17	0	0	0	0	
	Permit.hbm.xml	14	15	**1**	**0**	**0**	**1**	
	Hibernate.cfg.xml	18	18	0	0	0	0	
	HibernateUtil.java	16	16	0	0	0	0	
	Total	300	327	**19**	**4**	**2**	**25**	

JDBC	CPSystem.java	283	283	0	**3**	0	**3**	**10** minutes

As we can see from the above table, there are significant differences between the two approached according to the effort measured by the size of the source code. The implementation of the new changes in requirements with Hibernate required a total of 25 lines of code, compared to only 3 lines of code using JDBC. In addition, the implementation of the new changes in requirements with Hibernate took about 40 minutes, whereas it only took 3 minutes with JDBC. Indeed, it can be seen that, using JDBC offered more flexibility in regard to time and effort.

5.5.5. Changes to Hibernate's application code to apply this modification

We Added a new attribute with Permit class with its getter and setter methods.

```
public class Permit {
. . .
  private int terminate;
   public void setTerminate(int terminate) {
       this.terminate = terminate;
   }
   public int getTerminate() {
       return terminate;
   }
. . .
}
```

Figure 16. Code portion from Permit class.

We Added a new method to CPSysteme to get the count of employee's permits

```
   public int  getCoutnOfPermits(Set<Permit> permits) {
       int count = 0;
       for(Permit p:permits){
        if(p.getTerminate()==0)
           ++count;
       }
       return count;
   }
```

Figure 17. Impact of change in CPSysteme class.

5.5.6. Changes to JDBC's application code to apply this modification

The terminatePermit method before and after the changes, respectively.

```
public  void terminatePermit (int permitNr)  {
. . .
String sqlStmt="DELETE FROM permits WHERE permitNr = ?";
. . .
}

public void terminatePermit (int  permitNr)  {
. . .
String sqlStmt="UPDATE permits SET terminate=1 WHERE permitNr = ?";
. . .
}
```

Figure 18. Impact of change in terminatePermit method

The countOfPermits method before and after the changes, respectively.

```
public int countOfPermits(int id)  {
. . .
String sqlStmt = "SELECT count(*) As rowCount FROM permits WHERE Employee_id = ?";
.  . .
}

public int countOfPermits(int id)  {
.  . .
String sqlStmt = "SELECT count(*) As rowCount FROM permits WHERE Employee_id = ? And
terminate = 0";
. . .
}
```

Figure 19. Impact of changes in countOfPermits method.

5.6. Further impact of requirements change on the two approaches

In the second change of requirements, suppose we need to distinguish between full-time and part-time employees. Part-time employees work on an hourly rate, while full-time employees are assigned a salary. Figure 1 shows the impact of requirement change on a class diagram.

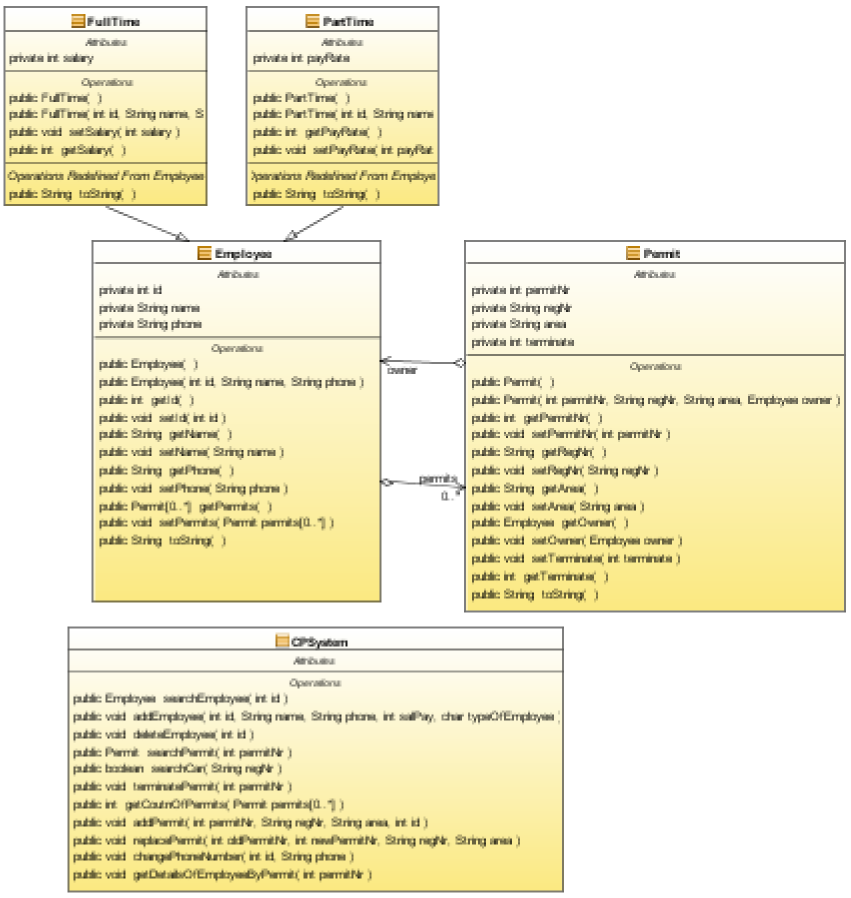

FullTime
Attributes
private int salary
Operations
public FullTime( )
public FullTime(int id, String name, S
public void setSalary(int salary)
public int getSalary( )
Operations Redefined From Employee
public String toString( )
PartTime
Attributes
private int payRate
Operations
public PartTime( )
public PartTime(int id, String name
public int getPayRate( )
public void setPayRate(int payRat
Operations Redefined From Employe
public String toString( )
Employee
Attributes
private int id
private String name
private String phone
Operations
public Employee( )
public Employee(int id, String name, String phone)
public int getId( )
public void setId(int id)
public String getName( )
public void setName(String name)
public String getPhone( )
public void setPhone(String phone)
public Permit[0..*] getPermits( )
public void setPermits(Permit permits[0..*])
public String toString( )
owner
permits
0..*
Permit
Attributes
private int permitNr
private String regNr
private String area
private int terminate
Operations
public Permit( )
public Permit(int permitNr, String regNr, String area, Employee owner)
public int getPermitNr( )
public void setPermitNr(int permitNr)
public String getRegNr( )
public void setRegNr(String regNr)
public String getArea( )
public void setArea(String area)
public Employee getOwner( )
public void setOwner(Employee owner)
public void setTerminate(int terminate)
public int getTerminate( )
public String toString( )
CPSystem
Attributes
Operations
public Employee searchEmployee(int id)
public void addEmployee(int id, String name, String phone, int salPay, char typeOfEmployee)
public void deleteEmployee(int id)
public Permit searchPermit(int permitNr)
public boolean searchCar(String regNr)
public void terminatePermit(int permitNr)
public int getCoutnOfPermits(Permit permits[0..*])
public void addPermit(int permitNr, String regNr, String area, int id)
public void replacePermit(int oldPermitNr, int newPermitNr, String regNr, String area)
public void changePhoneNumber(int id, String phone)
public void getDetailsOfEmployeeByPermit(int permitNr)

Figure 20. UML class diagram of the Car Park System

5.6.1. The use cases need to be modified

Use Case Name: Add an Employee
Trigger/Goal: To enter the details of a new employee
Actors: Operator (or car park manager, depending on assumption)
Main Flow:

1. *Operator enters employee's ID.*
2. *System validates that ID is new.*
3. *Operator enters employee's Type.*
4. *System validates that Type is valid .*
3. *Operator enters name and phone number.*
4. *System saves the details.*

Extensions:

2a. *ID is not new:*
2a1. System notifies Operator, and terminates the use case.
4a. ***Type is invalid:***
4a1. System notifies Operator, and terminates the use case.

5.6.2. Mapping UML class diagram to a relational database

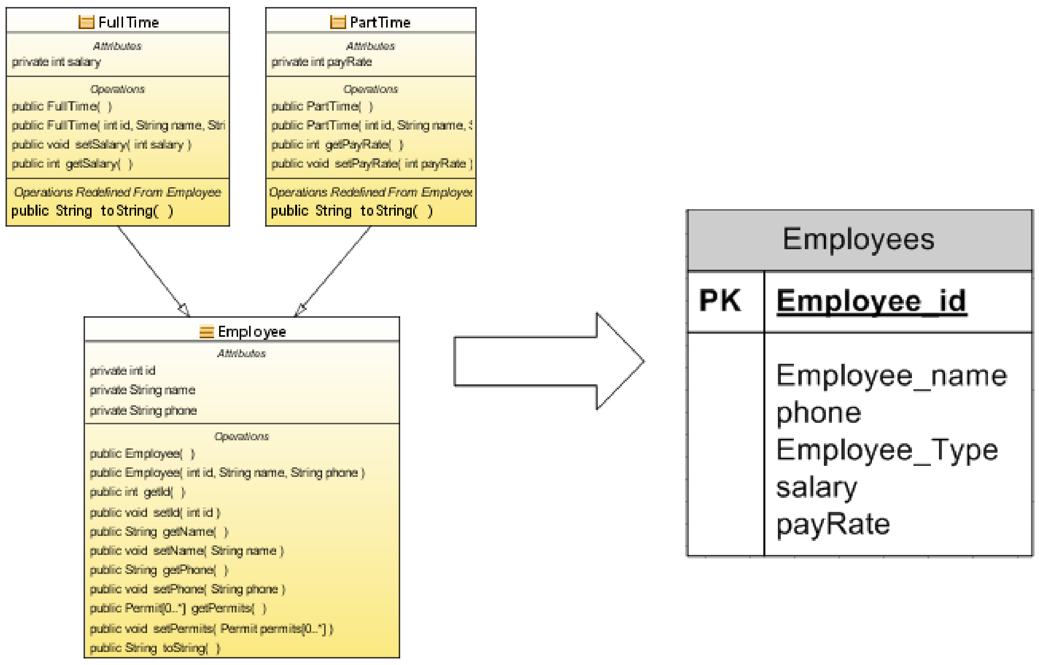

FullTime
Attributes
private int salary
Operations
public FullTime( )
public FullTime(int id, String name, Stri
public void setSalary(int salary)
public int getSalary( )
Operations Redefined From Employee
public String toString( )
PartTime
Attributes
private int payRate
Operations
public PartTime( )
public PartTime(int id, String name, :
public int getPayRate( )
public void setPayRate(int payRate)
Operations Redefined From Employee
public String toString( )
Employee
Attributes
private int id
private String name
private String phone
Operations
public Employee( )
public Employee(int id, String name, String phone)
public int getId( )
public void setId(int id)
public String getName( )
public void setName(String name)
public String getPhone( )
public void setPhone(String phone)
public Permit[0..*] getPermits( )
public void setPermits(Permit permits[0..*])
public String toString( )
Employees
PK
Employee_id
Employee_name
phone
Employee_Type
salary
payRate

Figure 21. Mapping UML class diagram to a relational database

As we can see from the above figure, we used Table per Class Hierarchy strategy to model Employee class hierarchy in the database. This is very useful when selecting rows from the database of multiple employee types. The selection goes only to this table and is therefore more efficient. Moreover, it is very simple to add a column to all the employee types.

5.6.3. Making changes using Hibernate to apply this modification :

1. As a result of using **Table per Class Hierarchy strategy**, add a new column to employees table (Employess_type).
2. Create new class for full-Time employee (20 1lines).
3. Create new class for part-Time employee (20 1lines).
4. Mapping full-Time and part-Time class as subclasses with Employee.hbm.xml (7 1lines).
5. Modify toString() method of Employee class (1 1lines).
6. Modify CPSystem.java class (7 1lines) :
    a. Modified addEmployee method to determine the type of employee.

5.6.4.Making changes using JDBC to apply this modification :

1. add a new column to the employees table (Employess_type).
2. Modify CPSystem.java class (13 1lines):
   a. Modify addEmployee method to determine the type of employee.

5.6.5. Comparison of the two approaches

Table 3. Summary of the findings after the second change requirement

Program	File	V(2)	V(3)	Added	Modified	Deleted	A+M+D	Estimated Time
Hibernate	CPSystem.java	149	154	**5**	**2**	**0**	**7**	1hour
	Permit.java	65	65	0	0	0	0	
	Employee.java	47	47	**0**	**1**	**0**	**1**	
	PartTime.java	-	20	**20**	**0**	**0**	**20**	
	FullTime.java	-	20	**20**	**0**	**0**	**20**	
	HibernateUtil.java	16	16	0	0	0	0	
	Employee.hbm.xml	17	24	**7**	**0**	**0**	**7**	
	Permit.hbm.xml	15	15	0	0	0	0	
	Hibernate.cfg.xml	18	18	0	0	0	0	
	Total	**327**	**379**	**52**	**3**	**0**	**55**	
JDBC	CPSystem.java	**283**	**296**	**13**	**3**	**0**	**16**	20 minutes

As we can see from the above table, the new requirements have had a greater impact on the program implemented through Hibernate, whether in terms of time or effort to implement these changes. The implementation of the new changes in requirements with Hibernate required a total of 55 lines of code, in contrast to JDBC which required only 16 lines of code. This difference represents nearly a 3:1 ratio in quantity of code. Although, one of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses, the majority of new code comes due to inheritance code. Moreover, the implementation with Hibernate took about one hour compared to only 20 minutes using JDBC. As a result, increasing the number of classes that need to be persisted automatically can lead to increased levels of effort and time.

5.6.6. Some of Hibernate's application code has changed to apply this modification

For ease of reference, we listing the code related to changes made.

```
public class PartTime extends Employee{

  private int payRate;

   public PartTime(){};

   public PartTime(int id, String name, String phone , int payRate) {
       super(id, name, phone);
       this.payRate = payRate;
   }

   public int getPayRate() {
       return payRate;
   }

   public void setPayRate(int payRate) {
       this.payRate = payRate;
   }
  @Override
  public String toString()
  {
       String desc = super.toString() + " , PayRate: " + payRate + " ]";
       return desc;
   }
}
```

Figure 22. Part-time class

```
public class FullTime extends Employee{

   private int salary;

   public FullTime(){};

   public FullTime(int id, String name, String phone, int salary) {
       super(id, name, phone);
       this.salary = salary;
   }

   public void setSalary(int salary) {
       this.salary = salary;
   }
   public int getSalary() {
       return salary;
   }
   @Override
   public String toString()
   {
       String desc = super.toString() + " , Salary: " + salary + " ]";
       return desc;
   }
}
```

Figure 23. full-Time class

```
<?xml version="1.0" encoding='utf-8'?>
<!DOCTYPE Hibernate-mapping PUBLIC
       "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
       "http://Hibernate.sourceforge.net/Hibernate-mapping-3.0.dtd">
<Hibernate-mapping>
   <class name="Employee" table="employees">
         . . .
   <discriminator column="Employee_type" type= "string"/>

   <subclass name="FullTime" discriminator-value="FullTime">
      <property name="salary"/>
   </subclass>

    <subclass name="PartTime" discriminator-value="PartTime">
      <property name="payRate"/>
    </subclass>

   </class>
</Hibernate-mapping>
```

Figure 24. Mapping new classes as subclasses with Employee.hbm.xml.

5.6.7. Some of JDBC's application code has changed to apply this modification

For ease of reference, we listing the code related to changes made.

```
public void addEmployee(int id, String name, String phone , String type , int salpayRate)  {
            . . .
      try {

      String sqlStmt = null;
      if(type.equals("FullTime"))
       sqlStmt="INSERT into Employees (Employee_id,Employee_name,phone,"+
                 "Employee_Type,payRate) VALUES(?,?,?,?,?)";
      else if(type.equals("PartTime"))
       sqlStmt="INSERT into Employees (Employee_id,Employee_name,phone,"+
                 "Employee_Type,salary) VALUES(?,?,?,?,?)";
      else
      {
        System.out.println("ERROR: wrong type of employee!");
        return;
      }
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setInt(1,id);
      pStmt.setString(2,name);
      pStmt.setString(3,phone);
      pStmt.setString(4,type);
      pStmt.setInt(5,salpayRate);
           . . .
}
```

Figure 25. Impact of changes in addEmployee method.

Chapter 5

Second Case Study

This case study is more complicated in order to produce more statistics to compare the two approaches. We also make changes that reflect the change in business policy, namely allowing more than one kind of items to be stored at a shelf location. This change in policy required a change in the structure of the classes. It will provide more data to compare the two approaches.

6.1. Problem Statement

A database is needed to maintain information about the items stored in various warehouses of a company. Design a relational database, which can store the information contained in the view below.

Figure 26. Screen display on where item F120 is Stored

Part#	: F120
Description	: Trek Frame
Unit	: Each
Total on Hand	: 100

Warehouse	Shelf Location	Qty	Location Type
W1	*A – 10*	*40*	*Single Access*
W1	*B – 5*	*30*	*Single Access*
W2	*B – 5*	*20*	*Double Access*
W2	*D – 8*	*10*	*Double Access*

1. Each warehouse has a phone (not shown) to contact the staff at the warehouse.
2. Shelf location codes (such as A – 10) are unique within a warehouse only.
3. Shelf locations are of two types: single access and double access.
4. The present policies require that each shelf location, at any time, can be used to store only one kind of items.

6.2. Use Case Description

1. Add a new Warehouse.
2. Add a new Part.
3. Add a new Shelf Location.

4. Store a Part.
5. Retrieve the details where part is stored.

Due to the maximum limit of papers we shift the use case description to the appendix.

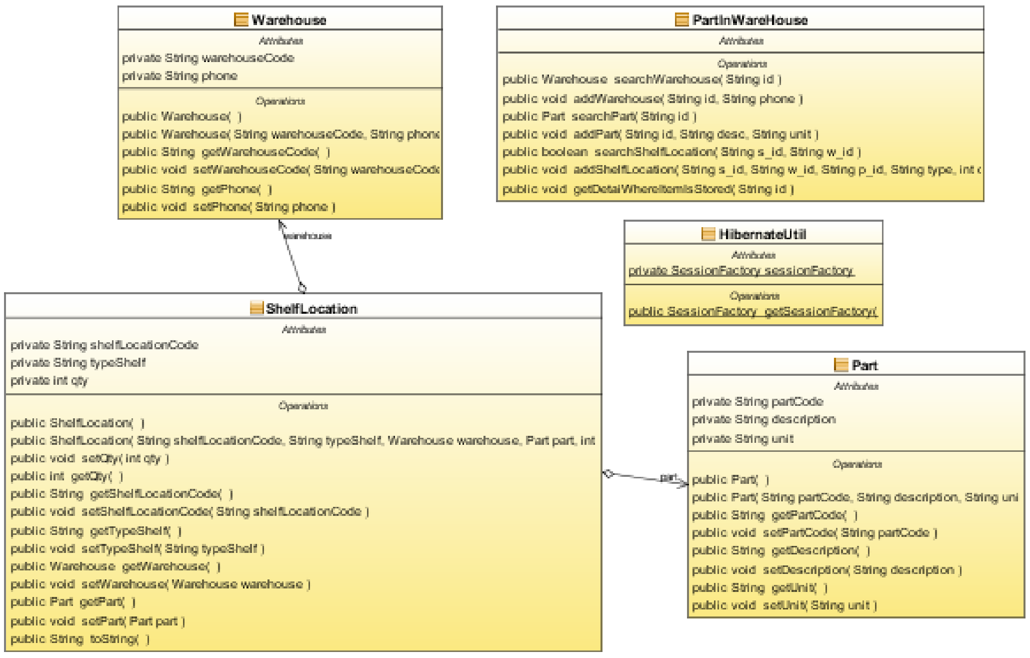

Warehouse
Attributes
private String warehouseCode
private String phone
Operations
public Warehouse( )
public Warehouse(String warehouseCode, String phone
public String getWarehouseCode( )
public void setWarehouseCode(String warehouseCode
public String getPhone( )
public void setPhone(String phone)
PartInWareHouse
Attributes
Operations
public Warehouse searchWarehouse(String id)
public void addWarehouse(String id, String phone)
public Part searchPart(String id)
public void addPart(String id, String desc, String unit)
public boolean searchShelfLocation(String s_id, String w_id)
public void addShelfLocation(String s_id, String w_id, String p_id, String type, int
public void getDetailWhereItemIsStored(String id)
HibernateUtil
Attributes
private SessionFactory sessionFactory
Operations
public SessionFactory getSessionFactory(
warehouse
ShelfLocation
Attributes
private String shelfLocationCode
private String typeShelf
private int qty
Operations
public ShelfLocation( )
public ShelfLocation(String shelfLocationCode, String typeShelf, Warehouse warehouse, Part part, int
public void setQty(int qty)
public int getQty( )
public String getShelfLocationCode( )
public void setShelfLocationCode(String shelfLocationCode)
public String getTypeShelf( )
public void setTypeShelf(String typeShelf)
public Warehouse getWarehouse( )
public void setWarehouse(Warehouse warehouse)
public Part getPart( )
public void setPart(Part part)
public String toString( )
part
Part
Attributes
private String partCode
private String description
private String unit
Operations
public Part( )
public Part(String partCode, String description, String uni
public String getPartCode( )
public void setPartCode(String partCode)
public String getDescription( )
public void setDescription(String description)
public String getUnit( )
public void setUnit(String unit)

Figure 27. UML class diagram of the Part in Warehouse

6.3.Comparison of the two approach

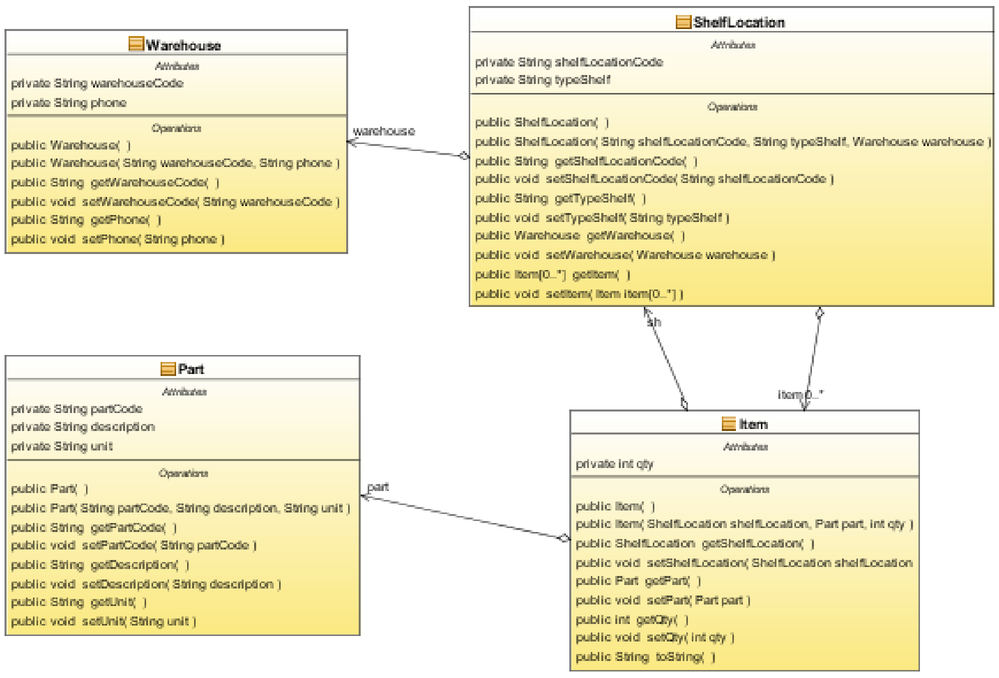
Table 4. Summary the findings of the initial construction of PWS

Program	File	SLOC	Total/Lines	Estimated Time
Hibernate	PartInWareHouse.java	141	300	4 hour
	Part.java	30		
	Warehouse.java	31		
	ShelfLocation.java	45		
	ShelfLocationPK.java	37		
	HibernateUtil.java	16		
	Warehouse.hbm.xml	12	58	
	Part.hbm.xml	13		
	ShelfLocation.hbm.xml	15		
	Hibernate.cfg.xml	18		
	Total	358		
JDBC	PartInWareHouse.java	202		2.30 hour

The table above summarizes the findings to implement the Parts in Warehouses with two approaches. Hibernate required a total of 358 lines of code, in contrast to JDBC which required 202 lines. In addition, the Hibernate required about 4 hour whereas using JDBC required 2.30 hour. It can be seen that, Hibernate required more effort and time than JDBC.

6.4. Impact of requirements change on the two approaches

The storage rules change to allow more than one kind of items to be stored at a shelf location.



Warehouse
Attributes
private String warehouseCode
private String phone
Operations
public Warehouse( )
public Warehouse(String warehouseCode, String phone)
public String getWarehouseCode( )
public void setWarehouseCode(String warehouseCode)
public String getPhone( )
public void setPhone(String phone)
warehouse
ShelfLocation
Attributes
private String shelfLocationCode
private String typeShelf
Operations
public ShelfLocation( )
public ShelfLocation(String shelfLocationCode, String typeShelf, Warehouse warehouse)
public String getShelfLocationCode( )
public void setShelfLocationCode(String shelfLocationCode)
public String getTypeShelf( )
public void setTypeShelf(String typeShelf)
public Warehouse getWarehouse( )
public void setWarehouse(Warehouse warehouse)
public Item[0..*] getItem( )
public void setItem(Item item[0..*])
sh
item 0..*
Part
Attributes
private String partCode
private String description
private String unit
Operations
public Part( )
public Part(String partCode, String description, String unit)
public String getPartCode( )
public void setPartCode(String partCode)
public String getDescription( )
public void setDescription(String description)
public String getUnit( )
public void setUnit(String unit)
part
Item
Attributes
private int qty
Operations
public Item( )
public Item(ShelfLocation shelfLocation, Part part, int qty)
public ShelfLocation getShelfLocation( )
public void setShelfLocation(ShelfLocation shelfLocation
public Part getPart( )
public void setPart(Part part)
public int getQty( )
public void setQty(int qty)
public String toString( )

Figure 28. Impact of change on class diagram of the Part in Warehouse System.

6.4.1.The use cases need to be modified

1. <u>Use Case Name</u>: Store Part.

<u>Trigger/Goal</u>: To Store the quantity of a part to existing Shelf Location .

<u>Actors:</u> Operator.

<u>Main Flow</u>:

1. Operator enters the Shelf Location's ID and part's ID.
2. The System validates that ID's exists.
3. **The System validates that compost key (ID's) is new.**

3. Operator enters quantity of that part.

4. **System save the details.**

Extensions:

2a. Shelf Location's ID does not exist:
   2a1. System notifies Operator, and terminates the use case.

2b. part's ID does not exist:
   2b1. System notifies Operator, and terminates the use case.

2a. Item ID does not exist:
   2a1. System notifies Operator, and terminates the use case.

Use Case Name: Get the Details where Part is Stored.

Trigger/Goal: The operator wants find out where the part is stored , given the product ID.

Actors: Operator.

Main Flow:

1. Operator enters part's ID.
2. **System validates that ID is exist and the set of items is not empty for that part**.
3. System retrieves and displays the part's' id, description and unit.
   --- System Do steps 4 for each of the Items.
4. System retrieves the warehouse's ID , Shelf Location's ID , quantity , Shelf Location's type
5. System displays the details list.

Extensions:

2a. ID does not exist:
   2a1. System notifies Operator, and terminates the use case.

2b. **The Item set is empty for a particular part** :
   2b1. System notifies Operator, and terminates the use case.

6.4.2. Making changes using Hibernate to apply this modification :

1. *Create new table called Items .*
2. *Create Item class (291lines).*
3. *Create ItemPK class as composite key for Item class (37 lines).*
7. *Create new file Mapping clalled Item.hbm.xml (16 lines).*
8. *Add Item.hbm.xml mapping to Hibernate.cfg.xml (1line).*
9. *Modify ShelfLocation class (22 lines):*
   a. *Add new Attribute to represent a one-to-many relationship with Item class with its getter and setter methods.*
   b. *Delete some attribute such as quantity of part with its getter and setter methods.*
4. *Modify ShelfLocation.hbm.xml mapping (8lines):*
   a. *to represent a one-to-many relationship with Item class.*
   b. *delete mapping of some attribute such as quantity.*
5. *Modify PartInWareHouse.java class (19 lines) :*

a. *Add new search method for item.*
b. *Modify storPart method.*
c. *Modify getDetaiWhereItemIsStored method.*

6.4.3.Making changes using JDBC to apply this modification :

1. *Create new table called Items (this step same as Hibernate) .*
2. *Modify PartInWareHouse.java class (38lines):*
    a. *Add new search method for item.*
    b. *Modify getDetaiWhereItemIsStored method.*
    c. *Modify storPart method.*

6.4.4.Comparison of the two approach

Table 5. Summary the findings after change requirement

Program	File	V(1)	V(2)	Added	Modified	Deleted	A+M+D	Estimated Time
Hibernate	PartInWareHouse.java	141	**155**	**14**	**5**	**0**	**19**	**1** h **30** m
	Part.java	30	30	0	0	0	0	
	Warehouse.java	31	31	0	0	0	0	
	ShelfLocation.java	45	**32**	**2**	**6**	**15**	**23**	
	ShelfLocationPK.java	37	37	0	0	0	0	
	Item.java	**-**	**29**	**29**	**0**	**0**	**29**	
	ItemPK.java	**-**	**37**	**37**	**0**	**0**	**37**	
	HibernateUtil.java	16	16	0	0	0	0	
	Warehouse.hbm.xml	12	12	0	0	0	0	
	Part.hbm.xml	13	**19**	**6**	**0**	**0**	**6**	
	ShelfLocation.hbm.xml	15	**20**	**5**	**2**	**1**	**8**	
	Item.hbm.xml	**-**	**16**	**16**	**0**	**0**	**16**	
	Hibernate.cfg.xml	18	**19**	**1**	**0**	**0**	**1**	
	Total	**358**	**453**	**110**	**13**	**16**	**139**	
JDBC	PartInWareHouse.java	**202**	**218**	**16**	**20**	**2**	**38**	**30** minutes

As we can see from the above table. the new requirements have had a greater impact on the program implemented through Hibernate, whether in terms of time or effort to implement these changes. The implementation of the new changes in requirements with Hibernate required a total of 139 lines of code, in contrast to JDBC which required only 38 lines of code. This difference represents nearly a 4:1 ratio in quantity of code.

Indeed, the source of increase in the number of codes was due to added a Item class with its composite key, which are not necessary in JDBC. Moreover, the implementation with Hibernate took about 1.30 hour compare to only 30 minutes using JDBC.

6.4.5. Some of Hibernate's application code has changed to apply this modification

For ease of reference, we listing the code related to changes made.

```
import java.io.Serializable;
public class ItemPK implements Serializable {
   private ShelfLocation sh;
   private Part part;

   public ItemPK() {
   }

  public ItemPK(ShelfLocation sh, Part part) {
      this.sh = sh;
      this.part = part;
   }
   public ShelfLocation getSh() {
      return sh;
   }
   public void setSh(ShelfLocation sh) {
      this.sh = sh;
   }
   public Part getPart() {
      return part;
   }
   public void setPart(Part part) {
      this.part = part;
   }

   @Override
   public boolean equals(Object obj) {
   if (!(obj instanceof ItemPK)) return false;
   ItemPK other = (ItemPK) obj;
   return this.part.equals(other.part) &&
        this.sh.equals(other.sh);
   }

   @Override
   public int hashCode() {
      int hash = 3;
      hash = 37 * hash + (this.sh != null ? this.sh.hashCode() : 0);
      hash = 37 * hash + (this.part != null ? this.part.hashCode() : 0);
      return hash;
   }}
```

Figure 29. Composite primary key of Item class

```
public class Item  {

   private ItemPK primarykey;
   private int qty;

   public Item() {
   }

   public Item(ItemPK primarykey, int qty) {
       this.primarykey = primarykey;
       this.qty = qty;
   }

   public ItemPK getPrimarykey() {
       return primarykey;
   }

   public void setPrimarykey(ItemPK primarykey) {
       this.primarykey = primarykey;
   }

   public int getQty() {
       return qty;
   }

   public void setQty(int qty) {
       this.qty = qty;
   }

   @Override
   public String toString(){
     String x = primarykey.getSh().getPrimarykey().getWarehouse().getWarehouseCode()+
          "\t\t"+ primarykey.getSh().getPrimarykey().getShelfLocationCode()+
          "\t\t"+qty+"\t\t"+primarykey.getSh().getTypeShelf()+"\n";
     return x;
   }
}
```

Figure 30. Item class

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE Hibernate-mapping
  PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://Hibernate.sourceforge.net/Hibernate-mapping-3.0.dtd">
<Hibernate-mapping>
  <class name="Item" table="item">
  <composite-id name="primarykey" class="ItemPK">
      <key-many-to-one name="sh" class="ShelfLocation">
              <column name="W_ID"/>
              <column name="S_ID"/>
          </key-many-to-one>
      <key-many-to-one name="part" class="Part" column="P_ID"/>
  </composite-id>
  <property name="qty" column="QTY"/>
   </class>
</Hibernate-mapping>
```

Figure 31. Map a composite primary key of Item class

```
public Item searchItem(ItemPK id)   {
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    Item item = (Item)session.get(Item.class,id);
    return item;
}
```

Figure 32. SearchItem method with Hibernate

6.4.6.Some of JDBC's application code has changed to apply this modification

For ease of reference, we listing the code related to changes made.

```
public boolean searchItem(String w_id , String s_id , String p_id){

 String sql = "SELECT * FROM item WHERE W_ID = ? AND S_ID = ? AND P_ID = ?";
 boolean isExist = true;
 try
 {
      PreparedStatement pstmt  = con.prepareStatement(sql);
      pstmt.setString(1,w_id);
      pstmt.setString(2,s_id);
      pstmt.setString(3,p_id);
      ResultSet rs = pstmt.executeQuery();
      isExist = rs.next();
 }
 catch( SQLException e ) {
      System.out.println(e.getMessage());
      e.printStackTrace();
   }
  return isExist;
}
```

Figure 33. searchItem method with JDBC

Chapter 6
Third case study

Issue of relational representation/Navigation

Representation of the relationship is fundamental issue. In fact, the different between hierarchy, network, relational and object oriented database is the way to represent the relationship. Therefore, if we construct the application with JDBC, we will not experience the problem of navigation whereas the problem arises when choose to construct the application with ORM. Thus, we have to make decision how can represent the navigation objects. This we going to discuses in this chapter.

7.1 Problem Statement

A distribution company supplies various kinds of products to customers on a daily basis according to the standing orders placed by the customers. The company wants to set up a system to maintain information about the products that the company can supply, its customers, the standing orders.

7.2 Use Case Description

1. Add an Customer
2. Add new Product
3. Add a standing order.

   Due to the maximum limit of papers we shift the use case description to the appendix.

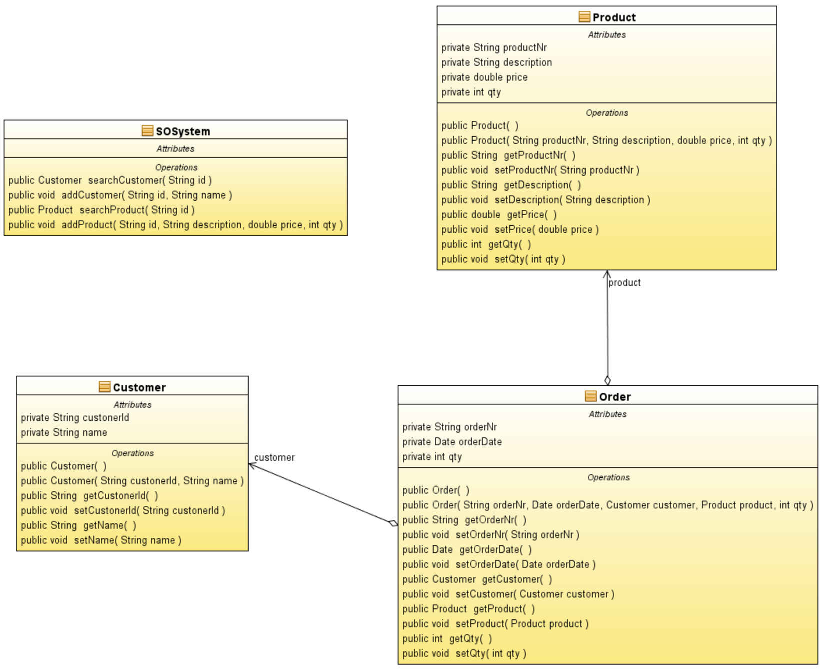

Product
Attributes
private String productNr
private String description
private double price
private int qty
Operations
public Product( )
public Product(String productNr, String description, double price, int qty)
public String getProductNr( )
public void setProductNr(String productNr)
public String getDescription( )
public void setDescription(String description)
public double getPrice( )
public void setPrice(double price)
public int getQty( )
public void setQty(int qty)
SOSystem
Attributes
Operations
public Customer searchCustomer(String id)
public void addCustomer(String id, String name)
public Product searchProduct(String id)
public void addProduct(String id, String description, double price, int qty)
product
Customer
Attributes
private String custonerId
private String name
Operations
public Customer( )
public Customer(String custonerId, String name)
public String getCustonerId( )
public void setCustonerId(String custonerId)
public String getName( )
public void setName(String name)
customer
Order
Attributes
private String orderNr
private Date orderDate
private int qty
Operations
public Order( )
public Order(String orderNr, Date orderDate, Customer customer, Product product, int qty)
public String getOrderNr( )
public void setOrderNr(String orderNr)
public Date getOrderDate( )
public void setOrderDate(Date orderDate)
public Customer getCustomer( )
public void setCustomer(Customer customer)
public Product getProduct( )
public void setProduct(Product product)
public int getQty( )
public void setQty(int qty)

Figure 34. UML class diagram of the Standing Order System .

7.3 Comparison of the two approach

Table 6. Summary the findings of the initial construction

Program	File	SLOC	Total/Lines	Estimated Time
Hibernate	SOSystem.java	86	212	3 hour
	Customer.java	22		
	Order.java	47		
	Product.java	41		
	HibernateUtil.java	16		
	Customer.hbm.xml	10	56	
	Order.hbm.xml	15		
	Product.hbm.xml	12		
	Hibernate.cfg.xml	19		
	Total	268		
JDBC	SOSystem.java	141	141	2 hour

The table above summarizes the findings to implement the Standing Order System with two approaches. Hibernate required a total of 268 lines of code, in contrast using JDBC required 141 lines. In addition, the Hibernate required about 3 hour whereas using JDBC required 2hour. It can be seen that , Hibernate required more effort and time than JDBC.,

7.4 Impact of requirements change on two approaches

As we can see from the next figure, we change navigation rule between the objects from unidirectional association to bidirectional.

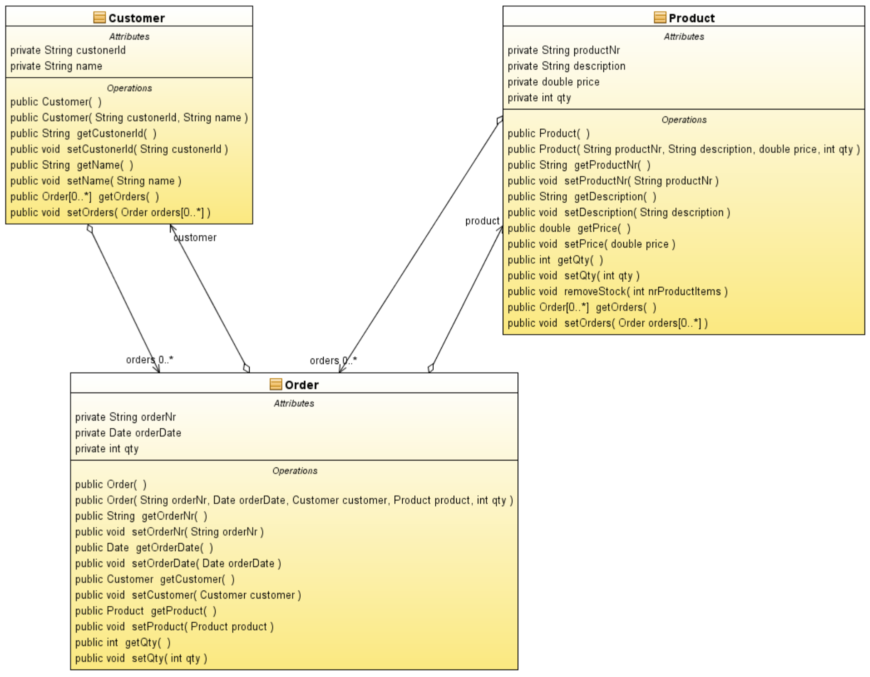

Customer
Attributes
private String custonerId
private String name
Operations
public Customer( )
public Customer(String custonerId, String name)
public String getCustonerId( )
public void setCustonerId(String custonerId)
public String getName( )
public void setName(String name)
public Order[0..*] getOrders( )
public void setOrders(Order orders[0..*])
Product
Attributes
private String productNr
private String description
private double price
private int qty
Operations
public Product( )
public Product(String productNr, String description, double price, int qty)
public String getProductNr( )
public void setProductNr(String productNr)
public String getDescription( )
public void setDescription(String description)
public double getPrice( )
public void setPrice(double price)
public int getQty( )
public void setQty(int qty)
public void removeStock(int nrProductItems)
public Order[0..*] getOrders( )
public void setOrders(Order orders[0..*])
customer
product
orders 0..*
orders 0..*
Order
Attributes
private String orderNr
private Date orderDate
private int qty
Operations
public Order( )
public Order(String orderNr, Date orderDate, Customer customer, Product product, int qty)
public String getOrderNr( )
public void setOrderNr(String orderNr)
public Date getOrderDate( )
public void setOrderDate(Date orderDate)
public Customer getCustomer( )
public void setCustomer(Customer customer)
public Product getProduct( )
public void setProduct(Product product)
public int getQty( )
public void setQty(int qty)

Figure 35.Impact the change on class diagram of the Standing Order System.

7.4.1 Making changes using Hibernate to apply this modification :

1. Modify Customer.java class (10 lines):
    a. Add new Attribute with its getter and setter methods to represent bidirectional **one-to-many** association with order class.
2. Modify Product.java class (10 lines):
    a. Add new Attribute with its getter and setter methods to represent bidirectional **one-to-many** association with order class.
3. Modify Customer.hbm.xml mapping (4 lines) :
    a. to represent a one-to-many relationship with Order class.
4. Modify Product.hbm.xml mapping (4 lines):
    a. to represent a one-to-many relationship with Order class.

7.4.2. Making change using Hibernate to apply this modification :

We don't need to do anything.

7.4.3.Comparison of the two approach

Table 7. Summary the findings after the change requirement.

Program	File	V(1)	V(2)	Added	Modified	Deleted	A+M+D	Estimated Time
Hibernate	SOSystem.java	86	86	0	0	0	0	**30**
	Customer.java	22	**32**	**10**	0	0	**10**	
	Order.java	47	47	0	0	0	0	
	Product.java	41	**51**	**10**	0	0	**10**	
	HibernateUtil.java	16	16	0	0	0	0	
	Customer.hbm.xml	10	**14**	**4**	0	0	**4**	
	Order.hbm.xml	15	**15**	**0**	0	0	**0**	
	Product.hbm.xml	12	**16**	**4**	0	0	**4**	
	Hibernate.cfg.xml	19	19	0	0	0	0	
	Total	**268**	**296**	**28**	0	0	**28**	
JDBC	SOSystem.java	141	141	**0**	**0**	**0**	**0**	**0**

As we can see from the above table. the new requirements have had a greater impact on the program implemented through Hibernate, whether in terms of time or effort to implement these changes. The implementation of the new changes in requirements with Hibernate required a total of 28 lines of code and 30 minutes , in contrast using JDBC did not required any change because , navigation dose not an issue for it.

7.4.4. Some of Hibernate 's application code has changed to apply this modification

For ease of reference, we listing the code related to changes made.

```
import java.util.HashSet;
import java.util.Set;
public class Customer {
. . .
   private Set<Order> orders;
  . . .
   public Customer(String custonerId, String name) {
       this.custonerId = custonerId;
       this.name = name;
       this.orders = new HashSet<Order>();
   }
 . . .
   public Set<Order> getOrders() {
       return orders;
   }

   public void setOrders(Set<Order> orders) {
       this.orders = orders;
   }
           . . .
}
```

Figure 36. Impact of change on Customer class.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://Hibernate.sourceforge.net/Hibernate-mapping-3.0.dtd">
<Hibernate-mapping>
  <class name="Customer" table="customer">
           . . .
    <set name="orders" table="orders"  inverse="true">
      <key column="CustomerID"/>
      <one-to-many class="Order"/>
    </set>
  </class>
</Hibernate-mapping>
```

Figure 37. Impact the change on mapping file of Customer class.

Chapter 7

Fourth Case Study

This case study is more complicated and realistic to produce much more statistical data to compare the two approaches. It will also highlight the issue of relationship representation. Furthermore, it also illustrate that object oriented approach is more sensitive to class model than the relational model.

8.1. Problem Statement

Eastern Suburb Gymnastics (ESG) is a regional organization that is responsible for running competitions among the gymnastics clubs in eastern suburbs of Melbourne. The competitions are organized into seasons. ESG needs a system to help organize and maintain records of the competitions that take place in a single season. The system, in essence, needs to keep information on the gymnasts, their clubs, the organization of the competitions, and the competition results.

Clubs and Club Members

At the beginning of the season, before any competition can take place, every club that wishes to participate in the competition must register with ESG. Each club is known by a unique name; however, for ease of reference, each is also given an ID, which is a short sequence of characters. Each club must provide a contact person's name and phone number, and the club's address and fax number.

In registering with ESG, a club submits a list of their members who will participate in the season's competitions. During the season, additional members can be registered while the season's competitions are in progress. The date a gymnast registers for competition is recorded. Those who register when the club is registered have the first day of the season as the registered date. For each gymnast, ESG requires their name, date of birth, gender and phone contact. ESG then issues them with a unique ID.

Meets, Competitions and Events

The competitions are organized into a series of meets. Each meet is held in the course of one day at one particular venue.

Each meet consists of competitions in four divisions: Women's Junior (WJ), Women's Senior (WS), Men's Junior (MJ), and Men's Senior (MS). A division is identified by a code (e.g. WJ) and has a name (e.g. Women’s Junior). Junior divisions are for gymnasts up to 15 years of age by the first date of the season.

Each competition consists of a series of events run on different equipment. The events in a competition are drawn from a standard list of event types. Each type of event has a code and a descriptive name that is also unique. Certain event types are for women or for men only.

Each meet is identified by an ID (e.g. “M01”), and has a name (e.g. Vacation Classic). A competition within a meet is identified across the system by the combination of the meet ID and the division code.

Teams in Competitions

When a club registers for a meet, the club enters a subset of its members. This subset is known as a team. When a team is at a meet, it must participate in all the events of that competition. A team must have the same set of members competing for each event within a competition.

Scoring

Each event in a meet has a judging panel assigned to it. These people are qualified to give scores for this event. ESG (and the system) maintains a list of judges including their personal details (name, phone number) and the types of events they are qualified to judge. For ease of reference, each judge is given a unique ID.

Each judge rates the performance of a gymnast on an event. The highest and lowest scores will be thrown out, and the rest averaged to be the gymnast's score for the event. This average will be entered into the system. The event score for a team is the sum of all its members' scores for the event. The competition score for a team (which is also its meet score) is the sum of the team’s event scores.

8.2. Use Case Description

1. Add a club
2. Add a gymnast
3. Add a division
4. Add an event type
5. Add a judge

6. Add a meet (and the competitions for the meet)
7. Add an event for a competition
8. Assign a judge to an event (in a competition)
9. Register a team

Due to the maximum limit of papers we shift the use case description to the appendix.

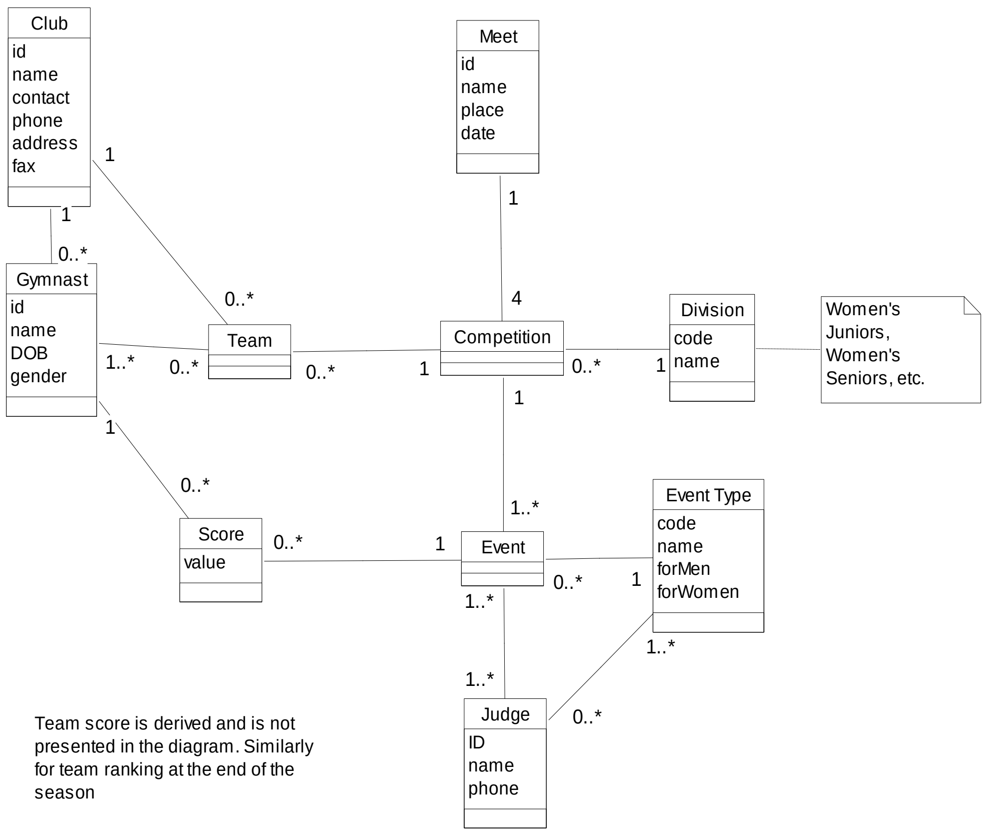


Figure 38. UML class diagram of the Eastern Suburb Gymnastics.

8.3. Comparison of the two approach

Table 8. Summary the findings of the initial construction

Program	File	SLOC	Total/Lines	Estimated Time
Hibernate	GScoringSystem	237	810	6
	Club	44		
	Competition	24		
	CompetitionPk	35		
	Division	60		
	EventPk	46		
	Event	37		
	EventType	58		
	Gymnast	60		
	Judge	40		
	Meet	52		
	TeamPk	46		
	Team	33		
	Score	22		
	HibernateUtil	16		
	Club.hbm.xml	13	177	
	Competition.hbm.xml	12		
	Division.hbm.xml	15		
	Event.hbm.xml	21		
	EventType.hbm.xml	14		
	Gymnast.hbm.xml	15		
	Judge.hbm.xml	17		
	Meet.hbm.xml	14		
	Team.hbm.xml	14		
	Score.hbm.xml	16		
	Hibernate.cfg.xml	26		
	Total	987		
JDBC	GScoringSystem	259	259	3

The table above summarizes the findings to implement the Eastern Suburb Gymnastics with two approaches. Hibernate required a total of 987 lines of code , in contrast using JDBC required 259lines. In addition, the Hibernate required about 6 hour whereas using JDBC required 3 hour. It can be seen that, Hibernate required more effort and time than JDBC. This difference represents nearly a 4:1 ratio in quantity of code. Indeed, the source of the increase in the number of codes was due to POJO and its mapping files.

8.4. Impact of change on two approaches

we investigation how sensitive the two approaches are to the choose of domains modeled

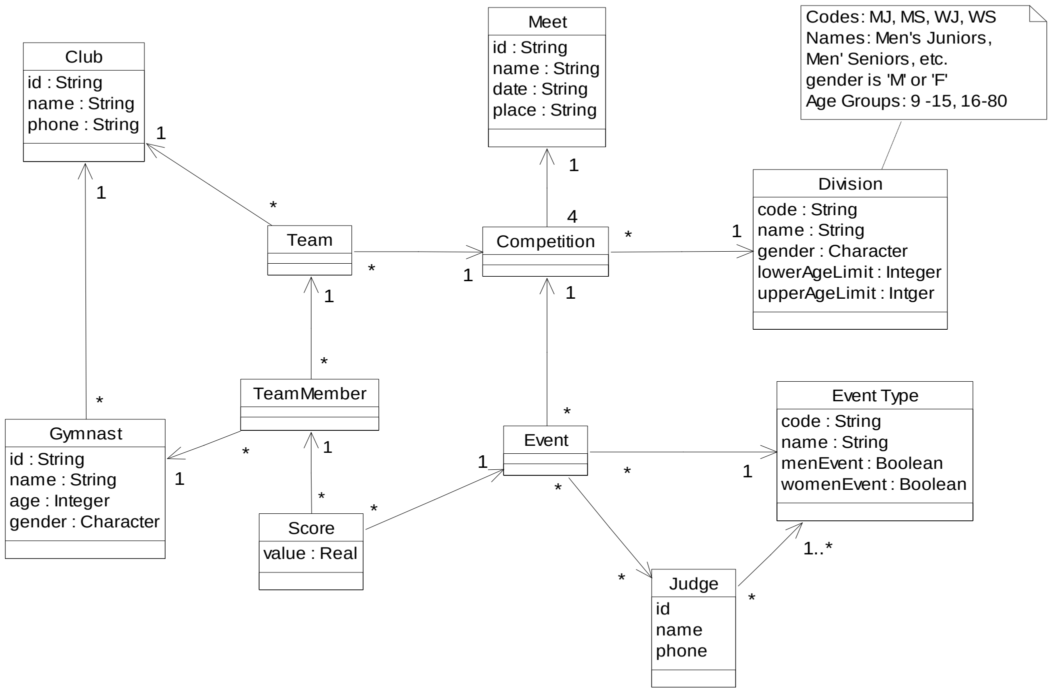


Figure 39.Impact the change on class diagram of the ESG.

8.4.1. Making changes using Hibernate to apply this modification:

1. Create new table called TeamMember .
2. Create new class TeamMember (551lines).
3. Create Mapping file for TeamMember class(14 lines).
4. Modify GScoringSystem class (37 lines):
    a. Add new method called registerMember.

8.4.2. Making changes using Hibernate to apply this modification:

1. Create new table called TeamMember(this step same as Hibernate) .
2. Modify GScoringSystem class (32 lines):
    a. Add new method called registerMember.

8.4.2. Comparison of two approach

Table 9. Summary the findings after change domain model.

Program	File	V(1)	V(2)	Added	Modified	Deleted	A+M+D	Estimated Time
	GScoringSystem	237	274	**37**	**0**	**0**	**37**	
	Club	44	44	0	0	0	0	
	Competition	24	24	0	0	0	0	
	CompetitionPk	35	35	0	0	0	0	
	Division	60	60	0	0	0	0	
	EventPk	46	46	0	0	0	0	
	Event	37	37	0	0	0	0	
	EventType	58	58	0	0	0	0	
	Gymnast	60	60	0	0	0	0	
	Judge	40	40	0	0	0	0	
	Meet	52	52	0	0	0	0	
	TeamPk	46	46	0	0	0	0	
	Team	33	33	0	0	0	0	
	TeamMember	**-**	**55**	**55**	**0**	**0**	**55**	
Hibernate	Score	22	22	0	0	0	0	**1** hour
	HibernateUtil	16	16	0	0	0	0	
	Club.hbm.xml	13	13	0	0	0	0	
	Competition.hbm.xml	12	12	0	0	0	0	
	Division.hbm.xml	15	15	0	0	0	0	
	Event.hbm.xml	21	21	0	0	0	0	
	EventType.hbm.xml	14	14	0	0	0	0	
	Gymnast.hbm.xml	15	15	0	0	0	0	
	Judge.hbm.xml	17	17	0	0	0	0	
	Meet.hbm.xml	14	14	0	0	0	0	
	Team.hbm.xml	14	14	0	0	0	0	
	TeamMember..xml	**-**	**14**	**14**	**0**	**0**	**14**	
	Score.hbm.xml	16	16	0	0	0	0	
	Hibernate.cfg.xml	26	27	1	0	0	1	
	Total	**987**	**1093**	**106**	**0**	**0**	**106**	
JDBC	**GScoringSystem**	**259**	**291**	**32**	**0**	**0**	**32**	**25** minutes

As we can see from the above table. the new requirements have had a greater impact on the program implemented through Hibernate, whether in terms of time or effort to implement these changes. The implementation of the new changes in requirements with Hibernate required a total of 106 lines of code, in contrast to JDBC which required only 32 lines of code. This difference represents nearly a 3:1 ratio in quantity of code. Moreover, the implementation with Hibernate took about one hour compared to only 25 minutes using JDBC.

9.4.3. Some of code has changed or added with Hibernate to apply this modification

For ease of reference, we listing the code related to changes made.

```
public class TeamMember {
  private Gymnast gymnast;
  private Division division;
  private Meet meet;
  private Club club;
    public TeamMemberPK(Gymnast gymnast, Division division, Meet meet, Club club){
        this.gymnast = gymnast;
        this.division = division;
        this.meet = meet;
        this.club = club;
    }
    public Gymnast getGymnast() {
        return gymnast;
    }
    public void setGymnast(Gymnast gymnast) {
        this.gymnast = gymnast;
    }
    public Division getDivision() {
        return division;
    }
    public void setDivision(Division division) {
        this.division = division;
    }
    public Meet getMeet() {
        return meet;
    }
    public void setMeet(Meet meet) {
        this.meet = meet;
    }
    public Club getClub() {
        return club;
    }
    public void setClub(Club club) {
        this.club = club;
    }
}
```

Figure 40. TeamMember class

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE Hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://Hibernate.sourceforge.net/Hibernate-mapping-3.0.dtd">
<Hibernate-mapping>
   <class name="TeamMember" table="teammember">
   <composite-id name="teamMemberPK" class="TeamMemberPK">
    <key-many-to-one name="gymnast" class="Gymnast" column="gymnast_id"/>
    <key-many-to-one name="club" class="Club" column="club_id"/>
    <key-many-to-one name="division" class="Division" column="division_code"/>
    <key-many-to-one name="meet" class="Meet" column="meet_id"/>
   </composite-id>
    </class>
</Hibernate-mapping>
```

Figure 41. TeamMember.hbm.xml

```
public void registerMember(String id,String code , String gymnastid){

    Gymnast gymnast = (Gymnast) search(Gymnast.class,gymnastid);

     boolean pre1 = (gymnast == null);
     if(!pre1){
         System.out.println("ERROR: The gymnast number does not exist!");
         return;
      }

    Division division = (Division) search(Division.class,code);
    Meet meet = (Meet) search(Meet.class,id);
    Club club = gymnast.getClub();

    TeamPk obj = new TeamPk(division,meet,club);
    boolean pre2 = (obj == null);
     if(pre2){
         System.out.println("ERROR: The team number does not exist!");
         return;
      }
    boolean pre3 = true;

     if((gymnast.getGender()!= division.getGender()) ||
            (gymnast.getAge() > division.getUpperAgeLimit()||
             gymnast.getAge() < division.getLowerAgeLimit())){

             pre3 = false;
     }

    if(!pre3){
        System.out.println("ERROR: member does not meet the gender and age criteria!");
        return;
    }

     TeamMember teamMember = new TeamMember(gymnast,division,meet,club);
     saveOrUpdate(teamMember);
}
```

Figure 42. registerMember method of GScoringSystem class.

8.4.4 Some of code has changed or added with JDBC to apply this modification

For ease of reference, we listing the code related to changes made.

```
public void registerMember(String id,String code , String gymnastid) throws SQLException{

    String sql = "SELECT * FROM gymnast WHERE id =" + gymnastid;
    ResultSet rs = search(sql);
    boolean pre1 = rs.next();
    if(!pre1){
        System.out.println("ERROR: The gymnast number does not exist!");
        return;
     }
    String  club_id = rs.getString("club_id");
    sql = "SELECT * FROM team WHERE club_id =" + club_id +
              " And meet_id="+id+ " And division_code="+code;
    boolean pre2 = (search(sql)).next();
    if(!pre2){
        System.out.println("ERROR: The team number does not exist!");
        return;
     }
   sql = "SELECT * FROM division division code =" + code;
   ResultSet rsDivision = search(sql);

   boolean pre3 = true;

    if(!(rs.getString("gender").equals(rsDivision.getString("gender")) ||
          (rs.getInt("age") > rsDivision.getInt("upperAgeLimit"))||
           rs.getInt("age") < rsDivision.getInt("lowerAgeLimit"))){

           pre3 = false;
    }

   if(!pre3){
       System.out.println("ERROR: member does not meet the gender and age criteria!");
       return;
   }
    Object[] values = getValues(gymnastid,club_id,id,code);
    insert("teammember" , values);
}
```

Figure 43. registerMember method of GScoringSystem class.

9. Conclusion

The result of our critical comparison of the two paradigms in terms of flexibility according to the findings of implementations indicate that using ORM is much cost in the initial construction of the application than using JDBC. In other words , the level of effort and time to implement the application is much higher using Hibernate than using JDBC. For instance, the initial construction of the Eastern Suburb Gymnastics with ORM required a total of 987 lines of code, in contrast using JDBC required 259 lines. This difference represents nearly a 4:1 ratio in quantity of code. In addition, ORM required about 6 hour whereas using JDBC required 3 hour. Indeed, increasing the number of classes that need to be persisted automatically can lead to increased levels of effort and time.

Moreover, JDBC is more flexible to deal with the requirement change compared to ORM. To illustrate that, for an object to be persisted to a database, Hibernate needs a mapping file for all of the objects that are to be persisted as well as POJO , which is not required when using the approach of JDBC. This means that, if we would like to add/delete a attribute to a class, we need to modify mapping file of that class to map or delete that a attribute, after that we need to modify that class its self to add/delete that a attribute with its getter and setter methods whereas we do not need these steps to using JDBC. For example , form table 9 we can see that , the implementation of the new changes in requirements with Hibernate required a total of 106 lines of code , about 70 lines just for mapping file and to create the POJO class , in contrast using JDBC required only 32 lines of code.

Furthermore, object oriented paradigm has an issue of navigation between objects through association links whereas navigation doses not an issue for JDBC. In addition, to determine the direction with UML is not an easy task which cans one of the common mistakes in design decision.

In addition, object oriented approach is more sensitive to class model than the relational model.

10. Future Work.

Our examination is only the beginning .We need to do more investigation. In particular, we should conduct surveys or experiments, based on experiments design principals, to answer that question of whether object-oriented system is better than relational system or vice versa in terms of the applicability and flexibility to requirements change.

addition, to determine the direction with UML is not an easy task which cans one of the common mistakes in design decision.

In addition, object oriented approach is more sensitive to class model than the relational model.

10. Future Work.

Our examination is only the beginning .We need to do more investigation. In particular, we should conduct surveys or experiments, based on experiments design principals, to answer that question of whether object-oriented system is better than relational system or vice versa in terms of the applicability and flexibility to requirements change.

11. REFERENCES

1. Codd, E.F. A Relational Model of Data for Large Shared Data Banks. CACM 13(6): 377-387. 1970.

2. Aguirre-Urreta, M. I. and Marakas, G. M. 2008. Comparing conceptual modeling techniques: a critical review of the EER vs. OO empirical literature. SIGMIS Database 39, 2 (Apr. 2008), 9-32.

3. Lodhi, F. and Ghazali, M. A. 2007. Design of a simple and effective object-to-relational mapping technique. In Proceedings of the 2007 ACM Symposium on Applied Computing (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY,

4. Urban, S. D. and Dietrich, S. W. 2003. Using UML class diagrams for a comparative analysis of relational, object-oriented, and object-relational database mappings. In Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (Reno, Navada, USA, February 19 - 23, 2003). SIGCSE '03. ACM, New York, NY, 21-25.

5. Sinha, A. P. and Vessey, I. 1999. An empirical investigation of entity-based and object-oriented data modeling: a development life cycle approach. In Proceedings of the 20th international Conference on information Systems (Charlotte, North Carolina, United States, December 12 - 15, 1999). International Conference on Information Systems. Association for Information Systems, Atlanta, GA, 229-244.

6. Chen, P. P. 1994. The entity-relationship model—toward a unified view of data. In Readings in Database Systems (2nd Ed.), M. Stonebraker, Ed. Morgan Kaufmann Series In Data Management Systems. Morgan Kaufmann Publishers, San Francisco, CA, 741-754.

7. Blaha, M. R., Premerlani, W. J., and Rumbaugh, J. E. 1988. Relational database design using an object-oriented methodology. Commun. ACM 31, 4 (Apr. 1988), 414-427.

8. O'Neil, E. J. 2008. Object/relational mapping 2008: Hibernate and the entity data model (edm). In Proceedings of the 2008 ACM SIGMOD international Conference on Management of Data (Vancouver, Canada, June 09 - 12, 2008). SIGMOD '08. ACM, New York, NY, 1351-1356.

9. Keller, R. K., Schauer, R., and Cockburn, A. 1997. Object-oriented design quality. In Addendum To the 1997 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum) (Atlanta, Georgia, United States, October 05 - 09, 1997). OOPSLA '97. ACM, New York, NY, 63-67.

10. Booch, G. 1991. Object-oriented Analysis and Design with Applications. Benjamin-Cummings Publishing.

11. Russell, C. 2008. Bridging the Object-Relational Divide. Queue 6, 3 (May. 2008), 18-28.

12. Hall, John M., The Maintainability of Object-Oriented Software, 2003.

13. Douglas Barry, Torsten Stanienda, "Solving the Java Object Storage Problem," Computer, vol. 31, no. 11, pp. 33-40, Nov. 1998, doi:10.1109/2.730734

14. Crowther, PC and Hartnett, JS, *Handling Spatial Objects in a GIS Database-Relational v Object Oriented Approaches*, Proceedings of GeoComputation 2001, September 23 - 26, 2001, Brisbane, pp. 6

15. R. Polan, "Impedance Mismatch of Business Process Modeling and Object-Oriented Implementation," seep,pp.370, 1998 International Conference on Software Engineering: Education & Practice, 1998

16. Briand L., Bunse L., Daly J., Differding C. An Experimental comparison of the Maintainability of Object-Oriented and Structured Design Documents, Empirical Software Engineering – An International Journal, 1997. Available at: http://citeseer.nj.nec.com/12374.html.

17. Unger B., Prechelt L. The impact of inheritance depth on maintenance tasks: Detailed description and evaluation of two experiment replications. Technical Report 19/1998, Universität Karlsruhe, Fakultät für Informatik, Germany, July 1998. Available at: http://www.ipd.uka.de/~prechelt/Biblio/inheritTR.pdf.

18. Corritore C., Wiedenbeck S. Direction and Scope of Comprehension-Related Activities by Procedural and Object-Oriented Programmers: An Empirical Study, Proceedings of the 8th International Workshop on Program Comprehension (IWPC'00). 2000.
Available
at:http://www2.umassd.edu/SWComprehension/compdocs/nebraska/corritoreiwpc00.pdf

19. E. Bertino, L. Martino: Object-oriented Database Management Systems: Concepts and Issues; IEEE Computer 24 (4) 1991, 33-47

20. Hibernate as an object-relational mapping framework for eHealth systems. Software Technologies Session Alberto Moreno Jiménez, Elena Villalba Mora, M.T. Arredondo Life Supporting Technologies ETSI Telecomunicación Universidad Politécnica de Madrid

21. Downs K: Why I do not use ORM. August 25, 2008.
Available at: http://database-programmer.blogspot.com/2008/06/why-i-do-not-use-orm.html

22. Melnik, S., Adya, A., and Bernstein, P. A. 2007. Compiling mappings to bridge applications and databases. In Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data (Beijing, China, June 11 - 14, 2007).

23. Engels G., Kappel G. Object-Oriented System Development: Will the New Approach Solve Old Problems? IFIP 1994 Congress, Vol. 3, K. Duncan and K. Krueger (eds.), North-Holland, September 1994. Available at: http://citeseer.nj.nec.com/engels94objectoriented.html.

24. Scott W. Ambler, "Mapping objects to relational database", White Paper, Ronin International, October 21, 2000.

12. Appendix A

12.1.Original Source Code of Car Park System

12.1.1. Hibernate

- **Employee Class**

```
import java.util.HashSet;
import java.util.Set;
public class Employee {

  private int id;
  private String name;
  private String phone;
  private Set<Permit> permits;

   public Employee(){

   }
   public Employee(int id, String name, String phone) {
      this.id = id;
      this.name = name;
      this.phone = phone;
      this.permits = new HashSet<Permit>();
   }
   public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id = id;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getPhone() {
      return phone;
   }
   public void setPhone(String phone) {
      this.phone = phone;
   }
   public Set<Permit> getPermits() {
      return permits;
   }
   public void setPermits(Set<Permit> permits) {
      this.permits = permits;
   }

 @Override
  public String toString() {
    String desc = "Employee[ id: " + id + " , name: " + name
           + " , phone: " + phone +  "]";
      return desc;
   }
}
```

- **Employee.hbm.xml**

```
<?xml version="1.0" encoding='utf-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Employee" table="employees">
      <id name="Id" column="Employee_id">
         <generator class="assigned"/>
      </id>
      <property name="name" column="Employee_name"/>
      <property name="phone"/>
      <set name="permits" table="Permits"  inverse="true" cascade="all">
            <key column="Employee_id"/>
            <one-to-many class="Permit"/>
      </set>
   </class>
</hibernate-mapping>
```

- **Permit.java**

```
public class Permit {
   private int permitNr;
   private String regNr;
   private String area;
   private Employee owner;

    public Permit(){

    }
    public Permit(int permitNr, String regNr, String area , Employee owner) {
       this.permitNr = permitNr;
       this.regNr = regNr;
       this.area = area;
       this.owner =  owner;
    }

    /**
     * @return the permitNr
     */
    public int getPermitNr() {
       return permitNr;
    }

    /**
     * @param permitNr the permitNr to set
     */
    public void setPermitNr(int permitNr) {
       this.permitNr = permitNr;
    }

    /**
     * @return the regNr
     */
    public String getRegNr() {
       return regNr;
    }

    /**
     * @param regNr the regNr to set
     */
    public void setRegNr(String regNr) {
       this.regNr = regNr;
    }

    /**
     * @return the area
     */
    public String getArea() {
       return area;
    }

    /**
     * @param area the area to set
     */
    public void setArea(String area) {
       this.area = area;
    }

    /**
     * @return the owner
     */
    public Employee getOwner() {
       return owner;
    }

    /**
     * @param owner the owner to set
     */
    public void setOwner(Employee owner) {
       this.owner = owner;
    }

   @Override
   public String toString()
    {
```

```
    String desc = "Permit[permitNr: " + permitNr
         + ", regNr: " + regNr
         + ", area: " + area
         + ", owner: " + owner.getId() + "]";

    return desc;
  }
}
```

- **Permit.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Permit" table="permits">
      <id name="permitNr" >
         <generator class="assigned"/>
      </id>
      <property name="regNr" type="string"/>
      <property name="area" type="string"/>
      <many-to-one name="owner" column="Employee_id"/>
   </class>
</hibernate-mapping>
```

- **HibernateUtil.java**

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
  private static final SessionFactory sessionFactory;
  static {
    try {
      // Create the SessionFactory from hibernate.cfg.xml
      sessionFactory = new Configuration().configure().buildSessionFactory();
    } catch (Throwable ex) {
      // Make sure you log the exception, as it might be swallowed
      System.err.println("Initial SessionFactory creation failed." + ex);
      throw new ExceptionInInitializerError(ex);
    }
  }
  public static SessionFactory getSessionFactory() {
    return sessionFactory;
  }
}
```

- **hibernate.cfg.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost/hibernatetutorial</property>
        <property name="connection.username">root</property>
        <property name="connection.password"> root </property>
       <!-- session properties -->
        <property
name="hibernate.current_session_context_class">org.hibernate.context.Thr
eadLocalSessionContext</property>
       <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>
        <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">false</property>
        <!-- Mapping files -->
        <mapping resource="Permit.hbm.xml"/>
        <mapping resource="Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

- **CPSystem.java**

```
import java.util.List;
import org.hibernate.*;
public class CPSystem
{

    public Employee searchEmployee(int id){

              Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Employee employee = (Employee)session.get(Employee.class,id);
        return employee;
    }

   public void addEmployee(int id, String name, String phone)
   {
      //check precondition
       Employee employee = searchEmployee(id);
       boolean pre1 = (employee == null);

      if(!pre1)
      {
        System.out.println("ERROR: The id already exists!");
        return;
      }
        Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        employee = new Employee(id,name,phone);
        session.save(employee);
        session.getTransaction().commit();
}

   public void deleteEmployee(int id)
   {
      //check precondition
      Employee employee = searchEmployee(id);
      boolean pre1 = (employee == null);

      if(pre1)
      {
        System.out.println("ERROR: The employee number does not exist!");
        return;
      }

      Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      session.delete(employee);
      session.getTransaction().commit();
   }

   public Permit searchPermit(int permitNr)
    {
              Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Permit permit = (Permit)session.get(Permit.class,permitNr);
        return permit;
    }

    public boolean searchCar(String regNr)
    {
      Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      Query q = session.createQuery("from Permit as c where
c.regNr=:regNr");
      q.setString("regNr",regNr);
      List cResult = q.list();
      return cResult.size()>0;
    }
   public void terminatePermit(int permitNr)
   {
     Permit permit = searchPermit(permitNr);
     boolean per1 = (permit == null);

     if(per1)
     {
       System.out.println("Permit number does not exist");
       return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     session.delete(permit);
     session.getTransaction().commit();
   }
   //---------

   public void addPermit(int permitNr, String regNr,String area, int id)
   {
      //check permit number is new

      Permit permit = searchPermit(permitNr);
      boolean pre1= (permit == null);

      if(!pre1)
      {
         System.out.println("ERROR: The permit number already exists!");
         return;
      }

      //check that owner exists and has less than 2 permits

      Employee owner = searchEmployee(id);
      boolean pre2 = (owner == null);

      if(pre2)
      {
         System.out.println("ERROR: The owner id does not exist");
         return;
      }

      boolean pre3 = owner.getPermits().size() < 2;

      if(!pre3)
      {
         System.out.println("ERROR: The owner already has 2 permits!");
         return;
      }

      //check that the car is new (no such car in the system)

      boolean pre4 = searchCar(regNr);

      if(pre4)
      {
         System.out.println("ERROR: The car already exists");
         return;
      }

      //add new permit

       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       permit= new Permit(permitNr,regNr,area, owner);
       session.save(permit);
       session.getTransaction().commit();
  }

 public void replacePermit(int oldPermitNr,int newPermitNr,
                          String regNr, String area)
   {
      Permit permit = searchPermit(oldPermitNr);
      boolean pre= (permit == null);
      if(pre)
      {
         System.out.println("Permit number does not exist");
         return;
      }
      terminatePermit(oldPermitNr);
      addPermit(newPermitNr,  regNr, area,permit.getOwner().getId());
   }

   public void changePhoneNumber(int id, String phone)
   {
      Employee employee = searchEmployee(id);
      boolean pre =(employee == null);

      if(pre)
      {
        System.out.println("ERROR: The id already exists!");
        return;
      }

       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       employee.setPhone(phone);
       session.update(employee);
       session.getTransaction().commit();
   }
 public void getDetailsOfEmployeeByPermit(int permitNr){

    Permit permit = searchPermit(permitNr);
    boolean pre= (permit == null);
    if(pre){
         System.out.println("Permit number does not exist");
         return;
      }
     System.out.println(permit.getOwner());
   }
}
```

12.1.2. JDBC

- **CPSystem**

```
package carparkinjdbc;
import java.sql.*;
public class CPSystem
{
    public static String url = "jdbc:mysql://localhost/hibernatetutorial";
    public static String dbdriver = "com.mysql.jdbc.Driver";
    public static String username = "root";
    public static String password = " root";
    static Connection con;

    public CPSystem()
    {
       con = getConnection();
    }

     public void ConnClose(){
       try
               {
                             con.close();
               }
               catch(SQLException ex)
               {
                                          System.err.println("SQLException: " +
ex.getMessage());
               }
     }

    public static Connection getConnection()
              {

                try
                {

              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(java.lang.ClassNotFoundException e)
                {
                             System.err.print("ClassNotFoundException: ");
                             System.err.println(e.getMessage());
                }
                try
                {
                              con = DriverManager.getConnection(url,username,
password);
                }
                catch(SQLException ex)
                {
                                          System.err.println("SQLException: " +
ex.getMessage());
                }
                           return con;
              }

   public boolean searchEmployee(int id){

      String sqlStmt = "SELECT Employee_id FROM Employees WHERE
Employee_id = ?";
      boolean isExist = true;
      try
      {
           PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
           pstmt.setInt(1,id);
           ResultSet rs = pstmt.executeQuery();
           isExist = rs.next();
      }
      catch( SQLException e ) {
           System.out.println(e.getMessage());
           e.printStackTrace();
        }

       return isExist;
   }

  public void addEmployee(int id, String name, String phone)
   {
      //check precondition

       boolean pre1 = searchEmployee(id);

      if(pre1)
      {
        System.out.println("ERROR: The id already exists!");
        return;
      }
      try {
       String sqlStmt="INSERT into Employees
(Employee_id,Employee_name,phone) VALUES(?,?,?)";
       PreparedStatement pStmt = con.prepareStatement(sqlStmt);
       pStmt.setInt(1, id);
       pStmt.setString(2, name);
       pStmt.setString(3, phone);
       pStmt.executeUpdate();
      } catch (SQLException e) {
             System.out.println(e.getMessage());
            e.printStackTrace();
   }
}

  public void deleteEmployee(int id)
   {
      //check precondition

      boolean pre1 = searchEmployee(id);

      if(!pre1)
      {
        System.out.println("ERROR: The employee number does not exist!");
        return;
      }
     try {
        String sqlStmt="DELETE FROM Employees WHERE Employee_id =
?";
        PreparedStatement pStmt = con.prepareStatement(sqlStmt);
        pStmt.setInt(1, id);
        pStmt.executeUpdate();
      } catch (SQLException e) {
             System.out.println(e.getMessage());
            e.printStackTrace();
   }
  }
```

```
   public boolean searchPermit(int permitNr)
    {
      String sqlStmt = "SELECT permitNr FROM permits WHERE permitNr
= ?";
      boolean isExist = true;
      try
      {
           PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
           pstmt.setInt(1,permitNr);
           ResultSet rs = pstmt.executeQuery();
           isExist = rs.next();
      }
      catch( SQLException e ) {
           System.out.println(e.getMessage());
           e.printStackTrace();
        }

       return isExist;
    }

    public boolean searchCar(String regNr)
    {
       String sqlStmt = "SELECT permitNr FROM permits WHERE regNr =
?";
       boolean isExist = true;
      try
      {
           PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
           pstmt.setString(1,regNr);
           ResultSet rs = pstmt.executeQuery();
           isExist = rs.next();
      }
      catch( SQLException e ) {
           System.out.println(e.getMessage());
           e.printStackTrace();
        }

       return isExist;
    }

    public int countOfPermits(int id)
    {
      String sqlStmt = "SELECT count(*) As rowCount FROM permits
WHERE Employee_id = ?";
       int rowCount =0;
      try
      {
           PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
           pstmt.setInt(1, id);
           ResultSet rs = pstmt.executeQuery();
           rs.first();
           rowCount= rs.getInt("rowCount");
      }
      catch( SQLException e ) {
           System.out.println(e.getMessage());
           e.printStackTrace();
        }

       return rowCount;
    }

    public void terminatePermit(int permitNr)
    {

     boolean per1 = searchPermit(permitNr);

      if(!per1)
      {
        System.out.println("Permit number does not exist");
        return;
      }
      try
      {
       String sqlStmt="DELETE FROM permits WHERE permitNr = ?";
       PreparedStatement pStmt = con.prepareStatement(sqlStmt);
       pStmt.setInt(1, permitNr);
       pStmt.executeUpdate();
      } catch (SQLException e) {
             System.out.println(e.getMessage());
            e.printStackTrace();
    }
    }
```

```
  public void addPermit(int permitNr, String regNr,String area, int id)
  {
    //check permit number is new
    boolean pre1= searchPermit(permitNr);

    if(pre1)
    {
      System.out.println("ERROR: The permit number already exists!");
      return;
    }

    //check that owner exists and has less than 2 permits

    boolean pre2 = searchEmployee(id);

    if(!pre2)
    {
      System.out.println("ERROR: The owner id does not exist");
      return;
    }

    boolean pre3 = countOfPermits(id) < 2;

    if(!pre3)
    {
      System.out.println("ERROR: The owner already has 2 permits!");
      return;
    }

    //check that the car is new (no such car in the system)

    boolean pre4 = searchCar(regNr);

    if(pre4)
    {
      System.out.println("ERROR: The car already exists");
      return;
    }

    //add new permit
   try {
    String sqlStmt="INSERT into permits
(permitNr,regNr,area,Employee_id) VALUES(?,?,?,?)";
    PreparedStatement pStmt = con.prepareStatement(sqlStmt);
    pStmt.setInt(1, permitNr);
    pStmt.setString(2, regNr);
    pStmt.setString(3, area);
    pStmt.setInt(4, id);
    pStmt.executeUpdate();
   } catch (SQLException e) {
          System.out.println(e.getMessage());
          e.printStackTrace();
  }
 }

 public void replacePermit(int oldPermitNr,int newPermitNr,
                         String regNr, String area)
  {
    boolean pre= searchPermit(oldPermitNr);
    if(!pre)
    {
      System.out.println("Permit number does not exist");
      return;
    }
    String sqlStmt = "SELECT Employee_id FROM permits WHERE
permitNr = ?";
    try
    {
      PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
      pstmt.setInt(1,oldPermitNr);
      ResultSet rs = pstmt.executeQuery();
      rs.first();
      int id = rs.getInt("Employee_id");
      rs.close();
      terminatePermit(oldPermitNr);
      addPermit(newPermitNr,area,regNr,id);
    }
   catch( SQLException e ) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
  }
  public void changePhoneNumber(int id, String phone)
  {
    boolean pre1 = searchEmployee(id);

    if(!pre1)
    {
      System.out.println("ERROR: The employee number does not exist!");
      return;
    }

    try
    {
      String sqlStmt="UPDATE employees SET phone = ? WHERE
Employee_id = ?";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1, phone);
      pStmt.setInt(2, id);
      pStmt.executeUpdate();

    } catch (SQLException e) {
          System.out.println(e.getMessage());
         e.printStackTrace();
  }
 }

 public void getDetailsOfEmployeeByPermit(int permitNr){

    boolean per1 = searchPermit(permitNr);

     if(!per1)
     {
      System.out.println("Permit number does not exist");
      return;
     }

    try
     {
    String  sqlStmt ="SELECT e.Employee_id, e.Employee_name, e.phone "
+
                 "FROM permits p ,employees e " +
                 "WHERE(p.Employee_id = e.Employee_id)" +
                 "And (p.permitNr = ?) ";
    PreparedStatement pStmt = con.prepareStatement(sqlStmt);
    pStmt.setInt(1, permitNr);
    ResultSet rs = pStmt.executeQuery();
    while (rs.next()) {

      int id = rs.getInt("e.Employee_id");
      String phone = rs.getString("e.phone");
      String name = rs.getString("e.Employee_name");

      System.out.println("Employee[ id: " + id + " , name: " + name
          + " , phone: " + phone + "]");
     }

    } catch (SQLException e) {
          System.out.println(e.getMessage());
         e.printStackTrace();
  }
 }
}
```

12.2.The Source Code of CPS After Programmatic Adjustment
12.2.1. Hibernate

- **Employee**

```
import java.util.HashSet;
import java.util.Set;

public class  Employee {

  private int id;
  private String name;
  private String phone;
  private Set<Permit> permits;

   public Employee(){

   }
   public Employee(int id, String name, String phone) {
      this.id = id;
      this.name = name;
      this.phone = phone;
      this.permits = new HashSet<Permit>();
   }

   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public String getPhone() {
      return phone;
   }
   public void setPhone(String phone) {
      this.phone = phone;
   }
   public Set<Permit> getPermits() {
      return permits;
   }
   public void setPermits(Set<Permit> permits) {
      this.permits = permits;
   }

   @Override
  public String toString()
  {
     String desc = "Employee[ id: " + id + " , name: " +
           name+ " , phone: " + phone;
      return desc;
   }
}
```

- **FullTime**

```
public class FullTime extends Employee{
   private int salary;

 public FullTime(){};

   public FullTime(int id, String name, String phone, int salary) {
      super(id, name, phone);
      this.salary = salary;
   }

   public void setSalary(int salary) {
      this.salary = salary;
   }

   public int getSalary() {
      return salary;
   }

   @Override
   public String toString()
   {
      String desc = super.toString() + " , Salary: " + salary + " ]";
      return desc;
   }
}
```

```
PartTime
public class PartTime extends Employee{
   private int payRate;

    public PartTime(){};

    public PartTime(int id, String name, String phone , int payRate) {
       super(id, name, phone);
       this.payRate = payRate;
    }
    public int getPayRate() {
       return payRate;
    }
    public void setPayRate(int payRate) {
       this.payRate = payRate;
    }

   @Override
   public String toString()
   {
       String desc = super.toString() + " , PayRate: " + payRate + " ]";
       return desc;
    }
}
```

- **Employee.hbm.xml**

```
<?xml version="1.0" encoding='utf-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Employee" table="employees">
      <id name="Id" column="Employee_id">
         <generator class="assigned"/>
      </id>
   <discriminator column="Employee_type" type= "string"/>
   <property name="name" column="Employee_name"/>
```

- **Permit.java**

```
public class Permit {
  private int permitNr;
  private String regNr;
  private String area;
  private int terminate;
  private Employee owner;

   public Permit(){

   }
   public Permit(int permitNr, String regNr, String area , Employee owner) {
      this.permitNr = permitNr;
      this.regNr = regNr;
      this.area = area;
      this.owner =  owner;
      this.terminate = 0;
   }
   public int getPermitNr() {
      return permitNr;
   }
   public void setPermitNr(int permitNr) {
      this.permitNr = permitNr;
   }
   public String getRegNr() {
      return regNr;
   }
   public void setRegNr(String regNr) {
      this.regNr = regNr;
   }
   public String getArea() {
      return area;
   }
   public void setArea(String area) {
        this.area = area;
    }
    public Employee getOwner() {
        return owner;
    }
    public void setOwner(Employee owner) {
        this.owner = owner;
    }
    public void setTerminate(int terminate) {
        this.terminate = terminate;
    }

    public int getTerminate() {
        return terminate;
    }

   @Override
   public String toString()
   {
      String desc = "Permit[permitNr: " + permitNr
            + ", regNr: " + regNr
            + ", area: " + area
            + ", owner: " + owner.getId() + "]";

      return desc;
   }
}
```

- **Permit.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Permit" table="permits">
      <id name="permitNr" >
         <generator class="assigned"/>
      </id>
      <property name="regNr" type="string"/>
      <property name="area" type="string"/>
      <many-to-one name="owner" column="Employee_id"/>
      <property name="terminate"/>
   </class>
</hibernate-mapping>
```

- **CPSystem.java**

```
import java.util.List;
import java.util.Set;
import org.hibernate.*;
public class CPSystem
{

public Employee searchEmployee(int id){

                Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Employee employee = (Employee)session.get(Employee.class,id);
        return employee;
    }

   public void addEmployee(int id, String name, String phone,int salPay,
char typeOfEmployee)
   {
      //check precondition
       Employee employee = searchEmployee(id);
       boolean pre1 = (employee == null);

      if(!pre1)
      {
         System.out.println("ERROR: The id already exists!");
         return;
      }
        Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        if(typeOfEmployee =='F'){
           employee= new FullTime(id, name, phone, salPay);
        }
        else {
           employee = new PartTime(id, name, phone, salPay);
        }
         session.save(employee);
         session.getTransaction().commit();
    }

   public void deleteEmployee(int id)
    {
       //check precondition
       Employee employee = searchEmployee(id);
       boolean pre1 = (employee == null);

       if(pre1)
       {
         System.out.println("ERROR: The employee number does not exist!");
         return;
       }
       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       session.delete(employee);
       session.getTransaction().commit();
    }

   public Permit searchPermit(int permitNr)
     {
                Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
         session.beginTransaction();
         Permit permit = (Permit)session.get(Permit.class,permitNr);
         return permit;
     }

    public boolean searchCar(String regNr)
     {
       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Query q = session.createQuery("from Permit as c where
c.regNr=:regNr");
       q.setString("regNr",regNr);
       List cResult = q.list();
       return cResult.size()>0;
     }

    public void terminatePermit(int permitNr)
     {
       Permit permit = searchPermit(permitNr);
       boolean per1 = (permit == null);

       if(per1)
       {
         System.out.println("Permit number does not exist");
         return;
       }
       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       permit.setTerminate(1);
       session.update(permit);
       session.getTransaction().commit();
     }

    public int  getCoutnOfPermits(Set<Permit> permits) {
        int count = 0;
        for(Permit p:permits){
         if(p.getTerminate()==0)

            ++count;
        }
        return count;
     }

    public void addPermit(int permitNr, String regNr,String area, int id)
     {
       //check permit number is new

       Permit permit = searchPermit(permitNr);
       boolean pre1= (permit == null);
    if(!pre1)
    {
      System.out.println("ERROR: The permit number already exists!");
      return;
    }

    //check that owner exists and has less than 2 permits

    Employee owner = searchEmployee(id);
    boolean pre2 = (owner == null);

    if(pre2)
    {
      System.out.println("ERROR: The owner id does not exist");
      return;
    }

    boolean pre3 = getCoutnOfPermits(owner.getPermits()) < 2;

    if(!pre3)
    {
      System.out.println("ERROR: The owner already has 2 permits!");
      return;
    }

    //check that the car is new (no such car in the system)

    boolean pre4 = searchCar(regNr);

    if(pre4)
    {
      System.out.println("ERROR: The car already exists");
      return;
    }

    //add new permit

     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     permit= new Permit(permitNr,regNr,area, owner);
     session.save(permit);
     session.getTransaction().commit();
  }

  public void replacePermit(int oldPermitNr,int newPermitNr,
                    String regNr, String area){
    Permit permit = searchPermit(oldPermitNr);
    boolean pre= (permit == null);
    if(pre){
      System.out.println("Permit number does not exist");
      return;
    }
    terminatePermit(oldPermitNr);
    addPermit(newPermitNr,  regNr, area,permit.getOwner().getId());
  }

   public void changePhoneNumber(int id, String phone)
  {
    Employee employee = searchEmployee(id);
    boolean pre =(employee == null);

    if(pre)
    {
      System.out.println("ERROR: The id already exists!");
      return;
    }

     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     employee.setPhone(phone);
     session.update(employee);
     session.getTransaction().commit();
  }

  public void getDetailsOfEmployeeByPermit(int permitNr){

   Permit permit = searchPermit(permitNr);
   boolean pre= (permit == null);
   if(pre){
      System.out.println("Permit number does not exist");
       return;
     }
   System.out.println(permit.getOwner());
  }
}
```

12.2.2. JDBC

- **CPSystem**

```
package carparkinjdbc;

import java.sql.*;

public class CPSystem {
   public static String url = "jdbc:mysql://localhost/hibernatetutorial";
   public static String dbdriver = "com.mysql.jdbc.Driver";
   public static String username = "root";
   public static String password = "root";
   static Connection con;

   public CPSystem(){
      con = getConnection();
   }
    public void ConnClose(){
      try {
                        con.close();
              }
               catch(SQLException ex)
              {
            System.err.println("SQLException: " + ex.getMessage());
              }
    }

   public static Connection getConnection()  {

              try  {

            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
              }
              catch(java.lang.ClassNotFoundException e)  {
                        System.err.print("ClassNotFoundException: ");
                        System.err.println(e.getMessage());
              }

              try   {
    con = DriverManager.getConnection(url,username, password);
              }
               catch(SQLException ex)
              {
System.err.println("SQLException: " + ex.getMessage());
              }
                      return con;
            }

   public boolean searchEmployee(int id){

    String sqlStmt = "SELECT Employee_id FROM Employees WHERE
Employee_id = ?";
     boolean isExist = true;
     try
     {
        PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
        pstmt.setInt(1,id);
        ResultSet rs = pstmt.executeQuery();
        isExist = rs.next();
     }
     catch( SQLException e ) {
        System.out.println(e.getMessage());
        e.printStackTrace();
      }

     return isExist;
   }

   public void addEmployee(int id, String name, String phone)
   {
     //check precondition

     boolean pre1 = searchEmployee(id);
    if(pre1)
    {
      System.out.println("ERROR: The id already exists!");
      return;
    }
    try {
     String sqlStmt="INSERT into Employees
(Employee_id,Employee_name,phone) VALUES(?,?,?)";
     PreparedStatement pStmt = con.prepareStatement(sqlStmt);
     pStmt.setInt(1, id);
     pStmt.setString(2, name);
     pStmt.setString(3, phone);
     pStmt.executeUpdate();
    } catch (SQLException e) {
            System.out.println(e.getMessage());
           e.printStackTrace();
   }
}

  public void deleteEmployee(int id)
  {
    //check precondition

    boolean pre1 = searchEmployee(id);

    if(!pre1)
    {
      System.out.println("ERROR: The employee number does not exist!");
      return;
    }
   try {
      String sqlStmt="DELETE FROM Employees WHERE Employee_id =
?";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setInt(1, id);
      pStmt.executeUpdate();
    } catch (SQLException e) {
            System.out.println(e.getMessage());
           e.printStackTrace();
   }
  }

  public boolean searchPermit(int permitNr)
   {
    String sqlStmt = "SELECT permitNr FROM permits WHERE permitNr
= ?";
    boolean isExist = true;
    try
    {
        PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
        pstmt.setInt(1,permitNr);
        ResultSet rs = pstmt.executeQuery();
        isExist = rs.next();
    }
    catch( SQLException e ) {
        System.out.println(e.getMessage());
        e.printStackTrace();
      }

     return isExist;
   }

   public boolean searchCar(String regNr)
   {
     String sqlStmt = "SELECT permitNr FROM permits WHERE regNr =
?";
     boolean isExist = true;
    try
    {
        PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
        pstmt.setString(1,regNr);
        ResultSet rs = pstmt.executeQuery();
        isExist = rs.next();
    }
    catch( SQLException e ) {
        System.out.println(e.getMessage());
        e.printStackTrace();
      }

     return isExist;
   }
```

```
  public int countOfPermits(int id)
  {
   String sqlStmt = "SELECT count(*) As rowCount FROM permits
WHERE Employee_id = ?";
    int rowCount =0;
   try
   {
        PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();
        rs.first();
        rowCount= rs.getInt("rowCount");
   }
   catch( SQLException e ) {
        System.out.println(e.getMessage());
        e.printStackTrace();
     }

    return rowCount;
  }

  public void terminatePermit(int permitNr)
  {

   boolean per1 = searchPermit(permitNr);

   if(!per1)
   {
     System.out.println("Permit number does not exist");
     return;
   }
   try
   {
     String sqlStmt="DELETE FROM permits WHERE permitNr = ?";
     PreparedStatement pStmt = con.prepareStatement(sqlStmt);
     pStmt.setInt(1, permitNr);
     pStmt.executeUpdate();
   } catch (SQLException e) {
           System.out.println(e.getMessage());
          e.printStackTrace();
  }
  }

 public void addPermit(int permitNr, String regNr,String area, int id)
  {
    //check permit number is new
    boolean pre1= searchPermit(permitNr);

    if(pre1)
    {
      System.out.println("ERROR: The permit number already exists!");
      return;
    }

    //check that owner exists and has less than 2 permits

    boolean pre2 = searchEmployee(id);

    if(!pre2)
    {
      System.out.println("ERROR: The owner id does not exist");
      return;
    }

    boolean pre3 = countOfPermits(id) < 2;

    if(!pre3)
    {
      System.out.println("ERROR: The owner already has 2 permits!");
      return;
    }

    //check that the car is new (no such car in the system)

    boolean pre4 = searchCar(regNr);

    if(pre4)
    {
      System.out.println("ERROR: The car already exists");
      return;
    }
```

```
      //add new permit
     try {
      String sqlStmt="INSERT into permits
(permitNr,regNr,area,Employee_id) VALUES(?,?,?,?)";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setInt(1, permitNr);
      pStmt.setString(2, regNr);
      pStmt.setString(3, area);
      pStmt.setInt(4, id);
      pStmt.executeUpdate();
     } catch (SQLException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
    }
   }

  public void replacePermit(int oldPermitNr,int newPermitNr,
                         String regNr, String area)
    {
      boolean pre= searchPermit(oldPermitNr);
      if(!pre) {
        System.out.println("Permit number does not exist");
        return;
      }
      String sqlStmt = "SELECT Employee_id FROM permits WHERE
permitNr = ?";
      try {
        PreparedStatement pstmt  = con.prepareStatement(sqlStmt);
        pstmt.setInt(1,oldPermitNr);
        ResultSet rs = pstmt.executeQuery();
        rs.first();
        int id = rs.getInt("Employee_id");
        rs.close();
        terminatePermit(oldPermitNr);
        addPermit(newPermitNr,area,regNr,id);
     }
     catch( SQLException e ) {
          System.out.println(e.getMessage());
          e.printStackTrace();
       }
    }
   public void changePhoneNumber(int id, String phone)
   {
     boolean pre1 = searchEmployee(id);

     if(!pre1)
     {
       System.out.println("ERROR: The employee number does not exist!");
       return;
     }

     try
     {
       String sqlStmt="UPDATE employees SET phone = ? WHERE
Employee_id = ?";
       PreparedStatement pStmt = con.prepareStatement(sqlStmt);
       pStmt.setString(1, phone);
       pStmt.setInt(2, id);
       pStmt.executeUpdate();

     } catch (SQLException e) {
            System.out.println(e.getMessage());

           e.printStackTrace();
    }
   }

 public void getDetailsOfEmployeeByPermit(int permitNr){

    boolean per1 = searchPermit(permitNr);

     if(!per1)
     {
      System.out.println("Permit number does not exist");
      return;
     }

    try
     {
    String  sqlStmt ="SELECT e.Employee_id, e.Employee_name, e.phone "
+
                  "FROM permits p ,employees e " +
                  "WHERE(p.Employee_id = e.Employee_id)" +
                  "And (p.permitNr = ?) ";
    PreparedStatement pStmt = con.prepareStatement(sqlStmt);
    pStmt.setInt(1, permitNr);
    ResultSet rs = pStmt.executeQuery();
    while (rs.next()) {

       int id = rs.getInt("e.Employee_id");
       String phone = rs.getString("e.phone");
       String name = rs.getString("e.Employee_name");

       System.out.println("Employee[ id: " + id + " , name: " + name
            + " , phone: " + phone + "]");
     }

    } catch (SQLException e) {
           System.out.println(e.getMessage());
          e.printStackTrace();
    }
  }
}
```

Appendix B

13.1 USE Case Description

Use Case Name: Add a Warehouse

Trigger/Goal: To enter the details of a new warehouse

Actors: Operator

Main Flow:

1. Operator enters warehouse's ID.
2. System validates that ID is new.
3. Operator enters phone number.
4. System saves the details.

Extensions:

2a.ID is not new:

2a1. System notifies Operator, and terminates the use case.

Use Case Name: Add new part

Trigger/Goal: To enter the details of new product

Actors: Operator

Main Flow:

1. Operator enters part 's ID.
2. System validates that ID is new.
3. Operator enters description and unit.
4. System saves the details.

Extensions:

2a.ID is not new:

2a1. System notifies Operator, and terminates the use case

Use Case Name: Add Shelf Location

Trigger/Goal: To enter the details of new Shelf Location and Store the quantity of the particular part.

Actors: Operator.

Main Flow:

1. Operator enters the warehouse's ID.
2. The System validates that ID's exists.
3. Operator enters Shelf Location's ID.
4. System validates that ID is new with that warehouse.
5. Operator enters Shelf Location's type.
6. The System validates that type is valid.
7. System saves the details.

Extensions:

2a. warehouse's ID does not exist:
 2a1. System notifies Operator, and terminates the use case.

2b. part's ID does not exist:
 2b1. System notifies Operator, and terminates the use case.

4a. Shelf Location's ID is not new:
 4a1. System notifies Operator, and terminates the use case.

6a. Shelf Location's type is not valid:
 6a1. System notifies Operator, and terminates the use case.

Use Case Name: Store Part.

Trigger/Goal: To Store the quantity of a part to existing Shelf Location .

Actors: Operator.

Main Flow:

1. Operator enters the Shelf Location's ID and part's ID.

2.The System validates that ID's exists.

3.Operator enters quantity of that part.

4.System update the details.

Extensions:

2a.Shelf Location's ID does not exist:

2a1. System notifies Operator, and terminates the use case.

2b.part's ID does not exist:

2b1. System notifies Operator, and terminates the use case.

Use Case Name: Get the Details where Part is Stored.

Trigger/Goal: The operator wants find out where the part is stored , given the product ID.

Actors: Operator.

Main Flow:

1. Operator enters part's ID.

2.System validates that ID is exist and the set of Shelf Location is not empty for that part.

3.System retrieves and displays the part's' id ,description and unit.

--- System Do steps 4 for each of the Shelf Location.

4.System retrieves the warehouse's ID , Shelf Location's ID , quantity , Shelf Location's type

5.System displays the details list.

Extensions:

2a.ID does not exist:

2a1. System notifies Operator, and terminates the use case.

2b.The Shelf Location set is empty for a particular product:

2b1. System notifies Operator, and terminates the use case.

13.2. Original Source Code of Parts in Warehouses System.

13.2.1. Hibernate

- **Warehouse**

```
public class Warehouse {
   private String warehouseCode;
   private String phone;

   public Warehouse() {
   }

   public Warehouse(String warehouseCode, String phone) {
      this.warehouseCode = warehouseCode;
      this.phone = phone;
   }

   public String getWarehouseCode() {
      return warehouseCode;
   }


   public void setWarehouseCode(String warehouseCode) {
      this.warehouseCode = warehouseCode;
   }


   public String getPhone() {
      return phone;
   }

   public void setPhone(String phone) {
      this.phone = phone;
   }

   @Override
   public boolean equals(Object obj) {
   if (!(obj instanceof Warehouse)) return false;
   Warehouse other = (Warehouse) obj;
   return this.warehouseCode.equals(other.warehouseCode) &&
        this.phone.equals(other.phone);
   }
}
```

- **Warehouse.hbm.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
  <class name="Warehouse" table="warehouse">
   <id column="W_ID" name="warehouseCode">
     <generator class="assigned"/>
   </id>
   <property name="phone"/>
  </class>
</hibernate-mapping>
```

- **Part**

```
public class Part {

   private String partCode;
   private String description;
   private String unit;

   public Part() {
   }

   public Part(String partCode, String description, String unit) {
      this.partCode = partCode;
      this.description = description;
      this.unit = unit;
   }

   /**
    * @return the partCode
    */
   public String getPartCode() {
      return partCode;
   }

   /**
    * @param partCode the partCode to set
    */
   public void setPartCode(String partCode) {
      this.partCode = partCode;
   }

   /**
    * @return the description
    */
   public String getDescription() {
      return description;
   }

   /**
    * @param description the description to set
    */
   public void setDescription(String description) {
      this.description = description;
   }

   /**
    * @return the unit
    */
   public String getUnit() {
      return unit;
   }

   /**
    * @param unit the unit to set
    */
   public void setUnit(String unit) {
      this.unit = unit;
   }
}
```

- **Part.hbm.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
  <class name="Part" table="part">
   <id column="P_ID" name="partCode">
     <generator class="assigned"/>
   </id>
   <property name="description"/>
   <property column="Unit" name="unit"/>
  </class>
</hibernate-mapping>
```

- **ShelfLocationPK**

```
import java.io.Serializable;

public class ShelfLocationPK implements Serializable {

  private Warehouse warehouse;
   private String shelfLocationCode;

   public ShelfLocationPK() {
   }

   public ShelfLocationPK(Warehouse warehouse, String
shelfLocationCode) {
      this.warehouse = warehouse;
```

```
        this.shelfLocationCode = shelfLocationCode;
    }

    public void setWarehouse(Warehouse warehouse) {
        this.warehouse = warehouse;
    }

    /**
     * @return the warehouse
     */
    public Warehouse getWarehouse() {
        return warehouse;
    }

    /**
     * @return the shelfLocationCode
     */
    public String getShelfLocationCode() {
        return shelfLocationCode;
    }

    /**
     * @param shelfLocationCode the shelfLocationCode to set
     */
    public void setShelfLocationCode(String shelfLocationCode) {
        this.shelfLocationCode = shelfLocationCode;
    }

    @Override
    public boolean equals(Object obj) {
    if (!(obj instanceof ShelfLocationPK)) return false;
    ShelfLocationPK other = (ShelfLocationPK) obj;
    return this.shelfLocationCode.equals(other.shelfLocationCode) &&
            this.warehouse.equals(other.warehouse);
}

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 67 * hash + (this.warehouse != null ?
this.warehouse.hashCode() : 0);
        hash = 67 * hash + (this.shelfLocationCode != null ?
this.shelfLocationCode.hashCode() : 0);
        return hash;
    }

}
```

- **ShelfLocation.java**

```
public class ShelfLocation  {

    private ShelfLocationPK primarykey;
    private String typeShelf;
    private Part part;
    private int qty;

    public ShelfLocation() {
    }

    public ShelfLocation(ShelfLocationPK primarykey ,String typeShelf) {
        this.primarykey = primarykey;
        this.typeShelf = typeShelf;
        this.part = null;
        this.qty = 0;
    }

    public ShelfLocationPK getPrimarykey() {
        return primarykey;
    }

    public void setPrimarykey(ShelfLocationPK primarykey) {
        this.primarykey = primarykey;
    }
    public void setQty(int qty) {
        this.qty = qty;
    }

    public int getQty() {
        return qty;
    }

    /**
     * @return the typeShelf
     */
    public String getTypeShelf() {
        return typeShelf;
    }

    /**
     * @param typeShelf the typeShelf to set
     */
    public void setTypeShelf(String typeShelf) {
        this.typeShelf = typeShelf;
    }

    /**
     * @return the part
     */
    public Part getPart() {
        return part;
    }

    /**
     * @param part the part to set
     */
    public void setPart(Part part) {
        this.part = part;
    }

    @Override
    public String toString(){
       String x = primarykey.getWarehouse().getWarehouseCode()+
               "\t\t"+ primarykey.getShelfLocationCode()+
               "\t\t"+qty+"\t\t"+typeShelf+"\n";
       return x;
    }
}
```

- **ShelfLocation.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="ShelfLocation" table="shelflocation">
   <composite-id name="primarykey" class="ShelfLocationPK">
        <key-property name="shelfLocationCode" column="S_ID"/>
        <key-many-to-one name="warehouse" class="Warehouse"
column="W_ID"/>
   </composite-id>
      <property name="typeShelf" column="type"/>
      <property name="Qty" column="qty"/>
      <many-to-one name="part" class="Part" column="P_ID"/>
   </class>
</hibernate-mapping>
```

- **hibernate.cfg.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost/partwarehouse_jdbc</propert
y>
        <property name="connection.username">root</property>
        <property name="connection.password"> root </property>

       <!-- session properties -->
        <property
name="hibernate.current_session_context_class">org.hibernate.context.Thr
eadLocalSessionContext</property>
```

```
     <!-- JDBC connection pool (use the built-in) -->
      <property name="connection.pool_size">1</property>

      <!-- SQL dialect -->
      <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>

      <!-- Echo all executed SQL to stdout -->
      <property name="show_sql">False</property>

      <!-- Mapping files -->
      <mapping resource="Warehouse.hbm.xml"/>
      <mapping resource="Part.hbm.xml"/>
      <mapping resource="ShelfLocation.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

- **PartInWareHouse.java**

```
import java.util.List;
import org.hibernate.*;
public class PartInWareHouse{

   public Warehouse searchWarehouse(String id){

              Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      Warehouse warehouse = (Warehouse)session.get(Warehouse.class,id);
      return warehouse;
   }

  public void addWarehouse(String id, String phone)
  {
     //check precondition
    Warehouse warehouse = searchWarehouse(id);
    boolean pre1 = (warehouse == null);

    if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     warehouse = new Warehouse(id,phone);
     session.save(warehouse);
     session.getTransaction().commit();
}



  public Part searchPart(String id)
   {
              Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      Part part = (Part)session.get(Part.class,id);
      return part;
   }

  public void addPart(String id, String desc , String unit)
  {
     //check precondition
    Part part = searchPart(id);
    boolean pre1 = (part == null);

    if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     part = new Part(id,desc,unit);
     session.save(part);
     session.getTransaction().commit();
}

   public ShelfLocation searchShelfLocation(ShelfLocationPK id)
   {
      Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      ShelfLocation shelfLocation =
(ShelfLocation)session.get(ShelfLocation.class,id);
      return shelfLocation;
   }

   public void addShelfLocation(String s_id,String w_id,String type)
   {

   Warehouse warehouse = searchWarehouse(w_id);
   boolean pre1 = (warehouse == null);
   if(pre1)
   {
       System.out.println("ERROR: The warehouse number does not exist!");
       return;
    }

   ShelfLocationPK pk = new ShelfLocationPK(warehouse,s_id);

   ShelfLocation sh = searchShelfLocation(pk);
   boolean pre2 = (sh == null);

   if(!pre2)
   {
        System.out.println("ERROR: The ShelfLocation number already
exists!");
        return;
     }

     if(!type.equals("Single Accsess") && !type.equals("Double Accsess")){
       System.out.println("ERROR: The type is not a valid !");
       return;
     }
      Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      ShelfLocation  s = new ShelfLocation(pk,type);
      session.save(s);
      session.getTransaction().commit();
  }

  public void storPart(String s_id,String w_id,String p_id ,int qty) {

   Warehouse warehouse = searchWarehouse(w_id);
   boolean pre1 = (warehouse == null);
   if(pre1){
       System.out.println("ERROR: The warehouse number does not exist!");
       return;
    }
   ShelfLocationPK pk = new ShelfLocationPK(warehouse,s_id);
   ShelfLocation sh = searchShelfLocation(pk);
   boolean pre2 = (sh == null);
   if(pre2)    {
        System.out.println("ERROR: The ShelfLocation number number does
not exist!");
        return;
   }
    Part part = searchPart(p_id);
    boolean pre3 = (part == null);
    if(pre3){
       System.out.println("ERROR: The part  number does not exist!");
       return;
     }

     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     sh.setQty(qty);
     sh.setPart(part);
     session.update(sh);
     session.getTransaction().commit();
  }

  public void getDetaiWhereItemIsStored(String id){

   Part part = searchPart(id);
   boolean pre= (part == null);
   if(pre){
        System.out.println("Part number does not exist");
        return;
      }
      Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      Query q = session.createQuery("from ShelfLocation as c where
c.part.partCode=:id1");
      q.setString("id1",id);
      List cResult = q.list();
      if(cResult.isEmpty())
      {
         System.out.println("list is empty");
         return;
      }
      int qty = 0;
      String whereItme= null;
      for(Object o:  cResult)
      {
         qty+=((ShelfLocation)o).getQty();
      }

    whereItme ="Part#         :" + part.getPartCode() +"\n"+
            "Description    :" + part.getDescription()+"\n"+
            "Unit           :" + part.getUnit()+"\n"+
            "Total on Hand  :" + qty +"\n"+
            "Warehouse\t\tShelfLocation\t\tQty\t\tLocation Type\n";
      for(Object o:  cResult)
      {
          whereItme += o.toString();
      }


        System.out.println(whereItme);

   }
}
```

13.2.2. JDBC

- **PartInWareHouse.java**

```
package carparkinjdbc;
/**
 *
 * @author ALharthy
 */

import java.sql.*;

public class CPSystem
{
    public static String url = "jdbc:mysql://localhost/partwarehouse_jdbc";
    public static String dbdriver = "com.mysql.jdbc.Driver";
    public static String username = "root";
    public static String password = "asomalia";
    public static String sqlpart = "SELECT * FROM part WHERE P_ID = ?";

    static Connection con;

    public CPSystem()
    {
        con = getConnection();
    }

     public void ConnClose(){
       try
                 {
                             con.close();
                 }
                  catch(SQLException ex)
                 {
                                            System.err.println("SQLException: " +
ex.getMessage());
                 }
     }

    public static Connection getConnection()
              {

                try
                {

              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(java.lang.ClassNotFoundException e)
                {
                            System.err.print("ClassNotFoundException: ");
                            System.err.println(e.getMessage());
                }

                try
                {
                            con = DriverManager.getConnection(url,username,
password);
                }
                 catch(SQLException ex)
                {
                                            System.err.println("SQLException: " +
ex.getMessage());
                }
                             return con;
              }

    public boolean search(String id , String sql){

      boolean isExist = true;
      try
      {
           PreparedStatement pstmt  = con.prepareStatement(sql);
           pstmt.setString(1,id);
           ResultSet rs = pstmt.executeQuery();
           isExist = rs.next();
      }
      catch( SQLException e ) {
           System.out.println(e.getMessage());
           e.printStackTrace();
        }

       return isExist;
    }

   public void addWarehouse(String id, String phone)
   {
      //check precondition
      String sql = "SELECT * FROM warehouse WHERE W_ID = ?";
      boolean pre1 = search(id , sql);
      if(pre1)
      {
         System.out.println("ERROR: The id already exists!");
         return;
      }
      try {
       String sqlStmt="INSERT into warehouse VALUES(?,?)";
       PreparedStatement pStmt = con.prepareStatement(sqlStmt);
       pStmt.setString(1, id);
       pStmt.setString(2, phone);
       pStmt.executeUpdate();
      } catch (SQLException e) {
             System.out.println(e.getMessage());
            e.printStackTrace();
    }
}

    //---------

    public void addPart(String id, String  desc ,String unit)
    {

       boolean pre1= search(id , sqlpart);
       if(pre1)
        {
          System.out.println("ERROR: The part number already exists!");
          return;
     }

    try {
     String sqlStmt="INSERT into part VALUES(?,?,?)";
     PreparedStatement pStmt = con.prepareStatement(sqlStmt);
     pStmt.setString(1, id);
     pStmt.setString(2, desc);
     pStmt.setString(3, unit);
     pStmt.executeUpdate();
    } catch (SQLException e) {
            System.out.println(e.getMessage());
           e.printStackTrace();
   }
  }

   public boolean searchShelfLocation(String w_id , String s_id){

    String sql = "SELECT * FROM shelflocation WHERE W_ID = ? AND
S_ID = ? ";
    boolean isExist = true;
    try
    {
         PreparedStatement pstmt  = con.prepareStatement(sql);
         pstmt.setString(1,w_id);
         pstmt.setString(2,s_id);
         ResultSet rs = pstmt.executeQuery();
         isExist = rs.next();
    }
    catch( SQLException e ) {
         System.out.println(e.getMessage());
         e.printStackTrace();
     }
     return isExist;
   }

  public void addShelfLocation(String w_id, String s_id,String type){

     boolean pre1= searchShelfLocation(w_id,s_id);
     if(pre1)
     {
       System.out.println("ERROR: The ShelfLocation number already
exists!");
       return;
     }

    try {
     String sqlStmt="INSERT into ShelfLocation (S_ID ,W_ID,type)
VALUES(?,?,?)";
     PreparedStatement pStmt = con.prepareStatement(sqlStmt);
     pStmt.setString(1, s_id);
     pStmt.setString(2, w_id);
     pStmt.setString(3, type);
     pStmt.executeUpdate();
    } catch (SQLException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
   }
  }


  public void storPart(String w_id, String s_id,String p_id, int qty)  {

    boolean pre1= searchShelfLocation(w_id,s_id);
     if(pre1)
     {
       System.out.println("ERROR: The ShelfLocation number already
exists!");
       return;
     }

     boolean pre2= search(p_id , sqlpart);

     if(pre2) {
       System.out.println("ERROR: part number does not exist!");
       return;
     }
    try{

      String sqlStmt="UPDATE ShelfLocation SET P_ID = ?,Qty = ?
WHERE W_ID = ? AND S_ID = ?";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1, p_id);
          pStmt.setInt(2, qty);
          pStmt.setString(3, w_id);
          pStmt.setString(4, s_id);
          pStmt.executeUpdate();

       } catch (SQLException e) {
              System.out.println(e.getMessage());
             e.printStackTrace();
     }
    }
   public void getDetailsOfPart(String p_id){

      boolean per1= search(p_id , sqlpart);
      if(!per1)
      {
       System.out.println("part number does not exist");
       return;
      }

     try
      {
     String  sqlStmt ="SELECT s.W_ID, s.S_id,
s.Qty,s.type,p.P_ID,p.Desc,p.Unit " +
                   "FROM shelflocation s ,part p " +
                   "WHERE(p.P_ID = s.P_ID) " +
                   "AND (p.P_ID=?)";
     PreparedStatement pStmt = con.prepareStatement(sqlStmt);
     pStmt.setString(1, p_id);
     ResultSet rs = pStmt.executeQuery();
     boolean x = true;
     String whereItme=null;
     int qty=0;
     while (rs.next()) {
     if(x)
      whereItme ="Part#         :" + rs.getString("p.P_ID")+"\n"+
            "Description    :" + rs.getString("p.Desc")+"\n"+
            "Unit           :" +rs.getString("p.Unit")+"\n";
     qty+= rs.getInt("s.Qty");
      }

     whereItme +=  "Total on Hand :" + qty
+"\nWarehouse\t\tShelfLocation\t\tQty\t\tLocation Type\n";
     rs.beforeFirst();
     while (rs.next()) {
        whereItme += rs.getString("s.W_ID")+"\t\t\t"+
               rs.getString("s.S_id")+"\t\t\t"+
               rs.getString("s.Qty")+"\t\t\t"+
               rs.getString("s.type")+"\n";
        }

        System.out.println(whereItme);
      } catch (SQLException e) {
              System.out.println(e.getMessage());
             e.printStackTrace();
     }
   }
 }
```

12.2.The Source Code of Parts in Warehouses System. After Programmatic Adjustment

13.3.1. Hibernate

- **Warehouse.java**

```
public class Warehouse {
   private String warehouseCode;
   private String phone;

   public Warehouse() {
   }

   public Warehouse(String warehouseCode, String phone) {
      this.warehouseCode = warehouseCode;
      this.phone = phone;
   }

   public String getWarehouseCode() {
      return warehouseCode;
```

```
    }


    public void setWarehouseCode(String warehouseCode) {
        this.warehouseCode = warehouseCode;
    }


    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public boolean equals(Object obj) {
    if (!(obj instanceof Warehouse)) return false;
    Warehouse other = (Warehouse) obj;
    return this.warehouseCode.equals(other.warehouseCode) &&
            this.phone.equals(other.phone);
    }
}
```

- **Warehouse.hbm.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
  <class name="Warehouse" table="warehouse">
    <id column="W_ID" name="warehouseCode">
      <generator class="assigned"/>
    </id>
    <property name="phone"/>
  </class>
</hibernate-mapping>
```

- **Part.java**

```
public class Part {

    private String partCode;
    private String description;
    private String unit;

    public Part() {
    }

    public Part(String partCode, String description, String unit) {
        this.partCode = partCode;
        this.description = description;
        this.unit = unit;
    }

    /**
     * @return the partCode
     */
    public String getPartCode() {
        return partCode;
    }

    /**
     * @param partCode the partCode to set
     */
    public void setPartCode(String partCode) {
        this.partCode = partCode;
    }

    /**
     * @return the description
     */
    public String getDescription() {
        return description;
    }

    /**
     * @param description the description to set
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * @return the unit
     */
    public String getUnit() {
        return unit;
    }

    /**
     * @param unit the unit to set
     */
    public void setUnit(String unit) {
        this.unit = unit;
    }

    @Override
    public boolean equals(Object obj) {
    if (!(obj instanceof Part)) return false;
    Part other = (Part) obj;
    return this.partCode.equals(other.partCode);
    }
}
```

- **Part.hbm.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
  <class name="Part" table="part">
    <id column="P_ID" name="partCode">
      <generator class="assigned"/>
    </id>
    <property name="description"/>
    <property column="Unit" name="unit"/>
  </class>
</hibernate-mapping>
```

- **ShelfLocationPK.java**

```
import java.io.Serializable;

public class ShelfLocationPK implements Serializable {

    private Warehouse warehouse;
    private String shelfLocationCode;

    public ShelfLocationPK() {
    }

    public ShelfLocationPK(Warehouse warehouse, String
shelfLocationCode) {
        this.warehouse = warehouse;
        this.shelfLocationCode = shelfLocationCode;
    }

    public void setWarehouse(Warehouse warehouse) {
        this.warehouse = warehouse;
    }

    /**
     * @return the warehouse
     */
    public Warehouse getWarehouse() {
        return warehouse;
    }

    /**
     * @return the shelfLocationCode
     */
    public String getShelfLocationCode() {
        return shelfLocationCode;
    }

    /**
     * @param shelfLocationCode the shelfLocationCode to set
     */
    public void setShelfLocationCode(String shelfLocationCode) {
        this.shelfLocationCode = shelfLocationCode;
    }

    @Override
    public boolean equals(Object obj) {
    if (!(obj instanceof ShelfLocationPK)) return false;
    ShelfLocationPK other = (ShelfLocationPK) obj;
    return this.shelfLocationCode.equals(other.shelfLocationCode) &&
           this.warehouse.equals(other.warehouse);
}

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 67 * hash + (this.warehouse != null ?
this.warehouse.hashCode() : 0);
        hash = 67 * hash + (this.shelfLocationCode != null ?
this.shelfLocationCode.hashCode() : 0);
        return hash;
    }

}
```

- **ShelfLocation.java**

```
import java.util.HashSet;
import java.util.Set;

public class ShelfLocation  {

    private ShelfLocationPK primarykey;
    private String typeShelf;
    private Set<Item> items;

    public ShelfLocation() {
    }

    public ShelfLocation(ShelfLocationPK primarykey ,String typeShelf) {
        this.primarykey = primarykey;
        this.typeShelf = typeShelf;
        this.items = new HashSet<Item>();
    }

    public ShelfLocationPK getPrimarykey() {
        return primarykey;
    }

    public void setPrimarykey(ShelfLocationPK primarykey) {
        this.primarykey = primarykey;
    }

    public Set<Item> getItems() {
        return items;
    }

    public void setItems(Set<Item> items) {
        this.items = items;
    }

    /**
     * @return the typeShelf
     */
    public String getTypeShelf() {
        return typeShelf;
    }

    /**
     * @param typeShelf the typeShelf to set
     */
    public void setTypeShelf(String typeShelf) {
        this.typeShelf = typeShelf;
    }
}
```

- **ShelfLocation.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="ShelfLocation" table="shelflocation">
   <composite-id name="primarykey" class="ShelfLocationPK">
        <key-property name="shelfLocationCode" column="S_ID"/>
        <key-many-to-one name="warehouse" class="Warehouse"
column="W_ID"/>
   </composite-id>
      <property name="typeShelf" column="type"/>

    <set name="items" table="item"  inverse="true" cascade="all">
    <key>
    <column name="W_ID" />
    <column name="S_ID"/>
    </key>
    <one-to-many class="Item"/>
    </set>

   </class>
</hibernate-mapping>
```

- **ItemPK.java**

```
import java.io.Serializable;
public class ItemPK implements Serializable {
    private ShelfLocation sh;
    private Part part;

    public ItemPK() {
    }

   public ItemPK(ShelfLocation sh, Part part) {
        this.sh = sh;
        this.part = part;
    }
    public ShelfLocation getSh() {
        return sh;
    }
    public void setSh(ShelfLocation sh) {
        this.sh = sh;
    }
    public Part getPart() {
        return part;
    }
    public void setPart(Part part) {
        this.part = part;
    }

    @Override
    public boolean equals(Object obj) {
    if (!(obj instanceof ItemPK)) return false;
    ItemPK other = (ItemPK) obj;
    return this.part.equals(other.part) &&
           this.sh.equals(other.sh);
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 37 * hash + (this.sh != null ? this.sh.hashCode() : 0);
        hash = 37 * hash + (this.part != null ? this.part.hashCode() : 0);
        return hash;
    }
}
```

- **Item.java**

```
public class Item  {

    private ItemPK primarykey;
    private int qty;
```

```
   public Item() {
   }

   public Item(ItemPK primarykey, int qty) {
       this.primarykey = primarykey;
       this.qty = qty;
   }

   public ItemPK getPrimarykey() {
       return primarykey;
   }

   public void setPrimarykey(ItemPK primarykey) {
       this.primarykey = primarykey;
   }

   /**
    * @return the qty
    */
   public int getQty() {
       return qty;
   }

   /**
    * @param qty the qty to set
    */
   public void setQty(int qty) {
       this.qty = qty;
   }

   @Override
   public String toString(){
      String x =
primarykey.getSh().getPrimarykey().getWarehouse().getWarehouseCode()+
             "\t\t"+
primarykey.getSh().getPrimarykey().getShelfLocationCode()+
             "\t\t"+qty+"\t\t"+primarykey.getSh().getTypeShelf()+"\n";
      return x;
   }
}
```

- **Item.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
  PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="Item" table="item">
  <composite-id name="primarykey" class="ItemPK">
      <key-many-to-one name="sh" class="ShelfLocation">
             <column name="W_ID"/>
             <column name="S_ID"/>
         </key-many-to-one>
      <key-many-to-one name="part" class="Part" column="P_ID"/>
  </composite-id>
   </class>
</hibernate-mapping>
```

- **hibernate.cfg.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <!-- Database connection settings -->
      <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property
name="connection.url">jdbc:mysql://localhost/partwarehouse_jdbc</propert
y>
      <property name="connection.username">root</property>
      <property name="connection.password">asomalia</property>

     <!-- session properties -->
      <property
name="hibernate.current_session_context_class">org.hibernate.context.Thr
eadLocalSessionContext</property>
      <!-- JDBC connection pool (use the built-in) -->
       <property name="connection.pool_size">1</property>

       <!-- SQL dialect -->
       <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>

       <!-- Echo all executed SQL to stdout -->
       <property name="show_sql">False</property>

       <!-- Mapping files -->
       <mapping resource="Warehouse.hbm.xml"/>
       <mapping resource="Part.hbm.xml"/>
       <mapping resource="ShelfLocation.hbm.xml"/>
       <mapping resource="Item.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

- **PartInWareHouse.java**

```
import java.util.List;
import org.hibernate.*;
public class PartInWareHouse
{

   public Warehouse searchWarehouse(String id){

             Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Warehouse warehouse = (Warehouse)session.get(Warehouse.class,id);
       return warehouse;
   }

  public void addWarehouse(String id, String phone)
  {
     //check precondition
    Warehouse warehouse = searchWarehouse(id);
    boolean pre1 = (warehouse == null);

    if(!pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     warehouse = new Warehouse(id,phone);
     session.save(warehouse);
     session.getTransaction().commit();
}



  public Part searchPart(String id)
   {
             Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Part part = (Part)session.get(Part.class,id);
       return part;
   }

  public void addPart(String id, String desc , String unit)
  {
     //check precondition
    Part part = searchPart(id);
    boolean pre1 = (part == null);

    if(!pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     part = new Part(id,desc,unit);
     session.save(part);
     session.getTransaction().commit();
}
```

```
   public ShelfLocation searchShelfLocation(ShelfLocationPK id)
   {
       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       ShelfLocation shelfLocation =
(ShelfLocation)session.get(ShelfLocation.class,id);
       return shelfLocation;
   }


   public Item searchItem(ItemPK id)
   {
       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Item item = (Item)session.get(Item.class,id);
       return item;
   }

   public void addShelfLocation(String s_id,String w_id,String type)
   {

   Warehouse warehouse = searchWarehouse(w_id);
   boolean pre1 = (warehouse == null);
   if(pre1)
   {
        System.out.println("ERROR: The warehouse number does not exist!");
        return;
    }

   ShelfLocationPK pk = new ShelfLocationPK(warehouse,s_id);

   ShelfLocation sh = searchShelfLocation(pk);
   boolean pre2 = (sh == null);

   if(!pre2)
   {
         System.out.println("ERROR: The ShelfLocation number already
exists!");
         return;
    }

     if(!type.equals("Single Accsess") && !type.equals("Double Accsess")){
        System.out.println("ERROR: The type is not a valid !");
        return;
     }
       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       ShelfLocation  s = new ShelfLocation(pk,type);
       session.save(s);
       session.getTransaction().commit();
  }

   public void storPart(String s_id,String w_id,String p_id ,int qty) {

   Warehouse warehouse = searchWarehouse(w_id);
   boolean pre1 = (warehouse == null);
   if(pre1){
        System.out.println("ERROR: The warehouse number does not exist!");
        return;
    }
   ShelfLocationPK pk = new ShelfLocationPK(warehouse,s_id);
   ShelfLocation sh = searchShelfLocation(pk);
   boolean pre2 = (sh == null);
   if(pre2)    {
         System.out.println("ERROR: The ShelfLocation number number does
not exist!");
         return;
   }
    Part part = searchPart(p_id);
    boolean pre3 = (part == null);
    if(pre3){
        System.out.println("ERROR: The part  number does not exist!");
        return;
     }
   ItemPK pk1 = new ItemPK(sh,part);
   Item item = searchItem(pk1);
   boolean pre4 = (item == null);
   if(!pre4){
         System.out.println("ERROR: The item number already exists!");
        return;
      }
       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Item  s = new Item(pk1,qty);
       sh.getItems().add(s);
       session.save(s);
       session.getTransaction().commit();
   }

   public void getDetaiWhereItemIsStored(String id){

    Part part = searchPart(id);
    boolean pre= (part == null);
    if(pre){
          System.out.println("Part number does not exist");
          return;
      }
       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Query q = session.createQuery("from Item as c where
c.primarykey.part.partCode=:id1");
       q.setString("id1",id);
       List cResult = q.list();
       if(cResult.isEmpty())
       {
          System.out.println("list is empty");
          return;
       }
       int qty = 0;
       String whereItme= null;
       for(Object o:  cResult)
       {
          qty+=((Item)o).getQty();
       }

    whereItme ="Part#         :" + part.getPartCode() +"\n"+
              "Description    :" + part.getDescription()+"\n"+
              "Unit           :" + part.getUnit()+"\n"+
              "Total on Hand  :" + qty +"\n"+
              "Warehouse\t\tShelfLocation\t\tQty\t\tLocation Type\n";
       for(Object o:  cResult)
       {
           whereItme += o.toString();
       }
          System.out.println(whereItme);
   }
}
```

13.3.2. JDBC

- **PartInWareHouse.java**

```
import java.sql.*;

   public class PartInWareHouse {
    public static String url = "jdbc:mysql://localhost/partwarehouse_jdbc";
    public static String dbdriver = "com.mysql.jdbc.Driver";
    public static String username = "root";
    public static String password = "asomalia";
    public static String sqlpart = "SELECT * FROM part WHERE P_ID = ?";

    static Connection con;

    public CPSystem()
    {
       con = getConnection();
    }

     public void ConnClose(){
       try
              {
                           con.close();
              }
              catch(SQLException ex)
              {
                                      System.err.println("SQLException: " +
ex.getMessage());
              }
    }

   public static Connection getConnection()
              {

                try
                {

              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(java.lang.ClassNotFoundException e)
                {
                            System.err.print("ClassNotFoundException: ");
                            System.err.println(e.getMessage());
                }

                try
                {
                            con = DriverManager.getConnection(url,username,
password);
                }
                 catch(SQLException ex)
                {
                                        System.err.println("SQLException: " +
ex.getMessage());
                }
                          return con;
              }

   public boolean search(String id , String sql){

     boolean isExist = true;
     try
     {
         PreparedStatement pstmt  = con.prepareStatement(sql);
         pstmt.setString(1,id);
         ResultSet rs = pstmt.executeQuery();
         isExist = rs.next();
     }
     catch( SQLException e ) {
         System.out.println(e.getMessage());
         e.printStackTrace();
       }

      return isExist;
   }

  public void addWarehouse(String id, String phone)
   {
     //check precondition
     String sql = "SELECT * FROM warehouse WHERE W_ID = ?";
     boolean pre1 = search(id , sql);
     if(pre1)
     {
       System.out.println("ERROR: The id already exists!");
       return;
     }
     try {
      String sqlStmt="INSERT into warehouse VALUES(?,?)";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1, id);
      pStmt.setString(2, phone);
      pStmt.executeUpdate();
     } catch (SQLException e) {
           System.out.println(e.getMessage());
          e.printStackTrace();
   }
}

   //---------

   public void addPart(String id, String  desc ,String unit)
   {

      boolean pre1= search(id , sqlpart);
      if(pre1)
      {
        System.out.println("ERROR: The part number already exists!");
        return;
      }

     try {
      String sqlStmt="INSERT into part VALUES(?,?,?)";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1, id);
      pStmt.setString(2, desc);
      pStmt.setString(3, unit);
      pStmt.executeUpdate();
     } catch (SQLException e) {
           System.out.println(e.getMessage());
          e.printStackTrace();
   }
  }

    public boolean searchShelfLocation(String w_id , String s_id){

     String sql = "SELECT * FROM shelflocation WHERE W_ID = ? AND
S_ID = ? ";
     boolean isExist = true;
     try
     {
         PreparedStatement pstmt  = con.prepareStatement(sql);
         pstmt.setString(1,w_id);
         pstmt.setString(2,s_id);
         ResultSet rs = pstmt.executeQuery();
         isExist = rs.next();
     }
     catch( SQLException e ) {
         System.out.println(e.getMessage());
         e.printStackTrace();
       }
      return isExist;
    }

   public void addShelfLocation(String w_id, String s_id,String type){

      boolean pre1= searchShelfLocation(w_id,s_id);
      if(pre1)
      {
        System.out.println("ERROR: The ShelfLocation number already
exists!");
        return;
      }

     try {
      String sqlStmt="INSERT into ShelfLocation (S_ID ,W_ID,type)
VALUES(?,?,?)";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1, s_id);
      pStmt.setString(2, w_id);
      pStmt.setString(3, type);
      pStmt.executeUpdate();
     } catch (SQLException e) {
           System.out.println(e.getMessage());
           e.printStackTrace();
   }
  }

    public boolean searchItem(String w_id , String s_id , String p_id){

     String sql = "SELECT * FROM item WHERE W_ID = ? AND S_ID = ?
AND P_ID = ?";
     boolean isExist = true;
     try
     {
         PreparedStatement pstmt  = con.prepareStatement(sql);
         pstmt.setString(1,w_id);
         pstmt.setString(2,s_id);
         pstmt.setString(3,p_id);
         ResultSet rs = pstmt.executeQuery();
         isExist = rs.next();
     }
     catch( SQLException e ) {
         System.out.println(e.getMessage());
         e.printStackTrace();
       }
      return isExist;
    }

   public void storPart(String w_id, String s_id,String p_id, int qty)
   {
      boolean pre1= searchItem(w_id,s_id,p_id);
      if(pre1)
      {
        System.out.println("ERROR: The item number already exists!");
        return;
      }

     try {
      String sqlStmt="INSERT into item  VALUES(?,?,?,?)";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1, w_id);
      pStmt.setString(2, s_id);
      pStmt.setString(3, p_id);
      pStmt.setInt(4, qty);
      pStmt.executeUpdate();

     } catch (SQLException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
   }
  }
 public void getDetailsOfPart(String p_id){

     boolean per1= search(p_id , sqlpart);
     if(!per1)
     {
       System.out.println("Permit number does not exist");
       return;
     }

    try
     {
    String  sqlStmt ="SELECT s.W_ID, s.S_id,
i.QTY,s.type,p.P_ID,p.description,p.Unit " +
                   "FROM shelflocation s ,part p , item i " +
                   "WHERE(p.P_ID = i.P_ID) AND (s.S_ID = i.S_ID) AND
(s.W_ID = i.W_ID) " +
                   "AND (i.P_ID=?)";
    PreparedStatement pStmt = con.prepareStatement(sqlStmt);
    pStmt.setString(1, p_id);
    ResultSet rs = pStmt.executeQuery();
    boolean x = true;
    String whereItme=null;
    int qty=0;
    while (rs.next()) {
    if(x){
     whereItme ="Part#         :" + rs.getString("p.P_ID")+"\n"+
            "Description    :" + rs.getString("p.description")+"\n"+
            "Unit           :" +rs.getString("p.Unit")+"\n";
    qty+= rs.getInt("i.QTY");}
    }

    whereItme +=  "Total on Hand :" + qty
+"\nWarehouse\t\tShelfLocation\t\tQty\t\tLocation Type\n";
    rs.beforeFirst();
    while (rs.next()) {
       whereItme += rs.getString("s.W_ID")+"\t\t\t"+
              rs.getString("s.S_id")+"\t\t\t"+
              rs.getString("i.QTY")+"\t\t\t"+
              rs.getString("s.type")+"\n";
       }

       System.out.println(whereItme);
     } catch (SQLException e) {
            System.out.println(e.getMessage());
           e.printStackTrace();
    }
  }
}
```

Appendix C

14.1 USE Case Description

Use Case Name: Add an Customer
Trigger/Goal: To enter the details of a new customer
Actors: Operator
Main Flow:

1. Operator enters employee's ID.
2. System validates that ID is new.
3. Operator enters name.
4. System saves the details.

Extensions:
2a.ID is not new:
2a1. System notifies Operator, and terminates the use case.

Use Case Name: Add new Product
Trigger/Goal: To enter the details of new product
Actors: Operator
Main Flow:

1. Operator enters product's ID.
2. System validates that ID is new.
3. Operator enters description and price and quantity.
4. System saves the details.

Extensions:
2a. ID is not new:
2a1. System notifies Operator, and terminates the use case.

Use Case Name: Add a standing order.
Trigger/Goal: Add new a standing order to an existing customer.
Actors: Operator

Main Flow:

1. Operator enters customer's ID and product's ID.
2. System validates that ID's are exist .
3. Operator enters order's ID
4. System validates that ID is new.
5. Operator enters quantity.
6. System check there is sufficient stock available.
7. System saves the details.

Extensions:
2a.customer's ID does not exist:
2a1. System notifies Operator, and terminates the use case.
2b.product's ID does not exist:
2b1. System notifies Operator, and terminates the use case.
4a.ID is not new:
4a1. System notifies Operator, and terminates the use case.
6a.Insufficient stock:
6a1. System notifies Operator, and terminates the use case.

```
          System.out.println("ERROR: The item number already exists!");
          return;
        }

       try {
        String sqlStmt="INSERT into item  VALUES(?,?,?,?)";
        PreparedStatement pStmt = con.prepareStatement(sqlStmt);
        pStmt.setString(1, w_id);
        pStmt.setString(2, s_id);
        pStmt.setString(3, p_id);
        pStmt.setInt(4, qty);
        pStmt.executeUpdate();

       } catch (SQLException e) {
               System.out.println(e.getMessage());
               e.printStackTrace();
     }
    }
  public void getDetailsOfPart(String p_id){

       boolean per1= search(p_id , sqlpart);
       if(!per1)
       {
         System.out.println("Permit number does not exist");
         return;
       }

      try
       {
      String  sqlStmt ="SELECT s.W_ID, s.S_id,
i.QTY,s.type,p.P_ID,p.description,p.Unit " +
                    "FROM shelflocation s ,part p , item i " +
                    "WHERE(p.P_ID = i.P_ID) AND (s.S_ID = i.S_ID) AND
(s.W_ID = i.W_ID) " +
                    "AND (i.P_ID=?)";
    PreparedStatement pStmt = con.prepareStatement(sqlStmt);
    pStmt.setString(1, p_id);
    ResultSet rs = pStmt.executeQuery();
    boolean x = true;
    String whereItme=null;
    int qty=0;
    while (rs.next()) {
    if(x){
     whereItme ="Part#          :" + rs.getString("p.P_ID")+"\n"+
               "Description    :" + rs.getString("p.description")+"\n"+
               "Unit           :" +rs.getString("p.Unit")+"\n";
    qty+= rs.getInt("i.QTY");}
    }

    whereItme +=  "Total on Hand :" + qty
+"\nWarehouse\t\tShelfLocation\t\tQty\t\tLocation Type\n";
    rs.beforeFirst();
    while (rs.next()) {
       whereItme += rs.getString("s.W_ID")+"\t\t\t"+
                rs.getString("s.S_id")+"\t\t\t"+
                rs.getString("i.QTY")+"\t\t\t"+
                rs.getString("s.type")+"\n";
       }

       System.out.println(whereItme);
     } catch (SQLException e) {
            System.out.println(e.getMessage());
           e.printStackTrace();
   }
  }
}
```

Appendix C

14.1 USE Case Description

Use Case Name: Add an Customer
Trigger/Goal: To enter the details of a new customer
Actors: Operator
Main Flow:

1. Operator enters employee's ID.
2. System validates that ID is new.
3. Operator enters name.
4. System saves the details.

Extensions:

2a.ID is not new:
2a1. System notifies Operator, and terminates the use case.

Use Case Name: Add new Product
Trigger/Goal: To enter the details of new product
Actors: Operator
Main Flow:

1. Operator enters product's ID.
2. System validates that ID is new.
3. Operator enters description and price and quantity.
4. System saves the details.

Extensions:

2a. ID is not new:
2a1. System notifies Operator, and terminates the use case.

Use Case Name: Add a standing order.
Trigger/Goal: Add new a standing order to an existing customer.
Actors: Operator

Main Flow:

1. Operator enters customer's ID and product's ID.
2. System validates that ID's are exist .
3. Operator enters order's ID
4. System validates that ID is new.
5. Operator enters quantity.
6. System check there is sufficient stock available.
7. System saves the details.

Extensions:

2a.customer's ID does not exist:
2a1. System notifies Operator, and terminates the use case.
2b.product's ID does not exist:
2b1. System notifies Operator, and terminates the use case.
4a.ID is not new:
4a1. System notifies Operator, and terminates the use case.
6a.Insufficient stock:
6a1. System notifies Operator, and terminates the use case.

14.2. Original Source Code of Parts in Warehouses System.

13.2.1. Hibernate

- **Customer.java**

```
public class Customer {
    private String custonerId;
    private String name;

    public Customer() {
    }

    public Customer(String custonerId, String name) {
        this.custonerId = custonerId;
        this.name = name;
    }


    /**
     * @return the custonerId
     */
    public String getCustonerId() {
        return custonerId;
    }

    /**
     * @param custonerId the custonerId to set
     */
    public void setCustonerId(String custonerId) {
        this.custonerId = custonerId;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }
}
```

- **Customer.hbm.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
  <class name="Customer" table="customer">
    <id column="CustomerID" name="custonerId">
      <generator class="assigned"/>
    </id>
    <property column="Name" name="name"/>
  </class>
</hibernate-mapping>
```

- **Product.java**

```
public class Product {
    private String productNr;
    private String description;
    private double price;
    private int qty;

    public Product() {
    }

    public Product(String productNr, String description, double price, int qty)
{
        this.productNr = productNr;
        this.description = description;
        this.price = price;
        this.qty = qty;
    }

    /**
     * @return the productNr
     */
    public String getProductNr() {
        return productNr;
    }

    /**
     * @param productNr the productNr to set
     */
    public void setProductNr(String productNr) {
        this.productNr = productNr;
    }

    /**
     * @return the description
     */
    public String getDescription() {
        return description;
    }

    /**
     * @param description the description to set
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * @return the price
     */
    public double getPrice() {
        return price;
    }

    /**
     * @param price the price to set
     */
    public void setPrice(double price) {
        this.price = price;
    }

    /**
     * @return the qty
     */
    public int getQty() {
        return qty;
    }

    /**
     * @param qty the qty to set
     */
    public void setQty(int qty) {
        this.qty = qty;
    }
    public void removeStock(int nrProductItems){
        qty -= nrProductItems;
    }
}
```

- **Product.hbm.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
  <class name="Product" table="product">
    <id column="ProductNr" name="productNr">
      <generator class="assigned"/>
    </id>
    <property column="Description" name="description"/>
    <property column="Price" name="price"/>
    <property column="QryStock" name="qty"/>
  </class>
</hibernate-mapping>
```

- **Order.java**

```
import java.util.Date;

public class Order {

    private String orderNr;
    private Date orderDate;
    private Customer customer;
    private Product product;
    private int qty;

    public Order() {
    }


public Order(String orderNr, Date orderDate, Customer customer, Product
product, int qty) {
        this.orderNr = orderNr;
        this.orderDate = orderDate;
        this.customer = customer;
        this.product = product;
        this.qty = qty;
    }

    /**
     * @return the orderNr
     */
    public String getOrderNr() {
        return orderNr;
    }

    /**
     * @param orderNr the orderNr to set
     */
    public void setOrderNr(String orderNr) {
        this.orderNr = orderNr;
    }

    /**
     * @return the orderDate
     */
    public Date getOrderDate() {
        return orderDate;
    }

    /**
     * @param orderDate the orderDate to set
     */
    public void setOrderDate(Date orderDate) {
        this.orderDate = orderDate;
    }

    /**
     * @return the customer
     */
    public Customer getCustomer() {
        return customer;
    }

    /**
     * @param customer the customer to set
     */
    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    /**
     * @return the product
     */
    public Product getProduct() {
        return product;
    }

    /**
     * @param product the product to set
     */
    public void setProduct(Product product) {
        this.product = product;
    }

    public int getQty() {
        return qty;
    }

    public void setQty(int qty) {
        this.qty = qty;
    }
}
```

- **Order.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Order" table="orders">
    <id column="OrderNr" name="orderNr">
      <generator class="assigned"/>
    </id>
    <property name="orderDate" column="OrderDate" type="timestamp"/>
    <property name="qty" column="Quantity"/>
    <many-to-one name="customer" class="Customer"
column="CustomerID"/>
    <many-to-one name="product" class="Product" column="ProductNr"/>
   </class>
</hibernate-mapping>
```

- **hibernate.cfg.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost/standingordersystem</proper
ty>
        <property name="connection.username">root</property>
        <property name="connection.password">asomalia</property>

       <!-- session properties -->
        <property
name="hibernate.current_session_context_class">org.hibernate.context.Thr
eadLocalSessionContext</property>

       <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

```
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">False</property>

        <!-- Mapping files -->
        <mapping resource="Product.hbm.xml"/>
        <mapping resource="Customer.hbm.xml"/>
        <mapping resource="Order.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

- **SOSystem.java**

```
import java.util.Date;
import java.util.List;
import org.hibernate.*;
public class SOSystem{

    public Customer searchCustomer(String id){

                  Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Customer customer = (Customer)session.get(Customer.class,id);
        return customer;
    }

   public void addCustomer(String id, String name)
   {
      //check precondition
     Customer customer = searchCustomer(id);
     boolean pre1 = (customer == null);

     if(!pre1){
         System.out.println("ERROR: The id already exists!");
         return;
      }
      Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      customer = new Customer(id,name);
      session.save(customer);
      session.getTransaction().commit();
}

   public Product searchProduct(String id)
    {
                  Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Product product = (Product)session.get(Product.class,id);
        return product;
    }


   public void addProduct(String id, String description, double price, int qty)
   {
      //check precondition
     Product product = searchProduct(id);
     boolean pre1 = (product == null);

     if(!pre1){
         System.out.println("ERROR: The id already exists!");
         return;
      }
      Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      product = new Product(id,description,price,qty);
      session.save(product);
      session.getTransaction().commit();
}

    public Order searchOrder(String id)
    {
                  Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Order order = (Order)session.get(Order.class,id);
        return order;
    }
```

```
   public void addOrder(String orderNr, String customerid, String productNr,
int qty) {

     Customer customer = searchCustomer(customerid);
     boolean pre1 = (customer == null);
    if(pre1){
        System.out.println("ERROR: The customer number does not exist!");
        return;
     }

    Product product = searchProduct(productNr);
    boolean pre2 = (product == null);
    if(pre2)    {
         System.out.println("ERROR: The product number number does not
exist!");
         return;
    }

    Order order = searchOrder(orderNr);
    boolean pre3 = (order == null);
    if(!pre3){
         System.out.println("ERROR: The order number already exists!");
         return;
       }

    boolean pre4 = (qty > product.getQty());
    if(pre4){
         System.out.println("ERROR: Insufficient stock!");
         return;
       }
      Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
      session.beginTransaction();
      order = new Order(orderNr,new Date(),customer,product,qty);
      product.removeStock(qty);
      session.update(product);
      session.save(order);
      session.getTransaction().commit();
   }
}
```

13.2.2. JDBC

- **SOSystem.java**

```
/**
 *
 * @author ALharthy
 */

import java.sql.*;

public class SOSystem {
    public static String url = "jdbc:mysql://localhost/standingordersystem";
    public static String dbdriver = "com.mysql.jdbc.Driver";
    public static String username = "root";
    public static String password = "root";
    static Connection con;
    public static String sqlCustomer = "SELECT * FROM customer WHERE
CustomerID = ?";
    public static String sqlProduct = "SELECT * FROM product WHERE
ProductNr = ?";
    public static String sqlOrder = "SELECT * FROM orders WHERE
OrderNr = ?";

    public SOSystem() {
        con = getConnection();
    }

     public void ConnClose(){
       try
                  {
                                  con.close();
                  }
                  catch(SQLException ex)
                  {
                                          System.err.println("SQLException: " +
ex.getMessage());
                  }
     }
```

```
   public static Connection getConnection()
              {

                try
                {

              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                }
                catch(java.lang.ClassNotFoundException e)
                {
                              System.err.print("ClassNotFoundException: ");
                              System.err.println(e.getMessage());
                }

                try
                {
                              con = DriverManager.getConnection(url,username,
password);
                }
                 catch(SQLException ex)
                {
                                          System.err.println("SQLException: " +
ex.getMessage());
                }
                            return con;
              }

   public boolean search(String id , String sql){

     boolean isExist = true;
     try
     {
          PreparedStatement pstmt  = con.prepareStatement(sql);
          pstmt.setString(1,id);
          ResultSet rs = pstmt.executeQuery();
          isExist = rs.next();
     }
     catch( SQLException e ) {
          System.out.println(e.getMessage());
          e.printStackTrace();
       }

      return isExist;
   }

  public void addCustomer(String id, String name)
  {
     //check precondition
     boolean pre1 = search(id,sqlCustomer);

     if(pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
     try {
      String sqlStmt="INSERT into customer VALUES(?,?)";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1, id);
      pStmt.setString(2, name);
      pStmt.executeUpdate();
     } catch (SQLException e) {
            System.out.println(e.getMessage());
           e.printStackTrace();
   }
}

  public void addProduct(String id, String desc , double price , int qty){

    boolean pre1 = search(id,sqlProduct);
    if(pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
     try {
      String sqlStmt="INSERT into product VALUES(?,?,?,?)";
      PreparedStatement pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1,id);
      pStmt.setString(2,desc);
      pStmt.setDouble(3,price);
      pStmt.setInt(4,qty);
      pStmt.executeUpdate();
     } catch (SQLException e) {
            System.out.println(e.getMessage());
           e.printStackTrace();
   }
}

   public void addQrder(String ordid, String customerID,String productNr,
int qty){

     boolean pre1= search(customerID,sqlCustomer);

      if(!pre1){
        System.out.println("ERROR: The customer number does not exist!");
        return;
      }

      boolean pre2 = search(productNr,sqlProduct);

      if(!pre2) {
        System.out.println("ERROR: The product id does not exist");
        return;
      }

     boolean pre3 = search(ordid,sqlOrder);
      if(pre3)  {
       System.out.println("ERROR: The order id already exists");
       return;
      }


     try {
     PreparedStatement pStmt =
con.prepareStatement(sqlProduct,ResultSet.TYPE_SCROLL_SENSITIVE,

ResultSet.CONCUR_UPDATABLE);
     pStmt.setString(1, productNr);
     ResultSet rs = pStmt.executeQuery();
     rs.first() ;
     int qtyStock = rs.getInt("QryStock");
     boolean pre4 = ( qty > qtyStock);
      if(pre4){
       System.out.println("ERROR: Insufficient stock!");
       return;
      }

      rs.updateInt("QryStock",qtyStock - qty);
      rs.updateRow();
      String sqlStmt="INSERT into orders VALUES(?,?,?,?,?)";
      pStmt = con.prepareStatement(sqlStmt);
      pStmt.setString(1, ordid);
      pStmt.setDate(2,new Date(System.currentTimeMillis()));
      pStmt.setString(3, customerID);
      pStmt.setString(4, productNr);
      pStmt.setInt(5, qty);
      pStmt.executeUpdate();
     } catch (SQLException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
    }
   }
}
```

14.3. The Source Code of the Standing Order System After Programmatic Adjustment .

14.3.1. Hibernate

- **Customer.java**

```
import java.util.HashSet;
import java.util.Set;

public class Customer {
   private String custonerId;
   private String name;
   private Set<Order> orders;
```

```
    public Customer() {
    }

    public Customer(String custonerId, String name) {
        this.custonerId = custonerId;
        this.name = name;
        this.orders = new HashSet<Order>();

    }

    /**
     * @return the custonerId
     */
    public String getCustonerId() {
        return custonerId;
    }

    /**
     * @param custonerId the custonerId to set
     */
    public void setCustonerId(String custonerId) {
        this.custonerId = custonerId;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    public Set<Order> getOrders() {
        return orders;
    }

    public void setOrders(Set<Order> orders) {
        this.orders = orders;
    }
}
```

- **Customer.hbm.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
  <class name="Customer" table="customer">
    <id column="CustomerID" name="custonerId">
      <generator class="assigned"/>
    </id>
    <property column="Name" name="name"/>
     <set name="orders" table="orders"  inverse="true">
       <key column="CustomerID"/>
       <one-to-many class="Order"/>
     </set>
  </class>
</hibernate-mapping>
```

- **Product.java**

```
import java.util.HashSet;
import java.util.Set;

public class Product {

    private String productNr;
    private String description;
    private double price;
    private int qty;
    private Set<Order> orders;
    public Product() {
    }

    public Product(String productNr, String description, double price, int qty)
{
        this.productNr = productNr;
        this.description = description;
        this.price = price;
        this.qty = qty;
        this.orders = new HashSet<Order>();
    }

    /**
     * @return the productNr
     */
    public String getProductNr() {
        return productNr;
    }

    /**
     * @param productNr the productNr to set
     */
    public void setProductNr(String productNr) {
        this.productNr = productNr;
    }

    /**
     * @return the description
     */
    public String getDescription() {
        return description;
    }

    /**
     * @param description the description to set
     */
    public void setDescription(String description) {
        this.description = description;
    }

    /**
     * @return the price
     */
    public double getPrice() {
        return price;
    }

    /**
     * @param price the price to set
     */
    public void setPrice(double price) {
        this.price = price;
    }

    /**
     * @return the qty
     */
    public int getQty() {
        return qty;
    }

    /**
     * @param qty the qty to set
     */
    public void setQty(int qty) {
        this.qty = qty;
    }

    public void removeStock(int nrProductItems){
        qty -= nrProductItems;
    }

    public Set<Order> getOrders() {
        return orders;
    }

    public void setOrders(Set<Order> orders) {
        this.orders = orders;
    }

}
```

- **Product.hbm.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
  <class name="Product" table="product">
    <id column="ProductNr" name="productNr">
      <generator class="assigned"/>
    </id>
    <property column="Description" name="description"/>
    <property column="Price" name="price"/>
    <property column="QryStock" name="qty"/>

    <set name="orders" table="orders"  inverse="true">
       <key column="ProductNr"/>
       <one-to-many class="Order"/>
    </set>

  </class>
</hibernate-mapping>
```

- **Order.java**

```
public class Order {

    private String orderNr;
    private Date orderDate;
    private Customer customer;
    private Product product;
    private int qty;

    public Order() {
    }

    public Order(String orderNr, Date orderDate, Customer customer,
Product product, int qty) {
        this.orderNr = orderNr;
        this.orderDate = orderDate;
        this.customer = customer;
        this.product = product;
        this.qty = qty;
    }

    /**
     * @return the orderNr
     */
    public String getOrderNr() {
        return orderNr;
    }

    /**
     * @param orderNr the orderNr to set
     */
    public void setOrderNr(String orderNr) {
        this.orderNr = orderNr;
    }

    /**
     * @return the orderDate
     */
    public Date getOrderDate() {
        return orderDate;
    }

    /**
     * @param orderDate the orderDate to set
     */
    public void setOrderDate(Date orderDate) {
        this.orderDate = orderDate;
    }

    /**
     * @return the customer
     */
    public Customer getCustomer() {
        return customer;
    }

    /**
     * @param customer the customer to set
     */
    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    /**
     * @return the product
     */
    public Product getProduct() {
        return product;
    }

    /**
     * @param product the product to set
     */
    public void setProduct(Product product) {
        this.product = product;
    }

    public int getQty() {
        return qty;
    }

    public void setQty(int qty) {
        this.qty = qty;
    }
}
```

- **Order.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Order" table="orders">
    <id column="OrderNr" name="orderNr">
      <generator class="assigned"/>
    </id>
    <property name="orderDate" column="OrderDate" type="timestamp"/>
    <property name="qty" column="Quantity"/>
    <many-to-one name="customer" class="Customer"
column="CustomerID"/>
    <many-to-one name="product" class="Product" column="ProductNr"/>
   </class>
</hibernate-mapping>
```

- **hibernate.cfg.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="connection.url">jdbc:mysql://localhost/standingordersystem</proper
ty>
        <property name="connection.username">root</property>
        <property name="connection.password">asomalia</property>

       <!-- session properties -->
        <property
name="hibernate.current_session_context_class">org.hibernate.context.Thr
eadLocalSessionContext</property>

       <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">False</property>
    <!-- Mapping files -->
    <mapping resource="Product.hbm.xml"/>
    <mapping resource="Customer.hbm.xml"/>
    <mapping resource="Order.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

- **SOSystem.java**

```
import java.util.Date;
import java.util.List;
import org.hibernate.*;
public class SOSystem
{

   public Customer searchCustomer(String id){

       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Customer customer = (Customer)session.get(Customer.class,id);
       return customer;
   }

  public void addCustomer(String id, String name)
  {
     //check precondition
    Customer customer = searchCustomer(id);
    boolean pre1 = (customer == null);

    if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     customer = new Customer(id,name);
     session.save(customer);
     session.getTransaction().commit();
}

  public Product searchProduct(String id)
   {
              Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Product product = (Product)session.get(Product.class,id);
       return product;
   }

  public void addProduct(String id, String description, double price, int qty)
  {
     //check precondition
    Product product = searchProduct(id);
    boolean pre1 = (product == null);

    if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     product = new Product(id,description,price,qty);
     session.save(product);
     session.getTransaction().commit();
}
   public Order searchOrder(String id)
   {
              Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Order order = (Order)session.get(Order.class,id);
        return order;
   }

  public void addOrder(String orderNr, String customerid, String productNr,
int qty) {

    Customer customer = searchCustomer(customerid);
    boolean pre1 = (customer == null);
   if(pre1){
       System.out.println("ERROR: The customer number does not exist!");
       return;
    }

   Product product = searchProduct(productNr);
   boolean pre2 = (product == null);
   if(pre2)    {
         System.out.println("ERROR: The product number number does not
exist!");
         return;
   }

   Order order = searchOrder(orderNr);
   boolean pre3 = (order == null);
   if(!pre3){
         System.out.println("ERROR: The order number already exists!");
         return;
      }

   boolean pre4 = (qty > product.getQty());
   if(pre4){
         System.out.println("ERROR: Insufficient stock!");
         return;
      }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     order = new Order(orderNr,new Date(),customer,product,qty);
     product.removeStock(qty);
     session.update(product);
     session.save(order);
     session.getTransaction().commit();
  }
}
```

14.3.2. JDBC

- **SOSystem.java**

The same as original one.

Appendix D

15.1 USE Case Description

1

Use Case:Add Club

Trigger/Goal: To enter the details of a club.

العنوان:	Design and Implementation of Business Logic Layer Object-Oriented Design vs. Relational Design
المؤلف الرئيسي:	Al Harthy, Ali Ahmed
مؤلفين آخرين:	Nguyen, Kinh(Super)
التاريخ الميلادي:	2009
موقع:	بندورا ، فيكتوريا
الصفحات:	1 - 82
رقم MD:	615558
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	La Trobe University
الكلية:	Faculty of Science, Technology and Engineering
الدولة:	أستراليا
قواعد المعلومات:	Dissertations
مواضيع:	الحاسبات الالكترونية، هندسة البرمجيات، تكنولوجيا المعلومات
رابط:	https://search.mandumah.com/Record/615558

```
    <!-- Mapping files -->
    <mapping resource="Product.hbm.xml"/>
    <mapping resource="Customer.hbm.xml"/>
    <mapping resource="Order.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

- **SOSystem.java**

```
import java.util.Date;
import java.util.List;
import org.hibernate.*;
public class SOSystem
{

   public Customer searchCustomer(String id){

       Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Customer customer = (Customer)session.get(Customer.class,id);
       return customer;
   }

  public void addCustomer(String id, String name)
  {
     //check precondition
    Customer customer = searchCustomer(id);
    boolean pre1 = (customer == null);

    if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     customer = new Customer(id,name);
     session.save(customer);
     session.getTransaction().commit();
}

  public Product searchProduct(String id)
   {
              Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
       Product product = (Product)session.get(Product.class,id);
       return product;
   }

  public void addProduct(String id, String description, double price, int qty)
  {
     //check precondition
    Product product = searchProduct(id);
    boolean pre1 = (product == null);

    if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     product = new Product(id,description,price,qty);
     session.save(product);
     session.getTransaction().commit();
}
   public Order searchOrder(String id)
   {
             Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Order order = (Order)session.get(Order.class,id);
        return order;
   }

  public void addOrder(String orderNr, String customerid, String productNr,
int qty) {

    Customer customer = searchCustomer(customerid);
    boolean pre1 = (customer == null);
   if(pre1){
       System.out.println("ERROR: The customer number does not exist!");
       return;
    }

   Product product = searchProduct(productNr);
   boolean pre2 = (product == null);
   if(pre2)    {
         System.out.println("ERROR: The product number number does not
exist!");
         return;
   }

   Order order = searchOrder(orderNr);
   boolean pre3 = (order == null);
   if(!pre3){
         System.out.println("ERROR: The order number already exists!");
         return;
      }

   boolean pre4 = (qty > product.getQty());
   if(pre4){
         System.out.println("ERROR: Insufficient stock!");
         return;
      }
     Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
     session.beginTransaction();
     order = new Order(orderNr,new Date(),customer,product,qty);
     product.removeStock(qty);
     session.update(product);
     session.save(order);
     session.getTransaction().commit();
   }
}
```

14.3.2. JDBC

- **SOSystem.java**

The same as original one.

Appendix D

15.1 USE Case Description

1

Use Case:Add Club

Trigger/Goal: To enter the details of a club.

Actors: clerk
Main Flow:
1. clerk enters club's ID.
2.System validates that ID is new.
3. clerk enters club's name.
4. System validates that club's ID is new.
5. clerk enters contant name , contant phone, fax number and address.
6.System saves the details of the club.
Extensions:

2a.ID is not new:
2a1. System notifies Operator, and terminates the use case .
4a.club's name is not new:
2a1. System notifies Operator, and terminates the use case .

2

Use Case: Add Gymnast
Trigger/Goal: To enter the details of a gymnast (a member of a club).
Actors: clerk
Main Flow:
1. clerk enters gymnast's ID.
2.System validates that ID is new.
3. clerk enters club's ID.
4. System validates that club'sID is exist.
5. clerk enters gymnast's name, DOB,genderphone number and registeration date.
6.System saves the details of the gymnast.
Extensions:
2a.gymnast ID is not new:
2a1. System notifies Operator, and terminates the use case.
4a.club's ID is not exist:
4a1. System notifies Operator, and terminates the use case.

3

Use Case: Add Divisons.
Trigger/Goal: To enter the details of a division .
Actors: Operator
Main Flow:
clerk enters Divisions' code(WJ/WS/MJ/MS).
1. System validates that code is new.
3. clerk enters Divisions' name.
4. System validates that name is new.
5. clerk enters Divisions' gender,lower age limit and upper age limit.
6. System saves the details of the division.

Extensions:

2a. code is not new:
2a1. System notifies Operator, and terminates the use case .
4a. name is not new:
2a1. System notifies Operator, and terminates the use case .

4

Use Case: Add Event's type
Trigger/Goal: To enter the details of event's type.
Actors: clerk
Main Flow:
1. clerk enters event's type code.
2. System validates that code is new.
3. clerk enters event's type name.
4. System validates that name is new.
5. clerk indicates whether the event type is for men, for women or for both.
2a1. System notifies Operator, and terminates the use case
4a. division code is not exist:

6.System saves the details .
Extensions:
2a.event's type code is not new:
2a1. System notifies Operator, and terminates the use case.
4a.event's type name is not new:
2a1. System notifies Operator, and terminates the use case.

5

Use Case: Add judge
Trigger/Goal: To enter the details of judge.
Actors: clerk
Main Flow:
1. clerk enters judge's ID .
2. System validates judge's ID is new.
3. clerk enters judge's name and contact phone number.
4. clerk enters event's type code.
5. System validates event's type code is exist.
--Do Step 1to 5for each event type the judge is qualified for.
6.System saves the details of the judge.
Extensions:

2a.judge's ID is not new:
2a1. System notifies Operator, and terminates the use case.
5a.event's type code is not exist:
2a1. System notifies Operator, and terminates the use case.

6

Use Case: Add meet
Trigger/Goal: To enter the details of a meet to organize the competitions.
Actors: clerk
Main Flow:
1. clerk enters meet's ID.
2.System validates that ID is new.
3. clerk enters meet name,and the place and date it is held.
4.System saves the details.
5. System creates the competitions of the meet(for the divisions).

Extensions:
3a.meet ID is not new:
3a1. System notifies Operator, and terminates the use case .
5a.event type code does not exist:
2a1. System notifies Operator, and terminates the use case .

7

Use Case: Add an event.
Trigger/Goal: to assign judge to the event.
Actors: clerk
Main Flow:
1. clerk enters meet ID.
2.System validates meet ID is exist.
3.clerk enters division code.
4. System validates division code is exist.
(hence ,the competition identified by (meet id and division code) must exist).
5. clerk enters event type code.
6.System validates event type code is exist.
7.System validates event(which can be identified by meet id,division code and event type code)does not exist.
8. System saves the details of the event.

Extensions:
2a. meet ID is not exist:

2a1. System notifies Operator, and terminates the use case

6a. event type code is not exist:
2a1. System notifies Operator, and terminates the use case .

8

Use Case: Assign judge to event.
Trigger/Goal: to assign judge to the event.
Actors: clerk
Main Flow:
1. clerk enters meet ID.
2.System validates meet ID is exist.
3.clerk enters division code.
4. System validates division code is exist.
5. clerk enters event type code.
6.System validates event type code is exist.
7.System validates event is exist.
8.clerk enters judge's ID
9. System validates judge's ID is exist.
10. System validates judge has not been assigned to the event.
11. System saves the details of the assignment .
Extensions:
2a. meet ID is not exist:
2a1. System notifies Operator, and terminates the use case
4a.division code is not exist:
2a1. System notifies Operator, and terminates the use case
6a. event type code is not exist:
2a1. System notifies Operator, and terminates the use case .
9a. judge's ID is not exist:
2a1. System notifies Operator, and terminates the use case

9

Use Case: Add team
Trigger/Goal: To register a team for a competition in a meet including members of the team.
Actors: clerk
Main Flow:

1.clerk enters meet's ID.
2.System validates that meet ID is exist.
3. clerk enters Divisions' code(WJ/WS/MJ/MS).
4.System validates that division code is exist.(the competition should exist).
5. clerk enters club's ID.
6. System validates that club's ID is exist.
7. System validates that the team has not been registered yet.
8. clerk enters the ID of member of the team.
9. System validates that the member belongs to the club and the division.
--Do Step 8 to 9 for each member of the team.
10.System saves the details of the team including the member of the team.
Extensions:
2a.meet's ID is not exist:
2a1. System notifies Operator, and terminates the use case.
4a.division code is not exist:
2a1. System notifies Operator, and terminates the use case.
6a.club's ID is not exist:
2a1. System notifies Operator, and terminates the use case.

10

Use Case: Enter score of gymnast for event.
Trigger/Goal: . To register a team for a competition in a meet including members of the team.

Actors: clerk.
Main Flow:
1. clerk enters meet ID.
2.System validates meet ID is exist.
3.clerk enters Divisions' code.
4.System validates that division code is exist.
5. clerk enters event's type code.
6. System validates that event type code is exist.
7.clerk enters gymnast's ID.
8.System validates that gymnast ID is exist
9. System validates that the gymnast is a member of a team appropriate for the event.
10. clerk enters score.
11. System validates the gymnast has not had a score for the event.
12.System saves the score of gymnast for event.
Extensions:

2a.meet ID is not exist:
2a1. System notifies Operator, and terminates the use case
4a.division code is not exist:
2a1. System notifies Operator, and terminates the use case.
6a.event type code is not exist:
2a1. System notifies Operator, and terminates the use case
8a. gymnast ID is not new:
2a1. System notifies Operator, and terminates the use case.

```
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Club other = (Club) obj;
        if ((this.club_id == null) ? (other.club_id != null) :
!this.club_id.equals(other.club_id)) {
            return false;
        }
        return true;
    }
}
```

- **Club.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="Club" table="clubs">
    <id name="id">
      <generator class="assigned"/>
    </id>
    <property name="name"/>
    <property name="phone"/>
  </class>
</hibernate-mapping>
```

- **CompetitionPk.java**

```
import java.io.Serializable;

public class CompetitionPk implements Serializable  {

 private Division division;
 private Meet meet;

    public CompetitionPk(Division division, Meet meet) {
        this.division = division;
        this.meet = meet;
    }

    public Division getDivision() {
        return division;
    }

    public void setDivision(Division division) {
        this.division = division;
    }
    /**
     * @return the meet
     */
    public Meet getMeet() {
        return meet;
    }

    /**
     * @param meet the meet to set
     */
    public void setMeet(Meet meet) {
        this.meet = meet;
    }

    @Override
    public boolean equals(Object obj) {

    if (!(obj instanceof CompetitionPk)) return false;
    CompetitionPk other = (CompetitionPk) obj;
```

15.2. Original Source Code of the Eastern Suburb Gymnastics.

15.2.1. Hibernate

- **Club.java**

```
public class Club {

    private String club_id ;
    private String name;
    private String phone;

    public Club() {
    }

    public Club(String club_id, String name, String phone) {
        this.club_id = club_id;
        this.name = name;
        this.phone = phone;
    }

    /**
     * @return the id
     */
    public String getId() {
        return club_id;
    }

    /**
     * @param id the id to set
     */
    public void setId(String id) {
        this.club_id = id;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the phone
     */
    public String getPhone() {
        return phone;
    }

    /**
     * @param phone the phone to set
     */
    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
```

```
    <key-many-to-one name="division" class="Division"
column="division_code"/>
    <key-many-to-one name="meet" class="Meet"
column="meet_id"/>
   </composite-id>
    </class>
</hibernate-mapping>
```

- **Gymnast.java**

```
public class Gymnast {
    private String id;
    private String name;
    private int age;
    private char gender;
    private Club club;

    public Gymnast() {
    }

    public Gymnast(String id, String name, int age, char gender,
Club club) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.club = club;
    }



    /**
     * @return the id
     */
    public String getId() {
        return id;
    }

    /**
     * @param id the id to set
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the age
     */
    public int getAge() {
        return age;
    }

    /**
     * @param age the age to set
     */
    public void setAge(int age) {
        this.age = age;
    }

    /**
     * @return the gender
```

```
    return this.division.equals(other.division) &&
           this.meet.equals(other.meet);
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 89 * hash + (this.division != null ?
this.division.hashCode() : 0);
        hash = 89 * hash + (this.meet != null ?
this.meet.hashCode() : 0);
        return hash;
    }
}
```

- **Competition.java**

```
public class Competition {
private CompetitionPk competitionPk;

    public Competition() {
    }

    public Competition(Division division, Meet meet) {
        this.competitionPk = new CompetitionPk (division,meet);

    }

    /**
     * @return the competitionPk
     */
    public CompetitionPk getCompetitionPk() {
        return competitionPk;
    }

    /**
     * @param competitionPk the competitionPk to set
     */
    public void setCompetitionPk(CompetitionPk competitionPk)
{
        this.competitionPk = competitionPk;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Competition other = (Competition) obj;
        if (this.competitionPk != other.competitionPk &&
(this.competitionPk == null ||
!this.competitionPk.equals(other.competitionPk))) {
            return false;
        }
        return true;
    }
}
```

- **Competition.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Competition" table="competitions">
   <composite-id name="competitionPk"
class="CompetitionPk">
```

```
  public Division(String code, String name, char gender, int
lowerAgeLimit, int upperAgeLimit) {
      this.code = code;
      this.name = name;
      this.gender = gender;
      this.lowerAgeLimit = lowerAgeLimit;
      this.upperAgeLimit = upperAgeLimit;
  }


  /**
   * @return the code
   */
  public String getCode() {
      return code;
  }

  /**
   * @param code the code to set
   */
  public void setCode(String code) {
      this.code = code;
  }

  /**
   * @return the name
   */
  public String getName() {
      return name;
  }

  /**
   * @param name the name to set
   */
  public void setName(String name) {
      this.name = name;
  }

  /**
   * @return the gender
   */
  public char getGender() {
      return gender;
  }

  /**
   * @param gender the gender to set
   */
  public void setGender(char gender) {
      this.gender = gender;
  }

  /**
   * @return the lowerAgeLimit
   */
  public int getLowerAgeLimit() {
      return lowerAgeLimit;
  }

  /**
   * @param lowerAgeLimit the lowerAgeLimit to set
   */
  public void setLowerAgeLimit(int lowerAgeLimit) {
      this.lowerAgeLimit = lowerAgeLimit;
  }

  /**
   * @return the upperAgeLimit
   */
  public int getUpperAgeLimit() {
      return upperAgeLimit;
  }

  /**
   * @param upperAgeLimit the upperAgeLimit to set
   */
```

```
   */
  public char getGender() {
      return gender;
  }

  /**
   * @param gender the gender to set
   */
  public void setGender(char gender) {
      this.gender = gender;
  }

  /**
   * @return the club
   */
  public Club getClub() {
      return club;
  }

  /**
   * @param club the club to set
   */
  public void setClub(Club club) {
      this.club = club;
  }

  @Override
  public boolean equals(Object obj) {
      if (obj == null) {
          return false;
      }
      if (getClass() != obj.getClass()) {
          return false;
      }
      final Gymnast other = (Gymnast) obj;
      if ((this.id == null) ? (other.id != null) :
!this.id.equals(other.id)) {
          return false;
      }
      return true;
  }
}
```

- **Gymnast.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
  PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="Gymnast" table="gymnast">
    <id name="id" >
      <generator class="assigned"/>
    </id>
    <property name="name" type="string"/>
    <property name="age" type="int"/>
     <property name="gender"/>
    <many-to-one name="club" column="club_id"/>
  </class>
</hibernate-mapping>
```

- **Division.java**

```
public class Division {
  private String code;
  private String name;
  private char gender;
  private int lowerAgeLimit;
  private int upperAgeLimit;

  public Division() {
  }
```

```
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the date
     */
    public String getDate() {
        return date;
    }

    /**
     * @param date the date to set
     */
    public void setDate(String date) {
        this.date = date;
    }

    /**
     * @return the place
     */
    public String getPlace() {
        return place;
    }

    /**
     * @param place the place to set
     */
    public void setPlace(String place) {
        this.place = place;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Meet other = (Meet) obj;
        if ((this.id == null) ? (other.id != null) :
!this.id.equals(other.id)) {
            return false;
        }
        return true;
    }
}
```

- **Meet.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Meet" table="meet">
      <id name="id" >
         <generator class="assigned"/>
      </id>
      <property name="name"/>
     <property name="date" column="dataOfMetting"/>
      <property name="place"/>
   </class>
</hibernate-mapping>
```

- **Judge.java**

```
import java.util.HashSet;
```

```
    public void setUpperAgeLimit(int upperAgeLimit) {
        this.upperAgeLimit = upperAgeLimit;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Division other = (Division) obj;
        if ((this.code == null) ? (other.code != null) :
!this.code.equals(other.code)) {
            return false;
        }
        return true;
    }
}
```

- **Division.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Division" table="division">
      <id name="code" >
         <generator class="assigned"/>
      </id>
      <property name="name"/>
      <property name="gender"/>
      <property name="lowerAgeLimit"/>
      <property name="upperAgeLimit"/>
   </class>
</hibernate-mapping>
```

- **Meet.java**

```
public class Meet {
    private String id;
    private String name;
    private String date;
    private String place;
    public Meet() {
    }
    public Meet(String id, String name, String date, String place)
{
        this.id = id;
        this.name = name;
        this.date = date;
        this.place = place;
    }

    /**
     * @return the id
     */
    public String getId() {
        return id;
    }

    /**
     * @param id the id to set
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * @return the name
     */
    public String getName() {
```

```
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Judge" table="judge">
      <id name="id" >
         <generator class="assigned"/>
      </id>
      <property name="name"/>
      <property name="phone"/>
      <set name="eventTypes" table="qualified" cascade="all">
            <key column="judge_id" />
            <many-to-many column="eventtype_code"
class="EventType" />
       </set>
   </class>
</hibernate-mapping>
```

- **EventType.java**

```
public class EventType {

    private String id;
    private String name;
    private boolean menEvent;
    private boolean womenEvent;

    public EventType() {
    }

    public EventType(String id, String name, boolean menEvent,
boolean womenEvent) {
        this.id = id;
        this.name = name;
        this.menEvent = menEvent;
        this.womenEvent = womenEvent;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the menEvent
     */
    public boolean isMenEvent() {
        return menEvent;
    }

    /**
     * @param menEvent the menEvent to set
     */
    public void setMenEvent(boolean menEvent) {
        this.menEvent = menEvent;
    }

    /**
     * @return the womenEvent
```

```
import java.util.Set;

public class Judge {
    private String id;
    private String name;
    private String phone;
    private Set<EventType> eventTypes = new
HashSet<EventType>(0);

    public Judge() {
    }

    public Judge(String id, String name, String phone ,
Set<EventType> eventTypes) {
        this.id = id;
        this.name = name;
        this.phone = phone;
        this.eventTypes = eventTypes;
    }


    /**
     * @return the id
     */
    public String getId() {
        return id;
    }

    /**
     * @param id the id to set
     */
    public void setId(String id) {
        this.id = id;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the phone
     */
    public String getPhone() {
        return phone;
    }

    /**
     * @param phone the phone to set
     */
    public void setPhone(String phone) {
        this.phone = phone;
    }

    public Set<EventType> getEventTypes() {
        return eventTypes;
    }

    public void setEventTypes(Set<EventType> eventTypes) {
        this.eventTypes = eventTypes;
    }
}
```

- **Judge.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
```

```
        return division;
    }

    /**
     * @param division the division to set
     */
    public void setDivision(Division division) {
        this.division = division;
    }

    /**
     * @return the meet
     */
    public Meet getMeet() {
        return meet;
    }

    /**
     * @param meet the meet to set
     */
    public void setMeet(Meet meet) {
        this.meet = meet;
    }

    /**
     * @return the eventType
     */
    public EventType getEventType() {
        return eventType;
    }

    /**
     * @param eventType the eventType to set
     */
    public void setEventType(EventType eventType) {
        this.eventType = eventType;
    }

    @Override
    public boolean equals(Object obj) {

    if (!(obj instanceof EventPk))
        return false;
    EventPk other = (EventPk) obj;
    return this.eventType.equals(other.eventType) &&
           this.division.equals(other.division)   &&
           this.meet.equals(other.meet);
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 61 * hash + (this.division != null ?
this.division.hashCode() : 0);
        hash = 61 * hash + (this.meet != null ?
this.meet.hashCode() : 0);
        hash = 61 * hash + (this.eventType != null ?
this.eventType.hashCode() : 0);
        return hash;
    }
}
```

- **Event.java**

```
import java.util.HashSet;
import java.util.Set;

public class Event {
private EventPk eventPk;
private Set<Judge> judges = new HashSet<Judge>(0);


    public Event() {
    }
```

```
     */
    public boolean isWomenEvent() {
        return womenEvent;
    }

    /**
     * @param womenEvent the womenEvent to set
     */
    public void setWomenEvent(boolean womenEvent) {
        this.womenEvent = womenEvent;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final EventType other = (EventType) obj;
        if ((this.id == null) ? (other.id != null) :
!this.id.equals(other.id)) {
            return false;
        }
        return true;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 61 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}
```

- **EventType.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="EventType" table="eventtype">
      <id name="id" >
         <generator class="assigned"/>
      </id>
      <property name="name"/>
      <property name="menEvent"/>
      <property name="womenEvent"/>
      </class>
</hibernate-mapping>
```

- **EventPk.java**

```
import java.io.Serializable;

public class EventPk  implements Serializable{

 private Division division;
 private Meet meet;
 private EventType eventType;

    public EventPk(Division division, Meet meet, EventType
eventType) {
        this.division = division;
        this.meet = meet;
        this.eventType = eventType;
    }


    /**
     * @return the division
     */
    public Division getDivision() {
```

```
public class Score{

 private EventType eventType;
 private Gymnast gymnast;
 private Division division;
 private Meet meet;
 private Club club;

    public Score(EventType eventType, Gymnast gymnast,
Division division, Meet meet, Club club) {
        this.eventType = eventType;
        this.gymnast = gymnast;
        this.division = division;
        this.meet = meet;
        this.club = club;
    }

   public EventType getEventType() {
        return eventType;
    }

    /**
     * @param eventType the eventType to set
     */
    public void setEventType(EventType eventType) {
        this.eventType = eventType;
    }

    /**
     * @return the gymnast
     */
    public Gymnast getGymnast() {
        return gymnast;
    }

    /**
     * @param gymnast the gymnast to set
     */
    public void setGymnast(Gymnast gymnast) {
        this.gymnast = gymnast;
    }

    /**
     * @return the division
     */
    public Division getDivision() {
        return division;
    }

    /**
     * @param division the division to set
     */
    public void setDivision(Division division) {
        this.division = division;
    }

    /**
     * @return the meet
     */
    public Meet getMeet() {
        return meet;
    }

    /**
     * @param meet the meet to set
     */
    public void setMeet(Meet meet) {
        this.meet = meet;
    }

    /**
     * @return the club
     */
    public Club getClub() {
        return club;
```

```
    public Event(EventPk eventPk) {
        this.eventPk = eventPk;
     }

    public Set<Judge> getJudges() {
        return judges;
    }

    public void setJudges(Set<Judge> JudgeS) {
        this.judges = JudgeS;
    }


    /**
     * @return the eventPk
     */
    public EventPk getEventPk() {
        return eventPk;
    }

    /**
     * @param eventPk the eventPk to set
     */
    public void setEventPk(EventPk eventPk) {
        this.eventPk = eventPk;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Event other = (Event) obj;
        if (this.eventPk != other.eventPk && (this.eventPk == null
|| !this.eventPk.equals(other.eventPk))) {
            return false;
        }
        return true;
    }

}
```

- **Event.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Event" table="event">
   <composite-id name="eventPk" class="EventPk">
    <key-many-to-one name="eventType" class="EventType"
column="eventType_code"/>
    <key-many-to-one name="division" class="Division"
column="division_code"/>
    <key-many-to-one name="meet" class="Meet"
column="meet_id"/>
   </composite-id>
      <set name="judges" table="panel" cascade="all">
            <key>
            <column name="eventType_code"/>
            <column name="meet_id"/>
            <column name="division_code"/>
            </key>
            <many-to-many column="judge_id"  class="Judge" />
      </set>
    </class>
</hibernate-mapping>
```

- **Score.java**

```
    public TeamPk(Division division, Meet meet, Club club) {
        this.division = division;
        this.meet = meet;
        this.club = club;
    }

    public Division getDivision() {
        return division;
    }

    public void setDivision(Division division) {
        this.division = division;
    }



    /**
     * @return the meet
     */
    public Meet getMeet() {
        return meet;
    }

    /**
     * @param meet the meet to set
     */
    public void setMeet(Meet meet) {
        this.meet = meet;
    }

    /**
     * @return the club
     */
    public Club getClub() {
        return club;
    }

    /**
     * @param club the club to set
     */
    public void setClub(Club club) {
        this.club = club;
    }

    @Override
    public boolean equals(Object obj) {

    if (!(obj instanceof TeamPk))
        return false;
    TeamPk other = (TeamPk) obj;
    return this.club.equals(other.club) &&
           this.division.equals(other.division) &&
           this.meet.equals(other.meet);
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 97 * hash + (this.division != null ?
this.division.hashCode() : 0);
        hash = 97 * hash + (this.meet != null ?
this.meet.hashCode() : 0);
        hash = 97 * hash + (this.club != null ? this.club.hashCode()
: 0);
        return hash;
    }
}
```

- **Team.java**

```
public class Team {
private TeamPk teamPk;
private Competition competition;
```

```
    }

    /**
     * @param club the club to set
     */
    public void setClub(Club club) {
        this.club = club;
    }

    @Override
    public boolean equals(Object obj) {
    if (!(obj instanceof Score)) return false;
    Score other = (Score) obj;
    return this.eventType.equals(other.eventType) &&
           this.gymnast.equals(other.gymnast) &&
           this.division.equals(other.division) &&
           this.meet.equals(other.meet) &&
           this.club.equals(other.club) ;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 73 * hash + (this.eventType != null ?
this.eventType.hashCode() : 0);
        hash = 73 * hash + (this.gymnast != null ?
this.gymnast.hashCode() : 0);
        hash = 73 * hash + (this.division != null ?
this.division.hashCode() : 0);
        hash = 73 * hash + (this.meet != null ?
this.meet.hashCode() : 0);
        hash = 73 * hash + (this.club != null ? this.club.hashCode()
: 0);
        return hash;
    }
 }
```

- **Score.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
   PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Score" table="score">
   <composite-id name="scorePk" class="ScorePk">
    <key-many-to-one name="gymnast" class="Gymnast"
column="gymnast_id"/>
    <key-many-to-one name="club" class="Club"
column="club_id"/>
    <key-many-to-one name="eventType" class="EventType"
column="eventType_code"/>
    <key-many-to-one name="division" class="Division"
column="division_code"/>
    <key-many-to-one name="meet" class="Meet"
column="meet_id"/>
    </composite-id>
    <property name="score"/>
    </class>
</hibernate-mapping>
```

- **TeamPk.java**

```
import java.io.Serializable;

public class TeamPk implements Serializable {

  private Division division;
  private Meet meet;
  private Club club;
```

```
      sessionFactory = new
Configuration().configure().buildSessionFactory();
    } catch (Throwable ex) {
      // Make sure you log the exception, as it might be swallowed
      System.err.println("Initial SessionFactory creation failed." +
ex);
      throw new ExceptionInInitializerError(ex);
    }
  }

  public static SessionFactory getSessionFactory() {
    return sessionFactory;
  }
}
```

- **hibernate.cfg.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-
3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
        <property
name="connection.url">jdbc:mysql://localhost/gymnastics</pro
perty>
        <property name="connection.username">root</property>
        <property
name="connection.password">asomalia</property>


       <!-- session properties -->
        <property
name="hibernate.current_session_context_class">org.hibernate.
context.ThreadLocalSessionContext</property>

       <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property
>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">False</property>

        <!-- Mapping files -->

        <mapping resource="Club.hbm.xml"/>
        <mapping resource="Division.hbm.xml"/>
        <mapping resource="Gymnast.hbm.xml"/>
        <mapping resource="Judge.hbm.xml"/>
        <mapping resource="Meet.hbm.xml"/>
        <mapping resource="Competition.hbm.xml"/>
        <mapping resource="Team.hbm.xml"/>
        <mapping resource="TeamMember.hbm.xml"/>
        <mapping resource="Score.hbm.xml"/>
        <mapping resource="EventType.hbm.xml"/>
        <mapping resource="Event.hbm.xml"/>


    </session-factory>
</hibernate-configuration>
```

- **GScoringSystem**

```
import java.io.Serializable;
import java.util.HashSet;
```

```
    public Team() {
    }

    public Team(Division division, Meet meet, Club club) {
        this.teamPk = new TeamPk(division, meet, club);
        this.competition = new Competition(division, meet);
    }

    public Competition getCompetition() {
        return competition;
    }

    /**
     * @return the teamPk
     */
    public TeamPk getTeamPk() {
        return teamPk;
    }

    /**
     * @param teamPk the teamPk to set
     */
    public void setTeamPk(TeamPk teamPk) {
        this.teamPk = teamPk;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Team other = (Team) obj;
        if (this.teamPk != other.teamPk && (this.teamPk == null ||
!this.teamPk.equals(other.teamPk))) {
            return false;
        }
        return true;
    }
}
```

- **Team.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="Team" table="team">
    <composite-id name="teamPk" class="TeamPk">
     <key-many-to-one name="club" class="Club"
column="club_id"/>
     <key-many-to-one name="division" class="Division"
column="division_code"/>
     <key-many-to-one name="meet" class="Meet"
column="meet_id"/>
     </composite-id>

  <property name="teamPk" type="TeamPk" insert="false"
update="false" access="field" />

  </class>
</hibernate-mapping>
```

- **HibernateUtil.java**

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
  private static final SessionFactory sessionFactory;
  static {
    try {
      // Create the SessionFactory from hibernate.cfg.xml
```

```
     //check precondition
    Division obj = (Division) search(Division.class,id);

    boolean pre1 = (obj == null);

    if(!pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

     obj = new
Division(id,name,gender,lowerAgeLimit,upperAgeLimit);
     saveOrUpdate(obj);
}

  public void addMeet(String id, String name, String date, String
place)
   {
     //check precondition
    Meet obj = (Meet) search(Meet.class,id);

    boolean pre1 = (obj == null);

    if(!pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

     obj = new Meet(id, name, date,place);
     saveOrUpdate(obj);
}


public void addEventType(String id, String name, boolean
menEvent, boolean womenEvent)
   {
     //check precondition
    EventType obj = (EventType) search(EventType.class,id);

    boolean pre1 = (obj == null);

    if(!pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

     obj = new EventType(id,name,menEvent,womenEvent);
     saveOrUpdate(obj);
}

  public void addJudge(String id, String name, String phone ,
Set<String> listOfeventtypes)
   {
     //check precondition
    Judge obj = (Judge) search(Judge.class,id);

    boolean pre1 = (obj == null);

    if(!pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

   boolean pre2 = true;
   for(String code : listOfeventtypes)
    {
              if(search(EventType.class,code)== null)
     {
                          pre2 = false;
     }
    }
   if(!pre2){
       System.out.println("ERROR: some of eventTypes numbers
do not exist!");
       return;
    }
```

```
import java.util.Set;
import org.hibernate.*;
public class GScoringSystem
{
  private Session session;

   private void connect() {
             session =
HibernateUtil.getSessionFactory().getCurrentSession();
       session.beginTransaction();
   }

   public Object search(Class clazz, Serializable id){

       connect();
       Object obj = null;
       obj = session.get(clazz,id);
       return obj;
   }

  private void saveOrUpdate(Object obj) {

      connect();
     try{
        session.saveOrUpdate(obj);
        session.getTransaction().commit();
     }
     catch(HibernateException e) {
          System.err.print(e);
   }
  }

  public void addClub(String id,String name, String phone)
   {
     //check precondition
    Club obj = (Club) search(Club.class,id);

    boolean pre1 = (obj == null);

    if(!pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

     obj = new Club(id,name,phone);
     saveOrUpdate(obj);
}

  public void addGymnast(String id,String clubId,String name ,
int age, char gender)
   {
     //check precondition
    Gymnast obj = (Gymnast) search(Gymnast.class,id);

    boolean pre1 = (obj == null);

    if(!pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

   Club objClub = (Club) search(Club.class,clubId);

   boolean pre2 = (objClub == null);
   if(pre2){
       System.out.println("ERROR: The club number does not
exist!");
       return;
     }
     obj = new Gymnast(id, name ,age, gender ,objClub);
     saveOrUpdate(obj);
}

 public void addDivision(String id, String name, char gender, int
lowerAgeLimit, int upperAgeLimit)
   {
        System.out.println("ERROR: The EventType number does
not exist!");
        return;
     }

   EventPk eventPk = new EventPk(obj2,obj,obj3);
   Event obj4 = (Event) search(Event.class,eventPk);
   boolean pre4 = (obj4 == null);
   if(!pre4){
       System.out.println("ERROR: The id already exists!");
       return;
     }

     obj4 = new Event(eventPk);
     saveOrUpdate(obj4);
}


public void addAssignJudgeToEvent(String id,String code,String
eventTypeCode,String judgeId)
  {
     //check precondition

   Judge obj = (Judge) search(Judge.class,id);
   boolean pre1 = (obj == null);
   if(pre1){
       System.out.println("ERROR: The Meet number does not
exist!");
       return;
     }

   Division obj2 = (Division) search(Division.class,code);
   Meet obj3 = (Meet) search(Meet.class,id);
   EventType obj4 = (EventType)
search(EventType.class,eventTypeCode);

   EventPk eventPk = new EventPk(obj2,obj3,obj4);
   Event obj5 = (Event) search(Event.class,eventPk);
   boolean pre2 = (obj5 == null);
   if(pre2){
       System.out.println("The Event number does not exist!");
       return;
     }

    obj5.getJudges().add(obj);
     saveOrUpdate(obj5);
}

 public void registerTeam(String id,String code , String club_id)
   {
    //check precondition

   Club obj = (Club) search(Club.class,club_id);
   boolean pre1 = (obj == null);
   if(pre1){
       System.out.println("ERROR: The Club number does not
exist!");
       return;
     }

   Division obj1 = (Division) search(Division.class,code);
   Meet obj2 = (Meet) search(Meet.class,id);
   CompetitionPk competitionPk = new
CompetitionPk(obj1,obj2);
   Competition obj3 = (Competition)
search(Competition.class,competitionPk);
   boolean pre2 = (obj3 == null);
    if(pre2){
       System.out.println("ERROR: The Competition number
does not exist!");
       return;
     }

    Team obj5 = new Team(obj1,obj2,obj);
    saveOrUpdate(obj5);
```

```
   Set<EventType> eventTypes = new HashSet<EventType>(0);

   for(String code : listOfeventtypes)
   {

eventTypes.add((EventType)search(EventType.class,code));
   }

    obj = new Judge( id , name , phone , eventTypes);
    saveOrUpdate(obj);
}

  public void addCompetition (String id,String code)
  {
     //check precondition

   Meet obj = (Meet) search(Meet.class,id);
   boolean pre1 = (obj == null);
   if(pre1){
       System.out.println("ERROR: The Meet number does not
exist!");
       return;
     }

   Division obj2 = (Division) search(Division.class,code);
   boolean pre2= (obj2 == null);
   if(pre2){
       System.out.println("ERROR: The Division number does
not exist!");
       return;
     }

   CompetitionPk competitionPk = new
CompetitionPk(obj2,obj);

   Competition obj3 = (Competition)
search(Competition.class,competitionPk);

    boolean pre3 = (obj3 == null);

    if(!pre3){
       System.out.println("ERROR: The id already exists!");
       return;
     }

     obj3 = new Competition(obj2,obj);
     saveOrUpdate(obj3);
}

  public void addEvent (String id,String code,String
eventTypeCode)
  {
     //check precondition

   Meet obj = (Meet) search(Meet.class,id);
   boolean pre1 = (obj == null);
   if(pre1){
       System.out.println("ERROR: The Meet number does not
exist!");
       return;
     }

   Division obj2 = (Division) search(Division.class,code);
   boolean pre2= (obj2 == null);
   if(pre2){
       System.out.println("ERROR: The Division number does
not exist!");
       return;
     }

   EventType obj3 = (EventType)
search(EventType.class,eventTypeCode);
   boolean pre3= (obj3 == null);
   if(pre3){
```

```
            {

              try
              {

            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
              }
              catch(java.lang.ClassNotFoundException e)
              {

System.err.print("ClassNotFoundException: ");
                          System.err.println(e.getMessage());
              }

              try
              {
                          con =
DriverManager.getConnection(url,username, password);
              }
               catch(SQLException ex)
              {

            System.err.println("SQLException: " +
ex.getMessage());
              }
                        return con;
            }

   public ResultSet search(String sql){

     ResultSet rs=null;
     try
     {
         Statement stmt  = con.createStatement();
          rs = stmt.executeQuery(sql);
      //   isExist = rs.next();
     }
     catch( SQLException e ) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }
      return rs;
   }

    public static void insert(String tableName, Object[] values){

      String s = "insert into " + tableName + " values (";

      for (int i = 0; i < values.length; i++) {
         s += "?, ";
      }

      s = s.substring(0, s.length() - 2) + ")";
     try{
      PreparedStatement insert = con.prepareStatement(s);

      for (int i = 0; i < values.length; i++) {
         insert.setObject(i + 1, values[i]);
      }
      insert.executeUpdate();
      } catch (SQLException e) {
           System.out.println(e.getMessage());
           e.printStackTrace();
   }
   }

  private Object[] getValues(Object...objects){
     return objects;
  }
  public void addClub(String id,String name, String phone)
throws SQLException
  {
    //check precondition
    String sql = "SELECT * FROM clubs WHERE id =" + id;
    boolean pre1 = (search(sql)).next();
```

```
   }

 public void enterScoreForGymnast(String id,String code,String
eventTypeCode,
                           String gymnastId , int score)
 {
    //check precondition
   Division  division= (Division) search(Division.class,code);
   Meet meet = (Meet) search(Meet.class,id);
   EventType eventType = (EventType)
search(EventType.class,eventTypeCode);

   EventPk eventPk = new EventPk(division,meet,eventType);
   Event obj5 = (Event) search(Event.class,eventPk);
   boolean pre1 = (obj5 == null);
   if(pre1){
       System.out.println("The Event number does not exist!");
       return;
     }

   Gymnast gymnast = (Gymnast)
search(Gymnast.class,gymnastId);

   Club club = gymnast.getClub();

   TeamPk  teamPk = new TeamPk(division,meet,club);

   Team team = (Team) search(Team.class,teamPk);

   boolean pre2 = (team == null);
   if(pre2){
       System.out.println("The club is not registered for the
competition !");
       return;
     }

  Score s =  new
Score(eventType,gymnast,division,meet,club,score);
     saveOrUpdate(s);
}
```

15.2.2. JDBC

- **GScoringSystem**

```
import java.sql.*;

public class GScoringSystem {

   public static String url = "jdbc:mysql://localhost/gymnastics";
   public static String dbdriver = "com.mysql.jdbc.Driver";
   public static String username = "root";
   public static String password = "root";
   static Connection con;

   public GScoringSystem() {
      con = getConnection();
   }

    public void ConnClose(){
      try
              {
                        con.close();
              }
               catch(SQLException ex)
              {

            System.err.println("SQLException: " +
ex.getMessage());
              }
    }

   public static Connection getConnection()
```

```
    if(pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }
    Object[] values =
getValues(id,name,menEvent,womenEvent);
    insert("eventType" , values);
}

  public void addJudge(String id, String name, String phone ,
String []listOfeventtypes) throws SQLException
   {
     //check precondition
    String sql = "SELECT * FROM judge WHERE id =" + id;
    boolean pre1 = (search(sql)).next();

    if(pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

    boolean pre2 = true;
   for(String code : listOfeventtypes)
    {
     sql = "SELECT * FROM eventType WHERE id =" + code;
              if(!(search(sql)).next())
     {
                              pre2 = false;
       break;
     }
    }
   if(!pre2){
      System.out.println("ERROR: some of eventTypes numbers
do not exist!");
       return;
    }

   Object[] values = getValues(id,name,phone);
   insert("judge" , values);

   for(String code : listOfeventtypes)
    {
               values = getValues(code,id);
      insert("qualified" , values);
    }
}

   public void addCompetition (String code,String id) throws
SQLException
   {
     //check precondition
    String sql = "SELECT * FROM meet WHERE id =" + id;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
       System.out.println("ERROR:The Meet number does not
exist!");
       return;
     }

    sql = "SELECT * FROM division division code =" + id;
    boolean pre2 = (search(sql)).next();

    if(pre2){
       System.out.println("ERROR: The division number does
not exist!");
       return;
     }

    Object[] values = getValues(code,id);
    insert("competitions" , values);
}

   public void addEvent (String eventTypeCode,String code
,String id) throws SQLException
   {
     //check precondition
    if(pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

    Object[] values = getValues(id,name,phone);
    insert("clubs" , values);
}

   public void addGymnast(String id,String name , int age, String
gender,String clubId) throws SQLException
   {
     //check precondition

    String sql = "SELECT * FROM clubs WHERE id =" +
clubId;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
       System.out.println("ERROR: The club number does not
exist!");
       return;
     }

    sql = "SELECT * FROM gymnast WHERE id =" + id;
    boolean pre2 = (search(sql)).next();
    if(pre2){
       System.out.println("ERROR: The id already exists!");
       return;
     }

    Object[] values = getValues(id,name,age,gender,clubId);
    insert("gymnast" , values);
}

 public void addDivision(String id, String name, String gender,
int lowerAgeLimit, int upperAgeLimit) throws SQLException
   {
    //check precondition
    String sql = "SELECT * FROM division division code =" +
id;
    boolean pre1 = (search(sql)).next();

    if(pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }

    Object[] values = getValues(id,name,
gender,lowerAgeLimit,upperAgeLimit);
    insert("division" , values);
}

   public void addMeet(String id, String name, String date, String
place) throws SQLException
   {
     //check precondition
    String sql = "SELECT * FROM meet WHERE id =" + id;
    boolean pre1 = (search(sql)).next();

    if(pre1){      System.out.println("ERROR: The id already
exists!");
       return;
     }

    Object[] values = getValues(id,name,date,place);
    insert("meet" , values);
}


public void addEventType(String id, String name, boolean
menEvent, boolean womenEvent) throws SQLException
   {
     //check precondition
    String sql = "SELECT * FROM eventType WHERE id =" +
id;
    boolean pre1 = (search(sql)).next();
```

```
}


 public void enterScoreForGymnast(String id,String code,String
eventTypeCode,
                        String gymnastId , int score) throws
SQLException
  {
      //check precondition
   String sql = "SELECT * FROM event WHERE
eventType_code =" + eventTypeCode +
              " And meet_id="+id+ " And division_code="+code;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
       System.out.println("ERROR:The event number does not
exist!");
       return;
     }

    sql = "SELECT * FROM gymnast WHERE id =" + id;
    String clubid = (search(sql)).getString("club_id");
    sql = "SELECT * FROM team WHERE club_id =" + clubid +
        " And meet_id="+id+ " And division_code="+code;

    boolean pre2 = (search(sql)).next();
    if(pre2){
       System.out.println("The club is not registered for the
competition !");
       return;
     }
    Object[] values =
getValues(clubid,eventTypeCode,id,code,gymnastId,score);
    insert("score" , values);
}

}
```

15.3. The Source Code of the Standing Order System After Programmatic Adjustment .

15.3.1. Hibernate

some classes are not effected , so there is no necessary to re-print

- **TeamMemberPK.java**

```
import java.io.Serializable;

public class TeamMemberPK implements Serializable{
  private Gymnast gymnast;
  private Division division;
  private Meet meet;
  private Club club;

   public TeamMemberPK(Gymnast gymnast, Division division,
Meet meet, Club club){
       this.gymnast = gymnast;
       this.division = division;
       this.meet = meet;
       this.club = club;
   }

   /**
    * @return the gymnast
    */
   public Gymnast getGymnast() {
```

```
    String sql = "SELECT * FROM meet WHERE id =" + id;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
       System.out.println("ERROR:The Meet number does not
exist!");
       return;
     }

    sql = "SELECT * FROM division WHERE division code ="
+ code;
    boolean pre2 =(search(sql)).next();
    if(pre2){
       System.out.println("ERROR: The division number does
not exist!");
       return;
     }

    sql = "SELECT * FROM eventType WHERE id =" +
eventTypeCode;
    boolean pre3 = (search(sql)).next();
    if(pre3){
       System.out.println("ERROR: The eventType number does
not exist!");
       return;
     }
    Object[] values = getValues(eventTypeCode,code,id);
    insert("event" , values);
}


public void addAssignJudgeToEvent(String
eventTypeCode,String id,String code,String judgeId) throws
SQLException
  {
     //check precondition
    String sql = "SELECT * FROM event WHERE
eventType_code =" + eventTypeCode +
              " And meet_id="+id+ " And division_code="+code;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
       System.out.println("ERROR:The event number does not
exist!");
       return;
     }
    Object[] values =
getValues(eventTypeCode,id,code,judgeId);
    insert("panel" , values);
}

 public void registerTeam(String clubId,String id,String code)
throws SQLException
   {
     //check precondition

    String sql = "SELECT * FROM clubs WHERE id =" +
clubId;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
       System.out.println("ERROR: The club number does not
exist!");
       return;
     }

    sql = "SELECT * FROM competitions WHERE meet_id ="
+id+
              " AND division_code="+code;
    boolean pre2 = (search(sql)).next();
    if(!pre2){
       System.out.println("ERROR: The Competition number
does not exist!");
       return;
     }

    Object[] values = getValues(clubId,id,code);
    insert("team" , values);
```

- **TeamMember.hbm.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-mapping
  PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="TeamMember" table="teammember">
  <composite-id name="teamMemberPK"
class="TeamMemberPK">
   <key-many-to-one name="gymnast" class="Gymnast"
column="gymnast_id"/>
   <key-many-to-one name="club" class="Club"
column="club_id"/>
   <key-many-to-one name="division" class="Division"
column="division_code"/>
   <key-many-to-one name="meet" class="Meet"
column="meet_id"/>
  </composite-id>
   </class>
</hibernate-mapping>
```

- **hibernate.cfg.xml**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-
3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <!-- Database connection settings -->
      <property
name="connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
      <property
name="connection.url">jdbc:mysql://localhost/gymnastics</pro
perty>
      <property name="connection.username">root</property>
      <property name="connection.password">root </property>


     <!-- session properties -->
      <property
name="hibernate.current_session_context_class">org.hibernate.
context.ThreadLocalSessionContext</property>

     <!-- JDBC connection pool (use the built-in) -->
      <property name="connection.pool_size">1</property>

      <!-- SQL dialect -->
      <property
name="dialect">org.hibernate.dialect.MySQLDialect</property
>

      <!-- Echo all executed SQL to stdout -->
      <property name="show_sql">False</property>

      <!-- Mapping files -->

      <mapping resource="Club.hbm.xml"/>
      <mapping resource="Division.hbm.xml"/>
      <mapping resource="Gymnast.hbm.xml"/>
      <mapping resource="Judge.hbm.xml"/>
      <mapping resource="Meet.hbm.xml"/>
      <mapping resource="Competition.hbm.xml"/>
      <mapping resource="Team.hbm.xml"/>

      <mapping resource="TeamMember.hbm.xml"/>

      <mapping resource="Score.hbm.xml"/>
      <mapping resource="EventType.hbm.xml"/>
      <mapping resource="Event.hbm.xml"/>
```

```
      return gymnast;
   }

   /**
    * @param gymnast the gymnast to set
    */
   public void setGymnast(Gymnast gymnast) {
      this.gymnast = gymnast;
   }

   /**
    * @return the division
    */
   public Division getDivision() {
      return division;
   }

   /**
    * @param division the division to set
    */
   public void setDivision(Division division) {
      this.division = division;
   }

   /**
    * @return the meet
    */
   public Meet getMeet() {
      return meet;
   }

   /**
    * @param meet the meet to set
    */
   public void setMeet(Meet meet) {
      this.meet = meet;
   }

   /**
    * @return the club
    */
   public Club getClub() {
      return club;
   }

   /**
    * @param club the club to set
    */
   public void setClub(Club club) {
      this.club = club;
   }
}
```

- **TeamMember.java**

```
public class TeamMember {
private TeamMemberPK teamMemberPK;

   public TeamMember() {
   }

   public TeamMember(Gymnast gymnast, Division division,
Meet meet, Club club) {
      this.teamMemberPK = new
TeamMemberPK(gymnast,division,meet,club);
   }

   public TeamMemberPK getTeamMemberPK() {
      return teamMemberPK;
   }

   public void setTeamMemberPK(TeamMemberPK
teamMemberPK) {
      this.teamMemberPK = teamMemberPK;
   }
}
```

```
        System.out.println("ERROR: The club number does not
exist!");
        return;
      }
      obj = new Gymnast(id, name ,age, gender ,objClub);
      saveOrUpdate(obj);
}

 public void addDivision(String id, String name, char gender, int
lowerAgeLimit, int upperAgeLimit)
   {
      //check precondition
     Division obj = (Division) search(Division.class,id);

     boolean pre1 = (obj == null);

     if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
      }

      obj = new
Division(id,name,gender,lowerAgeLimit,upperAgeLimit);
      saveOrUpdate(obj);
}

   public void addMeet(String id, String name, String date, String
place)
   {
      //check precondition
     Meet obj = (Meet) search(Meet.class,id);

     boolean pre1 = (obj == null);

     if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
      }

      obj = new Meet(id, name, date,place);
      saveOrUpdate(obj);
}


public void addEventType(String id, String name, boolean
menEvent, boolean womenEvent)
   {
      //check precondition
     EventType obj = (EventType) search(EventType.class,id);

     boolean pre1 = (obj == null);

     if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
      }

      obj = new EventType(id,name,menEvent,womenEvent);
      saveOrUpdate(obj);
}

  public void addJudge(String id, String name, String phone ,
Set<String> listOfeventtypes)
   {
      //check precondition
     Judge obj = (Judge) search(Judge.class,id);

     boolean pre1 = (obj == null);

     if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
      }

    boolean pre2 = true;
    for(String code : listOfeventtypes)
```

```
    </session-factory>
</hibernate-configuration>
```

- **GScoringSystem**

```
import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;
import org.hibernate.*;
public class GScoringSystem
{
  private Session session;

    private void connect() {
               session =
HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
    }

    public Object search(Class clazz, Serializable id){

        connect();
        Object obj = null;
        obj = session.get(clazz,id);
        return obj;
    }

   private void saveOrUpdate(Object obj) {

       connect();
      try{
         session.saveOrUpdate(obj);
         session.getTransaction().commit();
      }
      catch(HibernateException e) {
           System.err.print(e);
    }
   }

   public void addClub(String id,String name, String phone)
   {
      //check precondition
     Club obj = (Club) search(Club.class,id);

     boolean pre1 = (obj == null);

     if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
      }

      obj = new Club(id,name,phone);
      saveOrUpdate(obj);
}

   public void addGymnast(String id,String clubId,String name ,
int age, char gender)
   {
      //check precondition
     Gymnast obj = (Gymnast) search(Gymnast.class,id);

     boolean pre1 = (obj == null);

     if(!pre1){
        System.out.println("ERROR: The id already exists!");
        return;
      }

    Club objClub = (Club) search(Club.class,clubId);

    boolean pre2 = (objClub == null);
    if(pre2){
```

```
    if(pre2){
        System.out.println("ERROR: The Division number does
not exist!");
        return;
      }

    EventType obj3 = (EventType)
search(EventType.class,eventTypeCode);
    boolean pre3= (obj3 == null);
    if(pre3){
        System.out.println("ERROR: The EventType number does
not exist!");
        return;
      }

    EventPk eventPk = new EventPk(obj2,obj,obj3);
    Event obj4 = (Event) search(Event.class,eventPk);
    boolean pre4 = (obj4 == null);
    if(!pre4){
        System.out.println("ERROR: The id already exists!");
        return;
      }

      obj4 = new Event(eventPk);
      saveOrUpdate(obj4);
}


public void addAssignJudgeToEvent(String id,String code,String
eventTypeCode,String judgeId)
  {
      //check precondition

    Judge obj = (Judge) search(Judge.class,id);
    boolean pre1 = (obj == null);
    if(pre1){
        System.out.println("ERROR: The Meet number does not
exist!");
        return;
      }

    Division obj2 = (Division) search(Division.class,code);
    Meet obj3 = (Meet) search(Meet.class,id);
    EventType obj4 = (EventType)
search(EventType.class,eventTypeCode);

    EventPk eventPk = new EventPk(obj2,obj3,obj4);
    Event obj5 = (Event) search(Event.class,eventPk);
    boolean pre2 = (obj5 == null);
    if(pre2){
        System.out.println("The Event number does not exist!");
        return;
      }

     obj5.getJudges().add(obj);
      saveOrUpdate(obj5);
}

 public void registerTeam(String id,String code , String club_id)
   {
      //check precondition

    Club obj = (Club) search(Club.class,club_id);
    boolean pre1 = (obj == null);
    if(pre1){
        System.out.println("ERROR: The Club number does not
exist!");
        return;
      }

    Division obj1 = (Division) search(Division.class,code);
    Meet obj2 = (Meet) search(Meet.class,id);
    CompetitionPk competitionPk = new
CompetitionPk(obj1,obj2);
    Competition obj3 = (Competition)
search(Competition.class,competitionPk);
```

```
    {
               if(search(EventType.class,code)== null)
     {
                         pre2 = false;
     }
    }
    if(!pre2){
        System.out.println("ERROR: some of eventTypes numbers
do not exist!");
        return;
    }

    Set<EventType> eventTypes = new HashSet<EventType>(0);

    for(String code : listOfeventtypes)
    {

eventTypes.add((EventType)search(EventType.class,code));
    }

     obj = new Judge( id , name , phone , eventTypes);
     saveOrUpdate(obj);
}

   public void addCompetition (String id,String code)
   {
      //check precondition

    Meet obj = (Meet) search(Meet.class,id);
    boolean pre1 = (obj == null);
    if(pre1){
        System.out.println("ERROR: The Meet number does not
exist!");
        return;
      }

    Division obj2 = (Division) search(Division.class,code);
    boolean pre2= (obj2 == null);
    if(pre2){
        System.out.println("ERROR: The Division number does
not exist!");
        return;
      }

    CompetitionPk competitionPk = new
CompetitionPk(obj2,obj);

    Competition obj3 = (Competition)
search(Competition.class,competitionPk);

     boolean pre3 = (obj3 == null);

     if(!pre3){
        System.out.println("ERROR: The id already exists!");
        return;
      }

      obj3 = new Competition(obj2,obj);
      saveOrUpdate(obj3);
}

   public void addEvent (String id,String code,String
eventTypeCode)
   {
      //check precondition

    Meet obj = (Meet) search(Meet.class,id);
    boolean pre1 = (obj == null);
    if(pre1){
        System.out.println("ERROR: The Meet number does not
exist!");
        return;
      }

    Division obj2 = (Division) search(Division.class,code);
    boolean pre2= (obj2 == null);
   Club club = gymnast.getClub();

   TeamPk  teamPk = new TeamPk(division,meet,club);

   Team team = (Team) search(Team.class,teamPk);

   boolean pre2 = (team == null);
   if(pre2){
       System.out.println("The club is not registered for the
competition !");
       return;
    }

   TeamMemberPK teamMemberPK = new
TeamMemberPK(gymnast,division,meet,club);
   TeamMember teamMember = (TeamMember)
search(TeamMember.class,teamMemberPK);
   boolean pre3 = (teamMember == null);
   if(pre3){
       System.out.println("ERROR: The team gymnast is not
registered for the team!");
       return;
    }

  Score s =  new
Score(eventType,gymnast,division,meet,club,score);
     saveOrUpdate(s);
}

}
```

15.3.2. JDBC

- **GScoringSystem**

```
package carparkinjdbc;
/**
 *
 * @author ALharthy
 */

import java.sql.*;

public class GScoringSystem {

   public static String url = "jdbc:mysql://localhost/gymnastics";
   public static String dbdriver = "com.mysql.jdbc.Driver";
   public static String username = "root";
   public static String password = "root";
   static Connection con;

   public    GScoringSystem() {
      con = getConnection();
   }

    public void ConnClose(){
      try
              {
                          con.close();
              }
               catch(SQLException ex)
              {

             System.err.println("SQLException: " +
ex.getMessage());
              }
    }

   public static Connection getConnection()
             {

               try
               {

             Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
   boolean pre2 = (obj3 == null);
    if(pre2){
        System.out.println("ERROR: The Competition number
does not exist!");
        return;
     }

    Team obj5 = new Team(obj1,obj2,obj);
    saveOrUpdate(obj5);

    }

public void               registerMember(String id,String code ,
String gymnastid)
{

   Gymnast gymnast = (Gymnast)
search(Gymnast.class,gymnastid);

    boolean pre1 = (gymnast == null);
    if(!pre1){
        System.out.println("ERROR: The gymnast number does
not exist!");
        return;
     }
   Division division = (Division) search(Division.class,code);
   Meet meet = (Meet) search(Meet.class,id);
   Club club = gymnast.getClub();

   TeamPk obj = new TeamPk(division,meet,club);
   boolean pre2 = (obj == null);
    if(pre2){
        System.out.println("ERROR: The team number does not
exist!");
        return;
     }
   boolean pre3 = true;

    if((gymnast.getGender()!= division.getGender()) ||
          (gymnast.getAge() > division.getUpperAgeLimit()||
           gymnast.getAge() < division.getLowerAgeLimit())){

           pre3 = false;
    }

   if(!pre3){
       System.out.println("ERROR: member does not meet the
gender and age criteria!");
       return;
   }

    TeamMember teamMember = new
TeamMember(gymnast,division,meet,club);
    saveOrUpdate(teamMember);
}
 public void enterScoreForGymnast(String id,String code,String
eventTypeCode,
                         String gymnastId , int score)
  {
     //check precondition
   Division  division= (Division) search(Division.class,code);
   Meet meet = (Meet) search(Meet.class,id);
   EventType eventType = (EventType)
search(EventType.class,eventTypeCode);

   EventPk eventPk = new EventPk(division,meet,eventType);
   Event obj5 = (Event) search(Event.class,eventPk);
   boolean pre1 = (obj5 == null);
   if(pre1){
       System.out.println("The Event number does not exist!");
       return;
    }

   Gymnast gymnast = (Gymnast)
search(Gymnast.class,gymnastId);
```

```
    Object[] values = getValues(id,name,phone);
    insert("clubs" , values);
}

  public void addGymnast(String id,String name , int age, String
gender,String clubId) throws SQLException
   {
     //check precondition

    String sql = "SELECT * FROM clubs WHERE id =" +
clubId;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
        System.out.println("ERROR: The club number does not
exist!");
        return;
     }

    sql = "SELECT * FROM gymnast WHERE id =" + id;
    boolean pre2 = (search(sql)).next();
    if(pre2){
        System.out.println("ERROR: The id already exists!");
        return;
     }

    Object[] values = getValues(id,name,age,gender,clubId);
    insert("gymnast" , values);
}

 public void addDivision(String id, String name, String gender,
int lowerAgeLimit, int upperAgeLimit) throws SQLException
   {
    //check precondition
    String sql = "SELECT * FROM division division code =" +
id;
    boolean pre1 = (search(sql)).next();

    if(pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }

    Object[] values = getValues(id,name,
gender,lowerAgeLimit,upperAgeLimit);
    insert("division" , values);
}

  public void addMeet(String id, String name, String date, String
place) throws SQLException
   {
     //check precondition
    String sql = "SELECT * FROM meet WHERE id =" + id;
    boolean pre1 = (search(sql)).next();

    if(pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }

    Object[] values = getValues(id,name,date,place);
    insert("meet" , values);
}


public void addEventType(String id, String name, boolean
menEvent, boolean womenEvent) throws SQLException
   {
     //check precondition
    String sql = "SELECT * FROM eventType WHERE id =" +
id;
    boolean pre1 = (search(sql)).next();
    if(pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }
```

```
                    }
                    catch(java.lang.ClassNotFoundException e)
                    {

System.err.print("ClassNotFoundException: ");
                                System.err.println(e.getMessage());
                    }

                    try
                    {
                                con =
DriverManager.getConnection(url,username, password);
                    }
                     catch(SQLException ex)
                    {

                 System.err.println("SQLException: " +
ex.getMessage());
                    }
                              return con;
                 }

   public ResultSet search(String sql){

     ResultSet rs=null;
     try
     {
          Statement stmt  = con.createStatement();
           rs = stmt.executeQuery(sql);
       //   isExist = rs.next();
     }
     catch( SQLException e ) {
          System.out.println(e.getMessage());
          e.printStackTrace();
      }
      return rs;
   }


    public static void insert(String tableName, Object[] values){

       String s = "insert into " + tableName + " values (";

       for (int i = 0; i < values.length; i++) {
          s += "?, ";
       }

       s = s.substring(0, s.length() - 2) + ")";
      try{
       PreparedStatement insert = con.prepareStatement(s);

       for (int i = 0; i < values.length; i++) {
          insert.setObject(i + 1, values[i]);
       }
       insert.executeUpdate();
       } catch (SQLException e) {
             System.out.println(e.getMessage());
            e.printStackTrace();
    }
    }

   private Object[] getValues(Object...objects){
      return objects;
   }
   public void addClub(String id,String name, String phone)
throws SQLException
    {
     //check precondition
    String sql = "SELECT * FROM clubs WHERE id =" + id;
    boolean pre1 = (search(sql)).next();

    if(pre1){
       System.out.println("ERROR: The id already exists!");
       return;
     }
```

```
        System.out.println("ERROR:The Meet number does not
exist!");
        return;
     }

    sql = "SELECT * FROM division WHERE division code ="
+ code;
    boolean pre2 =(search(sql)).next();
    if(pre2){
        System.out.println("ERROR: The division number does
not exist!");
        return;
     }

    sql = "SELECT * FROM eventType WHERE id =" +
eventTypeCode;
    boolean pre3 = (search(sql)).next();
    if(pre3){
        System.out.println("ERROR: The eventType number does
not exist!");
        return;
     }
    Object[] values = getValues(eventTypeCode,code,id);
    insert("event" , values);
}


public void addAssignJudgeToEvent(String
eventTypeCode,String id,String code,String judgeId) throws
SQLException
  {
     //check precondition
    String sql = "SELECT * FROM event WHERE
eventType_code =" + eventTypeCode +
                 " And meet_id="+id+ " And division_code="+code;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
        System.out.println("ERROR:The event number does not
exist!");
        return;
     }
    Object[] values =
getValues(eventTypeCode,id,code,judgeId);
    insert("panel" , values);
}

 public void registerTeam(String clubId,String id,String code)
throws SQLException
   {
     //check precondition

    String sql = "SELECT * FROM clubs WHERE id =" +
clubId;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
        System.out.println("ERROR: The club number does not
exist!");
        return;
     }

    sql = "SELECT * FROM competitions WHERE meet_id ="
+id+
                 " AND division_code="+code;
    boolean pre2 = (search(sql)).next();
    if(!pre2){
        System.out.println("ERROR: The Competition number
does not exist!");
        return;
     }

    Object[] values = getValues(clubId,id,code);
    insert("team" , values);
}

 public void registerMember(String id,String code , String
gymnastid) throws SQLException
```

```
    Object[] values =
getValues(id,name,menEvent,womenEvent);
    insert("eventType" , values);
}

  public void addJudge(String id, String name, String phone ,
String []listOfeventtypes) throws SQLException
   {
     //check precondition
    String sql = "SELECT * FROM judge WHERE id =" + id;
    boolean pre1 = (search(sql)).next();

    if(pre1){
        System.out.println("ERROR: The id already exists!");
        return;
     }

    boolean pre2 = true;
   for(String code : listOfeventtypes)
    {
     sql = "SELECT * FROM eventType WHERE id =" + code;
              if(!(search(sql)).next())
     {
                              pre2 = false;
       break;
     }
    }
   if(!pre2){
       System.out.println("ERROR: some of eventTypes numbers
do not exist!");
        return;
   }

   Object[] values = getValues(id,name,phone);
   insert("judge" , values);

   for(String code : listOfeventtypes)
   {
                 values = getValues(code,id);
      insert("qualified" , values);
   }
}

   public void addCompetition (String code,String id) throws
SQLException
   {
     //check precondition
    String sql = "SELECT * FROM meet WHERE id =" + id;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
        System.out.println("ERROR:The Meet number does not
exist!");
        return;
     }

    sql = "SELECT * FROM division division code =" + id;
    boolean pre2 = (search(sql)).next();

    if(pre2){
        System.out.println("ERROR: The division number does
not exist!");
        return;
     }

    Object[] values = getValues(code,id);
    insert("competitions" , values);
}

   public void addEvent (String eventTypeCode,String code
,String id) throws SQLException
   {
     //check precondition

    String sql = "SELECT * FROM meet WHERE id =" + id;
    boolean pre1 = (search(sql)).next();
    if(!pre1){
```

Abstract

Object Oriented Programming has become one of the mainstream programming paradigms in the software engineering society. On the other hand, relational model is the dominant model for commercial data processing applications. There is strong competition between these models for dominance in building modern applications, especially after the emergence and spread of ORM (object-relational mapping) technology. This research project tries to answer the question of whether object-oriented approach is better than traditional approach in terms of flexibility with respect to requirements change.

Table of Contents

List of Tables

List of Figures

La Trobe University

Bundoora, Victoria, Australia

Faculty of Science, Technology & Engineering

Department of Computer Science and Computer Engineering

Design and Implementation of Business Logic Layer

Object-Oriented Design vs. Relational Design

Author:	Ali Ahmed Alharthy
Student No:	15038770
Course:	Master of Information Technology
Submission Date:	October 2009
Supervisors:	Dr. Kinh Nguyen

Abstract

Object Oriented Programming has become one of the mainstream programming paradigms in the software engineering society. On the other hand, relational model is the dominant model for commercial data processing applications. There is strong competition between these models for dominance in building modern applications, especially after the emergence and spread of ORM (object-relational mapping) technology. This research project tries to answer the question of whether object-oriented approach is better than traditional approach in terms of flexibility with respect to requirements change.

Acknowledgement

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I want to thank the Department of Computer Science And Computer Engineering.

I am deeply indebted to my supervisor, Dr. Kinh Nguyen, who helped me in all the time of research for and writing of this thesis.

Especially, I would like to give my special thanks to my wife , Amani , whose patient love enabled me to complete this work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Nowadays, the business logic layers of the modern applications are mostly constructed using either object-oriented model or relational model. The object oriented model is based on software engineering principles such as coupling, inheritance, cohesion, and encapsulation, whereas the relational model is based on predicate logic and set theory principles [1]. The object oriented model chains the building of applications within objects that have both data and behavior.

Relational model support the storage of data in tables and treatment of that data through data manipulation language (DML) inside within the database through stored procedures and externally through SQL (Structured Query Language). The relational model is in use by many database systems [1].

Object oriented technology is also commonly used in database application development. The differences between the two technologies was called the object-relational impedance mismatch [17][11]. In particular, when objects need to be stored in relational database. Object-relational mapping (ORM) is appearing to play an important role to overcome the problem of impedance mismatch. ORM is a new technology that allows applications to access relational data in an object-oriented manner. With the wide-spread use of ORM technology, domain object are built as objects and the application logic manipulates these objects in a pure object oriented manner.

The critical issue that arises is that whether such an object oriented model for business logic layer is a good choice in general. Proponents of object oriented approach tend to assume that an object oriented business model will made the system easier to maintain, easier to extend, and easier to re-use.

The aim of this paper is to critically examine this assumption and to carefully compare the applicability and flexibility of the object oriented system in comparison to relational system. The findings from this project will be significant for practical application where the business logic layer is implemented in an object oriented fashion as the present trend in enterprise computing seems to be heading.

1.2 Outline of the thesis

This thesis is structured as follows. Chapter 2, provides a theoretical background to the thesis which includes basic concept regarding relational model and an object oriented programming. The second half of the chapter is dedicated to introduce the problem of impedance mismatch and the basic concept of ORM. Chapter 3, will introduce and discuss some researches and papers regarding the comparison between the two paradigms. In addition , it will present the research proposal. Chapter 4, will present a simple case study for both technological approaches to make an initial comparison regarding the different of effects involved in implementing them and change them with respect to requirements change. Chapter 5, introduces a complicated case study regarding to change requirements in business policy. Chapter 6, discusses issue of relational representation. The final chapter, introduce a realistic case study to produce much more statistical data to compare the two approaches. Moreover, it will also highlight the issue of relationship representation. The thesis concludes with a very brief discussion of the achieved results and gives some suggestions of future work.

Chapter 2

Background

2.1 The Relational model

As already mentioned, the relational model is based on the predicate logic and set theory, developed by E.F. Code in 1970 [1]. It is built around data normalization, which simplifies maintenance and data storage by minimizing the duplication of information in turn to enforce data consistency. It used the basic concept of a relation or table. Each table has a number of columns with unique names. The columns in the table identify the attributes such as name, age, and so on. A row contains all the data of a single instance of the table such as a person named Michael. Moreover ,In the relational model, each row have to have a unique identification or key based on the data. For example , in figure 1, a social security account number (SSAN) is the key that uniquely identifies each row in the table. Normally, keys are used to join data from two or more tables based on same identification. The relational approach also includes concepts such as foreign keys , which are primary keys in one table that kept in another table to allow for the joining of data. For example of foreign keys is storing father's SSAN and your mother's in the row that represent person. Thus , parents' SSANs are keys for the rows that represent them and they are foreign keys in the row that represents person.

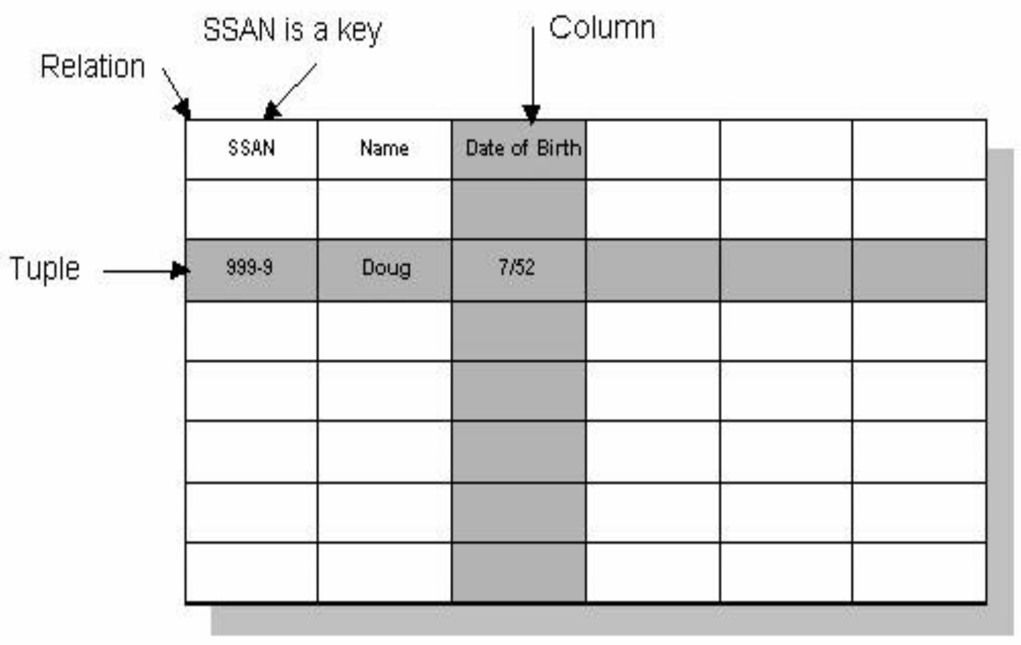


Figure 1: A table is a data structure used to organize information.

Now the relational model is the dominant model for commercial data processing applications such as A Relational database management system (RDBMS) . RDBMSs use Structured Query Language (SQL) as the data definition language (DDL) and the data manipulation language (DML).

Structured Query Language (SQL) was developed at IBM in the early 1970s. This version, firstly called SEQUEL, was considered to manipulate and retrieve data stored in IBM's original relational database product.SQL provides the capability to create and define relational database tables. After these tables are defined, the language permits one to add data to these tables. Once data has been added, one can modify, retrieve, or remove that data. SQL provides the ability of defining what type of authority one might have when accessing the data. Indeed , SQL Statements play an important role when developing database application.

Data definition language (DDL) describes the section of SQL that allows you to create, alter, and drop database objects. These database objects include schemas, , sequences views, tables, indexes ,catalogs and aliases .Currently , it refers to any formal language for describing data or information structures, like XML schemas.

The Data Manipulation Language (DML) is a language which enables users to access and manipulate data. All processing is based on values in fields of records. A SQL data manipulation statement (DML) provides four commands: Insert, Select, Update, and Delete. Database users used these commands during the routine operation of the database. Moreover, it is flexible, powerful and easy to learn. The types of queries vary from complicated multi-table queries to simple single-table queries to involving joins, nesting, set union/differences, and others. Examples of RDBMSs include Microsoft Access, developed by Microsoft and Oracle, developed by Oracle Corporation.

Relational Databases inability to handle application areas like spatial databases , applications involving special types databases , images and other applications that involve complex interrelationships of data. Furthermore , Relational databases do not natively support inheritance (It will be describe later on).[1] for further information about relational model.

2.2The Object oriented programming

Object-Oriented Programming (OOP) is a programming paradigm that enable programmer to divide a program in building blocks known as objects. Programming techniques contain features as coupling, inheritance, cohesion, and encapsulation. However, there are fundamental concepts found in the strong majority of definition of OOP. The definition of OOP is inspired by the paper of Deborah J. Armstrong.
They are the following:

2.2.1Class : In OOP a class is a template definition for a particular object or groups of objects that will always have the same features. The object created using the class is called instances Instance contains real values instead of variables. For instance, the class dog consist of traits shared by all dogs , such as fur and breed color as well as the ability to sit and bark. However , a class can have subclasses that can inherit all or some of the characteristics of the class. As a result this class become the super class for all subclasses.

2.2.2 Objects: are the building blocks of OOP and are usually defined as variables or data structures that encapsulate behavior and data in a complete unit. Objects are items that can be independently created, represent, and manipulated real world things in an abstract way. Figure 2 illustrations is a common visual representation of a software object.

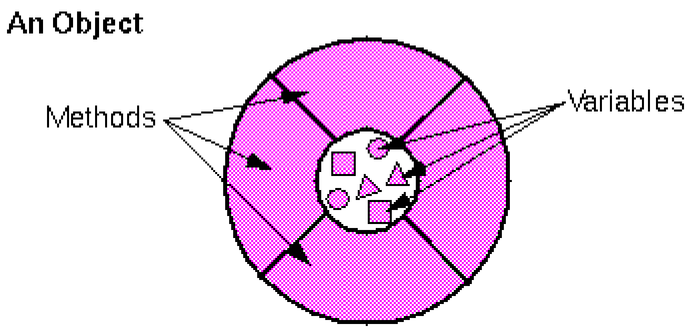


Figure 2 illustrations is a common visual representation of a software object.

2.2.3 Inheritance : In OOP is a mechanism that enable classes to inherit usually state and behavior form other classes. For example , in figure 3 , there are 3 classes: Mammal , Cat and Dog. The Dog and Cat classes both inherit from Mammal. Thus , an object of Cat or Dog can implement the method getEyeColor from the class Mammal . However , the main advantage of inheritance is that intended to assist reuse existing code with little or no modification.

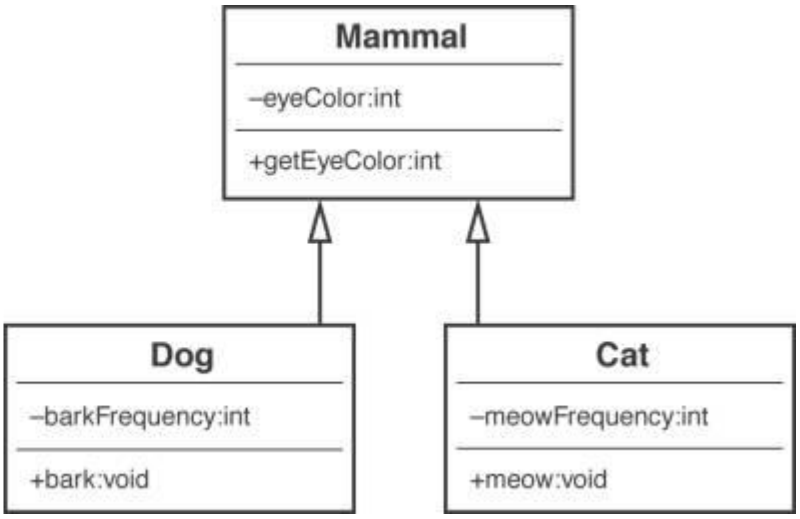


Figure 3:The Dog and Cat classes both inherit from Mammal.

2.2.4 Abstraction: involves the formulation of representations by focusing on differences and similarities with a set of entities to extract intrinsic essential characteristics (relevant common features) and avoid extrinsic incidental characteristics (irrelevant distinguishing features) with the purpose of define a single representation having those characteristics that are related to defining every element in the set.

2.2.5 Encapsulation: makes the methods and data private within an object. The gale of encapsulation is to hiding details of the internal mechanisms and data structures in turn to facilitate abstraction. Therefore, this allows the implementation to be changed without impacting the users of the interface.

2.2.6 Polymorphism: in different ways , the different objects have ability to receive the seam message and behave. Developers only need to understand the interface to use the object .As a result, this can lead to improve code readability.

2.2.7 Coupling: describes or measures dependant one object is on another object. Objects that are tightly coupled can be changed quite radically without impacting each other. The least change to objects that are tightly coupled can cause a mass of problems.

2.2.8 Cohesion : An object with high cohesion is defined for one reason and it only performs that purpose. An object with low cohesion able to do a lot of different things.

2.3 Impedance mismatch

As stated in introduction of this paper, the object oriented technology chains the building of applications within objects that have both data and behavior. Relational technology support the storage of data in tables and treatment of that data through (DML) inside within the database through stored procedures and externally through SQL .Due to the difference technologies in underlying paradigms they are do not work together seamlessly. They are have different type language. For example , java as object oriented language has string whereas Oracle as database has a varchar. Moreover , the majority of relational databases do not support schema inheritance. However , today object oriented considered the fashionable programming languages at the same time the most used database system are relational database. Indeed , the problem came out when objects need to be stored in relational database. There has been a lot of efforts have been made to overcome this. One of the best solutions so far to bridge the gap between these two systems was Object Relational Mapping (ORM or also known as O-R)[15][11][24].

2.4 Object Relational Mapping

Object Relational Mapping (ORM) is a programming technique work as bridge to convert data between incompatible type systems in object-oriented programming and relational databases [11][22]. ORM is a specialization of the general concept of object persistence. It helps making the software more robust via reduces most often the lines of code programmed. The ORM support database-access technology such as separation of concerns , inheritance , information hiding , change detection and database independence. Separation of concerns is the process to divide a program into logical parts in order to prevent overlap or at least reduce it. There are several part must be separated in database programming such as business methods that find domain objects. Information hiding is an implementation strategy that reduces cost and complexity via defining behavior in terms of interface that are implemented via certain classes. Thus ,the behavior of the subclasses is separate from the behavior of the callers. On the other words , changing on one side does not effect to change the other side. Inheritance allows to reuse of code which can help programmers to spend less time rewriting the same code. Change detection is tracking changes to domain object that are used in a database transaction so that at the end of the transaction, will ultimately reflect those changes to the database. Database independence allows applications to work with various databases without the need for change in the application for the purpose of compatibility with the new database.

2.4.1 The modus operandi of ORM

After mapping domain object model classes to relational tables, developers can access them through an API , which is implemented by a persistence provider. In the application layer, data stored in relational databases is represented as objects in the native object programming language. While the application navigates among objects, the data store transparently maps the objects into the cache [14]. Also , the database manages objects in the cache, tracking changed objects and automatically writing them back to the database for a transaction. Moreover , queries against the database are expressed in terms of the domain object model. SQL statements generates directly from the of the domain object model by the provider.

2.4.2 Mapping classes to tables

Inheritance in the domain object model is a relationship between two or more in which one class is specialization of another. There are many ways to map domain object model to relational database. For implementing Inheritance structures to a relational database there are three strategies.

2.4.3 A single table inheritance approach maps all classes in the inheritance hierarchy to a single table that contains a column for each of the fields in any of the class. We present an example to illustrate the issues related to Inheritance mapping. another. Figure 2 shows a animal relations object model in which Cat and Dog are specializations of Mammal. Figure $ depicts the persistence model for the class hierarchy of Figure 2 when this approach is taken. Notice that a MammaOID column was introduced for the primary key of the table.[11][22][13] [24]

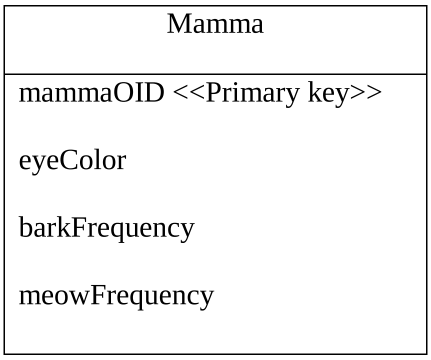


Figure 4: Mapping all classes to a single table

The advantages of A single table inheritance approach are that it is simple, that polymorphism is supported when a person changes roles, and that ad hoc reporting (reporting performed for the specific purposes of a small group of users, who commonly write the reports themselves) is also very easy with this approach because all of the personal data you need is found in one table. The disadvantages are that when a new attribute is added anywhere in the class hierarchy a new attribute should be added to the table[11][22] [24]

2.4.4 A table-per-concrete-class inheritance strategy each data entity includes both the attributes and the inherited attributes of the class that it represents. Figure 5 depicts the persistence model for the class hierarchy of Figure 2 when this approach is taken. There are data entities corresponding to both the Cat class and the Dog class because they are concrete, but not the Mammal class because it is abstract (indicated by the fact that its name is depicted in italics). Each of the data entities was assigned its own primary key, dogOID and catOID respectively. The main advantage to A table-per-class inheritance approach is that it is still fairly easy to perform ad hoc reporting, given that all the data you need about a single class is stored in only one table. There are a number of disadvantages, however. One of them is that when you change a class you must change its table and the table of any of its subclasses. For example, if you were to add weight and height to the Mammal class, you would need to update both tables, which is a lot of work. In addition third, it is difficult to support multiple roles and still maintain data integrity.[11][22][24]

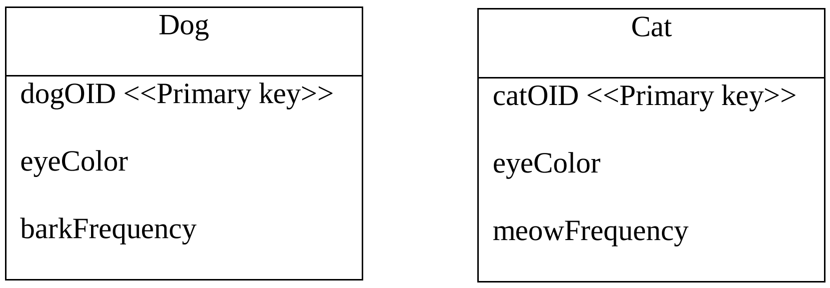


Figure 5. A table-per-concrete-class.

2.4.5 A table-per-class inheritance strategy you create one table per class, the attributes of which are the OID and the attributes that are specific to that class. Figure 6 depicts the persistence model for the class hierarchy of Figure 6 when A table-per-class inheritance approach is taken. Notice that mammalOID is used as the primary key for all three tables. The major advantage of this approach is that it conforms best to object-oriented concepts [11]. It supports polymorphism very well as you just have records in the appropriate tables for each role that an object might have. Moreover , extremely simple to modify superclasses and add new subclasses because you merely need to modify or add one table. On the other hand ,there are a number of disadvantages to this approach. First , there are many tables in the database -- one for every class, actually (plus tables to maintain relationships). Second, it takes longer to write and read data using this technique because you must access multiple tables. Third, ad hoc reporting on your database is difficult, except you add views to simulate the desired tables[11][22].

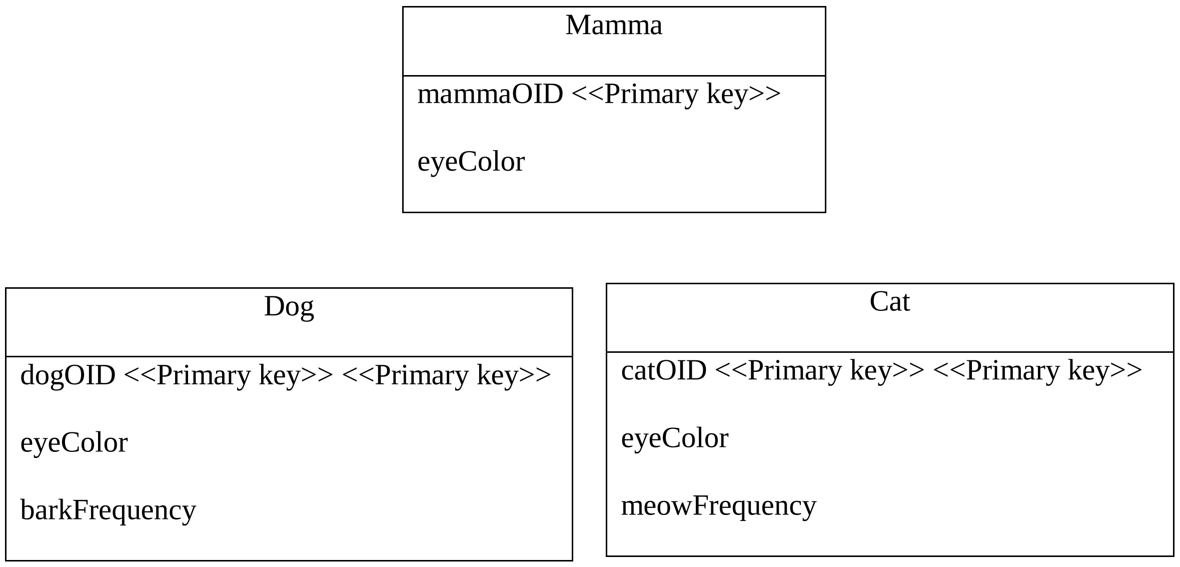


Figure 6. Mapping all classes in to its own table .

2 .5 Competition between the two paradigms

The important issue is that which paradigm is more appropriate in terms of flexibility with respect to requirements change. In the next chapter, we will discuss this point in more detail .

3. Need to Study

Today, changing requirements have become an accepted fact of life for software developer’s .Therefore; we looking for flexible approach can deal with requirements change. However, ORM is very popular in use. According to [11], most of the modern applications are built using ORM technology to access data stored in relational database rather than traditional approach. In addition, it is also claiming that using ORM tools can help reduce cost of a project. Moreover, proponents of object oriented approach tend to assume that an object oriented business model will made the system easier to maintain, easier to extend, and easier to re-use. On the other hand, proponents of traditional fashion argue that not all the world must be handled in objects. In addition, they are arguing that there is some native incompatibility between ORM code and databases. For example, Kenneth argues that you can write efficient and clean code if you know how databases work on their own terms. Moreover , Object-oriented development promises to reduce the maintenance effort, but, these promises are not based on reliable experimentation[12].Indeed , there is a significant lack of research to answer the question of whether object-oriented approach are better than traditional approach in terms of flexibility to requirements change.

The aim of this paper is to critically examine this assumption and to carefully compare the applicability and flexibility of the object oriented system in comparison to relational system. The findings from this project will be significant for practical application where the business logic layer is implemented in an object oriented fashion as the present trend in enterprise computing seems to be heading.

Chapter 3

Related work and Thesis's proposal

In this chapter, we review further some papers that are highly relevant to the issue of addressed by this thesis.

4.1 Previous works

There are few complete experimental results comparing the effectiveness of object oriented design methods. The results are available in [16].This experiment compared the effectiveness of subjects when maintaining both object oriented and procedural designs. Generally, they found that object-oriented designs were more difficult to maintain. Particularly, they found strong results that bad object-oriented designs were more difficult to maintain than procedural designs. In addition, they also found statistically weaker results suggesting that good procedural designs were easier to modify then good object oriented designs. This study was based on a small sample size of 20 students. The conclusion is that, the shift to object-oriented methods will cause a substantial drop in productivity, especially if developers are experienced with procedural methods. Moreover , the study conducted by [15] mention that traditional and some object oriented business process modeling techniques lead to a procedural process model which causes mismatch problems and impedes implementation when they have to be coded in an object based GUI language

The study conducted by [17] was looking at a smaller subset of object-oriented maintainability characteristics. in order to enhanced control the experiment, they looked only at whether inheritance, a fundamental object oriented construct, made programs more difficult or less difficult to maintain. They studied two groups about 57 students. The result was that for complex programs, inheritance makes maintenance more difficult, not less.

The study described in [18] evaluates the comprehension of both object-oriented and procedural code. This research used 30 professional developers in two phases that were at least a week apart. Comprehension was considered as the number of files each subject opened. The results of this research showed that procedural group had larger scope of comprehension. As a result, this means that those subjects maintaining a procedural project needed to open a larger percentage of the source files than subjects maintaining an object-oriented project. However, the largest gap in this study is regarding the definition of comprehension. The number of files for each version of the system is not clear from the experimental design. However, if the number of files in the object-oriented version was larger, then defining comprehension as a percentage of the files that was viewed does not seem to be an accurate measure.

In the contrast, this paper [22] argues against traditional approach. They stated that building application using triggers, views, and stored procedures is problematic for a number of reasons. First, unions or views containing joins are usually not updatable. Second, triggers and defining custom database views for every application accessing mission-critical enterprise data, is rarely acceptable due to security and manageability risks. Furthermore, object-relational features, SQL dialects, and procedural extensions vary significantly from one DBMS to the next.

Moreover, several examples in the literature find that object – oriented approach has many advantages over traditional approach. The study described in [14] compares the implementation of a GIS (Geographic Information System) database using RDBMS and OODBMS developed. They stated that, the most important advantage of using objects as the basis of database design is the ease of using them for both user verification and design. They also found, the object model can be directly implemented in a relational form, particularly the *has a* relationship or aggregation. In addition, they mention that , although normalization is a well developed methodology but , requires training and is difficult for users to understand. In contrast, Object modeling technique uses user information in the form of uses cases, which becomes the basis for the class structure. As a result, this makes the structure and the naming of objects in the structure easier for users to understand and hence verify. The conclusion is that, object models appear to be a more useful way of communicating analysis and design concepts to users.

The study conducted by [20] was based on using Hibernate as an object-relational mapping framework for eHealth systems. They found that Hibernate provokes an outstanding effect on an application development. They also stated that , the first remarkable characteristic consists of a more natural object oriented programming

11. REFERENCES

1. Codd, E.F. A Relational Model of Data for Large Shared Data Banks. CACM 13(6): 377-387. 1970.

2. Aguirre-Urreta, M. I. and Marakas, G. M. 2008. Comparing conceptual modeling techniques: a critical review of the EER vs. OO empirical literature. SIGMIS Database 39, 2 (Apr. 2008), 9-32.

3. Lodhi, F. and Ghazali, M. A. 2007. Design of a simple and effective object-to-relational mapping technique. In Proceedings of the 2007 ACM Symposium on Applied Computing (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY,

4. Urban, S. D. and Dietrich, S. W. 2003. Using UML class diagrams for a comparative analysis of relational, object-oriented, and object-relational database mappings. In Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (Reno, Navada, USA, February 19 - 23, 2003). SIGCSE '03. ACM, New York, NY, 21-25.

5. Sinha, A. P. and Vessey, I. 1999. An empirical investigation of entity-based and object-oriented data modeling: a development life cycle approach. In Proceedings of the 20th international Conference on information Systems (Charlotte, North Carolina, United States, December 12 - 15, 1999). International Conference on Information Systems. Association for Information Systems, Atlanta, GA, 229-244.

6. Chen, P. P. 1994. The entity-relationship model—toward a unified view of data. In Readings in Database Systems (2nd Ed.), M. Stonebraker, Ed. Morgan Kaufmann Series In Data Management Systems. Morgan Kaufmann Publishers, San Francisco, CA, 741-754.

7. Blaha, M. R., Premerlani, W. J., and Rumbaugh, J. E. 1988. Relational database design using an object-oriented methodology. Commun. ACM 31, 4 (Apr. 1988), 414-427.

8. O'Neil, E. J. 2008. Object/relational mapping 2008: Hibernate and the entity data model (edm). In Proceedings of the 2008 ACM SIGMOD international Conference on Management of Data (Vancouver, Canada, June 09 - 12, 2008). SIGMOD '08. ACM, New York, NY, 1351-1356.

9. Keller, R. K., Schauer, R., and Cockburn, A. 1997. Object-oriented design quality. In Addendum To the 1997 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum) (Atlanta, Georgia, United States, October 05 - 09, 1997). OOPSLA '97. ACM, New York, NY, 63-67.

10. Booch, G. 1991. Object-oriented Analysis and Design with Applications. Benjamin-Cummings Publishing.

11. Russell, C. 2008. Bridging the Object-Relational Divide. Queue 6, 3 (May. 2008), 18-28.

12. Hall, John M., The Maintainability of Object-Oriented Software, 2003.

13. Douglas Barry, Torsten Stanienda, "Solving the Java Object Storage Problem," Computer, vol. 31, no. 11, pp. 33-40, Nov. 1998, doi:10.1109/2.730734

14. Crowther, PC and Hartnett, JS, *Handling Spatial Objects in a GIS Database-Relational v Object Oriented Approaches*, Proceedings of GeoComputation 2001, September 23 - 26, 2001, Brisbane, pp. 6

15. R. Polan, "Impedance Mismatch of Business Process Modeling and Object-Oriented Implementation," seep,pp.370, 1998 International Conference on Software Engineering: Education & Practice, 1998

16. Briand L., Bunse L., Daly J., Differding C. An Experimental comparison of the Maintainability of Object-Oriented and Structured Design Documents, Empirical Software Engineering – An International Journal, 1997. Available at: http://citeseer.nj.nec.com/12374.html.

17. Unger B., Prechelt L. The impact of inheritance depth on maintenance tasks: Detailed description and evaluation of two experiment replications. Technical Report 19/1998, Universität Karlsruhe, Fakultät für Informatik, Germany, July 1998. Available at: http://www.ipd.uka.de/~prechelt/Biblio/inheritTR.pdf.

18. Corritore C., Wiedenbeck S. Direction and Scope of Comprehension-Related Activities by Procedural and Object-Oriented Programmers: An Empirical Study, Proceedings of the 8th International Workshop on Program Comprehension (IWPC'00). 2000. Available at:http://www2.umassd.edu/SWComprehension/compdocs/nebraska/corritoreiwpc00.pdf

19. E. Bertino, L. Martino: Object-oriented Database Management Systems: Concepts and Issues; IEEE Computer 24 (4) 1991, 33-47

20. Hibernate as an object-relational mapping framework for eHealth systems. Software Technologies Session Alberto Moreno Jiménez, Elena Villalba Mora, M.T. Arredondo Life Supporting Technologies ETSI Telecomunicación Universidad Politécnica de Madrid

21. Downs K: Why I do not use ORM. August 25, 2008. Available at: http://database-programmer.blogspot.com/2008/06/why-i-do-not-use-orm.html

22. Melnik, S., Adya, A., and Bernstein, P. A. 2007. Compiling mappings to bridge applications and databases. In Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data (Beijing, China, June 11 - 14, 2007).

23. Engels G., Kappel G. Object-Oriented System Development: Will the New Approach Solve Old Problems? IFIP 1994 Congress, Vol. 3, K. Duncan and K. Krueger (eds.), North-Holland, September 1994. Available at: http://citeseer.nj.nec.com/engels94objectoriented.html.

24. Scott W. Ambler, "Mapping objects to relational database", White Paper, Ronin International, October 21, 2000.

La Trobe University

Bundoora, Victoria, Australia

Faculty of Science, Technology & Engineering

Department of Computer Science and Computer Engineering

Design and Implementation of Business Logic Layer

Object-Oriented Design vs. Relational Design

Author:	Ali Ahmed Alharthy
Student No:	15038770
Course:	Master of Information Technology
Submission Date:	October 2009
Supervisors:	Dr. Kinh Nguyen