**Dublin Institute of Technology**

14

**Patterns Report**

**Assignment - Bookstore**

**Gillian Rowan**

Contents

[Transfer Object Pattern – POJO called by action classes 2](#_Toc385190547)

[Singleton Pattern – database object 2](#_Toc385190548)

[Data Access Object Pattern – Book 2](#_Toc385190549)

[Strategy Pattern – Payment 3](#_Toc385190550)

[MVC Pattern 3](#_Toc385190551)

[Prototype Pattern 3](#_Toc385190552)

[Features 4](#_Toc385190553)

[Github Account 4](#_Toc385190554)

I had started doing this project using Node.js, express and MySQL but I felt with the little amount of time left to complete it that I should use a language that I am more familiar with so I changed to Java with Struts2 and MySQL because I could use the knowledge gained in my final year project.

# Transfer Object Pattern – POJO called by action classes

The transfer object pattern is used when we want to pass data with multiple attributes from client to server. The transfer object can also be known as the value object. It is a simple POJO with getter and setter methods. It doesn’t have any behaviour. The server send connects to the database and gets the values for the client. The transfer objects in my bookstore are the entities, book, customer, administrator and the business objects on the server side are the action classes related to the entities. The business object gets the data and puts the data into the database for the entities or POJO’s.

# Singleton Pattern – database object

I used the singleton pattern to create a connection to the database. The singleton pattern involves only one class that is responsible to instantiate itself to make sure that not more than one instance is created in the application. It will also provide a global access point for the object being created. The class ConnectionFactory uses that singleton pattern and has an instance of it. It is able to get a reference to the instance. The other classes in the application can then just call the ConnectionFactory.getConnection () method in order to connect to the database properly.

# Data Access Object Pattern – Book

The Data access object pattern is used to separate low level data accessing and operations from the business services. There are three parts to the DAO pattern, the interface, the model and the concrete class. I used the DAO pattern for a Book object. The interface defines methods for the concrete class to implement. These methods are then used in the action class to access on the web application.

# Strategy Pattern – Payment

The strategy pattern is used to change class behaviour at run time. Objects are created that represent various strategies and a context object whose behaviour varies per strategy object. I used the strategy pattern to implement the different ways customers could pay for their books in the store. I have an interface PaymentStrategy with one method, pay(). The concrete classes implement the interface and define different implementations of the pay method. In my application, there were two concrete implementations, PayPal and credit card strategies.

# MVC Pattern

The MVC pattern is the Model View Controller pattern. It is used to separate each of the applications concerns. The model represents the object or a POJO. The view is the visualisation of the data that the model concerns and the controller acts on both the model and the view. The controller will control the flow of data into the model object and updates the view when data changes. I used the Struts2 MVC for the application. The models in my application are the entities, Book, Customer, Administrator, Cart and Order. The views in the struts MVC are represented on JSP pages and are accessed through the struts.xml file using actions. The Action classes in the struts MVC are the controllers for the application. It defines methods to change or update data that the user of the web application enters into the view pages.

# Prototype Pattern

The Prototype pattern is known as a**creational** pattern, as it is used to construct objects such that they can be decoupled from their implementing systems. In my example, I created an abstract LineItem class that included a clone method. Then I provided a concrete implementation using BookLineItem. Then I tried to create a basic item using the clone method in the order class.

# Features

I have included my SQL files in order to create the database needed for this application. There will be an initial administrator with login details of email: [gill@book.com](mailto:gill@book.com) password: test. There will also be customers and books included in the initial set up.

Customers can log in; the book catalogue shows up on their home page. They can also search the catalogue using three different methods, by title, author and category. Each search allows for partial matching from the database. The results can be sorted into ascending and descending order by clicking on the column header. If the customer clicks the checkbox and clicks add to cart, a simulation of adding to the cart and proceeding to checkout and payment of the items will begin. A customer will also be able to simulate reviewing a book.

Administrators can log in, add a new administrator to the book store, and add a book to the catalogue, view stock in the store and search the bookstore the same way customers can.

# Github Account

<https://github.com/gillroro/Bookstore.git>