Persistence Framework for

Database Management System

* Mohana Deepthi Vadlamudi(1001258860)

CSE 6324, Fall 2016

University of Texas at Arlington

Table of Contents

[1. Introduction & Abstract 3](#_Toc466494248)

[2. Assumptions 3](#_Toc466494249)

[3. Technology Stack 4](#_Toc466494250)

[4. Persistence Framework 4](#_Toc466494251)

[5. Design Problems & Solutions 4](#_Toc466494252)

[Bridge: 5](#_Toc466494253)

[Command Pattern: 5](#_Toc466494254)

[Factory 5](#_Toc466494255)

[Template 6](#_Toc466494256)

[6. Sequence Diagram 6](#_Toc466494257)

[7. Class Diagram 12](#_Toc466494258)

[8. Screenshots 13](#_Toc466494259)

# Introduction & Abstract

Software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design. It is a description or template for how to solve a problem that can be used in many different situations. Design patterns are best practices that the programmer can use to solve common problems when designing an application or system.

Using these solution providing design patterns, need to design and implement a persistence framework for accessing a local database.

The project objectives are to

* Use controller, bridge, command, template method, and factory method patterns.
* Handle database management system (DBMS) by supporting operations like saving, retrieving, updating and deleting data objects
* Support undo and redo of above DML operations
* Considered two Database tables : TextBook and DigitalBook with the following fields.

**TextBook :**

`bookId` int(11) NOT NULL,

`bookName` varchar(255) DEFAULT NULL,

`authorName` varchar(255) DEFAULT NULL,

`ISBN` varchar(255) DEFAULT NULL,

`price` float DEFAULT NULL,

PRIMARY KEY (`bookId`)

**DigitalBook :**

`bookId` int(11) NOT NULL,

`bookName` varchar(255) DEFAULT NULL,

`authorName` varchar(255) DEFAULT NULL,

`devicecompability` varchar(255) DEFAULT NULL,

`price` float DEFAULT NULL,

PRIMARY KEY (`bookId`)

Please refer the attached cse1\_digitalbook.sql and cse1\_textbook.sql for the database tables.

# Assumptions

Below are the assumptions made:

* Application created will be desktop application and UI is built using Java Swings
* Database tables and structures will be setup in local MySQL database before using the framework

# Technology Stack

* Programming language – Java
* DBMS – MySQL

# Persistence Framework

A persistence framework is middleware that assists and automates the storage of program data into databases. It acts as a layer of abstraction between the application and the database.

These frameworks map the objects in application domain to data that needs to be persisted in database. I.e. There must be some mapping between a class and its persistent store (for example, a table in a database), and between object attributes and the fields in a table record.

Using these mappings the DB operations on tables can be performed via objects in application and persistence can be directly handled by application. So persistence framework is a service used to provide object to record mapping.

# Design Problems & Solutions

Below are the design patterns applied to build the persistence framework for local database.

**Problem1:** Need to provide loosely coupled logic between client interface and DB layer .

**Solution :** Used Controller pattern. This separates the client interface implementation from Database

**Problem2:** Need to provide different database implementations and need to take dynamic decisions to connect to any specific database at runtime. If the new database implementation is introduced, which will tightly couple the existing database implementation in the project.

**Solution2 :** Implementation of bridge pattern, which will decouple the different database implementation.

## Bridge:

Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern decouples implementation class and abstract class by providing a bridge structure between them.

This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

**Problem 3:**

Duplication of database operation logic for different set of operations as commands will cause less maintainability and duplication of the business logic .

**Solution3:**

Implementation of Command pattern. It encapsulates operations as command objects and allow them to execute in flexible way. If new command has to be introduced we can plug in that command by implementing a separate command object and plug it to the existing business logic. This pattern is perfect apt for doing undo and redo operations also.

## Command Pattern:

Command pattern is a data driven design pattern. A request is wrapped under an object as command and passed to invoker object.

Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

**Problem4:**

Difficulty in varying the types of the objects which are used by different business logic algorithms.

**Solution 4:**

Implementation of Factory pattern. It introduces abstract class with a skeleton of the algorithm in which types of objects that vary are returned by abstract method calls. Factory methods are implemented by subclasses to return different types of objects.

## Factory

Factory pattern provides one of the best ways to create an object. In this pattern we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

**Problem5:**

Introducing different set of conditional statements is tedious to handle and execute the business algorithm logic dynamically

**Solution5:** Implement Template pattern.it introduces a generic abstract object with a skeleton of business logic in which the steps to vary the business logic calls using abstract methods and refer hook methods. By implementing sub classes hook methods to provide various behavior dynamically.

It requires less effort to add concrete sub classes, which reduces the redundancy of the business logic. The behavior of the template methods can be changed dynamically by changing the pointing to different concrete class object.

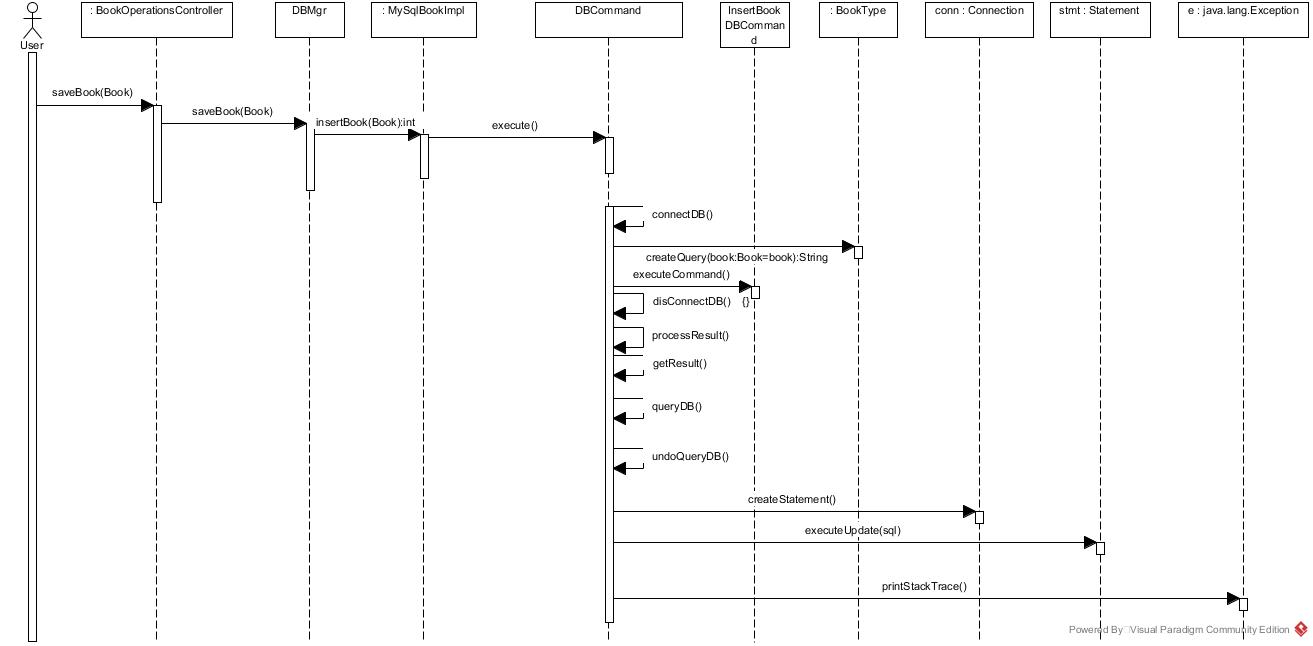
## Template

In Template pattern, an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class.

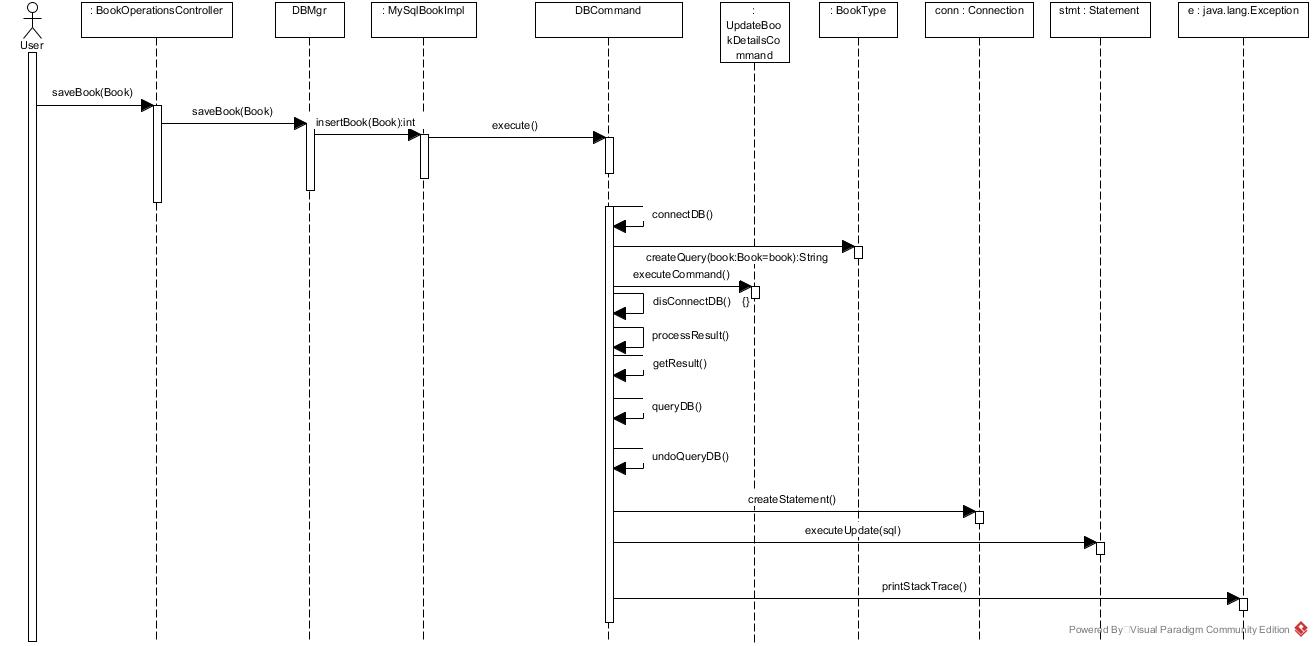
# Sequence Diagram

Below diagram shows the implementation of sequence diagrams.

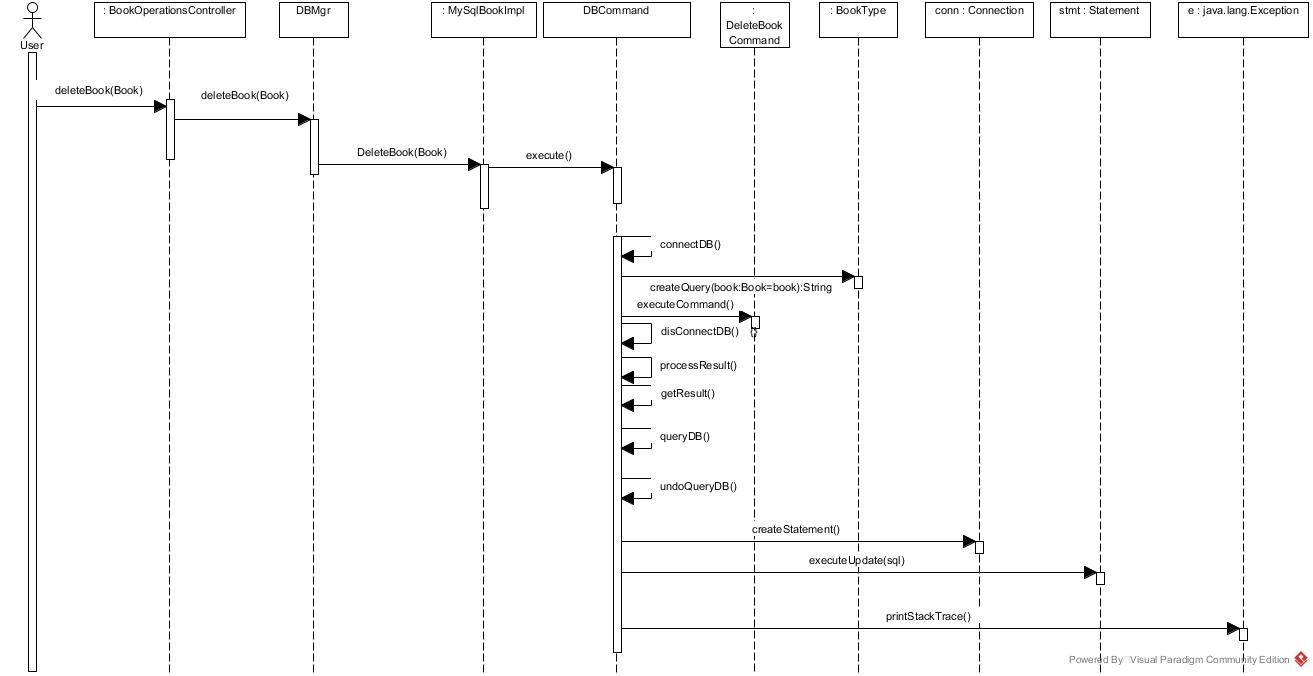
InsertBook:



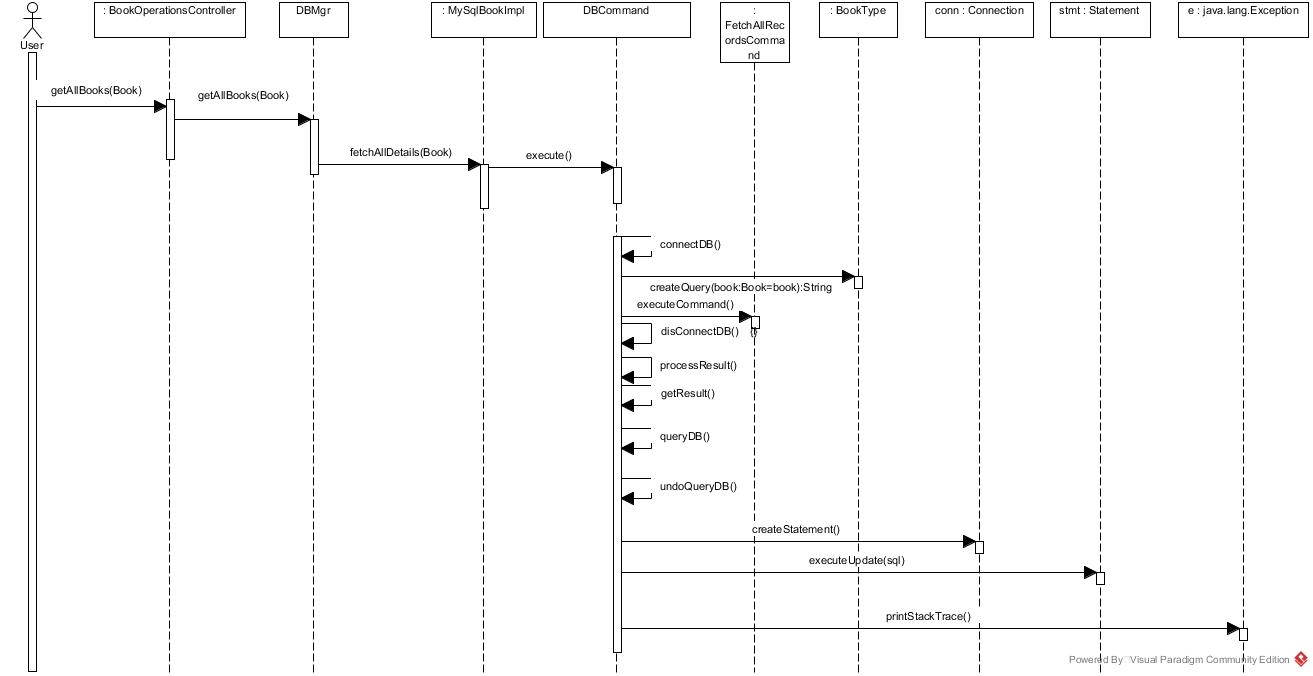
Update Book Details sequence diagram:



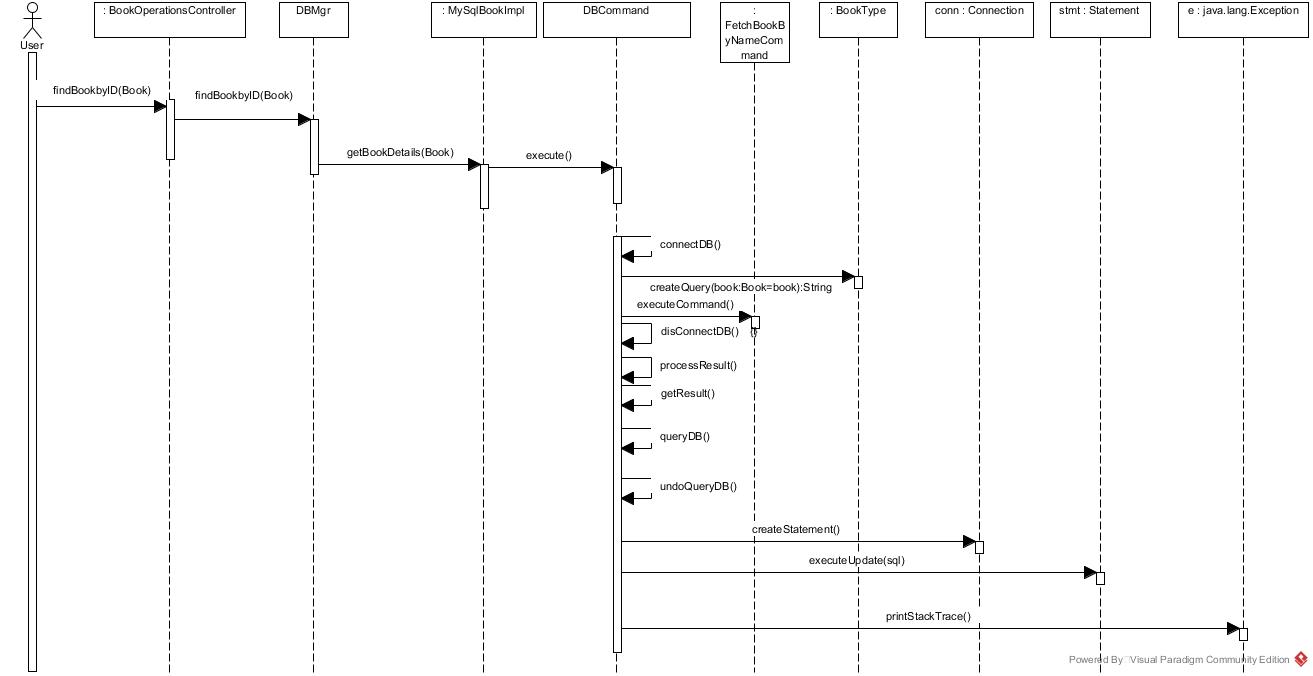
Delete Book Sequence Diagram:



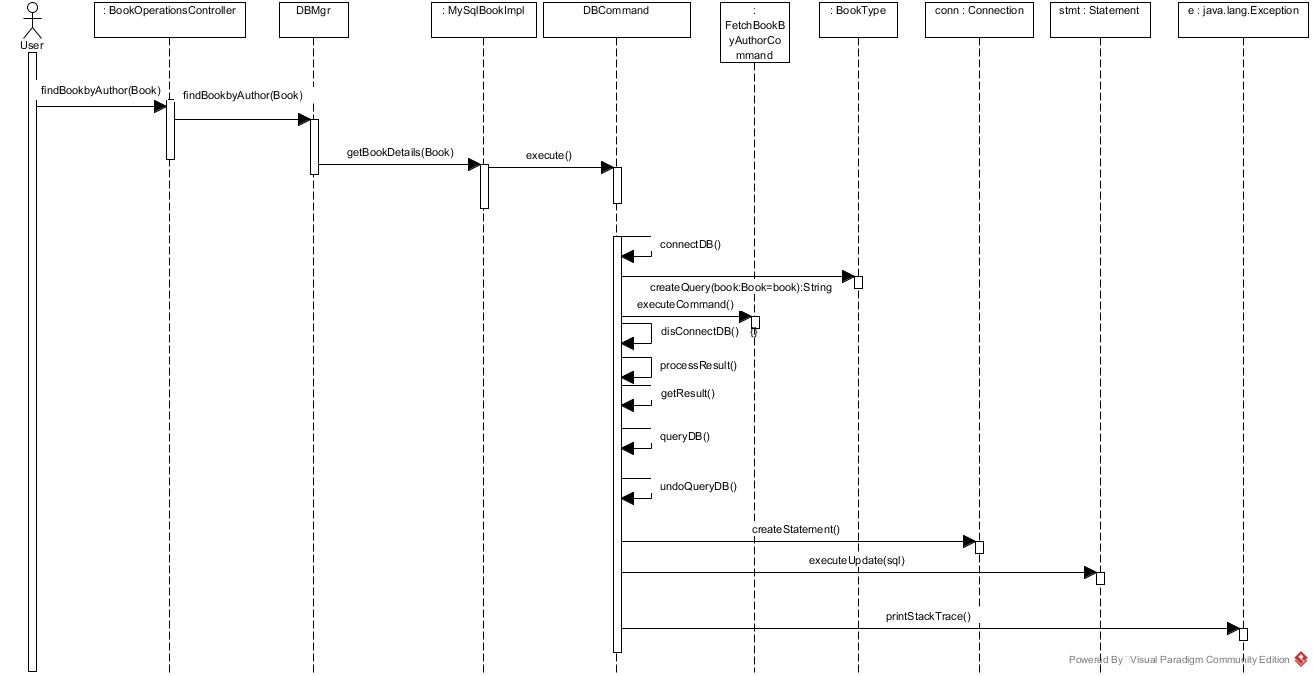
Fetch all Sequence diagram:



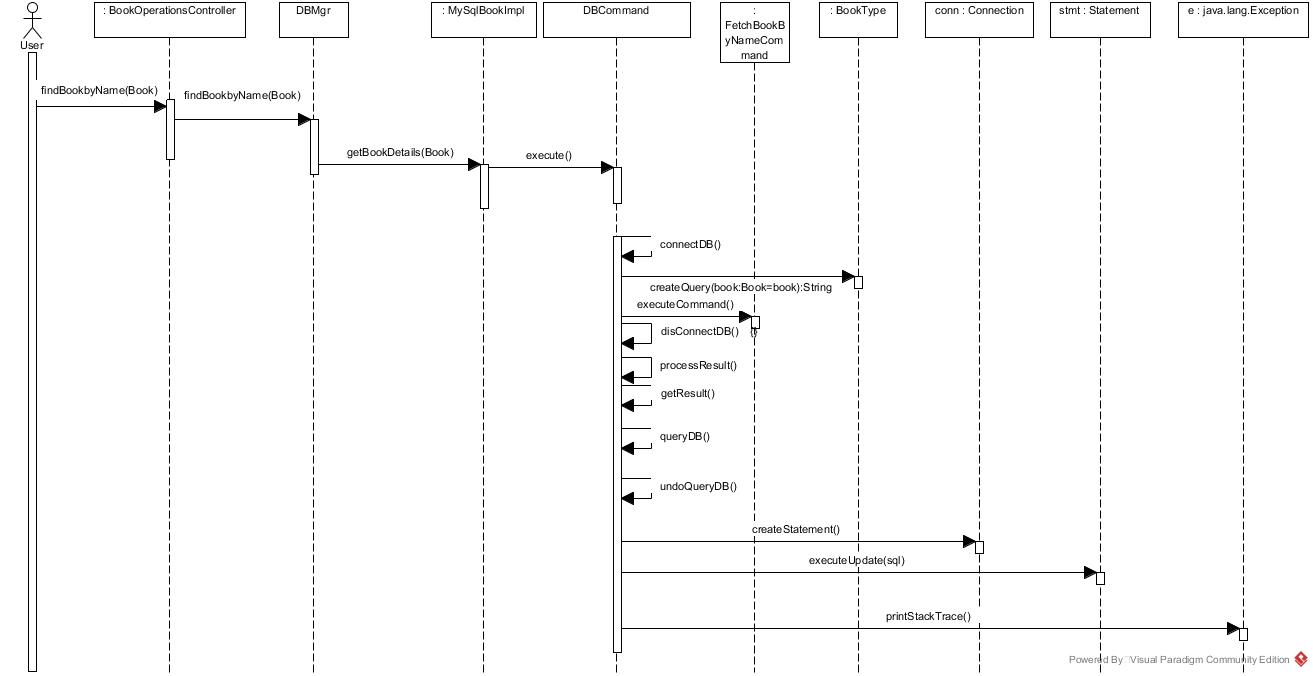
Fetch by Id sequence diagram:



Fetch by Author sequence diagram:

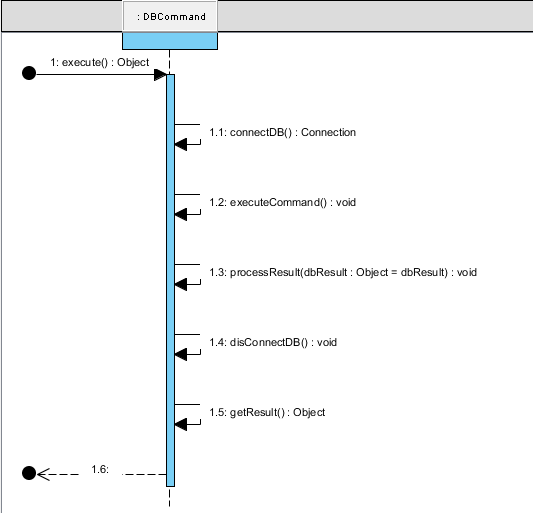


Fetch by Book Name:

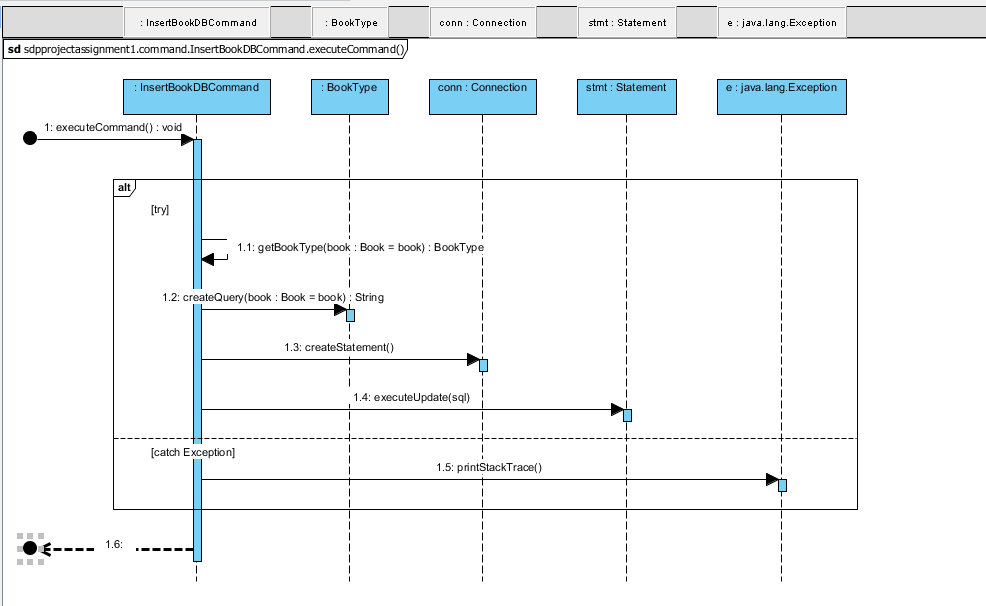


Below diagram shows the implementation of patterns with sequence diagrams.

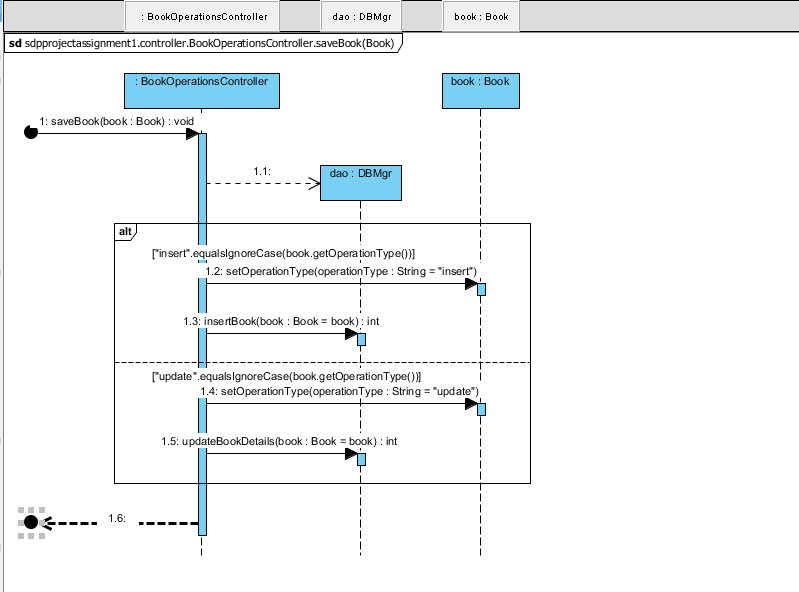
Template Pattern:



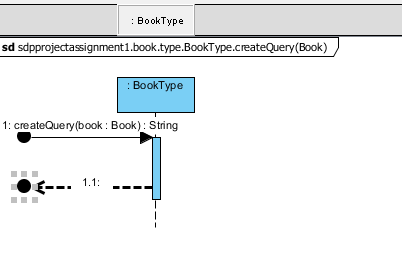
Command Pattern - Sample Insert DB Command

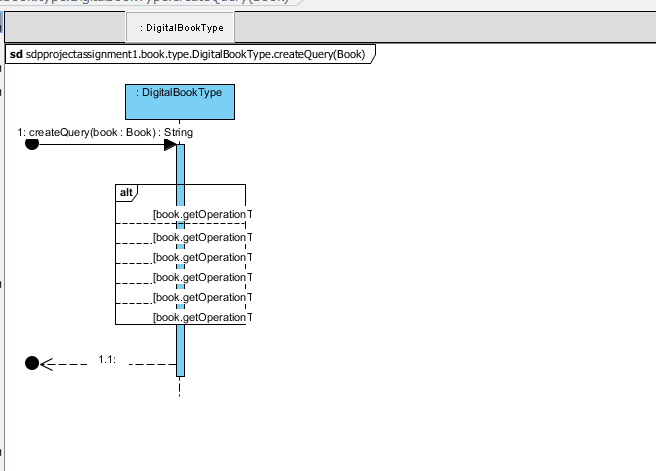


Bridge pattern:



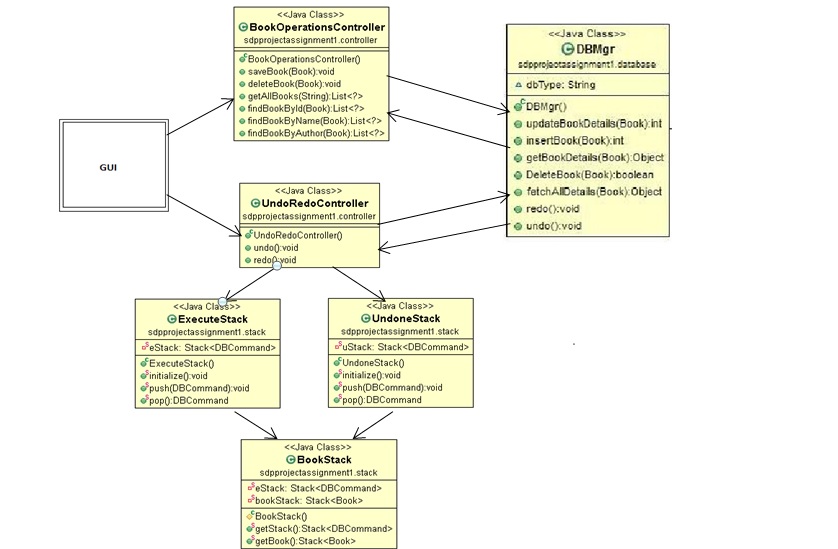
Factory pattern:

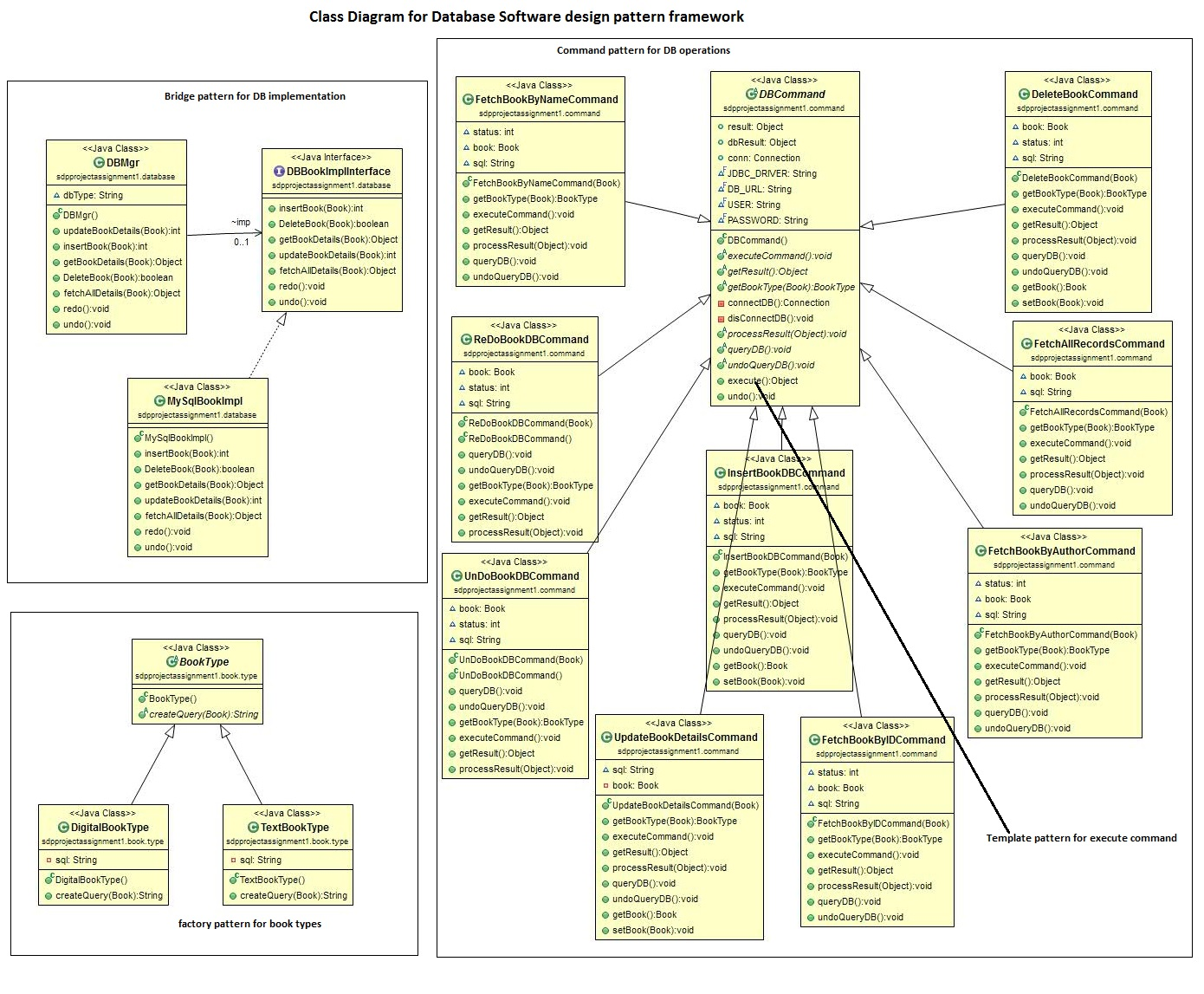




# Class Diagram

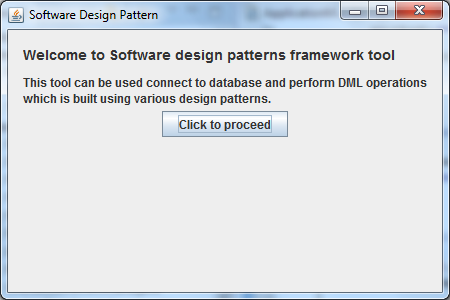
Below is the class diagram with all classes showing the implementation of design patterns





# Screenshots

Application Launch screen on running ApplicationManager.java:





Instructions to Run the project:

Each DB operation such as Update , Delete, Insert will be opened in new GUI window and operations Like Undo Redo will be executed in that window.