

# **A PROJECT ON BITS LIBRARY MANAGEMENT SYSTEM**



## **CS F213 OBJECT ORIENTED PROGRAMMING**

Group No: 54

Members-

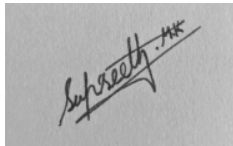
- Aryan Barapatre 2020B4A70833P
- Supreeth M K 2020B5A70596P
- Rikhil Gupta 2021A7PS0533P
- Anish Taori 2021A7PS0939P

Submitted to:  
Prof. Amit Dua

## PLAGIARISM DECLARATION

1. I know that plagiarism means taking and using the ideas, writings, works or inventions of another as if they were one's own. I know that plagiarism not only includes verbatim copying, but also the extensive use of another person's ideas without proper acknowledgement (which includes the proper use of quotation marks). I know that plagiarism covers this sort of use of material found in textual sources and from the Internet.
2. I acknowledge and understand that plagiarism is wrong.
3. I understand that my research must be accurately referenced. I have followed the rules and conventions concerning referencing, citation and the use of quotations as set out in the Departmental Guide.
4. This assignment is my own work, or my group's own unique group assignment. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism.
5. I have not allowed, nor will I in the future allow, anyone to copy my work with the intention of passing it off as their own work.

Supreeth M K



Aryan Barapatre:



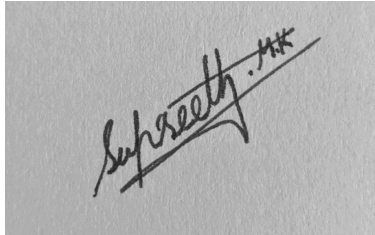



Rikhil Gupta:



Anish Taori:



## CONTRIBUTION TABLE

Name of Member	Contribution	Signature
Supreeth M K	User Details class, database query handling.	
Aryan Barapatre	Admin management class, Admin query class, Library Management class and Multithreading.	
Rikhil Gupta	Library class and User query class.	
Anish Taori	Report, GUI implementation, Admin class methods	

## Design pattern

Our project follows a singleton design pattern which is used in the `javaMySqlConnection`. The methods for connecting, updating and viewing the database remain the same for all the methods. These methods are utilized by every class. These methods are also synchronized so that we can implement multithreading on them. Since every class accesses the `javaMySqlConnection` class it needs to be Synchronized so that the values of the database do not change simultaneously causing an error.

Example: Say we have 2 threads issuing a book for a student. If the student already has 2 books issued under his name the program must let him issue one of the books but deny him to issue the other. Now if the above method was not synchronized, both the threads would have issued the book under the students name even though the maximum limit for issuing books is 3

If we were to redo the project, we would have used the creational pattern. This is because the pattern is especially useful when using polymorphism and choosing between different classes at runtime. Creational patterns allow objects to be created in a system without having to identify a specific class type in the code, so you do not have to write large, complex code to instantiate an object.

## **SOLID Principles**

**Single Responsibility Principle-** Class should do one thing and should therefore have a single reason to change.

This principle has been followed in our project. Every class has been assigned one responsibility and has methods pertaining to that. The library class only deals with library books. User details class and only responsibility of maintaining login portal page. And similarly in the admin classes.

Advantage-

Classes, software components and microservices that have only one responsibility are much easier to explain, understand and implement than the ones that provide a solution for everything.

**Open-closed principle-** classes should be open for extension and closed for modification.

This principle has only been followed only in Library class. In the user details class every method is very specific and in the admin class we have used final when using the database in order to satisfy the Liskov substitution principle.

Drawbacks-

It makes it difficult to add new features as we would have to change the existing code.

**Liskov substitution principle-** Subclasses should be substitutable for their base class.

In the admin class we have used final when using databases and implemented Liskov substitution principle.

**Interface segregation principle**-Separating the interfaces. Many interfaces are better than one general purpose interface.

We have not used the Interface segregation principle.

If we were to implement this principle, for each class that interacts with the database we could have created an interface that had a method which connects to the database and its return type must be boolean indicating whether the admin query was successfully implemented or not. For queries related methods implemented in user query class, we could have made 2 different classes implementing 2 separate interfaces, one of which has an abstract method whose return type is boolean since indicates whether an update to the database is successful. The other interface must contain a method whose return type is void since there is no use for a boolean value when simply viewing a database fails.

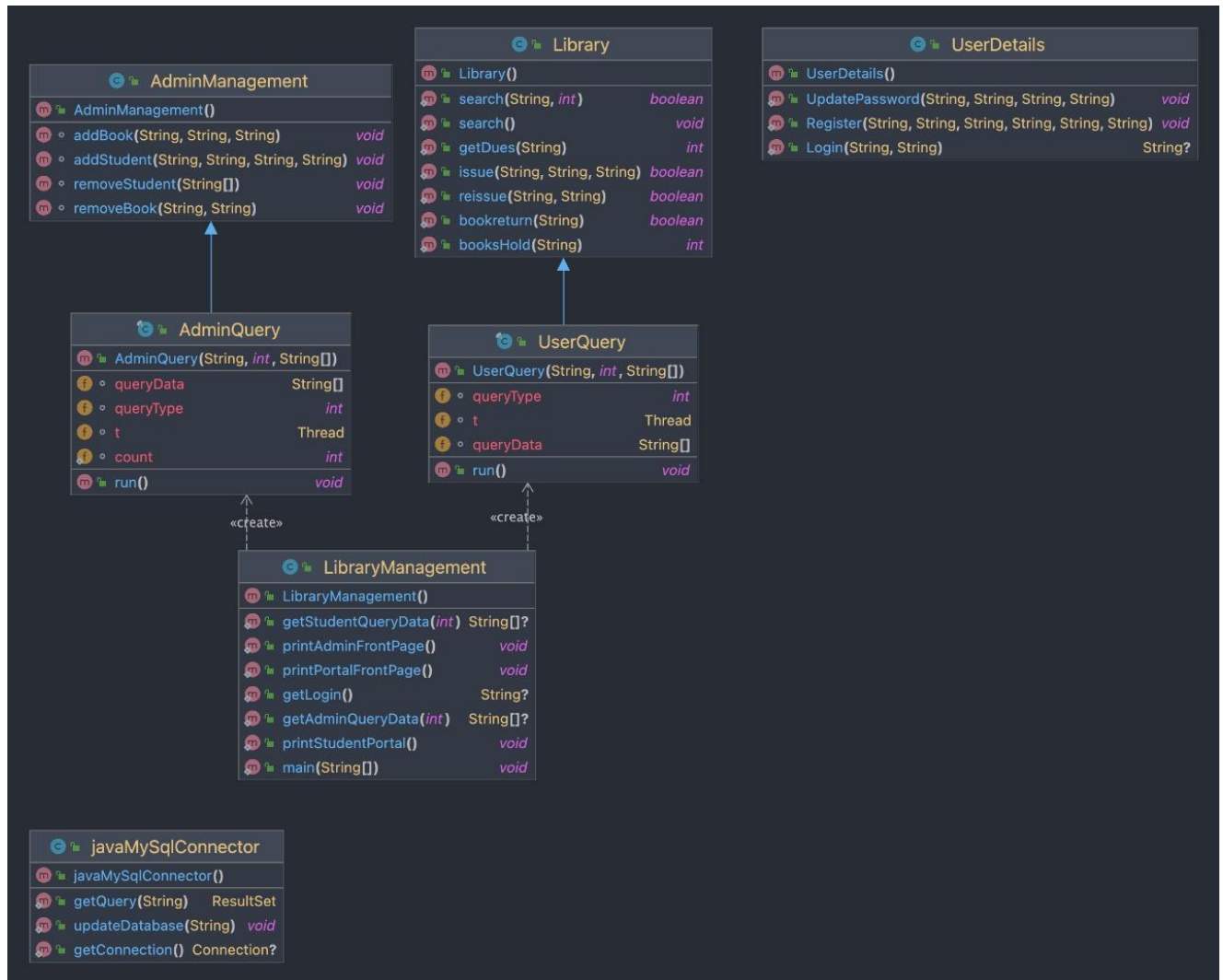
**Dependency Inversion Principle**- classes should depend on interfaces or abstract classes instead of concrete classes and functions.

The Dependency Inversion Principle is not applicable on our project as we have not used abstraction.

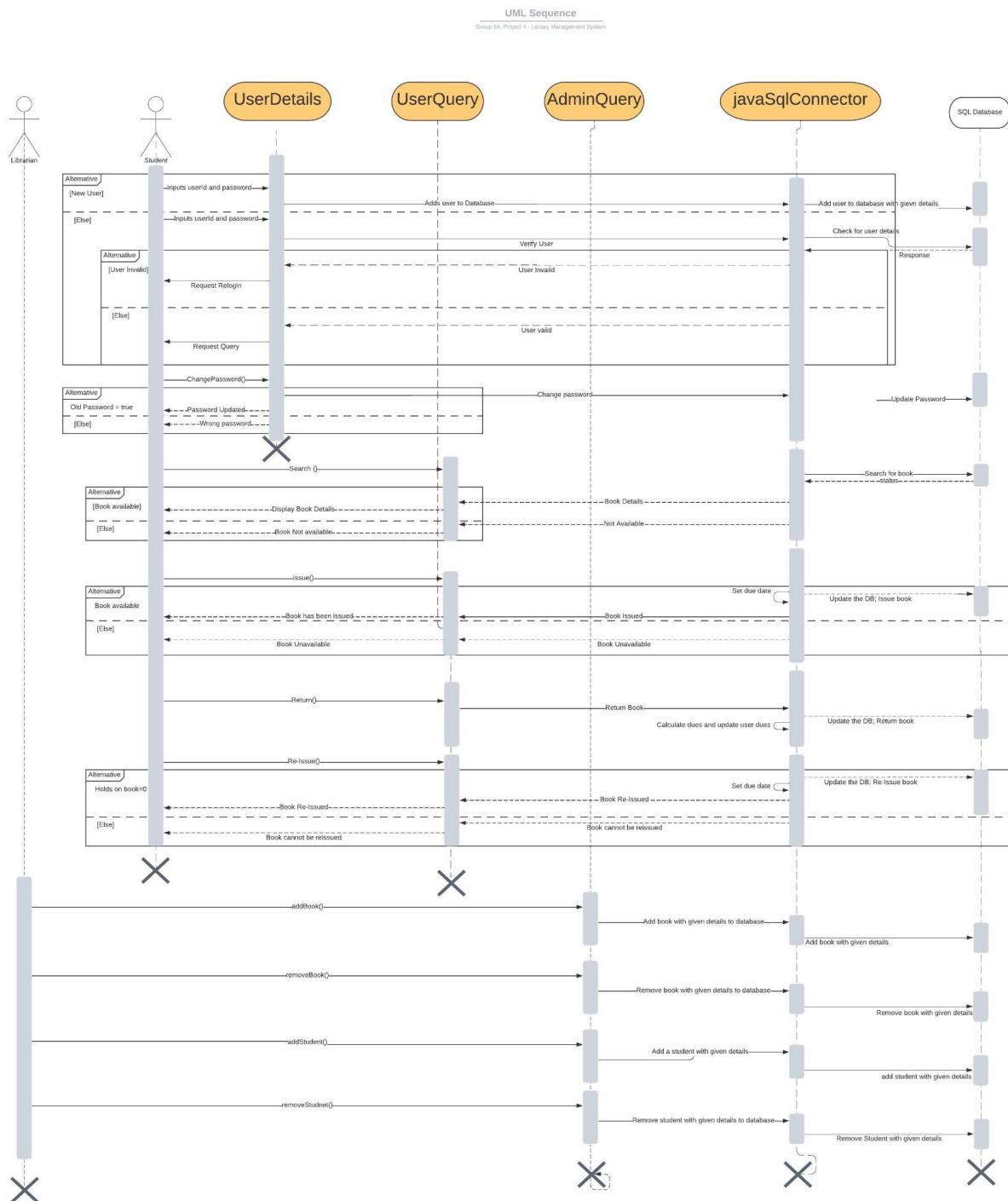
Drawback-

High-level modules, which provide complex logic, should be easily reusable and unaffected by changes in low-level modules, which provide utility features. To achieve that, we need to introduce an abstraction that decouples the high-level and low-level modules from each other.

# UML Class Diagrams



# UML Sequence Diagram





# UML Use Case Diagram

## UML Use Case

Group 54, Project 4 - Library Management System

