

Object Oriented Programming Project Report

OOP CS-F213

First Semester 2022-23

By

Yash Mundada

2021A7PS0001P

Aditya Khandelwal

2021A7PS2422P

Priyam Verma

2020A2PS1776P

Amarpal Singh

2019B4A80336P

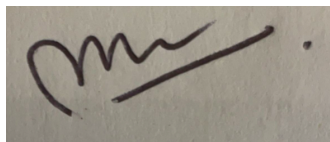


Submitted to - Prof. Amit Dua

Anti Plagiarism Statement

This is to declare that we, the students of group 50 have completed this project on our own and without external help or contribution from a third party. All the codes and design patterns have been written by us and nothing has been copied from any other source. We understand that copying someone else's assignment is wrong and constitutes a form of plagiarism. We also declare that any help taken from the internet or other sources has been strictly for the purpose of achieving conceptual clarity and/or resolving any errors that we encountered while creating this project. We declare that we have not plagiarized from any source and all the work is our own original creation.

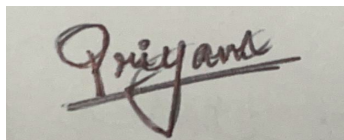
Yash Mundada

A handwritten signature in dark ink, appearing to be 'Yash Mundada', written on a light-colored surface.

Aditya Khandelwal

A handwritten signature in dark ink, appearing to be 'Aditya Khandelwal', written on a light-colored surface.

Priyam Verma

A handwritten signature in dark ink, appearing to be 'Priyam Verma', written on a light-colored surface.

Amarpal Singh

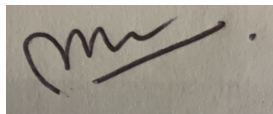
A handwritten signature in dark ink, appearing to be 'Amarpal Singh', written on a light-colored surface.

Contribution Table

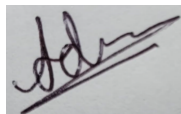
Member Contribution

Yash Mundada	Created the final report and the login page and the implemented database(file handling) for login. Researched socket multithreading.
Aditya Khandelwal	Created the Grades/ CGPA Calculator, Task Manager, Productivity Tracker and Timetable.Researched socket multithreading.
Priyam Verma	Created activity map, created the notepad feature, worked on loading page and login page and enhanced overall GUI. Researched Java Swing Implementation and JAR files.
Amarpal Singh	Collaboration (Multithreading)

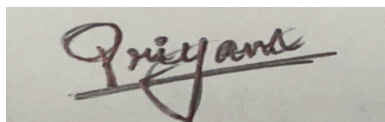
Yash Mundada



Aditya Khandelwal

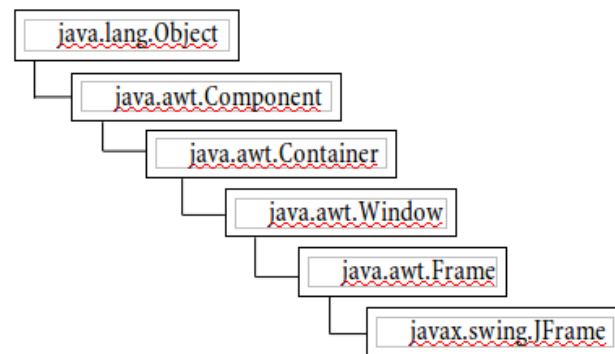


Priyam Verma



Design Patterns

This project primarily uses the Factory Pattern, wherein we have the JFrame class that is the basis of all the other classes. To create all the other classes, inheritance of the JFrame is used. It helps in reusing previously written code and makes the code more readable.



Chain of Responsibility pattern is also used wherein the user's choices decides which objects are created and executed. If the login button is used, then loginPage class whereas if the register button is used then it successively calls signUpPage class and then you can further call a page of registerCourses.

Singleton pattern is also utilized wherein instances of the error class are created and used throughout the project- eg at the login page with a dynamic text button that changes according to the error message.

Strategy pattern is also utilized wherein different objects are called depending on the user input- this includes all the different scenes. Can be best seen to be implemented at the menu page.

Analysis w.r.t SOLID Principles

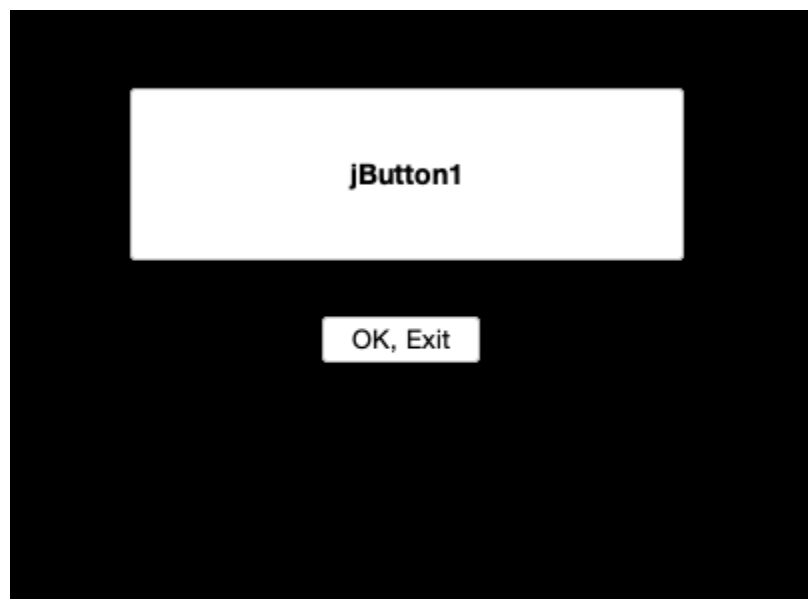
1. **Single Responsibility Principle:** The idea behind the SRP is that every class, module, or function in a program should have one responsibility/purpose in a program.

- In our opinion this has been implemented very well in our project. Each class corresponds to a screen/ window which is mainly responsible for only one function. For example the sticky notes package corresponds only to the notes that are being created. The separate user directory is also created with a subdirectory for each user whenever they sign up. This can similarly be demonstrated for all the remaining classes. CGPA calculator, weekly schedule manager classes correspond to one responsibility only.

2. **Open-Closed Principle:** Objects should be open for extension, but closed for modification.

- The Error class can be extended to add more types of errors on different pages as the basic structure is the same. The methods `setText()` would still work and only the button content would change. This is just a small example.

- To add more functionality to the CGPA calculator for example, we can add more methods to this class like different methods for finding data correlations/trends. Also we have the potential to add methods such as `createCharts` in productive hours without modification. etc.



3. Liskov Substitution Principle: Every class should be substitutable for a derived class without break in functionality.

- Unfortunately we were unable to implement this principle as we heavily relied on Java Frame we had the requirement of implementation of a UI.

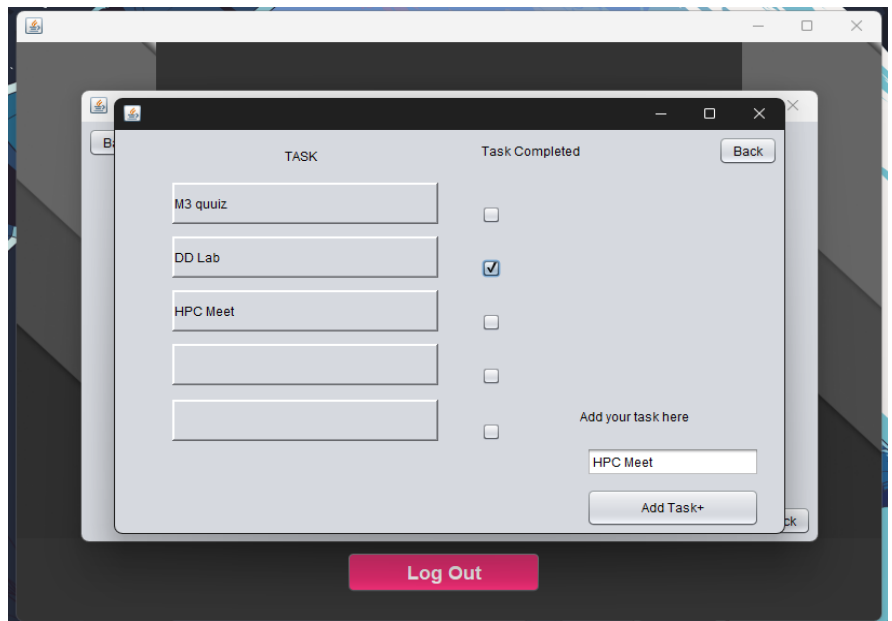
4. Interface Segregation Principle: A client should never be forced to implement an interface they don't use.

- Although we didn't make extensive use of interfaces we used them when creating the generics for the ArrayLists that were used for collecting data that the user inputted. If the client is not using this interface then they are not forced to implement it.

5. Dependency Inversion

Principle: Entities must depend on abstractions and not on concretions.

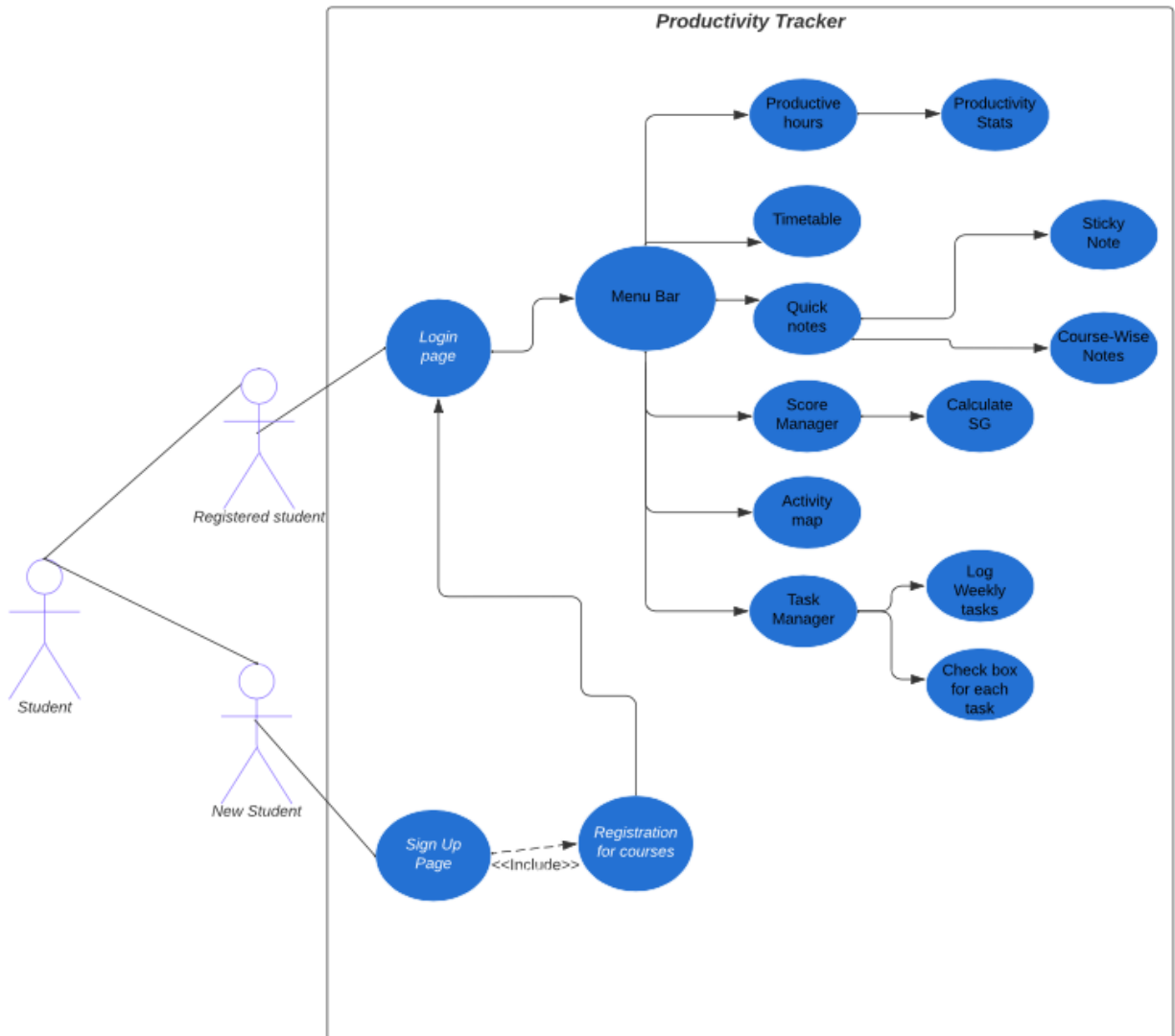
- We have made heavy use of this principle. Throughout our CGPA calculator and task setter we have used the method `setText()` for example. Attached is an image of the taskSetter. Here the text is added dynamically according to what is typed in the textbox and then added when clicked on `addTask`.



OOP Principles

- **Abstraction** : Abstraction is a process of hiding the implementation details and showing only functionality to the user. We have made sure to hide unnecessary details from the user by making use of **private methods** and constructors.
- **Inheritance** : Inheritance in Java is a concept that acquires the properties from one class to other classes. All the classes we have made use have inherited the javax.swing class and at various places have
- **Encapsulation** : Encapsulation in Java refers to integrating data (variables) and code (methods) into a single unit. We have made extensive use of **objects** and **classes** to combine fields and methods as a single entity.
- **Exception Handling** : Java provides a robust and object-oriented way to handle exception scenarios known as Java Exception Handling. We have made extensive use of Extensive use of Exception Handling in our code using the **throws keyword** and the **try-catch-finally** blocks. Exception handling has been taken care of in BufferedWriter, BufferedReader, PrintWriter, PrintReader, FileWriter and FileReader classes.

Use Case Diagram

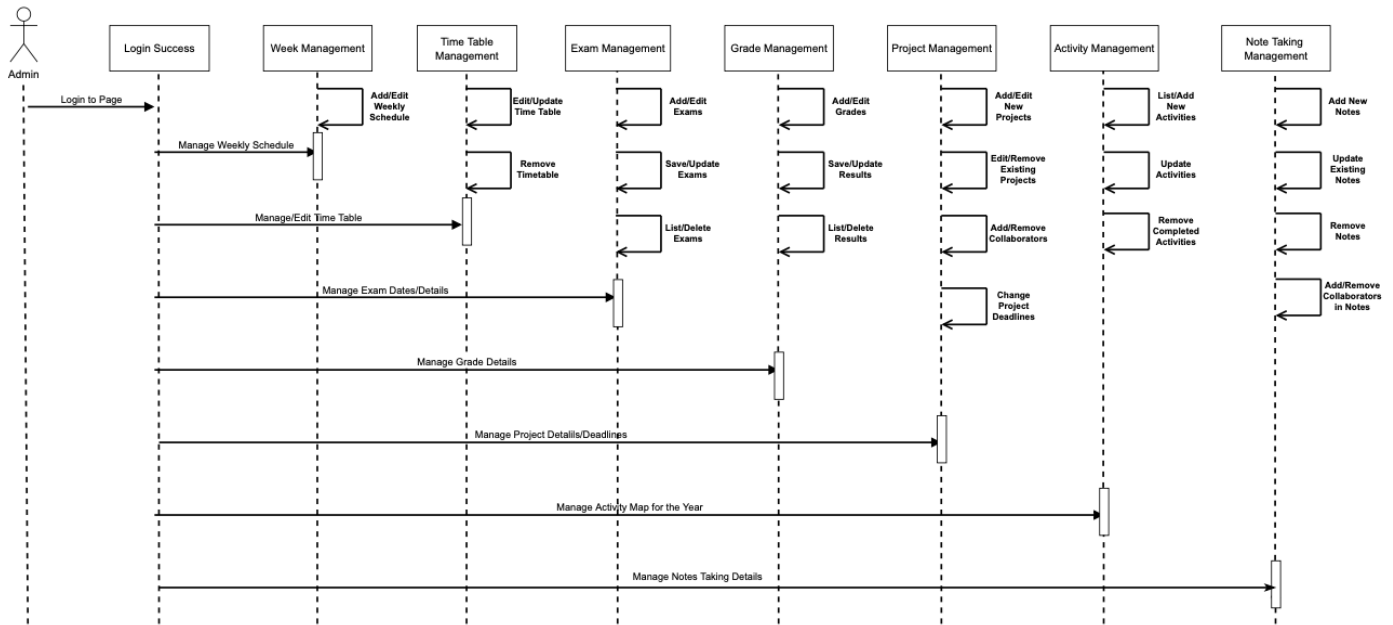


Sequence Diagram



Since the above diagram is not clear enough, it can be referred to in the image attached in the .zip file shared with you

Class Diagram



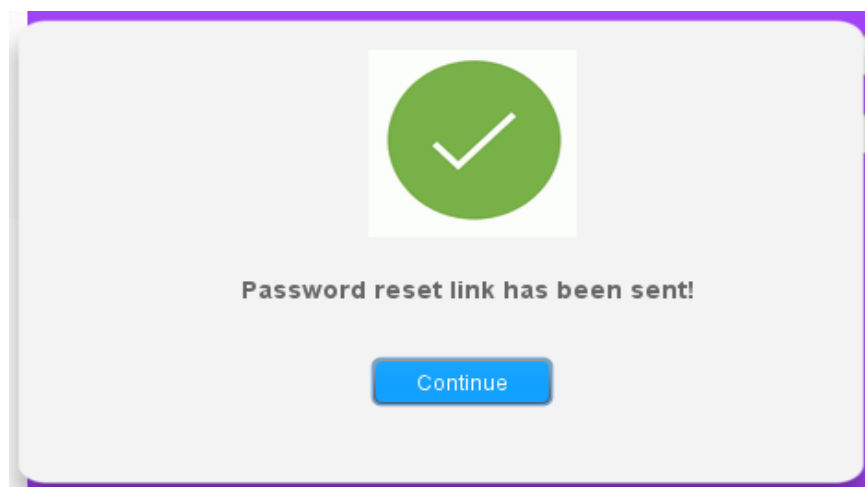
All UML Diagrams have been attached in the .zip file for better clarity.

Bonus for Extra Effort

Text files used for storing the data of the users who are registered on the application. This is a local database.

GUI is excellent- we had to learn JavaFX independently before deciding to transfer to Java **Swing** (which again we had to learn independently) - our loading page and animations at the start with the login page (including the “**forgot password**” and “**show password**” effects). This took a huge amount of effort and we would appreciate it if we were awarded for this.

GUI was a huge and integral part of this project. Using JavaSwing created many limitations in our work as we were not familiar with the usage of Swing.



Any assumptions

CGPA Calculator - average gets a C grade (av \pm 5% C) (av+5-10%B-) (10-20%B) (20-25%A-) (av+25% gets an A).

Refer to 'Refer.pdf' file in '/50_2422_0001_1776_0336/' for application working and understanding major features.

Videos

https://drive.google.com/drive/u/0/folders/1QYpj4El-ginJmddbeWg-33nZOT_PONAz