Object Oriented Programming
Lab 11 Designing, Implementation and UML

April 17, 2023

1 Lab Goal

The purpose of this lab is to look at the problems that were given to you in the midterm and try to solve them with the added experience that you have gained since then. Try to apply the SOLID principles where ever you can. For both the questions, you will need to make the UML first and then try to code them out. Go to Canva and register, select whiteboard and draw the UML using the various shapes given to you to the left of the screen. You have the share option on top right from where you can download your diagram as a PNG file. Do upload the diagram with your submission.

You can access Java Documentation at this link.

2 Problems

We will read the problems carefully as these have been updated since the exam took place.

2.1 Problem 1 - Exam's First Deja vu

A company has following types of **Employees**: **Programmer** and **Tester**. Employee must be an **abstract** class which has an abstract method called **work()** that the classes Programmer and Tester will provide an implementation for.

The Employee class contains functionality common to the two types of employees in the company. For example, all employees have a **name** and **ID number**. Write the **constructor** and **get** and **set** methods. It also has an abstract method of **salary()** which calculates how the employees are paid. Programmers are paid on the hours they have worked while testers are paid on the number of lines they tested.

A given programmer can write a certain number of lines of code per day, captured by an instance variable **linesOfCodePerDay**. A programmer's work method returns the number of lines of code he/she wrote on a given day, which is a random number between 500 and 1500 of linesOfCodePerDay (some days are not as good as others, and some tasks are harder/easier than others). The salary is calculated by the rate per lines of code.

The programmer class has instance variables of **rate** and time. Rate is amount paid per hour while time is amount of minutes it takes programmer to write a single line of code. Override the salary method and calculate what the programmer made in a day.

A given Tester can test a given number of lines of code per day, captured by an instance variable **linesOfCode-PerDay**. A tester's work method returns the number of lines of code he/she tested that day, which is a random number between 150 and 250 of lines Of code tested per day. Override the salary method and calculate the amount tester made in a day.

Write the class **Project** that models a software development project. It should have the following attributes:

- linesOfCode: An estimate of the number of lines of code the project will require (that's never known ahead of time, but let's just assume that).
- linesOfCodeWritten: The number of lines of code that have been written by the programmers working on it.
- linesOfCodeTested: The number of lines of code that have been tested so far.
- duration: How many days managers have given for completion of the project.

- employees: An ArrayList that stores the employees that are associated with the project.

The Project class should have the following methods:

- Constructor: should take in the number of lines of code for the project and required duration.
- addEmployee(): adds an employee to the project.
- int doProject(): do the project. Each day, as long as the project is not complete, it should call each employee's work method. It must return the number of days the project took beyond the allotted number of days. If a project is running behind (or you estimate that it is running behind), put more testers or programmers on the job.

The main method of this class should create a new instance of Project, add a few programmers and testers, run the doProject() method, and report how long it took and whether it was completed on time.

Make sure the constructors for the Programmer and Tester classes use the constructor of the super class. Also, write toString() methods that use the toString() method of Employee.

2.2 Problem 2 - Exam's second Deja vu

You have been assigned the task of designing and implementing a library management system for a local library. The system must meet the following requirements:

1. Users should be able to search for books based on their title, author, or subject.
2. Users should be able to view book details such as its availability, location, and checkout history.
3. Users should be able to check out, return, and renew books.
4. Librarians should be able to manage the library's inventory, which includes adding and removing books, updating book information, and managing user accounts.
5. Library's inventory system must be capable of generating reports on library usage, such as circulation statistics (how many books were borrowed and returned and in what category), popular books, and overdue items.

The main() method should be created to test the system and perform necessary operations, such as adding books, adding customers, placing orders, generating user usage and book availability statistics. Again, it will be best that you create a UML and then write the code.