Registration System

Software Design Patterns

CPIT-252

Students:

Yousif Bogari - 2136374.

Abdulaziz saddig jastanieh - 2135813.

Instructor:

Dr. Rizwan Qureshi.

Contents

Introduction:

At the start of every semester, in the first few weeks, students must always go through the process of registering courses. This process Is anxiety inducing and stressful, as students always fear missing out on good Sections or Good lecture times. So students Rush to the faculty, to go to the Student affairs who are responsible for registering courses for students, and to the waiting line at the start of every semester. And every student that goes in starts asking the same questions: “what are the sections available?” , “what are their times?” “What Are The Good College Free Courses?” and so , This process can take so much time, and the students' precious time is being wasted. And Doctors have to delay the start of the lectures as not all the students have registered for the course, which causes a delay in the Schedule.

Problem Definition:

The problem is that there isn’t a Simple, Good and easy to use Registration system that the student can access anytime and anywhere in order to register a course, the systems that are being used by the students to register the courses are not suitable for Mobile use. And so, when the students try to use them from their mobile phones, they feel clunky and hard to use. That's because they are made with the intent that they will be used from a laptop or a personal computer, and not a mobile phone. And that is not the only issue, Using them from your mobile phone can take a lot of your phone's hardware resources E.g: CPU , RAM.

Suggested solution:

We will Design and build a Simple Registration system with Our focus being on Mobile use and User experience, as well as Reducing the usage of mobile phone resources, so that students can register courses from anywhere at any time with ease.

Main Functionalities:

1. The student can Browse through the list of courses Available for Registration.
2. The student can Register a Course at a certain Section.
3. The student can see the list of Registered Courses.
4. The student can Delete a Registered Course.

Design Patterns:

In our program we used 3 Different Design patterns.

Creational:

Builder pattern: we used this pattern to make creating any complex object in our program a much easier task, and a simpler one E.g.: creating Courses and Students objects.

```
public class courseDirector {
    private final courseBuilder course;

    public courseDirector(courseBuilder builder) {
        this.course = builder;
    }

    public void makeCourse() {
        this.course.buildCourse();
    }
    public Course getCourse() {
        return this.course.getCourse();
    }
}
```

```
public interface courseBuilder {
    public void buildCourse();
    public Course getCourse();
}
```

Structural:

Proxy pattern: we used this to create a placeholder object that control the acess to other object. We intercepted the methods as a real object. This gave us the flexibility of modifying behavior of the system without modifying the real object implementation or the service. As our system provide multiple services. Proxy helps us to use them fluently.

```
public interface Service {
    public String Operation();
    public String Operation(Student student, Section section, Course course);
    //this method will be implemented by each service accordingly.
}
```

```
public class serviceProxy implements Service {

    Student student;
    Section section;
    Course course;
    Service RealService;

    public serviceProxy(Student student, Section section, Course course,Service RealService) {
        this.student = student;
        this.section = section;
        this.course = course;
        this.RealService = RealService;
    }
public String Operation(Student student, Section section, Course course) {
   return RealService.Operation(student, section, course);
   //this method will call the operation that the real service provides. e.g.: registering a
course
}

public String Operation(){
    return Operation(student, section, course);
    //this is the method we will use to call any operation of any service
}
```

Behavioral:

State pattern: we used this pattern to execute different behavior depending on the state of the object E.g.: the section object can behave differently depending on its current state, The full State or Not Full State.

```
public interface SectionState {
    public boolean Check();
}

public class notFullState implements SectionState {
    @Override
    public boolean Check() {
        return true;
    }
}
```

Observer pattern: Observer pattern has been used to enhance the concept of applying MVC pattern. In our system, the observer is responsible for the interaction between the controller and the model. The model notified the controller about the status of the required function that comes from the view.

```
public interface Service {
    public String Operation();
    public String Operation(Student student, Section section, Course course);
    //this method will be implemented by each service accordingly.
}
```

Architectural pattern:

Model View Controller (MVC):

We used this pattern because it allows us to separate the application logic from the business logic. The Model contains the Business logic of our system, and the view contains the application logic (the processing logic), and the controller works as the manager and the middleman between.

This allows us to have many different views but only one model, this gives us the ability to easily show different information and to provide different services according to who is the user and what View they are using. E.g.: the student can browse through the available courses and register for a course or delete one. while an administrator can add students or courses and see information about the students and what courses they registered. This can easily be implemented in the future as We only just need to Add a different View.

```
public class Model {
    Student LoggedStudent;
    private List<Observer> observers = new ArrayList<>();
    private void notifyObservers(String u) {
        for (Observer observer : observers) {
            observer.update(u);
        }
    }

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public Model(Student LoggedStudent) {
        this.LoggedStudent = LoggedStudent;
    }

public class StudentView {

    public void printBasedonModel(String s) {
        System.out.println(s);
    }
public class Controller implements Observer{
    Student LoggedStudent;
    private Model Model;
    private StudentView view;
    public void update(String updateType) {
        System.out.println("Required operation in progress..");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {e.getStackTrace();
        }
    }
}
```

Class Diagrams:

Code:

The Program and codes will be Available at **GitHub**:

You can check it by **HERE**.

Contributors:

- **Yousef Bogari:**
  - ❖ Implementing State & Builder pattern.
  - ❖ Initializing Data.
  - ❖ Proxy pattern.
  - ❖ Implementing backend methods logic.

- **Abdulaziz Jastanieh:**
  - ❖ Designed the system
  - ❖ Initialize proxy pattern
  - ❖ Implemented MVC pattern
  - ❖ Data coordinator. Ensuring data is correctly transmitted and shared between the processes.
  - ❖ Integrate the patterns.

Expected Results:

Students will be able to register courses easily without the need to visit the students affairs and waste so much time, which will result in students being Satisfied and happy, and Doctors can start giving lectures early and be on schedule.