Name: Manvi M Shetty

Github: oswardshetty/Assignment-2 (github.com)

Exercise 2:

The objective is to develop a simple console application for managing a virtual classroom. This application will facilitate various functionalities, including creating classrooms, enrolling students, scheduling assignments, and submitting assignments. Additionally, it will incorporate design patterns to enhance its architecture and maintainability.

Key Features

1. **Classroom Management**:
   - Users will have the ability to create and list virtual classrooms, allowing for organized management of different educational groups.
2. **Student Enrollment**:
   - The application will enable the enrollment of students into specific classrooms, ensuring that each classroom can maintain its own roster.
3. **Assignment Management**:
   - Users can schedule assignments for classrooms and handle submission processes for students, thereby facilitating academic accountability.
4. **Undo Actions**:
   - The system will support the ability to undo the last action taken, enhancing user control over classroom management operations.

Design Patterns

1. Observer Pattern

The Observer pattern will be utilized to notify students when a new assignment is added. This will ensure that all enrolled students receive real-time updates regarding their assignments, fostering engagement and accountability.

2. Command Pattern

The Command pattern will be employed to implement actions such as adding and removing assignments as discrete command objects. This allows for the execution and potential undoing of these actions, providing a flexible and organized approach to managing classroom tasks.

3. Factory Method Pattern

The Factory Method pattern will facilitate the creation of assignment instances based on their type, such as Homework, Project, or Quiz. This approach encapsulates the instantiation logic within a factory class, promoting code reusability and separation of concerns.

Implementation Steps

1. Create Interfaces and Classes

- **Define an Assignment Interface**: Establish a common contract for all assignment types, ensuring they implement essential methods.
- **Implement Classes**: Develop concrete classes like Homework, Project, and Quiz that adhere to the Assignment interface.
- **Create a Classroom Class**: This class will manage the list of students and assignments, handling their respective functionalities.

2. Implement Observer Pattern

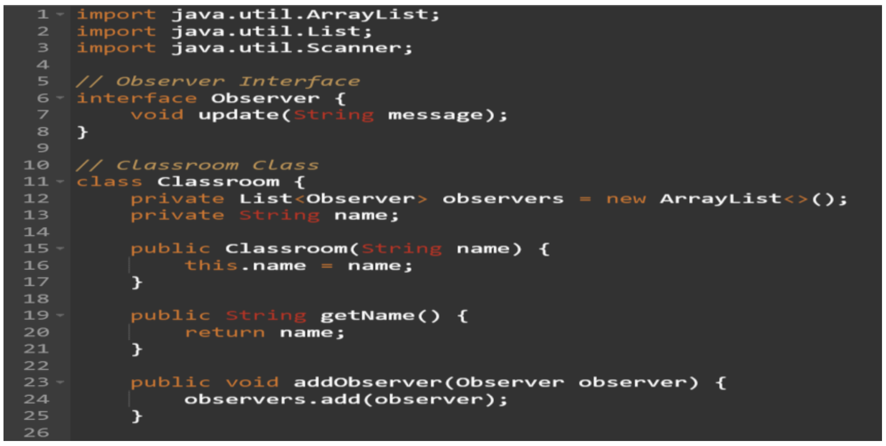
- **Create an Observer Interface**: Define an interface for observers (students) to receive updates.
- **Implement a Student Class**: Create a Student class that implements the Observer interface, allowing it to receive notifications about new assignments.

3. Implement Command Pattern

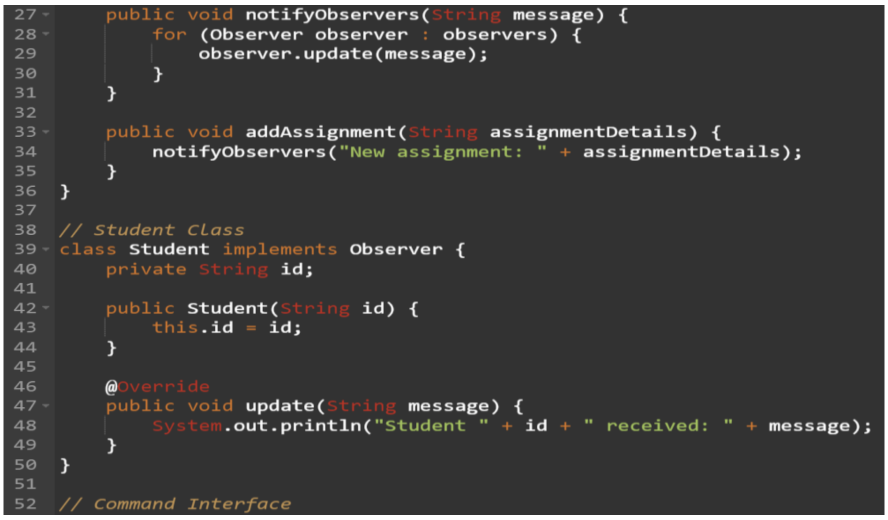
- **Define a Command Interface**: Create an interface that outlines the methods for executing and undoing commands.
- **Implement Commands**: Develop specific command classes for adding and removing assignments, allowing these actions to be treated as first-class objects.

4. Use Factory Method

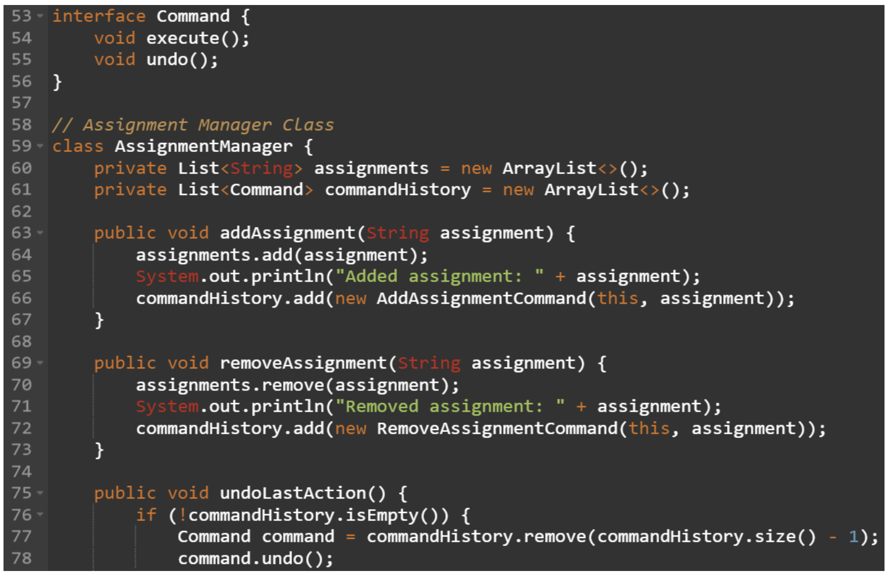
- **Create an AssignmentFactory**: Implement a factory class that generates assignment instances based on the type specified by the user. This encapsulates the logic for instantiating different assignment types, adhering to the principles of the Factory Method pattern.



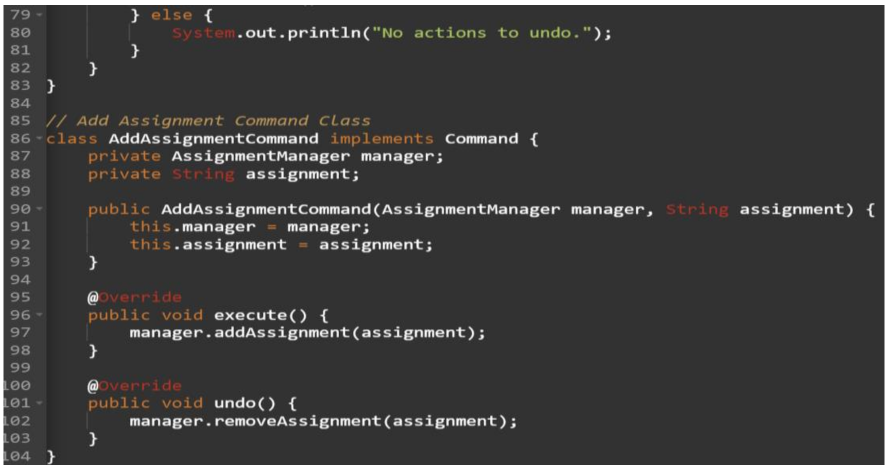
```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

// Observer Interface
interface Observer {
    void update(String message);
}

// Classroom Class
class Classroom {
    private List<Observer> observers = new ArrayList<>();
    private String name;

    public Classroom(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

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

```



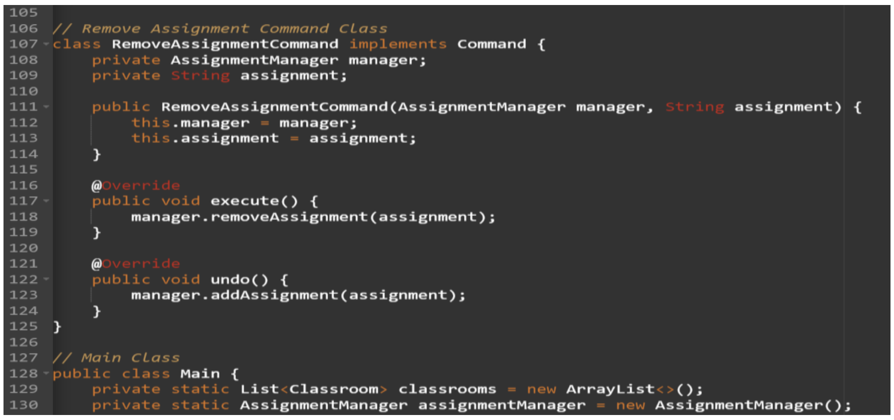
```java
    public void notifyObservers(String message) {
        for (Observer observer : observers) {
            observer.update(message);
        }
    }

    public void addAssignment(String assignmentDetails) {
        notifyObservers("New assignment: " + assignmentDetails);
    }
}

// Student Class
class Student implements Observer {
    private String id;

    public Student(String id) {
        this.id = id;
    }

    @Override
    public void update(String message) {
        System.out.println("Student " + id + " received: " + message);
    }
}

// Command Interface
```



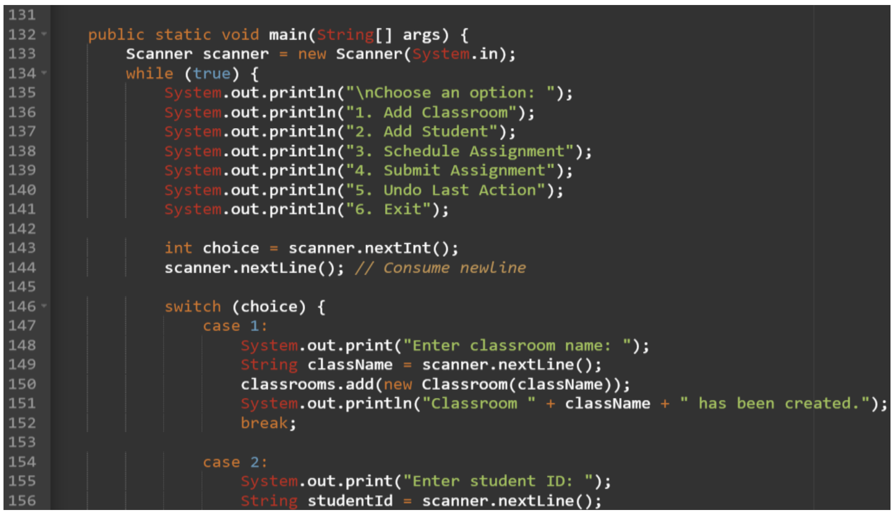
```java
interface Command {
    void execute();
    void undo();
}

// Assignment Manager Class
class AssignmentManager {
    private List<String> assignments = new ArrayList<>();
    private List<Command> commandHistory = new ArrayList<>();

    public void addAssignment(String assignment) {
        assignments.add(assignment);
        System.out.println("Added assignment: " + assignment);
        commandHistory.add(new AddAssignmentCommand(this, assignment));
    }

    public void removeAssignment(String assignment) {
        assignments.remove(assignment);
        System.out.println("Removed assignment: " + assignment);
        commandHistory.add(new RemoveAssignmentCommand(this, assignment));
    }

    public void undoLastAction() {
        if (!commandHistory.isEmpty()) {
            Command command = commandHistory.remove(commandHistory.size() - 1);
            command.undo();
```



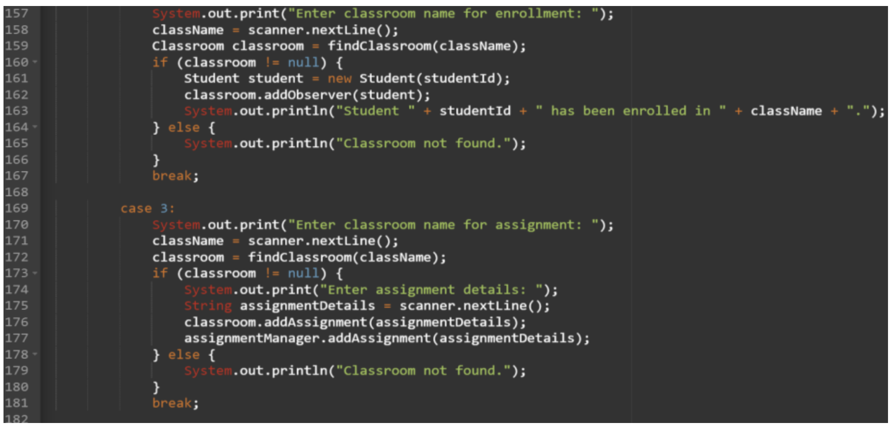
```java
        } else {
            System.out.println("No actions to undo.");
        }
    }
}

// Add Assignment Command Class
class AddAssignmentCommand implements Command {
    private AssignmentManager manager;
    private String assignment;

    public AddAssignmentCommand(AssignmentManager manager, String assignment) {
        this.manager = manager;
        this.assignment = assignment;
    }

    @Override
    public void execute() {
        manager.addAssignment(assignment);
    }

    @Override
    public void undo() {
        manager.removeAssignment(assignment);
    }
}
```



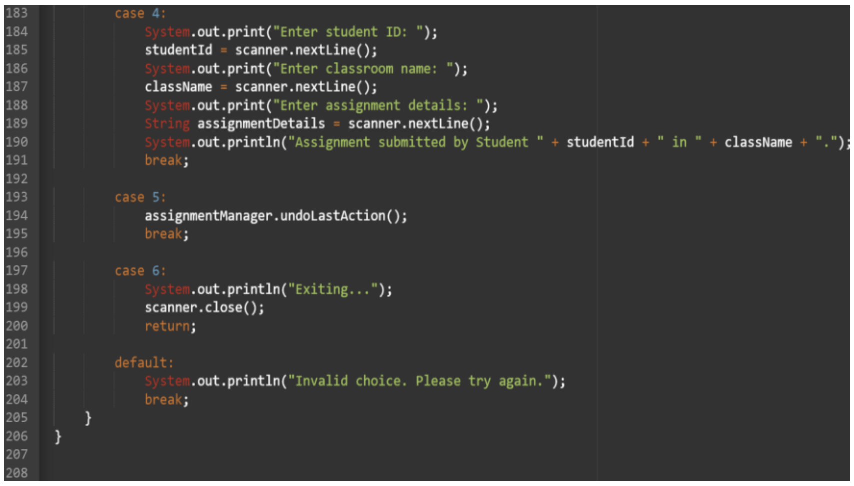
```java

// Remove Assignment Command Class
class RemoveAssignmentCommand implements Command {
    private AssignmentManager manager;
    private String assignment;

    public RemoveAssignmentCommand(AssignmentManager manager, String assignment) {
        this.manager = manager;
        this.assignment = assignment;
    }

    @Override
    public void execute() {
        manager.removeAssignment(assignment);
    }

    @Override
    public void undo() {
        manager.addAssignment(assignment);
    }
}

// Main Class
public class Main {
    private static List<Classroom> classrooms = new ArrayList<>();
    private static AssignmentManager assignmentManager = new AssignmentManager();
```



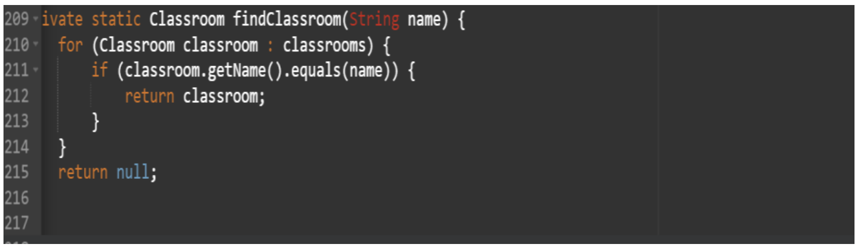
```java

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            System.out.println("\nChoose an option: ");
            System.out.println("1. Add Classroom");
            System.out.println("2. Add Student");
            System.out.println("3. Schedule Assignment");
            System.out.println("4. Submit Assignment");
            System.out.println("5. Undo Last Action");
            System.out.println("6. Exit");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter classroom name: ");
                    String className = scanner.nextLine();
                    classrooms.add(new Classroom(className));
                    System.out.println("Classroom " + className + " has been created.");
                    break;

                case 2:
                    System.out.print("Enter student ID: ");
                    String studentId = scanner.nextLine();
```



```java
                    System.out.print("Enter classroom name for enrollment: ");
                    className = scanner.nextLine();
                    Classroom classroom = findClassroom(className);
                    if (classroom != null) {
                        Student student = new Student(studentId);
                        classroom.addObserver(student);
                        System.out.println("Student " + studentId + " has been enrolled in " + className + ".");
                    } else {
                        System.out.println("Classroom not found.");
                    }
                    break;

                case 3:
                    System.out.print("Enter classroom name for assignment: ");
                    className = scanner.nextLine();
                    classroom = findClassroom(className);
                    if (classroom != null) {
                        System.out.print("Enter assignment details: ");
                        String assignmentDetails = scanner.nextLine();
                        classroom.addAssignment(assignmentDetails);
                        assignmentManager.addAssignment(assignmentDetails);
                    } else {
                        System.out.println("Classroom not found.");
                    }
                    break;
```

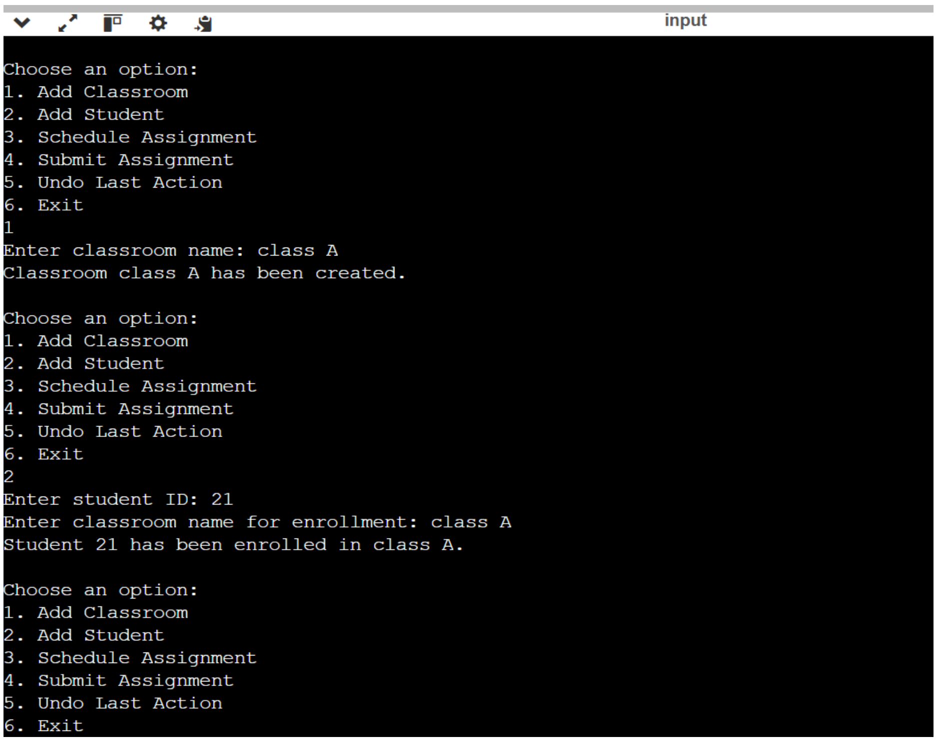


```
            case 4:
                System.out.print("Enter student ID: ");
                studentId = scanner.nextLine();
                System.out.print("Enter classroom name: ");
                className = scanner.nextLine();
                System.out.print("Enter assignment details: ");
                String assignmentDetails = scanner.nextLine();
                System.out.println("Assignment submitted by Student " + studentId + " in " + className + ".");
                break;

            case 5:
                assignmentManager.undoLastAction();
                break;

            case 6:
                System.out.println("Exiting...");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice. Please try again.");
                break;
        }
    }

```



```
ivate static Classroom findClassroom(String name) {
  for (Classroom classroom : classrooms) {
      if (classroom.getName().equals(name)) {
          return classroom;
      }
  }
  return null;

```

Output:



```
Choose an option:
1. Add Classroom
2. Add Student
3. Schedule Assignment
4. Submit Assignment
5. Undo Last Action
6. Exit
1
Enter classroom name: class A
Classroom class A has been created.

Choose an option:
1. Add Classroom
2. Add Student
3. Schedule Assignment
4. Submit Assignment
5. Undo Last Action
6. Exit
2
Enter student ID: 21
Enter classroom name for enrollment: class A
Student 21 has been enrolled in class A.

Choose an option:
1. Add Classroom
2. Add Student
3. Schedule Assignment
4. Submit Assignment
5. Undo Last Action
6. Exit
```