Student Management System Project

Team names

- Nour mostafa metwaly 21-01370
- Seif taher Mohamed 21-01582
- Nada emad sabry 21-01332
- Sara Ibrahim bakhet 21-01575
- Salma Mohamed saad 21-01533
- Shrouk khaiy soliman 2101635

Student Management System Project

- **Purpose**: to manage student records
- **Scope**: Managing students, tracking records, and adding/removing data.
- **Summary** : The project is divided into a group of classes, including the gui part, including models and data manipulation

System design to our system:

- **Database**
- **Factories**
- **Models**
- **Patterns**
- **Singletons**
- **Home (GUI and run point)**

Database Class

-This database class is designed to manage the connection to a SQL Server database

-The class follows the Singleton Pattern, meaning it ensures that only one instance of the Connection object is created and used throughout the application.

- The class contains:

- A private static variable connection to hold the database connection.
- A getConnection() method that checks if a connection already exists or is open; if not, it creates a new connection using the provided database URL, username, and password.
- A closeConnection() method to cleanly close the connection when it's no longer needed.

This class helps centralize database connection management, ensuring efficient and safe access to the database throughout the application.

Factories Class

- Contains two class:

1- CourseFactory: is a factory class that creates different types of Course objects based on the provided type and name parameters.

- Contains Method: The createCourse(String type, String name) method uses a switch statement to determine the type of course

2- StudentFactory: is a factory class responsible for creating different types of Student objects based on the provided type. It contains a single static method, createStudent(String type), which accepts a type parameter and returns a corresponding Student object.

Models

- Course class : The Course class is an **abstract class** that serves as a blueprint for different types of courses. It defines common properties and behaviors that all courses should have.

- Since Course is abstract, it cannot be instantiated directly. It is meant to be extended by concrete subclasses (like CoreCourse, ElectiveCourse), which can provide specific implementations for course types.

- It contains three sub-types of courses (CoreCourse, ElectiveCourse , LabCourse): These classes extends the Course class and represents a specific type of course,

- **Constructor**: inherits the constructor of the Course class and calls the superclass constructor (super(name, type)) to initialize the course with a name and type.
- **Method Override**: getDetails(String name) method from the Course class. In its implementation, it returns a string indicating that the course is a **Core Course** followed by the course name.

-Student class: The **Student** class is an **abstract class** that serves as a blueprint for different types of students. It defines common properties and behaviors that all students that extends from it.

- It contains three sub-types of courses

(UndergraduateStudent, GraduateStudent, PartTimeStudent): These classes extends the Student class and represents a specific type of student,

Method Override: It overrides the getDetails() method inherited from the Student class. In its implementation, it returns the string for type of student.

Patterns Classes

- Contains (Command, GradeAccess, Observer) these interfaces: As an interface, does not provide an implementation but specifies that any class that implements it must provide a concrete version of these classes methods. This allows different classes to define how student grades are fetched or updated or executed, enabling flexibility and abstraction in grade management systems.

- GradeProxy class:

- The GradeProxy class implements the GradeAccess interface and acts as a proxy for accessing a student's grade from the GradeProcessingSystem.

- **Constructor**: The constructor initializes the GradeProxy by obtaining an instance of the GradeProcessingSystem singleton through GradeProcessingSystem.getInstance().

- **Method Override**: The fetchGrade(String student) method is overridden from the GradeAccess interface. It:

- The GradeProxy class provides a controlled way to access the GradeProcessingSystem, adding a layer of logging and handling potential missing data. This design follows the Proxy Pattern, where the proxy controls access to the real object (GradeProcessingSystem).

- RegisterCourseCommand class :

The RegisterCourseCommand class implements the Command interface and encapsulates the action of registering a course within the CourseRegistrationSystem.

-Attributes:

- courseSystem: A reference to the CourseRegistrationSystem singleton, responsible for handling course registration.

- course: The Course object to be registered.

-Constructor: The constructor takes a CourseRegistrationSystem instance and a Course object, initializing the class with these dependencies.

-Method Override: The execute() method is overridden from the Command interface. It:

- Registers the specified course using the registerCourse() method of courseSystem.

- Prints a message confirming that the course has been successfully registered, including the course details.

This class follows the Command Pattern, which encapsulates a request (registering a course) as an object, allowing for parameterization, queuing, and executing the command at a later time. The RegisterCourseCommand serves as the concrete implementation of a command to register a course in the system.

- Subject class:

The Subject class is an abstract class that forms the core of the Observer Pattern, allowing multiple observers to subscribe and react to changes in the subject's state.

- **Attributes:**

  - observers: A list of Observer objects that are subscribed to the subject. These observers will be notified when the subject's state changes.

- **Methods:**

  - addObserver(Observer observer): Adds an observer to the list, allowing it to receive notifications.

  - removeObserver(Observer observer): Removes an observer from the list, preventing it from receiving future notifications.

  - notifyObservers(String message): Notifies all registered observers by calling their update() method with a given message, informing them of the subject's state change.

Singletons

- CourseRegistrationSystem:

- The CourseRegistrationSystem class is a singleton that manages the registration and unregistration of courses. It extends the Subject class, meaning it can notify observers (such as a user interface or logging system) when a course is registered or unregistered.

- **Singleton Pattern**: The class follows the Singleton pattern, ensuring only one instance of CourseRegistrationSystem exists. The getInstance() method provides access to this instance in a thread-safe manner.

- Course Management:

- **Register Course**: The registerCourse(Course course) method adds a course to the list of registered courses if it isn't already registered. It also notifies observers about the new registration.
- **Unregister Course**: The unregisterCourse(String courseName) method removes a course by name from the registered list and notifies observers.
- **Check for Duplicates**: The isCourseRegistered(Course course) method checks if a course has already been registered to prevent duplicates.

- **Observer Pattern**: By extending Subject, the system can notify all registered observers whenever a course is registered or unregistered.

- **Course List Management**:

- The printRegisteredCourses() method prints the list of all registered courses.
- The getRegisteredCourses() method returns the list of registered courses.

2- GradeProcessingSystem:

This class is a singleton that manages the storage and retrieval of student grades. It ensures only one instance of the class exists throughout the application, providing a centralized system for grade management.

- **Singleton Pattern**: The class follows the Singleton pattern, guaranteeing a single instance of GradeProcessingSystem. The getInstance() method provides access to this instance in a thread-safe manner.
- **Grade Storage**:

  - It uses a Map<String, String> (grades) to store student names (as keys) and their corresponding grades (as values).

- **Methods**:

  - addGrade(String student, String grade): Adds or updates a student's grade in the grades map.

  - getGrade(String student): Retrieves the grade of a specific student by their name.