جامعة مصر للمعلوماتية
EGYPT UNIVERSITY
OF INFORMATICS

Egypt University of Informatics

Computer and Information Systems

Software Engineering Course

Technical Report

Deliverable 4 Behavioral Design Patterns

Submitted by:

Nada Ashraf 22-101043

Aly Zaki 22-101096

Ahmed Waleed 22-101058

Omar Bayoumi 22-101022

Submission Date: 15th May 2025

Overview

This report focuses on implementing the behavioral design patterns into our EduCertify system. It's a learning and certification system that enables learners worldwide to be able to enroll in courses and earn certification upon the completion of the courses. This report will tackle different behavioral design patterns in which each design pattern will communicate a specific problem and will be able to solve it in a maintainable and extensible manner. These solutions achieve the Single Responsibility and Open closed principles.

What are Behavioral Design Patterns?

Behavioral Design Patterns are concerned with the communication between objects and the interactions among them. These patterns are essential for creating flexible and scalable systems, where objects can communicate and cooperate with each other without tightly coupling their responsibilities.

Behavioral patterns help define how objects interact with one another and allow for more efficient problem-solving by promoting better communication, control flow, and responsibility distribution. They are particularly beneficial in systems where flexibility, maintainability, and easy extensibility are crucial. By implementing behavioral patterns, we can make our systems more adaptable to change, enabling them to respond to future requirements without major modifications. It follows multiple principles of SOLID Principles like Single Responsibility, Open-Closed, and much more.

Template Design Patterns

1. Certificate Generation Workflow

Intent:

- To define a standard abstract process of certificate generation (to validate, fetch data, and export format) while allowing different certificate types to override that specific formatting steps.

Problem:

- Different certificate types like Completion and Excellence follow the same overall steps but vary in some ways.
- Hardcoding all the variations will lead to code duplication and hard to maintain in the future.

Solution:

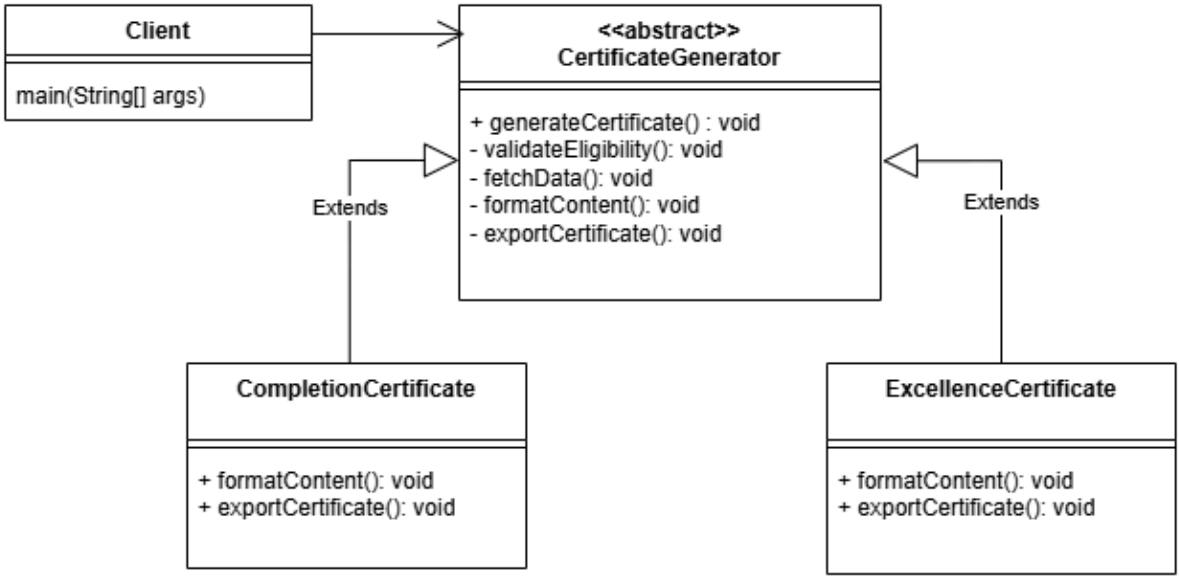
- Implementing a template method pattern by creating an abstract base class (Certificate Generator) that defines the template method which is called (generate Certificate) that includes all the steps of generating a certificate.
- Subclasses can override methods to reflect their own behavior.

Components:

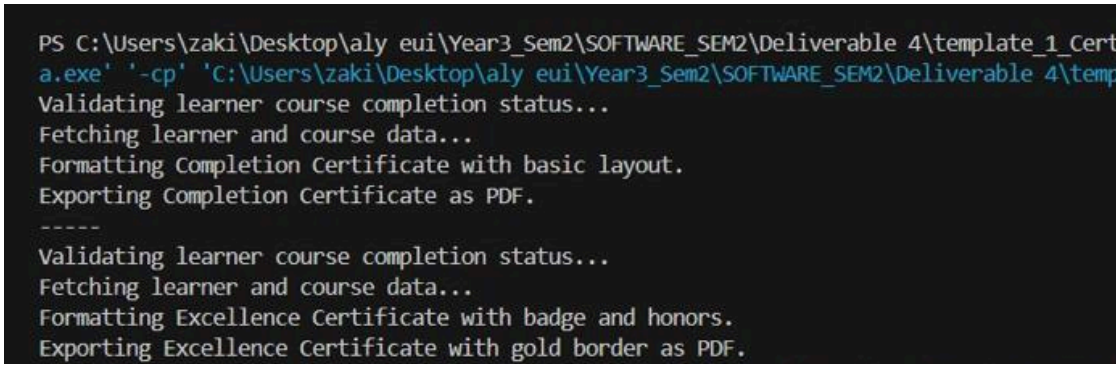
- *Abstract Class*: Certificate Generator

- *Concrete Classes*: Completion Certificate and Excellence Certificate
- *Hook Methods*: Format Content() and Export Certificate()
    - Implemented differently in each subclass (overridden)

UML



Running Code:



2. User Account Creation:

Intent:

- To define consistent steps for creating user accounts while allowing each user role (Learner, Instructor, Admin) to customize their permission setup during account creation.

Problem:

The core process of creating an account is the same for all user roles:

- Validate email
- Create profile
- Assign permissions
- Send a welcome email

However, the permissions step differs from each one to the other. Embedding all logic in one method leads to a less maintainable code.

Solution:

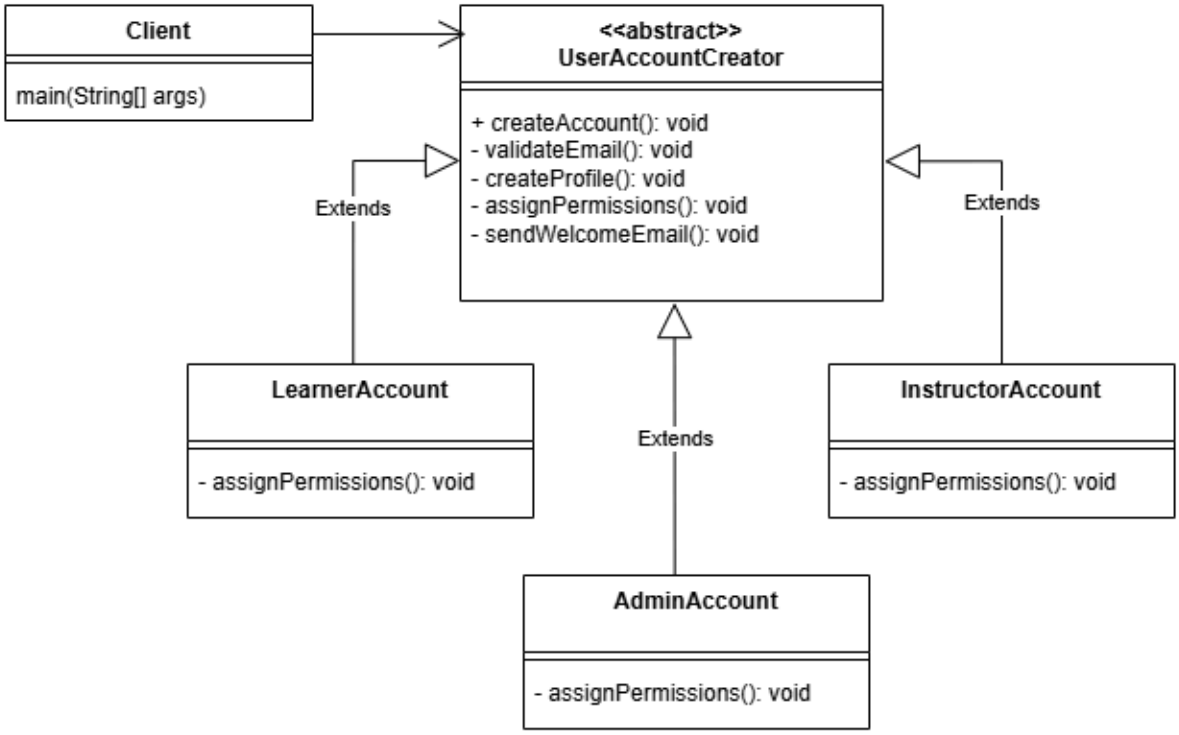
- Use the Template Method pattern with a User Account Creator base abstract class that defines the create Account() method with standard steps.
- Each subclass (Learner Account, Instructor Account, Admin Account) customizes the assignPermissions() step only.

Components:

Abstract Class: User Account Creator (Overall Structure)

Concrete Classes: Learner, Instructor, and Admin Accounts

UML:



Running Code:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  QUERY RESULTS (PREVIEW)

PS C:\Users\zaki\Desktop\aly eui\Year3_Sem2\SOFTWARE_SEM2\Deliverable 4\template_2_User
i\Desktop\aly eui\Year3_Sem2\SOFTWARE_SEM2\Deliverable 4\template_2_User Account Creati
Validating email format and uniqueness...
Creating user profile in database...
Assigning learner dashboard and course access.
Sending welcome email to new user...
-----
Validating email format and uniqueness...
Creating user profile in database...
Assigning instructor privileges and course creation tools.
Sending welcome email to new user...
-----
Validating email format and uniqueness...
Creating user profile in database...
Assigning full system access and admin tools.
Sending welcome email to new user...
```

Mediator Design Pattern

1. Learner Instructor Communication:

Intent:

- To manage all interactions between learners and instructors (like submissions, etc.) via single mediator object to maintain loose coupling.

Problem:

- Learners and instructors communicate constantly
- Learners submit assignments or ask for help.
- Instructors provide feedback or grades.
- Direct interaction creates tight coupling

Solution:

- Use a Mediator class that mediates between learners and instructors.
- Learners send messages or submissions to the mediator.
- The mediator gives the instructor the message.
- Instructor replies through the mediator.

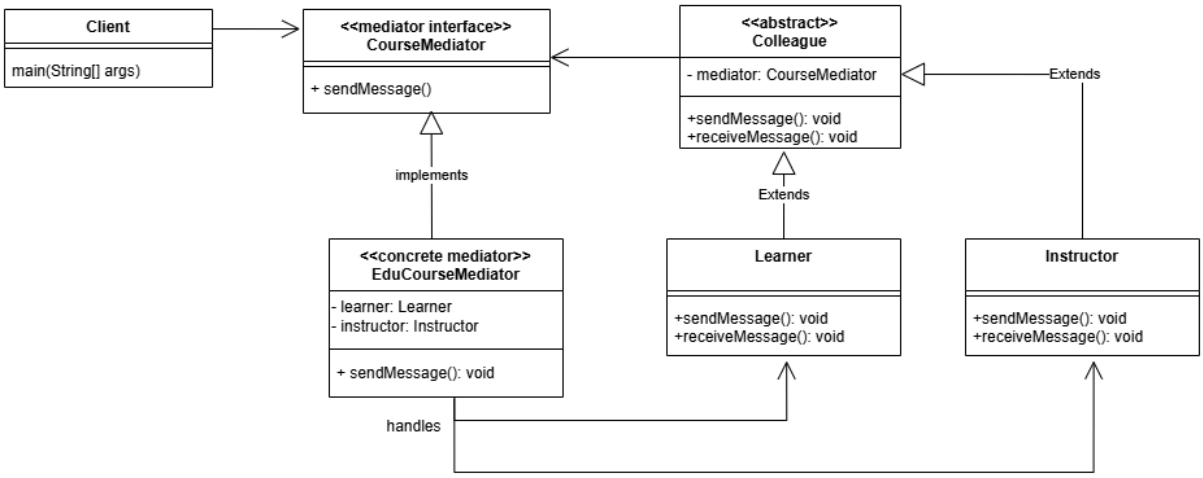
Components:

Mediator Interface: Course Mediator

Concrete Mediator: Edu Course Mediator

Colleagues: Learners and Instructors

UML:



Running Code:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  QUERY RESULTS (PREVIEW)

PS C:\Users\zaki\Desktop\aly eui\Year3_Sem2\SOFTWARE_SEM2\Deliverable 4\Mediator_1_
:\Users\zaki\Desktop\aly eui\Year3_Sem2\SOFTWARE_SEM2\Deliverable 4\Mediator_1_Lear
Learner sends: I need help with assignment 2.
Instructor received from learner: I need help with assignment 2.
Instructor sends: Sure! I?ll explain it in today's session.
Learner received from instructor: Sure! I?ll explain it in today's session.
```

2. Course Enrollment Coordination:

Intent:

- To coordinate multiple components involved in course enrollment through a single mediator.

Problem:

When a learner enrolls:

- The system must check prerequisites
- Assign instructors if needed
- Notify the learner
- If they all talk to each other it will create a spaghetti code.

Solution:

- Creating an Enrollment Mediator

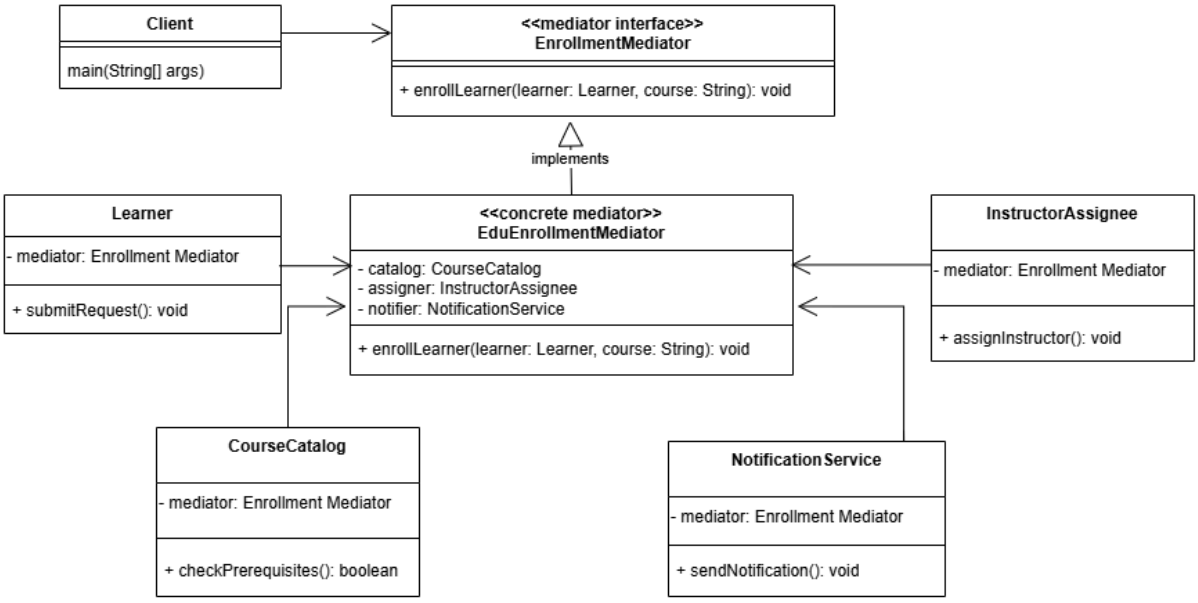
Components:

Mediator Interface: Enrollment Mediator

Concrete Mediator: Edu Enrollment Mediator

Colleagues: Learner, Course Catalog, Notification Service, Instructor Assigner

UML:



Running Code:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  QUERY RESULTS (PREVIEW)

PS C:\Users\zaki\Desktop\aly eui\Year3_Sem2\SOFTWARE_SEM2\Deliverable 4\Mediator_2_ Course Enroll
\Users\zaki\Desktop\aly eui\Year3_Sem2\SOFTWARE_SEM2\Deliverable 4\Mediator_2_ Course Enrollment
Nada Ashraf is attempting to enroll in Advanced Java Programming
Checking prerequisites for Nada Ashraf in Advanced Java Programming
Assigning instructor to the course: Advanced Java Programming
Notification sent to Nada Ashraf: Enrollment successful for Advanced Java Programming
```

Strategy Design Patterns:

1. Notify Learners of Scheduled Sessions:

Context Class (Notification):

- It holds a reference to the Notification Strategy interface and delegates strategy execution
  through the notify(session Details: string) method.

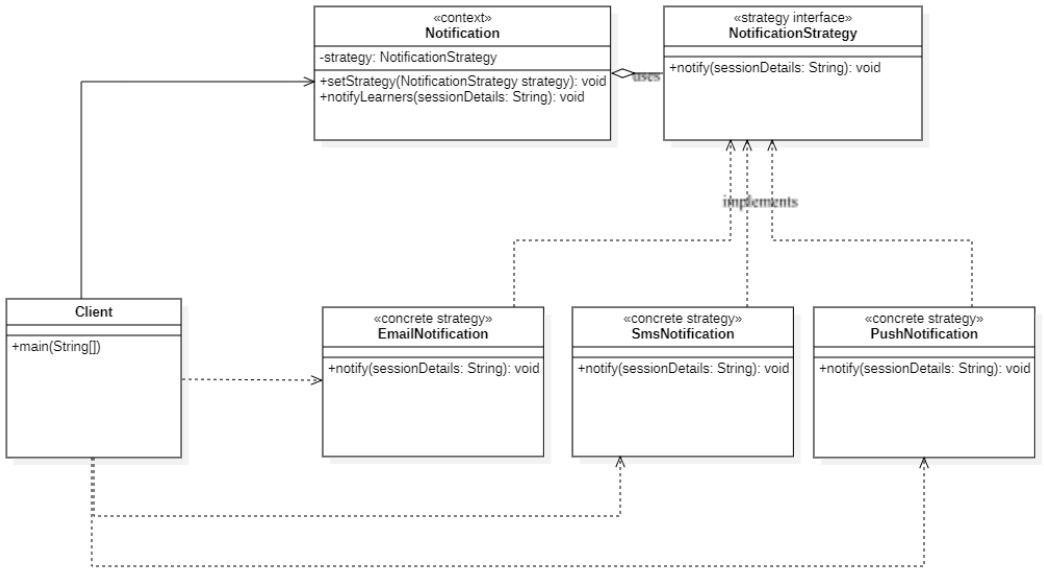
Strategy Interface (Notification Strategy):

- Defines the method notify(session Details: string) that all concrete strategies implement.

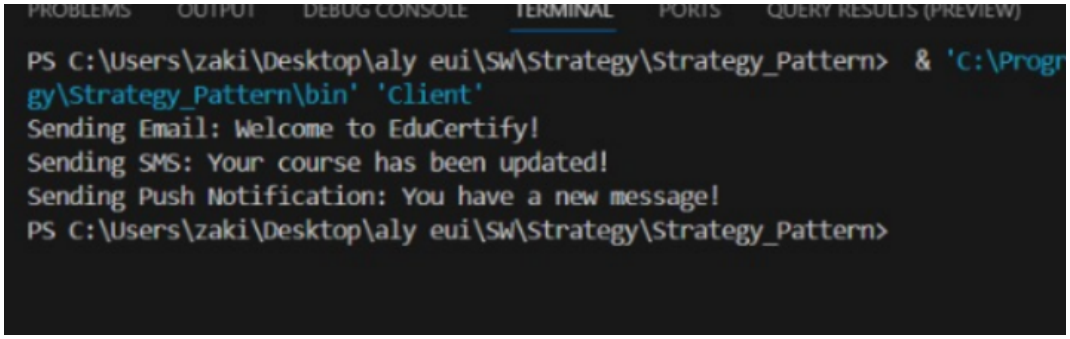
Concrete Strategies (Email Notification, SMS Notification, Push Notification):

- Implement the notify(session Details: string) method with specific notification logic.

UML:



Running Code:



2. Discount Strategy for Checkout:

Intent:

Allow flexible and interchangeable discount algorithms to be applied during checkout, based on learner type or promotional condition.

Problem:

Learners may be eligible for different types of discounts:

- Students get 15%
- Returning users get 10%
- First-time users get 20%
- Referral users get 50 EGP off

Hardcoding all these into one payment class, codebase becomes bloated and hard to maintain/extend.

Solution:

- Using the Strategy Pattern to encapsulate each discount rule in its own class. The checkout system selects the strategy dynamically.

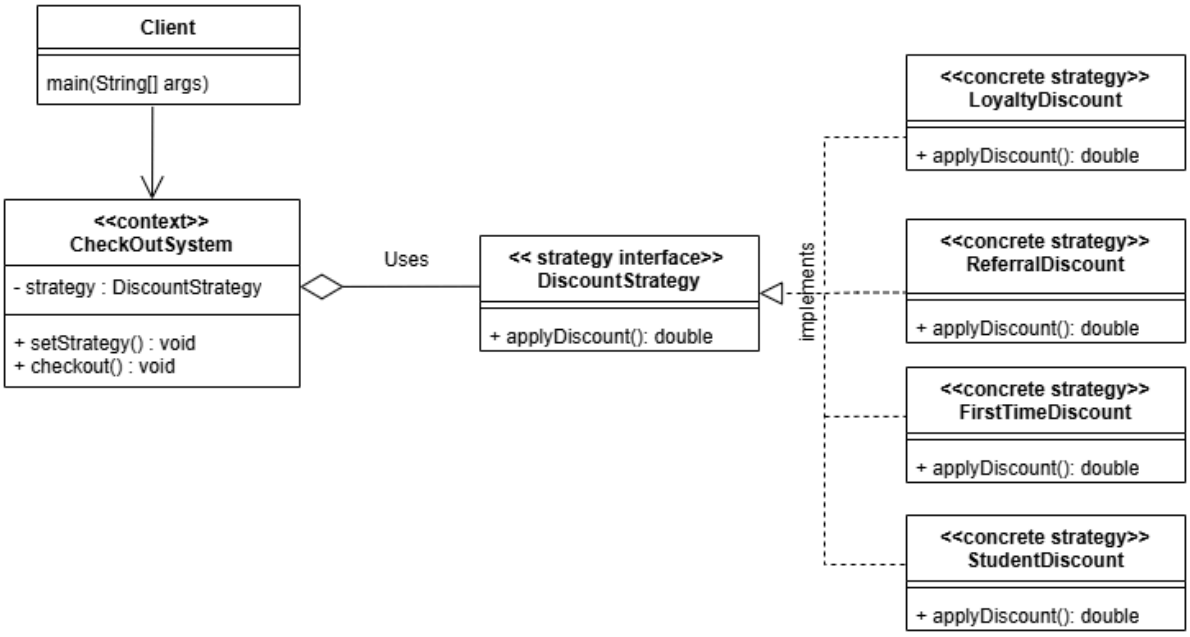
Components:

Context Class: Checkout System

Strategy Interface: Discount Strategy

Concrete Strategies: Student Discount, Loyalty Discount, First Time Discount, Referral Discount

UML:



Running Code:

```
sktop\aly eui\Year3_Sem2\SOFTWARE_SEM2\Deliverable 4\Strategy\bin' 'DiscountStrategyDemo'
Applying Loyalty Discount:
Final price after discount: 900.0

Applying Referral Discount:
Final price after discount: 950.0

Applying FirstTime Discount:
Final price after discount: 800.0

Applying Student Discount:
Final price after discount: 850.0
PS C:\Users\zaki\Desktop\aly eui\Year3_Sem2\SOFTWARE_SEM2\Deliverable 4\Strategy>
```

Iterator Design Pattern:

Course Module Navigation

Intent:

- Provide a way to access course modules sequentially without exposing the internal structure of the module collection.

Problem

- Learners access course content in a sequence (Module 1 then Module 2, etc.)
- If we let them directly manipulate the module list (like List.get()), we break encapsulation and lose control over how modules are accessed.

Solution:

- An Iterator that:
  - Walks through modules one-by-one
  - Keeps the internal list hidden
  - Can be extended to support filtering, etc.

Components:

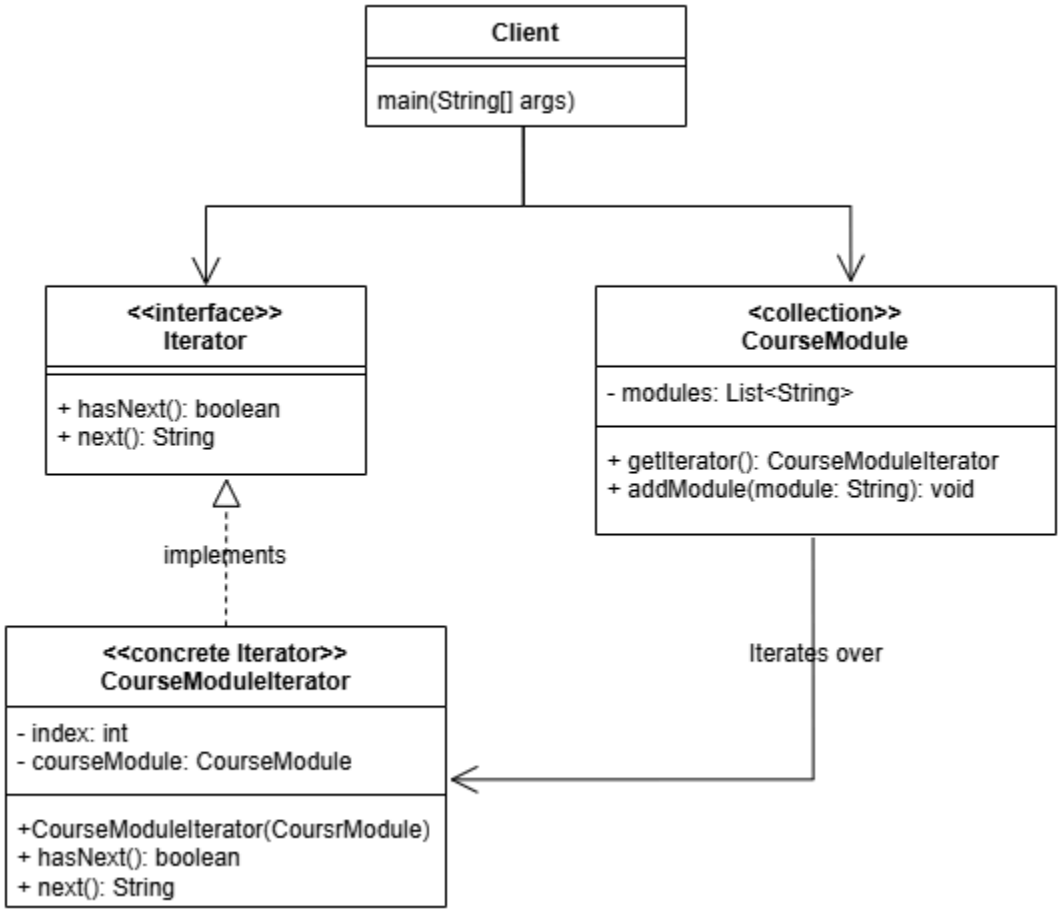
Iterator Interface: Module Iterator

Concrete Iterator: Course Module Iterator

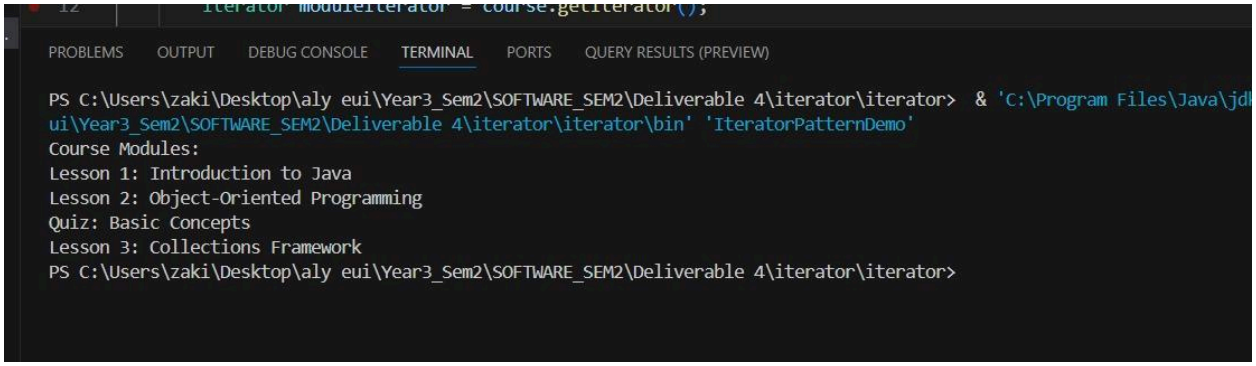
Collection Interface: Module Collection

Concrete Collection: Course Modules (stores a list of Module objects)

UML:



Running Code:



Observer Design Pattern:

Notify Learners of Grades

Subject Interface:

- Defines methods to add, remove, and notify observers.

Concrete Subject (Grade Notifier):

- Maintaining a list of Observers and notifying them when grades are updated.

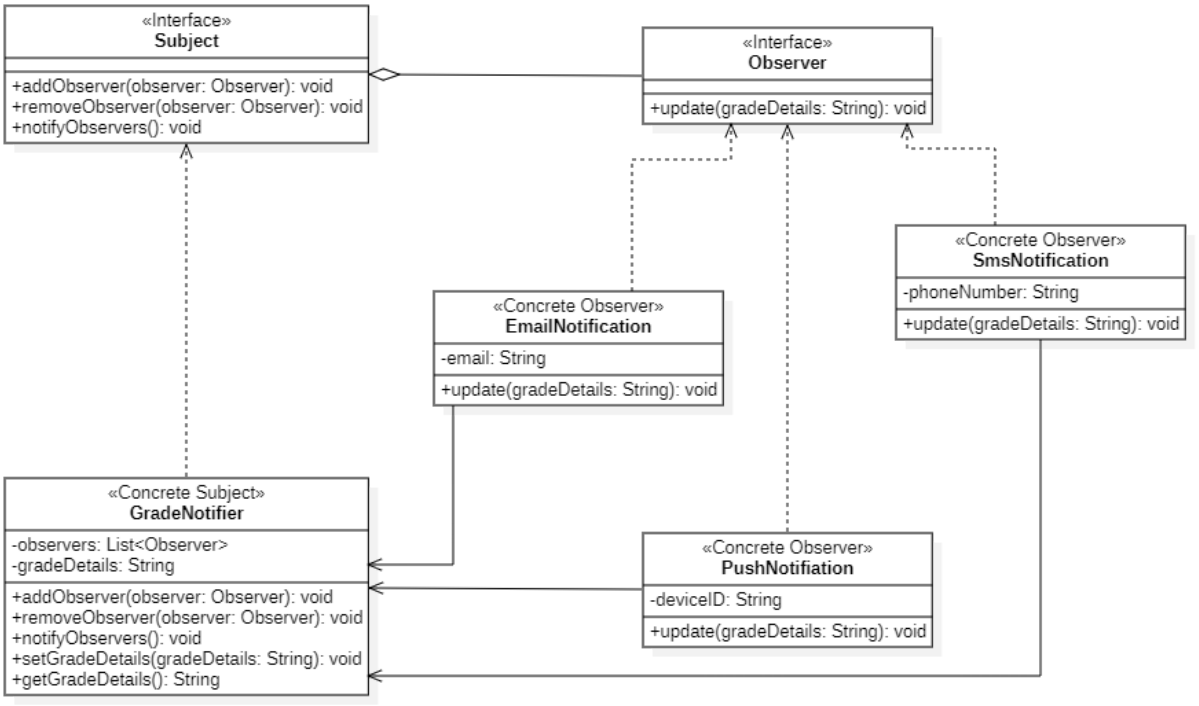
Observer Interface:

- Defines the method update(grade Details: String) for Observers.

Concrete Observers (Email Notification, SMS Notification, Push Notification).

- Implements the update() method to handle grade notifications.

UML:



Running Code

```
PS C:\Users\zaki\Desktop\aly eui\SW\Observer_Pattern\Observer_Pattern\src>  & 'C:\Program
ng\Code\User\workspaceStorage\9c8933e42549c8f8b9214410c3e6fb25\redhat.java\jdt_ws\src_16f
Email sent to student@example.com: Your grade for Course XYZ is A+.
SMS sent to 1234567890: Your grade for Course XYZ is A+.
Push notification sent to device Device123: Your grade for Course XYZ is A+.
Email sent to student@example.com: Your grade for Course XYZ is updated to A.
Push notification sent to device Device123: Your grade for Course XYZ is updated to A.
PS C:\Users\zaki\Desktop\aly eui\SW\Observer_Pattern\Observer_Pattern\src>
```