SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA GAI 3rd semester

Object Oriented Analysis and Design (PGI20C07J)

Lab Manual

Lab 1: Case study - the Next Gen POS system

Title

Case Study: The Next Gen Point of Sale (POS) System

Aim

To understand and analyze the requirements and functionalities of a modern Point of Sale (POS) system through a case study, focusing on object-oriented principles.

Procedure

1. **Introduction to Next Gen POS:** Research and understand the core features and common use cases of a typical Point of Sale (POS) system.
2. **Identify Key Actors:** Determine the primary users and external systems interacting with the POS system (e.g., Cashier, Customer, Manager, Inventory System, Payment Gateway).
3. **Identify Core Processes:** Outline the main business processes involved in a POS system (e.g., Sales Transaction, Product Lookup, Payment Processing, Inventory Management).
4. **Analyze System Requirements:** Based on the identified actors and processes, list functional and non-functional requirements for the Next Gen POS system.
5. **Discussion:** Discuss the challenges and object-oriented design considerations for such a system.

Source Code

(Not applicable for a case study; this section would be for design artifacts or code if a specific part were to be implemented.)

Input

(Not applicable for a case study.)

Expected Output

A detailed understanding and documentation of the Next Gen POS system's actors, core processes, and requirements.

Lab 2: Identify a software system that needs to be developed.

Title

System Identification for Object-Oriented Development

Aim

To identify and select a suitable software system for development that can serve as a comprehensive case study for applying Object-Oriented Analysis and Design (OOAD) principles throughout the lab series.

Procedure

1. **Brainstorming:** Discuss potential software systems that are relevant, have clear requirements, and are complex enough to demonstrate various OOAD concepts (e.g., Passport Automation System, Book Bank, Exam Registration, Stock Maintenance, Online Course Reservation System).
2. **Selection Criteria:** Define criteria for selecting the system, such as:
   - Clear scope and boundaries.
   - Multiple user roles and interactions.
   - Presence of various business rules.
   - Opportunity to apply different UML diagrams.
3. **System Selection:** Based on the brainstorming and criteria, select one system.
   - **Selected System for this Lab Manual:** Online Course Registration System
4. **Initial Scope Definition:** Provide a brief overview of the selected system's purpose and its primary functionalities.

Source Code

(Not applicable for this lab.)

Input

(Not applicable for this lab.)

Expected Output

A clearly identified software system (e.g., Online Course Registration System) with a brief description of its purpose and initial scope.

Lab 3: Document the Software Requirements Specification (SRS) for the identified system.

Title

Software Requirements Specification (SRS) for Online Course Registration System

Aim

To document the functional and non-functional requirements of the identified software system (Online Course Registration System) in a formal Software Requirements Specification (SRS) document.

Procedure

1. **Understand SRS Structure:** Familiarize with a standard SRS document structure (e.g., IEEE 830 standard).
2. **Gather Requirements:**
   - **Functional Requirements:** List all the functionalities the system must perform (e.g., User Registration, Course Browsing, Course Enrollment, Payment Processing, Admin Course Management).
   - **Non-Functional Requirements:** Specify quality attributes like performance, security, usability, reliability, maintainability, and scalability.
3. **Define Actors:** Clearly define all actors interacting with the system (e.g., Student, Instructor, Administrator, Guest).
4. **Create Use Case Descriptions (Initial):** Briefly describe the main use cases for each actor.
5. **Document Constraints:** List any design, implementation, or operational constraints.
6. **Review and Refine:** Review the documented requirements for clarity, completeness, consistency, and feasibility.

Source Code

(Not applicable for this lab; the output is a document.)

Input

(Not applicable for this lab.)

Expected Output

A comprehensive SRS document for the Online Course Registration System, detailing its functional and non-functional requirements.

Lab 4: Identify use cases

Title

Use Case Identification for Online Course Registration System

Aim

To identify and list the primary use cases for the Online Course Registration System based on the documented SRS.

Procedure

1. **Review SRS:** Go through the functional requirements documented in the SRS.
2. **Actor-Goal Analysis:** For each identified actor (Student, Instructor, Administrator, Guest), determine their goals when interacting with the system. Each goal typically corresponds to a use case.
3. **Identify Main Use Cases:** List the high-level functionalities that the system provides to its actors (e.g., Register for an Account, Browse Courses, Enroll in Course, View Enrolled Courses, Manage Courses, View Student Enrollments, Make Payment).
4. **Refine Use Case Names:** Ensure use case names are concise, action-oriented, and clearly describe the system's functionality from the actor's perspective.
5. **Initial Use Case Diagram Sketch:** Optionally, sketch a preliminary use case diagram to visualize the relationships between actors and use cases.

Source Code

(Not applicable for this lab; the output is a list of use cases.)

Input

(Not applicable for this lab.)

Expected Output

A comprehensive list of identified use cases for the Online Course Registration System.

Lab 5: Develop the Use Case model

Title

Use Case Model Development for Online Course Registration System

Aim

To develop a detailed Use Case Model for the Online Course Registration System, including a Use Case Diagram and detailed Use Case Descriptions.

Procedure

1. **Draw Use Case Diagram:**
   - Place all identified actors.
   - Draw the system boundary.
   - Place all identified use cases within the system boundary.
   - Draw association lines between actors and the use cases they interact with.
   - Identify and represent `<<include>>` and `<<extend>>` relationships between use cases where applicable (e.g., "Make Payment" might `<<include>>` "Process Credit Card").
2. **Write Detailed Use Case Descriptions:** For each significant use case, create a detailed description including:
   - **Use Case Name:**
   - **Actors:**
   - **Purpose:**
   - **Preconditions:**
   - **Postconditions:**
   - **Main Flow of Events:** Step-by-step description of the normal interaction.
   - **Alternative Flows:** Descriptions of alternative paths or error conditions.
   - **Exceptions:**
3. **Review and Validate:** Ensure the Use Case Model accurately reflects the system's functional requirements and is consistent with the SRS.

Source Code

(Not applicable for this lab; the output is a set of UML diagrams and documents.)

Input

(Not applicable for this lab.)

Expected Output

A complete Use Case Model for the Online Course Registration System, comprising a Use Case Diagram and detailed Use Case Descriptions for all identified use cases.

Lab 6: Identify the conceptual classes and develop a Domain Model and also derive a Class Diagram from that.

Title

Conceptual Class Identification, Domain Model, and Class Diagram Derivation for Online Course Registration System

Aim

To identify conceptual classes from the problem domain, develop a Domain Model representing these classes and their relationships, and subsequently derive an initial Class Diagram for the Online Course Registration System.

Procedure

1. **Identify Conceptual Classes:**
   - Read through the SRS and Use Case Descriptions.
   - Look for nouns and noun phrases that represent significant concepts, objects, or information in the problem domain (e.g., Course, Student, Instructor, Enrollment, Payment, Schedule).
   - Filter out redundant or irrelevant concepts.
2. **Develop Domain Model (Conceptual Model):**
   - Represent each conceptual class as a box.
   - Identify attributes for each class (e.g., Course has `courseId`, `title`, `credits`).
   - Identify associations between classes (e.g., Student `enrolls in` Course, Instructor `teaches` Course).
   - Specify multiplicities for associations (e.g., 1 Student can enroll in * many Courses).
3. **Derive Initial Class Diagram:**
   - Translate the conceptual classes and their associations from the Domain Model into a UML Class Diagram.
   - Add basic visibility (public, private) to attributes and methods (if any are obvious from the domain).
   - Consider inheritance or generalization if natural hierarchies exist (e.g., Person -> Student, Instructor).
4. **Review and Refine:** Ensure the Domain Model and Class Diagram accurately reflect the system's data and relationships.

Source Code

(Not applicable for this lab; the output is a set of UML diagrams.)

Input

(Not applicable for this lab.)

Expected Output

A Domain Model and an initial Class Diagram for the Online Course Registration System, showing conceptual classes, their attributes, and relationships.

Lab 7: Using the identified scenarios, find the interaction between objects and represent them using UML

Title

Object Interaction and UML Interaction Diagrams for Online Course Registration System

Aim

To identify and model the interactions between objects within the Online Course Registration System for specific scenarios, representing these interactions using appropriate UML interaction diagrams.

Procedure

1. **Select Key Scenarios:** Choose a few critical or complex scenarios from the Use Case Descriptions (e.g., "Student enrolls in a course," "Administrator adds a new course," "Student makes payment").
2. **Identify Participating Objects:** For each selected scenario, determine the objects that will interact (instances of the conceptual classes identified earlier).
3. **Trace Object Interactions:** For each scenario, trace the sequence of messages passed between these objects to achieve the scenario's goal.
4. **Represent using UML Interaction Diagrams:**
    - **Sequence Diagram:** Create a Sequence Diagram for each scenario, showing objects as lifelines and messages as directed arrows in chronological order. Include activation bars and return messages.
    - *(Optional: Collaboration Diagram/Communication Diagram if preferred for specific scenarios, showing objects and links between them, with messages numbered to indicate sequence.)*
5. **Review and Validate:** Ensure the diagrams accurately depict the flow of control and data between objects for the chosen scenarios.

Source Code

(Not applicable for this lab; the output is a set of UML diagrams.)

Input

(Not applicable for this lab.)

Expected Output

UML Interaction Diagrams (primarily Sequence Diagrams) for key scenarios of the Online Course Registration System, illustrating object interactions.

Lab 8: Sequence and Collaboration Diagrams.

Title

Detailed Sequence and Collaboration Diagrams for Online Course Registration System

Aim

To further develop and refine Sequence and Collaboration Diagrams for various scenarios within the Online Course Registration System, demonstrating different perspectives of object interaction.

Procedure

1. **Review Previous Interaction Diagrams:** Revisit the Sequence Diagrams created in Lab 7.
2. **Develop Additional Sequence Diagrams:** Create Sequence Diagrams for other important or complex scenarios not covered previously (e.g., "Instructor views enrolled students," "Guest browses courses," "System validates prerequisites").
3. **Develop Collaboration Diagrams (Communication Diagrams):** For a selected subset of scenarios, create corresponding Collaboration Diagrams.
   - Show objects as nodes and links between objects that communicate.
   - Number messages to indicate the sequence of interactions.
   - Highlight the structural relationships and message flow.
4. **Compare and Contrast:** Discuss the strengths and weaknesses of Sequence Diagrams vs. Collaboration Diagrams for different purposes.
5. **Refine Object Responsibilities:** As you draw these diagrams, identify and refine the responsibilities of each object.

Source Code

(Not applicable for this lab; the output is a set of UML diagrams.)

Input

(Not applicable for this lab.)

Expected Output

A collection of detailed Sequence and Collaboration Diagrams for the Online Course Registration System, covering a range of scenarios.

Lab 9: Draw relevant State Chart and Activity Diagrams for the same system

Title

State Chart and Activity Diagrams for Online Course Registration System

Aim

To model the dynamic behavior of objects and the flow of activities within the Online Course Registration System using State Chart Diagrams and Activity Diagrams.

Procedure

1. **Identify State-Dependent Objects:** Select objects or system elements that exhibit significant state changes throughout their lifecycle (e.g., Course Enrollment, User Account, Payment Transaction).
2. **Draw State Chart Diagrams:** For each identified state-dependent object:
   - Define all possible states the object can be in.
   - Identify events that trigger transitions between states.
   - Specify actions performed during state entry/exit or on transitions.
   - Represent initial and final states.
3. **Identify Business Processes/Workflows:** Choose key business processes or workflows from the system (e.g., "Course Enrollment Process," "New Course Creation Workflow," "User Registration Flow").
4. **Draw Activity Diagrams:** For each identified process/workflow:
   - Identify the sequence of activities.
   - Represent decision points, forks (parallel activities), and joins.
   - Use swimlanes to show which actor or object performs which activity.
   - Represent initial and final nodes.
5. **Review and Validate:** Ensure the diagrams accurately represent the dynamic aspects of the system.

Source Code

(Not applicable for this lab; the output is a set of UML diagrams.)

Input

(Not applicable for this lab.)

Expected Output

Relevant State Chart Diagrams for key objects and Activity Diagrams for important processes within the Online Course Registration System.

Lab 10: Implement the system as per the detailed design.

Title

Implementation of Online Course Registration System

Aim

To implement a functional prototype or a significant portion of the Online Course Registration System based on the detailed design derived from the previous OOAD phases.

Procedure

1. **Choose a Programming Language and Framework:** Select an appropriate object-oriented programming language (e.g., Java, Python, C#, PHP) and relevant frameworks/libraries.
2. **Set up Development Environment:** Configure the necessary tools (IDE, database, etc.).
3. **Translate Design to Code:**
    - **Class Implementation:** Implement classes identified in the Class Diagram, including attributes, methods, and relationships.
    - **Database Integration:** Design and implement the database schema based on the class model and integrate it with the application.
    - **Business Logic:** Implement the business rules and logic derived from Use Case Descriptions, Sequence Diagrams, and Activity Diagrams.
    - **User Interface (Basic):** Develop a basic user interface to interact with the system's core functionalities.
4. **Modular Development:** Develop the system in modules or components, following good software engineering practices.
5. **Unit Testing:** Perform unit tests for individual classes and methods.

Source Code

```
// Example: A simplified Java class for a Course
// This is a placeholder. Actual implementation will be much more extensive.

public class Course {
    private String courseId;
    private String title;
    private int credits;
    private String description;
    private int maxCapacity;
    private int currentEnrollment;

    public Course(String courseId, String title, int credits, String
description, int maxCapacity) {
        this.courseId = courseId;
        this.title = title;
        this.credits = credits;
        this.description = description;
        this.maxCapacity = maxCapacity;
        this.currentEnrollment = 0; // Initially no students enrolled
    }

    public String getCourseId() {
        return courseId;
    }
    public String getTitle() {
        return title;
    }

    public int getCredits() {
        return credits;
    }

    public String getDescription() {
        return description;
    }

    public int getMaxCapacity() {
        return maxCapacity;
    }

    public int getCurrentEnrollment() {
        return currentEnrollment;
    }

    public boolean enrollStudent() {
        if (currentEnrollment < maxCapacity) {
            currentEnrollment++;
            System.out.println("Student enrolled in " + title);
            return true;
        } else {
            System.out.println("Course " + title + " is full.");
            return false;
        }
    }

    public boolean dropStudent() {
        if (currentEnrollment > 0) {
            currentEnrollment--;
            System.out.println("Student dropped from " + title);
            return true;
        } else {
            System.out.println("No students to drop from " + title);
            return false;
        }
    }

    @Override
    public String toString() {
        return "Course [ID=" + courseId + ", Title=" + title + ", Credits=" +
credits +
               ", Enrollment=" + currentEnrollment + "/" + maxCapacity + "]";
    }

    // Main method for a simple test
    public static void main(String[] args) {
        Course ooad = new Course("CS101", "OOAD", 3, "Object-Oriented Analysis
and Design", 2);
        System.out.println(ooad);

        ooad.enrollStudent();
        System.out.println(ooad);

        ooad.enrollStudent();
        System.out.println(ooad);

        ooad.enrollStudent(); // Should fail
        System.out.println(ooad);

        ooad.dropStudent();
        System.out.println(ooad);
    }
```

```
}
```

Input

(Specific input will depend on the implemented functionalities, e.g., user credentials, course details, enrollment choices.)

Expected Output

A runnable software application that demonstrates the core functionalities of the Online Course Registration System, allowing users to perform actions like registration, course browsing, and enrollment.

Lab 11: package diagrams - Component and Deployment Diagrams.

Title

Component and Deployment Diagrams for Online Course Registration System

Aim

To model the physical architecture of the Online Course Registration System, including its software components and their dependencies (Component Diagram), and its deployment onto physical nodes (Deployment Diagram).

Procedure

1. **Identify Major Components:**
   - Review the implemented system (from Lab 10) and identify its major logical and physical components (e.g., User Interface Component, Business Logic Component, Database Component, API Gateway Component).
   - Consider reusable modules or subsystems.
2. **Draw Component Diagram:**
   - Represent each component as a rectangle with two smaller rectangles on its side.
   - Show interfaces provided (lollipop notation) and required (socket notation) by components.
   - Draw dependency relationships between components.
3. **Identify Deployment Nodes:**
   - Determine the physical hardware or software execution environments (nodes) where the system will run (e.g., Web Server, Application Server, Database Server, Client Machine).
4. **Draw Deployment Diagram:**
   - Represent each node as a 3D box.
   - Show communication paths between nodes.
   - Place components and artifacts (e.g., executable files, libraries, configuration files) inside the nodes where they are deployed.
   - Show dependencies between deployed artifacts.
5. **Review and Validate:** Ensure the diagrams accurately represent the system's physical structure and deployment strategy.

Source Code

(Not applicable for this lab; the output is a set of UML diagrams.)

Input

(Not applicable for this lab.)

Expected Output

Component Diagrams showing the software components and their relationships, and Deployment Diagrams illustrating the physical deployment of the Online Course Registration System.

Lab 12: Test the software system for all the scenarios identified as per the use case diagram.

Title

System Testing for Online Course Registration System

Aim

To systematically test the implemented Online Course Registration System against all the scenarios identified in the Use Case Diagram, ensuring that all functional requirements are met.

Procedure

1. **Develop Test Cases:** For each use case and its main/alternative flows (from Lab 5), create detailed test cases. Each test case should include:
   - **Test Case ID:**
   - **Use Case:**
   - **Scenario:**
   - **Preconditions:**
   - **Test Steps:** Detailed instructions to execute the test.
   - **Expected Result:** The anticipated outcome if the system functions correctly.
   - **Actual Result (to be filled during execution):**
   - **Status (Pass/Fail):**
2. **Prepare Test Data:** Create necessary test data for each test case (e.g., valid/invalid user credentials, course details, enrollment data).
3. **Execute Tests:** Run each test case on the implemented system.
4. **Record Results:** Document the actual results and compare them with the expected results. Mark each test case as Pass or Fail.
5. **Report Defects:** For any failed test cases, log defects with detailed information for debugging.
6. **Retest (if applicable):** After defects are fixed, retest the affected functionalities.

Source Code

(Not applicable for this lab; this involves executing the system and documenting results.)

Input

(Varies per test case, e.g., entering student details, selecting courses, making payments, administrative actions.)

Expected Output

A comprehensive test report detailing the execution of test cases, identified defects, and the overall functional readiness of the Online Course Registration System.

Lab 13: Improve the reusability and maintainability of the software system

Title

Enhancing Reusability and Maintainability of Online Course Registration System

Aim

To analyze the existing implementation of the Online Course Registration System and apply principles and techniques to improve its reusability and maintainability.

Procedure

1. **Code Review:** Conduct a thorough review of the existing source code (from Lab 10).
2. **Identify Areas for Improvement:**
   - **Duplication:** Look for redundant code blocks that can be refactored into reusable functions or classes.
   - **Tight Coupling:** Identify components or classes that are too dependent on each other, making changes difficult.
   - **Lack of Modularity:** Find areas where responsibilities are not clearly separated.
   - **Poor Readability:** Assess code for clarity, comments, and adherence to coding standards.
3. **Apply Refactoring Techniques:**
   - **Extract Method/Class:** Create new methods or classes for repetitive logic.
   - **Introduce Interfaces/Abstract Classes:** Promote loose coupling and enable polymorphism.
   - **Encapsulate Fields:** Ensure proper data hiding.
   - **Rename Variables/Methods:** Improve clarity.
4. **Implement Design Principles:** Apply principles like SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion) to guide improvements.
5. **Document Changes:** Clearly document the refactoring efforts and their impact on reusability and maintainability.

Source Code

(This section would contain refactored code snippets or entire refactored modules. Example below shows a general concept.)

```
// Example: Before refactoring (tightly coupled logic)
/*
public class EnrollmentService {
    public void enrollStudent(Student student, Course course) {
        // Direct database interaction here
        // Direct email sending logic here
        // Direct logging here
    }
}
*/

// Example: After refactoring (improved modularity and reusability)
// Introducing interfaces and separate services for better maintainability and
testability.

public interface EnrollmentRepository {
    void saveEnrollment(Enrollment enrollment);
}

public interface NotificationService {
    void sendEnrollmentConfirmation(Student student, Course course);
}

public class DatabaseEnrollmentRepository implements EnrollmentRepository {
    @Override
    public void saveEnrollment(Enrollment enrollment) {
        // Logic to save enrollment to database
        System.out.println("Enrollment saved to database: " +
enrollment.getStudent().getName() + " in " + enrollment.getCourse().getTitle());
    }
}

public class EmailNotificationService implements NotificationService {
    @Override
    public void sendEnrollmentConfirmation(Student student, Course course) {
        // Logic to send email
        System.out.println("Email sent to " + student.getName() + " for course "
+ course.getTitle());
    }
}

public class EnrollmentService {
    private EnrollmentRepository enrollmentRepo;
    private NotificationService notificationService;

    public EnrollmentService(EnrollmentRepository enrollmentRepo,
NotificationService notificationService) {
        this.enrollmentRepo = enrollmentRepo;
        this.notificationService = notificationService;
    }

    public boolean enrollStudent(Student student, Course course) {
        if (course.enrollStudent()) { // Assume course.enrollStudent() handles
capacity check
            Enrollment newEnrollment = new Enrollment(student, course);
            enrollmentRepo.saveEnrollment(newEnrollment);
            notificationService.sendEnrollmentConfirmation(student, course);
            return true;
        }
        return false;
    }
}

// Placeholder for Student, Course, Enrollment classes
class Student { String name; public Student(String name) { this.name = name; }
public String getName() { return name; }}
class Course { String title; private int currentEnrollment = 0; private int
maxCapacity = 2; public Course(String title) { this.title = title; } public
String getTitle() { return title; } public boolean enrollStudent() { if
(currentEnrollment < maxCapacity) { currentEnrollment++; return true; } return
false; }}
class Enrollment { Student student; Course course; public Enrollment(Student
student, Course course) { this.student = student; this.course = course; } public
Student getStudent() { return student; } public Course getCourse() { return
course; }}

// Main method to demonstrate refactored service
public static void main(String[] args) {
    EnrollmentRepository repo = new DatabaseEnrollmentRepository();
    NotificationService notifier = new EmailNotificationService();
    EnrollmentService service = new EnrollmentService(repo, notifier);

    Student alice = new Student("Alice");
    Course math = new Course("Mathematics");

    service.enrollStudent(alice, math);
}
```

Input

(Not applicable for this lab; the focus is on code restructuring.)

Expected Output

A refactored codebase for the Online Course Registration System demonstrating improved reusability through reduced duplication, better modularity, and adherence to design principles, leading to easier maintenance.

Lab 14: By applying appropriate design patterns.

Title

Applying Design Patterns to Online Course Registration System

Aim

To identify and apply appropriate object-oriented design patterns to specific problems or areas within the Online Course Registration System, further enhancing its design quality, flexibility, and maintainability.

Procedure

1. **Review System Design and Code:** Re-examine the system's architecture and existing code.
2. **Identify Problem Areas:** Look for recurring design problems or areas where flexibility, extensibility, or testability could be improved.
3. **Select Relevant Design Patterns:** Based on the identified problems, choose suitable Gang of Four (GoF) design patterns (e.g., Singleton, Factory Method, Observer, Strategy, Decorator, Adapter).
    - **Example Problem:** If creating different types of users (Student, Instructor, Admin) requires complex object creation logic, consider a **Factory Method** or **Abstract Factory** pattern.
    - **Example Problem:** If different payment methods are needed, consider a **Strategy** pattern.
    - **Example Problem:** If multiple components need to be notified when a course enrollment status changes, consider an **Observer** pattern.
4. **Implement Design Patterns:** Refactor the relevant parts of the code to incorporate the chosen design patterns.
5. **Document Pattern Application:** Explain which pattern was applied, why it was chosen, and how it improved the design.
6. **Test Changes:** Ensure that the application of design patterns does not introduce new bugs and that the system still functions as expected.

Source Code

(This section would contain code snippets or modules demonstrating the application of specific design patterns. Example below shows a general concept for Strategy Pattern.)

```
// Example: Strategy Pattern for Payment Processing
// Before: If-else or switch for different payment types within a single class.

// Step 1: Define the Strategy Interface
public interface PaymentStrategy {
    void pay(double amount);
}

// Step 2: Implement Concrete Strategies
public class CreditCardPayment implements PaymentStrategy {
    private String cardNumber;
    private String cvv;

    public CreditCardPayment(String cardNumber, String cvv) {
        this.cardNumber = cardNumber;
        this.cvv = cvv;
    }

    @Override
    public void pay(double amount) {
        System.out.println("Paying $" + amount + " using Credit Card (ending " +
cardNumber.substring(cardNumber.length() - 4) + ")");
        // Actual credit card processing logic
    }
}

public class PayPalPayment implements PaymentStrategy {
    private String email;

    public PayPalPayment(String email) {
        this.email = email;
    }

    @Override
    public void pay(double amount) {
        System.out.println("Paying $" + amount + " using PayPal account: " +
email);
        // Actual PayPal processing logic
    }
}

// Step 3: Create the Context Class that uses the Strategy
public class PaymentContext {
    private PaymentStrategy paymentStrategy;

    public void setPaymentStrategy(PaymentStrategy paymentStrategy) {
        this.paymentStrategy = paymentStrategy;
    }

    public void executePayment(double amount) {
        if (paymentStrategy == null) {
            System.out.println("No payment strategy set.");
            return;
        }
        paymentStrategy.pay(amount);
    }

    // Main method to demonstrate usage
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();

        // Pay with Credit Card
        context.setPaymentStrategy(new CreditCardPayment("1234-5678-9012-3456",
"123"));
        context.executePayment(100.50);

        // Pay with PayPal
        context.setPaymentStrategy(new PayPalPayment("student@example.com"));
        context.executePayment(75.25);
    }
}
```

Input

(Not applicable for this lab; the focus is on code restructuring.)

Expected Output

A refined codebase for the Online Course Registration System demonstrating the effective application of one or more design patterns, leading to a more flexible, extensible, and robust design.

Lab 15: Implement the modified system and test it for various scenarios.

Title

Implementation and Comprehensive Testing of Modified Online Course Registration System

Aim

To implement the changes and improvements made in Labs 13 and 14 (reusability, maintainability, design patterns) into the Online Course Registration System, and then perform comprehensive testing across various scenarios to ensure stability and correctness.

Procedure

1. **Integrate Modifications:** Incorporate all the refactored code and design pattern implementations into the main codebase of the Online Course Registration System.
2. **Perform Regression Testing:** Re-run all previously passed test cases (from Lab 12) to ensure that the modifications have not introduced any regressions or broken existing functionalities.
3. **Develop New Test Cases (if necessary):** If the design pattern application or reusability improvements introduced new functionalities or changed existing behavior significantly, create new test cases to cover these changes.
4. **Execute New Test Cases:** Run any newly developed test cases.
5. **Perform Scenario-Based Testing:** Test the system end-to-end for various realistic user scenarios, including edge cases and error conditions, to validate the overall system behavior.
6. **Performance and Usability Testing (Basic):** Conduct basic checks for system performance and user interface usability.
7. **Document Results:** Record all test results, including any new defects found, and the overall assessment of the modified system's quality.

Source Code

(This section would refer back to the fully integrated and modified codebase from Labs 10, 13, and 14. No new code is typically written here unless a minor bug fix is needed during testing.)

Input

(Varies per test case and scenario, covering all aspects of the system, including valid, invalid, and boundary inputs.)

Expected Output

A fully functional and stable Online Course Registration System that incorporates the improvements from previous labs, along with a detailed test report confirming its correctness and robustness across various scenarios.