HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

KIẾN TRÚC VÀ THIẾT KẾ PHẦN MỀM

Giảng viên: Nguyễn Văn Hữu Hoàng

Sinh viên thực hiện:

Nguyễn Thị Thu Hiền – N20DCCN100

Nguyễn Thị Hiền – N20DCCN099

Nguyễn Hoài Hân – N20DCCN098

Nguyễn Trọng Nhân – N20DCCN122

Lớp: D20CQCNPM02 – N

Mục lục

BẢNG PHÂN CÔNG CÔNG VIỆC

Nguyễn Thị Thu Hiền	Bài tập chương 5: Bài 1,2,3 Viết báo cáo
Nguyễn Thị Hiền	Bài tập chương 5: Bài 4 Viết báo cáo
Nguyễn Hoài Hân	Bài tập chương 6: 1,2,3 Viết báo cáo
Nguyễn Trọng Nhân	bài tập chương 6: 3 ,4 Viết báo cáo

BÀI TẬP CHƯƠNG 5

Bài 1

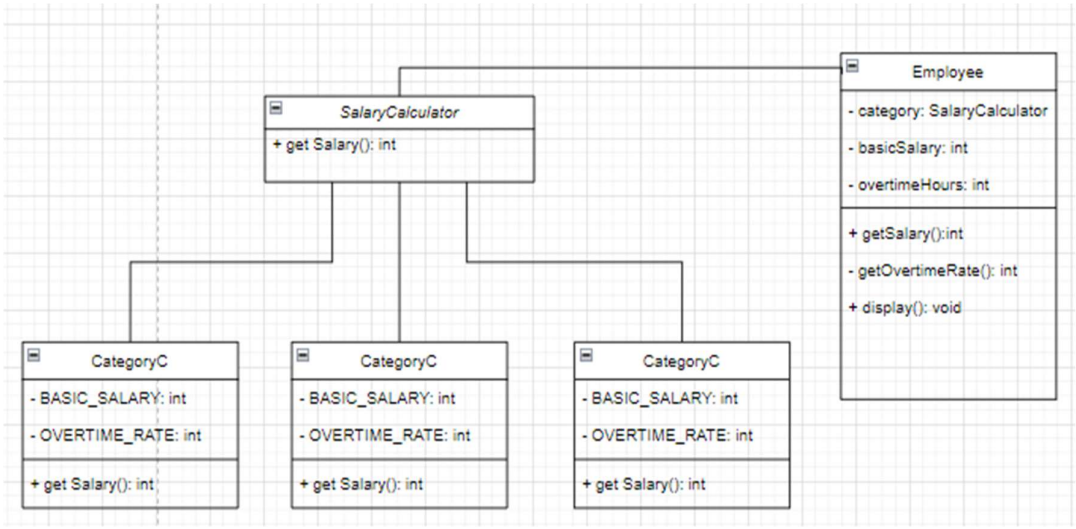
Yêu cầu: Sử dụng khái niệm lớp giao diện interface để xây dựng một ứng dụng tính và hiển thị lương của các loại nhân viên khác nhau trong một doanh nghiệp phần mềm cho trong Bảng sau. Yêu cầu cài đặt như sau: Dịch vụ tính lương cung cấp bởi các đối tượng thuộc loại khác nhau như CategoryA, CategoryB…được cài đặt từ getSalary() của interface SalaryCaculator; Lớp nhân viên Employee triển khai interface SalaryCaculator và cài đặt phương thức hiển thị display().

a) Vẽ biểu đồ UML để thể hiện các quan hệ trên.

b) Cài đặt đầy đủ với lớp thực thi MainApp. Lương được tính theo công thức lương cơ bản cộng với lương làm thêm giờ.

Bài làm:

a) a) Biểu đồ UML:



b) Cài đặt lớp thực thi MainApp

```
package Bai1;

interface SalaryCalculator {
    int getSalary();
}

class Employee implements SalaryCalculator {
    private SalaryCalculator category;
    private int basicSalary;
    private int overtimeHours;

    public Employee(SalaryCalculator category, int basicSalary, int overtimeHours) {
        this.category = category;
        this.basicSalary = basicSalary;
        this.overtimeHours = overtimeHours;
    }
    @Override
    public int getSalary() {
        return basicSalary + (overtimeHours * getOvertimeRate());
    }
    private int getOvertimeRate() {
        if (category instanceof CategoryA) {
            return 15;
        } else if (category instanceof CategoryB) {
            return 10;
        } else if (category instanceof CategoryC) {
            return 5;
        } else {
            return 0;
        }
    }

    public void display() {
        System.out.println("Loại: " + category.getClass().getSimpleName());
        System.out.println("Lương cơ bản: " + basicSalary);
        System.out.println("Số giờ làm thêm: " + overtimeHours);
        System.out.println("Tổng lương: " + getSalary());
    }
}
class CategoryA implements SalaryCalculator {
    private static final int BASIC_SALARY = 2000;
    private static final int OVERTIME_RATE = 15;

    CategoryA() {
    }
    @Override
    public int getSalary() {
```

```
        return BASIC_SALARY;
    }
}
// Class CategoryB
class CategoryB implements SalaryCalculator {
    private static final int BASIC_SALARY = 1500;
    private static final int OVERTIME_RATE = 10;
    @Override
    public int getSalary() {
        return BASIC_SALARY;
    }
}
// Class CategoryC
class CategoryC implements SalaryCalculator {
    private static final int BASIC_SALARY = 800;
    private static final int OVERTIME_RATE = 5;
    @Override
    public int getSalary() {
        return BASIC_SALARY;
    }
}
public class MainApp {
    public static void main(String[] args) {
        Employee emp1 = new Employee(new CategoryA(), 2000, 15);
        Employee emp2 = new Employee(new CategoryB(), 1500, 10);
        Employee emp3 = new Employee(new CategoryC(), 800, 8);

        System.out.println("Lập trình, thiết kế và tư vấn:");
        emp1.display();
        System.out.println("\nĐại diện bán hàng, quản lý bán hàng, kế toán, nhân viên kiểm chứng:");
        emp2.display();
        System.out.println("\nNhân viên bán hàng, nhân viên tiếp thị:");
        emp3.display();
    }
}
```

Kết quả:

```
run:
Lập trình, thiết kế và tư vấn:
Loại: CategoryA
Lương cơ bản: 2000
Số giờ làm thêm: 15
Tổng lương: 2225

Đại diện bán hàng, quản lý bán hàng, kế toán, nhân viên kiểm chứng:
Loại: CategoryB
Lương cơ bản: 1500
Số giờ làm thêm: 10
Tổng lương: 1600

Nhân viên bán hàng, nhân viên tiếp thị:
Loại: CategoryC
Lương cơ bản: 800
Số giờ làm thêm: 8
Tổng lương: 840
BUILD SUCCESSFUL (total time: 0 seconds)
```

Bài 2

Yêu cầu: Thiết kế interface Search khai báo phương thức tìm một mặt hàng như sách Book trong một danh sách. Thiết kế và cài đặt hai lớp tìm kiếm nhị phân BinarySearch và tìm kiếm tuần tự LinearSearch để thực thi các cách tìm kiếm trong danh sách.

Bài làm:

1. **Class Book**

```
package Bai2;
// Class Book
class Book implements Comparable<Book> {
    private String title;
    private String author;
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }
    public String getTitle() {
        return title;
    }
    public String getAuthor() {
        return author;
    }

    @Override
    public String toString() {
        return "Book{" +
                "title='" + title + '\'' +
                ", author='" + author + '\'' +
```

```
            '}';
    }
    @Override
    public int compareTo(Book other) {
        return this.title.compareTo(other.title);
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Book book = (Book) obj;
        return title.equals(book.title) && author.equals(book.author);
    }
}
```

2. Interface Search

```
package Bai2;

// Interface Search
public interface Search<T> {
    int search(T[] arr, T target);
}
```

3. Class BinarySearch

```
package Bai2;

// Class BinarySearch
class BinarySearch<T extends Comparable<T>> implements Search<T> {
    @Override
    public int search(T[] arr, T target) {
        int low = 0;
        int high = arr.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            int cmp = target.compareTo(arr[mid]);
            if (cmp == 0) {
                return mid;
            } else if (cmp < 0) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        return -1;
    }
}
```

4. Class LinearSearch

```
package Bai2;
// Class LinearSearch
class LinearSearch<T> implements Search<T> {
    @Override
    public int search(T[] arr, T target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i].equals(target)) {
                return i;
            }
        }
        return -1;
    }
}
```

5. Main

```
package Bai2;
// Main class

import java.util.Arrays;
import java.util.Comparator;

public class Main {
    public static void main(String[] args) {
        Book[] books = {
            new Book("Java Programming", "John Smith"),
            new Book("Python Basics", "Alice Johnson"),
            new Book("C++ Programming", "Bob Williams"),
            new Book("JavaScript Essentials", "Emily Davis")

        };

        // LinearSearch
        Search<Book> linearSearch = new LinearSearch<>();
        System.out.println("Tìm kiếm sách: Python Basics");
        int linearResult = linearSearch.search(books, new Book("Python Basics", "Alice Johnson"));
        if (linearResult != -1) {
            System.out.println("Linear Search: Sách được tìm thấy tại vị trí"+ linearResult);
        } else {
            System.out.println("Linear Search: Không tìm thấy sách");
        }

        // BinarySearch
        Arrays.sort(books, Comparator.comparing(Book::getTitle));
        Search<Book> binarySearch = new BinarySearch<>();
        System.out.println("Tìm kiếm sách: C++ Programming");
        int binaryResult = binarySearch.search(books, new Book("C++ Programming", "Bob Williams"));
        if (binaryResult != -1) {
            System.out.println("Binary Search: Sách được tìm thấy tại vị trí:" + binaryResult);
```

```
        } else {
            System.out.println("Binary Search: Không tìm thấy sách");
        }
    }
}
```

Kết quả:

```
run:
Tìm kiếm sách: Python Basics
Linear Search: Sách được tìm thấy tại vị trí1
Tìm kiếm sách: C++ Programming
Binary Search: Sách được tìm thấy tại vị trí:0
BUILD SUCCESSFUL (total time: 1 second)
```

Bài 3

Yêu cầu: Thiết kế interface AddressValidator khai báo các phương thức để xác nhận các phần khác nhau của một địa chỉ. Thiết kế và cài đặt hai lớp USAAddress (Hoa kỳ) và VNAddress (Việt nam) để xác nhận các địa chỉ của các nước tương ứng.

Bài làm:

```
package Bai3;

import java.util.Arrays;
interface AddressValidator {
    boolean isValidStreet(String street);
    boolean isValidCity(String city);
    boolean isValidState(String state);
    boolean isValidPostalCode(String postalCode);
}
// Class USAAddress
class USAAddress implements AddressValidator {
    @Override
    public boolean isValidStreet(String street) {
        return street != null && street.trim().length() >= 5;
    }
    @Override
    public boolean isValidCity(String city) {
        return city != null && city.trim().length() > 0;
    }
    @Override
    public boolean isValidState(String state) {
        String[] states = {"AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL",
"IN", "IA", "KS", "KY", "LA", "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV",
"NH", "NJ", "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT",
"VT", "VA", "WA", "WV", "WI", "WY"};
        return state != null && state.trim().length() == 2 &&
Arrays.asList(states).contains(state.toUpperCase());
    }
```

```
    public boolean isValidZip(String zip) {
        return zip != null && zip.matches("^\\d{5}(-\\d{4})?$");
    }
    @Override
    public boolean isValidPostalCode(String postalCode) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
// Class VNAddress
class VNAddress implements AddressValidator {
    @Override
    public boolean isValidStreet(String street) {
        return street != null && street.trim().length() >= 3;
    }
    @Override
    public boolean isValidCity(String city) {
        return city != null && city.trim().length() > 0;
    }
    @Override
    public boolean isValidState(String state) {
        return true;
    }
    public boolean isValidZip(String zip) {
        return true;
    }
    @Override
    public boolean isValidPostalCode(String postalCode) {
        return true;
    }
}

public class Bai3 {

    public static void main(String[] args) {
    USAAddress usaAddress = new USAAddress();
    System.out.println("USA Address Validation:");
    System.out.println("Street: "+" 123 Main St \t" + usaAddress.isValidStreet("245 Main St"));
    System.out.println("City: "+ " New York \t" + usaAddress.isValidCity("New York"));
    System.out.println("State: "+" BB \t" + usaAddress.isValidState("BB"));
    System.out.println("Zip: "+"12345 \t" + usaAddress.isValidZip("12345"));

    VNAddress vnAddress = new VNAddress();
    System.out.println("\nXác thực địa chỉ tại Việt Nam:");
    System.out.println("Đường: "+ " 123 Đường 3 Tháng 2 \t" + vnAddress.isValidStreet("123 Đường 3
Tháng 2"));
    System.out.println("Thành phố: "+" Hồ Chí Minh \t" + vnAddress.isValidCity("Hồ Chí Minh"));
    System.out.println("Mã tỉnh/ thành phố: "+" NULL \t" + vnAddress.isValidState(""));
    System.out.println("ZipCode: "+" 123456 \t" + vnAddress.isValidZip("123456"));
```

```
    }
}
```

Kết quả:

```
run:
USA Address Validation:
Street:  123 Main St    true
City:  New York         true
State:  BB      false
Zip: 12345      true

Xác thực địa chỉ tại Việt Nam:
Đường:  123 Đường 3 Tháng 2     true
Thành phố:  Hồ Chí Minh         true
Mã tỉnh/ thành phố:  NULL       true
ZipCode:  123456        true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Bài 4

Đề: Trong một doanh nghiệp, nhân viên thường được thể hiện dạng phân cấp lớp với lớp cơ sở là nhân viên Employee và sau đó là nhân viên đại diện bán hàng SalesRep, nhân viên tư vấn Consultant… Hãy tạo một số phương thức sau đây trong lớp Employee:

- Lưu và hiển thị dữ liệu nhân viên.
- Truy nhập thuộc tính nhân viên như tên, id.
- Tính thu nhập hàng tháng cho nhân viên.

b) Sử dụng lớp trừu tượng abstract để cài đặt yêu cầu trên. Chú ý rằng 2 phương thức đầu là chung cho các loại nhân viên, nhưng tính thu nhập hàng tháng lại là khác nhau cho các loại nhân viên.

c) Sử dụng lớp interface để cài đặt. So sánh hai cách cài đặt này.

Bài làm:

b)

1. abstract

```
package Bai4.Abstract;

public abstract class Employee {
    protected int id;
    protected String name;

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }
```

```
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public abstract void saveEmployeeData();
    public void dislayInfo(){
        System.out.println("Id: " + id);
        System.out.println("Name: " + name);
    }

    public abstract double calculateMonthlyIncome();
}
```

2. class Consultant

```
package Bai4.Abstract;

public class Consultant extends Employee{
    protected int hoursWorked;
    protected double hoursRate;
    public Consultant(int id, String name, int hoursWorked, double hoursRate) {
        super(id, name);
        this.hoursWorked = hoursWorked;
        this.hoursRate = hoursRate;
    }
    @Override
    public void saveEmployeeData() {
        System.out.println("\nId: " + id);
        System.out.println("Tên: " + name);
        System.out.println("Thời gian làm việc: " + hoursWorked);
        System.out.println("Tỷ lệ giờ: " + hoursRate);
        System.out.println("Thu nhập hàng tháng: " + calculateMonthlyIncome());
    }

    @Override
    public double calculateMonthlyIncome() {
        return hoursWorked * hoursRate;
    }
}
```

3. class SalesRep

```
package Bai4.Abstract;

public class SalesRep extends Employee{
    protected double saleAmount;
    public SalesRep(int id, String name, double saleAmount) {
```

```
        super(id, name);
        this.saleAmount = saleAmount;
    }
    @Override
    public void saveEmployeeData() {
        System.out.println("\nId: " + id);
        System.out.println("Tên: " + name);
        System.out.println("Số lượng bán: " + saleAmount);
        System.out.println("Thu nhập hàng tháng: " + calculateMonthlyIncome());
    }
    @Override
    public double calculateMonthlyIncome() {
        return saleAmount * 0.1;
    }
}
```

c)

1. interface

```
package Bai4.Interface;
public interface IEmployee {
    public int getId();
    public String getName();
    public void saveEmployeeData();
    public double calculateMonthlyIncome();
}
```

2. class Iconsultant

```
package Bai4.Interface;
public class IConsultant implements IEmployee{
    protected int id;
    protected String name;
    protected int hoursWorked;
    protected double hoursRate;

    public IConsultant(int id, String name, int hoursWorked, double hoursRate) {
        this.id = id;
        this.name = name;
        this.hoursWorked = hoursWorked;
        this.hoursRate = hoursRate;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
```

```
    @Override
    public void saveEmployeeData() {
        System.out.println("\nId: " + id);
        System.out.println("Tên: " + name);
        System.out.println("Thời gian làm việc:" + hoursWorked);
        System.out.println("Tỷ lệ giờ: " + hoursRate);
        System.out.println("Thu nhập hàng tháng: " + calculateMonthlyIncome());
    }

    @Override
    public double calculateMonthlyIncome() {
        return hoursWorked * hoursRate;
    }
}
```

3. class IsalesRep

```
package Bai4.Interface;

public class ISalesRep implements IEmployee{
    protected int id;
    protected String name;
    protected double salesAmount;

    public ISalesRep(int id, String name, double salesAmount) {
        this.id = id;
        this.name = name;
        this.salesAmount = salesAmount;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    @Override
    public void saveEmployeeData() {
        System.out.println("\nId: " + id);
        System.out.println("Tên: " + name);
        System.out.println("Số lượng bán: " + salesAmount);
        System.out.println("Thu nhập hàng tháng: " + calculateMonthlyIncome());
    }

    @Override
    public double calculateMonthlyIncome() {
        return salesAmount * 0.1;
    }
}
```

Main

```java
package Bai4;

import Bai4.Abstract.Consultant;
import Bai4.Abstract.SalesRep;
import Bai4.Interface.IConsultant;
import Bai4.Interface.ISalesRep;
import java.util.ArrayList;
import java.util.List;

/**
 * @author NTTHien
 */
public class Main {

    public static void main(String[] args){

        //b) Abstract

        List<SalesRep> sr = new ArrayList<>();
        sr.add(new SalesRep(1, "Nguyen Thi Thu Hien", 2000));
        sr.add(new SalesRep(2, "Nguyen Thi Hien", 1500));
        for (SalesRep salesRep : sr) {
            salesRep.saveEmployeeData();
        }
        List<Consultant> consultants = new ArrayList<>();
        consultants.add(new Consultant(3, "Nguyen Hoai Han", 9, 0.4));
        consultants.add(new Consultant(4, "Nguyen Tran Trong Nhan", 10, 1.2));
        for (Consultant consultant : consultants) {
            consultant.saveEmployeeData();
        }

        //c) Interface
        List<ISalesRep> isr = new ArrayList<>();

        isr.add(new ISalesRep(1, "Nguyen Thi Thu Hien", 2000));
        isr.add(new ISalesRep(2, "Nguyen Thi Hien", 1500));
        for (ISalesRep iSalesRep : isr) {
            iSalesRep.saveEmployeeData();
        }
        List<IConsultant> iConsultants = new ArrayList<>();
        iConsultants.add(new IConsultant(3, "Nguyen Hoai Han", 9, 0.4));
        iConsultants.add(new IConsultant(4, "Nguyen Tran Trong Nhan", 10, 1.2));
        for (IConsultant iConsultant : iConsultants) {
            iConsultant.saveEmployeeData();
        }
    }
}
```

Kết quả:

b)

```
run:

Id: 1
Tên: Nguyen Thi Thu Hien
Số lượng bán: 2000.0
Thu nhập hàng tháng: 200.0

Id: 2
Tên: Nguyen Thi Hien
Số lượng bán: 1500.0
Thu nhập hàng tháng: 150.0

Id: 3
Tên: Nguyen Hoai Han
Thời gian làm việc: 9
Tỷ lệ giờ: 0.4
Thu nhập hàng tháng: 3.6

Id: 4
Tên: Nguyen Tran Trong Nhan
Thời gian làm việc: 10
Tỷ lệ giờ: 1.2
Thu nhập hàng tháng: 12.0
```

c)

```
Id: 1
Tên: Nguyen Thi Thu Hien
Số lượng bán: 2000.0
Thu nhập hàng tháng: 200.0

Id: 2
Tên: Nguyen Thi Hien
Số lượng bán: 1500.0
Thu nhập hàng tháng: 150.0

Id: 3
Tên: Nguyen Hoai Han
Thời gian làm việc:9
Tỷ lệ giờ: 0.4
Thu nhập hàng tháng: 3.6

Id: 4
Tên: Nguyen Tran Trong Nhan
Thời gian làm việc:10
Tỷ lệ giờ: 1.2
Thu nhập hàng tháng: 12.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

BÀI TẬP CHƯƠNG 6

Yêu cầu: Xây dựng các mẫu thiết kế Factory, Singleton cho hệ thống bán sách online.
Ghi chú: Chỉ thiết kế và cài đặt mẫu, không cần lập trình cả hệ thống.
Gợi ý: Liệt kê các chức năng cơ bản cần có của hệ thống và thiết kế các mẫu để đáp ứng các chức năng này.

Bài làm

Để xây dựng các mẫu thiết kế Factory và Singleton cho hệ thống bán sách online, chúng ta cần xác định các chức năng cơ bản của hệ thống và thiết kế các mẫu để đáp ứng các chức năng này.

1. Quản lý sách:

- **Mẫu Factory**: Sử dụng mẫu Factory để tạo ra các đối tượng sách từ các lớp con cụ thể, ví dụ như sách giáo trình, sách tiểu thuyết, sách kỹ thuật, v.v.

```
// Mẫu Factory: Tạo ra các đối tượng sách từ các lớp con cụ thể
public abstract class Book {
    public abstract void display();
}

public class Textbook extends Book {
    @Override
    public void display() {
        System.out.println("This is a textbook.");
    }
}

public class Novel extends Book {
    @Override
    public void display() {
        System.out.println("This is a novel.");
    }
}
```

- **Mẫu Singleton:** Sử dụng mẫu Singleton để đảm bảo rằng chỉ có một đối tượng quản lý sách duy nhất trong hệ thống.

```
// Mẫu Singleton: Đảm bảo chỉ có một đối tượng quản lý sách duy nhất trong hệ thống
public class BookManager {
    private static BookManager instance;

    private BookManager() {
        // Khởi tạo các tài nguyên và dữ liệu cho quản lý sách
    }

    public static BookManager getInstance() {
        if (instance == null) {
            instance = new BookManager();
        }
        return instance;
    }

    // Các phương thức quản lý sách
    public void addBook(Book book) {
        // Thêm sách vào hệ thống
    }

    public void removeBook(Book book) {
        // Xóa sách khỏi hệ thống
    }

    public void displayBooks() {
        // Hiển thị danh sách sách trong hệ thống
    }
}
```

2. Quản lý người dùng:

- **Mẫu Factory:** Sử dụng mẫu Factory để tạo ra các đối tượng người dùng từ các lớp con cụ thể, ví dụ như người dùng quản trị, người dùng thường, v.v.

```
// Mẫu Factory: Tạo ra các đối tượng người dùng từ các lớp con cụ thể
public abstract class User {
    public abstract void display();
}

public class AdminUser extends User {
    @Override
    public void display() {
        System.out.println("This is an admin user.");
    }
}

public class RegularUser extends User {
    @Override
    public void display() {
        System.out.println("This is a regular user.");
    }
}

public class UserFactory {
    public User createUser(String userType) {
        if (userType.equals("admin")) {
            return new AdminUser();
        } else if (userType.equals("regular")) {
            return new RegularUser();
        }
        return null;
    }
}
```

- **Mẫu Singleton:** Sử dụng mẫu Singleton để đảm bảo rằng chỉ có một đối tượng quản lý người dùng duy nhất trong hệ thống.

```
// Mẫu Singleton: Đảm bảo chỉ có một đối tượng quản lý người dùng duy nhất trong hệ thống
public class UserManager {
    private static UserManager instance;

    private UserManager() {
        // Khởi tạo các tài nguyên và dữ liệu cho quản lý người dùng
    }

    public static UserManager getInstance() {
        if (instance == null) {
            instance = new UserManager();
        }
        return instance;
    }

    // Các phương thức quản lý người dùng
    public void addUser(User user) {
        // Thêm người dùng vào hệ thống
    }

    public void removeUser(User user) {
        // Xóa người dùng khỏi hệ thống
    }

    public void displayUsers() {
        // Hiển thị danh sách người dùng trong hệ thống
    }
}
```

3. Quản lý đơn hàng:

- **Mẫu Factory**: Sử dụng mẫu Factory để tạo ra các đối tượng đơn hàng từ các lớp con cụ thể, ví dụ như đơn hàng mua sách, đơn hàng trả sách, v.v.

```
// Mẫu Factory: Tạo ra các đối tượng đơn hàng từ các lớp con cụ thể
public abstract class Order {
    public abstract void process();
}

public class PurchaseOrder extends Order {
    @Override
    public void process() {
        System.out.println("Processing purchase order.");
    }
}

public class ReturnOrder extends Order {
    @Override
    public void process() {
        System.out.println("Processing return order.");
    }
}

public class OrderFactory {
    public Order createOrder(String orderType) {
        if (orderType.equals("purchase")) {
            return new PurchaseOrder();
        } else if (orderType.equals("return")) {
            return new ReturnOrder();
        }
        return null;
    }
}
```

- **Mẫu Singleton:** Sử dụng mẫu Singleton để đảm bảo rằng chỉ có một đối tượng quản lý đơn hàng duy nhất trong hệ thống.

```
// Mẫu Singleton: Đảm bảo chỉ có một đối tượng quản lý đơn hàng duy nhất trong hệ thống
public class OrderManager {
    private static OrderManager instance;

    private OrderManager() {
        // Khởi tạo các tài nguyên và dữ liệu cho quản lý đơn hàng
    }

    public static OrderManager getInstance() {
        if (instance == null) {
            instance = new OrderManager();
        }
        return instance;
    }

    // Các phương thức quản lý đơn hàng
    public void addOrder(Order order) {
        // Thêm đơn hàng vào hệ thống
    }

    public void removeOrder(Order order) {
        // Xóa đơn hàng khỏi hệ thống
    }

    public void processOrders() {
        // Xử lý các đơn hàng trong hệ thống
    }
}
```

4. Quản lý giỏ hàng:

- **Mẫu Singleton:** Sử dụng mẫu Singleton để đảm bảo rằng chỉ có một đối tượng quản lý giỏ hàng duy nhất trong hệ thống.

```
public class CartManager {
    private static CartManager instance;
    private List<Book> cartItems;

    private CartManager() {
        cartItems = new ArrayList<>();
    }

    public static CartManager getInstance() {
        if (instance == null) {
            instance = new CartManager();
        }
        return instance;
    }

    public void addItem(Book book) {
        cartItems.add(book);
    }

    public void removeItem(Book book) {
        cartItems.remove(book);
    }

    public void displayCartItems() {
        for (Book book : cartItems) {
            System.out.println(book.getTitle());
        }
    }
}
```

5. Quản lý thanh toán:

- **Mẫu Factory:** Sử dụng mẫu Factory để tạo ra các đối tượng thanh toán từ các lớp con cụ thể, ví dụ như thanh toán bằng thẻ tín dụng, thanh toán bằng ví điện tử, v.v.

```
// Mẫu Factory: Tạo ra các đối tượng thanh toán từ các lớp con cụ thể
public abstract class Payment {
    public abstract void processPayment();
}

public class CreditCardPayment extends Payment {
    @Override
    public void processPayment() {
        System.out.println("Processing credit card payment.");
    }
}

public class EWalletPayment extends Payment {
    @Override
    public void processPayment() {
        System.out.println("Processing e-wallet payment.");
    }
}

public class PaymentFactory {
    public Payment createPayment(String paymentType) {
        if (paymentType.equals("creditCard")) {
            return new CreditCardPayment();
        } else if (paymentType.equals("eWallet")) {
            return new EWalletPayment();
        }
        return null;
    }
}
```

- **Mẫu Singleton:** Sử dụng mẫu Singleton để đảm bảo rằng chỉ có một đối tượng quản lý thanh toán duy nhất trong hệ thống.

```
// Mẫu Singleton: Đảm bảo chỉ có một đối tượng quản lý thanh toán duy nhất trong hệ thống
public class PaymentManager {
    private static PaymentManager instance;

    private PaymentManager() {
        // Khởi tạo các tài nguyên và dữ liệu cho quản lý thanh toán
    }

    public static PaymentManager getInstance() {
        if (instance == null) {
            instance = new PaymentManager();
        }
        return instance;
    }

    // Các phương thức quản lý thanh toán
    public void processPayment(Payment payment) {
        // Xử lý thanh toán trong hệ thống
        payment.processPayment();
    }
}
```