**WEEK-1**

**Design Patterns and Principles**

**Superset ID: 6419740**

**Exercise 1: Implementing the Singleton Pattern**

**Code:**

**Logger.java:**

public class Logger {

    private static Logger singleInstance;

    private Logger() {

        System.out.println("Logger created");

    }

    public static Logger getInstance() {

        if (singleInstance == null) {

            singleInstance = new Logger();

        }

        return singleInstance;

    }

    public void log(String message) {

        System.out.println("Log: " + message);

    }

}

**Main.java:**

public class Main {

    public static void main(String[] args) {

        Logger log1 = Logger.getInstance();

        log1.log("Start the application");

        Logger log2 = Logger.getInstance();

        log2.log("Continue the application");

        if (log1 == log2) {

            System.out.println("Singleton is created");

        } else {

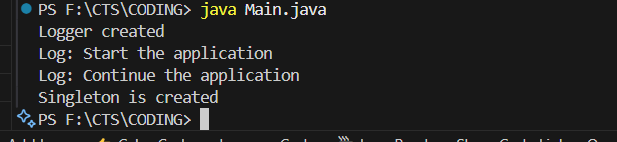
            System.out.println("Singleton is not created");

        }

    }

}

**Output:**



**Exercise 2: Implementing the Factory Method Pattern**

**Code:**

**Main.java**

package ex2;

interface Document {

    void open();

}

class WordDocument implements Document {

    public void open() {

        System.out.println("Open Word Doc");

    }

}

class PdfDocument implements Document {

    public void open() {

        System.out.println("Open PDF Doc");

    }

}

class ExcelDocument implements Document {

    public void open() {

        System.out.println("Open Excel Doc");

    }

}

abstract class DocumentFactory {

    public abstract Document createDocument();

}

class WordFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

class PdfFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

class ExcelFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

public class Main {

    public static void main(String[] args) {

        DocumentFactory wordGen = new WordFactory();

        Document doc1 = wordGen.createDocument();

        doc1.open();

        DocumentFactory pdfGen = new PdfFactory();

        Document doc2 = pdfGen.createDocument();

        doc2.open();

        DocumentFactory excelGen = new ExcelFactory();

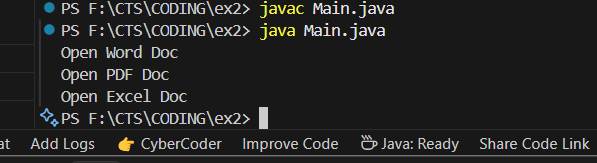
        Document doc3 = excelGen.createDocument();

        doc3.open();

    }

}

**Output:**



**Exercise 3: Implementing the Builder Pattern**

**Code:**

package ex3;

public class Main {

    public static void main(String[] args) {

        Computer basic = new Computer.Builder()

            .setCpu("Intel i3")

            .setRam("4GB")

            .setStorage("256GB SSD")

            .build();

        Computer gaming = new Computer.Builder()

            .setCpu("Intel i9")

            .setRam("32GB")

            .setStorage("1TB SSD")

            .setGraphics("NVIDIA RTX 4080")

            .build();

        System.out.println("Basic PC: " + basic);

        System.out.println("Gaming PC: " + gaming);

    }

}

class Computer {

    private String cpu;

    private String ram;

    private String storage;

    private String graphics;

    private Computer(Builder builder) {

        this.cpu = builder.cpu;

        this.ram = builder.ram;

        this.storage = builder.storage;

        this.graphics = builder.graphics;

    }

    public String toString() {

        return "CPU: " + cpu + ", RAM: " + ram + ", Storage: " + storage + ", Graphics: " + graphics;

    }

    public static class Builder {

        private String cpu;

        private String ram;

        private String storage;

        private String graphics;

        public Builder setCpu(String cpu) {

            this.cpu = cpu;

            return this;

        }

        public Builder setRam(String ram) {

            this.ram = ram;

            return this;

        }

        public Builder setStorage(String storage) {

            this.storage = storage;

            return this;

        }

        public Builder setGraphics(String graphics) {

            this.graphics = graphics;

            return this;

        }

        public Computer build() {

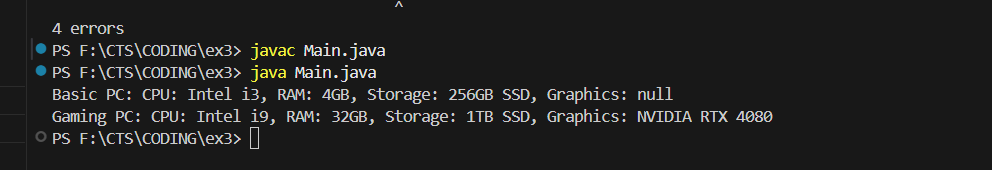
            return new Computer(this);

        }

    }

}

**Output:**

****

**Exercise 4: Implementing the Adapter Pattern**

**Code:**

package ex4;

public class Main {

    public static void main(String[] args) {

        PaymentProcessor paypal = new PayPalAdapter();

        paypal.processPayment(1200);

        PaymentProcessor stripe = new StripeAdapter();

        stripe.processPayment(2500);

    }

}

interface PaymentProcessor {

    void processPayment(double amount);

}

class PayPalGateway {

    public void makeTransaction(double amt) {

        System.out.println("PayPal processed: $" + amt);

    }

}

class StripeGateway {

    public void pay(double money) {

        System.out.println("Stripe processed: $" + money);

    }

}

class PayPalAdapter implements PaymentProcessor {

    private PayPalGateway paypal = new PayPalGateway();

    public void processPayment(double amount) {

        paypal.makeTransaction(amount);

    }

}

class StripeAdapter implements PaymentProcessor {

    private StripeGateway stripe = new StripeGateway();

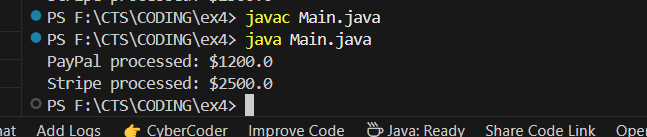
    public void processPayment(double amount) {

        stripe.pay(amount);

    }

}

**Output:**



**Exercise 5: Implementing the Decorator Pattern**

**Code:**

**Notifier.java**

package ex5;

public interface Notifier {

    void send(String message);

}

class EmailNotifier implements Notifier {

    public void send(String message) {

        System.out.println("Email: " + message);

    }

}

abstract class NotifierDecorator implements Notifier {

    protected Notifier wrappee;

    public NotifierDecorator(Notifier notifier) {

        this.wrappee = notifier;

    }

    public void send(String message) {

        wrappee.send(message);

    }

}

class SMSNotifierDecorator extends NotifierDecorator {

    public SMSNotifierDecorator(Notifier notifier) {

        super(notifier);

    }

    public void send(String message) {

        super.send(message);

        System.out.println("SMS: " + message);

    }

}

class SlackNotifierDecorator extends NotifierDecorator {

    public SlackNotifierDecorator(Notifier notifier) {

        super(notifier);

    }

    public void send(String message) {

        super.send(message);

        System.out.println("Slack: " + message);

    }

}

**Main.java:**

package ex5;

public class Main {

    public static void main(String[] args) {

        Notifier notifier = new EmailNotifier();

        Notifier sms = new SMSNotifierDecorator(notifier);

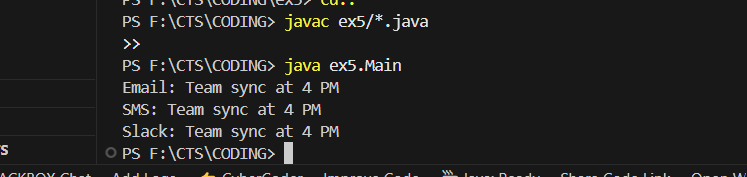
        Notifier slack = new SlackNotifierDecorator(sms);

        slack.send("Team sync at 4 PM");

    }

}

**Output:**



**Exercise 6: Implementing the Proxy Pattern**

**Code:**

**Images.java:**

package ex6;

public interface Image {

    void display();

}

class RealImage implements Image {

    private String filename;

    public RealImage(String filename) {

        this.filename = filename;

        loadFromServer();

    }

    private void loadFromServer() {

        System.out.println("Loading " + filename + " from server...");

    }

    public void display() {

        System.out.println("Displaying " + filename);

    }

}

class ProxyImage implements Image {

    private String filename;

    private RealImage realImage;

    public ProxyImage(String filename) {

        this.filename = filename;

    }

    public void display() {

        if (realImage == null) {

            realImage = new RealImage(filename);

        }

        realImage.display();

    }

}

**Main.java:**

package ex6;

public class Main {

    public static void main(String[] args) {

        Image image1 = new ProxyImage("sunset.jpg");

        Image image2 = new ProxyImage("beach.png");

        image1.display();

        System.out.println();

        image1.display();

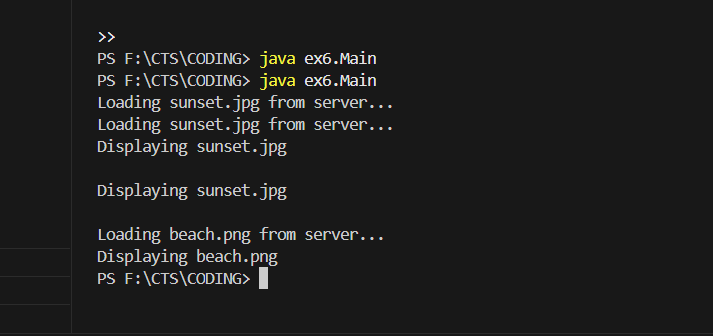
        System.out.println();

        image2.display();

    }

}

**Output:**

****

**Exercise 7: Implementing the Observer Pattern**

**Code:**

**StockSystem.java**

package ex7;

import java.util.\*;

interface Stock {

    void register(Observer obs);

    void deregister(Observer obs);

    void notifyObservers();

    void setPrice(double price);

}

interface Observer {

    void update(double price);

}

class StockMarket implements Stock {

    private List<Observer> observers = new ArrayList<>();

    private double stockPrice;

    public void register(Observer obs) {

        observers.add(obs);

    }

    public void deregister(Observer obs) {

        observers.remove(obs);

    }

    public void notifyObservers() {

        for (Observer obs : observers) {

            obs.update(stockPrice);

        }

    }

    public void setPrice(double price) {

        stockPrice = price;

        notifyObservers();

    }

}

class MobileApp implements Observer {

    public void update(double price) {

        System.out.println("Mobile App - New Stock Price: $" + price);

    }

}

class WebApp implements Observer {

    public void update(double price) {

        System.out.println("Web App - Updated Stock Price: $" + price);

    }

}

**Main.java:**

package ex7;

public class Main {

    public static void main(String[] args) {

        StockMarket market = new StockMarket();

        Observer mobile = new MobileApp();

        Observer web = new WebApp();

        market.register(mobile);

        market.register(web);

        market.setPrice(1050.50);

        market.setPrice(1085.75);

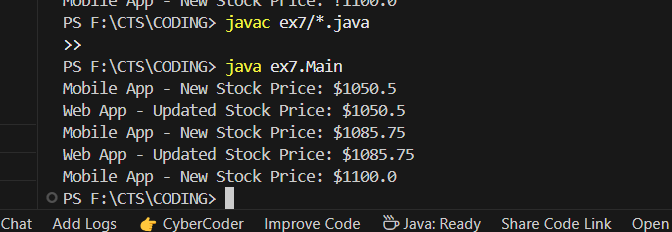
        market.deregister(web);

        market.setPrice(1100.00);

    }

}

**Output:**

****

**Exercise 8: Implementing the Strategy Pattern  
Code:**

**PaymentStrategy.java:**

package ex8;

interface PaymentStrategy {

void pay(double amount);

}

class CreditCardPayment implements PaymentStrategy {

public void pay(double amount) {

System.out.println("Paid ₹" + amount + " using Credit Card");

}

}

class PayPalPayment implements PaymentStrategy {

public void pay(double amount) {

System.out.println("Paid ₹" + amount + " using PayPal");

}

}

class PaymentContext {

private PaymentStrategy strategy;

public PaymentContext(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void setStrategy(PaymentStrategy strategy) {

this.strategy = strategy;

}

public void makePayment(double amount) {

strategy.pay(amount);

}

}

**Main.java:**

package ex8;

public class Main {

public static void main(String[] args) {

PaymentContext context = new PaymentContext(new CreditCardPayment());

context.makePayment(1500);

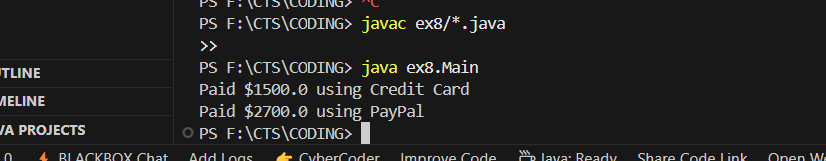
context.setStrategy(new PayPalPayment());

context.makePayment(2700);

}

}

**Output:**

****

**Exercise 9: Implementing the Command Pattern**

**Code:**

**HomeAutomation.java**

package ex9;

interface Command {

void execute();

}

class Light {

public void turnOn() {

System.out.println("Light is ON");

}

public void turnOff() {

System.out.println("Light is OFF");

}

}

class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOn();

}

}

class LightOffCommand implements Command {

private Light light;

public LightOffCommand(Light light) {

this.light = light;

}

public void execute() {

light.turnOff();

}

}

class RemoteControl {

private Command command;

public void setCommand(Command command) {

this.command = command;

}

public void pressButton() {

command.execute();

}

}

**Main.java:**

package ex9;

public class Main {

public static void main(String[] args) {

Light livingRoomLight = new Light();

Command turnOn = new LightOnCommand(livingRoomLight);

Command turnOff = new LightOffCommand(livingRoomLight);

RemoteControl remote = new RemoteControl();

remote.setCommand(turnOn);

remote.pressButton();

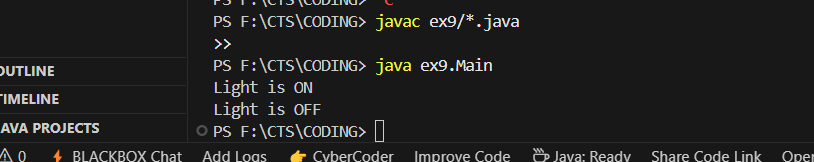
remote.setCommand(turnOff);

remote.pressButton();

}

}

**Output:**

****

**Exercise 10: Implementing the MVC Pattern**

**Code:**

**StudentSystem.java:**

package ex10;

class Student {

    private String name;

    private String id;

    private String grade;

    public Student(String name, String id, String grade) {

        this.name = name;

        this.id = id;

        this.grade = grade;

    }

    public String getName() {

        return name;

    }

    public String getId() {

        return id;

    }

    public String getGrade() {

        return grade;

    }

    public void setName(String name) {

        this.name = name;

    }

    public void setGrade(String grade) {

        this.grade = grade;

    }

}

class StudentView {

    public void displayStudentDetails(String name, String id, String grade) {

        System.out.println("Student Details");

        System.out.println("Name: " + name);

        System.out.println("ID: " + id);

        System.out.println("Grade: " + grade);

    }

}

class StudentController {

    private Student model;

    private StudentView view;

    public StudentController(Student model, StudentView view) {

        this.model = model;

        this.view = view;

    }

    public void updateView() {

        view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());

    }

    public void setStudentName(String name) {

        model.setName(name);

    }

    public void setStudentGrade(String grade) {

        model.setGrade(grade);

    }

}

**Main.java:**

package ex10;

public class Main {

    public static void main(String[] args) {

        Student student = new Student("Arun", "S123", "A");

        StudentView view = new StudentView();

        StudentController controller = new StudentController(student, view);

        controller.updateView();

        controller.setStudentName("Arun Kumar");

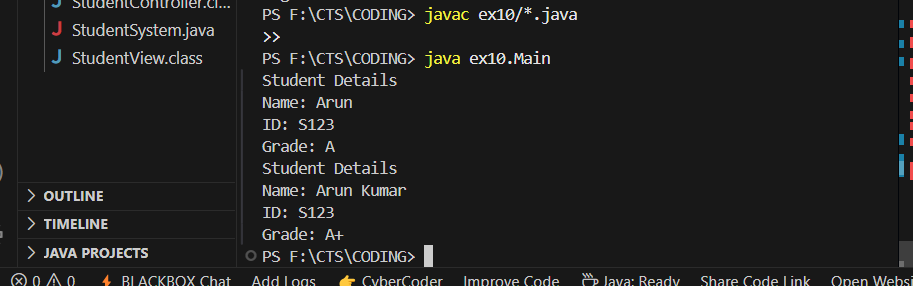
        controller.setStudentGrade("A+");

        controller.updateView();

    }

}

**Output:**



**Exercise 11: Implementing Dependency Injection**

**Code:**

**CustomerApp.java:**

package ex11;

interface CustomerRepository {

    String findCustomerById(String id);

}

class CustomerRepositoryImpl implements CustomerRepository {

    public String findCustomerById(String id) {

        return "Customer ID: " + id + ", Name: Joy, Location: Chennai";

    }

}

class CustomerService {

    private CustomerRepository repository;

    public CustomerService(CustomerRepository repository) {

        this.repository = repository;

    }

    public void showCustomer(String id) {

        String customer = repository.findCustomerById(id);

        System.out.println(customer);

    }

}

**Main.java:**

package ex11;

public class Main {

    public static void main(String[] args) {

        CustomerRepository repo = new CustomerRepositoryImpl();

        CustomerService service = new CustomerService(repo);

        service.showCustomer("CA102");

    }

}

**Output:**

