**DAY -5 ASSIGNMENT**

**BASICS OF JAVA**

**1)** **Singleton Design Pattern for BillingService**  
  
public class BillingService {  
    private static BillingService instance;  
  
    private BillingService() {  
        // Private constructor to prevent instantiation  
    }  
  
    public static BillingService getInstance() {  
        if (instance == null) {  
            synchronized (BillingService.class) {  
                if (instance == null) {  
                    instance = new BillingService();  
                }  
            }  
        }  
        return instance;  
    }  
  
    public void processPayment(String paymentDetails) {  
        // Implement payment processing logic here  
        System.out.println("Processing payment: " + paymentDetails);  
    }  
  
    public void generateInvoice(String orderDetails) {  
        // Implement invoice generation logic here  
        System.out.println("Generating invoice for: " + orderDetails);  
    }  
  
    public static void main(String[] args) {  
        BillingService billingService = BillingService.getInstance();  
  
        // Demonstrate the usage of the billing service  
        billingService.processPayment("Payment details for order #123");  
        billingService.generateInvoice("Order details for order #123");  
    }  
}

**2)** **Factory Design Pattern for Vehicle**  
  
 **1. Vehicle Interface**  
public interface Vehicle {  
    void start();  
    void accelerate();  
    void brake();  
}  
  
 **2. Concrete Vehicle Classes**

public class Car implements Vehicle {  
    @Override  
    public void start() {  
        System.out.println("Car started.");  
    }  
  
    @Override  
    public void accelerate() {  
        System.out.println("Car accelerating.");  
    }  
  
    @Override  
    public void brake() {  
        System.out.println("Car braking.");  
    }  
}  
  
public class Motorcycle implements Vehicle {  
    @Override  
    public void start() {  
        System.out.println("Motorcycle started.");  
    }  
  
    @Override  
    public void accelerate() {  
        System.out.println("Motorcycle accelerating.");  
    }  
  
    @Override  
    public void brake() {  
        System.out.println("Motorcycle braking.");  
    }  
}  
  
public class Truck implements Vehicle {  
    @Override  
    public void start() {  
        System.out.println("Truck started.");  
    }  
  
    @Override  
    public void accelerate() {  
        System.out.println("Truck accelerating.");  
    }  
  
    @Override  
    public void brake() {  
        System.out.println("Truck braking.");  
    }  
}  
  
 **3. VehicleFactory Class**

public class VehicleFactory {  
    public Vehicle createVehicle(String type) {  
        switch (type.toUpperCase()) {  
            case "CAR":  
                return new Car();  
            case "MOTORCYCLE":  
                return new Motorcycle();  
            case "TRUCK":  
                return new Truck();  
            default:  
                throw new IllegalArgumentException("Unknown vehicle type: " + type);  
        }  
    }  
}  
  
 **4. Main Class**

public class Main {  
    public static void main(String[] args) {  
        VehicleFactory factory = new VehicleFactory();  
  
        Vehicle car = factory.createVehicle("car");  
        car.start();  
        car.accelerate();  
        car.brake();  
  
        Vehicle motorcycle = factory.createVehicle("motorcycle");  
        motorcycle.start();  
        motorcycle.accelerate();  
        motorcycle.brake();  
  
        Vehicle truck = factory.createVehicle("truck");  
        truck.start();  
        truck.accelerate();  
        truck.brake();  
    }  
}  
  
**3) Abstract Factory Design Pattern for Shapes**  
 **1. Shape Interface**public interface Shape {  
    void draw();  
}  
  
 **2. Concrete Shape Classes**

public class Circle implements Shape {  
    @Override  
    public void draw() {  
        System.out.println("Drawing Circle.");  
    }  
}  
  
public class Rectangle implements Shape {  
    @Override  
    public void draw() {  
        System.out.println("Drawing Rectangle.");  
    }  
}  
  
public class Square implements Shape {  
    @Override  
    public void draw() {  
        System.out.println("Drawing Square.");  
    }  
}  
  
 **3. AbstractFactory Class**

public abstract class AbstractFactory {  
    abstract Shape getShape(String shapeType);  
}  
  
 **4. ShapeFactory Class**

public class ShapeFactory extends AbstractFactory {  
    @Override  
    public Shape getShape(String shapeType) {  
        if (shapeType == null) {  
            return null;  
        }  
        switch (shapeType.toUpperCase()) {  
            case "CIRCLE":  
                return new Circle();  
            case "RECTANGLE":  
                return new Rectangle();  
            case "SQUARE":  
                return new Square();  
            default:  
                return null;  
        }  
    }  
}  
  
 **5. FactoryProducer Class**

public class FactoryProducer {  
    public static AbstractFactory getFactory() {  
        return new ShapeFactory();  
    }  
}  
  
 **6. AbstractFactoryPatternDemo Class**

public class AbstractFactoryPatternDemo {  
    public static void main(String[] args) {  
        AbstractFactory shapeFactory = FactoryProducer.getFactory();  
  
        Shape circle = shapeFactory.getShape("CIRCLE");  
        circle.draw();  
  
        Shape rectangle = shapeFactory.getShape("RECTANGLE");  
        rectangle.draw();  
  
        Shape square = shapeFactory.getShape("SQUARE");  
        square.draw();  
    }  
}  
  
**4) Immutable Employee Class**  
  
import java.util.Date;  
  
public final class Employee {  
    private final String firstName;  
    private final String lastName;  
    private final Date dateOfBirth;  
    private final int employeeId;  
    private final Date joiningDate;  
    private final double salary;  
  
    public Employee(String firstName, String lastName, Date dateOfBirth, int employeeId, Date joiningDate, double salary) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.dateOfBirth = new Date(dateOfBirth.getTime());  
        this.employeeId = employeeId;  
        this.joiningDate = new Date(joiningDate.getTime());  
        this.salary = salary;  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
  
    public String getLastName() {  
        return lastName;  
    }  
  
    public Date getDateOfBirth() {  
        return new Date(dateOfBirth.getTime());  
    }  
  
    public int getEmployeeId() {  
        return employeeId;  
    }  
  
    public Date getJoiningDate() {  
        return new Date(joiningDate.getTime());  
    }  
  
    public double getSalary() {  
        return salary;  
    }  
  
    public static void main(String[] args) {  
        Date dob = new Date(1990, 1, 1);  
        Date joiningDate = new Date(2020, 1, 1);  
        Employee employee = new Employee("John", "Doe", dob, 12345, joiningDate, 50000.0);  
  
        System.out.println("Employee Details:");  
        System.out.println("First Name: " + employee.getFirstName());  
        System.out.println("Last Name: " + employee.getLastName());  
        System.out.println("Date of Birth: " + employee.getDateOfBirth());  
        System.out.println("Employee ID: " + employee.getEmployeeId());  
        System.out.println("Joining Date: " + employee.getJoiningDate());  
        System.out.println("Salary: " + employee.getSalary());  
    }  
}