# Day 5

1)

package day5;

class BillingService {

    // 1. The instance attribute that will hold the single instance of BillingService.

    private static BillingService instance;

    // 2. Private constructor to prevent direct instantiation.

    private BillingService() {

        // Private constructor to prevent instantiation from other classes.

    }

    // 3. Method to return the instance of BillingService (Lazy Initialization).

    public static BillingService getInstance() {

        if (instance == null) {

            instance = new BillingService();

        }

        return instance;

    }

    // 4. Method to process payment.

    public void processPayment(String paymentDetails) {

        // Simulating payment processing

        System.out.println("Processing payment with details: " + paymentDetails);

        // Additional logic for payment processing would go here.

    }

    // 5. Method to generate an invoice based on order details.

    public void generateInvoice(String orderDetails) {

        // Simulating invoice generation

        System.out.println("Generating invoice for order: " + orderDetails);

        // Additional logic for invoice generation would go here.

    }

}

public class BillingServiceTester {

    public static void main(String[] args) {

        // Get the instance of BillingService

        BillingService billingService1 = BillingService.getInstance();

        // Process payment and generate invoice using the same instance

        billingService1.processPayment("Payment ID: 12345, Amount: $100");

        billingService1.generateInvoice("Order ID: 56789, Total: $100");

        // Get another reference to BillingService to check Singleton behavior

        BillingService billingService2 = BillingService.getInstance();

        // Check if both references point to the same instance

        if (billingService1 == billingService2) {

            System.out.println("Both references point to the same instance.");

        } else {

            System.out.println("Different instances exist.");

        }

    }

}

Processing payment with details: Payment ID: 12345, Amount: $100

Generating invoice for order: Order ID: 56789, Total: $100

Both references point to the same instance.

2)

package day5;

// Vehicle interface

interface Vehicle {

    void start();

    void accelerate();

    void brake();

}

// Concrete class for Car

class Car implements Vehicle {

    @Override

    public void start() {

        System.out.println("Car is starting.");

    }

    @Override

    public void accelerate() {

        System.out.println("Car is accelerating.");

    }

    @Override

    public void brake() {

        System.out.println("Car is braking.");

    }

}

// Concrete class for Motorcycle

class Motorcycle implements Vehicle {

    @Override

    public void start() {

        System.out.println("Motorcycle is starting.");

    }

    @Override

    public void accelerate() {

        System.out.println("Motorcycle is accelerating.");

    }

    @Override

    public void brake() {

        System.out.println("Motorcycle is braking.");

    }

}

// Concrete class for Truck

class Truck implements Vehicle {

    @Override

    public void start() {

        System.out.println("Truck is starting.");

    }

    @Override

    public void accelerate() {

        System.out.println("Truck is accelerating.");

    }

    @Override

    public void brake() {

        System.out.println("Truck is braking.");

    }

}

// Factory class to create different types of vehicles

class VehicleFactory {

    // Method to create vehicle based on input type

    public Vehicle createVehicle(String type) {

        if (type == null) {

            return null;

        }

        if (type.equalsIgnoreCase("car")) {

            return new Car();

        } else if (type.equalsIgnoreCase("motorcycle")) {

            return new Motorcycle();

        } else if (type.equalsIgnoreCase("truck")) {

            return new Truck();

        }

        return null;

    }

}

public class FactoryImplementation {

    public static void main(String[] args) {

        // Create a VehicleFactory instance

        VehicleFactory vehicleFactory = new VehicleFactory();

        // Create a Car and call its methods

        Vehicle car = vehicleFactory.createVehicle("car");

        car.start();

        car.accelerate();

        car.brake();

        System.out.println(); // Add a newline between outputs

        // Create a Motorcycle and call its methods

        Vehicle motorcycle = vehicleFactory.createVehicle("motorcycle");

        motorcycle.start();

        motorcycle.accelerate();

        motorcycle.brake();

        System.out.println(); // Add a newline between outputs

        // Create a Truck and call its methods

        Vehicle truck = vehicleFactory.createVehicle("truck");

        truck.start();

        truck.accelerate();

        truck.brake();

    }

}

Car is starting.

Car is accelerating.

Car is braking.

Motorcycle is starting.

Motorcycle is accelerating.

Motorcycle is braking.

Truck is starting.

Truck is accelerating.

Truck is braking.

3)

package day5;

// Shape interface

interface Shape {

    void draw();

}

// Concrete class for Circle

class Circle implements Shape {

    @Override

    public void draw() {

        System.out.println("Drawing a Circle");

    }

}

// Concrete class for Rectangle

class Rectangle implements Shape {

    @Override

    public void draw() {

        System.out.println("Drawing a Rectangle");

    }

}

// Concrete class for Square

class Square implements Shape {

    @Override

    public void draw() {

        System.out.println("Drawing a Square");

    }

}

// AbstractFactory class that declares the method for creating shapes

abstract class AbstractFactory {

    abstract Shape getShape(String shapeType);

}

// Concrete Factory class that implements AbstractFactory

class ShapeFactory extends AbstractFactory {

    @Override

    public Shape getShape(String shapeType) {

        if (shapeType == null) {

            return null;

        }

        if (shapeType.equalsIgnoreCase("CIRCLE")) {

            return new Circle();

        } else if (shapeType.equalsIgnoreCase("RECTANGLE")) {

            return new Rectangle();

        } else if (shapeType.equalsIgnoreCase("SQUARE")) {

            return new Square();

        }

        return null;

    }

}

// FactoryProducer to get the appropriate factory based on the requirement

class FactoryProducer {

    public static AbstractFactory getFactory() {

        return new ShapeFactory();

    }

}

public class AbstractFactoryImplementation {

    public static void main(String[] args) {

        // Get the ShapeFactory

        AbstractFactory shapeFactory = FactoryProducer.getFactory();

        // Create and draw a Circle

        Shape circle = shapeFactory.getShape("CIRCLE");

        circle.draw();

        // Create and draw a Rectangle

        Shape rectangle = shapeFactory.getShape("RECTANGLE");

        rectangle.draw();

        // Create and draw a Square

        Shape square = shapeFactory.getShape("SQUARE");

        square.draw();

    }

}

Drawing a Circle

Drawing a Rectangle

Drawing a Square

4)

package day5;

import java.util.Date;

class Employee {

    // 1. Declare the attributes as final and private to ensure immutability

    private final String firstName;

    private final String lastName;

    private final Date dateOfBirth;

    private final int employeeId;

    private final Date joiningDate;

    private final double salary;

    // 2. Constructor that initializes all attributes

    public Employee(String firstName, String lastName, Date dateOfBirth, int employeeId, Date joiningDate, double salary) {

        // 3. Assign values to fields

        this.firstName = firstName;

        this.lastName = lastName;

        this.dateOfBirth = new Date(dateOfBirth.getTime());  // Create a new Date to ensure immutability

        this.employeeId = employeeId;

        this.joiningDate = new Date(joiningDate.getTime()); // Create a new Date to ensure immutability

        this.salary = salary;

    }

    // 4. Getter methods for all attributes, ensuring immutability

    public String getFirstName() {

        return firstName;

    }

    public String getLastName() {

        return lastName;

    }

    public Date getDateOfBirth() {

        return new Date(dateOfBirth.getTime());  // Return a copy to prevent modification

    }

    public int getEmployeeId() {

        return employeeId;

    }

    public Date getJoiningDate() {

        return new Date(joiningDate.getTime());  // Return a copy to prevent modification

    }

    public double getSalary() {

        return salary;

    }

}

public class Immutable {

    public static void main(String[] args) {

        // Create an instance of Employee

        Date dob = new Date(1985 - 1900, 5 - 1, 15);  // Date of Birth (15th May 1985)

        Date joiningDate = new Date(2010 - 1900, 8 - 1, 10);  // Joining Date (10th August 2010)

        Employee employee = new Employee("John", "Doe", dob, 12345, joiningDate, 5000.00);

        // Retrieve and print employee details

        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());

        // Demonstrate immutability by trying to change the Date of Birth (this will not affect the original object)

        Date dobCopy = employee.getDateOfBirth();

        dobCopy.setYear(2000 - 1900);  // Modify the copied Date object

        System.out.println("Modified Date of Birth (on copied object): " + dobCopy);

        // Verify that the original Date of Birth is not modified

        System.out.println("Original Date of Birth after modification: " + employee.getDateOfBirth());

    }

}

Employee Details:

First Name: John

Last Name: Doe

Date of Birth: Wed May 15 00:00:00 IST 1985

Employee ID: 12345

Joining Date: Tue Aug 10 00:00:00 IST 2010

Salary: 5000.0

Modified Date of Birth (on copied object): Mon May 15 00:00:00 IST 2000

Original Date of Birth after modification: Wed May 15 00:00:00 IST 1985