**WEEK 1**

**Design principles & Patterns**

**Exercise 1: Implementing the Singleton Pattern**

Code:

public class SingletonExample {

    private SingletonExample() {

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

    }

    private static SingletonExample instance;

    public static synchronized SingletonExample getInstance() {

        if (instance == null) {

            instance = new SingletonExample();

        }

        return instance;

    }

    public void showMessage(String message) {

        System.out.println("[Singleton] " + message);

    }

    public static void main(String[] args) {

        SingletonExample obj1 = SingletonExample.getInstance();

        SingletonExample obj2 = SingletonExample.getInstance();

        System.out.println("Object 1 HashCode: " + obj1.hashCode());

        System.out.println("Object 2 HashCode: " + obj2.hashCode());

        if (obj1 == obj2) {

            System.out.println("Only one instance exists.");

        } else {

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

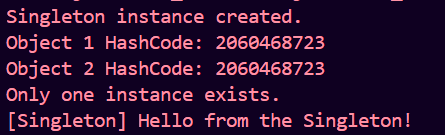
        }

        obj1.showMessage("Hello from the Singleton!");

    }

}

Output:



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

Code

*// Product interface*

interface Animal {

    void speak();

}

*// Concrete products*

class Dog implements Animal {

    public void speak() {

        System.out.println("Woof!");

    }

}

class Cat implements Animal {

    public void speak() {

        System.out.println("Meow!");

    }

}

*// Creator abstract class*

abstract class AnimalFactory {

    public abstract Animal createAnimal();

}

*// Concrete factories*

class DogFactory extends AnimalFactory {

    public Animal createAnimal() {

        return new Dog();

    }

}

class CatFactory extends AnimalFactory {

    public Animal createAnimal() {

        return new Cat();

    }

}

*// Client*

public class FactoryMethodExample {

    public static void main(String[] args) {

        AnimalFactory factory1 = new DogFactory();

        Animal animal1 = factory1.createAnimal();

        animal1.speak();

        AnimalFactory factory2 = new CatFactory();

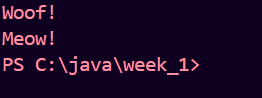
        Animal animal2 = factory2.createAnimal();

        animal2.speak();

    }

}

Output:



**Data structures and Algorithms**

**Exercise 2: E-commerce Platform Search Function**

Code:

import java.util.*\**;

public class EcommerceSearch {

    public static List<String> searchProducts(List<String> products, String keyword) {

        List<String> result = new ArrayList<>();

        String lowerKeyword = keyword.toLowerCase();

        for (String product : products) {

            if (product.toLowerCase().contains(lowerKeyword)) {

                result.add(product);

            }

        }

        return result;

    }

    public static void main(String[] args) {

        List<String> products = Arrays.asList(

            "Apple iPhone 15", "Samsung Galaxy S24", "Sony Headphones",

            "Asus Laptop", "Apple MacBook Pro", "Dell Monitor"

        );

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter search keyword: ");

        String keyword = scanner.nextLine();

        List<String> results = searchProducts(products, keyword);

        if (results.isEmpty()) {

            System.out.println("No matching products found.");

        } else {

            System.out.println("Matching products:");

            for (String product : results) {

                System.out.println("- " + product);

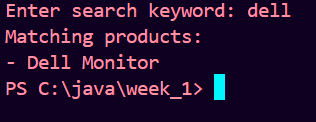
            }

        }

    }

}

Output:



**Exercise 7: Financial Forecasting**

Code:

import java.util.*\**;

public class FinancialForecasting {

    public static List<Double> forecast(List<Double> revenueData, int monthsToForecast, int windowSize) {

        List<Double> forecast = new ArrayList<>();

        int dataSize = revenueData.size();

        for (int i = 0; i < monthsToForecast; i++) {

            int start = Math.max(0, dataSize - windowSize);

            List<Double> window = revenueData.subList(start, dataSize);

            double sum = 0;

            for (double val : window) {

                sum += val;

            }

            double avg = sum / window.size();

            forecast.add(avg);

            revenueData.add(avg);

            dataSize++;

        }

        return forecast;

    }

    public static void main(String[] args) {

        List<Double> pastRevenue = Arrays.asList(

            10000.0, 12000.0, 11000.0, 13000.0, 12500.0, 14000.0,

            15000.0, 15500.0, 16000.0, 15800.0, 16500.0, 17000.0

        );

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of months to forecast: ");

        int months = scanner.nextInt();

        System.out.print("Enter moving average window size: ");

        int windowSize = scanner.nextInt();

        List<Double> forecastedRevenue = forecast(new ArrayList<>(pastRevenue), months, windowSize);

        System.out.println("Forecasted Revenue:");

        for (int i = 0; i < forecastedRevenue.size(); i++) {

            System.out.printf("Month %d: %.2f\n", i + 1, forecastedRevenue.get(i));

        }

    }

}

Output:

