Name-Amarjeet raj

Uid-22BCS17104

SECTION-636/B

SUBJECT-JAVA ASSIGNMENT

Q.1) The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.

You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2.

For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.

Return an array ans of length nums1.length such that ans[i] is the next greater element as described above.

Hint:

Input: nums1 = [4,1,2], nums2 = [1,3,4,2]

Output: [-1,3,-1]

Explanation: The next greater element for each value of nums1 is as follows:

- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.

- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

Ans- import java.util.*;

```
public class NextGreaterElement {
    public int[] nextGreaterElement(int[] nums1, int[] nums2) {
```

```
    // Stack for maintaining decreasing order
    Stack<Integer> stack = new Stack<>();
    // Map to store next greater element for each number in nums2
    Map<Integer, Integer> nextGreater = new HashMap<>();

    // Traverse nums2 from right to left
    for (int i = nums2.length - 1; i >= 0; i--) {
        int num = nums2[i];
        // Maintain a decreasing stack
        while (!stack.isEmpty() && stack.peek() <= num) {
            stack.pop();
        }
        // If stack is empty, there's no greater element
        nextGreater.put(num, stack.isEmpty() ? -1 : stack.peek());
        // Push current number onto stack
        stack.push(num);
    }

    // Construct result for nums1 using the map
    int[] result = new int[nums1.length];
    for (int i = 0; i < nums1.length; i++) {
        result[i] = nextGreater.get(nums1[i]);
    }

    return result;
}

// For testing
```

```
    public static void main(String[] args) {
        NextGreaterElement solver = new NextGreaterElement();
        int[] nums1 = {4, 1, 2};
        int[] nums2 = {1, 3, 4, 2};
        System.out.println(Arrays.toString(solver.nextGreaterElement(nums1, nums2)));
        // Output: [-1, 3, -1]
    }
}
```

Q.2) Develop a Java program showcasing the concept of inheritance. Create a base class and a derived class with appropriate methods and fields.

Ans-

```
// Base class
class Animal {
    String name;
    int age;

    // Constructor
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method to display info
    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age + " years");
    }
```

```
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

// Derived class
class Dog extends Animal {
    String breed;

    // Constructor using super to call base class constructor
    public Dog(String name, int age, String breed) {
        super(name, age); // Call the Animal constructor
        this.breed = breed;
    }

    // Overriding method
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }

    // New method specific to Dog
    public void displayBreed() {
        System.out.println("Breed: " + breed);
    }
}

// Main class to test inheritance
```

```
public class InheritanceDemo {
    public static void main(String[] args) {
        Dog myDog = new Dog("Buddy", 3, "Golden Retriever");

        myDog.displayInfo();    // Inherited from Animal
        myDog.makeSound();      // Overridden in Dog
        myDog.displayBreed();   // Specific to Dog
    }
}
```

Q.3) Implement a Java program that uses method overloading to perform different mathematical operations.

Ans-
```
public class MathOperations {

    // Method to add two integers
    public int calculate(int a, int b) {
        return a + b;
    }

    // Method to add three integers
    public int calculate(int a, int b, int c) {
        return a + b + c;
    }

    // Method to multiply two doubles
    public double calculate(double a, double b) {
        return a * b;
    }
```

```
    // Method to find power: a^b
    public double calculate(double base, int exponent) {
        return Math.pow(base, exponent);
    }

    public static void main(String[] args) {
        MathOperations math = new MathOperations();

        System.out.println("Addition of 2 integers: " + math.calculate(5, 10));            // 15
        System.out.println("Addition of 3 integers: " + math.calculate(2, 4, 6));          // 12
        System.out.println("Multiplication of 2 doubles: " + math.calculate(3.5, 2.0));    // 7.0
        System.out.println("Power of number: " + math.calculate(2.0, 3));                  // 8.0
    }
}
```

Q.4) Define an interface in Java and create a class that implements it, demonstrating the concept of abstraction.

Ans-

```
// Define an interface
interface Shape {
    // Abstract method
    double calculateArea();
}

// Implementing class
class Circle implements Shape {
    private double radius;

    // Constructor
```

```
    public Circle(double radius) {
        this.radius = radius;
    }

    // Implementation of abstract method
    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    // Additional method
    public void display() {
        System.out.println("Circle with radius: " + radius);
    }
}

// Main class to test abstraction
public class InterfaceDemo {
    public static void main(String[] args) {
        Shape shape = new Circle(5.0); // Using interface reference

        // Call method defined in interface
        System.out.println("Area of the circle: " + shape.calculateArea());

        // To access Circle-specific methods, we can cast if needed
        if (shape instanceof Circle) {
            ((Circle) shape).display();
        }
```

```
    }
}
```

Q.5) Create a custom exception class in Java. Write a program that throws this custom exception in a specific scenario.

Ans-

```
public class CustomExceptionDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(1000.0);

        try {
            System.out.println("Attempting to withdraw 1200...");
            account.withdraw(1200.0);
        } catch (InsufficientFundsException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }

        try {
            System.out.println("\nAttempting to withdraw 500...");
            account.withdraw(500.0);
        } catch (InsufficientFundsException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```