Lab mst assignment

NAME:SUDHANSU KUMAR

UID:22BCS16718

SECTION:631_AS

QUESTION 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.

Code:

```
import java.util.HashMap;

import java.util.Map;

import java.util.Stack;

public class NextGreaterElement {
```

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

        Map<Integer, Integer> nextGreaterMap = new HashMap<>();
        Stack<Integer> stack = new Stack<>();
for (int num : nums2) {
            while (!stack.isEmpty() && num > stack.peek()) {
                nextGreaterMap.put(stack.pop(), num);
            }

            stack.push(num);
        }

  while (!stack.isEmpty()) {
            nextGreaterMap.put(stack.pop(), -1);
        }
  int[] result = new int[nums1.length];
        for (int i = 0; i < nums1.length; i++) {
            result[i] = nextGreaterMap.get(nums1[i]);
        }

        return result;
    }
public static void main(String[] args) {
        NextGreaterElement solution = new NextGreaterElement();
        int[] nums1 = {4, 1, 2};
        int[] nums2 = {1, 3, 4, 2};
        int[] result = solution.nextGreaterElement(nums1, nums2);

        for (int num : result) {
            System.out.print(num + " ");
        }
```

```
    }
}
```

Output:

QUESTION 2.

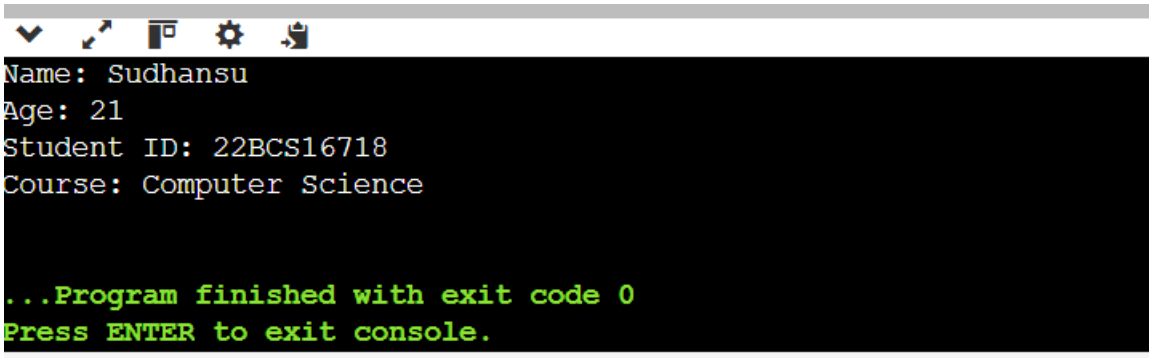
Develop a Java program showcasing the concept of inheritance. Create a base class and a derived class with appropriate methods and fields.

Code:

```
class Person {
    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
class Student extends Person {
    String studentId;
    String course;
```

```
    public Student(String name, int age, String studentId, String course) {

        super(name, age);
        this.studentId = studentId;
        this.course = course;
    }
    @Override
    public void displayInfo() {
        super.displayInfo();  // Call base class method
        System.out.println("Student ID: " + studentId);
        System.out.println("Course: " + course);
    }
}

public class InheritanceDemo {
    public static void main(String[] args) {

        Student student = new Student("Sudhansu", 21, "CSE2025001", "Computer Science");

        student.displayInfo();
    }
}
```

Output:



QUESTION 3.

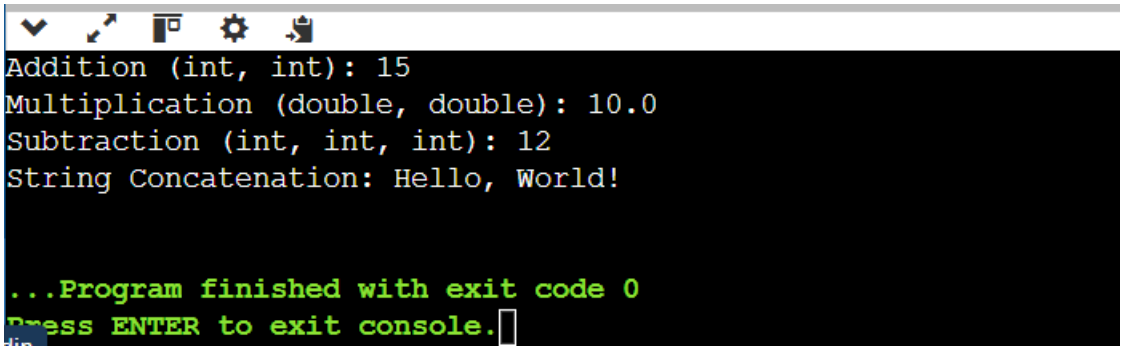
Implement a Java program that uses method overloading to perform different mathematical operations.

Code:

```
class MathOperations {
    public int operate(int a, int b) {
        return a + b;
    }

    public double operate(double a, double b) {
        return a * b;
    }

    public int operate(int a, int b, int c) {
        return a - b - c;
    }

    public String operate(String a, String b) {
        return a + b;
    }
}
```

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

        System.out.println("Addition (int, int): " + math.operate(10, 5));
        System.out.println("Multiplication (double, double): " + math.operate(2.5, 4.0));
        System.out.println("Subtraction (int, int, int): " + math.operate(20, 5, 3));
        System.out.println("String Concatenation: " + math.operate("Hello, ", "World!"));
    }
}
```

Output:



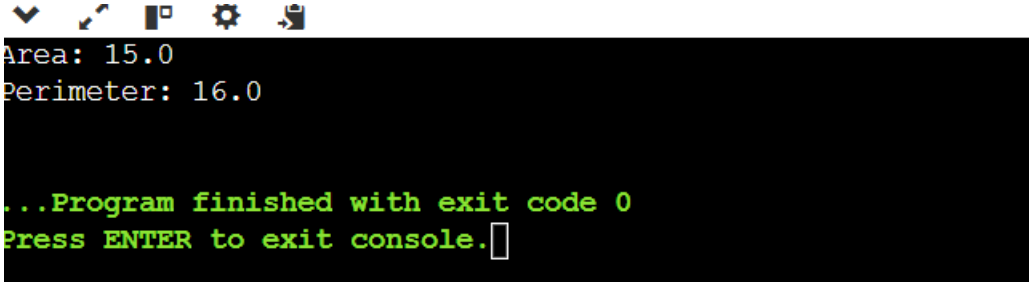
QUESTION 4. Define an interface in Java and create a class that implements it, demonstrating the concept of abstraction.

Code:

```
interface Shape {
    double area();
    double perimeter();
}

class Rectangle implements Shape {
```

```
    double length;
    double width;
Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    public double area() {
        return length * width;
    }

    public double perimeter() {
        return 2 * (length + width);
    }
}

public class Interface {
    public static void main(String[] args) {
        Rectangle rect = new Rectangle(5.0, 3.0);

        System.out.println("Area: " + rect.area());
        System.out.println("Perimeter: " + rect.perimeter());
    }
}
```

Output:



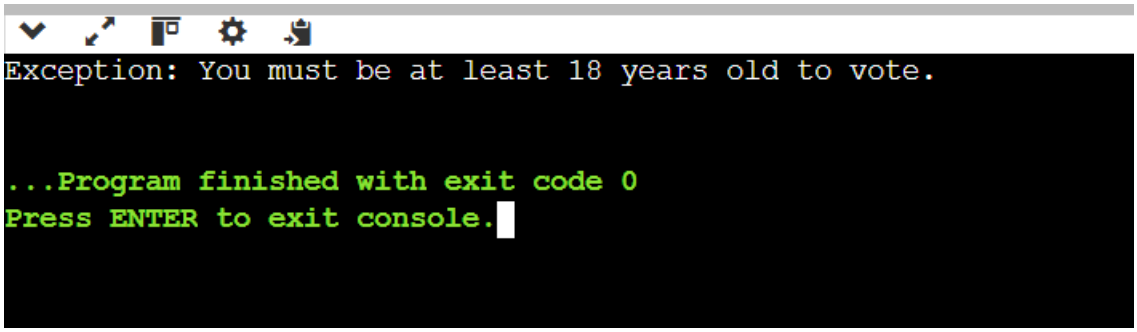
QUESTION 5. Create a custom exception class in Java. Write a program that throws this custom exception in a specific scenario.

Code:

```
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

class Voter {
    public void checkEligibility(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("You must be at least 18 years old to vote.");
        } else {
            System.out.println("You are eligible to vote.");
        }
    }
}

public class CustomExceptionDemo {
    public static void main(String[] args) {
        Voter voter = new Voter();

        try {
            voter.checkEligibility(16);
        } catch (InvalidAgeException e) {
            System.out.println("Exception: " + e.getMessage());
        }
```

```
    }
}
```

Output:


Exception: You must be at least 18 years old to vote.
...Program finished with exit code 0
Press ENTER to exit console.