1 Write a program for swapping and find a factorial value. Perform swapping without using third variable in java

```
import java.util.Scanner;

public class SwapAndFactorial {

    // Method to swap two numbers without using a third variable
    public static void swap(int a, int b) {
        System.out.println("Before swapping: a = " + a + ", b = " + b);

        // Swapping logic
        a = a + b; // a now contains the sum of both
        b = a - b; // b is now the original value of a
        a = a - b; // a is now the original value of b

        System.out.println("After swapping: a = " + a + ", b = " + b);
    }

    // Method to calculate factorial
    public static int factorial(int n) {
        if (n == 0) {
            return 1;
        }
        return n * factorial(n - 1);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input for swapping
        System.out.print("Enter first number (a): ");
        int a = scanner.nextInt();
        System.out.print("Enter second number (b): ");
        int b = scanner.nextInt();

        // Perform swapping
        swap(a, b);

        // Input for factorial calculation
        System.out.print("Enter a number to calculate factorial: ");
        int number = scanner.nextInt();

        // Calculate and print factorial
```

```
        int fact = factorial(number);
        System.out.println("Factorial of " + number + " is: " + fact);

        scanner.close();
    }
}
```

How It Works:

1. **Swapping Logic:**
    - `a = a + b;` adds both numbers.
    - `b = a - b;` subtracts `b` from the new value of `a` to get the original value of `a`.
    - `a = a - b;` subtracts the new value of `b` from the new value of `a` to get the original value of `b`.
2. **Factorial Calculation:**
    - The `factorial` method uses recursion to compute the factorial of a number. If the number is `0`, it returns `1` (since 0! is 1). Otherwise, it multiplies the number by the factorial of the number minus one.

2 Write a program to accept a number from the user through command line and display whether the given number is palindrome or not. in java

```
import java.util.Scanner;

public class PalindromeCheck {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter a number
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        // Check if the number is a palindrome
        if (isPalindrome(number)) {
            System.out.println(number + " is a palindrome.");
        } else {
            System.out.println(number + " is not a palindrome.");
        }

        scanner.close();
    }
```

```
    public static boolean isPalindrome(int num) {
        int originalNumber = num;
        int reversedNumber = 0;

        while (num != 0) {
            int digit = num % 10;
            reversedNumber = reversedNumber * 10 + digit;
            num /= 10;
        }

        return originalNumber == reversedNumber;
    }
}
```

How the Program Works:

1. **Input**: The program prompts the user to enter a number.
2. **Palindrome Check**: The `isPalindrome` function reverses the digits of the number and compares it to the original.
3. **Output**: It prints whether the number is a palindrome.

3 Write a program to accept an array of integer from the user through command line and find prime numbers from the array in java

```
import java.util.Scanner;

public class PrimeNumbersInArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        int[] numbers = new int[size];

        // Prompt the user to enter the elements of the array
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < size; i++) {
            numbers[i] = scanner.nextInt();
        }
```

```
        // Find and display prime numbers in the array
        System.out.println("Prime numbers in the array:");
        for (int num : numbers) {
            if (isPrime(num)) {
                System.out.print(num + " ");
            }
        }

        scanner.close();
    }

    // Method to check if a number is prime
    public static boolean isPrime(int num) {
        if (num <= 1) {
            return false; // 1 and numbers less than 1 are not prime
        }
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false; // If divisible by any number other than 1 and itself, it's not prime
            }
        }
        return true;
    }
}
```

How the Program Works:

1. **Input**: The program first asks the user to input the size of the array, then it prompts the user to input the elements of the array.
2. **Prime Check**: The `isPrime` method checks whether each element of the array is a prime number.
3. **Output**: The program prints out all the prime numbers from the array.

Explanation:

- The program reads an integer array from the user.
- It checks each element to see if it’s prime.
- Finally, it prints all prime numbers found in the array.

4 Create a class Stack that defines an integer stack that can hold 10 values. Perform push and pop actions in a stack in java

```
class Stack {
    private int[] stackArray;
    private int top;
    private static final int MAX_SIZE = 10;

    // Constructor to initialize the stack
    public Stack() {
        stackArray = new int[MAX_SIZE];
        top = -1; // Stack is initially empty
    }

    // Method to push an element onto the stack
    public void push(int value) {
        if (top >= MAX_SIZE - 1) {
            System.out.println("Stack Overflow! Unable to push " + value);
        } else {
            stackArray[++top] = value;
            System.out.println("Pushed " + value + " onto the stack.");
        }
    }

    // Method to pop an element from the stack
    public int pop() {
        if (top < 0) {
            System.out.println("Stack Underflow! No elements to pop.");
            return -1; // Returning -1 to indicate stack underflow
        } else {
            int poppedValue = stackArray[top--];
            System.out.println("Popped " + poppedValue + " from the stack.");
            return poppedValue;
        }
    }

    // Method to check if the stack is empty
    public boolean isEmpty() {
        return (top < 0);
    }

    // Method to check if the stack is full
    public boolean isFull() {
        return (top >= MAX_SIZE - 1);
```

```
    }

    // Method to display the stack elements
    public void display() {
        if (top < 0) {
            System.out.println("Stack is empty.");
        } else {
            System.out.print("Stack elements: ");
            for (int i = 0; i <= top; i++) {
                System.out.print(stackArray[i] + " ");
            }
            System.out.println();
        }
    }
}

public class StackDemo {
    public static void main(String[] args) {
        Stack stack = new Stack();

        // Perform stack operations
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.display();

        stack.pop();
        stack.display();

        stack.push(40);
        stack.push(50);
        stack.push(60);
        stack.push(70);
        stack.push(80);
        stack.push(90);
        stack.push(100);
        stack.push(110); // This should cause a stack overflow

        stack.display();

        stack.pop();
        stack.pop();
        stack.display();
    }
```

```
}
```

How the Program Works:

1. **Class Definition**: The `Stack` class is defined with a maximum size of 10, and it has an array `stackArray` to hold the elements. The `top` variable keeps track of the topmost element in the stack.
2. **Push Method**: The `push` method adds an element to the top of the stack if there is space. If the stack is full, it prints a stack overflow message.
3. **Pop Method**: The `pop` method removes and returns the top element from the stack. If the stack is empty, it prints a stack underflow message and returns `-1`.
4. **Utility Methods**: Methods like `isEmpty`, `isFull`, and `display` provide additional functionalities to check the stack's state and display its contents.
5. **StackDemo**: The `StackDemo` class demonstrates how the stack works by performing a series of push and pop operations.

5 Write a program to create a class Publisher with attributes publisher name and publisher id. Derive a subclass Book with attributes bookname, bookid and author name. All these data should be entered by the user. Create two methods getdata() and showdata() to display the details of book and publisher. in java

```
import java.util.Scanner;

// Publisher class
class Publisher {
    String publisherName;
    int publisherId;

    // Method to get publisher data
    void getData() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Publisher Name: ");
        publisherName = scanner.nextLine();
        System.out.print("Enter Publisher ID: ");
        publisherId = scanner.nextInt();
        scanner.nextLine(); // consume the newline
    }

    // Method to display publisher data
    void showData() {
        System.out.println("Publisher Name: " + publisherName);
        System.out.println("Publisher ID: " + publisherId);
```

```
    }
}

// Book class derived from Publisher
class Book extends Publisher {
    String bookName;
    int bookId;
    String authorName;

    // Method to get book data
    @Override
    void getData() {
        super.getData();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter Book Name: ");
        bookName = scanner.nextLine();
        System.out.print("Enter Book ID: ");
        bookId = scanner.nextInt();
        scanner.nextLine(); // consume the newline
        System.out.print("Enter Author Name: ");
        authorName = scanner.nextLine();
    }

    // Method to display book data
    @Override
    void showData() {
        super.showData();
        System.out.println("Book Name: " + bookName);
        System.out.println("Book ID: " + bookId);
        System.out.println("Author Name: " + authorName);
    }
}

// Main class to test the functionality
public class Main {
    public static void main(String[] args) {
        Book book = new Book();
        book.getData();
        System.out.println("\nDisplaying Book and Publisher Details:");
        book.showData();
    }
}
```

How the Program Works:

1. **Publisher Class**: Contains attributes `publisherName` and `publisherId`. The `getData()` method allows the user to input these values, and the `showData()` method displays them.
2. **Book Class**: Extends the `Publisher` class and adds `bookName`, `bookId`, and `authorName` as attributes. It overrides the `getData()` method to get data for both the book and publisher, and the `showData()` method to display all the information.
3. **Main Class**: The `Main` class creates an instance of the `Book` class, calls the `getData()` method to input the data, and then calls the `showData()` method to display the book and publisher details.

o/p
Enter Publisher Name: Penguin Random House
Enter Publisher ID: 101
Enter Book Name: Java Programming
Enter Book ID: 202
Enter Author Name: John Doe

Displaying Book and Publisher Details:
Publisher Name: Penguin Random House
Publisher ID: 101
Book Name: Java Programming
Book ID: 202
Author Name: John Doe

6 Write a program to create a class with two methods with same name addfunc(), one accepting two integer parameters and other accepting two double parameters. When method is called, the appropriate method should be selected depending on parameters passed(method overloading).

```
class Calculator {
    // Method to add two integers
    public int addfunc(int a, int b) {
        return a + b;
    }

    // Method to add two doubles
    public double addfunc(double a, double b) {
        return a + b;
    }

    public static void main(String[] args) {
```

```
        Calculator calc = new Calculator();

        // Calling the method with integer parameters
        int sumInt = calc.addfunc(10, 20);
        System.out.println("Sum of integers: " + sumInt);

        // Calling the method with double parameters
        double sumDouble = calc.addfunc(10.5, 20.5);
        System.out.println("Sum of doubles: " + sumDouble);
    }
}
```

Explanation:

- The `Calculator` class has two overloaded methods named `addfunc()`.
    - The first method accepts two `int` parameters and returns their sum.
    - The second method accepts two `double` parameters and returns their sum.
- The appropriate method is selected based on the parameter types when the method is called.
- In the `main()` method, the program demonstrates calling both versions of the `addfunc()` method.

7 Declare a variable called x with integer as the data type in base class and subclass. Make a method named as show() which displays the value of x in the superclass and subclass

```
// Base class (Superclass)
class SuperClass {
    // Declare an integer variable x in the superclass
    int x = 10;

    // Method to display the value of x in the superclass
    void show() {
        System.out.println("Value of x in SuperClass: " + x);
    }
}

// Subclass that extends the SuperClass
class SubClass extends SuperClass {
    // Declare an integer variable x in the subclass
    int x = 20;

    // Method to display the value of x in the subclass
```

```
    @Override
    void show() {
        // Display value of x in SuperClass
        super.show();

        // Display value of x in SubClass
        System.out.println("Value of x in SubClass: " + x);
    }
}

// Main class to test the program
public class Main {
    public static void main(String[] args) {
        // Create an object of SubClass
        SubClass obj = new SubClass();

        // Call the show() method to display values of x
        obj.show();
    }
}
```

o/p

```
Value of x in SuperClass: 10
Value of x in SubClass: 20
```

Explanation:

- **SuperClass:**
  - The base class, `SuperClass`, has an integer variable `x` initialized to `10`.
  - The `show()` method in `SuperClass` prints the value of `x`.
- **SubClass:**
  - The `SubClass` extends `SuperClass` and also declares an integer variable `x`, which is initialized to `20`.
  - The `show()` method is overridden in `SubClass` to display the value of `x` from both `SuperClass` and `SubClass`.
  - The `super.show()` call within the `show()` method of `SubClass` is used to invoke the `show()` method from the superclass, which displays the value of `x` from `SuperClass`.
- **Main Class:**

- In the `Main` class, an object of `SubClass` is created, and the `show()` method is called. This displays the value of `x` in both the superclass and subclass.

8 Write a program to calculate the area, circumference and volume for all shapes. [Perform this application using final class, abstract class and interface]

To implement a Java program that calculates the area, circumference, and volume for various shapes using `final class`, `abstract class`, and `interface`, you can design the program as follows:

1. Interface: Define an interface `Shape` with methods for area, circumference, and volume.
2. Abstract Class: Create an abstract class `TwoDimensionalShape` for shapes that have only area and circumference.
3. Final Class: Implement final classes for specific shapes, like `Circle` and `Sphere`, which extend the abstract class or implement the interface.

```
// Interface defining common methods for all shapes

interface Shape {

    double calculateArea();

    double calculateCircumference();

    double calculateVolume();

}



// Abstract class for two-dimensional shapes

abstract class TwoDimensionalShape implements Shape {

    @Override

    public double calculateVolume() {

        // Two-dimensional shapes do not have volume

        return 0;

    }
```

```
}

// Final class for Circle (2D shape)
final class Circle extends TwoDimensionalShape {
    private double radius;

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

    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    @Override
    public double calculateCircumference() {
        return 2 * Math.PI * radius;
    }
}

// Final class for Sphere (3D shape)
final class Sphere implements Shape {
    private double radius;
```

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

    @Override
    public double calculateArea() {
        return 4 * Math.PI * radius * radius;
    }

    @Override
    public double calculateCircumference() {
        // Circumference is usually not defined for 3D shapes
        // Returning the circumference of a great circle (circle on the sphere's surface)
        return 2 * Math.PI * radius;
    }

    @Override
    public double calculateVolume() {
        return (4.0 / 3) * Math.PI * Math.pow(radius, 3);
    }
}

// Final class for Rectangle (2D shape)
```

```
final class Rectangle extends TwoDimensionalShape {
    private double length, width;

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

    @Override
    public double calculateArea() {
        return length * width;
    }

    @Override
    public double calculateCircumference() {
        return 2 * (length + width);
    }
}

// Main class to test the application
public class Main {
    public static void main(String[] args) {
        // Creating objects for Circle, Sphere, and Rectangle
        Circle circle = new Circle(5);
```

```
        Sphere sphere = new Sphere(5);
        Rectangle rectangle = new Rectangle(4, 6);

        // Displaying results for Circle
        System.out.println("Circle:");
        System.out.println("Area: " + circle.calculateArea());
        System.out.println("Circumference: " + circle.calculateCircumference());
        System.out.println("Volume: " + circle.calculateVolume());

        // Displaying results for Sphere
        System.out.println("\nSphere:");
        System.out.println("Area: " + sphere.calculateArea());
        System.out.println("Circumference: " + sphere.calculateCircumference());
        System.out.println("Volume: " + sphere.calculateVolume());

        // Displaying results for Rectangle
        System.out.println("\nRectangle:");
        System.out.println("Area: " + rectangle.calculateArea());
        System.out.println("Circumference: " + rectangle.calculateCircumference());
        System.out.println("Volume: " + rectangle.calculateVolume());
    }
}
```

Explanation:

- **Interface (`Shape`)**: Defines the methods `calculateArea()`, `calculateCircumference()`, and `calculateVolume()` for all shapes.
- **Abstract Class (`TwoDimensionalShape`)**: Implements the `Shape` interface but provides a default implementation for `calculateVolume()` that returns `0` since 2D shapes don't have volume.
- **Final Classes**:
    - **`Circle`**: A final class representing a circle, which extends the `TwoDimensionalShape` abstract class. It implements `calculateArea()` and `calculateCircumference()`.
    - **`Sphere`**: A final class representing a sphere, which implements the `Shape` interface directly, as it has both area and volume.
    - **`Rectangle`**: A final class representing a rectangle,

9 Write a program to enter two integers from the command line and display the division of those two numbers. Handle all the exceptions (i.e. ArrayIndexOutOfBoundsException, NumberFormatException, ArithmeticException) for invalid arguments passed.