Title: Using packages and sub-packages

Objective: Understand the concept of packages and sub-packages in Java.

Problem Statement: Create a Java program to illustrate the use of packages and sub-packages. Define two packages: `com.lab.demo` and `com.lab.demo.sub`. Inside the `com.lab.demo` package, create a class named `MainClass` with a main method. Inside the `com.lab.demo.sub` sub-package, create another class named `SubClass`.

In the `MainClass` main method, create an instance of `SubClass` and demonstrate how packages and sub-packages help in organizing and structuring Java code.

Instructions:

1. Use proper package declarations for the classes.
2. Compile and run the program to ensure that the interaction between classes in different packages works correctly.

Java Program:

```
java
// File: MainClass.java
package com.lab.demo;

import com.lab.demo.sub.SubClass;

public class MainClass {
    public static void main(String[] args) {
        // Create an instance of SubClass from the sub-package
        SubClass subObj = new SubClass();

        // Call a method from SubClass
        subObj.displayMessage();
    }
}
java
// File: SubClass.java
package com.lab.demo.sub;

public class SubClass {
    public void displayMessage() {
        System.out.println("This is a message from SubClass in the sub-
package.");
    }
}
```

Explanation:

- The `MainClass` is part of the `com.lab.demo` package.
- The `SubClass` is part of the `com.lab.demo.sub` sub-package.
- The `MainClass` creates an instance of `SubClass` and calls the `displayMessage` method to demonstrate the interaction between classes in different packages and sub-packages.

Note: Ensure that the directory structure matches the package structure, and compile the classes using the appropriate directory structure. For example:

```
bash
javac -d . MainClass.java
javac -d . com/lab/demo/sub/SubClass.java
```

This program can serve as a starting point for understanding how packages and sub-packages contribute to better organization and structure in Java programs.

Title: Create and demonstrate programs using interface

Objective: Understand the concept of interfaces in Java and their implementation.

Problem Statement: Create a Java program to demonstrate the use of interfaces. Define an interface named `Shape` with two abstract methods: `calculateArea()` and `displayInfo()`. Implement this interface in two classes: `Circle` and `Rectangle`. The `Circle` class should include the radius parameter, and the `Rectangle` class should include length and width parameters.

In the main program, create instances of `Circle` and `Rectangle`, invoke the methods from the `Shape` interface, and display the area and additional information for each shape.

Instructions:

1. Use the `Shape` interface to declare abstract methods.
2. Implement the `Shape` interface in the `Circle` and `Rectangle` classes.
3. In the `main` method, create instances of both `Circle` and `Rectangle`.
4. Invoke the methods from the `Shape` interface to calculate the area and display information for each shape.

Java Program:

```
java
// File: Shape.java (Interface)
public interface Shape {
    double calculateArea();
    void displayInfo();
}
java
// File: Circle.java (Implementation of Shape)
public class Circle implements Shape {
    private double radius;

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

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

    @Override
```

```
    public void displayInfo() {
        System.out.println("Circle - Radius: " + radius);
    }
}
java
// File: Rectangle.java (Implementation of Shape)
public class Rectangle implements Shape {
    private double length;
    private double width;

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

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

    @Override
    public void displayInfo() {
        System.out.println("Rectangle - Length: " + length + ", Width: " +
width);
    }
}
java
// File: MainProgram.java
public class MainProgram {
    public static void main(String[] args) {
        // Create instances of Circle and Rectangle
        Circle circle = new Circle(5.0);
        Rectangle rectangle = new Rectangle(4.0, 6.0);

        // Invoke methods from the Shape interface
        displayShapeInfo(circle);
        displayShapeInfo(rectangle);
    }

    // A generic method to display information for any class implementing
the Shape interface
    private static void displayShapeInfo(Shape shape) {
        System.out.println("Area: " + shape.calculateArea());
        shape.displayInfo();
        System.out.println();
    }
}
```

Explanation:

- The `Shape` interface declares two abstract methods: `calculateArea()` and `displayInfo()`.
- The `Circle` and `Rectangle` classes implement the `Shape` interface and provide concrete implementations for the interface methods.
- In the `MainProgram`, instances of `Circle` and `Rectangle` are created, and the interface methods are invoked to calculate the area and display information for each shape.

This program illustrates the concept of interfaces in Java and how they can be used to achieve abstraction and provide a common set of methods for classes with different implementations.

Question: Create and demonstrate programs for exception handling

Objective: Learn the basics of exception handling in Java, including try, catch, and throw mechanisms.

Problem Statement: Create a Java program that demonstrates the use of exception handling. Define a class named `Calculator` with methods for basic arithmetic operations (`add`, `subtract`, `multiply`, `divide`). Implement exception handling to handle scenarios such as division by zero and invalid inputs.

In the main program, create an instance of the `Calculator` class and perform arithmetic operations, demonstrating the proper use of try-catch blocks to handle exceptions.

Instructions:

1. Implement a `Calculator` class with methods for `add`, `subtract`, `multiply`, and `divide`.
2. Use appropriate exception classes (e.g., `ArithmeticException`) to handle errors.
3. In the main method, create an instance of the `Calculator` class and perform arithmetic operations inside try-catch blocks.
4. Display appropriate error messages for exceptions.

Java Program:

```
java
// File: Calculator.java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

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

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) throws ArithmeticException {
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return a / b;
    }
}
java
// File: MainProgram.java
import java.util.Scanner;

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

```
        Calculator calculator = new Calculator();
        Scanner scanner = new Scanner(System.in);

        try {
            // Input two numbers from the user
            System.out.print("Enter the first number: ");
            int num1 = scanner.nextInt();

            System.out.print("Enter the second number: ");
            int num2 = scanner.nextInt();

            // Perform arithmetic operations
            int sum = calculator.add(num1, num2);
            System.out.println("Sum: " + sum);

            int difference = calculator.subtract(num1, num2);
            System.out.println("Difference: " + difference);

            int product = calculator.multiply(num1, num2);
            System.out.println("Product: " + product);

            int quotient = calculator.divide(num1, num2);
            System.out.println("Quotient: " + quotient);

        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("An unexpected error occurred: " +
e.getMessage());
        } finally {
            // Close the scanner in the finally block to ensure it is
always closed
            scanner.close();
        }
    }
}
```

Explanation:

- The `Calculator` class provides basic arithmetic operations with exception handling for division by zero.
- The `MainProgram` class creates an instance of `Calculator`, takes user input, and demonstrates the use of try-catch blocks to handle exceptions.
- The program displays appropriate error messages for division by zero and other unexpected errors.

This program serves as an introduction to exception handling in Java, demonstrating how to handle specific exceptions and providing a generic catch block for unexpected errors.