3. (a) Create a Java program that demonstrates the use of interfaces and packages.

1. **Create a package named shapes.**
2. **Inside the shapes package, define an interface named Shape** with the following methods:
    - double area();
    - double perimeter();
3. **Create two classes, Circle and Rectangle, within the shapes package that implement the Shape interface.**
    - The Circle class should have:
        - A constructor that takes the radius as a parameter.
        - Implementation of the area() and perimeter() methods.
    - The Rectangle class should have:
        - A constructor that takes the length and width as parameters.
        - Implementation of the area() and perimeter() methods.
4. **In the main program (in a separate package),** demonstrate the use of the Circle and Rectangle classes by:
    - Creating an instance of each.
    - Displaying the area and perimeter of both shapes.

Hint: Add exception handling to ensure that the radius, length, and width are positive values.

Algorithm:

Step 1: Create the shapes package and the Shape interface.

Step 2: Implement the Circle class in the shapes package.

Step 3: Implement the Rectangle class in the shapes package.

Step 4: Create the main program in a separate package to use the Circle and Rectangle classes.

Program:

Step 1:

```
//File: shapes/Shape.java

package shapes;

public interface Shape {

   double area();

   double perimeter();

}
```

Step 2:

```
// File: shapes/Circle.java
package shapes;
public class Circle implements Shape {
    private double radius;
    public Circle(double radius) {
        if (radius <= 0) {
            throw new IllegalArgumentException("Radius must be positive.");
        }
        this.radius = radius;
    }
    public double area() {
        return Math.PI * radius * radius;
    }
    public double perimeter() {
        return 2 * Math.PI * radius;
    }
}
```

Step 3:

```
// File: shapes/Rectangle.java
package shapes;
public class Rectangle implements Shape {
    private double length;
    private double width;
    public Rectangle(double length, double width) {
        if (length <= 0 || width <= 0) {
            throw new IllegalArgumentException("Length and width must be positive.");
```

```
        }
        this.length = length;
        this.width = width;
    }
    public double area() {
        return length * width;
    }
    public double perimeter() {
        return 2 * (length + width);
    }
}
```

Step 4:

```
// File: Main.java
import shapes.Circle;
import shapes.Rectangle;
import shapes.Shape;
public class Main {
    public static void main(String[] args) {
        try {
            Shape circle = new Circle(5);
            System.out.println("Circle:");
            System.out.println("Area: " + circle.area());
            System.out.println("Perimeter: " + circle.perimeter());
            Shape rectangle = new Rectangle(4, 7);
            System.out.println("\nRectangle:");
            System.out.println("Area: " + rectangle.area());
            System.out.println("Perimeter: " + rectangle.perimeter());
```

```
} catch (IllegalArgumentException e) {
    System.out.println(e.getMessage());        }    }    }
```

Output:

When you run the `MainProgram` class, the output should be as follows:

Circle:

Area: 78.53981633974483

Perimeter: 31.41592653589793

Rectangle:

Area: 28.0

Perimeter: 22.0

3 b.

Write a Java program to demonstrate the difference between single-threading and multi-threading by simulating a simple task of printing numbers.

1. **Create a class NumberPrinter** that implements the Runnable interface. This class should have:
    - A constructor that takes a String name and an integer maxNumber as parameters.
    - The run() method should print the numbers from 1 to maxNumber, along with the thread name.
2. **Implement the single-threaded version:**
    - In the main program, create an instance of NumberPrinter and use a single thread to run it.
3. **Implement the multi-threaded version:**
    - In the main program, create two instances of NumberPrinter (with different names and max numbers) and run them using two separate threads.
4. **Demonstrate the difference in execution by observing the output order of the numbers in both the single-threaded and multi-threaded versions.**

Hint: Add a small delay (e.g., Thread.sleep(100)) in the run() method to better visualize the differences in output between single-threaded and multi-threaded execution.

Algorithm:

Step 1: Create the `NumberPrinter` class that implements the `Runnable` interface.

Step 2: Implement the single-threaded version in the main program.

Step 3: Implement the multi-threaded version in the main program.

Program: Step 1:

```
// File: NumberPrinter.java
public class NumberPrinter implements Runnable {
    private String name;
    private int maxNumber;

    public NumberPrinter(String name, int maxNumber) {
        this.name = name;
        this.maxNumber = maxNumber;
    }

    @Override
    public void run() {
        for (int i = 1; i <= maxNumber; i++) {
            System.out.println(name + " prints: " + i);
            try {
                // Adding a small delay to better visualize the difference between single and multi-threading
                Thread.sleep(100);
            } catch (InterruptedException e) {
                System.out.println(name + " was interrupted.");
            }
        }
    }
}
```

Step 2:

```
// File: SingleThreadDemo.java
public class SingleThreadDemo {
    public static void main(String[] args) {
        System.out.println("Single-threaded execution:");
        NumberPrinter printer = new NumberPrinter("SingleThread", 5);
        printer.run();  // Running on the main thread
    }
}
```

Step 3:

```
// File: MultiThreadDemo.java
public class MultiThreadDemo {
    public static void main(String[] args) {
        System.out.println("Multi-threaded execution:");
        NumberPrinter printer1 = new NumberPrinter("Thread-1", 5);
        NumberPrinter printer2 = new NumberPrinter("Thread-2", 5);

        Thread thread1 = new Thread(printer1);
        Thread thread2 = new Thread(printer2);

        thread1.start();  // Start the first thread
        thread2.start();  // Start the second thread
    }
}
```

Output:

```
Single-threaded execution:
```

SingleThread prints: 1

SingleThread prints: 2

SingleThread prints: 3

SingleThread prints: 4

SingleThread prints: 5

Multi-threaded execution:

Thread-1 prints: 1

Thread-2 prints: 1

Thread-1 prints: 2

Thread-2 prints: 2

Thread-1 prints: 3

Thread-2 prints: 3

Thread-1 prints: 4

Thread-2 prints: 4

Thread-1 prints: 5

Thread-2 prints: 5