CS1 Java Assignment: Abstraction, Inheritance, and Polymorphism

Goal: To understand and implement Java Interfaces, Abstract Classes, and demonstrate Polymorphism through a robust class hierarchy.
Estimated Time: 2 Hours
Concepts Covered:

- Defining and implementing an Interface (Measurable).
- Defining an Abstract Class (Shape) that implements an Interface.
- Class Inheritance (extends).
- Polymorphism (using the Measurable or Shape type to refer to concrete objects).
- Method Overriding and Inheritance benefits (e.g., in Square).

Scenario: The Unified Shape Metrics System

We are expanding our system to manage geometric shapes. We need a way to enforce that all shapes can provide their area (the **Interface**), while also providing a common base class for all geometric objects (the **Abstract Class**). This structure allows for maximum flexibility and code reusability.

Tasks

You will be creating seven files to establish the complete hierarchy: one interface, one abstract class, four concrete classes, and one main runner class.

Task 1: Define the Contract (Measurable.java)

This contract remains the top level of abstraction.

1. Define the Java **Interface** named Measurable.
2. Declare a single public, abstract method: double getArea().

Task 2: Define the Base Class (Shape.java)

Create an abstract base class for all geometric objects.

1. Define the **Abstract Class** named Shape.
2. Ensure that Shape **implements** the Measurable interface.
3. Because Shape is abstract, you do **not** need to implement the getArea() method; it is implicitly passed down to concrete (non-abstract) classes.
4. Give Shape a single abstract method: String getName(). (This forces all concrete shapes to define their name, but does not enforce a specific area calculation.)

Task 3: Implement the Concrete Hierarchy

Create three classes that extend the Shape class.

3A: Rectangle.java

1. Define the class Rectangle and ensure it **extends** Shape.
2. Give it two private instance variables: length and width (both of type double).
3. Create a constructor that accepts values for length and width.
4. Implement the required methods:
    - double getArea(): Returns the area ().
    - String getName(): Returns the string "Rectangle".

3B: Square.java (Inheritance in action)

1. Define the class Square and ensure it **extends** Rectangle.
2. Create a constructor that accepts only one argument, side (type double). Use this value to call the parent class's constructor (super(side, side)).
3. **Note:** Since Square is a Rectangle, it automatically inherits the correct getArea() implementation.
4. **Override** only the getName() method to return the string "Square".

3C: Circle.java

1. Define the class Circle and ensure it **extends** Shape.
2. Give it one private instance variable: radius (type double).
3. Create a constructor that accepts a value for the radius.
4. Implement the required methods:
    - double getArea(): Returns the area (). Use Math.PI.
    - String getName(): Returns the string "Circle".

3D: Challenge: Triangle.java

1. Define the class Triangle and ensure it **extends** Shape.
2. Give it two private instance variables: base and height (both of type double).
3. Create an appropriate constructor.
4. Implement the required methods:
    - double getArea(): Returns the area ().
    - String getName(): Returns the string "Triangle".

Task 4: Demonstrate Polymorphism (ShapeManager.java)

Create a main class, ShapeManager, which will contain the main method to execute and test your design.

1. Declare and initialize an ArrayList that can hold references to any object that implements the Measurable interface (or, equivalently, the Shape abstract class).
   ```
   import java.util.ArrayList;
   // ... inside main method ...
   ArrayList<Measurable> shapeList = new ArrayList<>();
   ```

2. **Instantiate Objects:** Create at least two instances of each concrete class (Rectangle,

Circle, Square, Triangle) with varied dimensions (a total of 8 shapes).

3. **Populate List:** Add all instantiated objects to the shapeList.
4. **Polymorphic Iteration:** Use a for-each loop to iterate through the shapeList. For each item in the list:
    - Call the getArea() method.
    - Call the getName() method. *(Hint: You may need to cast the Measurable reference to a Shape reference to access getName())*.
    - Print a message showing the shape's name and its calculated area.

    Example Output Snippet:Shape: Square, Area: 25.0
    Shape: Circle, Area: 28.274333882308138
    ...
    (Hint: Use ((Shape)shape).getName() to access the method defined in the Abstract Class).
5. **Calculate Total Area:** Compute and print the sum of all calculated areas to the console.

Submission Requirements

Submit the following seven Java source files:

1. Measurable.java (Interface)
2. Shape.java (Abstract Class)
3. Rectangle.java (Extends Shape)
4. Square.java (Extends Rectangle)
5. Circle.java (Extends Shape)
6. Triangle.java (Extends Shape)
7. ShapeManager.java (Main Class)