Exercise 12

Run-time type information

By the end of this exercise you will be able to

- Use the `instanceof` operator for checking the run-time type of a reference.
- Understand how casting in Java is different from casting in C.

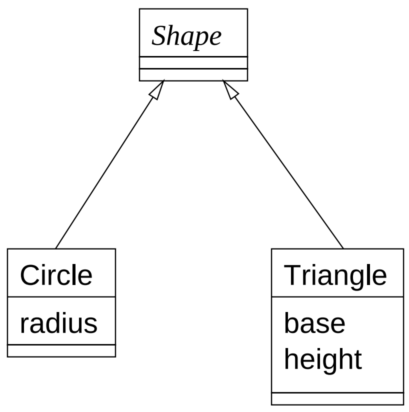
Introduction

Programmers of C will understand how type checking in C was all done by the compiler at compile time. When C's type checking system proved to restrictive for the problem at hand, the programmer was forced to bypass the type system using unsafe programming techniques such as casts, unions and pointers to `void`.

Java's type checking system is more powerful than C so it is no longer necessary nor possible to use unsafe practices like in C. Java has an extensive system of type checking that is done at compile time, as well as a run-time type system that C lacks. The questions of this exercise will show you how to access the run-time type information provided by the Java language.

Questions

1. Fetch the file `ShapeTest.java` which contains four classes: `Shape`, `Circle`, `Triangle` and a tester class `ShapeTest`. Here is the UML diagram showing how the `Circle` and `Triangle` classes inherit from the `Shape` class. The `Shape` class is shown in *italics* because it is an abstract class.



2. By copying the pattern for the `Circle` and `Triangle` classes, write a class `Rectangle` that inherits from `Shape`, has two properties `width` and `length`, and a two-argument constructor that sets these properties.

3. In the `doStuff` method, add a `new Rectangle(...)` to the end of the `myShapes` array by copying the pattern for the other shapes.

4. The `printNames` method scans along the `myShapes` array and uses the `instanceof` operator to determine, at run time, what kind of shape is in each slot of the `myShapes` array.

   Add an extra `if` statement to allow the `printNames` method to correctly identify instances of the `Rectangle` class that you have just created.

5. The `printAreas` method scans along the `myShapes` array and uses casting as well as the `instanceof` operator to print out the area of each shape.

   Casting is necessary so we can access the `radius` property of circles, the `base` and `height` properties of triangles and so on. Note that an expression like `myShapes[i].radius` is not valid in Java, because the object in the `i`$^{\text{th}}$ slot of the `myShapes` array my not be a circle, and therefore may not have a `radius` property that can be accessed.

   Note that unlike C, casting in Java is a completely safe operation. A cast in Java produces code that does a run-time check that the type of object being pointed to agrees with the type that it is being cast to. If the run-time check fails, a `ClassCastException` will be thrown.

   Even though casts in Java will throw exceptions when they fail, the code of the `printAreas` method can be guaranteed not to throw an exception because each cast lies inside an `if` statement that makes sure the cast will succeed.