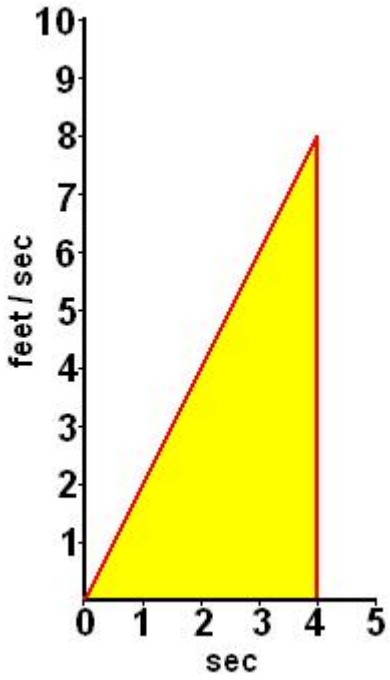
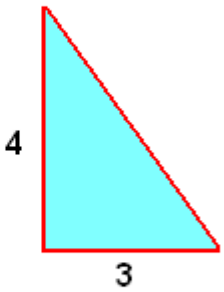
Lesson Instructions (Part 2)

It is good programming practice to design classes so that they can be reused without modifying the source code. For example, imagine that we need to find the area of a triangle in each of the following situations.

- In the first instance, the area beneath the curve of the velocity vs. time graph represents the distance an object traveled.
- In the second instance, we just need to know the area of a typical triangle with a base of 3 and a height of 4.



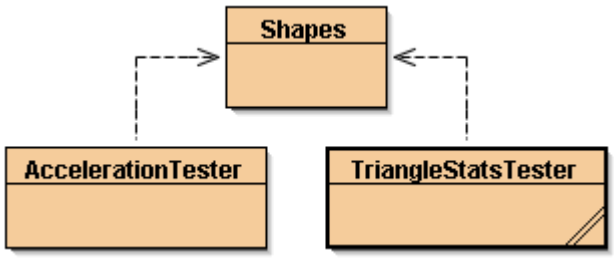


Clearly, these are two different contexts, but the calculation is the same.

Java provides several ways to write a program like this. You could continue to use the structure where instance variables, constructors, and methods are included in a single class but listed above the **`main()`** method in the source code. If you choose this approach, you will have to write an entirely new program and duplicate the code anytime you need to calculate the area of a triangle.

Instead, it would be more efficient to create a class called **`Shapes`** that would work for any instance in which you need to calculate the area of a triangle. This is easily accomplished by putting the code that calculates the area and hypotenuse of a triangle in its own class, separate from the **`main()`** method. There are two ways to approach this design issue, which involve creating what is often referred to as a tester class. A tester class usually contains the **`main()`** method and any code that is specific to the problem, such as print statements and method invocations. In this way, the **`Shapes`** class remains fully functional for any need.

The first way to disentangle the implementation of the **`Shapes`** class from the **`main()`** method is to create completely separate classes, each in its on file. This is visually represented by the following three class icons.



- The **`Shapes`** class only contains instance variables, a constructor, and methods, and bears responsibility for calculating the area and hypotenuse of a triangle.
- The **`AccelerationTester`** class contains a **`main()`** method and the statements related to a program involving acceleration.
- The **`TriangleStatsTester`** class contains another **`main()`** method and the appropriate code for determining basic triangle measurements.

Open the 08.06 Lesson Part 2 project and carefully study the code in each class. Notice the generic nature of the **`Shapes`** class, compared to how specific the other two classes are. Because the **`Shapes`** class is so general, it can work with any other class that needs methods for calculating the area or hypotenuse of a triangle.

A second approach involves putting two or more classes in the same source code, as is illustrated in the **`TriangleStatsTesterAlternative`** class.

Open the 08.06 Lesson Part 2 project and carefully study the code in the **`TriangleStatsAlternative()`** class. Notice that only the class with the **`main()`** method is declared to be public and the name of the file matches the public class.

There are several other features of these programs worth noting.

1. Methods can be invoked as the argument of a print statement or in an assignment statement.
2. Not all of the methods in a generic class like **`Shapes`** have to be invoked in any particular program. For example, in the **`AccelerationTester`** class, only the **`calcTriArea()`** method is called; however, in the **`TriangleStatsTester`** class **`calcTriArea()`** and **`calcHypoteneuse()`** are both called.

Experiment with these two techniques for separating the implementation of a class from its tester class. Make the following modifications:

1. Create an `AccelerationTesterAlternative` class that contains both the **`Shapes`** class and the **`AcceleratonTester`** class in a single file.
2. Add some overloaded methods to calculate the area and hypotenuse of a triangle that take double parameters. Write statements in the **`main()`** program that invoke the overloaded methods.
3. Add methods to the **`Shapes`** class to calculate the area and perimeter of a rectangle. Then write a **`RectangleStatsTester`** class to verify that your methods work.
4. Add methods to the **`Shapes`** class to calculate the area and circumference of a circle. Confirm that the new methods work by writing a **`CircleStatsTester`**. This one is a little trickier because it requires an additional **`Shapes`** constructor, which takes a single parameter. Just like methods, constructors can be overloaded as long as the parameter lists are different.