Interfaces

Subset of the Supplement Lesson slides from: Building Java Programs, Chapter 9.5
by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/) & thanks to Ms Martin.

Relatedness of types

Write a set of `Circle`, `Rectangle`, and `Triangle` classes.

- Certain operations that are common to all shapes.

perimeter	- distance around the outside of the shape
area	- amount of 2D space occupied by the shape

- Every shape has them but computes them differently.

Shape area, perimeter

- Rectangle (as defined by width w and height h):

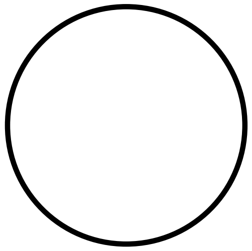
  area = $w\,h$

  perimeter = $2w + 2h$

- Circle (as defined by radius r):

  area = $\pi\, r^2$

  perimeter = $2\,\pi\, r$

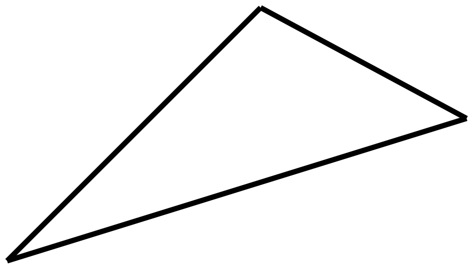


- Triangle (as defined by side lengths a, b, and c)

  area = $\sqrt{(s\,(s - a)\,(s - b)\,(s - c))}$

  where $s = ½\,(a + b + c)$

  perimeter = $a + b + c$



Common behavior

- Write shape classes with methods `perimeter` and `area`.

- We'd like to be able to write client code that treats different kinds of shape objects in the same way, such as:
  - Write a method that prints any shape's area and perimeter.
  - Create an array of shapes that could hold a mixture of the various shape objects.
  - Write a method that could return a rectangle, a circle, a triangle, or any other shape we've written.
  - Make a `DrawingPanel` display many shapes on screen.

Interfaces

- **interface**: A list of methods that a class can implement.
  - Interfaces give you an is-a relationship *without* code sharing.
    - A `Rectangle` object can be treated as a `Shape` but has no common code..
  - Analogous to the idea of roles or certifications:
    - "I'm certified as a CPA accountant. That means I know how to compute taxes, perform audits, and do consulting."
    - "I'm certified as a Shape. That means I know how to compute my area and perimeter."

Declaring an interface

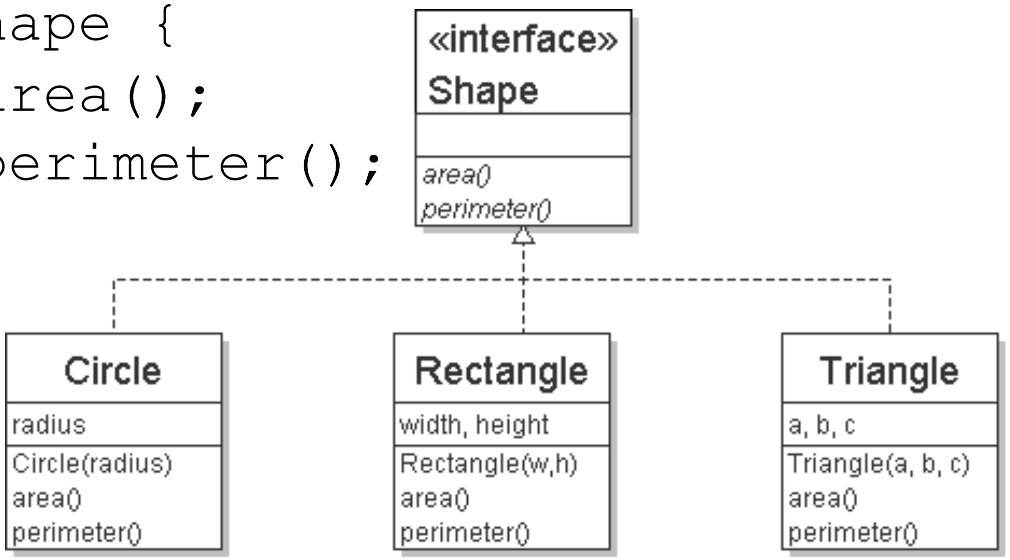
```
public interface name {
    public type name(type name, ..., type name);
    public type name(type name, ..., type name);
    ...
}
```

Example:

```
public interface Vehicle {
     public double speed();
     public void setDirection(int direction);
}
```

Shape interface

```
public interface Shape {
    public double area();
    public double perimeter();
}
```



- Saved as `Shape.java`
- This interface describes the features common to all shapes. (Every shape has an area and perimeter.)

- **abstract method**: A header without an implementation.
  - The actual body is not specified, to allow/force different classes to implement the behavior in its own way.

Implementing an interface

```
public class name implements interface {
    ...
}
```

– Example:

```
public class Bicycle implements Vehicle {
    ...
}
```

- A class can declare that it *implements* an interface.
  - This means the class must contain each of the abstract methods in that interface. (Otherwise, it will not compile.)

    (What must be true about the `Bicycle` class for it to compile?)

Interface requirements

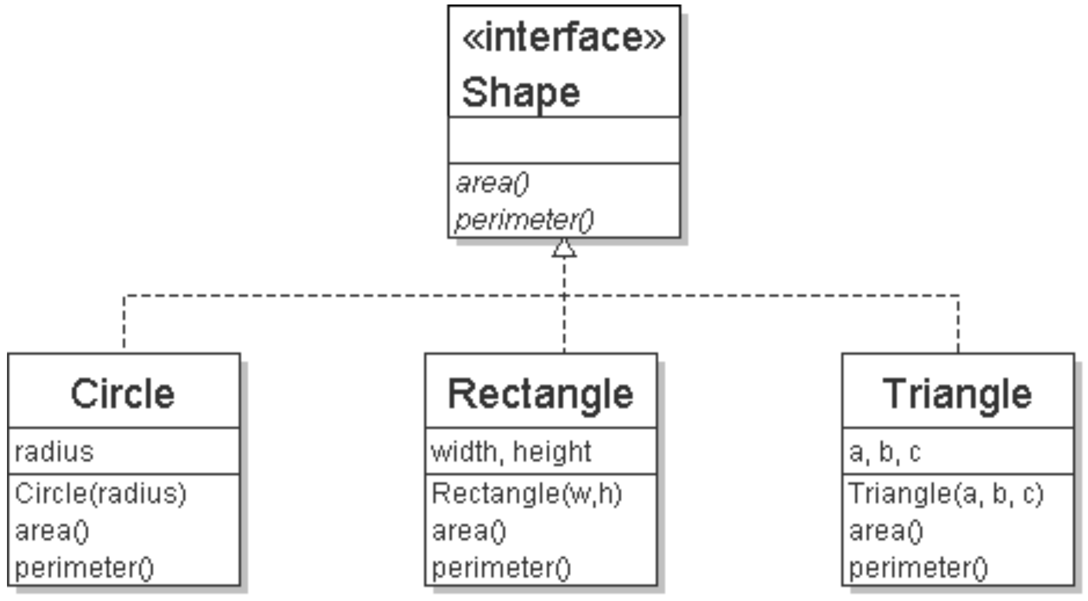
- If a class claims to be a `Shape` but doesn't implement the `area` and `perimeter` methods, it will not compile.
  - Example:

```
public class Banana implements Shape {
    ...
}
```

  - The compiler error message:

```
Banana.java:1: Banana is not abstract and does
not override abstract method area() in Shape
public class Banana implements Shape {
           ^
```

Interface diagram



- Arrow goes up from class to interface(s) it implements.
  - There is a supertype-subtype relationship here; e.g., all Circles are Shapes, but not all Shapes are Circles.
  - This kind of picture is also called a *UML class diagram*.

Interfaces Summary

- An interface defines a protocol of communication between two objects.
- An interface declaration contains signatures, but no implementations, for a set of methods, and might also contain constant definitions.
- A class that implements an interface must implement all the methods declared in the interface.
- An interface name can be used anywhere a type can be used.

Complete Circle class

```
// Represents circles.
public class Circle implements Shape {
    private double radius;

    // Constructs a new circle with the given radius.
    public Circle(double radius) {
        this.radius = radius;
    }

    // Returns the area of this circle.
    public double area() {
        return Math.PI * radius * radius;
    }

    // Returns the perimeter of this circle.
    public double perimeter() {
        return 2.0 * Math.PI * radius;
    }
}
```

Complete Rectangle class

```
// Represents rectangles.
public class Rectangle implements Shape {
    private double width;
    private double height;

    // Constructs a new rectangle with the given dimensions.
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    // Returns the area of this rectangle.
    public double area() {
        return width * height;
    }

    // Returns the perimeter of this rectangle.
    public double perimeter() {
        return 2.0 * (width + height);
    }
}
```

Complete Triangle class

```
// Represents triangles.
public class Triangle implements Shape {
    private double a;
    private double b;
    private double c;

    // Constructs a new Triangle given side lengths.
    public Triangle(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    // Returns this triangle's area using Heron's formula.
    public double area() {
        double s = (a + b + c) / 2.0;
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }

    // Returns the perimeter of this triangle.
    public double perimeter() {
        return a + b + c;
    }
}
```