Interfaces

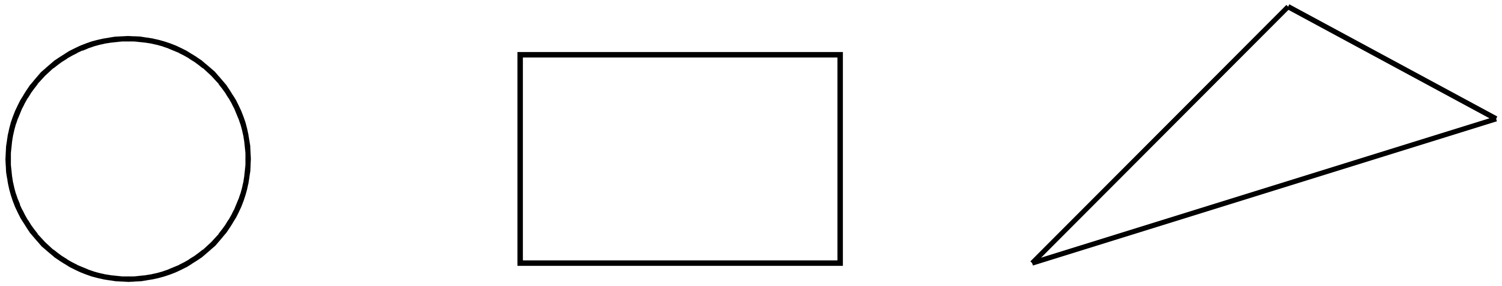
Example – Shapes

- Consider the task of writing classes to represent 2D shapes such as `Circle`, `Rectangle`, and `Triangle`.

- Certain operations 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 these, but each computes them differently.

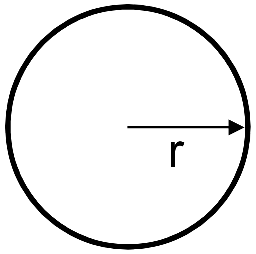


Let's define area & perimeter

- Circle (as defined by radius r):

  area = πr^2

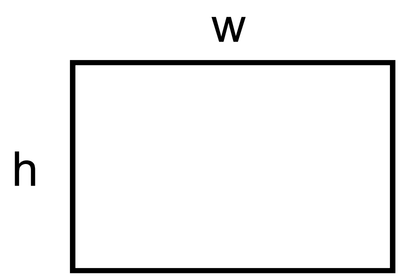
  perimeter = $2 \pi r$



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

  area = $w h$

  perimeter = $2w + 2h$

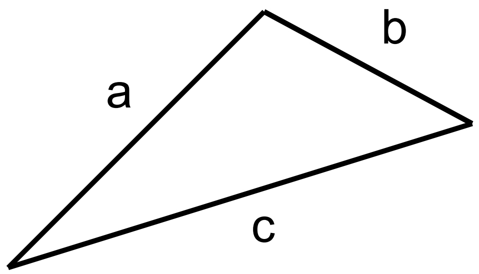


- Triangle (as defined by side lengths a, b, and c)

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

  where $s = ½ (a + b + c)$

  perimeter = $a + b + c$



What we want to code...

Suppose we have 3 classes `Circle`, `Rectangle`, `Triangle`.

- Each has the methods `perimeter()` and `area()`

We'd like our client code to be able to treat different kinds of shapes in the same way; e.g.,

- Write a method that prints any shape's area and perimeter.
- Create an array to hold a mixture of the various shape objects.
- Write a method that could return a rectangle, a circle, a triangle, or any other kind of shape.
- Make a `DrawingPanel` display many shapes on screen

BUT each class already subclass DrawableObject

Solution = Polymorphism! But we have only 1 shot at inheritance!

Interfaces to the rescue!!!

Definition

- A list of methods that a class promises to implement
- A contract in terms of what features / methods / behavior will be implemented
- Analogous to idea of roles / certifications:
  - "I'm certified as a CPA accountant.
    This assures you I know how to do taxes, audits, and consulting."
  - "I'm 'certified' as a Shape, because I implement the Shape interface.
    This assures you I know how to compute my area and perimeter."

How is this different from inheritance?

- Inheritance gives you an is-a relationship *and* code sharing
  - A `Lawyer` can be treated as an `Employee` and inherits its code
- Interfaces give you an is-a relationship *without* code sharing
  - A `Rectangle` object can be treated as a `Shape` but inherits no code
- You extend only 1 superclass but may implement many interfaces
- **Interfaces only feature abstract methods**
  - i.e. header w/o implementation
  - we want to allow each class to implement the behavior in its own way
- Interface only feature FINAL fields

Interface syntax

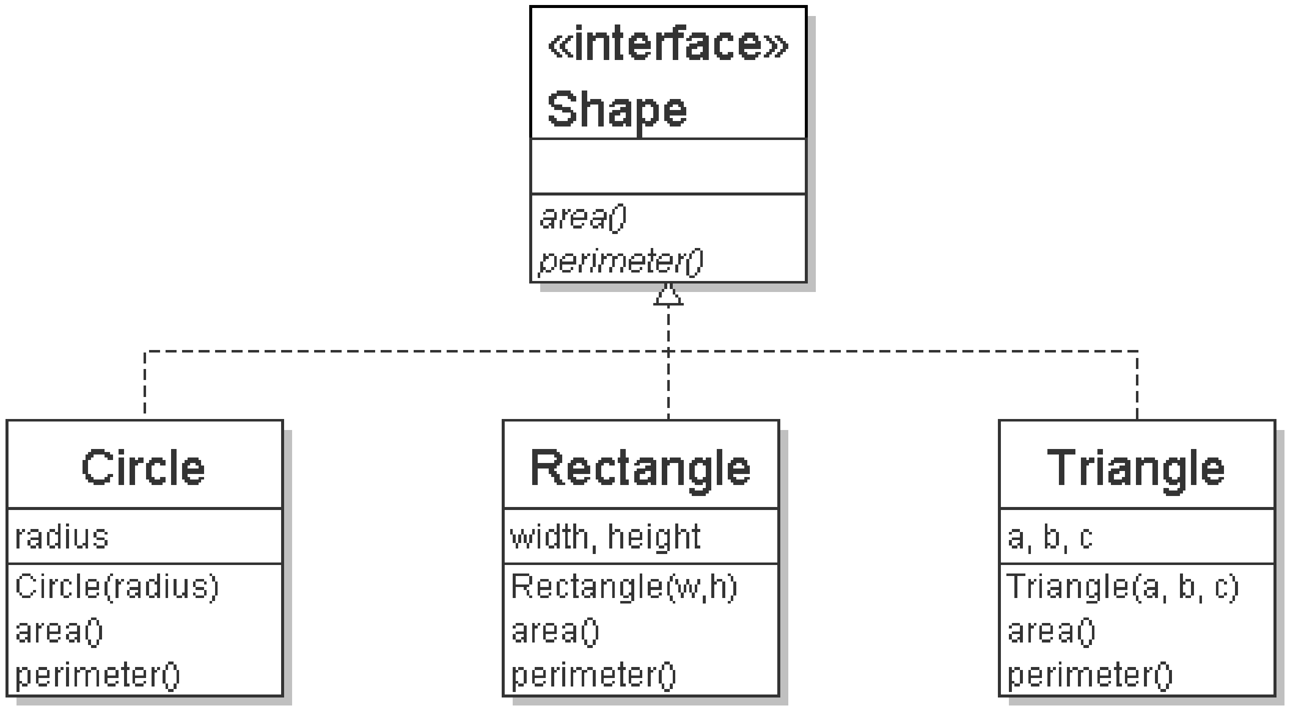
```
public interface name {
    public type name(type name, ..., type name);
    public type name(type name, ..., type name);
    ...
    public type name(type name, ..., type name);
}
```

Example

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

Example – Shape interface

```
// Shape.java Describes features of all shapes
public interface Shape {
 public double area();
 public double perimeter();
}
```



How do we **Implement** an interface?

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

Definition

A class can declare that it "implements" an interface.

Example

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

What if we implement an interface w/o providing code?

```
public class Banana implements Shape {
    // haha, no methods! pwned
}
```

If we write a class that claims to be a `Shape` but doesn't implement `area` and `perimeter` methods, it will not compile.

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

Interfaces + polymorphism?

Yes.

Interfaces allow **polymorphism**
(the same code can work with different types of objects)

```
public static void printInfo(Shape s) {
    System.out.println("The shape: " + s);
    System.out.println("area : " + s.area());
    System.out.println("perim: " + s.perimeter());
    System.out.println();
}
...
Circle circ = new Circle(12.0);
Triangle tri = new Triangle(5, 12, 13);
printInfo(circ);
printInfo(tri);
```