Interfaces

Ejemplo de figuras

- Considerar la tarea de escribir clases para representar figuras en 2D tales como círculos, rectángulos y triángulos.
- Algunas operaciones son comunes a todas las figuras:
  - Perímetro: distancia alrededor del exterior de la forma.
  - Área: cantidad de espacio ocupado por la forma.
- Cada figura tiene estas operaciones, pero cada uno las calcula de manera diferente.

Área y perímetro

- Círculo (definido por su radio r):
  - Área = $\pi \cdot r^2$
  - Perímetro = $2 \cdot \pi \cdot r$
- Rectángulo (definido por su ancho w y altura h):
  - Área = $w \cdot h$
  - Perímetro = $2 \cdot w + 2 \cdot h$
- Triángulo (definido por sus lados a, b, c):
  - Área = $\sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)}$ con $s = \frac{1}{2}(a+b+c)$
  - Perímetro = $a + b + c$

Comportamiento en común

- Se definen tres clases: Circle, Rectangle, Triangle.
- Cada uno tiene los métodos: perimeter y area, con distintas implementaciones.
- Se desea definir una clase, Shape que, de alguna forma, sea la superclase de Circle, Rectangle y Triangle.
- Los objetivos son:
  - Poder definir un método printShape que imprima el área y el perímetro de una figura.
  - Poder crear un arreglo que contenga distintas figuras.

Comportamiento en común

- El problema se puede resolver con herencia.
- Que las clases Circle, Rectangle y Triangle hereden de Shape.
- La desventaja es que el área y el perímetro no están definidos para una figura en general.
- Java ofrece un mecanismo para heredar comportamiento (métodos) sin heredar código (implementación) llamado interfaces.

Interfaces

- **Interface**. Una lista de métodos que una clase puede implementar.
- Las interfaces brindan una relación es-un **sin** heredar código.
- La herencia mediante `extends` produce una relación es-un y si permite heredar código.

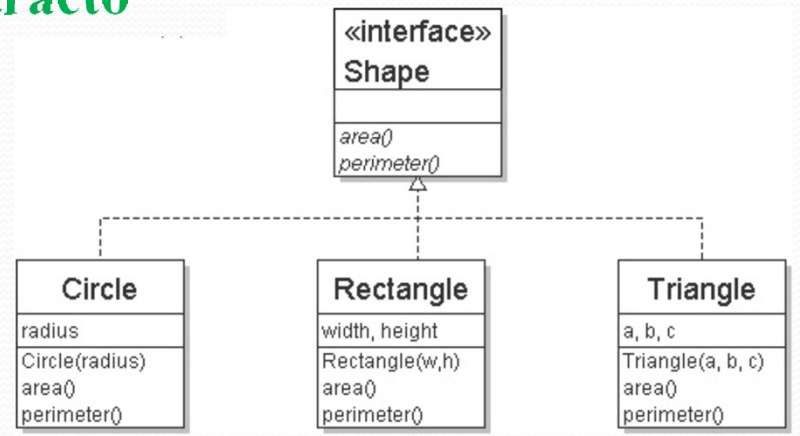
Sintaxis

```
public interface nombre {
    public tipo método-1(argumentos);
    …
    public tipo método-n(argumentos);
}
```

Interface Shape

```
// Describe características comunes a todas las figuras.
public interface Shape {
    public double area(); // método abstracto
    public double perimeter(); // método abstracto
}
```

- Se guarda en Shape.java.



Métodos abstractos

- **Método abstracto**. El encabezado del método sin una implementación.
- El cuerpo de un método abstracto no está especificado.
- La idea es permitir que cada clase implemente el método a su manera.

Implementar una interface

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

- Una clase puede declarar que *implementa* una interface.
- La clase debe contener todos los métodos de la interface.
- Si no lo hace, no compila.

Clase Circle

```
public class Circle implements Shape {
    private final double radius;
    public Circle(double radius) // Construye un círculo con el radio dado.
    {
        this.radius = radius;
    }
    public double area()          // Regresa el área de este círculo
    {
        return Math.PI * radius * radius;
    }
```

Clase Circle

```
    // Regresa el perímetro de este círculo
    public double perimeter()
    {
        return 2.0 * Math.PI * radius;
    }
}
```

Clase Rectangle

```
public class Rectangle implements Shape {
    private final double width;
    private final double height;
    // Construye un rectángulo con las dimensiones dadas
    public Rectangle(double width, double height)
    {
        this.width = width;
        this.height = height;
    }
```

Clase Rectangle

```
    // Regresa el área de este rectángulo.
    public double area()
    {
        return width * height;
    }
    // Regresa el perímetro de este rectángulo.
    public double perimeter()
    {
        return 2.0 * (width + height);
    }
}
```

Clase Triangle

```
public class Triangle implements Shape {
    private final double a;
    private final double b;
    private final double c;
    // Construye un triángulo dados sus lados.
    public Triangle(double a, double b, double c)
    {
        this.a = a;
        this.b = b;
        this.c = c;
    }
```

Clase Triangle

```
    // Regresa el área de este triángulo usando la fórmula de Heron.
    public double area()
    {
        double s = (a + b + c) / 2.0;
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }
    // Regresa el perímetro de este triángulo.
    public double perimeter()
    {
        return a + b + c;
    }
}
```

Interfaces y polimorfismo

- La relación es-un de la interface permite al cliente utilizar el polimorfismo.

```
public static void printShape(Shape s)
{
    System.out.println("The shape: " + s);
    System.out.println("area : " + s.area());
    System.out.println("perim: " + s.perimeter());
    System.out.println();
}
```

Interfaces y polimorfismo

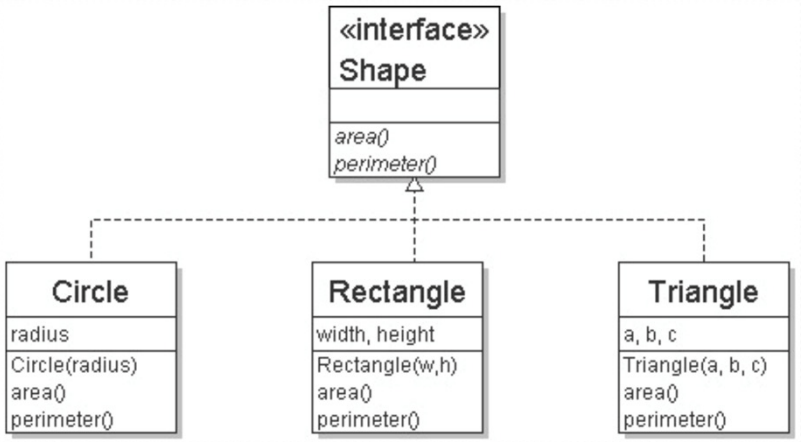
- Cualquier objeto que implemente la interface puede pasarse como argumento.

```
Shape c = new Circle(12.0);
Shape r = new Rectangle(4, 7);
Shape t = new Triangle(5, 12, 13);
printShape(c);
printShape(t);
printShape(r);
Shape[] shapes = {t, c, r};
```

Interfaces

- ¿Por qué existen interfaces en Java?.
- Java **no** permite herencia múltiple.
- Una clase **no** puede extender más de una clase.
- Una clase **si** puede implementar más de una interface.

Diagrama UML



- Unified Modeling Language.
- La flecha sube de la clase a las interfaces que implementa.