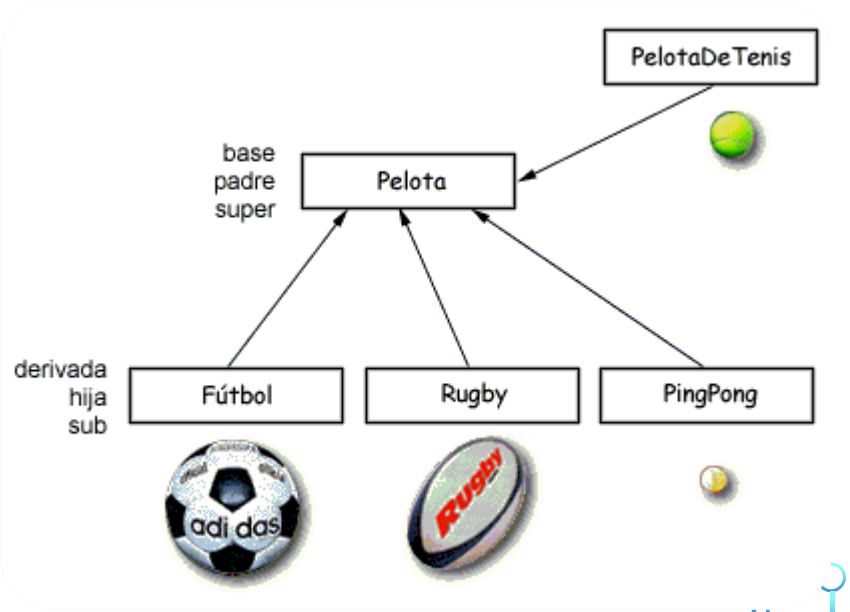
CLASE 9 -HERENCIA Y POLIMORFISMO



INTRODUCCIÓN

- Diferentes tipos de objetos con características y comportamiento comunes.

Triángulo

- lado1
- lado2
- lado3
- color de línea
- color de relleno
- punto

Círculo

- radio
- color de línea
- color de relleno
- punto

INTRODUCCIÓN

- Diferentes tipos de objetos con características y comportamiento comunes.

Triángulo

- Devolver y modificar el valor de cada atributo
  - lado1
  - lado2
  - lado3
  - color de línea
  - color de relleno
  - punto
- Calcular el área
- Calcular el perímetro

Círculo

- Devolver y modificar el valor de cada atributo
  - radio
  - color de línea
  - color de relleno
  - punto
- Calcular el área
- Calcular el perímetro

INCONVENIENTES HASTA AHORA. HERENCIA COMO SOLUCIÓN

- Esquema de trabajo hasta ahora:
  - Definimos las clases Triángulo y Circulo.
  - Problemas: Replicación de características y comportamiento común.
- Solución → Herencia
  - Permite que la clase ***herede*** características y comportamiento (atributos y métodos) de otra clase (clase padre o superclase). A su vez, la clase define características y comportamiento propio.
  - Potencia la reutilización. Este mecanismo no se encuentra en lenguajes imperativos.
  - Ejemplo. Se define lo común en una clase Figura y las clases Triángulo y Círculo lo heredan.

HERENCIA. EJEMPLO

- Diagrama de clases.

Herencia simple: sólo una superclase directa.

Las clases forman una jerarquía.

*Ambos **deben** implementar calcularArea() y calcularPerimetro pero de manera diferente*

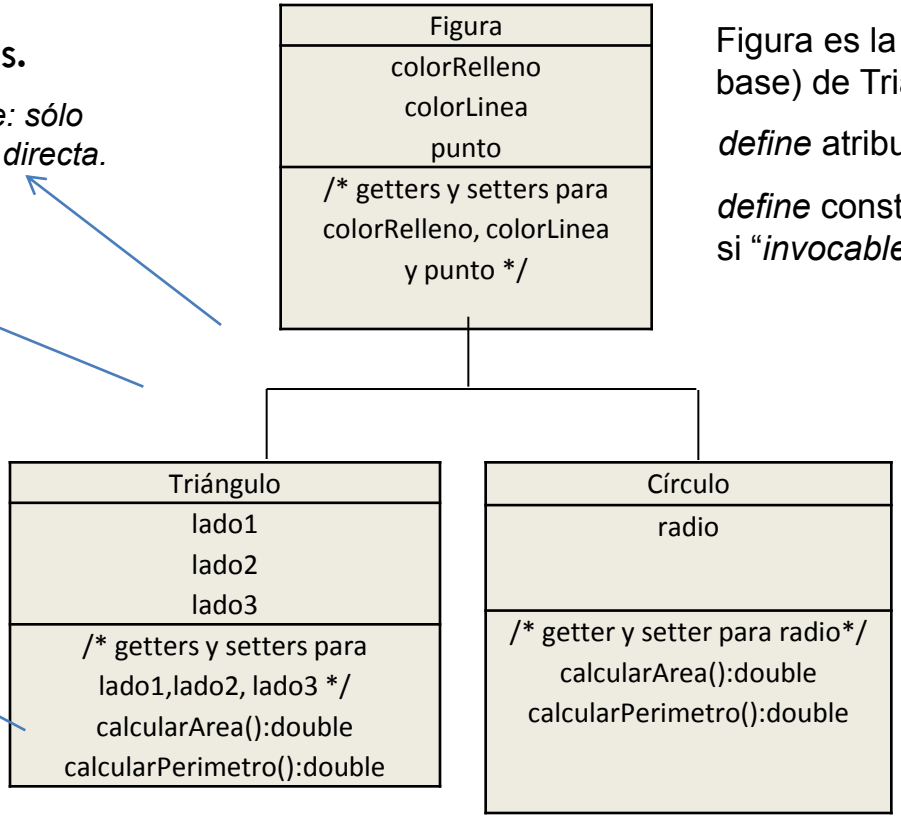


Figura es la **superclase** (clase padre o base) de Triángulo y Círculo.

define atributos y comportamiento común

define constructores (no heredables, si "*invocables*")

Triángulo y Círculo son **subclases** (clases hijas o derivadas) de Figura.

heredan atributos y métodos de Figura

definen atributos y métodos propios

definen constructores.

BÚSQUEDA DE MÉTODO EN LA JERARQUÍA DE CLASES

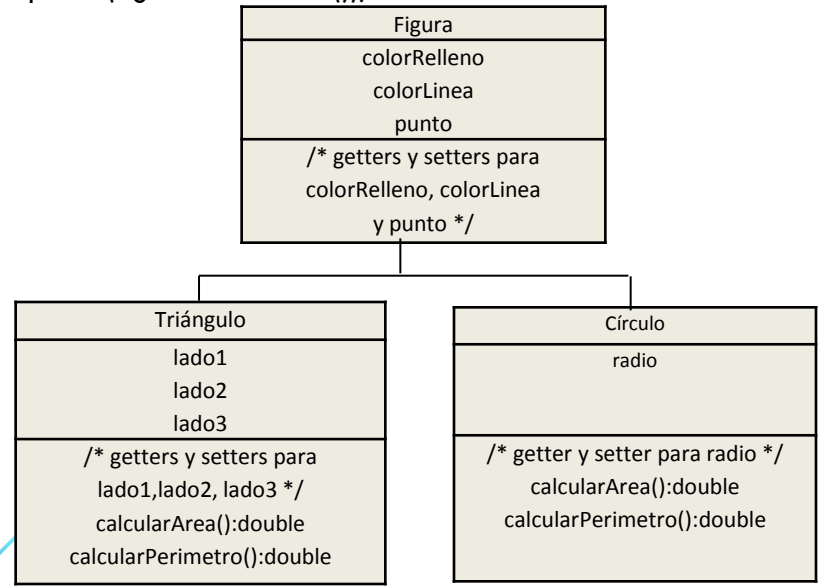
- Ejemplo

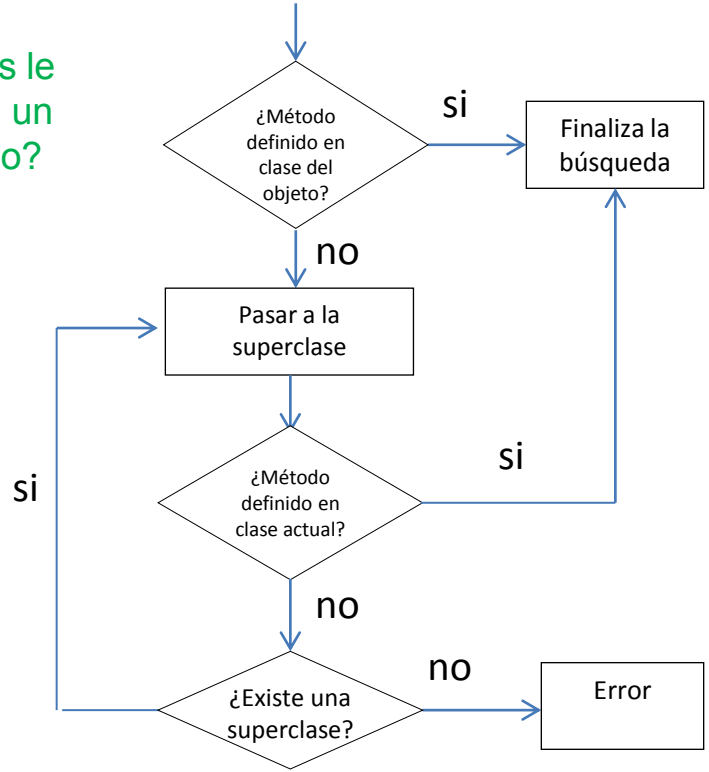
```
Triangulo t = new Triangulo(…);

System.out.println(t.calcularArea());

System.out.println(t.getColorRelleno());
```

¿Qué mensajes le puedo enviar a un objeto triángulo?





HERENCIA EN JAVA

- Definición de relación de herencia. Palabra clave *extends*.

```
public class NombreSubclase extends NombreSuperclase{
    /* Definir atributos propios */
    /* Definir constructores propios */
    /* Definir métodos propios */
}
```

- Si no se especifica una superclase con *extends*, *extiende* por defecto la clase Object.
- La subclase *hereda* los atributos declarados en la superclase, pero al ser *privados* son accesibles sólo en métodos de la clase que los declara. *En la subclase accederlos a través de getters y setters heredados.*
- La subclase *hereda* métodos de instancia.
- La subclase puede declarar atributos propios.
- La subclase puede declarar métodos propios.
- La subclase puede declarar constructores propios.

HERENCIA EN JAVA

- Los constructores de Triángulo y Círculo replican código de inicialización de atributos comunes a todas las Figuras.

```
public Triangulo(double lado1, double lado2, double lado3,
                 String colorRelleno, String colorLinea,
                 Punto punto){
    this.setColorRelleno(colorRelleno);
    this.setColorLinea(colorLinea);
    this.setPunto(punto);
    this.setLado1(lado1);
    this.setLado2(lado2);
    this.setLado3(lado3);
}
```

```
public Circulo(double radio,
               String colorRelleno, String colorLinea,
               Punto punto){
    this.setColorRelleno(colorRelleno);
    this.setColorLinea(colorLinea);
    this.setPunto(punto);
    this.setRadio(radio);
}
```

Factorizar el código común definiendo un constructor en la clase Figura

¿Cómo lo invoco?

HERENCIA EN JAVA

- Los métodos *dibujar* de Triángulo y Círculo replican código.

```
public class Triangulo{
….
 public void dibujar(){
        System.out.println("Triangulo: " );
        System.out.println("Color de Linea: " + this.getColorLinea() );
        System.out.println("Color de Relleno: " + this.getColorRelleno() );
        System.out.println("Ubicación: " + this.getPunto().toString() );
         System.out.println("L1: " + this.getLado1() +
                                  " L2: " + this.getLado2() +
                                   " L3: " + this.getLado3());
 }
}
```

```
public class Circulo{
….
 public void dibujar(){
        System.out.println("Circulo: " );
        System.out.println("Color de Linea: " + this.getColorLinea() );
        System.out.println("Color de Relleno: " + this.getColorRelleno() );
        System.out.println("Ubicación: " + this.getPunto().toString() );
        System.out.println("Radio: " + this.getRadio() );
    }
}
```

Factorizar el código común definiendo un *dibujar* en la clase Figura

¿Cómo lo invoco?

LA REFERENCIA SUPER

- Dentro de un *método de instancia* o de un *constructor,* la referencia ***super*** representa a la clase padre del objeto que recibió el mensaje o el objeto que está siendo instanciado respectivamente.
- Uso:

a) Dentro de un constructor se puede invocar al constructor de la **superclase.**

Sintaxis: super(parámetros) *Diferencia con this(…)*

```
public class Figura{
        …
        public Figura(String colorRelleno,
                      String colorLinea,
                       Punto punto){
            this.setColorRelleno(colorRelleno);
            this.setColorLinea(colorLinea);
            this.setPunto(punto);
        }
    …
}
```

Recomendación: Siempre definir en las clases el constructor sin parámetros

```
public class Circulo extends Figura{
    …
    public Circulo(double radio,
                   String colorRelleno, String colorLinea,
                   Punto punto){
        super(colorRelleno, colorLinea, punto);
        this.setRadio(radio);
    }
}
```

Si realizamos invocación explícita a un constructor de la superclase debe ser la primera línea

¿Cómo se construye un objeto?

Desde el constructor, en caso de no existir invocación explicita, Java invoca i*mplícitamente* al constructor sin parámetros de la superclase.

LA REFERENCIA SUPER

Uso:

b) Dentro de un método de instancia, el objeto puede enviarse un mensaje a sí mismo. El método es buscado a partir de la superclase *actual*. *Diferencia con this.nombreMetodo(…)*

Sintaxis: super.nombreMetodo(parametros).

```
public class Figura{
  …
 public void dibujar(){
       System.out.println("Color de Linea: " + this.getColorLinea() );
       System.out.println("Color de Relleno: " + this.getColorRelleno() );
       System.out.println("Ubicación: " + this.getPunto().toString() );
   }
}
```

Circulo:
Color de Línea: negro
Color de Relleno: azul
Ubicación: (100,100)
Radio: 5.0

Triangulo:
Color de Línea: negro
Color de Relleno: azul
Ubicación: (100,100)
L1: 5.0
L2: 10.2
L3: 8.0

Circulo *redefine* dibujar: modifica el comportamiento del método heredado

```
public class Circulo extends Figura{
  …
 public void dibujar(){
       System.out.println("Circulo: " );
       super.dibujar();
       System.out.println("Radio: " + this.getRadio() );
   }
}
```

CLASES Y MÉTODOS ABSTRACTOS

- Clase abstracta
  - Clase de la cual no se crearán instancias.
  - Ejemplos: la clase Figura.
  - Declaración en Java:
    - anteponer ***abstract*** a la palabra class.

```
public abstract class NombreClase {
    /* Definir atributos */
    /* Definir constructores */
    /* Definir métodos no abstractos */
    /* Definir métodos abstractos */
}
```

- Método abstracto
  - Métodos sin implementación en la clase que lo declara. Las subclases **tienen obligación** de implementarlos.
  - Ejemplo: calcularArea y calcularPerimetro de Figura.
  - Declaración en Java:
    - encabezado del método anteponiendo *abstract* al tipo de retorno.

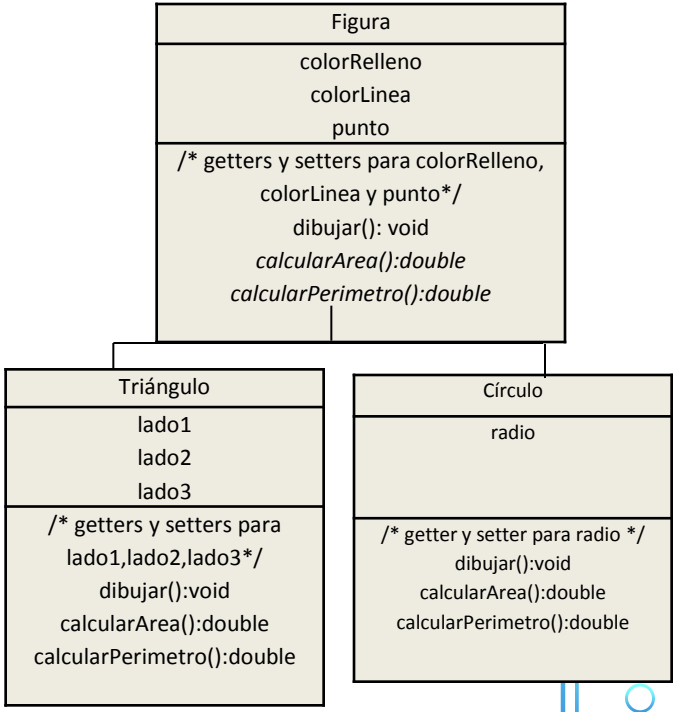
```
public abstract TipoRetorno nombreMetodo(lista parámetros);
```

CONVERSIÓN ASCENDENTE (*UPCASTING*)

- Cualquier objeto instancia de una *clase derivada* puede ser referenciado por una variable cuyo tipo es la *clase base* (conversión ascendente)

```
Figura fig1 = new Circulo(…);
Figura fig2 = new Triangulo(…);
```

- Siempre es posible: la herencia establece una relación "es-un"
- Pueden existir variables cuyo tipo es una clase abstracta que referencien a instancias de clases derivadas de esta.
- **Al objeto sólo se le puede enviar mensajes definidos en la interfaz de la *clase usada como tipo* para la variable referencia (*clase base*).**
- **La búsqueda del método a ejecutar comienza siempre desde la clase instanciada.**



POLIMORFISMO

- Posibilidad de enviar mensajes sintácticamente iguales a objetos de distintas clases.
- La misma operación se realiza de distinta forma según sea el objeto al que se le envía el mensaje.
- Ejemplo

```
Figura [] figuras = new Figura[10];
/* cargar arreglo con círculos y triángulos */
for (i=0; i<10; i++)
        figuras[i].dibujar();
```

El método dibujar() a ejecutar dependerá de la clase de figura geométrica.

Clases, métodos abstractos y Upcasting. Ejercitación.

- Definir la clase Figura como abstracta.
- Definir en Figura los métodos abstractos calcularArea y calcularPerimetro.
- Analizar qué hace el siguiente programa:

```
public class DemoFiguras {
        public static void main(String[] args) {
                Circulo circ =new Circulo(5, "amarillo", "negro", new Punto (100,100));
                System.out.println(circ.getRadio());
                circ.dibujar();
                Figura fig= circ;
                System.out.println(fig.getRadio());
                fig.dibujar();
                System.out.println("Color linea: " + fig.getColorLinea());
        }
}
```

  - Responder: ¿Qué mensajes le puedo enviar al objeto a través de la referencia circ? ¿Qué mensajes a través de la referencia fig?