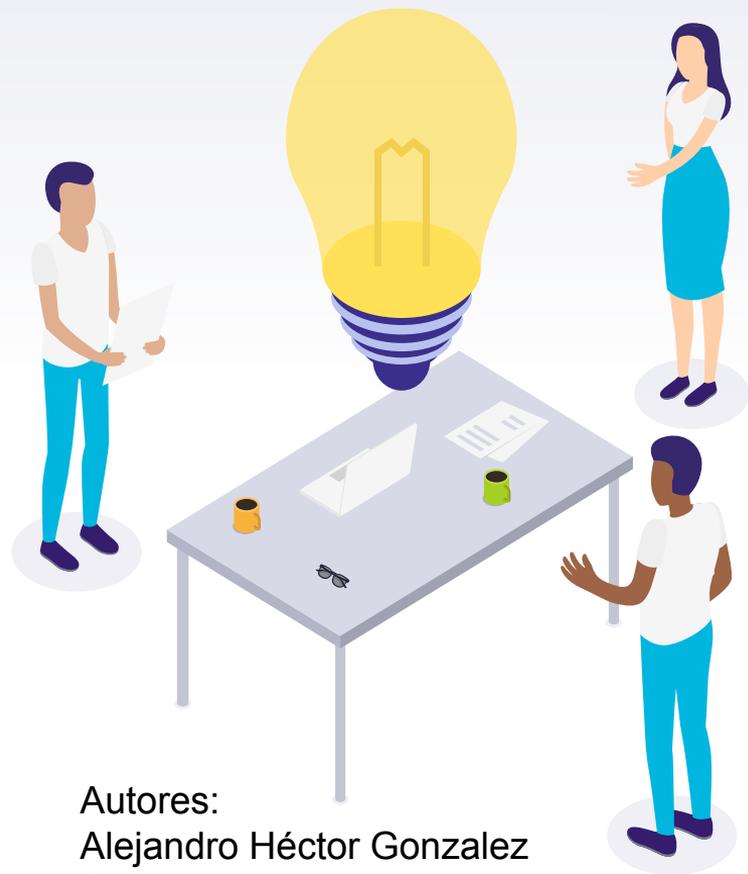


MÓDULO POO

Herencia y polimorfismo



Autores:
Alejandro Héctor Gonzalez
Silvana Lis Gallo
Junio 2021



RESUMEN

En esta clase se trabajan los conceptos de herencia y polimorfismo en POO. Se explica el manejo de la cláusula "super" y se explica el concepto de "upcasting"

Palabras clave

herencia, polimorfismo, upcasting, super

Introducción

- Diferentes tipos de objetos con comportamiento y características 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 comportamiento y características comunes.

Triángulo

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

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.

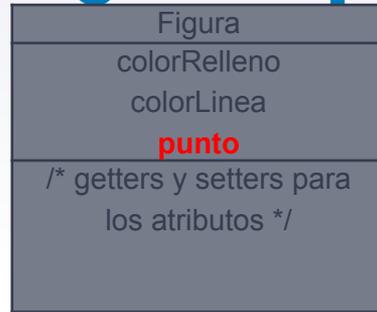


Figura es la **superclase** (clase padre o base) de Triángulo y Círculo.

define atributos y comportamiento común

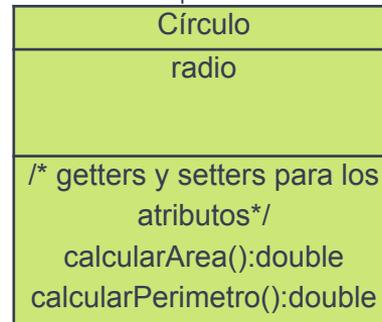
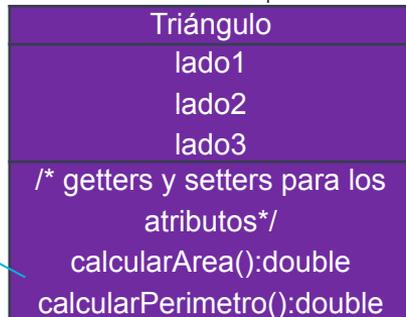
define constructores (no heredables, si “invocables”)

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

heredan atributos y métodos de Figura

definen atributos y métodos propios

definen constructores.



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

Búsqueda de método en la jerarquía de clases.

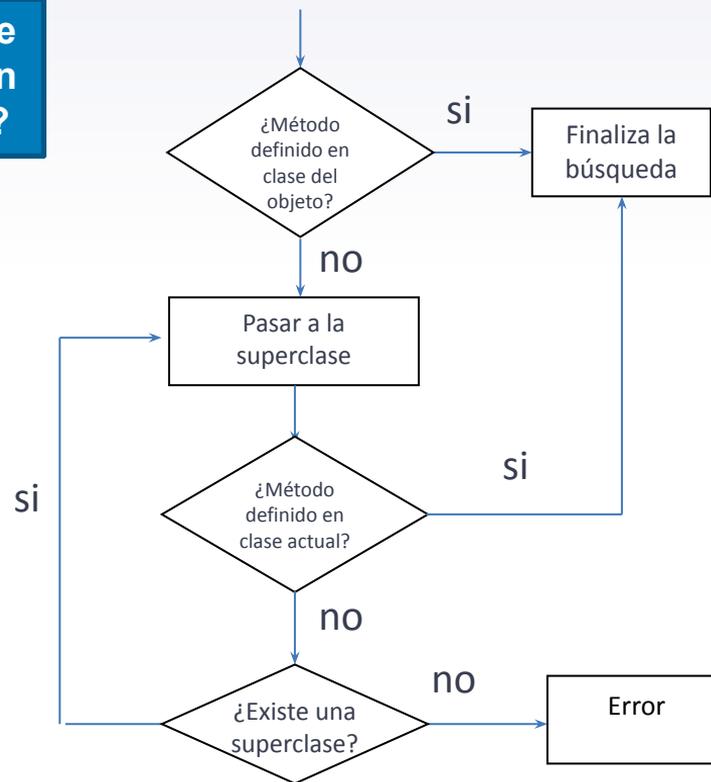
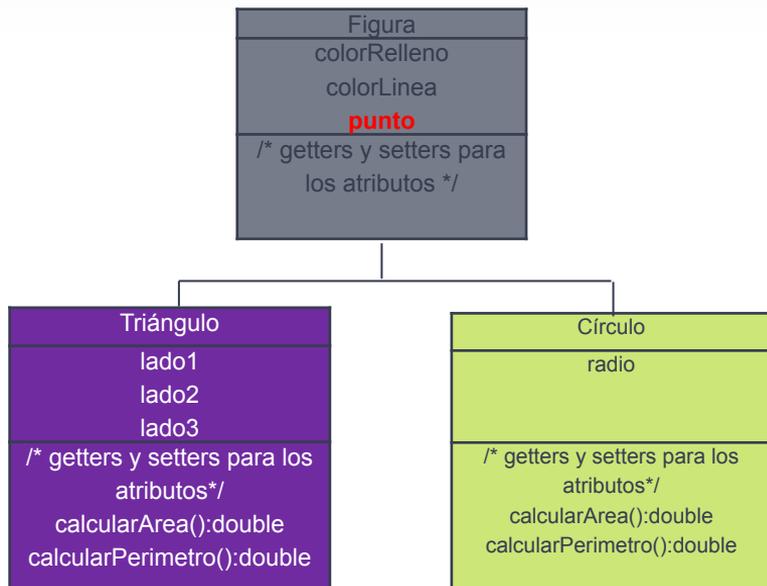
▶ Ejemplo

```
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 */  
}
```

Herencia en Java

- ▶ Si no se especifica una superclase con *extends*, *extiende* por defecto la clase `Object`.
- ▶ Los atributos declarados en la superclase los *hereda* la subclase, 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 públicos heredados.
- ▶ La subclase *hereda* métodos de instancia.
- ▶ La subclase puede declarar nuevos atributos.
- ▶ La subclase puede declarar nuevos métodos.
- ▶ La subclase puede declarar constructores propios

Herencia ejercicio (1)



- ▶ Definir una jerarquía de clases para representar Figuras Geométricas (triángulo y círculo).
- ▶ Las Figuras Geométricas tienen las siguientes características comunes: color de relleno, color de línea y su ubicación en el plano. Sin embargo, cada una tiene características propias:
 - ▶ Un triángulo se caracteriza por el tamaño de sus tres lados.
 - ▶ Un círculo se caracteriza por el radio.
- ▶ Las Figuras Geométricas poseen comportamiento común: deben saber responder cuál es el color de relleno y línea, cuál es su punto de origen. Sin embargo, cada una debe calcular su área y perímetro de forma distinta, y además devolver/modificar el valor de sus atributos propios.

Se pide:

- ▶ **Defina constructores en las clases Triángulo y Círculo.** *No defina constructores en Figura.*
- ▶ Realice un programa que instancie un triángulo y un círculo e imprima el área y perímetro de cada uno en consola.

Herencia ejercicio (2)

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

```
public class Triangulo{
....
public void dibujar(){
....
System.out.println( "Color de Línea " + this.getColorLinea() +
"Color de Relleno" + this.getColorRelleno +
"Punto: " + this.getPunto().toString() +
"Lados: " this.getLado1() +
this.getLado2() +
this.getLado3());
}
}
```

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

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

¿Cómo lo invoco? }

Herencia ejercicio (3)



- ▶ Ahora añadiremos el comportamiento para que las figuras se *dibujen* *void dibujar()*
- ▶ Todas las figuras se dibujan armando un string con su color de línea, su color de relleno y la ubicación en el plano. Además:
 - ▶ Los triángulos se dibujan con el tamaño de sus lados.
 - ▶ Los círculos se dibujan con el radio.
- ▶ Ejemplo `void dibujar() en clase Triángulo`

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

`void dibujar() en clase Círculo`

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

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?

La referencia super

- ▶ Dentro de un *método de instancia* o de un *constructor*, la referencia *super* representa al 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 explícita, Java invoca *implícitamente* 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)`.

Triangulo:

Color de Línea: negro

Color de Relleno: azul

Ubicación: (100,100)

L1: 5.0

L2: 10.2

L3: 8.0

Circulo:

Color de Línea: negro

Color de Relleno: azul

Ubicación: (100,100)

Radio: 5.0

```
public class Figura{
```

```
...
```

```
public void dibujar(){
```

```
    System.out.println ( "Color de Línea " + this.getColorLinea() +
```

```
        "Color de Relleno" + this.getColorRelleno +
```

```
        "Punto: " + this.getPunto().toString() )
```

```
}
```

```
}
```

```
public class Circulo extends Figura{
```

```
...
```

```
public void dibujar(){
```

```
    System.out.println("Circulo: " );
```

```
        super.dibujar();
```

```
    System.out.println("Radio: " + radio );
```

```
}
```

```
}
```

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

La referencia super. Ejercicio



- ▶ Defina el siguiente constructor en la clase Figura, invóquelo desde los constructores de las clases Triangulo y Circulo.

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

- ▶ Defina el siguiente método dibujar en la clase Figura, utilícelo en los métodos dibujar de las clases Triángulo y Círculo.

```
public void dibujar(){  
    System.out.println( "Color de Línea " + this.getColorLinea() +  
        "Color de Relleno" + this.getColorRelleno +  
        "Punto: " + this.getPunto().toString())  
}
```

Clase abstracta características



- ▶ Una clase abstracta puede heredar o extender cualquier clase (independientemente de que esta sea abstracta o no).
- ▶ Una clase abstracta *puede heredar de una sola clase* (abstracta o no).
- ▶ Una clase abstracta *puede tener métodos que sean abstractos* o que no lo sean
- ▶ En java concretamente en las clases abstractas la palabra **abstract** es obligatoria para definir un método abstracto (así como la clase).
- ▶ En una clase abstracta pueden existir variables static con cualquier modificador de acceso (public, private).

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

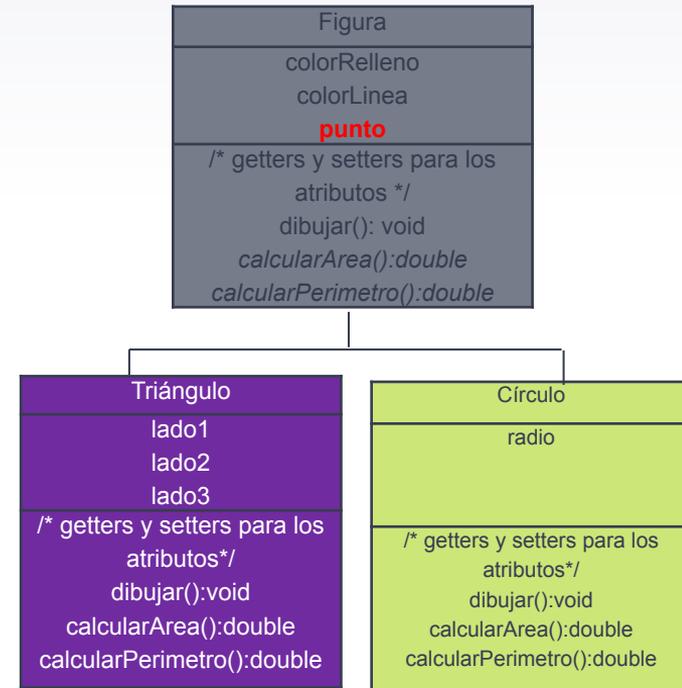
- ▶ Métodos sin implementación en la clase que lo declara. Las **subclases** tienen obligación de implementarlos.
- ▶ Ejemplo: calcularArea y calcularPerimetro de Figura.
- ▶ Declaración en Java:
 - ▶ encab `public abstract TipoRetorno nombreMetodo(lista parámetros);`

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.

Polimorfismo

- ▶ Más sobre polimorfismo en:

- ▶ <https://www.luaces-novo.es/polimorfismo-en-java/>

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:

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

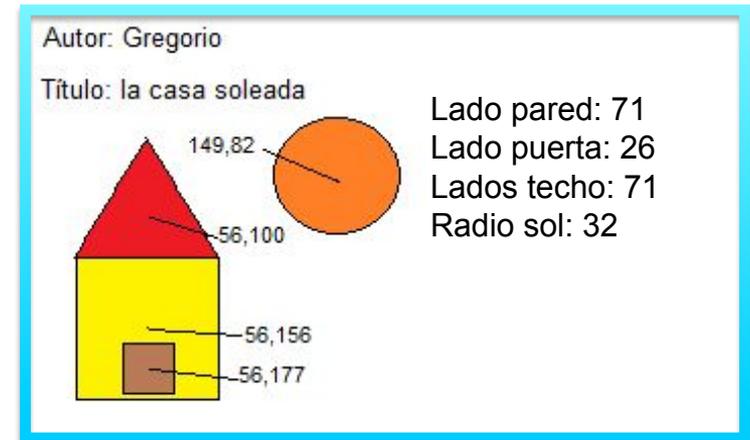
- ▶ Responder: ¿Qué mensajes le puedo enviar al objeto a través de la referencia circ? ¿Qué mensajes a través de la referencia fig?

Aplicación. Ejercicio



- ▶ Utilizando la jerarquía de figuras, generar una aplicación que permita realizar el dibujo mostrado.
- ▶ Un dibujo se caracteriza por su *título*, el *nombre* de su autor, y las *figuras* que lo componen (el máximo es establecido en la creación del dibujo).
- ▶ El dibujo debe saber *mostrarse* en consola, a través de dibujar las figuras que lo componen; responder si *está completo*, es decir si contiene el máximo de figuras admitidas; permitir *agregar* una nueva figura.

- ▶ Ayuda:
 - ▶ Agregar la clase Cuadrado a la jerarquía de figuras.
 - ▶ Implementar la clase Dibujo
 - ▶ ¿Atributos? ¿Métodos?
 - ▶ Implementar el programa que instancie el dibujo de la imagen y lo muestre.



Aplicación. Ejercicio



▶ Diagrama de clases

