élogos

# HERENCIA Y SOBRESCRITURA DE METODOS

élogos

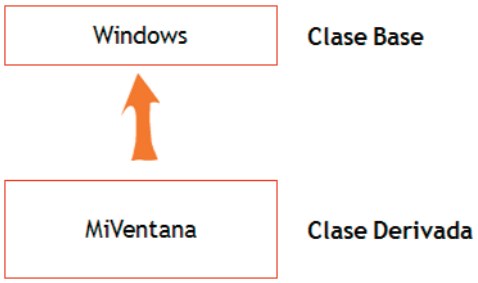
# ÍNDICE

## HERENCIA Y SOBRESCRITURA DE MÉTODOS

# 1. Herencia

Nos permite crear una clase a partir de otra clase heredando todas las características de la ya existente. A la clase que se hereda se le denomina la clase base o superclase y a la que hereda la clase se le denomina clase derivada o subclase.

Para crear una clase que herede otra clase, se emplea la palabra **extends.**
De esta forma las clases quedan organizadas de forma jerárquica.




```
class MiVentana extends Window {
// Cuerpo de la clase.
}
```

**Características de la herencia**

**Conversión implícita  explicita con referencias a objetos**

Se pueden hacer conversiones, convertir una referencia a clase derivada en una referencia a clase base implícitamente o convertir una referencia clase base en una referencia clase derivada utilizando casting.

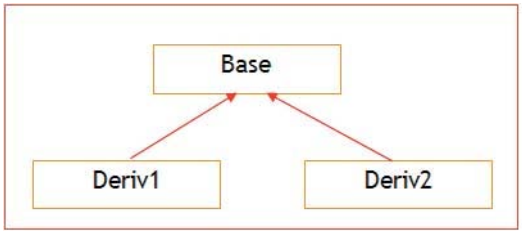
**Agregación y Composición** son otro tipo de relaciones entre objetos.

Agregación: Consiste en la creación de un objeto desde una clase que se denomina objeto contenedor, el cual tiene referencias a objetos de otra clase a los que llamamos objetos contenidos. La vida de los objetos contenidos no está limitada a la vida del contenedor, ambos son independientes.

Composición: Consiste en que hay un objeto que llamamos contenedor el cual está formado por objetos de otra clase a los que llamamos contenidos. De forma que la vida de los contenidos está limitada a la del contenedor.

**Conversiones entre clases hermanas**

Entre clases hermanas no se pueden realizar conversiones ni con el operador casting.




```
Deriv1 d1 = new Deriv1();
Deriv2 d2 = new Deriv2();
d1 = d2;          // Error.
d1= (Deriv1) d2; // Error.
```

**El operador instanceof**

Sirve para preguntar si un objeto al que se está apuntando con una referencia es de una determinada clase.

El formato es el siguiente:

**<Referencia> instanceof <clase>**

**El método finalize()**

Es un método que se ejecuta justo antes de que el sistema de recogida de basura de Java libere la memoria del objeto. El método finalize() se puede redefinir para poner las operaciones que se quieran hacer antes de destruir el objeto.

Sólo es ejecutado por JVM.

Se encuentra en la clase Object, vacío. En las clases derivadas es donde se redefine.

## 2. Ejecución de constructores en la herencia. Uso de super y this

**Orden de Ejecución de los constructores en la herencia**

La clase derivada hereda todos los métodos y atributos excepto el constructor de la clase base (ni constructor por defecto ni los constructores generales).

Cuando se crea un objeto de la clase derivada, primero se ejecuta el constructor por defecto de la clase base y posteriormente el constructor de la clase derivada.

# Herencia y sobrescritura de métodos

Ejemplo

```
class Punto{
	int x,y;
	Punto(){
		this.x=-1;
		this.y=-1;
	}
	Punto(int x, int y){
		this.x=x;
		this.y=y;

	}

}
Pixel px= new Pixel(1);
```

```
class Pixel extends Punto{
	int color;
	Pixel(int color){
		this.color= color;
	}
}
```

Al crearse el objeto Pixel se ejecutará primero el constructor sin parámetros de Punto por lo que el estado de los atributos de crear el objeto seria: x=-1, y=-1, color=1

**El operador super:**

1. Indica qué constructor de la clase base se quiere ejecutar cuando se crea un objeto de la clase derivada.
2. Indica que se quiere ejecutar un método de la clase base que existe también en la clase derivada.

```
class Pixel extends Punto{
	int color;
	Pixel(int x, int y, int color){
	super(x,y); // llama al constructor Punto (int,int)
	this.color= color;
		}
	}

	Pixel px= new Pixel(3,6,8);// tras ejecutar esta instrucción
				//los valores de los atributos quedarían
					//x:3, y:6, color:8
```

**El operador this:**

1. Es una referencia sobre el objeto que se ejecuta en el momento actual.
2. Llama a los atributos y métodos del objeto.
3. Llama alguno de los constructores de la clase. Esto solo puede realizarse desde otro constructor.

## 3. Sobrescritura de métodos

La sobrescritura de un método consiste en volver a definir en la clase derivada un método que ha sido heredado de la clase base.

```
class Punto{
    int x, y;
    Punto(int x, int y){
       this.x=x;
       this.y=y;
    }
 public void dibujar(){
  System.out.println("coordenadas:"+x+","+y);
    }
}
```

```
class Pixel extends Punto{
    int color;
    Pixel(int x, int y, int color){
       super(x,y);
       this.color=color;
    }
    //sobrescritura de dibujar
    public void dibujar(){
       super.dibujar();//llama a la
versión original
       System.out.println("color:
"+color);
    }
}
```

Al llamar al método con un objeto de la subclase, se ejecutará la nueva versión del método definida en la misma.

**Reglas de sobrescritura de métodos**

1. La nueva versión del método debe tener exactamente el mismo nombre, parámetros y tipo de devolución que el original. Si cambiamos el número de parámetros, estaremos ante un caso de sobrecarga, no sobrescritura.
2. La nueva versión del método puede tener un ámbito menos restrictivo que el original.
3. Desde el interior del método se puede llamar a la versión original a través de super .
4. En la creación del nuevo método en la subclase, se puede utilizar la anotación @Override delante de la declaración de dicho método para indicar al compilador que estamos realizando una sobrescritura.

**El modificador final**

El modificador final puede utilizarse en la definición de una clase y método.

**Clase final** es aquella que no puede ser heredada por otras clases sino provocaría un error de compilación. Existen numerosas clases finales como por ejemplo String o las clases de envoltorio (Integer, Long, etc.)

# Herencia y sobrescritura de métodos

**Método final** el método no podrá ser sobrescrito por las clases derivadas. El motivo por el que se define un método como final es porque se considera completo y, por tanto, su funcionalidad no debe ser alterada en las subclases.

Ejemplo de métodos finales en Java estándar son los métodos wait() y notify() de la clase Object.

**El modificador protected**

Cuando se utiliza la herencia se puede emplear el modificador protected para el acceso a los atributos y métodos que solo podrán ser accedidos desde la propia clase y desde la clase derivadas de ella.