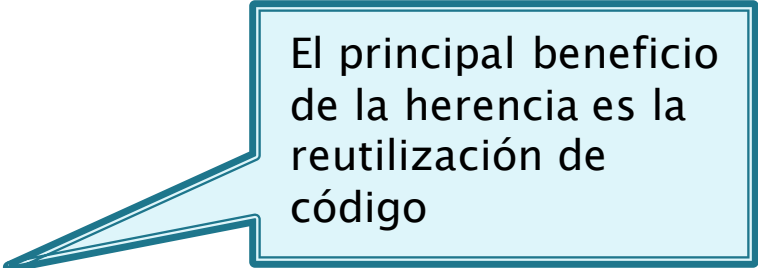


# Herencia

# Definición

- Herencia es una característica que permite crear nuevas clases a partir de clases ya existentes, de forma que la nueva clase adquiera (herede) los miembros de la ya existente.
- A la clase “padre” se le conoce como superclase, mientras que la “hija” es la subclase
- Se emplea **extends**:

```
class Clase1{  
    public void metodo(){}  
}  
class Clase2 extends Clase1{  
    //automaticamente adquiere metodo()  
}
```



El principal beneficio de la herencia es la reutilización de código

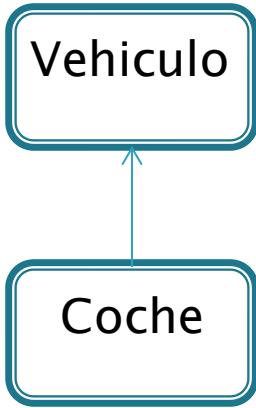
# Consideraciones

- Una clase solo puede heredar otra clase, aunque la superclase puede heredar a su vez a otra y así hasta n niveles.
- Varias clases pueden heredar la misma clase
- Los miembros privados de la superclase no son accesibles directamente desde la subclase
- Si queremos que una clase no se pueda heredar, la definiremos con *final*:

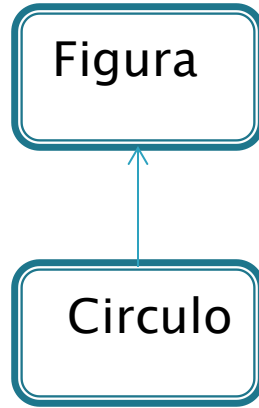
```
final class Clase1{ //no podrá ser heredada
    public void metodo(){
    }
    class Clase2 extends Clase1{} //error de compilación
```

# Relación “Es un”

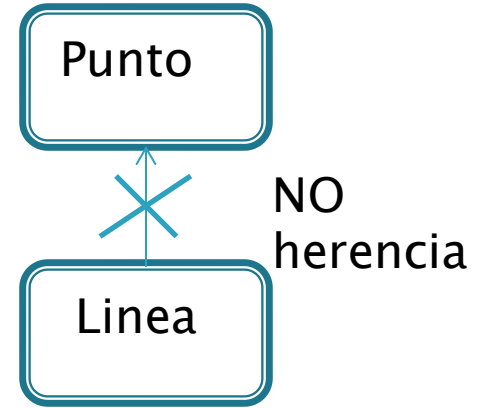
- Entre la subclase y la superclase hay una relación de tipo “Es un”.
- Un objeto de la subclase “es un” objeto del tipo de la superclase:



Un Coche “es un”  
Vehiculo



Un Circulo “es un”  
Figura



Una Linea NO “es un”  
Punto

# Herencia de Object

- Todas las clases Java heredan Object
- Si una clase no hereda explícitamente otra clase, implícitamente heredarán Object

<pre>class Test{ }</pre>		<pre>class Test extends Object{ }</pre>
------------------------------	------------------------------------------------------------------------------------	---------------------------------------------

- Todas las clases disponen de los métodos de Object, entre ellos: toString(), equals() y hashCode()

# Llamada a constructor de la superclase

➤ Toda clase incluye de forma implícita en sus constructores como primera línea de código la instrucción `super();`, que es una llamada al constructor sin parámetros de la superclase

```
class Clase1{  
    public Clase1(int a){  
        System.out.println(a);  
    }  
}
```

equivale

```
class Clase1{  
    public Clase1(int a){  
        super();  
        System.out.println(a);  
    }  
}
```

➤ Si la superclase no dispone de constructor sin parámetros, se produce un error de compilación en la subclase

# Ejemplos

Llamaría al sin parámetros de Object

```
class Clase1{
    Clase1(){System.out.println("C1");}
}
class Clase2 extends Clase1{
    Clase2(){System.out.println("C2");}
}
class Clase3 extends Clase2{
    Clase3(){System.out.println("C3");}
}
```

Clase3 c=new Clase3();

Muestra

C1  
C2  
C3

Constructor por defecto llama a constructor sin parámetros de superclase

Llamada a constructor por defecto

```
Class Clase1{
    Clase1(){System.out.println("C1");}
}
class Clase2 extends Clase1{
}
class Clase3 extends Clase2{
    Clase2(){System.out.println("C3");}
}
```

Clase3 c=new Clase3();

Muestra → C1  
C3


Error de compilación!

```
class Clase1{
    Clase1(int a){}
}
class Clase2 extends Clase1{
    Clase2(){
        System.out.println("C2");
    }
}
```

# Llamada a constructor con parámetros

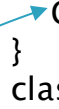
➤ Es posible llamar a otro constructor de la superclase que no sea el constructor sin parámetros, para ello, utilizaremos la instrucción: **super(argumentos)**

```
class Clase1{
    Clase1(int a){}
}
class Clase2 extends Clase1{
    Clase2(int x){
        super(x); //debe ser la primera instrucción
        System.out.println("C2");
    }
}
```

A blue arrow originates from the **super(x);** line in the Clase2 constructor and points to the **Clase1(int a){}** line in the Clase1 class definition.

En este caso no hay llamada a *super*, ni implícita ni explícita

```
class Clase1{
    Clase1(int a){}
}
class Clase2 extends Clase1{
    Clase2(int x){
        super(x);
        System.out.println("C2");
    }
    Clase2(){
        this(10); //llamada al otro constructor
    }
    Clase2(String s){
        super(2);
        this(); //error de compilación!!
    }
}
```

A blue arrow originates from the **super(x);** line in the Clase2 constructor and points to the **Clase1(int a){}** line in the Clase1 class definition.



# Clases finales

- Una clase final es una clase que no puede ser heredada.
- Se declaran con la palabra *final*:

```
final class ClaseFinal{  
    :  
}
```

- Si se intenta heredar una clase final, se producirá un error de compilación.
- Java SE incluye bastantes clases finales, como por ejemplo String o las clases de envoltorio.