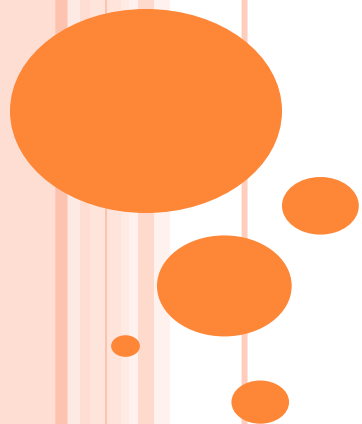
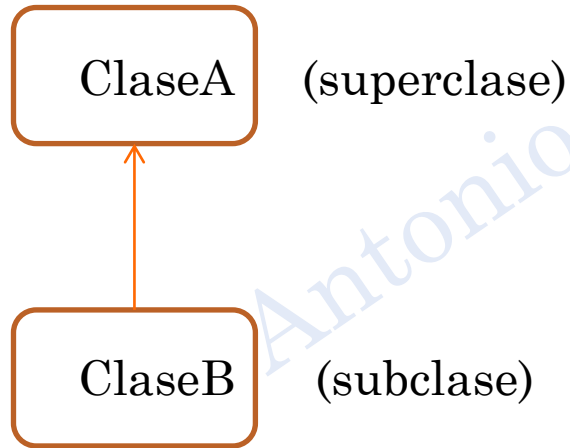


HERENCIA



DEFINICIÓN

➤ Herencia es la capacidad de crear clases, que adquieran automáticamente los miembros (métodos y atributos) de otra clase ya existente.

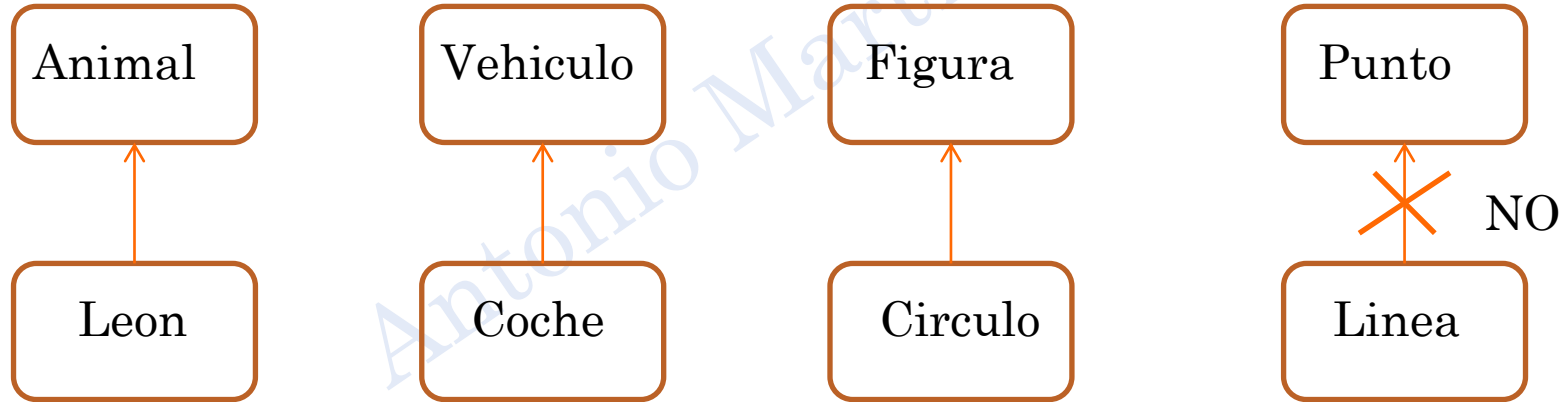


```
class ClaseB extends ClaseA{  
    //cuenta con los métodos de  
    //ClaseA y puede añadir los  
    //suyos propios  
}
```

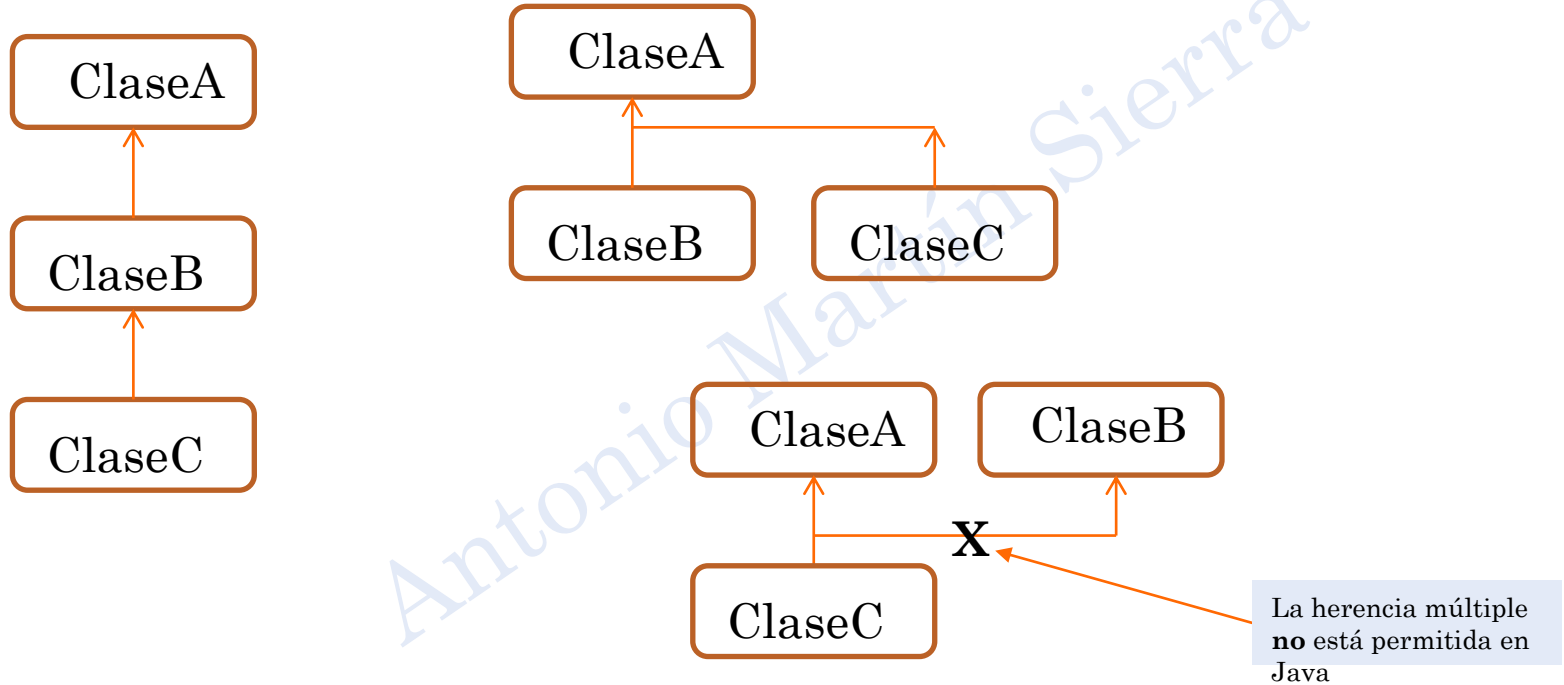


EJEMPLOS

➤ Herencia establece una relación "es un", entre las clases, pues un objeto de la subclases "es un" tipo de la superclase



RESTRICCIONES



VENTAJAS DE LA HERENCIA

➤ La principal ventaja que proporciona la herencia es la **REUTILIZACIÓN DE CÓDIGO** al permitir que las subclases cuenten con los métodos ya creados en las superclases:

```
class ClaseA{
    public void metodo1(){
        :
    }
    public int metodo2(){
        :
    }
}

class ClaseB extends ClaseA{
    public void metodo3(){
        :
    }
}
```



```
ClaseB cb=new ClaseB();
//puede llamar a los métodos heredados
cb.metodo1();
cb.metodo2();
//también a los propios
cb.metodo3();
```



CONSTRUCTORES EN LA HERENCIA I

➤ **Implicítamente, al crear un objeto de una subclase, desde sus constructores se hace primero una llamada al constructor sin parámetros de la superclase**

```
class Clase1{
    public Clase1(){
        System.out.println("Constructor clase 1");
    }
}
class Clase2 extends Clase1{
    public Clase2(){
        System.out.println("Constructor clase 2");
    }
}
```

Clase2 c2=new Clase2();



Constructor clase 1
Constructor clase 2



CONSTRUCTORES EN LA HERENCIA II

➤ Si no existiese constructor sin parámetros en la superclase, se produciría error de compilación

```
class Clase1{  
}  
class Clase2 extends Clase1{  
    public Clase2(){  
        System.out.println("Constructor clase 2");  
    }  
}
```

Clase2 c2=new Clase2();



Constructor clase 2

Aquí no hay problema porque se llama al constructor por defecto de Clase1

```
class Clase1{  
    public Clase1(int n){  
    }  
}  
class Clase2 extends Clase1{  
    public Clase2(){  
        System.out.println("Constructor clase 2");  
    }  
}
```

Error de compilación en Clase2, pues su constructor incluye una llamada al constructor sin parámetros de Clase1 y no existe



CONSTRUCTORES EN LA HERENCIA III

- Se puede llamar a un constructor distinto al sin parámetros utilizando: `super(arg1,arg2,...);`
- Por defecto, todo constructor incluye `super();`

```
class Punto{
    int x,y;
    public Punto(int x, int y){
        this.x=x;
        this.y=y;
    }
}
class Punto3D extends Punto{
    int z;
    public Punto3D(int x, int y, int z){
        super(x,y);
        this.z=z;
    }
}
```

Debe ser la primera
instrucción del constructor



LA CLASE OBJECT

- **Todas las clases Java heredan Object en algún punto de la jerarquía.**
- **Si una clase no hereda ninguna otra de forma explícita, implícitamente heredaré Object**

```
class Clase1{  
    :  
}
```



```
class Clase1 extends Object{  
    :  
}
```



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.

