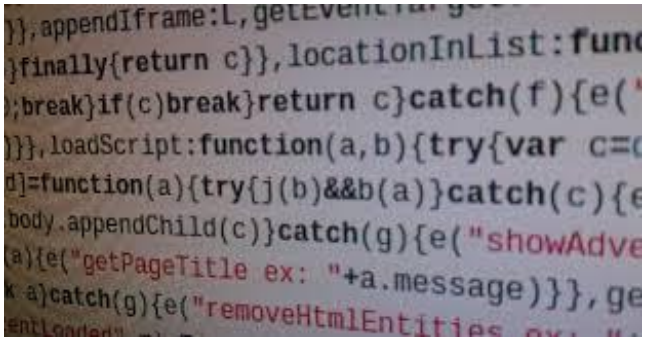



# UT 11

# UTILIZACIÓN AVANZADA DE CLASES

IES JUAN DE LA CIERVA
DPTO. INFORMÁTICA

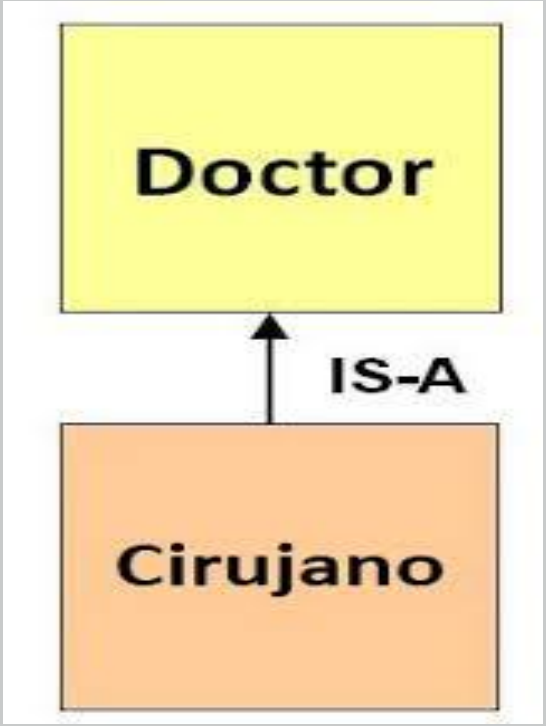
# CONTENIDOS DE LA PRESENTACIÓN

**Objetivos**

En esta unidad, se pretende que el alumno sepa manejar las relaciones entre clases, la herencia, polimorfismos, siendo capaz de crear sus propias clases en función de la necesidad de cada momento.

**Contenidos**

- Herencia.
- Superclases y subclases.
- Constructores y herencia.
- Clases y métodos: abstractos y finales.
- Polimorfismos.




# Herencia

# Herencia

Herramienta de la POO que permite crear clasificaciones jerárquicas.

Permite crear una clase general (clase padre) que define rasgos comunes a una serie de elementos relacionados.

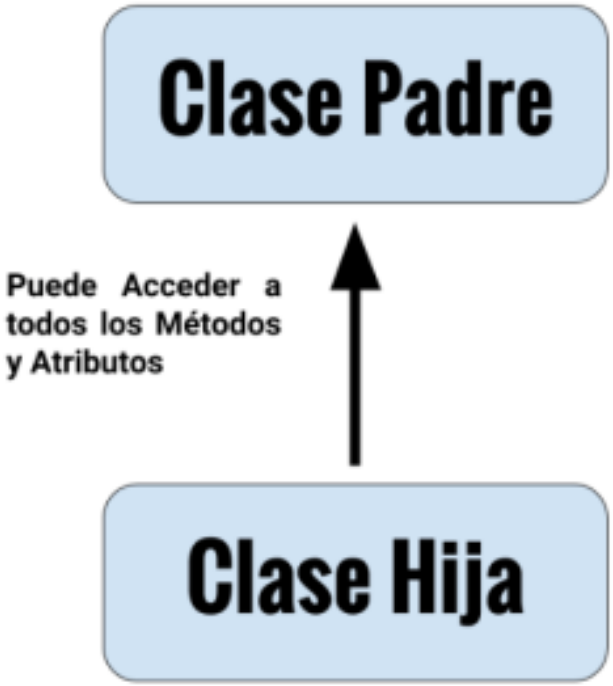
Esta clase general se puede heredar por otras más concretas con elementos própios (clases hijas)

**Clase heredada ……: Superclase**

**Clase que realiza la herencia: Subclase**

Una Subclase es una versión especializada de la Superclase.

La Subclase hereda todas las variables y métodos definidos por la superclase y añade sus propios elementos exclusivos.



# Composición vs Herencia

**Herencia:**

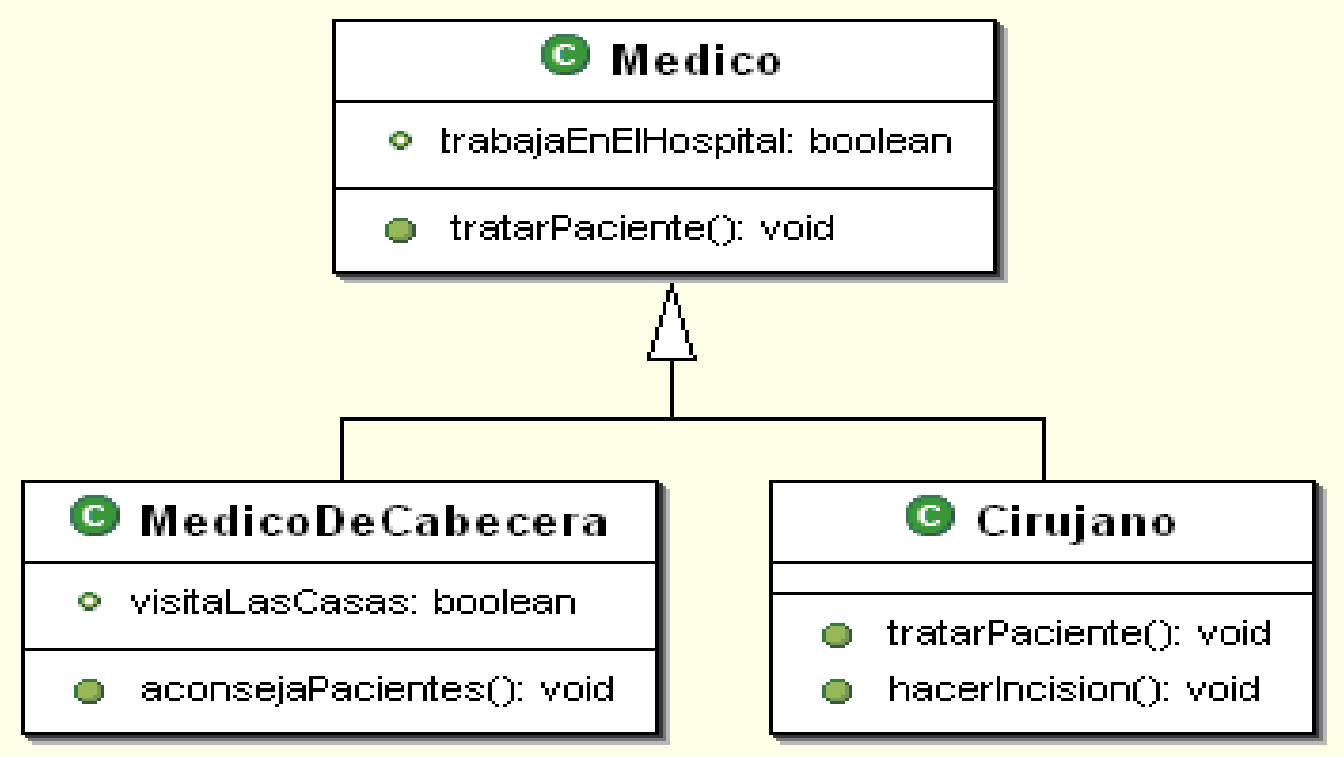
Clase hijo **<<es-un>>** Clasebase o padre

*todos los objetos(miembros) de una clase hija (subconjunto) son objetos también de la clase padre (conjunto).*

**Composición :**

Clase contenedora **<<tiene-un>>** Clase contenida

# Herencia. Ejemplo



# Herencia:UTILIDADES-VENTAJAS

- Modelado de la realidad:  modela las relaciones de especialización/generalización entre las entidades del mundo real.
- Evita redundancias, facilita la reutilización de código.
- Sirve de soporte para el polimorfismo (lo veremos más adelante)

# Tipos de Herencia

Existen dos tipos de herencia.

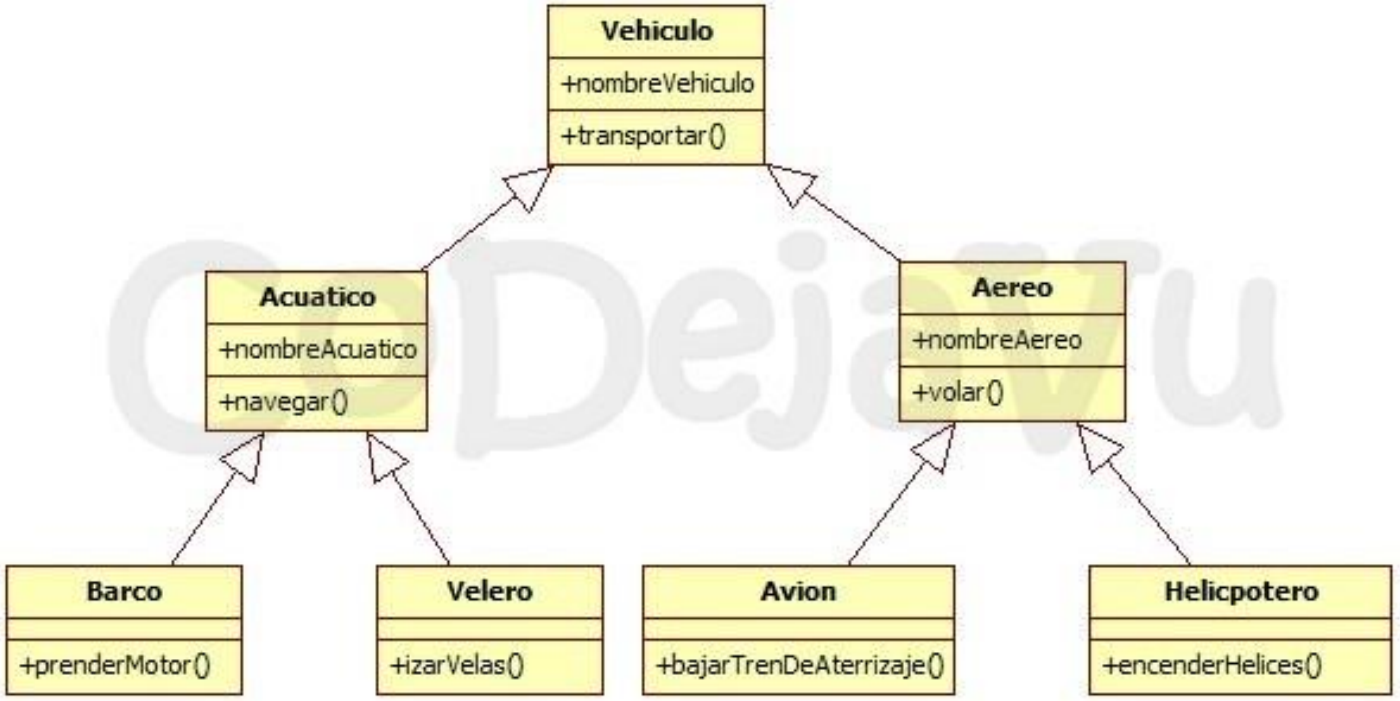
- **Herencia simple**: una clase solo puede tener un padre, por lo tanto la estructura de clases será en forma de árbol.

- **Herencia múltiple**: Una clase puede tener uno o varios padres. La estructura de clases es un grafo.

  **(*)No soportada en Java**

# Herencia Simple

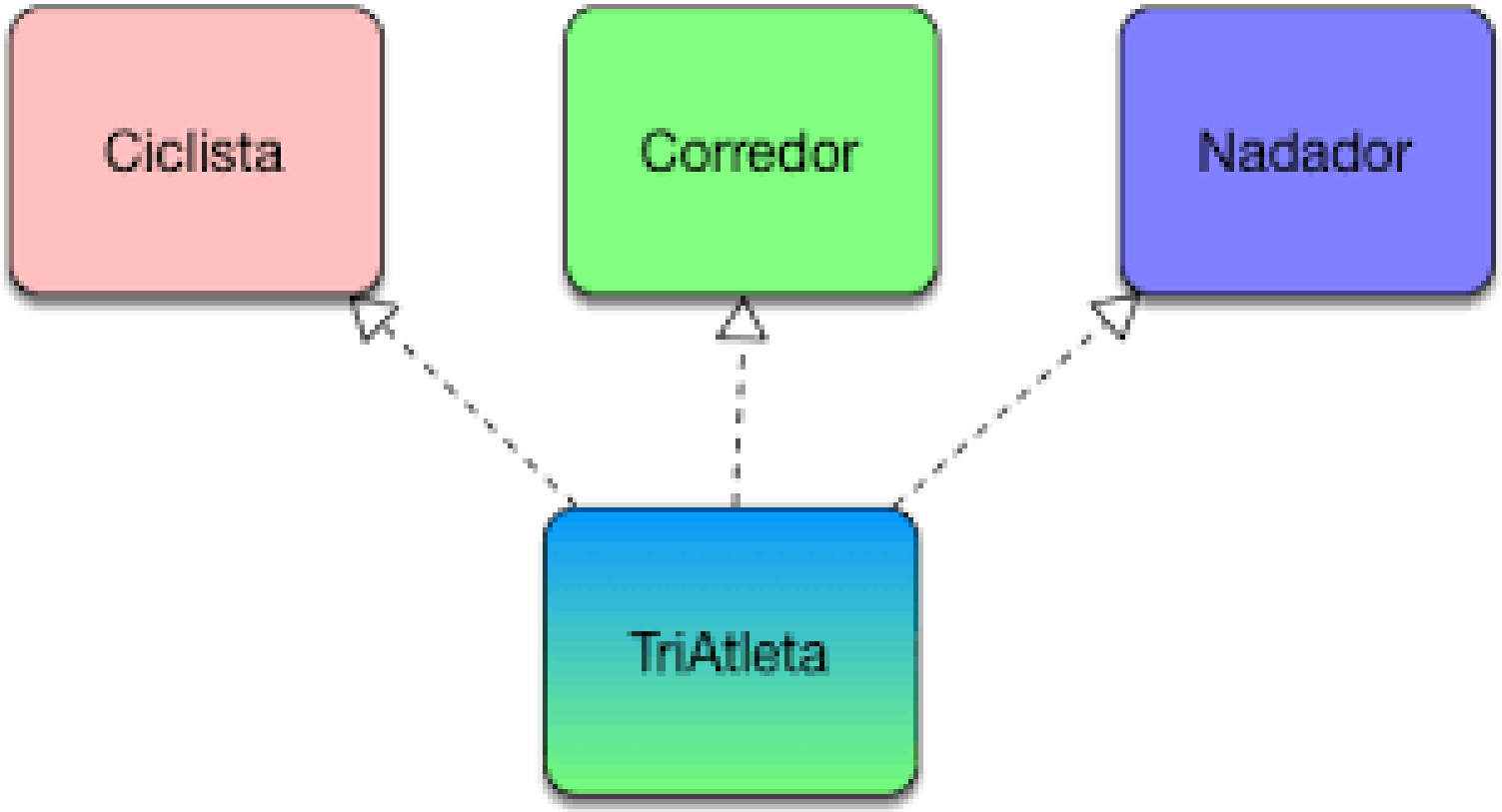
Una clase solo puede tener un padre, por lo tanto la estructura de clases será en forma de árbol.



# Herencia Múltiple

Una clase puede tener uno o varios padres. La estructura de clases es un grafo.

**(*)No soportada en Java**

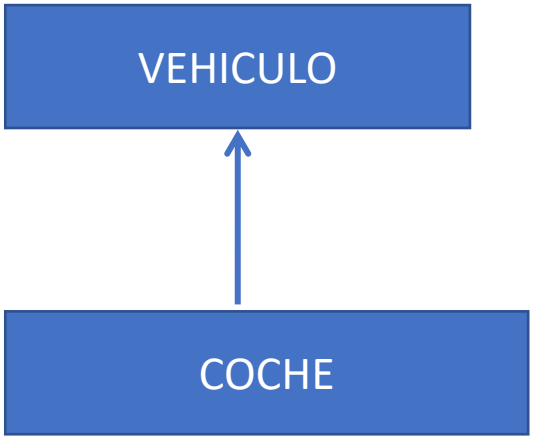


# SITUACIONES EN LAS QUE SE PUEDE APLICAR LA HERENCIA

- Especialización
- Extensión
- Especificación
- Construcción ?????

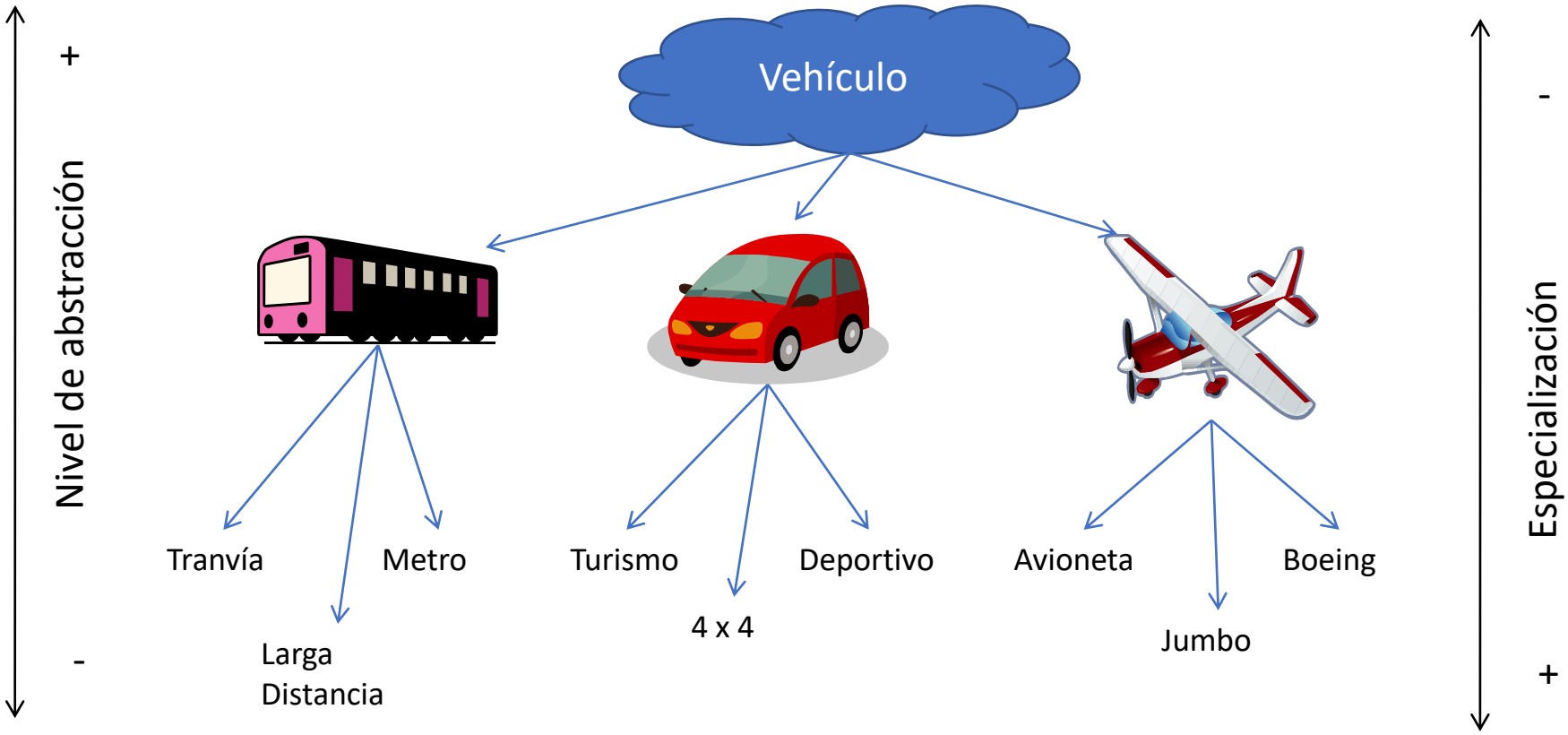
# Especialización

- Dado un concepto B y otro concepto A que representa una especialización de B, entonces puede establecerse una relación de herencia entre las clases de objetos que representan a A y B.

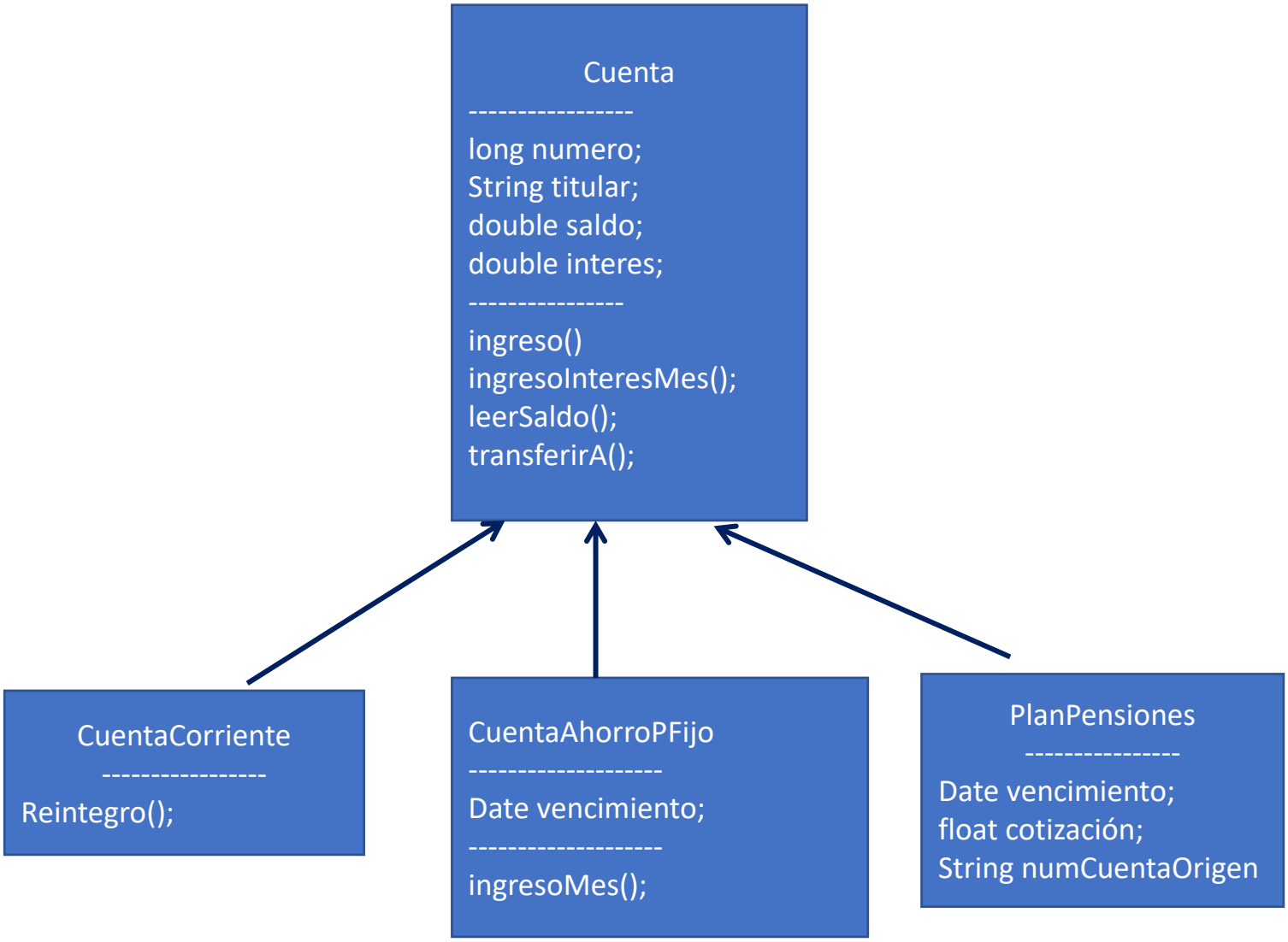



Un coche **es un** vehículo

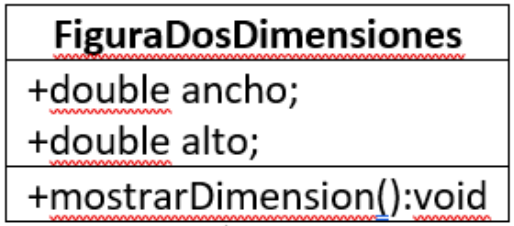
# *La herencia como especialización*

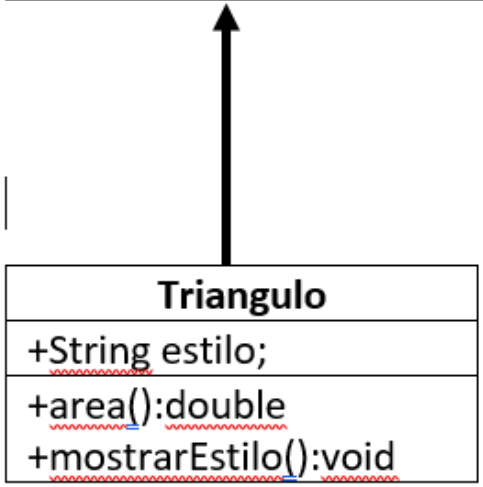


# Especialización: Ejemplo



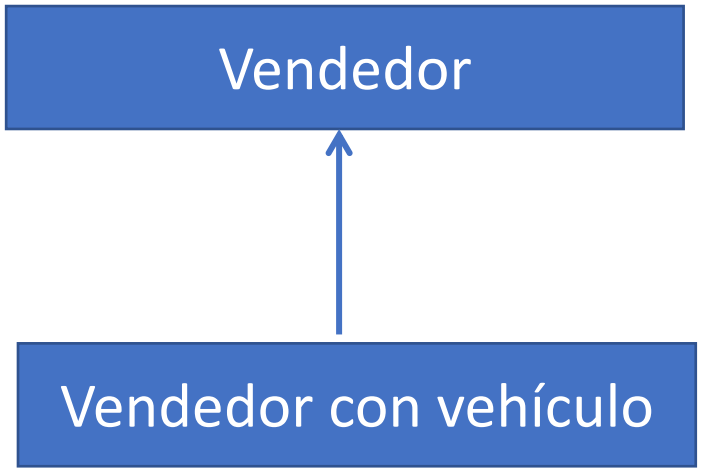
# Especialización: Ejemplo



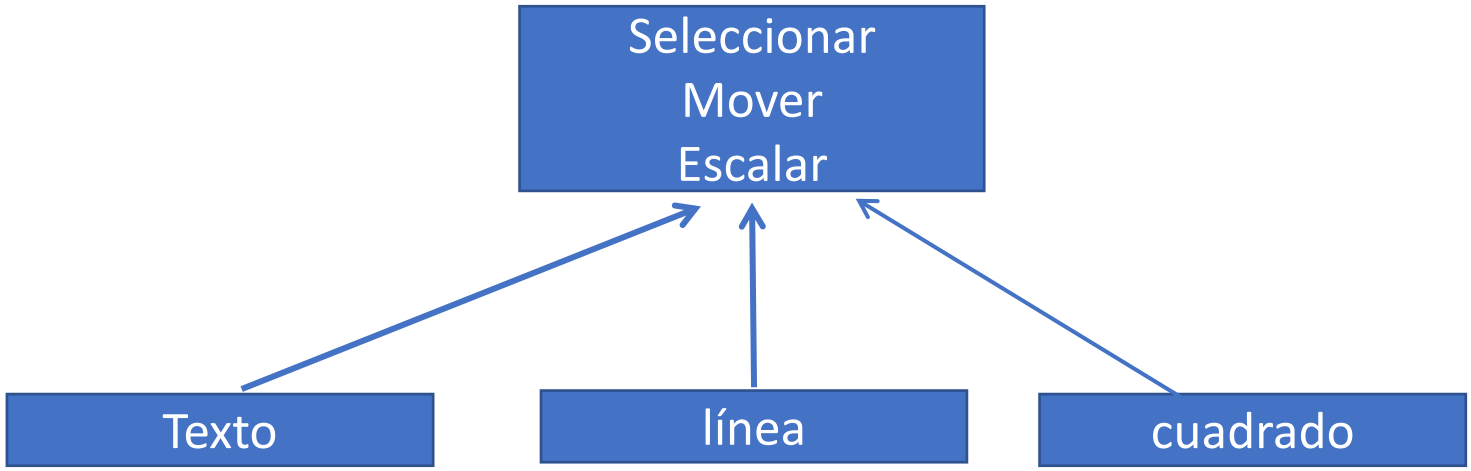

# Extensión

Una clase puede servir para extender la funcionalidad de una superclase sin que represente necesariamente un concepto más específico.



# Especificación

- Una superclase puede servir para especificar la funcionalidad mínima común de un conjunto de descendientes.



# Construcción ( )

- Una clase puede construirse a partir de otra, simplemente porque la hija puede aprovechar internamente parte o toda la funcionalidad del padre, aunque representen entidades sin conexión alguna.
- Ojo con esto, aunque se puede hacer, es un uso inadecuado de la herencia, ya que conecta clases sin relación alguna.
- En general debemos respetar la relación “es un”, la clase hija <<es un >> clase padre.

# Construcción ( )

Es decir, podríamos construir la clase alumno a partir de la clase empleado, porque consideramos que los atributos y métodos de empleado me sirven y así los aprovecho.

Pero esto no sería ortodoxo.

No se cumpliría la relación “es un”.

Un Alumno “es un” Empleado.


# Implementación

# Herencia. Implementación

Para que una clase herede las características de otra hay que utilizar la palabra clave (keyword) **extends** tras el nombre de la clase.

**[Modo de acceso] class <NombreClaseHija>**

**EXTENDS <NombreClasePadre>**

- En Java sólo se puede heredar de una clase (a la clase de la que se hereda se la llama **superclase** ).

# Implementación Ejemplo clase Triángulo

```
class FiguraDosDimensiones {
    // SUPERCLASE Figura de Dos Dimensiones
    double ancho;
    double alto;

    void mostrarDimension(){
        System.out.println("El alto y ancho de la figura son :
        "+alto+" cm. y "+ancho +" cm.");
    }

}

// SUBCLASE DE FiguraDosDimensiones para triángulos
    class Triangulo extends FiguraDosDimensiones {
        String estilo;

        double area(){
            return ancho*alto/2;
        }

        void mostrarEstilo(){
            System.out.println("El triangulo es :"+estilo);
        }
    }
```

**Ejemplo1. Figura de dos Dimensiones**

# Acceso a miembros y herencia.
# (*) A tener en cuenta.

- ✓ La herencia de una clase no anula la restricción de acceso **private**. Por tanto, aunque una subclase incluya todos los miembros de su superclase, no puede acceder a los que se hayan declarado como **private**, si no es a través de sus métodos public.
- ✓ Se heredan todos los atributos y métodos de las clases **public** y **protected ()**.
- ✓ En una jerarquía, tanto superclases como subclases pueden tener su propios constructores y cada uno crea su parte del objeto.
- ✓ Recordar que la superclase no puede acceder a los elementos de la subclase pero sí al revés.

# Acceso a miembros y herencia.
# (*) A tener en cuenta.

- ✓ La **Orden de ejecución** de los constructores en la jerarquía: *en orden de derivación , de superclase a subclase.*
- ✓ Cuando instanciamos un objeto de la subclase hay que tener en cuenta que el constructor de la superclase no se ejecuta automáticamente. Deberemos ejecutarlo mediante la clausula ***super().***

# Utilización de super

- ✓ Si existe un miembro (atributo o método) con el mismo nombre en la clase creada y en la heredada, para referirnos a la de la clase padre le anteponemos la palabra **super**.

  super.<miembro>

- ✓ Una subclase puede invocar un constructor definido por su superclase por medio del siguiente formato:

  super(lista-parámetros);

  *(*)Siempre debe ser la primera instrucción del constructor.*

  Ejemplo:

  ```
  super();
  super(par1,par2,….);
  ```

# Utilización de super

- ✓ Modifica el ejemplo Figura de Dos Dimensiones programando los contructores necesarios en las clases FiguraDosDimensiones y Triangulo.


# Herencia en Java

# Herencia en Java

- Estructura jerárquica en árbol en donde en la raíz podemos encontrar la clase **Object**, de las que heredan todas las clases.
  - Todas las clases en Java tienen un padre
  - Todos los objetos son “Object”

- Una clase derivada (o subclase) puede acceder directamente a todos los atributos y métodos heredados que no sean ***private;*** a estos se accederá a través de métodos (get(), set())

# Herencia en Java

- Se heredan todos los atributos y métodos, excepto los constructores, aunque estos últimos es posible reutilizarlos.
- Una clase **final** no puede tener subclases.
- Si un método es static en la superclase, en la subclase también.
- Si un método no es static en la superclase, en la subclase tampoco.

# La clase Object

En Java todas las clases heredan de otra clase:

- Si lo especificamos en el código con la *keyword extends,* nuestra clase heredará de la clase especificada.
- Si no lo especificamos en el código, el compilador hace que nuestra clase herede de la clase Object (raíz de la jerarquía de clases en Java).
- Ejemplo

```
public class MiClase extends Object {
    // Es redundante escribirlo puesto que el
    // compilador lo hará por nosotros.
}
```

# La clase Object

Esto significa que nuestras clases siempre van a contar con los atributos y métodos de la clase Object

- Algunos de sus métodos
  - **public boolean equals(Object o);**
    Compara dos objetos y dice si son iguales.
  - **public String toString();**
    Devuelve la representación visual de un objeto.
  - **public Class getClass();**
    Devuelve la clase de la cual es instancia el objeto.

# Redefinición de métodos

- ¿Qué pasa si en la superclase hay un método que funciona distinto a como nos gustaría que funcionara en la subclase?
- ¿Son las subclases responsables de inicializar en sus constructores las variables heredadas de las superclases?
- ¿Qué pasa si un método de la superclase no debiera aparecer en la subclase?

# Redefinir o Sobreescribir un método

Sobrescribir un método significa que una subclase reimplementa un método heredado.

- Para **sobrescribir** un método hay que respetar totalmente la declaración del método:
  - El **nombre** ha de ser el **mismo**.
  - **Los parámetros y tipo de retorno** han de ser los **mismos**.
  - El modificador de acceso no puede ser mas restrictivo.
- Al ejecutar un método, se busca su implementación de abajo hacia arriba en la jerarquía de clases.

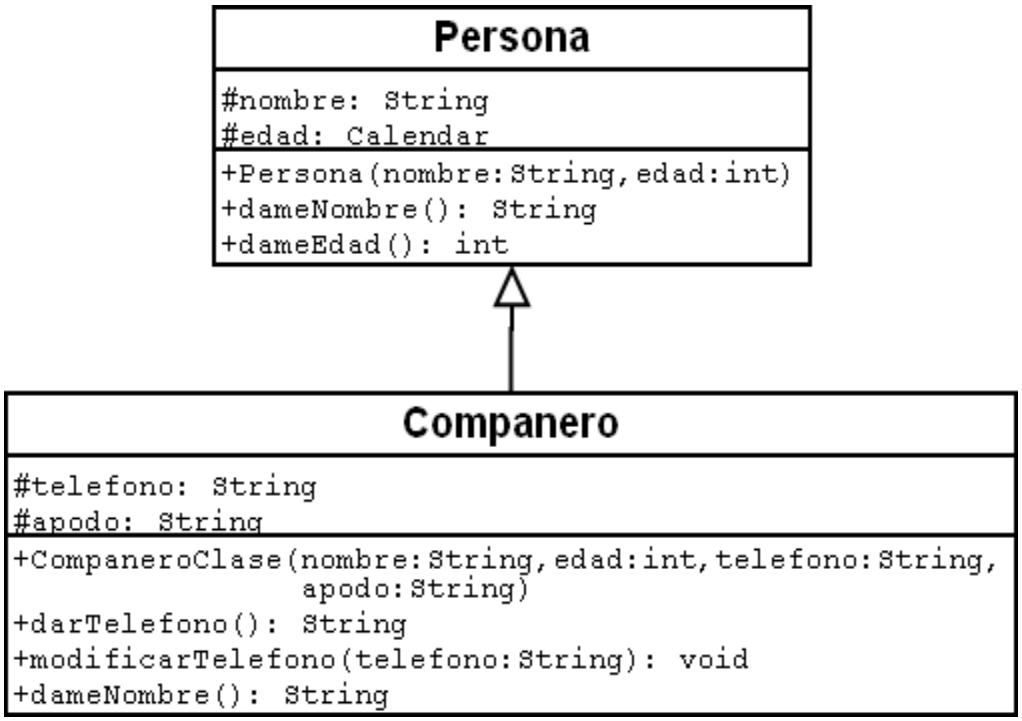
# Sobrescribir vs. Sobrecargar

Sobrecargar un método es un concepto distinto a sobrescribir un método.

- La **sobrecarga** de un método significa tener varias implementaciones del mismo método con parámetros distintos:

  - Los **parámetros** tienen que ser **distintos** ( en número o tipo).
  - El **modificador** de acceso **puede ser distinto**.
  - El **nombre** ha de ser el **mismo**.

# Redefinición de métodos




**(*)dameNombre() de la clase Companero sobreescribe dameNombre de la clase Persona**

# Redefinición de métodos. Ejemplo

```
public class Persona{
    protected String nombre;
    protected int edad;

    public Persona(String nombre, int edad){
        this.nombre = nombre;
        this.edad = edad;
    }
    public String dameNombre(){
        return this.nombre;
    }
    public int dameEdad(){
        return this.edad;
    }
}
```

*Ejemplo2. PersonaCompañero*

# Redefinición de métodos. Ejemplo

```
public class Companero extends Persona {
    protected String telefono;
    protected String apodo;


    public Companero(String nombre, int edad, String telefono, String apodo){
        this.nombre = nombre; this.edad = edad;
        this.telefono = telefono;
        this.apodo = apodo;
    }
    public String darTelefono(){
        return this.telefono;
    }
    public void modificarTelefono(String telefono){
        this.telefono = telefono;
    }
    public String dameNombre(){
        return this.apodo; // En la superclase era this.nombre
    }
}
```

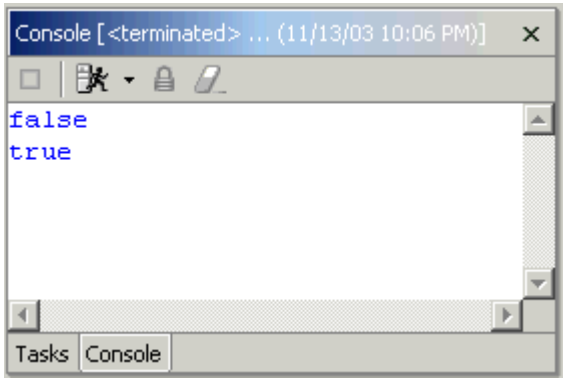
# super

- La palabra clave "***super***" es una ***facilidad del lenguaje*** para poder ejecutar constructores y métodos heredados que han sido redefinidos. **super** se utiliza para acceder desde un objeto a atributos y métodos (incluyendo constructores) del padre.
- Cuando el atributo o método al que accedemos no ha sido sobrescrito en la subclase, el uso de super es redundante, no hace falta ponerlo.
- Los constructores de las subclases incluyen una llamada a super() por defecto, si no hemos escrito un super o un this propio.

# super

Ejemplo de acceso a un atributo:

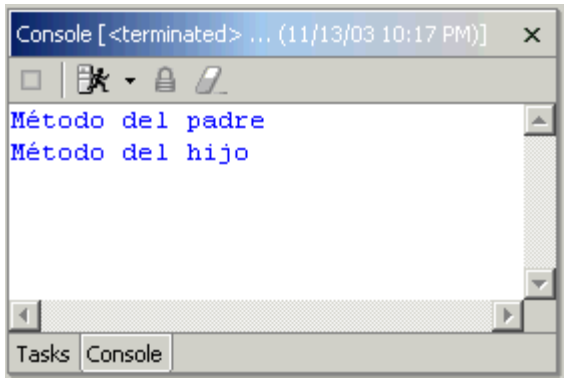
```
public class ClasePadre {
    public boolean atributo = true;
}
public class ClaseHija extends ClasePadre {
        public boolean atributo = false;
        public void imprimir()  {
            System.out.println(atributo);
           System.out.println(super.atributo);
   }
}
```



# super

Ejemplo de acceso a un método:

```
public class ClasePadre {
   public void imprimir()   {
     System.out.println("Método del padre");
   }
}
public class ClaseHija extends ClasePadre {
    public void imprimir()    {
          super.imprimir();
          System.out.println("Método del hijo");
    }
}
```



# Constructores

- Las subclases NO son responsables de inicializar las variables de instancia de las variables que hereda.
- Para pasarle la responsabilidad de inicializar esas variables a las superclases puede llamar al constructor de estas mediante la sentencia **super()**
- Hay una especie de “herencia” de constructores