# ***HERENCIA***

**Superclase o clase base:** la clase que se hereda (el padre)
**Subclase o clase derivada:** la clase que recibe la herencia(el hijo)
**¿Qué se hereda?:** la subclase hereda todos los atributos y métodos de su superclase.

**¿Cómo indicamos que una clase es subclase de otra clase?:** con la palabra reservada *extends*.

## Un ejemplo sencillo

Quiero crear objetos de una clase Persona que consta de nombre y apellidos y de una clase alumno que consta de nombre, apellidos y grupo. Sin utilizar el mecanismo de Herencia haría:

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

   public void imprimePersona() {System.out.println("Datos personales: " + nombre + ", "+ edad );
   }
}
class Alumno {
   String nombre;
   int edad ;
   char grupo;

   public void imprimePersona() {
         System.out.println("Datos personales: " + nombre + ", "+ edad );
   }
}
class Unidad5 {
   public static void main(String[] args) {
      Persona p1 = new Persona();
      p1.nombre="Elías";
      p1.edad=5;
      p1.imprimePersona();

      Alumno a1= new Alumno();
      a1.nombre="Román";
      a1.edad=3;
      a1.grupo='a';
      a1.imprimePersona();
   }
}
```

Observa que podemos considerar que un alumno no es más que un tipo o clase específica de persona que además tiene un atributo grupo. Si utilizamos el mecanismo de herencia podemos "sacar factor común" y aprovechar todos los atributos y métodos de *Persona* para A*lumno* evitando reescribirlos. Esto se consigue indicando que la clase Alumno hereda de la clase Persona con la palabra reservada *extends*. También se dice que Alumno es una subclase de Persona y que Persona es una superclase de alumno. En la siguiente versión el main() es exactamente el mismo, sólo que está reescrita la clase Alumno. Observa que la clase *alumno* puede utilizar los atributos *nombre* y *edad* por que los "hereda" de su clase padre.

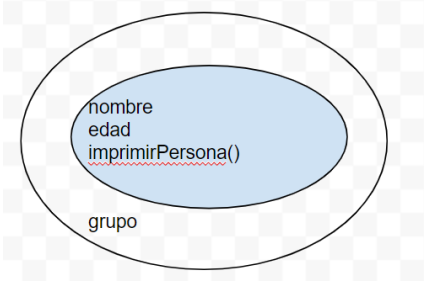
```
class Persona {
   String nombre;
   int edad;

   public void imprimirPersona() {System.out.println("Datos personales: " + nombre + ", "+ edad );
   }
}
class Alumno extends Persona{
   char grupo;
}
class Unidad5 {
   public static void main(String[] args) {
      Persona p1 = new Persona();
      p1.nombre="Elías";
      p1.edad=5;
      p1.imprimePersona();

      Alumno a1= new Alumno();
      a1.nombre="Román";
      a1.edad=3;
      a1.grupo='a';
      a1.imprimirPersona();
   }
}
```

**Volvemos a reflexionar. ¿Qué se hereda?**

Indicamos antes que: la subclase hereda todos los atributos y métodos de su superclase.

Lo anterior, lo matizamos ahora, desde el punto de vista de la creación de objetos indicando ahora que:

al crear un objeto subclase, también se crea un objeto superclase y es importante tener en cuenta que dicho objeto superclase está "dentro" del objeto subclase, siendo esta la razón por la que con una referencia subclase puede acceder a el objeto padre. En el ejemplo anterior, al crear un objeto alumno, en el interior de su memoria asignada está contenido el objeto padre. Esta observación no parece especialmente reveladora pero es más importante de lo que parece para entender varios conceptos asociados al mecanismo de herencia que veremos posteriormente.




**Herencia y static**

El dibujito anterior no dice nada sobre los elementos static ya que es un dibujito de la "zona de memoria de objetos" y recuerda que los miembros static se almacenan a nivel de clase en una zona de memoria especial para elementos static

No es correcto decir que se heredan los miembros static pues no se copian dentro del objeto, pero lo que sí ocurre es que desde un objeto subclase hay acceso a la zona

static de superclase. Por tanto, a efectos prácticos es como si se heredaran también los miembros static

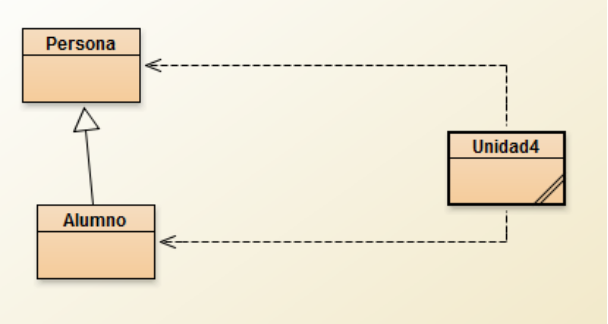
En el ejemplo observa que podemos hacer B.varstatic

```
class A{
   static int varstatic=15;
   int varnostatic;
}

class B extends A{
   int varb;
}

class Unidad5 {
   public static void main(String[] args) {
      System.out.println(B.varstatic);
   }
}
```

**Herencia y UML. La relación "Es un" ("Is a").**

Cuando 2 clases tienen una relación de herencia entre ellas se representa en UML con una flecha blanca. En el diagrama de abajo, que es una versión en bluej del ejemplo anterior, entre Persona y Alumno hay una relación de herencia.

Observa que una relación de composición el extremo era un diamante blanco o negro, pero en herencia el extremo es una flecha(triángulo) blanco.




A la relación de herencia también se le llama relación de Generalización/especialización y relación "Es un/-a", ya que por ejemplo en el ejemplo anterior:

- Una Persona es una generalización de un alumno
- Un alumno es una Especialización(un tipo concreto) de una Persona
- Un Alumno "Es una" Persona

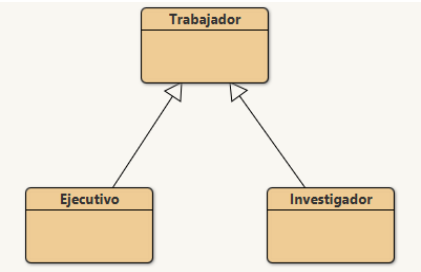
**Ejemplo:** una superclase puede tener muchas subclases.

me puede interesar dentro de todos los trabajadores, distinguir dos clases de trabajadores más específicas como *Ejecutivo* e *Investigador*.

```
class Trabajador{
   String dni;
   int sueldoBase;
```

```
    String departamento;
}

class Ejecutivo extends Trabajador{
    int primas;
}
class Investigador extends Trabajador{
    int horasExtra;
}

class Unidad5 {

    public static void main(String[] args) {
        Ejecutivo ejec1 = new Ejecutivo();
        //La clase Ejecutivo hereda los atributos de trabajador y por tanto los puede usar
        ejec1.dni="33311155x";
        ejec1.sueldoBase=6000;
        ejec1.departamento="comercial";
        //además Ejecutivo tiene un atributo no heredado
        ejec1.primas=2000;
        //ejec1.horasExtra=20; //no es posible

        Investigador inv1 = new Investigador();
        inv1.dni="11122233g";
        inv1.sueldoBase=1000;
        inv1.departamento="fabricación";
        inv1.horasExtra=15;
        //inv1.primas=2000 no es posible

        //un trabajador genércio, que no es ni ejecutivo ni investigador
        Trabajador t1=new Trabajador();
        t1.dni="44456712f";
        //etc...
    }
}
```

Ejemplo UML: Si el código de la jerarquía anterior lo pegas en Bluej (cada clase por separado) obtendrás el siguiente diagrama




***Una subclase no puede acceder a un miembro private de su superclase.***

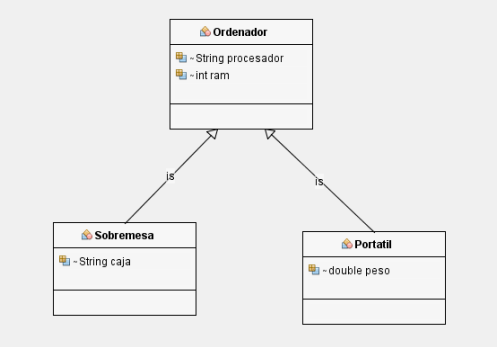
***Ejemplo:*** Si en el ejemplo anterior calificamos con private un atributo, por ejemplo:

*private int sueldoBase;*

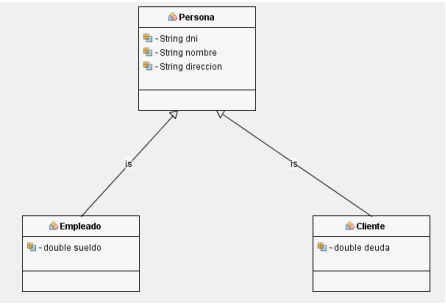
En main() aparecen errores (¡pruébalo!) por intentar acceder a atributos privados. Si queremos que el atributo sea privado habría que generar set y get para acceder a ellos ….Observa que ya que el objeto superclase se almacena dentro del subobjeto subclase, realmente el valor de sueldoBase está dentro del objeto subclase, pero a pesar de eso, por ser private en la superclase no está accesible.

En una superclase declarar los atributos private tiene el mismo sentido que en otra cualquiera, proteger a la clase de un mal uso por un tercero. Un programador A crea una superclase *ClaseA* otro programador B la extiende en una clase *ClaseB.* Si el programador A intuye un posible mal uso de su clase, puede defender su clase declarando private los atributos.

Al aparecer en escena el mecanismo de herencia aparecen nuevas consideraciones para el control de acceso a clases y miembros de *superclases* (palabra reservada protected, paquetes, ….) que veremos más adelante.
**Ejercicio U5_B3A_E1:** Un ordenador se caracteriza por su procesador(un String) y memoria RAM(un int). Los ordenadores pueden ser de sobremesa o portátiles. Para los ordenadores de sobremesa interesa su tipo de caja y para los portátiles su peso. Crea una jerarquía en java que recoja la situación anterior y desde un main() crea un ordenador de sobremesa con la CPU y RAM que tu elijas y con caja de tipo "micro-atx", lo mismo para un portátil de peso 1.5kg.




**Ejercicio U5_B3A_E2:**
Una persona se caracteriza por su dni, nombre y dirección. Queremos trabajar con dos tipos de personas: empleados y clientes. Los empleados son personas que además tienen un sueldo, los clientes son personas que además tienen una deuda. Crea una jerarquía en java para la situación anterior y desde un main() crea un empleado, un cliente y una persona genérica que no es ni empleado ni cliente. Imprime por pantalla los atributos de los objetos creados.




**Ejercicio U5_B3A_E3:**

Clases: Unidad5, Figura, Cuadrado, Circulo. Todas las clases en paquete por defecto.

Figura es superclase de las subclases Cuadrado y Circulo
la clase Figura:
atributo color(String)

la clase Cuadrado:
atributo lado(double)
la clase Circulo
atributo radio(double)

Todos los atributos son de acceso privado. **Utiliza sólo los set/get estrictamente necesarios.** Los constructores permiten crear Cuadrados y Circulos indicando su color e indicando la longitud del lado (caso cuadrado) o la longitud del radio (caso Circulo).
En main() de clase Unidad5

```
class Unidad5{
   public static void main(String[] args) {
      Cuadrado miCuadrado=new Cuadrado(2.5,"azul");
      System.out.println("Lado de miCuadrado: "+ miCuadrado.getLado());
      Circulo miCirculo=new Circulo(3.6,"blanco");
      System.out.println("Color de miCirculo: "+ miCirculo.getColor());

   }
}
```

**Ejercicio U5_B3A_E4:**

Escribe la siguiente jerarquía que usan en una clínica veterinaria:

- superclase Animal y subclases: Perro y Gato
- la clase Animal:
  - atributo edad
- la clase Perro:
  - atributo boolean puraRaza
- la clase Gato
  - atributo boolean razaEuropea
- La jerarquía anterior debe de pertenecer al paquete *animales*
- La clase Unidad5 pertenece al paquete por defecto
- Todos los atributos son de acceso privado.
- Si la edad que se indica en el constructor es mayor de 15, se pone al máximo posible que es 15(por razones de gestión clínica).
- **Utiliza los set/get estrictamente necesarios.**
- El main() debe ser obligatoriamente el siguiente

```
public static void main(String[] args) {
      Perro canKan=new Perro(16,true);
      Gato  cati=new Gato(13,false);

      System.out.println("Edad canKan: "+ canKan.getEdad() +"   Es pura raza canKan: "+
canKan.esPuraRaza());
      System.out.println("Edad cati: "+ cati.getEdad() +"   cati es raza europea: "+ cati.esRazaEuropea());
   }
```

Y genera:

```
Edad canKan: 15   Es pura raza canKan: true
Edad cati: 13   cati es raza europea: false
```