# TEMA VII: HERENCIA Y POLIMORFISMO

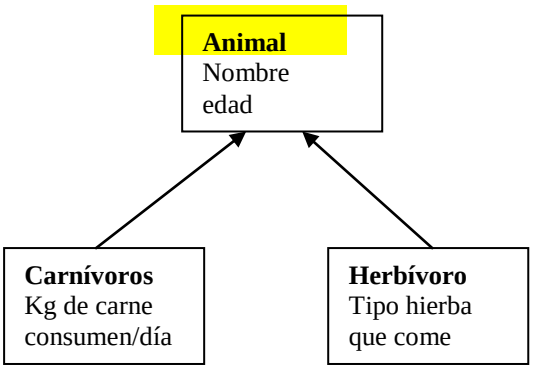
Ley de Alzheimer de la programación:
si lees un código que escribiste hace más de dos semanas
es como si lo vieras por primera vez

**CONCEPTO**

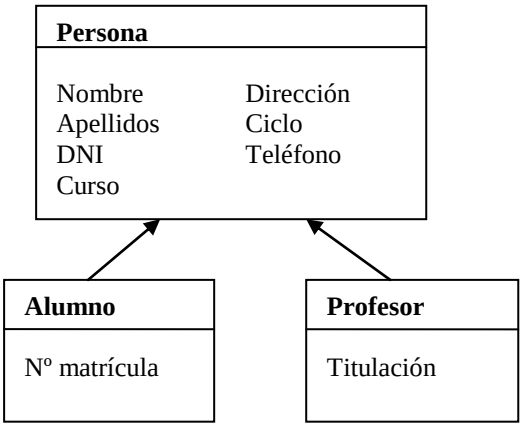
Se puede construir una clase a partir de otra mediante el mecanismo de la ***herencia***. La herencia es la base de la reutilización del código. Para indicar que una clase deriva de otra se utiliza la palabra ***extends.***
Cuando una clase deriva de otra, hereda todas sus variables y métodos. La clase principal se llama ***super-clase*** y la que hereda ***sub-clase***.

Ejemplos:

1. Animales una superclase (número, edad), carnívoro (Kg de carne que come al día), herbívoro (Tipo de hierba que come).

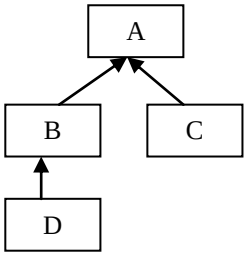



2. Aplicación que gestiona alumnos y profesores, de los primeros guarda nombre, apellidos, dni, teléfono, dirección, ciclo y curso, nº de matrícula. Para los segundos guarda nombre, apellidos, dni, teléfono, dirección, ciclo y curso de tutoría, titulación.




Las funciones y variables miembro pueden ser redefinidas (override) en la clase derivada, que puede también definir o añadir nuevas variables y métodos. En cierta forma la sub-clase (la clase derivada) "amplía" la clase padre con nuevas variables y métodos.

***Java*** permite múltiples niveles de herencia, pero no permite que una clase derive de varias (no es posible la herencia múltiple). Se pueden crear tantas clases derivadas de una misma clase como se quiera.




Un **constructor** de una subclase tiene que llamar en su primera línea a un constructor de su super-clase, si no invoca a ninguno el compilador asume que se llama al constructor sin parámetros de la super-clase, y si no existe da error.

El modificador **protected** hace visible un miembro a sus hijos, pero privado al exterior.

**POLIMORFISMO**

Cualquier objeto de una subclase puede ser asignado a un objeto de una superclase.

```
Jefe pp=new Jefe("Juan",200000,2);
```

```
Empleado a=pp;
```

Al declarar una variable de una clase base, dicha variable puede referenciar a cualquier objeto de cualquier clase derivada suya. Esto es un aspecto muy importante del **polimorfismo**

Al revés, mejor comprobamos antes la clase, porque si no nos da error en ejecución:

```
Empleado plantilla[]=new Empleado[10];
Jefe boss;
if (plantilla[i] instanceof Jefe)
{
        boss= (Jefe)plantilla[i];
}
```

Si invocamos a un método de un objeto la MVJ busca ese método en esa clase, si no lo encuentra lo busca en su padre y así sucesivamente. Esa habilidad para decidir qué método aplicar, es lo que se llama polimorfismo

**Ejemplo**:

1. Tenemos personas que pueden ser empleados normales o jefes. Para las personas guardo su nombre. Todos los empleados tienen un sueldo base. Los jefes cobrarán un 10% más que los empleados

```
public class Persona{
        prívate String nombre;

        public  String getNombre(){
                return nombre;
        }
        public void setNombre(String n){
                nombre=n;
        }
}

public class Empleado extends Persona{

        protected  double sueldoBase;

        public double getSueldo(){
                return sueldoBase;
        }
        public void setSueldo(double s){
                sueldoBase=s;
        }
}

public class Encargado extends empleado{
        private String puesto;
```

```
        public double getSueldo(){
                return sueldoBase*1,1;
        }
        public void setPuesto(String p){
                puesto=p;
        }
}
```

Una referencia a persona puede apuntar a la clase empleado, dicha variable podrá hacer llamadas a métodos de la clase persona pero no de la clase empleado porque daría error de compilación
Una referencia a empleado que apunta a un objeto encargado podrá llamar al método getSueldo y ejecutará el de la clase encargado, resuelve en tiempo de ejecución el tipo del objeto.

Tenemos por tanto dos tipos de vinculación (llamadas a métodos)

- Temprana: se realiza en tiempo de compilación. Para métodos normales o sobrecargados
- Tardía: se realiza en tiempo de ejecución. Para métodos redefinidos (sobreescritos)

Para solucionar los errores de compilación de una forma sencilla (más adelante veremos otros mecanismos) hacemos casting

```
persona p1;
( (empleado)p1).setSueldo(10000);
```

## SOBREESCRITURA DE MÉTODOS

Una de las propiedades fundamentales de la POO es la sobreescritura de métodos (overriding). Permite modificar el comportamiento de la clase padre. Para que un método sea sobrescrito:

- Tiene que tener el mismo nombre
- El retorno de la clase padre e hijo debe ser del mismo tipo
- Debe conservar la misma lista de argumentos que el método de la clase padre.

Si el hijo redefine un miembro, el del padre permanece oculto y sólo es accesible desde la clase del hijo poniendo por delante el identificador super.

El modificador final en una clase se usa para que no se permita derivar de ella.
El modificador final en un método indica que no puede ser redefinido. Casi no se usan.
No se pueden redefinir métodos estáticos.

## SOBRECARGA DE MÉTODOS

La sobrecarga es la implementación varias veces del mismo método con ligeras diferencias. Debe cumplirse:

Los métodos sobrecargados deben cambiar la lista de argumentos obligatoriamente
Los métodos sobrecargados pueden cambiar el tipo de retorno.

**CLASE ABSTRACTA**

Son clases pensadas para ser genéricas. No habrá objetos de esas clases. Por ejemplo tenemos un programa en el que manejamos coches y motos. Ambos son vehículos, ésta sería la clase padre y no tendría sentido crear vehículos, sino coches o motos. Esta superclase sólo servirá para definir los atributos y métodos comunes.

```
public abstract class vehiculo{
private int peso;

public void setPeso(int p){peso=p;}
public abstract int getVelocidadActual();
}
```

- Una clase abstracta tiene al menos un método abstracto.
- Un método es abstracto en una clase, si en ella sólo se declara, dejando su definición a las subclases.
- Las clases y los métodos abstractos llevan el modificador abstract.
- Una clase abstracta no se puede instanciar.
- Un método abstracto no puede ser estático

**Ejemplo:**

1. Programa que dibuja las vocales con un determinado carácter introducido por teclado. Hacer una clase letra y una subclase por cada vocal. Hacer un método abstracto en la clase letra que dibuje una vocal con un carácter determinado.

## CLASE OBJECT

Todas las clases de Java creadas por el programador tienen una super-clase. Cuando no se indica explícitamente una super-clase con la palabra extends, la clase deriva de java.lang.Object, que es la clase raíz de toda la jerarquía de clases de Java. Como consecuencia, todas las clases tienen algunos
métodos que han heredado de Object.

Se puede usar un objeto del tipo Object para almacenar objetos de cualquier tipo:

```
Object obj = new Empleado("Juan",150000,1);
Object obj= "Hola";
Object obj=5;  // error
```

### Métodos de la clase Object que pueden ser redefinidos

- public boolen equals( Object obj)

En Object indica si dos objetos apuntan a la misma zona de memoria.
Podemos redefinir este método en nuestra clase, por ejemplo en la clase empleado:

```
class Empleado{
  public boolean equals(Object obj){
        if (!(obj instanceof Empleado))
                        return false;
        Empleado b = (Empleado)obj;
        if  ( this.dni.equals(b.dni) )
                        return true;
        return false;
        }
    }
}
```

- public String toString()

En Object retorna un String con la dirección de memoria del objeto.
Cuando un objeto es concatenado con un String usando el operador "+", el compilador de Java llama automáticamente al método toString del objeto.
Por ejemplo, y siguiendo con el empleado:

```
class Empleado{
  public Empleado(String nom,double sal,int an){
                nombre=nom;
                salario=sal;
                antigüedad=an;
        }       ...
```

```
public String toString(){
        return "\nNombre:"+nombre+",sueldo:"+salario+",antigüedad:"+antigüedad;
        }
}
```

Si luego en el main hago:

```
Empleado e= new Empleado("Juan",200000,3);
System.out.println(e); // automáticamente llama a e.toString
```

**Otro método de la clase Object es el método:**

Class getClass()

Este método retorna un objeto de la clase Class que nos da información sobre la clase a la que pertenece el objeto.
La clase Class tiene varios métodos:

String getName() -→ Retorna el nombre de la clase
Class getSuperclass() → Retorna la clase del padre

Ejemplo:

```
Empleado e=new Empleado("Juan",200000,3);
System.out.println(e.getClass().getName() + "" + e.getNombre());
```

muestra por pantalla:
"Empleado Juan"