

Tema 7 Utilización avanzada de clases.

- 7.1.- La herencia.
- 7.2.- Restricciones de la herencia.
- 7.3.- Constructores.
- 7.3.- Atributos de clases.
- 7.4.- Clases y métodos abstractos y finales.
- 7.5.- Constructores y herencia
- 7.6.- Interfaces.
- 7.7.- Tipos de atributos y modificadores de ámbito.
- 7.8.- Librería de clases.

7.1.- Herencia.

La herencia permite que una clase herede las características y el comportamiento de otra clase (atributos y métodos), y a continuación modificar este comportamiento según lo necesite.

Para heredar una clase utilizamos la palabra reservada **extends** seguida del nombre de la clase de la que hereda.

Formato general de una clase

```
[ cualificadores ] class nombre_clase [ extends nombre_clase ] {  
    [ cualificadores ] tipo nomVar1;  
    [ cualificadores ] tipo nomVar2;  
    .....  
    .....  
    .....  
    [ cualificadores ] tipo nomMétodo1 ( [ lista_de_argumentos ] ) {  
        cuerpo  
    }  
    [ cualificadores ] tipo nomMétodo1 ( [ lista_de_argumentos ] ) {  
        cuerpo  
    }  
    [ cualificadores ] tipo nomMétodo1 ( [ lista_de_argumentos ] ) {  
        cuerpo  
    }  
}
```

7.2.- Restricciones de la herencia.

Superclase: Es la clase de la cual hereda otra clase, todos sus atributos y métodos.

Ejem. `class Nif extends Dni`

Declara una clase Nif que hereda todos los atributos y métodos de la clase Dni.

Se heredan:

- Los atributos y métodos con modo de acceso **public y protected**.
- Los atributos y métodos con modo de acceso package si la clase padre e hija pertenecen al mismo paquete.

No se heredan:

- Los atributos y métodos con modo de acceso private.
- Un atributo de la clase padre, si en la clase hija se define un atributo con el mismo nombre que en la clase padre.
- Un método si está sobrecargado.
- Los constructores de la superclase.

7.3.- Constructores.

Los constructores de la clase padre no se heredan pero si se pueden invocar desde la subclase. Por defecto exista o no constructor en la subclase, se hace una llamada al constructor por defecto de la superclase.

- *Si la primera instrucción de un constructor de una subclase no es una invocación a otro constructor con **this** o **super**, Java añade de forma invisible e implícita una llamada **super()** con la que invoca al constructor por defecto de la superclase. Luego continúa con las instrucciones de inicio de las variables de la subclase y luego sigue con la ejecución normal. **Si** en la superclase **no hay constructor por defecto** (sólo hay explícitos) **ocurrirá un error**.*
- *Si se invoca a constructores de superclases mediante **super(...)** en la primera instrucción, entonces se llama al constructor seleccionado de la superclase, luego inicia las propiedades de la subclase y luego sigue con el resto de sentencias del constructor.*

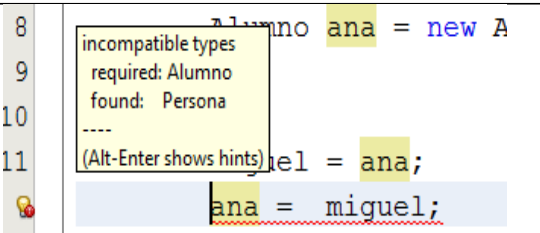
- Finalmente, si esa primera instrucción es una invocación a otro constructor de la clase con **this(...)**, entonces se llama al constructor seleccionado por medio de **this** y realiza sus instrucciones, y después continúa con las sentencias del constructor. La inicialización de variables la habrá realizado el constructor al que se llamó mediante **this(..)**.

El uso de **super** y **this** no puede ser simultáneo puesto que ambas tienen que ser **la primera instrucción**.

7.4.- Casting de clases.

El operador de casting o moldeado se puede utilizar con objetos de clases, igual que los utilizamos con los tipos básicos. Los objetos no se convierten de una clase a otra, igual que un tipo float no se convierte a double simplemente utilizando el operador (double), lo que podemos hacer es convertir referencias para indicar la subclase concreta a la que pertenece esa referencia.

La razón de los casting está en que es posible asignar referencias de una superclase a una subclase, pero no al revés.

<pre>public class PruebaInstituto { public static void main(String arg[]) { Persona miguel = new Persona("Miguel", "24781195"); Alumno ana = new Alumno("Ana", "22435688", 1); miguel = ana; ana = (Alumno) miguel; System.out.println(miguel); System.out.println(ana); } }</pre>	
--	--

Aunque el objeto miguel (Persona) reciba la referencia de ana (Alumno) no podrá acceder a los atributos y métodos de la clase Alumno, pero es necesario que contenga una referencia a Alumno para poder hacer el casting a alumno.

Un error de tipos incompatibles provoca una excepción del tipo **ClassCastException**.

InstanceOf nos permite comprobar si un objeto pertenece a una determinada clase.

<pre> public class PruebaInstituto { public static void main(String arg[]) { Persona miguel = new Persona("Miguel", "24781195"); Alumno ana = new Alumno("Ana", "22435688", 1); Empleado e = new Empleado(); Docente luis = new Docente("Luis", "24881270", 1500, "Fisica"); e = luis; System.out.println(luis); if (e instanceof Docente) { System.out.println("Es Profe"); } if (e instanceof Servicios) { System.out.println("Es P.A.S"); } miguel = ana; ana = (Alumno) miguel; System.out.println(miguel); if (miguel instanceof Persona) { System.out.println("Es Persona"); } System.out.println(ana); if (ana instanceof Alumno) { System.out.println("Es un alumno"); } System.out.println(); } } </pre>	<p>Visualiza:</p> <p>Nombre: Luis Dni: 24881270 1500.0 Fisica Es Profe</p> <p>Alumno : Nombre: Ana Dni: 22435688 Curso: 1 Es Persona</p> <p>Alumno : Nombre: Ana Dni: 22435688 Curso: 1 Es un alumno</p>
---	--

7.5.-Atributos de clase: tipos

Son palabras reservadas que se anteponen a la declaración de la clase. Los modificadores posibles son:

- **public**
- **abstract**
- **final**

public . Cuando se crean varias clases que se agrupan formando un paquete (package), sólo las declaradas public pueden ser accedidas desde otro paquete. Toda clase public, debe ser declarada en un fichero fuente con el nombre de esa clase pública: NombreClase.java. En un fichero fuente puede haber más de una clase, pero sólo una con el modificador public.

abstract. Las clases abstract no pueden ser instanciadas. Sirven para declarar subclases que deben redefinir los métodos declarados abstract.

Los métodos de una clase abstract pueden no ser abstract, en este último caso, tampoco se podrán declarar objetos de una clase declarada como abstract

Cuando existe algún método abstract, la clase debe ser declarada abstract , y la subclase que lo hereda debe implementarlo ,en caso contrario el compilador dará un error.

final .Una clase declarada final impide que pueda ser superclase de otras clases., es decir ,ninguna clase puede heredar de una clase final. A diferencia del abstract, pueden existir en la clase métodos final, sin que la clase que los contiene sea final.

Una clase puede ser a la vez:

public abstract
public final

Ejemplo: abstract

```
abstract class Animal {  
    String nombre;  
    int patas;  
    public Animal (String n, int p ) {  
        nombre= n;  
        patas = p;  
    }  
    abstract void habla ();  
}
```

```

class Perro extends Animal {                                // Perro es una subclase de la clase abstract Animal
String raza ;

public Perro (String n, int p, String r ) {
    Super (n, p );
    raza = r; }

public void habla () {                                     // Necesitamos redefinirlo para instanciar objetos de la clase
    Perro.

    System.out.println ("Me llamo " +nombre+ " :GUAU " );
    System.out.println ("Mi raza es " +raza );

} // cierra el método

} // cierra la clase

```

```

class Gallo extends Animal {                                // Gallo es una subclase de la clase abstract Animal

public Gallo (String n, int p) {
    super (n, p );
}

.....
public void habla () {                                     // Redefinimos porque tiene el modificador abstract en Animal
    System.out.println("Soy un gallo, me llamo "+nombre );
    System.out.println ("Kikirikiuuuuu " );
}

} // cierra la clase

```

```

class PruebaAbstracta {

public static void main ( String argm[ ] ) {

    Perro toby =new Perro ( "Toby", 4 ,"San Bernardo");      //creamos el objeto toby
    Gallo kiko = new Gallo ( "Kiko ", 2 );                    // creamos el objeto kiko
    kiko.habla();
    toby.habla();

}

} // cierra la clase

```

7.6.- Tipos de atributos y modificadores de ámbito.

- **static**
- **final**
- **transient**
- **volatile**

Atributo static - Define un atributo de clase

Ejemplo -

<pre>Abstract class Persona { static int numPersonas = 0 ; // atributo de clase String nombre ; // atributo de objeto public Persona (String n) { // Constructor nombre =n; numPersonas++; } public void muestra () { System.out.print ("Soy "+nombre); System.out.print("pero hay "+(numPersonas-1) + "personas más"); } }</pre>	<pre>class Grupo { public static void main(String args[]) { Persona p1, p2, p3; // se crean tres instancias del atributo nombre y sólo una del atributo persona p1 = new Persona ("Pedro"); p2 =new Persona ("Juan"); p3 =new Persona ("Susana"); p2.muestra() ; p1.muestra(); } }</pre>
---	--

Atributo final -La palabra final calificando a un atributo, sirve para declarar constantes. Si además es static, se puede acceder a dicha constante, anteponiendo el nombre de la clase, sin necesidad de crear un objeto de la misma.

El valor de un atributo final debe ser asignado en la declaración.

Ejemplo

<pre>class Circulo { final double PI=3.14159265; int radio; Circulo (int n) { Radio= n; } public double area () { return PI*radio*radio ; } }</pre>	<pre>class Atrib2I { public static void main (String args []) { Circulo c = new Circulo(15) ; System.out.println(c.area()) ; } }</pre>
--	--

Atributo transient .- Los atributos de un objeto , por defecto se consideran persistentes. Esto significa que a la hora de almacenarlos, por ejemplo en un fichero, los valores de los atributos deben almacenarse también. Aquellos atributos que no forman parte del estado persistente del objeto, porque almacenan estados transitorios o puntuales del objeto se declaran **transient**.

ejemplo - Atributos transient -

```
class Atrib3 {  
    int var1, var2;  
    transient int numVecesModificado=0;  
}  
void modifica(int v1, int v2) {  
    var1=v1;  
    var2=v2;  
    numVecesModificado++;  
}  
} // cierra clase
```

Atributo volatile .- Si una clase contiene atributos de objeto que son modificados asíncronamente por distintos threads que se ejecutan concurrentemente, se pueden utilizar atributos volatile, para indicarle a la MVJ este hecho, y cargar el atributo desde memoria antes de utilizarlo, y volver a almacenarlo en memoria después, para que cada thread pueda “verlo” en un estado coherente.

Modificadores del ámbito de los atributos :

- **private**
- **public**
- **protected**
- **El ámbito por efecto**

private - El modificador de ámbito private es el más restrictivo de todos. Todo atributo private es visible únicamente dentro de la clase en la que se declara. No existe ninguna forma de acceder al mismo si no es a través de algún método (no private) que devuelva o modifique su valor.

Una buena metodología de diseño de clases es declarar los atributos private siempre que sea posible, ya que esto evita que algún objeto pueda modificar su valor.

public - Es el menos restrictivo de todos. Un atributo public será visible en cualquier clase que desee acceder a él, simplemente anteponiendo el nombre de la clase.

Las aplicaciones bien diseñadas minimizan el uso de los atributos public y maximizan el uso de atributos private. La forma apropiada de acceder y modificar atributos de objetos es a través de métodos que accedan a los mismos.

Los atributos nombre y dirección podrán ser modificados por cualquier clase. ejem.

```
emple1.nombre="Pedro López";
```


Mientras que el sueldo no puede ser modificado directamente por otra clase que no sea Empleado.

Para que la clase estuviera bien diseñada, se deberían haber declarado `private` los tres atributos y declarar métodos para modificar los atributos. De estos métodos, el que modifica el atributo sueldo podría declararse de tipo `private` para que no pudiera ser utilizado por otra clase distinta de Empleado.

protected .Los atributos `protected` pueden ser accedidos por las clases y subclases del mismo paquete (package) pero no pueden ser accedidos por subclases de otro paquete.

<pre>package PProtegido; public class Protegida { protected int valorProtegido; public Protegida(int v) { valorProtegido=v; } }</pre>	<pre>package PProtegido; public class PruebaProtegida { public static void main(String args[]) { Protegida p1= new Protegida(0); p1.valorProtegido = 4; System.out.println (p1.valorProtegido); } }</pre>
--	--

PruebaProtegida pertenece al mismo paquete que la clase Protegida, y puede acceder a su atributo `protected`, `valorProtegido`.

<pre>package OtroPaquete; import PProtegido.*; public class Protegida2 extends Protegida { public Protegida2(int v) { super(v); } public void modifica(int v) { valorProtegido=v; } }</pre>	<pre>package OtroPaquete; import PProtegido.*; public class PruebaProtegida2 { public static void main(String args[]) { Protegida p1= new Protegida(0); p1.valorProtegido = 4; //Error, valor protegido System.out.println(p1.valorProtegido);//</pre>
---	---

E
r
r
o
r

v
a
l
o
r

Ejemplo : public
final class Empleado {
 public String nombre;
 public String dirección;
 private int sueldo;
}

protegido

```
    }  
    }  
  
package OtroPaquete;  
  
public class EjecutaProtegida2 {  
  
    public static void main(String args[]) {  
        Protegida2 p1= new Protegida4(0);  
        p1.valorProtegido = 4;  
        System.out.println(p1.valorProtegido);  
    }  
}  
  
package OtroPaquete;  
public class EjecutaProtegida2_2 {  
  
    public static void main(String args[]) {  
        Protegida2 p1= new Protegida4(0);  
        p1.modifica(4);  
    }  
}
```

Pueden ser accedidas las variables **protected heredadas** de la primera clase.

En este caso, se importa el paquete PProtegido para poder acceder a la clase Protegida, pero el paquete en el que se declara PruebaProtegida2 es distinto al que contiene Protegida, por lo que no se puede acceder a sus atributos protected.

Esta clase sí puede modificar el atributo protected pero únicamente a través del método de la clase Protegida2 denominado modifica.

El ámbito por defecto. Los atributos que no llevan ningún modificador de ámbito pueden ser accedidos desde las clases del mismo paquete, pero no desde otros paquetes.

Ejem. Añadir atributos de clase, al ejemplo de Alumnos y Profesores.

```
abstract class Persona {  
    final String nombre ;  
    String dni ;  
  
    public Persona(String n, String nif) {  
        nombre = n;  
        dni= nif;  
    }  
    abstract void Categoria ();  
  
}  
  
class Alumno extends Persona {  
    int nmatri;  
    String curso;  
    static int totalAlum ;  
  
    public Alumno (String n, String nif, String cur, int matri ) {
```

```
super(n, nif);  
nmatri=matri;  
curso= cur;  
totalAlum ++; }
```

```
public void Categoria() {  
    System.out.println ("Soy "+nombre+" de "+curso+" somos: "+ totalAlum );  
    System.out.println(" Mi DNI es " +dni );    }  
}
```

```
class Profe extends Persona {  
String espec;  
public Profe (String n, String nif, String espe) {  
    super (n,nif);  
    espec= espe;  
}  
public void Categoria () {  
System.out.println ("Soy profesor de" +espec+" me llamo " +nombre);  
    }  
}
```

```
class Instituto {  
  
public static void main ( String argm [ ] ) {  
    Alumno al1= new Alumno ("Lucas", "34869924", "1º ASI-B ", 835 );  
    Alumno al2 =new Alumno ("Dani", "35545444", "1º ASI-B ",956);  
    Profe prof1 = new Profe("Javi","32458869","Informática") ;  
  
    prof1.Categoria();  
    al1.Categoria();  
    System.out.println(al2.nombre); }  
}
```

7.6.- Interface.

Es un conjunto de constantes y métodos, pero de éstos últimos sólo el formato, no su implementación.

Cuando una clase declara una lista de interfaces, asume que se van a redefinir todos los métodos definidos en la interface.

```
class NombreClase implements Interface1, Interface2, .... ,InterfaceN.
```

```
class Nif extends Dni implements OperacionesAritmeticas, OperacionesLogica.
```

Sintaxis

modificador **class** NombreClase [**extends** NombreSuperclase] [**implements** listaDeInterfaces]

Si no se especifica ningún modificador , la clase será visible en todas las declaradas en el mismo paquete. Si no se especifica ningún paquete, se considera que la clase pertenece al paquete por defecto al que pertenecen todas las clases que no declaran explícitamente el paquete al que pertenecen.

En Java no está permitida la herencia múltiple (más de una superclase). Una aproximación a este concepto es la utilización de una sola superclase y una o varias interfaces (conjunto de constantes y métodos abstractos).

Cuando una clase declara una lista de interfaces mediante la cláusula implements hereda todas las constantes definidas en el interface y se compromete a redefinir todos los métodos del interface .Mediante el uso de interfaces se consigue que distintas clases implementen métodos con el mismo nombre ,de esta forma se comprometen a implementarlas. Los interfaces permiten al programador, definir un conjunto de funcionalidades sin tener “ni idea” de cómo serán implementadas

Por ejemplo, el interface java.lang Runnable especifica que cualquier clase que implemente este interface deberá redefinir el método run(). Así, la MVJ puede realizar llamadas a este método (sabe que existe) aunque no sepa qué hace realmente.

<pre>interface Mercancia{ public double damePrecio(); public String dameDescripcion(); }</pre>	<pre>interface MercanciaViva extends Mercancia{ public boolean necesitaComida(); public boolean necesitaRiego(); }</pre>
---	---

```

public class PlantaJardin2 implements
MercanciaViva{
    double precio;
    boolean estaRegada;
    String descripcion;

    PlantaJardin2(double nuevoPrecio,String
nuevaD){
        precio = nuevoPrecio;
        descripcion = nuevaD;
    }
    public boolean necesitaComida(){
    return false; }
    public boolean necesitaRiego(){
    return true; }
    public boolean tieneRiego(){
    return estaRegada; }
    public double damePrecio(){
    return precio; }
    public String dameDescripcion(){
return descripcion; }

// cierra clase

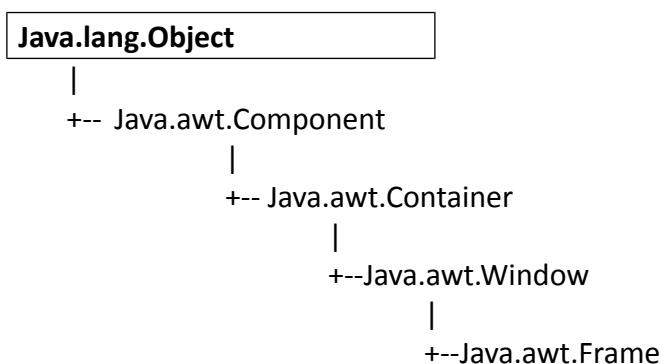
```

7.6.-LA LIBRERÍA DE CLASES JAVA

Un aspecto importante del lenguaje Java es la biblioteca de clases predefinidas, ya que incluye un grupo muy amplio de tipos y operaciones relacionados con problemas comunes de la programación actual. Así, la biblioteca de clases de Java permite tratar aspectos tan distintos como el tratamiento numérico, la gestión fiable de comunicaciones remotas y la realización de interfaces de usuario gráficas, entre otros muchos.

La librería de clases del lenguaje se encuentra organizada de forma jerárquica. Existe una clase inicial, primera en el árbol de la jerarquía de clases, denominada **Object**. Todas las clases Java son, de una forma u otra, descendientes de la clase Object. Igualmente todas las clases de Java heredan, a veces sobrescribiéndolos, los métodos de dicha superclase.

La herencia entre clases predefinidas es muy habitual en Java, pudiéndose representar la jerarquía de clases del lenguaje mediante un árbol de bastante profundidad. Por ejemplo, la jerarquía correspondiente a la clase Frame, tal y como aparece en la ayuda “on-line” del lenguaje:



Cada una de las clases del ejemplo: Object, Component, Container, Window, Frame, heredan en sus definiciones los atributos y métodos de las clases precedentes, sobrescribiéndolos cuando así lo necesiten, de forma que las funcionalidades de una clase quedan definidas por las de las clases que extienden, junto con las aportadas por ella misma.

Los nombres de las clases del ejemplo anterior vienen antepuestos por las etiquetas java.lang y java.awt. dichas etiquetas no son nombres de clases sino que son un mecanismo de agregación de clases, propio de Java, denominado *paquete* (Package).

Uso de Packages.

En Java los paquetes (Packages), se utilizan para poder agrupar un conjunto de clases que tienen funcionalidades comunes. Un Package es, por lo tanto, un mecanismo del lenguaje para facilitar la organización de un conjunto de clases. En Java existe un grupo de paquetes predefinidos en

los que se encuentran englobados todas las clases de la librería del lenguaje. Ejemplos de paquetes son los ya mencionados en el ejemplo anterior: java.lang y java.awt.

Resumen

Declaración de una clase : Modificadores class NombreClase **extends**
nombre_clase_padre

Modificadores de clase

public	Fichero fuente con el nombre de la clase.
abstract	No se podrán crear objetos de esta clase.
final	No puede ser superclase de otras. No se puede heredar.

Modificadores de atributos.

static	-Atributo de clase
final	- Constante
transient	- No persistentes
volatile	- Ser utilizados por varios procesos concurrentemente.

Modificadores de ámbito de los atributos.- Dificultan / Protegen el acceso a los atributos:

private
public
protected
El ámbito por efecto