



## Programación Orientada a Objetos (POO III)

### Objetos o archivos de clases

Las clases u objetos contienen tres partes importantes:

- **Constructores (o inicializadores).**
- **Funciones Públicas.**
- **Funciones Privadas.**

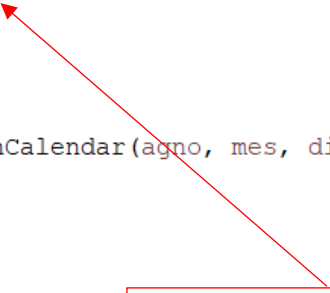
### Constructores

Los constructores inicializan nuestra clase (nuestro objeto), le dan un estado inicial estable listo para su uso. Siempre que declaramos una variable de tipo int esta tomaba el valor 0 como estado inicial para después cambiarlo a nuestro gusto, los constructores hacen lo mismo, pero para las clases.

“Constructor” es solo un nombre más elegante para referirnos a una función, el constructor es en cierta forma también una función, pero es una muy especial porque no tiene tipo y no devuelve valores, solo es una sección de código que se ejecutará antes que las demás.

Puesto que el constructor se ejecutará antes que ninguna otra parte, es un área perfecta para inicializar muchas variables a la vez por eso lo llaman constructor.

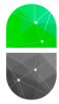
```
class Empleados{  
  
    public Empleados(String nom, double sue, int agno, int mes, int dia){  
        nombre=nom;  
        sueldo=sue;  
        calendario=new GregorianCalendar(agno, mes, dia);  
        Id=IdSiguiente;  
        IdSiguiente++;  
    }  
}
```



Ejemplo de constructor

### Funciones Públicas

Las funciones públicas son funciones que podemos usar libremente en el momento que lo necesitemos. Todas las funciones que hemos manejado hasta ahora han sido accesibles. ¿Bajo qué condiciones no lo son que debemos definir las como públicas?: cuando queremos llamarlas desde fuera de la clase. Crear objetos implica llamar a las funciones desde fuera de la clase.



Las funciones que son accesibles desde fuera de la clase, son funciones públicas. Tal llamada desde el exterior es el equivalente de enviar una orden a un objeto. Un método público sirve para modificar el estado interno del objeto.

```
public String getDatosEmpleado() {  
    return "El empleado " + nombre + " tiene el Id " + Id;  
}  
  
public double getSuelo() {  
    return sueldo;  
}
```

Ejemplo de funciones públicas

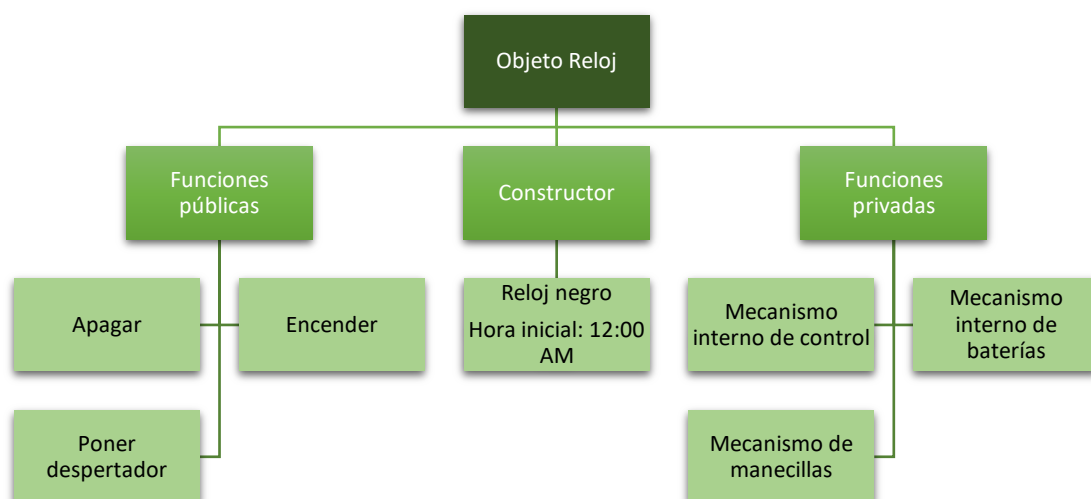
### Funciones Privadas

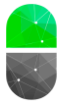
Las funciones privadas son las que no son accesibles desde el exterior de la clase, estas funciones solo se necesitan dentro de la misma clase.

Las funciones privadas son como las herramientas que posee internamente la clase para operar y por tal motivo no es importante que se tenga acceso a ellas desde el exterior de la clase.

Al igual que las funciones también existen variables públicas y privadas, estas tienen la misma utilidad que los métodos al ser modificadas. En ocasiones puede ser muy útil declarar una variable como pública para posteriormente poder modificarla y en otras ocasiones será más útil declararla como privada. Todo depende de la situación y del propósito de nuestra clase.

- **Ejemplo en pseudocódigo sobre cómo crear un objeto reloj que demuestre cuales serían sus funciones públicas y sus funciones privadas:**





Al utilizar uno de estos relojes nos importa su operación y no su mecanismo interno. Por eso existen funciones públicas o privadas. Las funciones públicas son la interfaz que usaremos. El constructor inicializa el objeto en un estado inicial estable para su operación.

- **Distintos modificadores y las restricciones que imponen a sus variables o métodos:**

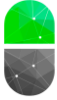
Acceso/modificador	Sin modificador	Public	Private	Protected
Misma clase	Sí	Sí	Sí	Sí
Subclase	Sí	Sí	No	Sí
Clase ajena	Sí	Sí	No	Sí
Clase ajena de otro paquete	Sí	Sí	No	No

### Sintaxis en Java para crear clases

La forma más sencilla de definir una clase en Java es:

```
Class NombreClase {  
    Constructor1  
    Constructor2 // opcional (sobrecarga de constructores cuando hay más de uno)  
    .....  
    Método1  
    Método2  
    .....  
    Campo1  
    Campo2  
    ..... }  
}
```

A continuación, se puede ver un ejemplo de definición de clase “Empleados” con constructor, métodos **public** de acceso y campos de clase **private**:



```
class Empleados{
```

```
    public Empleados(String nom, double sue, int agno, int mes, int dia){  
        nombre=nom;  
        sueldo=sue;  
        calendario=new GregorianCalendar(agno, mes, dia);  
        Id=IdSiguiente;  
        IdSiguiente++;  
    }
```

Constructor

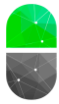
```
    public String getDatosEmpleado(){  
        return "El empleado " + nombre + " tiene el Id " + Id;  
    }  
    public double getSueldo(){  
        return sueldo;  
    }  
    public GregorianCalendar getFechaAlta(){  
        return calendario;  
    }  
    public void setSubeSueldo(double porcentaje){  
        double aumento=sueldo*porcentaje/100;  
        sueldo+=aumento;  
    }  
  
    public static String getIdSiguiente(){  
        return "El Id del siguiente Empleado será: " + IdSiguiente;  
    }  
}
```

Métodos de acceso  
(todos public)

```
    private final String nombre; // Esto es una constante. No se podrá modificar ur  
    private double sueldo;  
    private GregorianCalendar calendario;  
    private int Id;  
    private static int IdSiguiente=1; // Esto es un campo de clase. Pertenece a la
```

Campos de clase  
(todos private)

```
}
```



En esta clase se definen cinco variables **private**. Al ser variables que no se van a necesitar fuera de esta clase, se declaran como **private** y así aseguramos que no se modifiquen accidentalmente desde el exterior.

- Se declara un método constructor que recibe cinco variables por parámetros.
- Se declaran cinco métodos. Uno de ellos (**setSubeSueldo**) no devuelve ningún resultado y se declara como **void**.

A continuación, implementamos una clase desde la cual crearemos objetos de tipo empleado.

```
package es.pildorasinformaticas.poo;

import java.util.GregorianCalendar;

public class UsoEmpleados {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Empleados Antonio=new Empleados("Antonio", 2300.5, 2005, 7,15);

        Empleados Ana=new Empleados("Ana", 2900, 2008, 8, 9);

        Ana.setSubeSueldo(15);

        System.out.println(Ana.getSueldo());
    }
}
```

En esta clase se crean dos objetos de tipo Empleados, se llama al constructor de la clase Empleados y se pasan los parámetros correspondientes en la llamada.

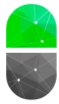
Uno de los objetos llama a los métodos **setSubeSueldo** y **getSueldo** de la clase Empleados.

### Uso de múltiples ficheros de código fuente

El ejemplo anterior consta de un único fichero. Pero de igual forma se podría haber creado el programa con dos ficheros independientes: uno con la clase **Empleados** y el otro con la clase **UsoEmpleados**.

En el caso de crear el programa en dos ficheros, para que todo funcione correctamente se deberán guardar los dos ficheros con extensión .java en el mismo directorio. A continuación, se deberán compilar ambos (esto lo hacen los IDEs de forma automática. En nuestro caso Eclipse) y por último ejecutar el fichero o clase que lleve el método main (debes situar el cursor en la clase principal, la que lleva el método main antes de ejecutar).

En el caso de optar por la opción de programar todo en un único fichero, lo único a tener en cuenta es que solo una de las clases deberá declararse como public. Esta clase será la que lleve el método main y la hora de compilar y ejecutar el programa lo haremos siempre a través de la clase public.



## Primeros pasos en el uso de constructores

Examinemos el constructor de la clase Empleados:

```
public Empleados(String nom, double sue, int agno, int mes, int dia){  
    nombre=nom;  
    sueldo=sue;  
    calendario=new GregorianCalendar(agno, mes, dia);  
    Id=IdSiguiente;  
    IdSiguiente++;  
}
```

El nombre del constructor ha de ser el mismo que la clase. Este constructor se ejecuta cuando se crean objetos de la clase Empleados y les da a los campos de clase su valor inicial.

El constructor solo se puede llamar a través del operador **new**.

- **Resumiendo:**
  - El constructor tiene el mismo nombre que la clase.
  - Una clase puede tener más de un constructor.
  - Un constructor puede tener cero, uno o más parámetros.
  - Un constructor carece de valor proporcionado.
  - Los constructores siempre se invocan mediante el operador new.

## Beneficios de la encapsulación

- Se puede modificar la implementación interna sin afectar a ningún código, salvo el de los métodos de clase.
- Los métodos de modificación pueden efectuar una comprobación de errores, mientras que un código que se limite a dar valores a un campo puede no tomarse la molestia.

## Métodos privados

Cuando se implementa una clase, hacemos privados todos los campos de datos porque los datos públicos son peligrosos.

A veces se necesita construir métodos auxiliares dentro de una clase, por ejemplo, para realizar cálculos. Estos métodos no deberían formar parte de la interfaz pública. Lo mejor en estos casos es declararlos como **private**.

Basta cambiar la palabra reservada **public** por **private**.



Lo importante es que mientras el método sea `private`, los diseñadores de la clase pueden tener la seguridad de que no se utilizará salvo en las operaciones de la propia clase.

Si un método es público, a la hora de eliminarlo hay que extremar las precauciones porque puede que haya otro código basado en él. Esta revisión nos llevaría además mucho tiempo.

### Campos finales de un ejemplar (de una clase)

Los campos de ejemplar (de clase) se pueden definir como **final**. Estos campos deben recibir valor inicial cuando se construye el objeto. Debemos garantizar que el valor del campo quede fijado al acabar todos los constructores.

Este campo no podrá ser modificado.

**Ejemplo:**

```
private final String nombre;
```

Una vez que el constructor le da valor a la variable `nombre`, este no se volverá a modificar.

### Campos y métodos estáticos

Si se define un campo como **static**, solo hay uno de estos campos por clase. Por el contrario, cada objeto posee su propia copia de todos los campos de ejemplar.

**Ejemplo:**

```
private final String nombre;  //  
  
private double sueldo;  
  
private GregorianCalendar calendar  
  
private int Id;  
  
private static int IdSiguiente=1;
```

Ahora todo objeto de tipo `Empleados` posee su propio campo `id`, pero solo hay un campo `IdSiguiente` que se comparte entre todos los objetos de la clase `Empleados`.

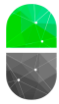
Vamos a construir un método sencillo:

**Ejemplo:**

```
Public void setId()
```

```
{  
  
    Id = idsig;  
  
    Idsig++; }  

```



Supongamos que se establece el número de identificación para María:

**María.setId();**

El campo `idsig` queda fijado para María, después se incrementa en 1. Si volviéramos a llamar al método `setId()` con otro objeto de la clase `Empleado`, el valor `id` que se asignaría a ese objeto sería 2 y posteriormente se volvería a incrementar.

### Métodos estáticos

Los métodos estáticos no operan sobre objetos. Por ejemplo, el método `pow` de la clase `Math` es un método estático.

**Ejemplo:**

**`Math.pow( x, a );`**

Calcula la potencia de  $x^a$ . No se utiliza ningún objeto `Math` para llevar a cabo esta tarea.

Como los métodos estáticos no operan sobre objetos, no se puede acceder a campos de ejemplar desde los métodos `static`. Pero los métodos `static` pueden acceder a los campos `static` de su clase.

**Ejemplo:**

```
public static int getIdSiguiente()  
{  
    return idSiguiente;  
}  
  
static int idSiguiente;
```

Para llamar a este método se proporciona el nombre de la clase:

**`int n=Empleados.getIdSiguiente(); // donde Empleados es el nombre de la clase.`**

Si se hubiera omitido la palabra `static` hubiéramos necesitado un objeto de la clase `Empleados` para llamar al método `getIdSiguiente`, pero al ser estático no precisamos tal objeto y lo hacemos directamente utilizando el nombre de la clase.

Los métodos `static` se utilizan en dos situaciones:

- Cuando un método no necesita acceder al estado del objeto porque todos los parámetros necesarios se proporcionan como parámetros explícitos (ejemplo: `Math.pow`).
- Cuando un método solo necesita acceder a campos `static` de la clase.





## El método main

Los métodos estáticos se pueden llamar sin tener objeto alguno, como por ejemplo los métodos de la clase Math ( `Math.pow` ).

Por la misma razón, el método main es un método estático.

El método main no actúa sobre objetos y debe ser el primer método que se ejecuta en un programa Java. Cuando arranca un programa no hay objeto alguno. Se ejecuta el método main y este construye objetos si el programa lo necesita.

## Construcción de objetos

Java ofrece toda una gama de mecanismos para escribir constructores.

- **SOBRECARGA**

La sobrecarga se produce cuando varios métodos poseen el mismo nombre pero distintos parámetros. El compilador tiene que averiguar cuál de los métodos es llamado y esto lo hace determinando el número correcto de parámetros en la llamada.

Se produce error de compilación si no se encuentra el método con el número de parámetros adecuado o si se encuentra más de un método con el mismo número de parámetros.

- **CONSTRUCTORES PREDETERMINADOS**

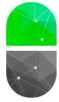
Un constructor predeterminado es un constructor sin parámetros.

Si se escribe una clase que carezca por completo de constructores, se nos proporciona un constructor predeterminado. Este constructor da a todas las variables de clase su valor predeterminado.

Si una clase proporciona al menos un constructor, pero no proporciona un constructor por defecto, no se permite construir objetos sin parámetros de construcción.

- **NOMBRES DE LOS PARÁMETROS**

Hay veces en que el nombre de los parámetros de un método, son los mismos que el nombre de las variables de clase. Por ejemplo, si se llama sueldo a un parámetro, entonces sueldo se refiere al parámetro, no a la variable de clase. Pero sigue siendo posible acceder a las variables de clase en la forma **this.sueldo**. Con la palabra reservada **this** delante del nombre de la variable hacemos referencia a la variable de clase y no al parámetro.

**Ejemplo:**

```
public Empleados(String nom, double sueldo, int agno, int mes, int dia){  
    nombre=nom;  
    this.sueldo=sueldo;  
    calendario=new GregorianCalendar(agno, mes, dia);  
    Id=IdSiguiente;  
    IdSiguiente++;  
}
```

- **LLAMADA A OTRO CONSTRUCTOR**

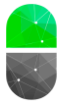
La palabra reservada **this** tiene también otra utilidad.

Si la primera sentencia de un constructor tiene la palabra **this**, entonces el constructor llama a otro constructor de la misma clase.

**Ejemplo:**

```
class Empleados{  
  
    public Empleados(String nom, double sue, int agno, int mes, int dia){  
        nombre=nom;  
        sueldo=sue;  
        calendario=new GregorianCalendar(agno, mes, dia);  
        Id=IdSiguiente;  
        IdSiguiente++;  
    }  
  
    public Empleados (double sue){  
        this("Pedro", sue, 2005, 5, 8);  
    }  
}
```

En este ejemplo hay sobrecarga de constructores. Desde el segundo constructor (el que tiene un único parámetro) se llama utilizando la palabra reservada **this** al constructor Empleados (String, double, int, int, int).



## Paquetes

Java permite agrupar las clases en una colección denominada paquete. Los paquetes son una forma cómoda de organizar nuestro trabajo, y también sirven para separar nuestro trabajo de las bibliotecas de código que proporcionan otras personas.

La biblioteca estándar de Java se distribuye en un cierto número de paquetes denominados **Java.lang**, **Java.util** y **Java.net** y así sucesivamente. Estos paquetes son jerárquicos.

Todos los paquetes estándar de Java se encuentran dentro de las jerarquías de paquetes **java** y **javax**.

La razón principal por la que se utilizan paquetes es para garantizar la exclusividad del nombre de las clases. Si dos clases se llaman exactamente igual, no habrá problema mientras estén almacenadas en paquetes diferentes.

## Importación de paquetes

Las clases pueden utilizar todas las clases de su propio paquete, así como las clases públicas de otros paquetes. Se puede acceder a las clases públicas de otros paquetes de dos maneras:

Añadiendo el nombre del paquete completo delante del nombre de todas las clases.

**Ejemplo:**

```
java.util.Date hoy = new java.util.Date();
```

Utilizando la sentencia **import**. Una vez utilizada la sentencia **import** no hace falta dar a las clases su nombre íntegro.

Se puede importar una clase en concreto o bien todo el paquete. Las sentencias **import** se ponen al principio de los ficheros de código fuente.

**Ejemplo:**

```
import java.util.*;
```

Y después ya podremos utilizar **Date hoy = new Date();** sin anteponer el paquete. También se puede importar una clase concreta perteneciente a un paquete:

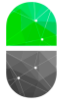
```
import Java.util.Date;
```

La sentencia **Java.util.\*** es menos tediosa.

No se puede importar más de un paquete en una única sentencia. Si queremos utilizar las clases pertenecientes a dos paquetes o más, habrá que incluir dos o más sentencias import.

## Añadir una clase a un paquete

Para ubicar las clases dentro de un paquete, es necesario poner el nombre del paquete en la parte superior del fichero del código fuente antes del código que define las clases del paquete.

**Ejemplo:**

```
package es.pildorasinformaticas.poo;  
import java.util.GregorianCalendar;  
  
public class UsoEmpleados {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
    }  
}
```

Los ficheros del paquete se ubican en un subdirectorio que coincide con el nombre completo del paquete. Por ejemplo, todos los ficheros de clases del paquete paq1 deberían residir en un subdirectorio de Windows llamado paq1.