CLASES: Constructores, elemento this y colaboración entre clases

En Java podemos definir un método que se ejecute inicialmente y en forma automática. Este método se llama **constructor**.

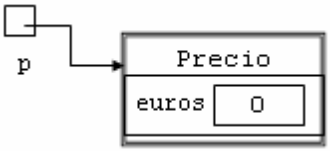
Aunque en un principio pueda parecer lo contrario, un *constructor* no es en realidad un método estrictamente hablando.

Un constructor es un método que se llama automáticamente cada vez que se crea un objeto de una clase. La principal misión del constructor es reservar memoria e inicializar las variables atributos de la clase.

El constructor es tan importante, que si el programador no prepara ningún constructor para una clase, el compilador crea uno por defecto, inicializando las variables de los tipos primitivos a su valor por defecto 0 o false, mientras que los atributos de tipo *objeto* (referencia) a null, los booleanos a false y los caracteres al carácter con el código ASCII cero

Por ejemplo:

```
public class Precio {
public static void main (String [] args ) {
Precio p; // Crea una referencia de la clase Precio
p = new Precio(); // Crea el objeto de la clase Precio y realiza
una llamada al método constructor
// Resto del codigo ...
}
}
```



La clase Precio, utiliza una instancia de la clase Precio, la llamada al constructor se produce en la sentencia p = new Precio (); Mientras que la ejecución de new genera una nueva instancia y devuelve su dirección de memoria, la ejecución del constructor Precio() inicializa los valores de los atributos .El constructor tiene las siguientes características:

- Tiene el mismo nombre de la clase.
- Es el primer método que se ejecuta para crear un objeto.
- Se ejecuta en forma automática al crear el objeto.

- No puede retornar datos.
- Se ejecuta una única vez en la creación del objeto concretamente.
- Un constructor tiene por objetivo inicializar atributos.
- La existencia o no de parámetros es opcional

Por otro lado, la *sobrecarga (*Un método sobrecargado se utiliza para reutilizar el nombre de un método pero con diferentes argumentos (opcionalmente un tipo diferente de retorno)) permite que puedan declararse varios constructores (con el mismo identificador que el de la clase), siempre y cuando tengan un tipo y/o número de parámetros distinto.

Por ejemplo, para la clase Fecha se declaran dos constructores, el primero sin parámetros y el segundo con tres parámetros:

```
public class Fecha {
// Atributos o variables miembro
private int dia;
private int mes;
private int anio;
/**
* Constructor 1
* Asigna los valores 1, 1 y 2000 a los atributos
* dia, mes y anio respectivamente
*/
public Fecha() {
dia = 1;
mes = 1;
anio = 2000;
}
```

```
/**
* Constructor 2
* ndia el dia del mes a almacenar
* nmes el mes del año a almacenar
* nanio el año a almacenar
*/
public Fecha(int ndia, int nmes, int nanho) {
dia = ndia;
mes = nmes;
anio = nanio;
}
public String toString() { /*devuelve los datos del objeto en una línea
separados por comas */
return  dia + "/" + mes + "/" + anio;}}
```

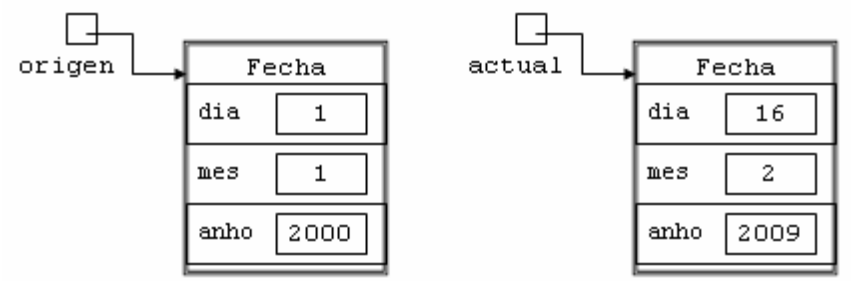
La **sobrecarga o polimorfismo** es una de las características más importantes de la programación orientada a objetos. La **sobrecarga** permite que existan en una clase varios métodos con el mismo nombre pero con distinto tipo o número de parámetros. De esta forma, cuando el método sea llamado, en función de los parámetros que recibe en la llamada, el compilador sabrá cuál de ellos tiene que ejecutar.

Esto lo podemos aplicar a los constructores. La ***sobrecarga*** permite que puedan declararse varios constructores (con el mismo identificador que el de la clase), siempre y cuando tengan un tipo y/o número de parámetros distinto.

Para probar el código anterior, se construye el siguiente programa:

```
public class PruebaFecha {
public static void main (String [] args) {
```

```
Fecha origen = new Fecha();

Fecha actual = new Fecha(16,2,2009);

System.out.println("Primera fecha: " + origen.toString());

System.out.println("Segunda fecha: " + actual.toString());

}

}
```



El código anterior puede compilarse y ejecutarse, mostrando la siguiente salida por pantalla:

Primera fecha: 1/1/2000

Segunda fecha: 16/2/2009

El elemento o palabra clave **this** en Java

Dentro de todo método disponemos del elemento this. Este elemento hace referencia al objeto que está llamando al método que estamos ejecutando. Es fundamental (también es sencillo) entender el funcionamiento de este elemento, veamos ejemplos:
La sintaxis de Java permite que el nombre de un parámetro o una variable local sea el mismo que el nombre de una variable de instancia.Cuando esto sucede, el nombre local oculta la variable de instancia. Puede obtener acceso a la variable de instancia oculta haciendo referencia a ella a través de **this**. Por ejemplo, la siguiente es una forma sintácticamente válida para escribir otro constructor de Fecha donde no tenemos que poner otros nombres distintos en los parámetros.

public **Fecha(int dia, int mes, int anho)** {

this.dia = dia;

this.mes = mes;

this.anio = anio;

}

Otro ejemplo que no tiene gran utilidad práctica pero si considero que es didáctico para entender este concepto es el siguiente:

Supongamos que en una clase Fraccion queremos hacer un método para comparar los numeradores y que indique en pantalla cual es el mayor numerador y cual el menor.

```
public class Fraccion {
        private int numerador;
        private int denominador;
        Fraccion(){
                numerador=0;
                denominador=1;
        }
        Fraccion(int numerador, int denominador){
                this.numerador=numerador;
                this.denominador=denominador;
        }
```

Nuestro método será ejecutado por un objeto mediante el nombre del objeto punto nombre del método: f1.ComparaNumerador (f2), luego el primer numerador a comparar será el del propio objeto que llama al método.

El segundo objeto tendrá que ser recibido como parámetro en el método.

¿De qué tipo será el elemento que recibe el método? Pues espero que no haya dudas, el objeto que va a recibir el método será también de tipo Fraccion, del mismo tipo que el objeto que llama al método, simplemente porque vamos a comparar sus numeradores. Es decir tendremos algo así:

```
Fraccion f1 = new Fraccion( );
Fraccion f2 = new Fraccion( );
f1.comparaNumerador( f2 ); //compara el numerador de f1 con el de f2
```

//También podríamos hacer
f2.comparaNumerador(f1); //compara el numerador de f2 con el de f1

¿No te recuerda esto a algunos métodos de los objetos que ya hemos visto como los strings? Método compareTo y similares.

El resultado que nosotros buscamos en la comparación será similar a este:

```
Dame el numerador de la fracción 1: 5
Dame el denominador de la fracción 1: 7
Dame el numerador de la fracción 2: 8
Dame el denominador de la fracción 2: 3
5/7 y 8/3
-------------------------------------------
El objeto tiene un numerador menor que el parámetro: 5 < 8
-------------------------------------------
El objeto tiene un numerador mayor o igual que el parámetro: 8 >= 5
```

Hemos obtenido este resultado haciendo las dos llamadas pidiendo los datos y llamando al método comparaNumerador desde ambos objetos:f1 y f2.
El método es muy sencillo, y en él veremos la utilización del elemento this.

```
public void comparaNumerador(Fraccion f){
    if (this.numerador>=f.numerador)
        System.out.println("El objeto tiene un numerador mayor o
igual que el parámetro: "+this.numerador+" >= "+f.numerador);
    else
        System.out.println("El objeto tiene un numerador menor que
el parámetro: "+this.numerador+" < "+f.numerador);
    }
```

El método comparaNumerador es llamado por un objeto Fraccion y recibe como parámetro otro objeto Fraccion. Ambos objetos tienen los mismos atributos (numerador y denominador). ¿Cómo podemos distinguir los de cada uno?
Pues bien, es muy sencillo, para los del parámetro siempre colocaremos delante del atributo el nombre del objeto que recibimos como parámetro seguido de un punto y el nombre del atributo que deseamos utilizar. Por ejemplo:

f.numerador hace referencia al numerador del objeto recibido como parámetro.
f.denominador hace referencia al denominador del objeto recibido como parámetro.

Pero como identificamos el numerador y el denominador del objeto que llama al método.
Pues bien aquí entra en juego el elemento this.
Hemos indicado que este elemento hace referencia al objeto que llama al método, pues bien para acceder a sus atributos y métodos bastará con hacerlo de la siguiente manera:

this.numerador hace referencia al numerador del objeto que llama al método.
this.denominador hace referencia al denominador del objeto que llama al método.

Ahora cuando vamos a utilizar el método y realizamos la llamada al método comparaNumerador, cuando hacemos:

f1.comparaNumerador(f2);
El objeto que hace la llamada es f1, y el parámetro es f2. Cuando utilizamos this nos estamos refiriendo al objeto f1.

Por el contrario, cuando hacemos:

f2.comparaNumerador(f1);

El objeto que hace la llamada es f2 y el parámetro es f1, Cuando utilizamos this nos estaremos refiriendo al objeto f2.
Con esto espero haya quedado claro la utilización del this.
Vamos ahora a ver los métodos get y set que son habituales en cualquier clase con atributos privados, con otro ejemplo:

```
public class Constructores {

        private String nombre, apellido, tipo;

        private  int horas;
```

```
public Constructores(String nombre, String apellido, String tipo, int horas){
                    this. nombre = nombre;
                    this. apellido = apellido;
                    this. tipo = tipo;
                    this. horas = horas;
    }
```

Se ha usado la sentencia this para distinguir las variables de la instancia(objeto), de los parámetros del constructor

Recordamos que en java existen tres tipos de variables:

Variables de clase: Es un campo declarado como static, y son comunes a todas las clases

Variables de instancia: son los atributos de cada clase (objeto) en particular y son campos no estáticos

Variables locales: son las variables propias de cada método

Hasta ahora hemos utilizado solo una clase pero si utilizamos varias clases y quisiéramos utilizar las variables de una clase en otra clase distinta, porque se han declarado como privadas, tenemos que implementar métodos para acceder a ellas

Métodos set: Modifican los valores de los atributos, por ejemplo

setNombre("Adela") cambia el nombre que hubieras antes en el atributo Nombre del objeto al que aplicamos el método por Adela

Métodos get: devuelve los valores de los atributos del objeto para poder utilizarlos en otras clases.

Por ejemplo:

```
public class Constructores {
```

```
        private String nombre, apellido, tipo;
        private  int horas;


public Constructores(String nombre, String apellido, String tipo, int horas){
        this. nombre = nombre;
        this. apellido = apellido;
        this. tipo = tipo;
        this. horas = horas;
        }
public String getNombre(){
                return nombre;
    }
public String getApellido (){
                        return apellido;
                        }
public String getTipo (){
                        return tipo;
                        }
public int getHoras (){
                        return horas;
                        }

public void setNombre(String nombre){
                        this.nombre = nombre;
```

```
                    }
public void setApellido (String apellido){
                    this. apellido = apellido;
                    }
public void setTipo (String tipo){
                    this. tipo = tipo;
                    }
public void setHoras (int horas){
                    this. horas = horas;
                    }
public String getNombreCompleto(){
                    return nombre +" " + apellido;
          }
public double getSueldoBruto (){
               if(tipo.equals("ciencia"))
                    return 3 * horas;
               else
                    return 5 * horas;
          }
   public double getDescuento (){
               return 0.10* getSueldoBruto ();
          }
      public double getNeto (){
               return getSueldoBruto () - getDescuento ();
          }
      public String toString() {
```

```
        return "Constructores [nombre=" + nombre + ", apellido=" +
apellido
                                        + ", tipo=" + tipo + ", horas=" + horas + "]";
                }
}//acaba la clase Constructores
```

En otra clase tendremos el correspondiente main:

```
 public class mainConstructores(){


public static void main(String[] args) {
Constructores c =new Constructores("Carmen","Rubio","ciencia",13);
Constructores c1 =new Constructores("Juan Emilio","Rivero","ciencia",14);
                        System.out.println(c.getNombre());
                        System.out.println(c1.getNombre());
                        c.getApellido();
                        c.getTipo();
                        c.getHoras();
                        c.setNombre("Adela");
        System.out.println("Nombre Completo: "+c.getNombreCompleto());
                System.out.println(c1.toString());
                  System.out.println("Descuento  "+c.getDescuento());
                  System.out.println("Neto  "+c.getNeto());
                }
        }
```

¿Que resultado tendríamos?

Los getters y setters pueden normalmente añadirlos automáticamente con ayuda del entorno que utilicemos.

Ejercicios propuestos:

1.-Implementar la clase posición que representa un punto (x, y) en el eje de coordenadas. Cada posición viene definida por dos valores enteros x e y.

Realiza:

- Constructor por defecto que se corresponda con la posición (0,0)
- Constructor al que se le pasa como parámetro los valores iniciales de las coordenadas x e y
- Métodos de modificación y consulta (set/get) de los atributos de la clase
- Métodos para incrementar y decrementar cada una de las coordenadas

2.-Crear una clase que se llame ExamenConducirTeorico, con un unico atributo que se llame numFallos.

- Poseerá un constructor al que se le pase como parametro el numero de fallos .
- Método que se llamara esApto, no se le pasa ningun parámetro. Si numFallos es menos o igual que 3, mostrara un mensaje por pantalla (por ejemplo: aprobado). Sino mostrara otro mensaje (por ejemplo: suspenso).
- Para probar la clase, en un programa principal, crearemos un objeto de la clase (asignándole el número de fallos que queramos) y haciendo una llamada a la función esApto.

3.- Realiza una clase número, cuyo atributo es un número entero y que tenga las siguientes características:

- Constructor por defecto que inicializa a cero el atributo.
- Constructor que inicializa el numero con un parámetro.
- Método suma que permite sumarle un numero al valor interno.
- Método resta que reste un numero al valor interno.
- Método getValor devuelve el valor interno.
- Método getDoble devuelve el doble del valor interno.
- Método setNumero inicializa de nuevo el valor interno.