

## 5.E. Utilizando métodos y atributos de clase.

Sitio: [Formación Profesional a Distancia](#)  
Curso: Programación  
Libro: 5.E. Utilizando métodos y atributos de clase.  
Imprimido por: Iván Jiménez Utiel  
Día: martes, 7 de enero de 2020, 22:19

## Tabla de contenidos

[1. Utilización de los métodos y atributos de una clase.](#)

[1.1. Declaración de un objeto.](#)

[1.2. Creación de un objeto.](#)

[1.3. Manipulación de un objeto: utilización de métodos y atributos.](#)

## 1. Utilización de los métodos y atributos de una clase.

Una vez que ya tienes implementada una [clase](#) con todos sus atributos y métodos, ha llegado el momento de utilizarla, es decir, de instanciar objetos de esa [clase](#) e interaccionar con ellos. En unidades anteriores ya has visto cómo declarar un [objeto](#) de una [clase](#) determinada, instanciarlo con el operador `new` y utilizar sus métodos y atributos.

### Para saber más

Puedes echar un vistazo a los artículos sobre la creación y uso de objetos en Java en los manuales de Oracle (en inglés):

[Creating Objects.](#)

[Using Objects.](#)

### 1.1. Declaración de un objeto.

Como ya has visto en unidades anteriores, la declaración de un [objeto](#) se realiza exactamente igual que la declaración de una [variable](#) de cualquier tipo:

```
<tipo> nombreVariable;
```

En este caso el tipo será alguna [clase](#) que ya hayas implementado o bien alguna de las proporcionadas por la biblioteca de Java o por alguna otra biblioteca escrita por terceros.

Por ejemplo:

```
Punto p1;
```

```
Rectangulo r1, r2;
```

```
Coche cocheAntonio;
```

```
String palabra;
```

Esas variables (p1, r1, r2, cocheAntonio, palabra) en realidad son referencias (también conocidas como punteros o direcciones de memoria) que apuntan (hacen "referencia") a un [objeto](#) (una zona de memoria) de la [clase](#) indicada en la declaración.

Como ya estudiaste en la unidad dedicada a los objetos, un [objeto](#) recién declarado (referencia recién creada) no apunta a nada. Se dice que la referencia está vacía o que es una referencia nula (la [variable objeto](#) contiene el valor `null`). Es decir, la [variable](#) existe y está preparada para guardar una dirección de memoria que será la zona donde se encuentre el [objeto](#) al que hará referencia, pero el [objeto](#) aún no existe (no ha sido creado o instanciado). Por tanto se dice que apunta a un [objeto](#) nulo o inexistente.

Para que esa [variable](#) (referencia) apunte realmente a un [objeto](#) (contenga una referencia o dirección de memoria que apunte a una zona de memoria en la que se ha reservado espacio para un [objeto](#)) es necesario crear o instanciar el [objeto](#). Para ello se utiliza el operador `new`.

#### Autoevaluación

Aunque la declaración de un [objeto](#) es imprescindible para poder utilizarlo, ese [objeto](#) no existirá hasta que no se construya una [instancia](#) de la [clase](#) del [objeto](#). Es decir, mientras la [clase](#) no sea instanciada, el [objeto](#) aún no existirá y lo único que se tendrá será una [variable](#) que contendrá un [objeto](#) vacío o nulo. ¿Verdadero o falso?

- ☐ Verdadero.
- ☐ Falso.

#### Ejercicio resuelto

Utilizando la [clase Rectangulo](#) implementada en ejercicios anteriores, indica como declararías tres objetos (variables) de esa [clase](#) llamados **r1**, **r2**, **r3**.

#### Solución

Se trata simplemente de realizar una declaración de esas tres variables:

```
Rectangulo r1;
```

```
Rectangulo r2:
```

```
Rectangulo r3:
```

También podrías haber declarado los tres objetos en la misma sentencia de declaración:

```
Rectangulo r1, r2, r3;
```

## 1.2. Creación de un objeto.

Para poder crear un [objeto](#) ([instancia](#) de una [clase](#)) es necesario utilizar el operador **new**, el cual tiene la siguiente [sintaxis](#):

```
nombreObjeto= new <ConstructorClase> ([listaParametros]);
```

El constructor de una [clase](#) (**ConstructorClase**) es un [método](#) especial que tiene toda [clase](#) y cuyo nombre coincide con el de la [clase](#). Es quien se encarga de crear o construir el [objeto](#), solicitando la reserva de memoria necesaria para los atributos e inicializándolos a algún valor si fuera necesario. Dado que el constructor es un [método](#) más de la [clase](#), podrá tener también su lista de parámetros como tienen todos los métodos.



De la tarea de reservar memoria para la estructura del [objeto](#) (sus atributos más alguna otra información de carácter interno para el entorno de ejecución) se encarga el propio entorno de ejecución de Java. Es decir, que por el hecho de ejecutar un [método](#) constructor, el entorno sabrá que tiene que realizar una serie de tareas (solicitud de una zona de memoria disponible, reserva de memoria para los atributos, enlace de la [variable objeto](#) a esa zona, etc.) y se pondrá rápidamente a desempeñarlas.

Cuando escribas el código de una [clase](#) no es necesario que implementes el [método](#) constructor si no quieres hacerlo. Java se encarga de dotar de un constructor por omisión (también conocido como **constructor por defecto**) a toda [clase](#). Ese constructor por omisión se ocupará exclusivamente de las tareas de reserva de memoria. Si deseas que el constructor realice otras tareas adicionales, tendrás que escribirlo tú. El constructor por omisión no tiene parámetros.

El constructor por defecto no se ve en el código de una [clase](#). Lo incluirá el [compilador](#) de Java al compilar la [clase](#) si descubre que no se ha creado ningún [método](#) constructor para esa [clase](#).

Algunos ejemplos de instanciación o creación de objetos podrían ser:

```
p1= new Punto ();
```

```
r1= new Rectangulo ();
```

```
r2= new Rectangulo;
```

```
cocheAntonio= new Coche();
```

```
palabra= String;
```

En el caso de los constructores, si éstos no tienen parámetros, pueden omitirse los paréntesis vacíos.

Un [objeto](#) puede ser declarado e instanciado en la misma línea. Por ejemplo:

```
Punto p1= new Punto ();
```

### Autoevaluación

Si una clase no tiene constructor porque el programador no lo ha implementado, Java se encargará de dotar a esa clase de un constructor por defecto de manera que cualquier clase instanciable siempre tendrá al menos un constructor. ¿Verdadero o falso?

- ☐ Verdadero.
- ☐ Falso.

### Ejercicio resuelto

Ampliar el ejercicio anterior instanciando los objetos **r1**, **r2**, **r3** mediante el constructor por defecto.

### Solución

Habría que añadir simplemente una sentencia de creación o instanciación (llamada al constructor mediante el operador **new**) por cada objeto que se desee crear:

```
Rectangulo r1, r2, r3;
```

```
r1= new Rectangulo ();
```

```
r2= new Rectangulo ();
```

```
r3= new Rectangulo ();
```

### 1.3. Manipulación de un objeto: utilización de métodos y atributos.

Una vez que un [objeto](#) ha sido declarado y creado ([clase](#) instanciada) ya sí se puede decir que el [objeto](#) existe en el entorno de ejecución, y por tanto que puede ser manipulado como un [objeto](#) más en el programa, haciéndose uso de sus atributos y sus métodos.

Para acceder a un miembro de un [objeto](#) se utiliza el operador **punto** (.) del siguiente modo:

```
<nombreObjeto>.<nombreMiembro>
```

Donde **<nombreMiembro>** será el nombre de algún miembro del [objeto](#) ([atributo](#) o [método](#)) al cual se tenga acceso.

Por ejemplo, en el caso de los objetos de tipo **Punto** que has declarado e instanciado en los apartados anteriores, podrías acceder a sus miembros de la siguiente manera:

```
Punto p1, p2, p3;
```

```
p1= new Punto();
```

```
p1.x= 5;
```

```
p1.y= 6;
```

```
System.out.printf ("p1.x: %d\np1.y: %d\n", p1.x, p1.y);
```

```
System.out.printf ("p1.x: %d\np1.y: %d\n", p1.obtenerX(), p1.obtenerY());
```

```
p1.establecerX(25);
```

```
p1.establecerX(30);
```

```
System.out.printf ("p1.x: %d\np1.y: %d\n", p1.obtenerX(), p1.obtenerY());
```

Es decir, colocando el operador **punto** (.) a continuación del nombre del [objeto](#) y seguido del nombre del miembro al que se desea acceder.

#### Ejercicio resuelto

Utilizar el ejemplo de los rectángulos para crear un rectángulo **r1**, asignarle los valores x1=0, y1=0, x2=10, y2=10, calcular su área y su perímetro y mostrarlos en pantalla.

#### Solución

Se trata de declarar e instanciar el [objeto](#) r1, rellenar sus atributos de ubicación (coordenadas de las esquinas), e invocar a los métodos **calcularSuperficie** y **calcularPerimetro** utilizando el operador **punto** (.). Por ejemplo:

```
Rectangulo r1= new Rectangulo ();
```

```
r1.x= 0;
```

```
r1.y= 0;
```



```
r2.x= 10;
```

```
r2.y= 10;
```

```
area= r1.calcularSuperficie ();
```

```
perímetro= r1.calcularPerimetro ();
```

Por último faltaría mostrar en pantalla la información calculada.

Los ficheros completos serían **Rectangulo.java** y **EjemploRectangulos01.java** que se muestran a continuación:

### Rectangulo.java

```
package ejemplorectangulos01;
```

```
/**-----
```

```
* Clase Rectangulo
```

```
-----*/
```

```
public class Rectangulo {
```

```
    // Atributos de clase
```

```
    private static int numRectangulos;           // Número total de rectángulos creados
```

```
    public static final String nombreFigura= "Rectángulo";    // Nombre de la clase
```

```
    public static final double PI= 3.1416;         // Constante PI
```

```
    // Atributos de objeto
```

```
    private String nombre;    // Nombre del rectángulo
```

```
    public double x1, y1;    // Vértice inferior izquierdo
```

```
    public double x2, y2;    // Vértice superior derecho
```

```
    // Método obtenerNombre
```

```
    public String obtenerNombre () {
```

```
        return nombre;
    }

    // Método establecerNombre
    public void establecerNombre (String nom) {
        nombre= nom;
    }
```

```
    // Método CalcularSuperficie
    public double CalcularSuperficie () {
        double area, base, altura;

        // Cálculo de la base
        base= x2-x1;

        // Cálculo de la altura
        altura= y2-y1;

        // Cálculo del área
        area= base * altura;

        // Devolución del valor de retorno
        return area;
    }
```

```
    // Método CalcularPerimetro
    public double CalcularPerimetro () {
        double perimetro, base, altura;

        // Cálculo de la base
        base= x2-x1;
```

```
// Cálculo de la altura
altura= y2-y1;

// Cálculo del perímetro
perimetro= 2*base + 2*altura;

// Devolución del valor de retorno
return perimetro;
}

// Método desplazar
public void desplazar (double X, double Y) {

// Desplazamiento en el eje X
x1= x1 + X;
x2= x2 + X;

// Desplazamiento en el eje Y
y1= y1 + Y;
y2= y2 + Y;

}

// Método obtenerNumRectangulos
public static int obtenerNumRectangulos () {
    return numRectangulos;
}

}
```

## /\*

 $\ast/$ 

/\*\*

\*

 $\ast/$ 

```
r1.desplazar (3,3);
```

```
        System.out.printf ("r1.x1: %4.2f\nr1.y1: %4.2f\n", r1.x1, r1.y1);  
        System.out.printf ("r1.x2: %4.2f\nr1.y2: %4.2f\n", r1.x2, r1.y2);  
  
    }  
}
```