5.F. Constructores.

Sitio: Formación Profesional a Distancia
Curso: Programación
Libro: 5.F. Constructores.
Imprimido por: Iván Jiménez Utiel
Día: martes, 7 de enero de 2020, 22:26

Tabla de contenidos

1. Constructores.

Como ya has estudiado en unidades anteriores, en el ciclo de vida de un objeto se pueden distinguir las fases de:

- Construcción del objeto.
- Manipulación y utilización del objeto accediendo a sus miembros.
- Destrucción del objeto.

Como has visto en el apartado anterior, durante la fase de construcción o instanciación de un objeto es cuando se reserva espacio en memoria para sus atributos y se inicializan algunos de ellos. Un **constructor** es un método especial con el **mismo nombre de la clase** y que se encarga de realizar este proceso.

El proceso de declaración y creación de un objeto mediante el operador **new** ya ha sido estudiado en apartados anteriores. Sin embargo las clases que hasta ahora has creado no tenían constructor. Has estado utilizando los constructores por defecto que proporciona Java al compilar la clase. Ha llegado el momento de que empieces a implementar tus propios constructores.

Los métodos constructores se encargan de llevar a cabo el proceso de creación o construcción de un objeto.

Autoevaluación

¿Con qué nombre es conocido el método especial de una clase que se encarga de reservar espacio e inicializar atributos cuando se crea un objeto nuevo? ¿Qué nombre tendrá ese método en la clase?

- Método constructor. Su nombre dentro de la clase será constructor.
- Método inicializador. Su nombre dentro de la clase será el mismo nombre que tenga la clase.
- Método constructor. Su nombre dentro de la clase será el mismo nombre que tenga la clase.
- Método constructor. Su nombre dentro de la clase será new.

1.1. Concepto de constructor.

Un **constructor** es un método que tiene el mismo nombre que la clase a la que pertenece y que no devuelve ningún valor tras su ejecución. Su función es la de proporcionar el mecanismo de creación de instancias (objetos) de la clase.

Cuando un objeto es declarado, en realidad aún no existe. Tan solo se trata de un nombre simbólico (una variable) que en el futuro hará referencia a una zona de memoria que contendrá la información que representa realmente a un objeto. Para que esa variable de objeto aún "vacía" (se suele decir que es una referencia nula o vacía) apunte, o haga referencia a una zona de memoria que represente a una instancia de clase (objeto) existente, es necesario "**construir**" el objeto. Ese proceso se realizará a través del método **constructor** de la clase.

Por tanto para crear un nuevo objeto es necesario realizar una llamada a un método constructor de la clase a la que pertenece ese objeto. Ese proceso se realiza mediante la utilización del operador `new`.

Hasta el momento ya has utilizado en numerosas ocasiones el operador `new` para instanciar o crear objetos. En realidad lo que estabas haciendo era una llamada al constructor de la clase para que reservara memoria para ese objeto y por tanto "crear" físicamente el objeto en la memoria (dotarlo de existencia física dentro de la memoria del ordenador). Dado que en esta unidad estás ya definiendo tus propias clases, parece que ha llegado el momento de que empieces a escribir también los constructores de tus clases.

Por otro lado, si un constructor es al fin y al cabo una especie de método (aunque algo especial) y Java soporta la sobrecarga de métodos, podrías plantearte la siguiente pregunta: ¿podrá una clase disponer de más de constructor? En otras palabras, ¿será posible la sobrecarga de constructores? La respuesta es afirmativa.

Una misma clase puede disponer de varios constructores. Los constructores soportan la sobrecarga.

Es necesario que toda clase tenga al menos un constructor. Si no se define ningún constructor en una clase, el compilador creará por nosotros un constructor por defecto vacío que se encarga de inicializar todos los atributos a sus valores por defecto (0 para los numéricos, `null` para las referencias, `false` para los `boolean`, etc.).

Algunas analogías que podrías imaginar para representar el constructor de una clase podrían ser:

- Los moldes de cocina para flanes, galletas, pastas, etc.
- Un cubo de playa para crear castillos de arena.
- Un molde de un lingote de oro.
- Una bolsa para hacer cubitos de hielo.

Una vez que incluyas un constructor personalizado a una clase, el compilador ya no incluirá el constructor por defecto (sin parámetros) y por tanto si intentas usarlo se produciría un error de compilación. Si quieres que tu clase tenga también un constructor sin parámetros tendrás que escribir su código (ya no lo hará por ti el compilador).

1.2. Creación de constructores.

Cuando se escribe el código de una clase normalmente se pretende que los objetos de esa clase se creen de una determinada manera. Para ello se definen uno o más constructores en la clase. En la definición de un constructor se indican:

- El tipo de acceso.
- El nombre de la clase (el nombre de un método constructor es siempre el nombre de la propia clase).
- La lista de parámetros que puede aceptar.
- Si lanza o no excepciones.
- El cuerpo del constructor (un bloque de código como el de cualquier método).

Como puedes observar, la estructura de los constructores es similar a la de cualquier método, con las excepciones de que **no tiene tipo de dato devuelto** (no devuelve ningún valor) y que **el nombre del método constructor debe ser obligatoriamente el nombre de la clase**.

Reflexiona

Si defines constructores personalizados para una clase, el constructor por defecto (sin parámetros) para esa clase deja de ser generado por el compilador, de manera que tendrás que crearlo tú si quieres poder utilizarlo.

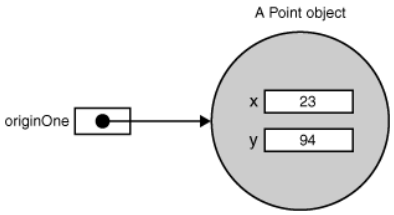
Si se ha creado un constructor con parámetros y no se ha implementado el constructor por defecto, el intento de utilización del constructor por defecto producirá un error de compilación (el compilador no lo hará por nosotros).

Un ejemplo de constructor para la clase `Punto` podría ser:

```
public Punto (int x, int y)
{
    this.x= x;
    this.y= y;
    cantidadPuntos++;  // Suponiendo que tengamos un atributo estático cantidadPuntos
}
```

En este caso el constructor recibe dos parámetros. Además de reservar espacio para los atributos (de lo cual se encarga automáticamente Java), también asigna sendos valores iniciales a los atributos x e y. Por último

incrementa un atributo (probablemente estático) llamado `cantidadPuntos`.



Autoevaluación

El constructor por defecto (sin parámetros) está siempre disponible para usarlo en cualquier clase. ¿Verdadero o falso?

- Verdadero.
- Falso.

1.3. Utilización de constructores.

Una vez que dispongas de tus propios constructores personalizados, la forma de utilizarlos es igual que con el constructor por defecto (mediante la utilización de la palabra reservada `new`) pero teniendo en cuenta que si has declarado parámetros en tu método constructor, tendrás que llamar al constructor con algún valor para esos parámetros.

Un ejemplo de utilización del constructor que has creado para la clase Punto en el apartado anterior podría ser:

```
Punto p1;
```

```
p1= new Punto (10, 7);
```

En este caso no se estaría utilizando el constructor por defecto sino el constructor que acabas de implementar en el cual además de reservar memoria se asigna un valor a algunos de los atributos.

Para saber más

Puedes echar un vistazo al artículo sobre constructores de una clase Java en los manuales de Oracle (en inglés):

Providing Constructors for Your Classes.

Ejercicio resuelto

Ampliar el ejercicio de la clase **Rectangulo** añadiéndole tres constructores:

1. Un constructor sin parámetros (para sustituir al constructor por defecto) que haga que los valores iniciales de las esquinas del rectángulo sean (0,0) y (1,1);
2. Un constructor con cuatro parámetros, **x1**, **y1**, **x2**, **y2**, que rellene los valores iniciales de los atributos del rectángulo con los valores proporcionados a través de los parámetros.
3. Un constructor con dos parámetros, **base** y **altura**, que cree un rectángulo donde el vértice inferior derecho esté ubicado en la posición (0,0) y que tenga una base y una altura tal y como indican los dos parámetros proporcionados.

Solución

En el caso del primer constructor lo único que hay que hacer es "rellenar" los atributos x1, y1, x2, y2 con los valores 0, 0, 1, 1:

```
public Rectangulo ()
```

```
{
```

```
    x1= 0.0;
```

```
    y1= 0.0;
```

```
    x2= 1.0;
```

```
    y2= 1.0;
}
```

Para el segundo constructor es suficiente con asignar a los atributos x1, y1, x2, y2 los valores de los parámetros x1, y1, x2, y2. Tan solo hay que tener en cuenta que al tener los mismos nombres los parámetros del método que los atributos de la clase, estos últimos son ocultados por los primeros y para poder tener acceso a ellos tendrás que utilizar el operador de autorrerferencia `this`:

```
public Rectangulo (double x1, double y1, double x2, double y2)
{
    this.x1= x1;
    this.y1= y1;
    this.x2= x2;
    this.y2= y2;
}
```

En el caso del tercer constructor tendrás que inicializar el vértice (x1, y1) a (0,0) y el vértice (x2,y2) a (0 + base, 0 + altura), es decir a (base, altura):

```
public Rectangulo (double base, double altura)
{
    this.x1= 0.0;
    this.y1= 0.0;
    this.x2= base;
    this.y2= altura;
}
```

El código completo lo compondrán los ficheros Rectangulo.java y EjemploRectangulos02.java que incorporaremos en un mismo proyecto:

Rectangulo.java

package ejemplorectangulos02;

/**----------

 * Clase Rectangulo.

 * Incluye constructores.

```
 ----------------------------------------------------------------*/
public class Rectangulo {

   // Atributos de clase (estáticos)
   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

//-----------------
// Constructores
//-----------------

   public Rectangulo ()
   {
         x1= 0.0;
         y1= 0.0;
         x2= 1.0;
         y2= 1.0;
   }

   public Rectangulo (double x1, double y1, double x2, double y2)
   {
         this.x1= x1;
         this.y1= y1;
         this.x2= x2;
         this.y2= y2;
```

```
    }

    public Rectangulo (double base, double altura)
    {
        this.x1= 0.0;
        this.y1= 0.0;
        this.x2= base;
        this.y2= altura;
    }


//--------------------------------
// Métodos estáticos (de clase)
//--------------------------------

    // Métodos de estáticos públicos
    // ----------------------------

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


//-------------------
// Métodos de objeto
//-------------------

    //Métodos públicos
    //-----------------
```

```
// 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 X
        y1= y1 + Y;
        y2= y2 + Y;
    }
}
```

EjemploRectangulos02.java

```
/*
 * Ejemplo de uso de la clase Rectangulo con constructores
 */
```

```
package ejemplorectangulos02;


/**
 *
 * Programa Principal (clase principal)
 */
public class EjemploRectangulos02 {



   public static void main(String[] args) {
      Rectangulo r1, r2, r3;


      System.out.printf ("PRUEBA DE USO DE LA CLASE RECTÁNGULO\n");
      System.out.printf ("-----------------------------------\n\n");
      System.out.printf ("Creando rectángulos...\n\n");
      r1= new Rectangulo ();
      r2= new Rectangulo (1,1, 3,3);
      r3= new Rectangulo (10, 5);


      System.out.printf ("Recángulo r1: \n");
      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);
      System.out.printf ("Perimetro: %4.2f\nSuperficie: %4.2f\n", r1.CalcularPerimetro(),
r1.CalcularSuperficie());
      System.out.printf ("Recángulo r2: \n");
      System.out.printf ("r2.x1: %4.2f\nr2.y1: %4.2f\n", r2.x1, r2.y1);
      System.out.printf ("r2.x2: %4.2f\nr2.y2: %4.2f\n", r2.x2, r2.y2);
      System.out.printf ("Perimetro: %4.2f\nSuperficie: %4.2f\n", r2.CalcularPerimetro(),
r2.CalcularSuperficie());
      System.out.printf ("Recángulo r3: \n");
      System.out.printf ("r3.x1: %4.2f\nr3.y1: %4.2f\n", r3.x1, r3.y1);
```

```
        System.out.printf ("r3.x2: %4.2f\nr3.y2: %4.2f\n", r3.x2, r3.y2);

        System.out.printf ("Perimetro: %4.2f\nSuperficie: %4.2f\n", r3.CalcularPerimetro(),
r3.CalcularSuperficie());


    }

}
```

1.4. Constructores de copia.

Una forma de iniciar un objeto podría ser mediante la copia de los valores de los atributos de otro objeto ya existente. Imagina que necesitas varios objetos iguales (con los mismos valores en sus atributos) y que ya tienes uno de ellos perfectamente configurado (sus atributos contienen los valores que tú necesitas). Estaría bien disponer de un constructor que hiciera copias idénticas de ese objeto.

Durante el proceso de creación de un objeto puedes generar objetos exactamente iguales (basados en la misma clase) que se distinguirán posteriormente porque podrán tener estados distintos (valores diferentes en los atributos). La idea es poder decirle a la clase que además de generar un objeto nuevo, que lo haga con los mismos valores que tenga otro objeto ya existente. Es decir, algo así como si pudieras **clonar** el objeto tantas veces como te haga falta. A este tipo de mecanismo se le suele llamar **constructor copia** o **constructor de copia**.

Un constructor copia es un método constructor como los que ya has utilizado pero con la particularidad de que recibe como parámetro una referencia al objeto cuyo contenido se desea copiar. Este método revisa cada uno de los atributos del objeto recibido como parámetro y se copian todos sus valores en los atributos del objeto que se está creando en ese momento en el método constructor.

Un ejemplo de constructor copia para la clase `Punto` podría ser:

```
public Punto (Punto p)
{
    this.x= p.obtenerX();
    this.y= p.obtenerY();
}
```

En este caso el constructor recibe como parámetro un objeto del mismo tipo que el que va a ser creado (clase `Punto`), inspecciona el valor de sus atributos (atributos `x` e `y`), y los reproduce en los atributos del objeto en proceso de construcción (`this`).

Un ejemplo de utilización de ese constructor podría ser:

```
Punto p1, p2;
p1= new Punto (10, 7);
p2= new Punto (p1);
```

En este caso el objeto `p2` se crea a partir de los valores del objeto `p1`.

Autoevaluación

Toda clase debe incluir un constructor copia en su implementación. ¿Verdadero o falso?

- Verdadero.
- Falso.

Ejercicio resuelto

Ampliar el ejercicio de la clase Rectangulo añadiéndole un constructor copia.

Solución

Se trata de añadir un nuevo constructor además de los tres que ya habíamos creado:

```
// Constructor copia
public Rectangulo (Rectangulo r) {
    this.x1= r.x1;
    this.y1= r.y1;
    this.x2= r.x2;
    this.y2= r.y2;
}
```

Para usar este constructor basta con haber creado anteriormente otro `Rectangulo` para utilizarlo como base de la copia. Por ejemplo:

```
Rectangulo r1, r2;
r1= new Rectangulo (0,0,2,2);
r2= new Rectangulo (r1);
```

1.5. Destrucción de objetos.

Como ya has estudiado en unidades anteriores, cuando un objeto deja de ser utilizado, los recursos usados por él (memoria, acceso a archivos, conexiones con bases de datos, etc.) deberían de ser liberados para que puedan volver a ser utilizados por otros procesos (mecanismo de **destrucción del objeto**).

Mientras que de la construcción de los objetos se encargan los métodos constructores, de la destrucción se encarga un proceso del entorno de ejecución conocido como **recolector de basura** (**garbage collector**). Este proceso va buscando periódicamente objetos que ya no son referenciados (no hay ninguna variable que haga referencia a ellos) y los marca para ser eliminados. Posteriormente los irá eliminando de la memoria cuando lo considere oportuno (en función de la carga del sistema, los recursos disponibles, etc.).

Normalmente se suele decir que en Java no hay método destructor y que en otros lenguajes orientados a objetos como **C++**, sí se implementa explícitamente el destructor de una clase de la misma manera que se define el constructor. En realidad en Java también es posible implementar el método destructor de una clase, se trata del método `finalize()`.

Este método `finalize` es llamado por el recolector de basura cuando va a destruir el objeto (lo cual nunca se sabe cuándo va a suceder exactamente, pues una cosa es que el objeto sea marcado para ser borrado y otra que sea borrado efectivamente). Si ese método no existe, se ejecutará un destructor por defecto (el método `finalize` que contiene la clase `Object`, de la cual heredan todas las clases en Java) que liberará la memoria ocupada por el objeto. Se recomienda por tanto que si un objeto utiliza determinados recursos de los cuales no tienes garantía que el entorno de ejecución los vaya a liberar (cerrar archivos, cerrar conexiones de red, cerrar conexiones con bases de datos, etc.), implementes explícitamente un método `finalize` en tus clases. Si el único recurso que utiliza tu clase es la memoria necesaria para albergar sus atributos, eso sí será liberado sin problemas. Pero si se trata de algo más complejo, será mejor que te encargues tú mismo de hacerlo implementando tu destructor personalizado (`finalize`).

Por otro lado, esta forma de funcionar del entorno de ejecución de Java (destrucción de objetos no referenciados mediante el recolector de basura) implica que no puedas saber exactamente cuándo un objeto va a ser definitivamente destruido, pues si una variable deja de ser referenciada (se cierra el ámbito de ejecución donde fue creada) no implica necesariamente que sea inmediatamente borrada, sino que simplemente es marcada para que el recolector la borre cuando pueda hacerlo.

Si en un momento dado fuera necesario garantizar que el proceso de finalización (método `finalize`) sea invocado, puedes recurrir al método `runFinalization ()` de la clase `System` para forzarlo:

```
System.runFinalization ();
```

Este método se encarga de llamar a todos los métodos `finalize` de todos los objetos marcados por el recolector de basura para ser destruidos.

Si necesitas implementar un destructor (normalmente no será necesario), debes tener en cuenta que:

- El nombre del método destructor debe ser `finalize ()`.
- No puede recibir parámetros.
- Sólo puede haber un destructor en una clase. No es posible la sobrecarga dado que no tiene parámetros.
- No puede devolver ningún valor. Debe ser de tipo `void`.

Autoevaluación

Cuando se abandona el ámbito de un objeto en Java éste es marcado por el recolector de basura para ser destruido. En muchas ocasiones una clase Java no tiene un método destructor, pero si fuera necesario hacerlo ¿podrías implementar un método destructor en una clase Java? ¿Qué nombre habría que ponerle?

- Sí es posible. El nombre del método sería finalize().
- No es posible disponer de un método destructor en una clase Java.
- Sí es posible. El nombre del método sería destructor ().
- Sí es posible. El nombre del método sería ~nombreClase, como en el lenguaje C++.