3.C. Trabajando con objetos.

Sitio: Formación Profesional a Distancia
Curso: Programación
Libro: 3.C. Trabajando con objetos.
Imprimido por: Iván Jiménez Utiel
Día: jueves, 7 de noviembre de 2019, 15:20

Tabla de contenidos

1. Utilización de objetos.

Una vez que hemos creado una clase, podemos crear objetos en nuestro programa a partir de esas clases.

Cuando creamos un objeto a partir de una clase se dice que hemos creado una **"instancia de la clase".** A efectos prácticos, "objeto" e "instancia de clase" son términos similares. Es decir, nos referimos a objetos como instancias cuando queremos hacer hincapié que son de una clase particular.

Los objetos se crean a partir de las clases, y representan casos individuales de éstas.

Para entender mejor el concepto entre un objeto y su clase, piensa en un **molde de galletas y las galletas**. El molde sería la clase, que define las características del objeto, por ejemplo su forma y tamaño. Las galletas creadas a partir de ese molde son los objetos o instancias.

Otro ejemplo, imagina una clase `Persona` que reúna las características comunes de las personas (color de pelo, ojos, peso, altura, etc.) y las acciones que pueden realizar (crecer, dormir, comer, etc.). Posteriormente dentro del programa podremos crear un objeto `Trabajador` que esté basado en esa clase `Persona`. Entonces se dice que el objeto `Trabajador` es una instancia de la clase `Persona`, o que la clase `Persona` es una abstracción del objeto `Trabajador`.

Cualquier objeto instanciado de una clase contiene una copia de todos los atributos definidos en la clase. En otras palabras, lo que estamos haciendo es reservar un espacio en la memoria del ordenador para guardar sus atributos y métodos. Por tanto, cada objeto tiene una **zona de almacenamiento propia** donde se guarda toda su información, que será distinta a la de cualquier otro objeto. A las variables miembro instanciadas también se les llama **variables instancia.** De igual forma, a los métodos que manipulan esas variables se les llama **métodos instancia.**

En el ejemplo del objeto `Trabajador`, las variables instancia serían `color_de_pelo`, `peso`, `altura`, etc. Y los métodos instancia serían `crecer()`, `dormir()`, `comer()`, etc.

Autoevaluación

Las variables instancia son un tipo de variables miembro.

- Verdadero
- Falso

1.1. Ciclo de vida de los objetos.

Todo programa en Java parte de una única clase, que como hemos comentado se trata de la clase principal. Esta clase ejecutará el contenido de su método **main()**, el cual será el que utilice las demás clases del programa, cree objetos y lance mensajes a otros objetos.

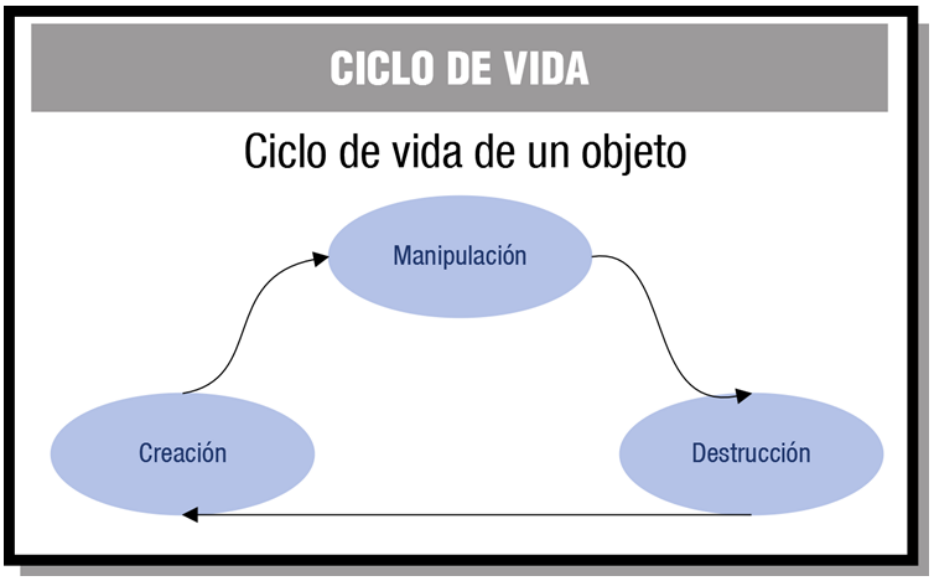

Imagen extraída de curso Programación del MECD.

Las instancias u objetos tienen un tiempo de vida determinado. Cuando un objeto no se va a utilizar más en el programa, es destruido por el recolector de basura para liberar recursos que pueden ser reutilizados por otros objetos.

A la vista de lo anterior, podemos concluir que los objetos tienen un ciclo de vida, en el cual podemos distinguir las siguientes fases:

- **Creación**, donde se hace la reserva de memoria e inicialización de atributos.
- **Manipulación**, que se lleva a cabo cuando se hace uso de los atributos y métodos del objeto.
- **Destrucción**, eliminación del objeto y liberación de recursos.

1.2. Declaración.

Para la creación de un objeto hay que seguir los siguientes pasos:

- **Declaración**: Definir el tipo de objeto.

- **Instanciación**: Creación del objeto utilizando el operador `new`**.**

Pero ¿en qué consisten estos pasos a nivel de programación en Java? Veamos primero cómo declarar un objeto. Para la definición del tipo de objeto debemos emplear la siguiente instrucción:

```
<tipo> nombre_objeto;
```

Donde:

- `tipo` es la clase a partir de la cual se va a crear el objeto, y

- `nombre_objeto` es el nombre de la variable referencia con la cual nos referiremos al objeto.

Los tipos referenciados o referencias se utilizan para guardar la dirección de los datos en la memoria del ordenador.

Nada más crear una referencia, ésta se encuentra vacía. Cuando una referencia a un objeto no contiene ninguna instancia se dice que es una referencia nula, es decir, que contiene el valor `null`**.** Esto quiere decir que la referencia está creada pero que el objeto no está instanciado todavía, por eso la referencia apunta a un objeto inexistente llamado "nulo".

Para entender mejor la declaración de objetos veamos un ejemplo. Cuando veíamos los tipos de datos en la Unidad 2, decíamos que Java proporciona un tipo de dato especial para los textos o cadenas de caracteres que era el tipo de dato `String`. Veíamos que realmente este tipo de dato es un **tipo referenciado** y creábamos una variable mensaje de ese tipo de dato de la siguiente forma:

```
String mensaje;
```

Los nombres de la clase empiezan con mayúscula, como `String`, y los nombres de los objetos con minúscula, como `mensaje`, así sabemos qué tipo de elemento utilizando.

Pues bien, `String` es realmente la clase a partir de la cual creamos nuestro objeto llamado `mensaje`.

Si observas poco se diferencia esta declaración de las declaraciones de variables que hacíamos para los tipos primitivos. Antes decíamos que `mensaje` era una variable del tipo de dato `String`**.** Ahora realmente vemos que `mensaje` es un objeto de la clase `String`. Pero `mensaje` aún no contiene el objeto porque no ha sido instanciado, veamos cómo hacerlo.

Por tanto, cuando creamos un objeto estamos haciendo uso de una variable que almacena la dirección de ese objeto en memoria. Esa variable es una **referencia** o un **tipo de datos referenciado**, porque no contiene el dato si no la posición del dato en la memoria del ordenador.

```
String saludo = new String ("Bienvenido a Java");
```

```
String s; //s vale null
```

```
s = saludo; //asignación de referencias
```

En las instrucciones anteriores, las variables `s` y `saludo` apuntan al mismo objeto de la clase `String`. Esto implica que cualquier modificación en el objeto saludo modifica también el objeto al que hace referencia la variable `s`, ya que realmente son el mismo.

1.3. Instanciación.

Una vez creada la referencia al objeto, debemos crear la instancia u objeto que se va a guardar en esa referencia. Para ello utilizamos la orden **new** con la siguiente sintaxis:

```
nombre_objeto = new <Constructor_de_la_Clase>([<par1>, <par2>, ..., <parN>]);
```

Donde:

- `nombre_objeto` es el nombre de la variable referencia con la cual nos referiremos al objeto,
- `new` es el operador para crear el objeto,
- `Constructor_de_la_Clase` es un método especial de la clase, que se llama igual que ella, y se encarga de inicializar el objeto, es decir, de dar unos valores iniciales a sus atributos, y
- `par1-parN` son parámetros que puede o no necesitar el constructor para dar los valores iniciales a los atributos del objeto.

Durante la instanciación del objeto, se reserva memoria suficiente para el objeto. De esta tarea se encarga Java y juega un papel muy importante el **recolector de basura,** que se encarga de eliminar de la memoria los objetos no utilizados para que ésta pueda volver a ser utilizada.

De este modo, para instanciar un objeto `String`, haríamos lo siguiente:

```
mensaje = new String;
```

Así estaríamos instanciando el objeto mensaje. Para ello utilizaríamos el operador `new` y el constructor de la clase `String` a la que pertenece el objeto según la declaración que hemos hecho en el apartado anterior. A continuación utilizamos el constructor, que se llama igual que la clase, `String`.

En el ejemplo anterior el objeto se crearía con la cadena vacía (""), si queremos que tenga un contenido debemos utilizar parámetros en el constructor, así:

```
mensaje = new String ("El primer programa");
```

Java permite utilizar la clase `String` como si de un tipo de dato primitivo se tratara, por eso no hace falta utilizar el operador new para instanciar un objeto de la clase `String`.

La declaración e instanciación de un objeto puede realizarse en la misma instrucción, así:

```
String mensaje = new String ("El primer programa");
```

1.4. Manipulación.

Una vez creado e instanciado el objeto ¿cómo accedemos a su contenido? Para acceder a los atributos y métodos del objeto utilizaremos el nombre del objeto seguido del **operador punto (.)** y el nombre del atributo o método que queremos utilizar. Cuando utilizamos el operador punto se dice que estamos enviando un mensaje al objeto. La forma general de enviar un mensaje a un objeto es:

```
nombre_objeto.mensaje
```

Por ejemplo, para acceder a las variables instancia o atributos se utiliza la siguiente sintaxis:

```
nombre_objeto.atributo
```

Y para acceder a los métodos o funciones miembro del objeto se utiliza la sintaxis es:

```
nombre_objeto.método( [par1, par2, ..., parN] )
```

En la sentencia anterior `par1`, `par2`, etc. son los parámetros que utiliza el método. Aparece entre corchetes para indicar son opcionales.

Para entender mejor cómo se manipulan objetos vamos a utilizar un ejemplo. Para ello necesitamos la Biblioteca de Clases Java o API (Application Programming Interface - Interfaz de programación de aplicaciones). Uno de los paquetes de librerías o bibliotecas es `java.awt.` Este paquete contiene clases destinadas a la creación de objetos gráficos e imágenes. Vemos por ejemplo cómo crear un rectángulo.

En primer lugar instanciamos el objeto utilizando el método constructor, que se llama igual que el objeto, e indicando los parámetros correspondientes a la posición y a las dimensiones del rectángulo:

```
Rectangle rect = new Rectangle(50, 50, 150, 150);
```

Una vez instanciado el objeto rectángulo si queremos cambiar el valor de los atributos utilizamos el operador punto. Por ejemplo, para cambiar la dimensión del rectángulo:

```
rect.height=100;
```

```
rect.width=100;
```

O bien podemos utilizar un método para hacer lo anterior:

```
rect.setSize(200, 200);
```

A continuación puedes ver el código del ejemplo:

```java
public class Manipular {


    public static void main(String[] args) {

        // Instanciamos el objeto rect indicando posicion y dimensiones
        Rectangle rect = new Rectangle( 50, 50, 150, 150 );

        //Consultamos las coordenadas x e y del rectangulo
        System.out.println( "-------- Coordenadas esquina superior izqda. --------");
        System.out.println("\tx = " + rect.x + "\n\ty = " + rect.y);

        // Consultamos las dimensiones (altura y anchura) del rectangulo
        System.out.println( "\n------------------- Dimensiones --------------------");
        System.out.println("\tAlto = " + rect.height );
        System.out.println( "\tAncho = " + rect.width);


        //Cambiar coordenadas del rectangulo
        rect.height=100;
        rect.width=100;

        rect.setSize(200, 200);
        System.out.println( "\n-- Nuevos valores de los atributos --");
        System.out.println("\tx = " + rect.x + "\n\ty = " + rect.y);
        System.out.println("\tAlto = " + rect.height );
        System.out.println( "\tAncho = " + rect.width);


    }

}
```

1.5. Destrucción de objetos y liberación de memoria.

Cuando un objeto deja de ser utilizado, es necesario liberar el espacio de memoria y otros recursos que poseía para poder ser reutilizados por el programa. A esta acción se le denomina **destrucción del objeto.**

En Java la destrucción de objetos corre a cargo del **recolector de basura (garbage collector)**. Es un sistema de destrucción automática de objetos que ya no son utilizados. Lo que se hace es liberar una zona de memoria que había sido reservada previamente mediante el operador `new`. Esto evita que los programadores tengan que preocuparse de realizar la liberación de memoria.

El recolector de basura se ejecuta en modo segundo plano y de manera muy eficiente para no afectar a la velocidad del programa que se está ejecutando. Lo que hace es que periódicamente va buscando objetos que ya no son referenciados, y cuando encuentra alguno lo marca para ser eliminado. Después los elimina en el momento que considera oportuno.

Justo antes de que un objeto sea eliminado por el recolector de basura, se ejecuta su método `finalize()`. Si queremos forzar que se ejecute el proceso de finalización de todos los objetos del programa podemos utilizar el método `runFinalization()` de la clase `System`. La clase `System` forma parte de la Biblioteca de Clases de Java y contiene diversas clases para la entrada y salida de información, acceso a variables de entorno del programa y otros métodos de diversa utilidad. Para forzar el proceso de finalización ejecutaríamos:

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

Autoevaluación

Las fases del ciclo de vida de un objeto son: Creación, Manipulación y Destrucción.

- Verdadero
- Falso