APRENDERAPROGRAMAR.COM

SOBREESCRIBIR MÉTODOS EN JAVA: TOSTRING, EQUALS. EJEMPLOS Y EJERCICIOS RESUELTOS. COMPARAR OBJETOS. (CU00694B)

Sección: Cursos
Categoría: Curso “Aprender programación Java desde cero”

Fecha revisión: 2029

Resumen: Entrega nº94 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

SOBREESCRIBIR MÉTODOS EN JAVA. EJEMPLO: TOSTRING

Con lo expuesto en anteriores apartados del tutorial, ya sabemos que mientras no se hayan sobreescrito, los métodos de la superclase universal Object estarán disponibles para todos los objetos. Dentro de los métodos de la clase Object hay varios importantes, entre los que podemos citar los métodos *toString()* que devuelve un tipo String y *equals (Object obj)* que devuelve un tipo boolean.

Veamos primero el método toString(). El propósito de este método es **asociar a todo objeto un texto representativo.** Llamar a toString() sobre un objeto Integer producirá un resultado que es más o menos obvio: nos devuelve el entero asociado, solo que en forma de String. ¿Pero qué ocurre cuando invocamos el método sobre un objeto definido por nosotros? Este sería el caso de una invocación como *System.out.println ("Obtenemos " + profesor1.toString());*. En este caso, el resultado que obtenemos es del tipo: "Obtenemos Profesor@1de9ac4". El método efectivamente nos ha devuelto un String, ¿pero qué sentido tiene lo que nos ha devuelto? El resultado obtenido consta del nombre de la clase seguido de una cadena "extraña" que representa la dirección de memoria en que se encuentra el objeto. Este resultado en general es poco útil por lo que el método toString() es un método que habitualmente se sobreescribe al crear una clase. De hecho, la mayoría de las clases de la biblioteca estándar de Java sobreescriben el método toString(). Por otro lado, es frecuente que cuando un programador incluye métodos como imprimir…, mostrar…, listar…, etc. de alguna manera dentro de ellos se realicen llamadas al método toString() para evitar la repetición de código. También es frecuente que toString() aparezca en tareas de depuración ya que nos permite interpretar de forma "humana" los objetos. Supongamos que en nuestra clase Persona redefinimos toString() de la siguiente manera:

```
public String toString() { return nombre.concat(" "). concat(apellidos); }
```

En la clase profesor, que hereda de Persona, podríamos tener:

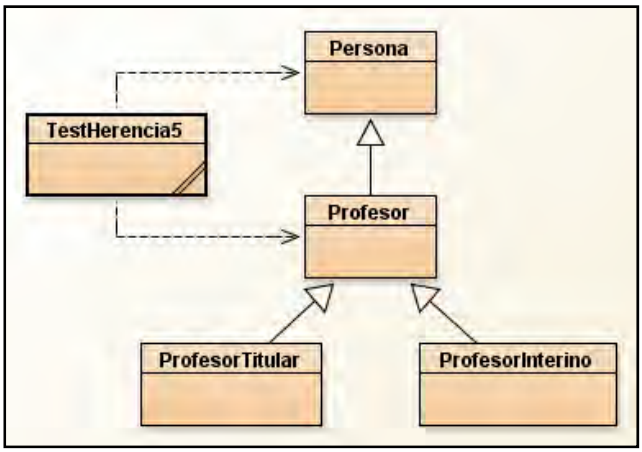
```
public String toString() { return super.toString().concat(" con Id de profesor: ").concat ( getIdProfesor() ); }
public void mostrarDatos() { System.out.println ("Los datos disponibles son: " + this.toString() ); }
```

En este ejemplo vemos, aparte del uso del método concat para concatenar Strings, una llamada a la superclase para recuperar el método toString de la superclase y cómo otro método (mostrarDatos) hace uso del método toString(). Usar toString es ventajoso porque no siempre nos interesa mostrar cadenas de texto por consolas en pantalla: tener los datos en un String nos permite p.ej. grabarlos en una base de datos, enviarlos en un correo electrónico, etc., además de poder mostrarlos por pantalla si queremos.

Aunque ya venimos usándolo, conviene remarcar que cuando usamos los métodos println y print del objeto System.out, cuando se incluye un término que no es un String, se invoca automáticamente el método toString() del objeto sin necesidad de escribirlo de forma explícita. Así *System.out.println (Profesor);* es equivalente a *System.out.println (Profesor.toString());*.

SOBREESCRIBIR MÉTODOS DE LA CLASE OBJECT: MÉTODO EQUALS

Ya hemos utilizado el método equals en diferentes ocasiones y sabemos que es la forma en que debemos comparar objetos. **Los objetos no se pueden comparar utilizando el operador ==.** El método equals está implementado en el API de Java para la mayoría de las clases. Por ello, podemos usarlo directamente para comparar Strings por ejemplo. Ahora bien, ¿qué ocurre en una clase creada por nosotros? Si escribimos algo como if (profesor1.equals(profesor2)), al no estar sobreescrito el método equals para la clase Profesor, el resultado es impredecible o incorrecto. Por tanto, para comparar objetos de tipo Profesor hemos de sobreescribir el método en la clase. Vamos a ver cómo realizaríamos la sobreescritura de este método dentro de la estructura de herencia en la que venimos trabajando. Escribe y compila el código que mostramos a continuación y trata de comprenderlo.




```
// Código que añadimos a la clase Persona. Sobreescritura del método equals ejemplo aprenderaprogramar.com
   public boolean equals (Object obj) {
      if (obj instanceof Persona) {
      Persona tmpPersona = (Persona) obj;
         if (this.nombre.equals(tmpPersona.nombre) && this.apellidos.equals(tmpPersona.apellidos) &&
 this.edad == tmpPersona.edad) { return true; } else { return false; }
   } else { return false; }
} //Cierre del método equals
```

```
// Código que añadimos a la clase Profesor. Sobreescritura del método equals ejemplo aprenderaprogramar.com
public boolean equals (Object obj) {
      if (obj instanceof Profesor) {
         Profesor tmpProfesor = (Profesor) obj;
         if (super.equals(tmpProfesor) && this.IdProfesor.equals(tmpProfesor.IdProfesor) ) {
            return true; } else { return false; }
      }  else { return false; }
   } // Cierre del método equals
```

```
//Clase test herencia método equals ejemplo aprenderaprogramar.com
import java.util.Calendar;
public class TestHerencia5{
   public static void main (String [ ] Args) {
      Profesor profesor1 = new Profesor ("Juan", "Hernández García", 33);
      profesor1.setIdProfesor("Prof 22-387-11");
      Profesor profesor2 = new Profesor ("Juan", "Hernández García", 33);
      profesor2.setIdProfesor("Prof 22-387-11");
      Profesor profesor3 = new Profesor ("Juan", "Hernández García", 33);
      profesor3.setIdProfesor("Prof 11-285-22");

      Persona persona1 = new Persona ("José", "Hernández López", 28);
      Persona persona2 = new Persona ("José", "Hernández López", 28);
      Persona persona3 = new Persona ("Ramiro", "Suárez Rodríguez", 19);

      System.out.println ("¿Son iguales la persona1 y la persona2? " + persona1.equals(persona2) );
      System.out.println ("¿Son el mismo objeto la persona1 y la persona2? " + (persona1 == persona2) );
      System.out.println ("¿Son iguales la persona1 y la persona3? " + persona1.equals(persona3) );

      System.out.println ("¿Son iguales el profesor1 y el profesor2? " + profesor1.equals(profesor2) );
      System.out.println ("¿Son iguales el profesor1 y el profesor3? " + profesor1.equals(profesor3) );

   } //Cierre del main
} //Cierre de la clase
```

Comrpueba que obtienes estos resultados al ejecutar el main:

```
¿Son iguales la persona1 y la persona2? true
¿Son el mismo objeto la persona1 y la persona2? false
¿Son iguales la persona1 y la persona3? false
¿Son iguales el profesor1 y el profesor2? true
¿Son iguales el profesor1 y el profesor3? false
```

Analicemos ahora lo que hemos hecho. En la clase Persona, mediante la sobreescritura del método equals, hemos definido que para nosotros dos personas van a ser iguales si coinciden sus nombre, apellidos y edad. El criterio lo hemos fijado nosotros. Otra opción hubiera sido establecer que dos personas son iguales si coinciden nombre y apellidos. Fíjate que los Strings los comparamos usando el método equals del API de Java para los objetos de tipo String, mientras que la edad la comparamos con el operador == por ser un tipo primitivo. Otra cuestión relevante es el tratamiento de tipos: **el método equals requiere como parámetro un tipo Object** y no un tipo Persona. Así hemos de escribirlo para que realmente sea una redefinición del método de la clase Object. Si usáramos otra signatura, no sería una redefinición o sobreescritura del método, sino un nuevo método. En primer lugar comprobamos si el objeto pasado como parámetro es un tipo Persona. Si no lo es, devolvemos como resultado del método false: los objetos no son iguales (no pueden serlo si ni siquiera coinciden sus tipos). En segundo lugar, una vez verificado que el objeto es portador de un tipo Persona, creamos una variable de tipo Persona a la que asignamos el objeto pasado como parámetro valiéndonos de casting (enmascaramiento). Esta variable la creamos para poder invocar campos y métodos de la clase Persona, ya que esto no podemos

hacerlo sobre un objeto de tipo Object. Con esta variable, realizamos las comparaciones oportunas y devolvemos un resultado.

En la clase Profesor hemos sobreescrito también el método equals con un tratamiento de tipos y uso de casting similar. En este caso invocamos el método equals de la superclase Persona, con lo que decimos que para que dos profesores sean iguales han de coincidir nombre, apellidos y edad. Además establecemos otro requisito para considerar a dos profesores iguales: que coincida su IdProfesor. Esto lo hacemos a nuestra conveniencia.

Finalmente en el test realizamos pruebas de los métodos implementados, comprobando por ejemplo que una persona1 puede ser igual a otra persona2 (de acuerdo con la definición dada al método equals) pero ambas ser diferentes objetos.

EJERCICIO

Define una clase Figura de la que hereden otras dos clases denominadas Cuadrado y Círculo. La clase figura debe tener al menos el campo dimensionPrincipal. Las clases Cuadrado y Círculo deben tener al menos un método calcularArea que permita calcular el área a partir de la dimensión principal, utilizando la fórmula matemática correspondiente. Además, sobreescribe el método equals para que dos cuadrados sean iguales si tienen igual dimensión principal, y dos círculos idem. A continuación crea un programa test donde crees varios círculos y cuadrados y hagas comprobaciones de igualdad usando el método equals.

Para comprobar si tu código es correcto puedes consultar en los foros aprenderaprogramar.com.

Próxima entrega: CU00695B

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188