Sobreescribir
toString()
hashCode()

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

Índice

1 | .toString()

.toString()

Toda clase hereda de Object el método toString(), es decir, si no lo implementamos, los objetos que instanciamos tendrán este método. Por ejemplo, en nuestra clase Empleado, con nombre, legajo, sueldo y descuentos como atributos, qué pasaría si uso el método toString():

```
public class Empleado{
      private   String nombre;
      private   String legajo;
      protected double sueldo;
      protected double descuentos;

}
```

.toString()

Al usar el método, no tendríamos un error, pero la información mostrada no sería algo comprensible:

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

   Empleado nuevoEmpleado=new Empleado("Juan","1111");

   System.out.println(nuevoEmpleado.toString());
}
```

Esta es la salida que obtenemos `com.company.Empleado@1540e19d`

.toString(): por qué sobreescribirlo

El método .toString(), intenta representar con texto el objeto, pero como no lo sobreescribimos, vamos a obtener ese tipo de salida. La solución es sobreescribir el método mostrando solo la información que deseamos mostrar y dándole a la cadena de salida el formato más adecuado.

Sobreescribir toString()

Recordemos que es importante no cambiar la firma del método, sino estaremos sobrecargando.

```
public class Empleado{
      private   String nombre;
      private   String legajo;
      protected double sueldo;
      protected double descuentos;

      @Override
      public String toString(){
            return “Nombre: ” + nombre + “\n” +
                  “Legajo: ” + legajo;
      }
}
```

Agregamos el método toString() y devolvemos la cadena con la información del objeto que queremos devolver.

La salida que obtenemos es la que nosotros programamos, en este caso, el nombre y el legajo.

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

    Empleado nuevoEmpleado=new Empleado("Juan","1111");

    System.out.println(nuevoEmpleado.toString());
}
```

```
Nombre: Juan
Legajo: 1111
```

2 | .hashCode()

.hashCode()

Este es otro de los métodos heredados de Object.
Cuando se utiliza este método nos devuelve un número único que identifica al objeto, es decir, si tengo dos objetos de la misma clase, el hashCode() generaría un número distinto para cada uno y ese número me va a servir para identificarlo.

La utilidad de este identificador la vamos a ver más adelante cuando veamos estructuras de datos más avanzadas.

Pero ahora lo que tenemos que asegurarnos es que este código identificador sea único para cada objeto.

.hashCode(): cómo sobreescribirlo

```
public class Empleado{
      private   String nombre;
      private   String legajo;
      protected double sueldo;
      protected double descuentos;

      @Override
      public int hashCode(){
            int hash=31;
            hash= hash* nombre.hashCode();
            hash= hash* legajo.hashCode();
            return hash;}
}
```

La clase Empleados con el .hashCode() sobrecargado. A continuación, veremos cómo se resuelve.

Para generar un número único se trabaja con números primos.
Puede ser cualquier número primo, en este caso se usó el 31.
Como nombre y legajo son strings, o sea, también son objetos, tienen su propio hashCode(). Multiplicamos todos los números y obtenemos el hashCode del objeto.
En una string, el hashCode se genera a partir de los caracteres.
Por ejemplo, el número de legajo es siempre distinto.

```
@Override
public int hashCode(){
      int hash=31;
      hash= hash* nombre.hashCode();
      hash= hash* legajo.hashCode();
      return hash;}
```

Con la sobrecarga que hicimos, obtenemos el valor que se muestra:

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

      Empleado nuevoEmpleado=new Empleado("Juan","1111");

    System.out.println(nuevoEmpleado.hashCode());
}
```

```
-1480218112
```


DigitalHouse>
Coding School