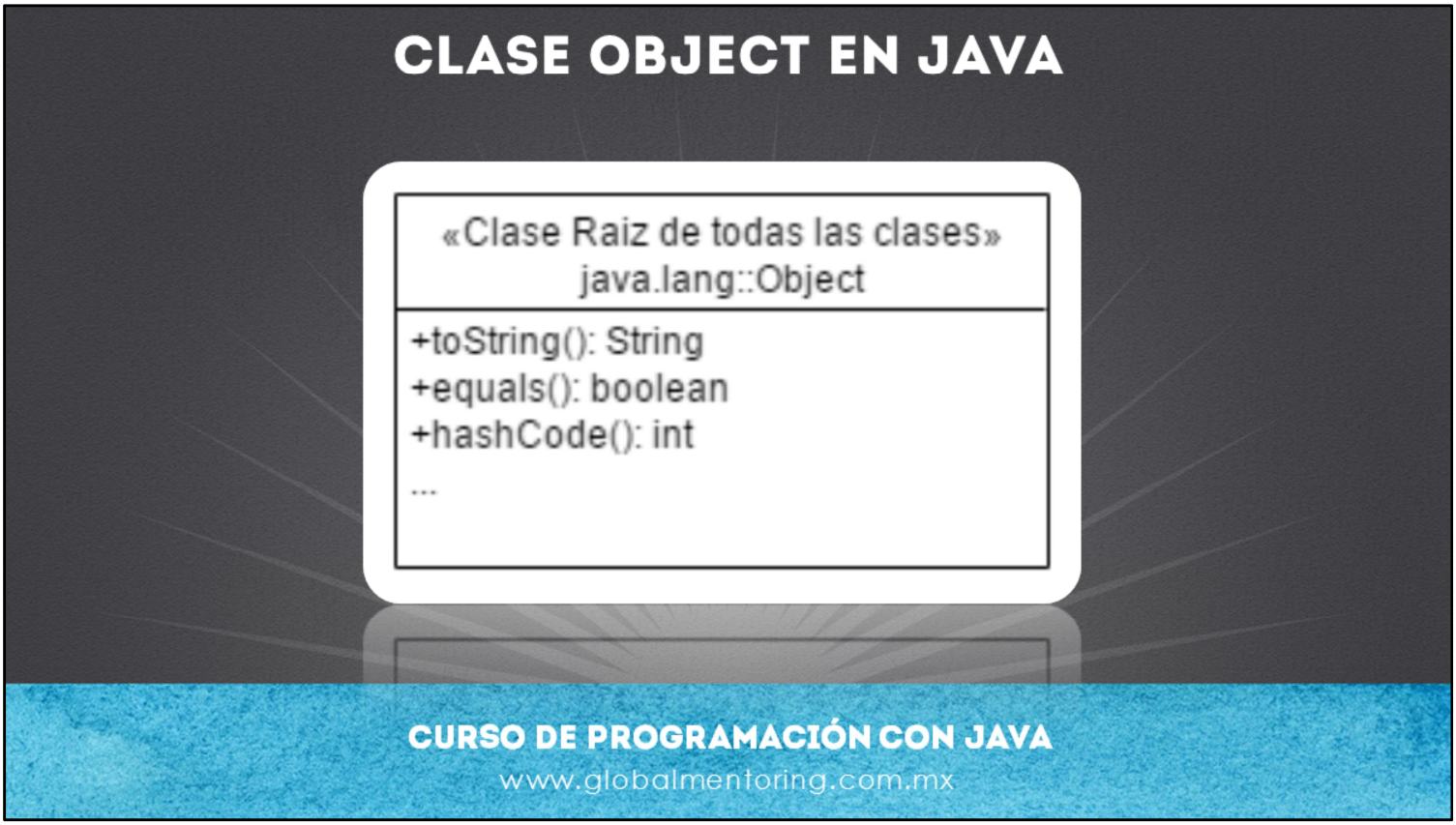
Hola, te saluda nuevamente Ubaldo Acosta. Espero que estés listo para comenzar con esta lección.

Vamos a estudiar el tema de la clase Object en Java, además de la sobreescritura de métodos como es el método toString, equals y hashcode.

¿Estás listo? ¡Vamos!



En Java, todas las clases heredan de la clase Object.

La clase Object es la clase raíz de todas las clases en Java. La clase Object se encuentra en el paquete java.lang, el cual es el paquete principal o core de Java, y es por esta razón que no hay necesidad de hacer import de las clases que pertenezcan al paquete Java, de esta manera se simplifica el uso de estas clases.

¿Y por qué es tan importante la clase Object? Por que TODAS las clase en Java de manera directa o indirecta heredan de la clase Object, de esta manera el lenguaje Java puede estar seguro que compartirá al menos esto en común entre todas las clases Java, y esto permite asegurar que cierta funcionalidad o características sea compartida entre todas las clases en Java.

Existen varios métodos muy importantes que estaremos utilizando constantemente de la clase Object, y mejor dicho, estaremos sobreescribiendo estos métodos heredados en nuestras propias clases. Algunos de estos métodos son el método toString, equals y hashCode.

Vamos a ver a más detalle cada uno de ellos.

MÉTODO TOSTRING

Ejemplo método toString en Java:

```
public class Empleado {

    private String nombre;

    private double sueldo;

    public Empleado(String nombre, double sueldo){
        this.nombre = nombre;
        this.sueldo = sueldo;
    }

    @Override
    public String toString() {
        return "Empleado{" + "nombre=" + nombre + ", sueldo=" + sueldo + '}';
    }
}
```


Todos los objetos en Java heredan de la clase Object, de manera directa si no indican la palabra extends en la definición de su clase, y de manera indirecta si heredan de otra clase, debido a la jerarquía de clases.

Por lo tanto hereda los métodos mencionados anteriormente. Uno de esos métodos heredados es el método toString. Este método nos sirve para que podamos mostrar una representación en texto de nuestras clases, y al sobreescribirlo entonces, podemos fácilmente mandar a imprimir o mostrar el estado de un objeto.

Si no sobreescribimos el método toString en nuestra clase, obtendremos un resultado como: MiClase@12ba40c3, el cual se compone del nombre de la clase, seguido de la dirección hexadecimal donde se ubica en memoria el objeto. Sin embargo esto no nos dice mucho acerca del estado de nuestro método. Por estado nos referimos a los valores actuales de cada atributo de nuestra clase.

Debido a esto, con el método toString tenemos la oportunidad de mostrar en una cadena, el estado de nuestro objeto, simplemente concatenando cada uno de los valores que nos interese mostrar. Casi siempre utilizaremos todos los atributos, pero debemos tener cuidado en los casos más complejos, en los que algunos atributos sean otros objetos, y en ocasiones podemos generar problemas de recursividad o llamadas circulares con otros objetos.

Podemos observar en el código mostrado, un ejemplo del uso del método toString. Es opcional agregar la anotación @Override, la simplemente indica al compilador que estamos sobreescribiendo el método toString de la clase object.

Vemos que básicamente estamos concatenado los atributos de nuestra clase. Cabe mencionar que somos libres de modicar la cadena según nuestras necesidades, por ejemplo vemos que estamos indicando el nombre de la clase y posteriormente concatenamos cada uno de los atributos de nuestra clase, y al final regresamos una sola cadena que representa el estado actual de nuestro objeto. Sin embargo podemos hacer los cambios que deseemos en esta cadena, ya que precisamente la sobreescritura nos da la libertad de agregar el código que deseemos siempre y cuando cumplamos con los requisitos del método, que básicamente es regresar un String.

MÉTODO EQUALS

Ejemplo método equals en Java:

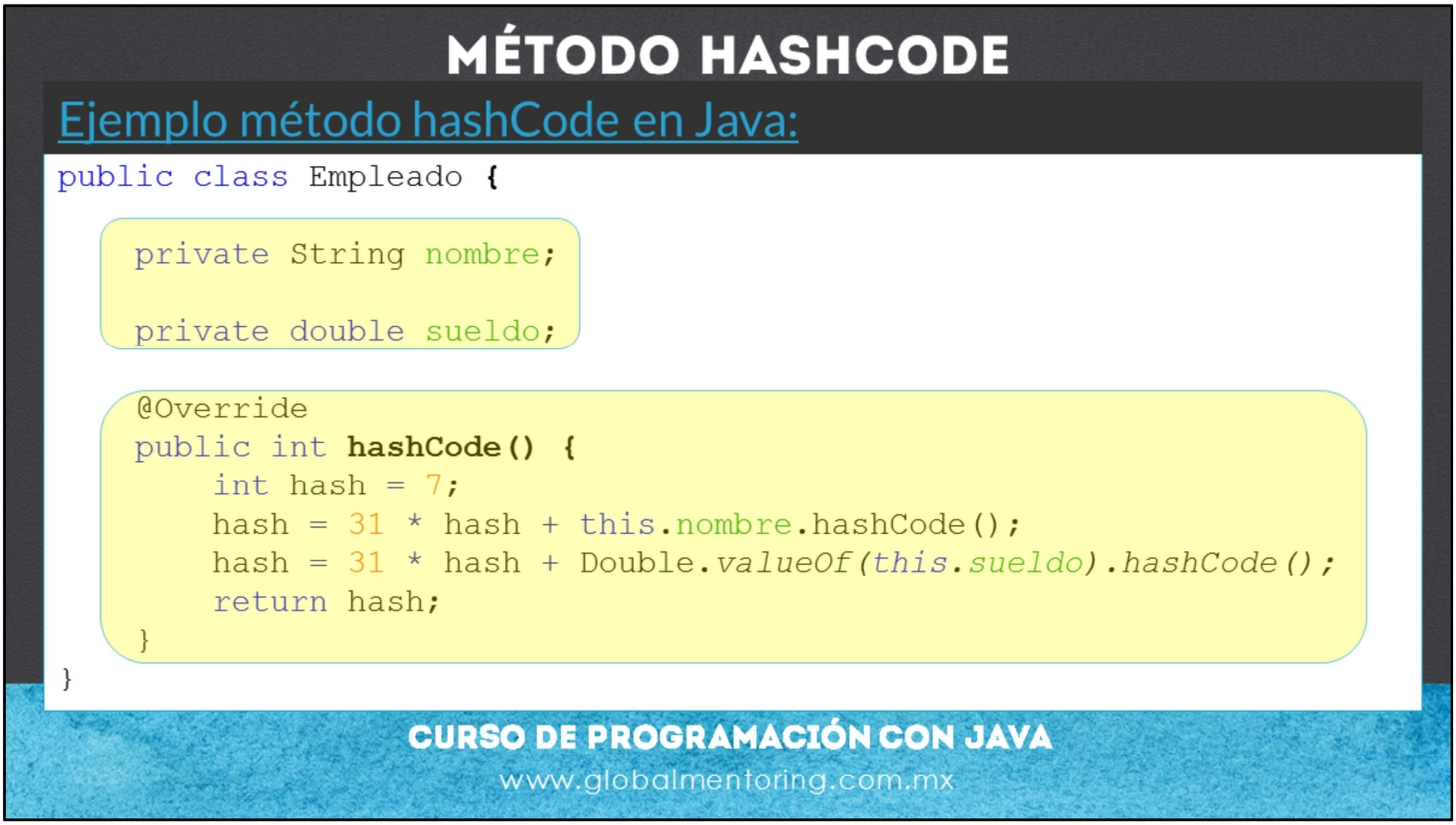
```
public class Empleado {

    protected String nombre;

    protected double sueldo;

    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (obj instanceof Empleado) {
            Empleado emp = (Empleado) obj;
            if (nombre.equals(emp.nombre) && Double.valueOf(sueldo).equals(emp.sueldo)) {
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
}
```

Otro de los métodos que sobreescribiremos con frecuencia en nuestras clases serán los métodos equals y hashCode. Estos métodos se utilizan para saber si dos objetos son iguales. Recordemos que si comparamos los objetos con el operador == nos comparará las referencias de los objetos (la ubicación en memoria). De igual manera si NO sobreescribimos el método equals heredado de la clase Object comparará la ubicación de memoria de los objetos, en lugar del contenido.

Esta comparación no nos sirve si lo que queremos es comparar el contenido del objeto, es decir, los valores de los atributos de nuestros objetos, y esto es lo que queremos comparar en la gran mayoría de las veces.

Podemos observar el código, en el cual estamos sobreescribiendo en método equals, este método básicamente necesitamos sobreescribirlo pero en muchos casos no haremos la llamada directamente nosotros, sino que serán otras clases las que comparen si nuestros objetos son iguales, por ejemplo métodos de ordenamiento, y esto aplica comúnmente cuando

Podemos observar que lo que se compara es cada uno de los atributos de la clase, primeramente revisando si el objeto recibido a comparar es del mismo tipo que la clase que estamos trabajando. Una vez que comparamos cada atributo entonces podemos garantizar que el contenido de los objetos es igual al objeto recibido, por lo que el método equals regresa true, de lo contrario el método regresa false.



Cuando definimos un objeto y redefinimos el método equals, debemos redefinir el método hashCode.

Si dos objetos son iguales (según equals), el valor retornado por sus respectivos hashCode debe ser igual

Por defecto, hashCode devuelve un entero diferente para cada objeto. Su uso principal es la optimización de las colecciones basadas en Hashtables para el ordenamiento de sus elementos, este tipo de colecciones las estudiaremos más adelante.

Implementar estos métodos nos permite saber si dos objetos iguales, ya sea comparando el contenido de dos objetos por medio de los atributos (equals) o el número entero que representa al objeto mismo (hashCode). De esta manera garantizamos que dos objetos sean iguales. Entre más elaborado el método hashCode más se garantiza que no existan coincidencias entre los objetos a comparar. Cabe mencionar que la mayoría de los IDEs nos ayudan a generar los métodos que hemos visto en esta lección, tanto los métodos toString, equals y hashCode, sin embargo recomendamos su uso únicamente cuando ya es de nuestro conocimiento estos conceptos.

En Global Mentoring promovemos la Pasión por la Tecnología Java. Te invitamos a visitar nuestro sitio Web donde encontrarás cursos Java Online desde Niveles Básicos, Intermedios y Avanzados, y así te conviertas en un experto programador Java.

Además agregamos nuevos cursos para que continúes con tu preparación como programador Java profesional. A continuación te presentamos nuestro listado de cursos:

- Programación con Java
- Fundamentos de Java
- Programación con Java
- Java con JDBC
- HTML, CSS y JavaScript
- Servlets y JSP's
- Struts Framework
- Hibernate Framework
- Spring Framework
- JavaServer Faces
- Java EE (EJB, JPA y Web Services)
- JBoss Administration
- Android con Java
- HTML5 y CSS3

Datos de Contacto:

Sitio Web: www.globalmentoring.com.mx
Email: informes@globalmentoring.com.mx