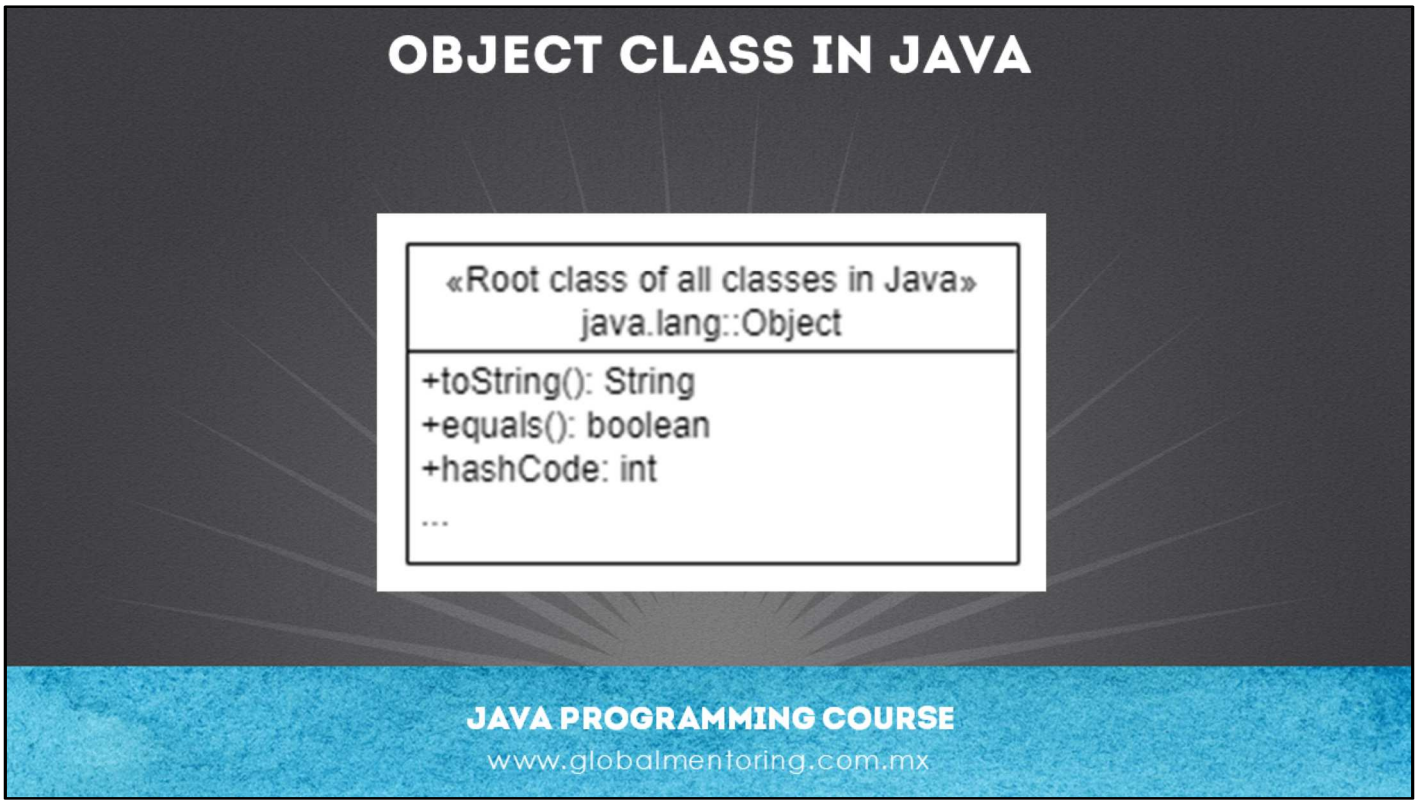

Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson.

We will study the subject of the Object class in Java, in addition to the overwriting of methods such as the toString, equals and hashcode method.

Are you ready? Come on!



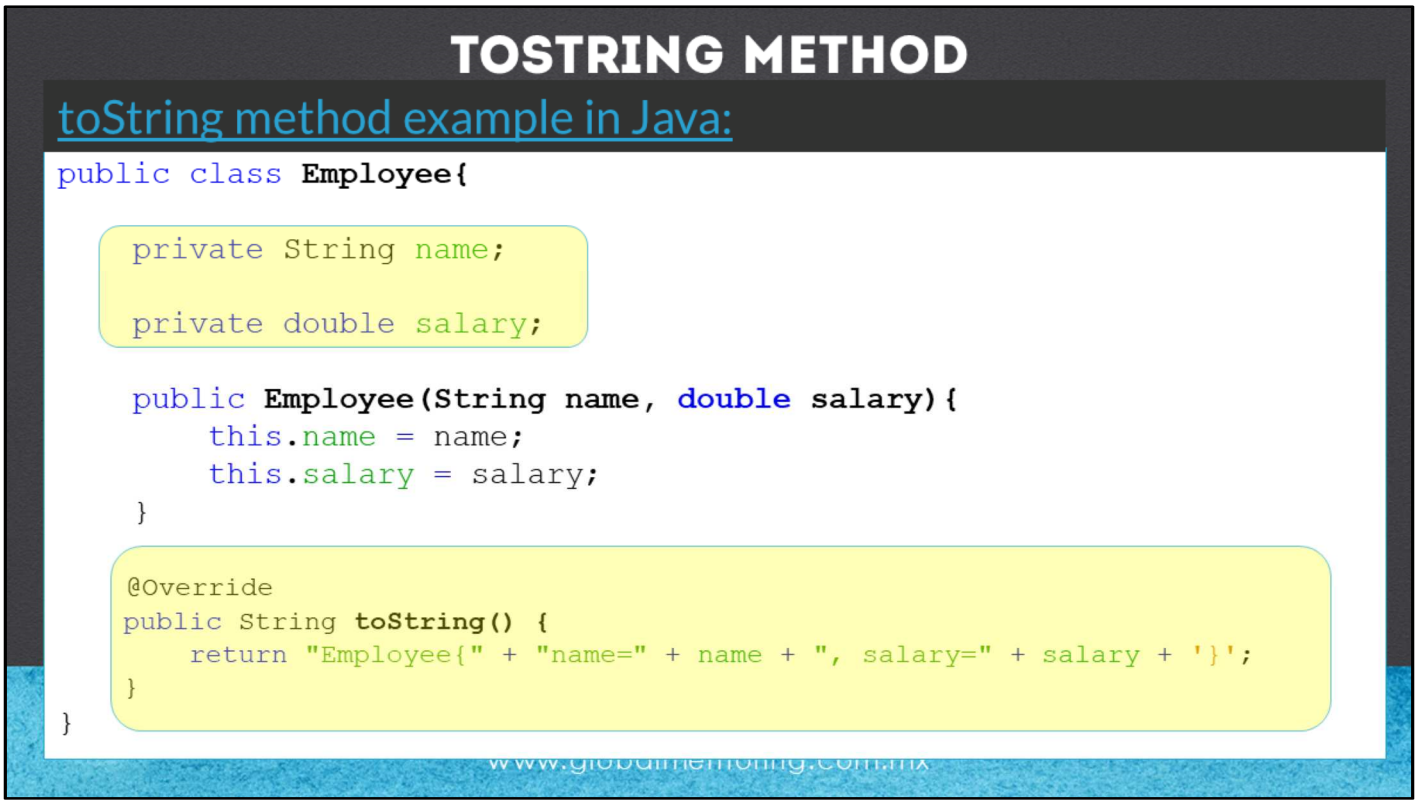

In Java, all classes inherit from the Object class.

The Object class is the root class of all classes in Java. The Object class is in the java.lang package, which is the main or core Java package, and for this reason there is no need to import the classes that belong to the Java package, thus simplifying the use of these classes.

And why is the Object class so important? Because ALL classes in Java directly or indirectly inherit from the Object class, in this way the Java language can be sure that it will share at least this in common among all Java classes, and this allows to ensure that certain functionality or features are shared among all classes in Java.

There are several very important methods that we will be constantly using from the Object class, and rather, we will be overwriting these legacy methods in our own classes. Some of these methods are the toString, equals and hashCode method.

Let's see in more detail each of them.




All objects in Java inherit from the Object class, directly if they do not indicate the word extends in the definition of their class, and indirectly if they inherit from another class, due to the class hierarchy.

Therefore, it inherits the methods mentioned above. One of those legacy methods is the toString method. This method helps us to show a text representation of our classes, and when overwriting it, we can easily send print or show the status of an object.

If we do not overwrite the toString method in our class, we will obtain a result like: MyClass@12ba40c3, which is composed of the name of the class, followed by the hexadecimal address where the object is located in memory. However this does not tell us much about the state of our method. By state we refer to the current values of each attribute of our class.

Due to this, with the toString method we have the opportunity to show in a chain, the state of our object, simply concatenating each one of the values that we are interested in showing. We will almost always use all the attributes, but we must be careful in the most complex cases, in which some attributes are other objects, and sometimes we can generate recursion problems or circular calls with other objects.

We can see in the code shown, an example of the use of the toString method. It is optional to add the annotation @Override, it simply tells the compiler that we are overwriting the toString method of the object class.

We see that basically we are concatenated the attributes of our class. It is worth mentioning that we are free to modify the chain according to our needs, for example we see that we are indicating the name of the class and then we concatenate each of the attributes of our class, and in the end we return a single chain that represents the current state of our object. However we can make the changes that we want in this chain, since precisely the overwriting gives us the freedom to add the code we want as long as we comply with the requirements of the method, which is basically to return a String.

# EQUALS METHOD

## equals method example in Java:

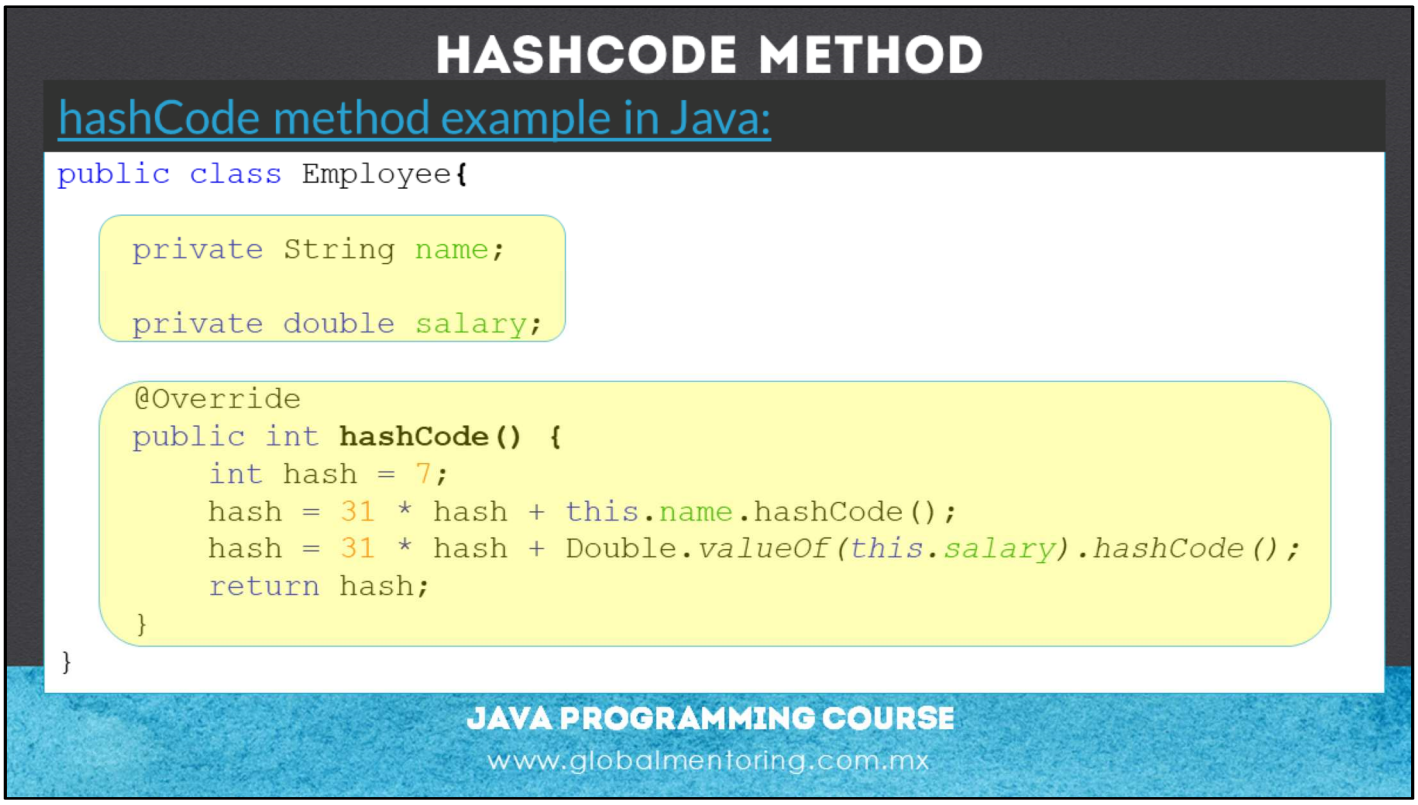
```
public class Employee{

    protected String name;

    protected double salary;

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

Another method that we will frequently overwrite in our classes will be the equals and hashCode methods. These methods are used to know if two objects are the same. Recall that if we compare the objects with the operator == we will compare the references of the objects (the location in memory). Likewise, if we do NOT overwrite the equals method inherited from the Object class, it will compare the memory location of the objects, instead of the content.

This comparison does not help us if we want to compare the content of the object, that is, the values of the attributes of our objects, and this is what we want to compare most of the time.

We can observe the code, in which we are overwriting in equals method, this method basically we need to overwrite it but in many cases we will not make the call directly to ourselves, but it will be other classes that compare if our objects are equal, for example ordering methods, and this applies commonly when

We can observe that what is compared is each of the attributes of the class, first checking if the object received to be compared is of the same type as the class that we are working on. Once we compare each attribute then we can guarantee that the content of the objects is equal to the received object, so the equals method returns true, otherwise the method returns false.




When we define an object and redefine the equals method, we must redefine the hashCode method.

If two objects are equal (according to equals), the value returned by their respective hashCode must be equal

By default, hashCode returns a different integer for each object. Its main use is the optimization of collections based on Hashtables for the ordering of its elements, this type of collections will be studied later.

Implementing these methods allows us to know if two objects are equal, either by comparing the content of two objects by means of the attributes (equals) or the integer number representing the object itself (hashCode). In this way we guarantee that two objects are equal. The more elaborated the hashCode method the more it guarantees that there are no matches between the objects to be compared. It is worth mentioning that most IDEs help us generate the methods we have seen in this lesson, both the toString, equals and hashCode methods, however we recommend using them only when these concepts are already known to us.