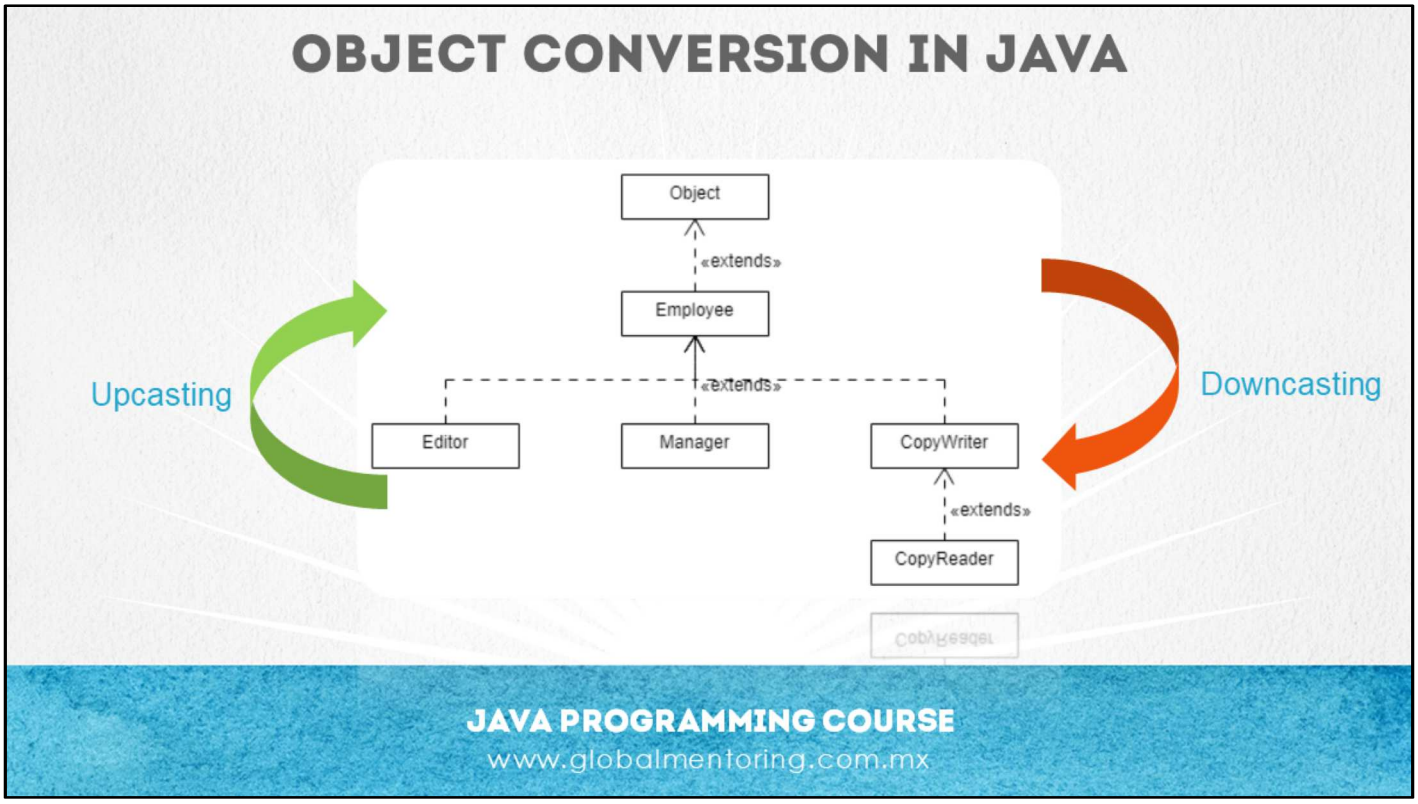
Hello, Ubaldo Acosta greets you. Welcome again to this Java Programming Course.

In this lesson, we will review the topic of object conversion in Java.

Let's start immediately.



We know that an object created in heap memory will always be of the same type, so its type will never change.

However, a variable of a certain type can point to different types, as long as there is a relationship between them. In the figure we can see that a variable of type Employee, can store references of objects of type Employee, or any subclass, such as Manager, CopyWriter, CopyReader, etc.

There are times when it is necessary to convert the type of variable that points to an object, in order to execute certain code or generalize our code and thus support multiple operations for multiple types of data from the same generalized function. This allows us to reuse code, among several other benefits of object-oriented programming and in general of good programming practices.

If we make a conversion of a more specific type (lower in the class hierarchy) to a higher type, it is known as upcasting, that is, conversion upwards, and if we convert from a higher type to a lower type we know as downcasting.

This topic will be used in Java in many cases, so it is important to have it very clear. The upcasting is done automatically, and there is no need to write some code explicitly, because if it is a superior type, the conversion is performed automatically, this is very similar to the primitive types in Java and the conversion between they.

On the other hand, the conversion down or downcasting is not done automatically, and it is necessary to specify which is the type we want to convert, and in fact in case of a wrong conversion will be thrown an exception of type ClassCastException. Later we will see the subject of exceptions.

To perform a downcasting we only need to specify in parentheses the type to which we want to convert our data type, and with that we will have a data type that consequently already includes all the characteristics of the more specific type that has just been converted. Let's look at an example of this.

OBJECT CONVERSION IN JAVA

Object conversion example in Java:

```
1  public class ObjectConversionExample {
2
3      public static void main(String[] args) {
4          //Assign a reference of a lower hierarchy object
5          Employee employee = new CopyWriter("John", 1500, WritingType.CLASSIC);
6
7          employee.getWritingTypeInText();//This is not possible, not accessible
8
9          //Object conversion - Downcasting
10         CopyWriter writer = (CopyWriter) employee;
11
12         // Finally we can access the methods of the CopyWriter class
13         result = writer.getWritingTypeInText();
14
15         result = ((CopyWriter)employee).getWritingTypeInText();
16
17         System.out.println("WritingType result:" + result);
18     }
19 }
```

We can see in the code an example of conversion, this example will be developed in detail, however we will mention several issues so we can see how the conversion of objects works.

The objective of converting objects is to be able to access the methods of each object as we need. For example, in line 5, we are creating a type object, remember that this object never changes its type, it will always be the same. This reference is assigned to a variable of type Employee, that is, to a class of greater Hierarchy according to our class diagram of the previous sheet. This is known as upcasting, so it does not require any special syntax to make an upward conversion in the class hierarchy. The only thing that happens is that it is no longer possible to access the methods created in the CopyWriter class, but only the methods of the Employee class are available. Therefore, if we wanted to send a method of the CopyWriter class directly, it would not be possible since the variable we are using is of a higher type, and can not directly access the methods of the lowest hierarchy class.

This is where the topic of object conversion comes in, since we can access the methods of the CopyWriter class, however we need to convert the variable of type Employee. In line 10 we can see this conversion, which is known as downcasting, the syntax is very simple, it is only to put in parentheses the type of data we want to convert, and then the variable to convert, but there must be a relationship in the hierarchy of classes of the types of data that we want to convert. This is similar to the conversion of primitive types, but with the advantages of object handling.

Another way to do this, is to save the variable, and convert into the same line of code, this can be seen on line 15. This type of conversion is very common, so we must understand that there is no need to create a variable and make the assignment so that we can execute the conversion of objects, it is certainly a more complex syntax, but once we get used to it, it will be easy to detect what happens.

Basically, the online conversion begins with the same syntax that is to write the desired type in parentheses and then the variable to be converted. The next thing is to wrap these two elements in parentheses, and with this we will practically have access as if it were the variable already converted to the desired type, with access to the methods and attributes of the lowest hierarchy class.

It is worth mentioning that if it is an upcasting, instead of downcasting, there is no need to specify the type of data to convert, since the compiler will detect that the class is of higher hierarchy and a conversion will be made automatically.

Once we have done the conversion down (line 10), we can use any of the methods of the lower hierarchy type but in more detail, for example let's suppose that the CopyWriter class has defined a method called getWritingTypeInText (), which will indicate us if the writer writes by hand (classic) or by computer. So we can already access this method, which is not accessible from the class used, but from the CopyWriter class.

In this way making a data conversion it is possible to access the methods and / or attributes of the lower hierarchy classes and thus recover the functionality that at the time of using more generic or higher hierarchy types is lost. Let's see an example of object conversion.