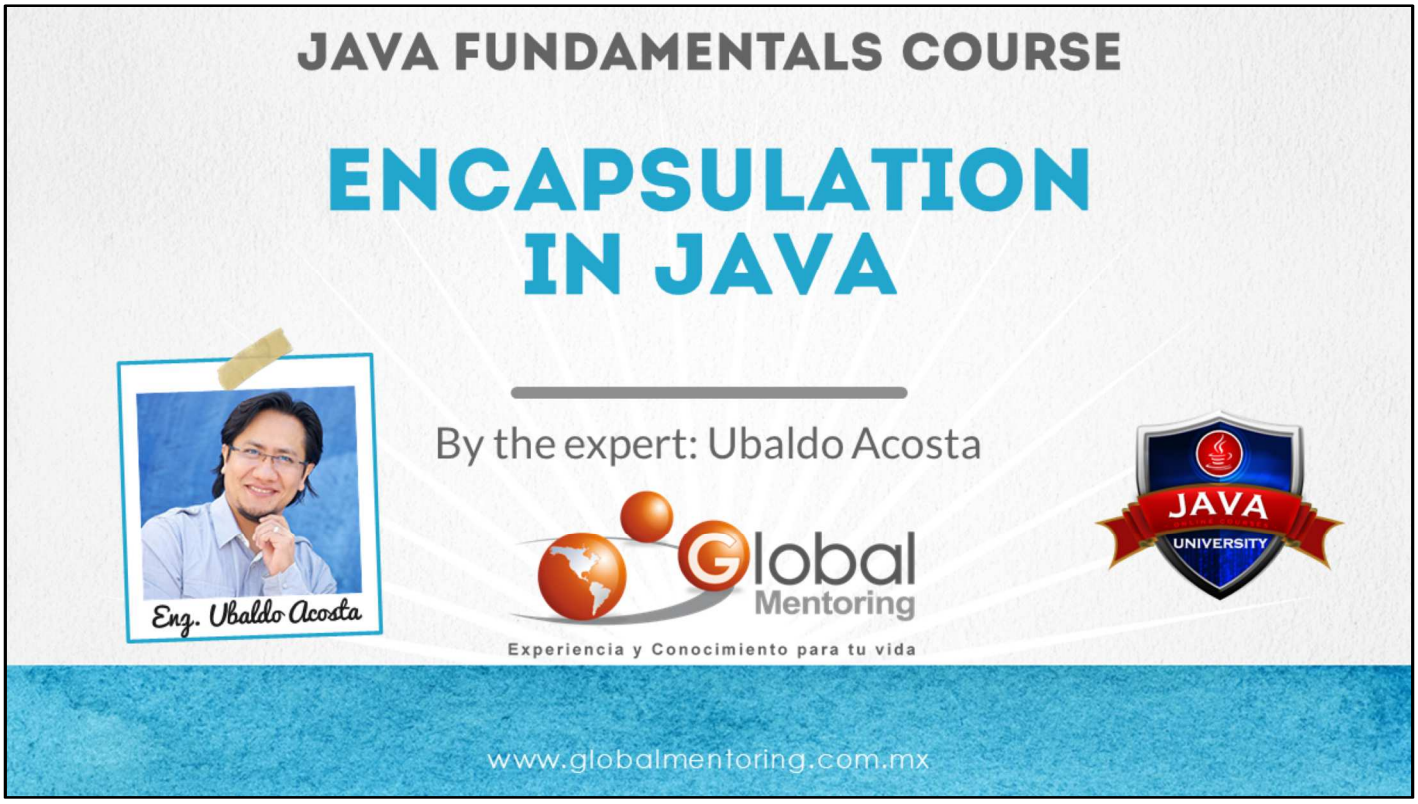
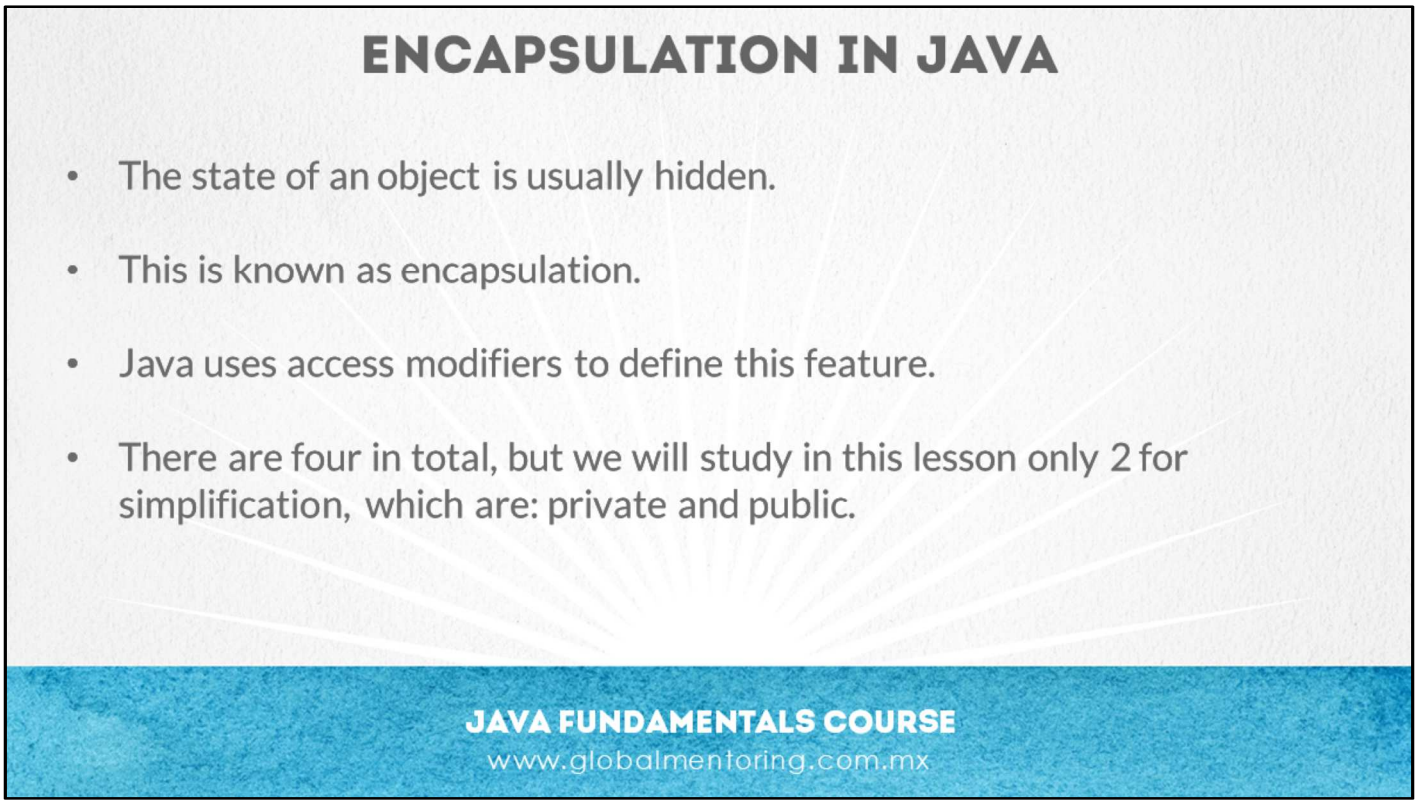


Hello, Ubaldo Acosta greets you. Welcome or welcome again. I hope you're ready to start with this lesson.

We are going to study the topic of Encapsulation in Java.

© All right reserved Global Mentoring



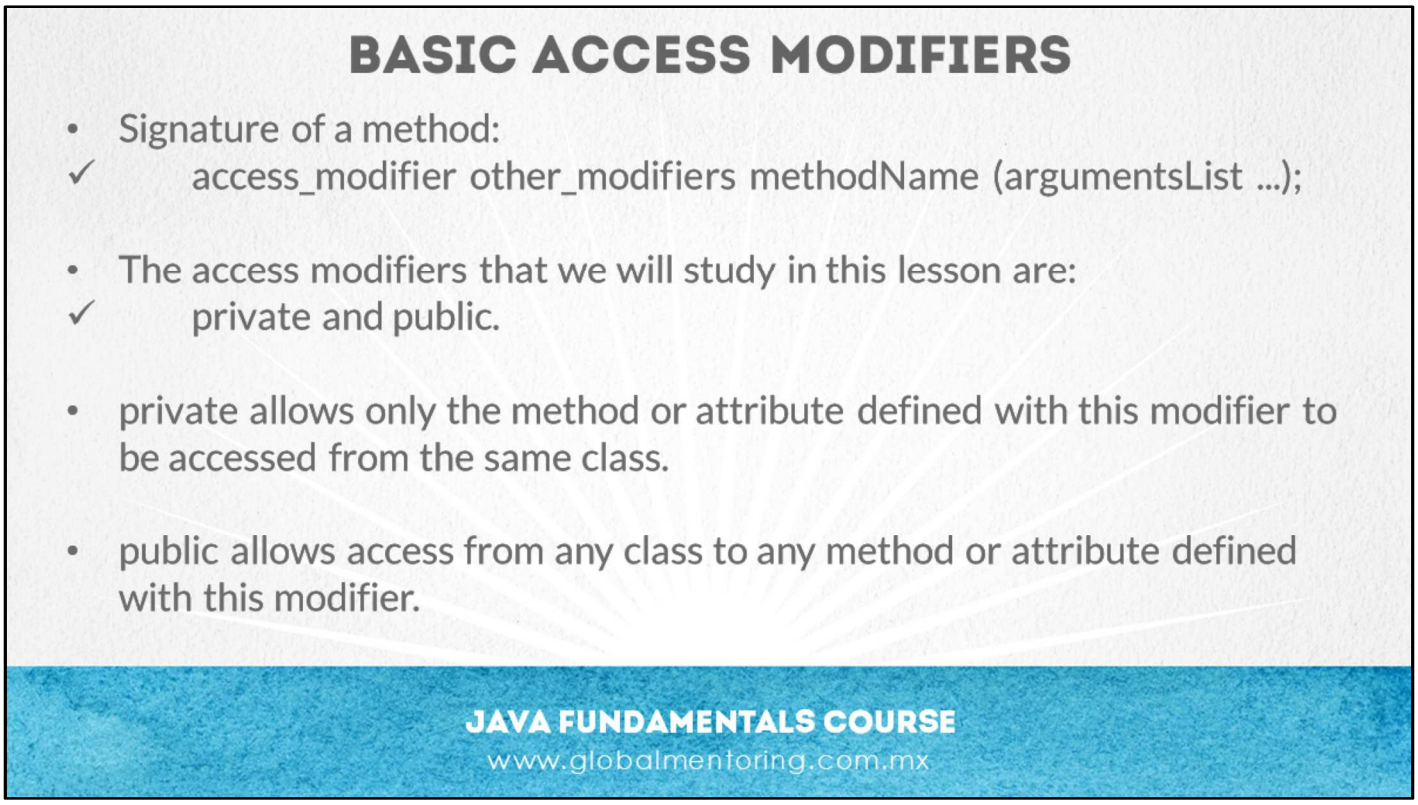
One of the most important characteristics of object-oriented programming is encapsulation.

This feature allows us to isolate the data of our objects from the access of other external objects, and in this way restrict the direct access to the attributes or methods that we do not want to allow, since the state of an object is generally hidden. We can understand the state of an object as the current values of each of the attributes of the object, and any change in these values changes the internal state of the object.

Being one of the most important principles, the encapsulation is achieved through the so-called access modifiers. There are four access modifiers in Java, which are: private, package or default, protected and public.

In this lesson we will study only the private and public modifiers, because for the level of this course it is enough to apply the concept of encapsulation, and in later courses we will see in detail how to implement each of these access modifiers.

© All right reserved Global Mentoring



As we have said, we will start with the study of the modifiers: **private** and **public**.

These modifiers apply in several parts of a class, however in this lesson we will study how they apply in two places, in the attributes of a class, as well as in the methods of a class.

The private modifier when adding it to an attribute or method of a class basically what we are indicating is that it will only be possible to access the attribute or method from the same class and not from any other, hence the name of private.

In contrast, the public modifier indicates that it is possible to access this attribute or method from any other class, hence the name of the public.

Next, we will see an example of how to apply these access modifiers to an attribute or method of a Java class in order to apply the concept of encapsulation.

© All right reserved Global Mentoring

EJEMPLO DE LOS MODIFICADORES public Y private

Ejemplo uso de public y private para lograr encapsulamiento:

```
1  public class EncapsulationTest{//Class1
2
3      public static void main(String[] args) {
4          Person p1 = new Person("John");
5          //Send an error,
6          //cannot access a private attribute from another class
7          p1.name = "Peter";
8          //It is posibble to access a public method from another class
9          p1.getName();
10     }
11 }
12 ----------------------------------------------------------------------
13 class Person {//Class2
14
15     private String name; //Private attribute
16
17     public Person(String name) { //public Constructor
18         this.name = name;
19     }
20
21     public String getName() { //public method
22         return name;
23     }
24 }
```

www.globalmentoring.com.mx

We can see in the code shown the use of private and public. Let's see some changes in the Person class.

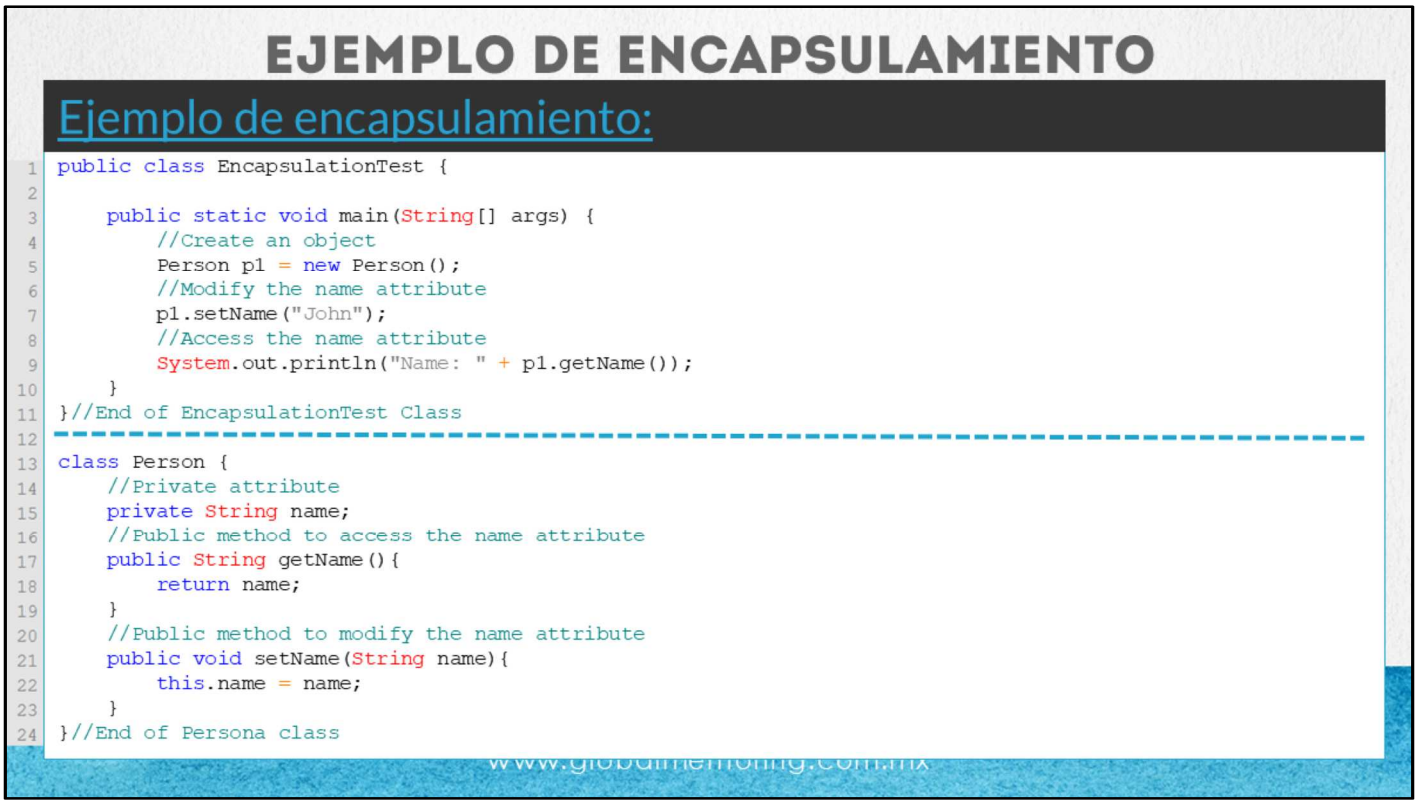
On line 15 the attribute name has been added the private access modifier, so that when we create a Person type object (line 4), we can not directly access to modify the value of the name attribute, since it will mark a compilation error (line 7) (Error: name has private access in Person).

If we want to read the value of this attribute, we create a method for it, and add the public access modifier so that any other class can access this value. Therefore, we create the method getName (line 21) and add the public access modifier, later in the test class access this method by means of the variable of type Person called p1 (line 9).

In this way what we are doing is modifying our Person class so that now the attribute is not accessible neither for reading nor for writing directly, and we have the need to access it by means of public methods that we create for each attribute of our class.

However, this is an informal way to apply encapsulation in Java, let's see how to apply it more formally in the next slide.

© All right reserved Global Mentoring



In the code we can see a more formal code for the concept of encapsulation, that is, we have converted our attributes to private (line 15), and for each attribute we have added two methods, one to access the attribute (getName - line 17), and another to modify the attribute (setName - line 21).

These methods receive a special name according to the function they perform. The methods of type **mutator** modify the value of the attribute of a class and generally the prefix **set** is used and later the name of the attribute of the class respecting the notation of lowercase / uppercase.

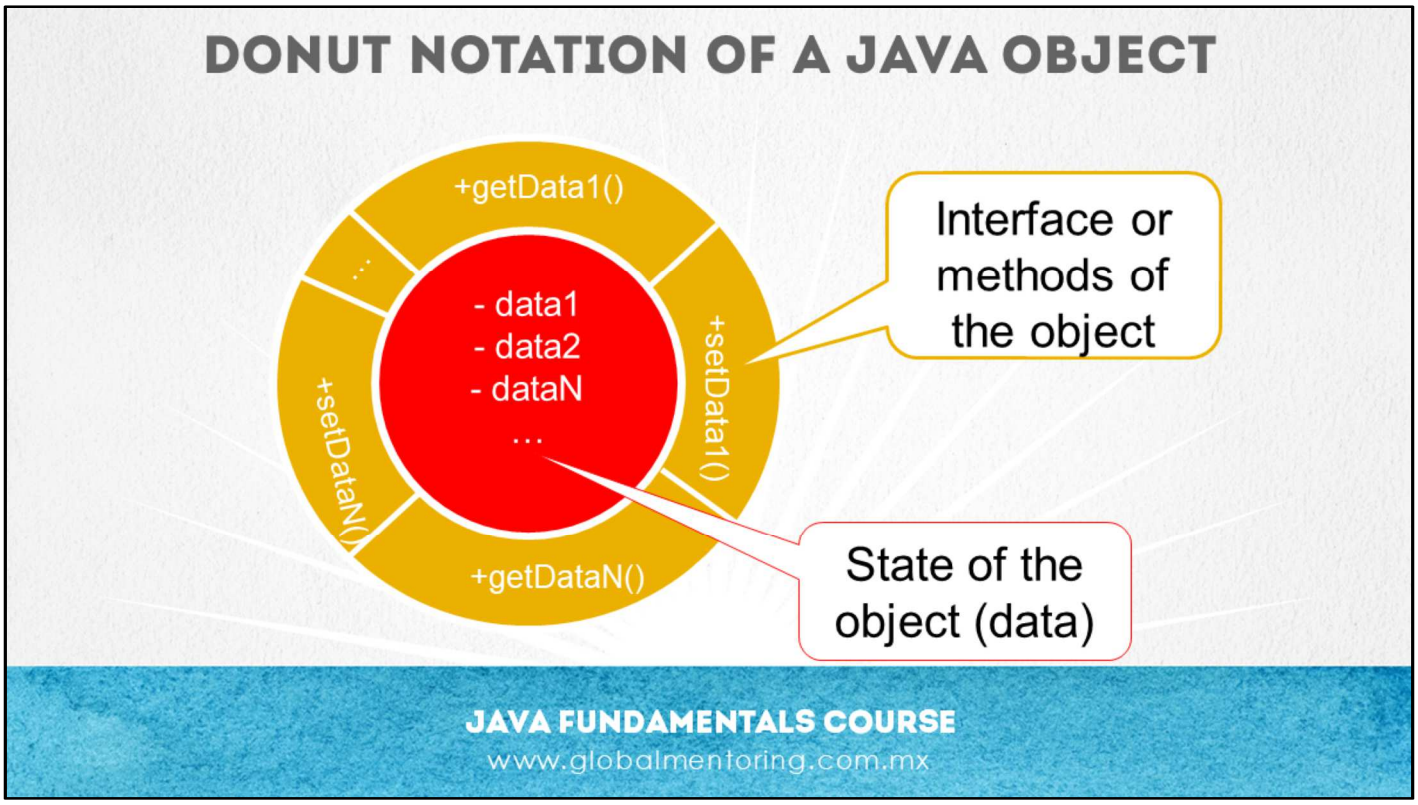
The methods of type **accessors** allow to recover the value of the attribute of a class and the prefix is used if it **is** of type boolean and **get** for the other types of data, later it carries the name of the attribute in the same way respecting the capitalization notation of lowercase / uppercase in Java.

In the code shown in the example we can observe the use of these mutators and accessors. As we have said, the goal is to generate an interface that allows the state of the Java object (data or attributes) to be encapsulated and that it is not possible to directly access the state of the object and modify it, but rather through the mutators and if we want to know the state of the object through of the accessors.

In this example we must understand that the test class is the public class called EncapsulationTest, and that the class in which the concept of encapsulation was applied was about the Person class. That is to say that they are different classes, and therefore the code that we have in the main method (lines 4 to 9) is code that is outside the Person class and there lies the concept of encapsulation, that is, to avoid classes that are external to the Person class can access the attributes of the Person class directly, regardless of whether a class is declared in the same file, they are different classes.

Next we will see another way to conceptualize the encapsulation.

© All right reserved Global Mentoring



As we can see in the picture, the idea of encapsulation of object-oriented programming is to avoid direct access and manipulation of other classes to the data of an object, by data we refer to any attribute of the class in question, that is, the state of the object.

To avoid direct access of other classes to the data of the class that we want to encapsulate, the first thing we must do, as we have said, is to add the private access modifier to our class attributes, in this way only the methods of the same class they are the ones that will be able to access (read / get) and manipulate (modify / set) the data of the class that we are creating.

On the other hand, how can an external class access the data of our encapsulated objects? The answer is to create public methods for each private attribute, which can both modify and recover the value of our data or class attributes. This is known as the interface of the object, because it is through these methods that we will be communicating with the created object and thus be able to read the status of each data and / or modify its data. Normally, two public methods are created for each private attribute, a set method and a get method.

The methods that modify the data are known as mutators and carry the prefix set (place) followed by the name of the variable in upper case / lowercase notation. On the other hand, the methods that read the information of the data are known as accessors and have the prefix of get (get) followed by the name of the variable, except in cases where the variable is of type boolean, then the prefix in place to get will be is (is).

Now, why avoid the direct modification of the state of an object in Java, and in general in object-oriented programming? What we are looking for is that we have control over the information and status of our objects, so that if we want to modify the data1, we can apply a validation on the information we want to place in the data1, and this is an appropriate value, as it can be the restriction of receiving only numbers, then, before making the update of the value of the data1 we can verify that indeed the value that is going to be placed is a numerical value, this only for putting a simple example, but the idea is that other objects can not directly modify the state of our objects if we do not wish, and in this way the concept of encapsulation in Java allows us to achieve this, hiding any validation but applying it at all times, respecting at all times the rules that apply to the state of the object.

In the sheet we can see that the attributes have a symbol of - (minus), and that the methods have a symbol of + (more), this is part of the notation of this diagram of donut or objects, and what it indicates is that the sign of - is the private access modifier, and the sign of + is the public access modifier, in this way we can graphically observe the private and public elements in our Java object.

This is just a summary of what we have seen above, but it is another way in which we can understand the concept of encapsulation. We will put this concept into practice in the following exercise.

© All right reserved Global Mentoring


ONLINE COURSE
JAVA
FUNDAMENTALS
Author: Ubaldo Acosta
Global Mentoring
JAVA
UNIVERSITY
JAVA FUNDAMENTALS COURSE
www.globalmentoring.com.mx