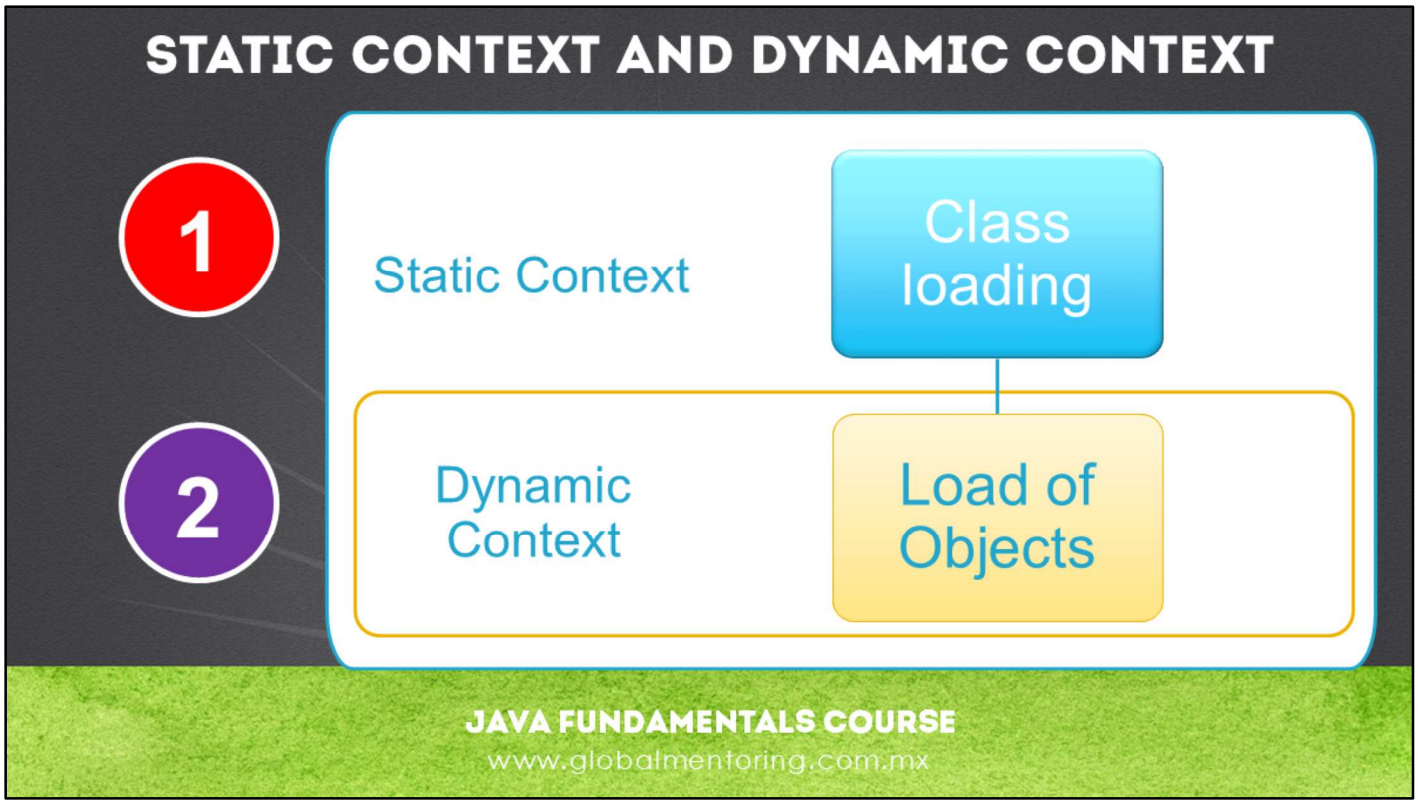
Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson .

We are going to study the subject of the static context.

Are you ready? Come on!



Let's start with this lesson by differentiating the static context from the dynamic context.

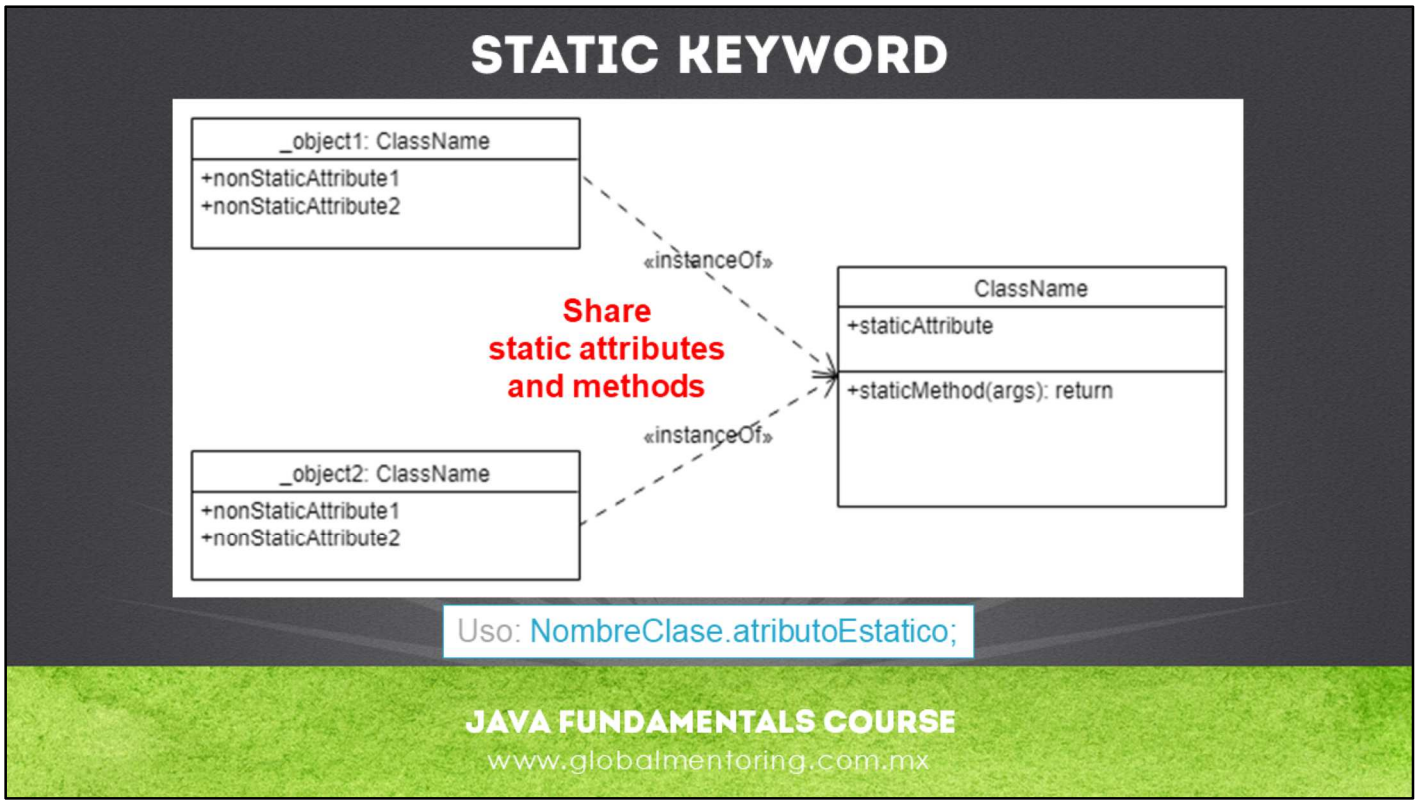
To start creating objects of a class, you must first load the class in memory by means of what is known as ClassLoader. Here the concept of static context (1) enters for the first time, and it can be used until the class is removed from memory, which normally occurs when the Java virtual machine process is stopped, that is, the class is loaded throughout the process of executing our application.

Once the class is already loaded into memory, it is possible to start creating objects of that class (2). This is known as the dynamic context, since at this time it is already possible to create objects and start interacting between them.

In short, the static context is loaded at first and therefore the classes that are to be used, and subsequently the dynamic context is created and therefore the objects of the classes that have been loaded in memory can already be created. As we can see in the picture, the static context has a longer duration than the dynamic context, and in fact the static context includes the dynamic context, but not the other way around.

The **this** operator only has use when an object of a class has been created, therefore it is not possible to use it in the static context (for example inside the main method, because is marked as static)

The static context can be enabled in Java code by means of the static word, which we will explain next.

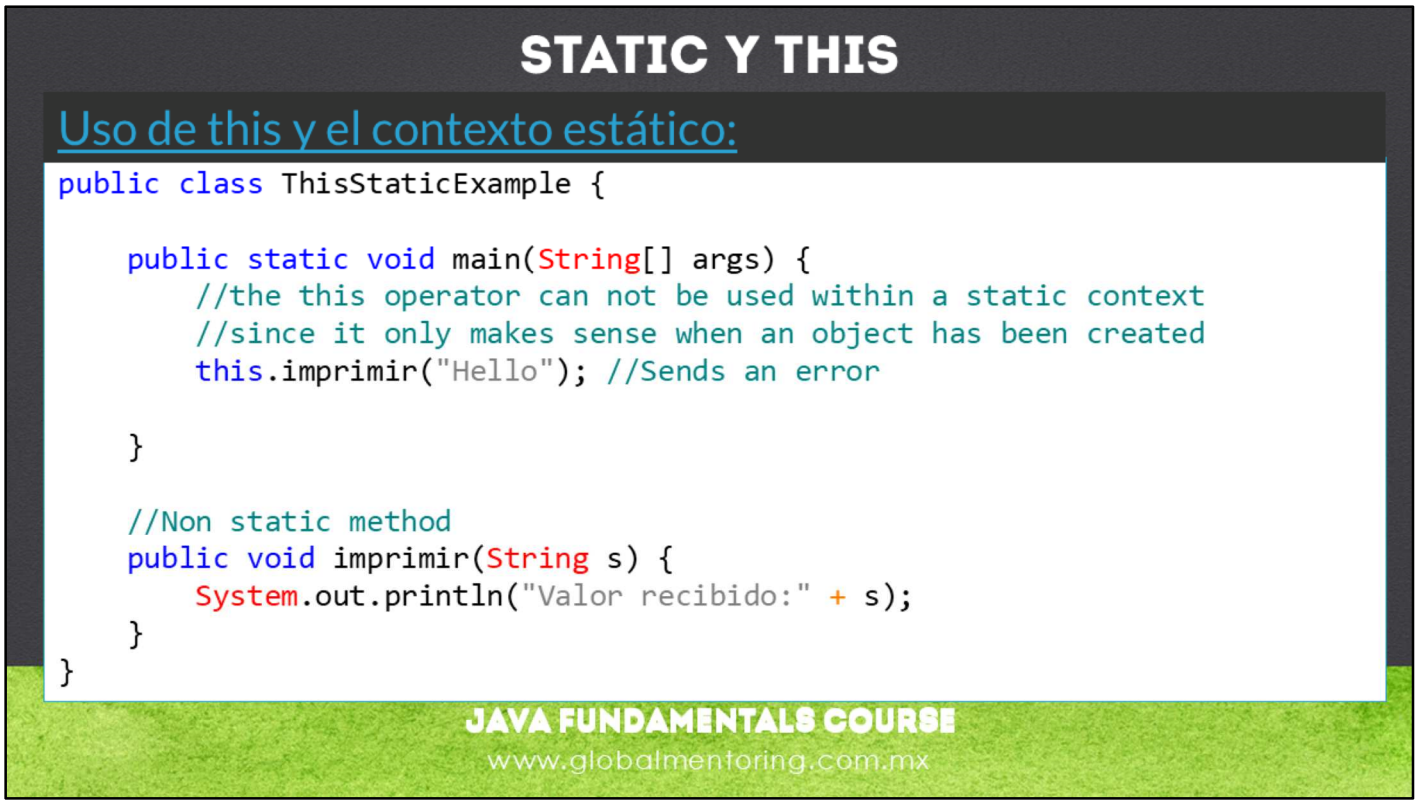


We can use the **static** keyword to interact with the static context. For example, if we define an attribute or a method as static, what we are indicating is that the attribute or method belongs to the **class** and not to the object.

For example, if we create an attribute without using the word static, which is how we normally define them, every time we create an object a variable associated with the object is created, but if we define the attribute as static, we are indicating that the attribute it is only created once, no matter how many objects are created, there will only be one variable which is associated with the class and not with the object. And if an object accesses the value of the static variable it will read the same value as the other objects, and if an object modifies the static value, all other objects will access the same value since this value is stored in the class and not in the objects of that class.

Using the word static it is possible to interact with the class without having to create an object of the same class. For example, we can access attributes and / or methods without having instantiated any object of the class we want to use. With the use of the static word either in the attribute or in the method that we want to access, we can directly use the attribute or method without generating an object of the class, only placing the name of the class, the dot operator, and finally the name of the static attribute or method defined in the class (e.g. ClassName.staticAttribute). Normally static attributes or methods if we want them to be accessed from other classes must contain the public access modifier so that they do not have any problem in being accessed.

So, attributes or methods marked with the word static are referred to as class members or class methods, since they belong to the class and not to the objects that are created from that class.



Once we have mentioned the static and the dynamic context, we can clarify the following.

The **this** operator is associated with the dynamic context, and because the static context loads the dynamic context first, the **this** operator does not work in the static context.

It is for this reason that in the main method or any other static method, it is not possible to use the **this** operator.

In short, a static method can access another static method, but not a non-static method, since the dynamic context is not yet created.

But, a dynamic method (not static) CAN access a static method, since this context has already been created.

Now, we can access a dynamic method from a static method as long as we create a new object. This is if we can access dynamic methods, but this is because once we create an object, for example within the main method, what we are doing is creating the dynamic context, and therefore it is already possible to access the methods of the object through it. In this way it is that an object can access both non-static or static methods. However, although a static method can be accessed by means of an object, it is correct to use the name of the class and not the name of the object.

EJEMPLO CÓDIGO ESTÁTICO

```
1  public class StaticExample {
2
3      public static void main(String[] args) {
4          Person p1 = new Person("John");
5          System.out.println("Person1: " + p1);
6
7          //We print the peopleCounter
8          System.out.println("# People:" + Person.getPeopleCounter());
9      }
10 }
11
12 class Person {
13
14     private String name;
15     private int idPerson;
16     private static int peopleCounter;
17
18     public Person(String name){
19         //Every time we create a Person object we increase the counter to obtain a new idPerson
20         peopleCounter++;
21         //we assign the new value to idPerson
22         idPerson = peopleCounter;
23         //We assign the received name
24         this.name = name;
25     }
26
27     public static int getPeopleCounter(){
28         return peopleCounter;
29     }
30 }
```

In the code shown, we can see an example of the use of the word static.

On one hand we declare a variable of type static called peopleCounter. By declaring this variable as static, what we are looking for is that every time we create an object of type Person, we will increase the value of this variable. We can achieve this by using the constructor of the class, since every time we create an object of type Person, the constructor of the class is called, and there we can increase this variable with each call made to the constructor (line 20).

Once this static variable has been increased, this value applies to all the objects, therefore each time that we increase this variable can be consulted by all the objects that we create and in this way it is that this variable does not start from scratch every time we create a new object, but it continues with the last value assigned and increases once more, this is because this variable belongs to the class and not to the objects.

Once we have the new value, we can then assign the new value to the attribute idPerson (line 22), and the value of idPerson since it is not static will be unique for each object, hence we can be sure that the value will be unique for each created object of type Person.

Another important section of this code is line 27, in which we declare the method getPeopleCounter(), with this method which is static, we can recover peopleCounter variable, to know how many objects of type Person have been created.

Now, in the main method, which is a static method, let's make use of the Person class. First we will create an object of type Person, and assign the value of John to the name attribute with the help of the constructor of the class.

From the moment we are using the word new, it means that we are already working with the dynamic context, this allows us to work with the dynamic context (objects) and the static context (classes).

Therefore, when creating the person object, the constructor of the class is responsible for assigning the idPersona with a unique value by increasing the peopleCounter variable, which, being static, allows the Person object count to be correct.

In this code we are omitting the implementation of the toString method for the Persona class, which must be added to observe the status of the person object printed on line 5.

Finally, once we have finished creating person-type objects, we can send to print the total number of Person objects that have been created, and this is where we can observe the notation that we have talked about, to call a static public method of a class is enough to put the name of the class, the dot operator and the name of the static method, that is to say that we should not use the object, in fact it is not good practice and if we insist on using an object type variable to call a static method will show us a warning, warning us that it is a bad practice.

With this we conclude the use of the static context.