JAVA FUNDAMENTALS COURSE

this KEYWORD IN JAVA

By the expert: Ubaldo Acosta

Global Mentoring

Experiencia y Conocimiento para tu vida

JAVA FUNDAMENTALS COURSE

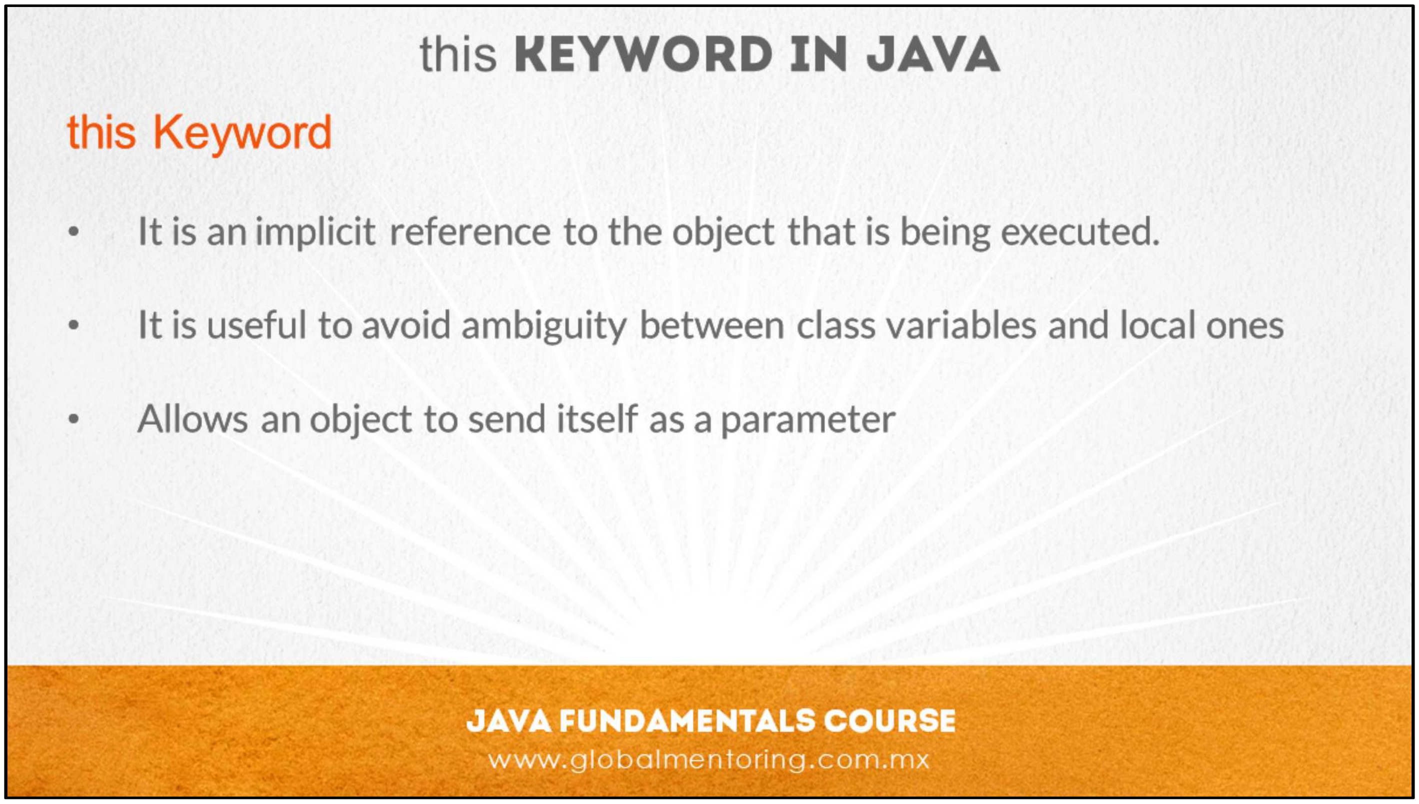
www.globalmentoring.com.mx

Eng. Ubaldo Acosta

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 subject of the this keyword in Java.

Are you ready? OK let's go!



this KEYWORD IN JAVA

this Keyword

- It is an implicit reference to the object that is being executed.
- It is useful to avoid ambiguity between class variables and local ones
- Allows an object to send itself as a parameter

JAVA FUNDAMENTALS COURSE
www.globalmentoring.com.mx

We have seen that the **this** keyword allows us to access the object that is currently running, however we are going to dedicate a few more minutes to this topic to comment some more details.

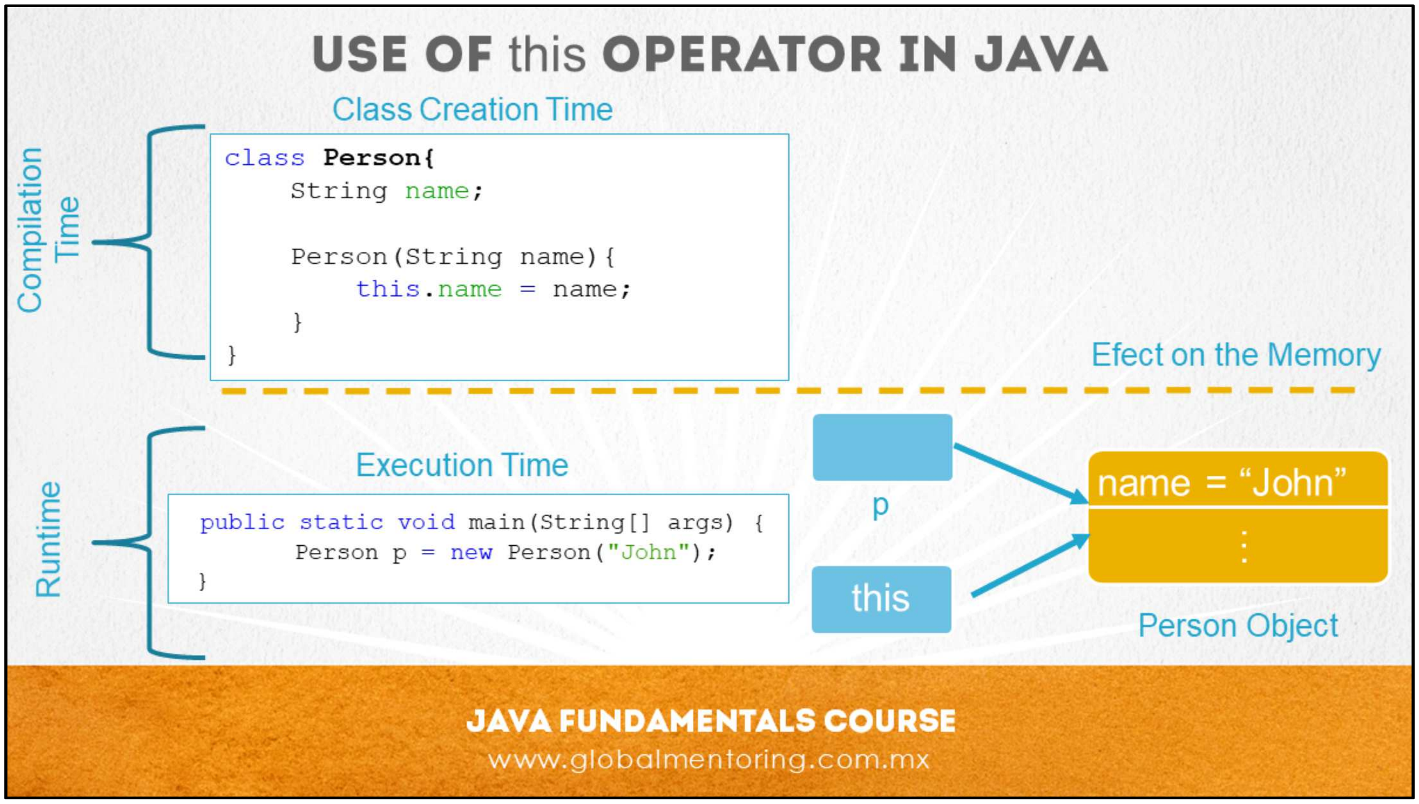
The **this** keyword is an *operator* and basically allows us to access the object that is running in any Java method, however this can be confusing at first.

We have two Java times as we have seen, the compile time, which is when we are creating our code, and the execution time that is when our program is running.

When creating our code, we must remember that no code is being executed, but that in the future it is precisely what will happen, especially when we are creating our classes. Therefore, the operator **this** has to be though in terms of execution of our code, that is, when an object of the class that we are coding is created.

The **this** operator is unique for each object, and will always correspond to the object that is being executed.

Another important thing to mention about the **this** operator is that this word can only be used within an object created from the Java class, later we will see what the static context means, but we can advance that the **this** operator can not be used in a static context, since in the static context the objects have not yet been created, and therefore the **this** operator has no use if the object to which it points has not been created.



USE OF this OPERATOR IN JAVA

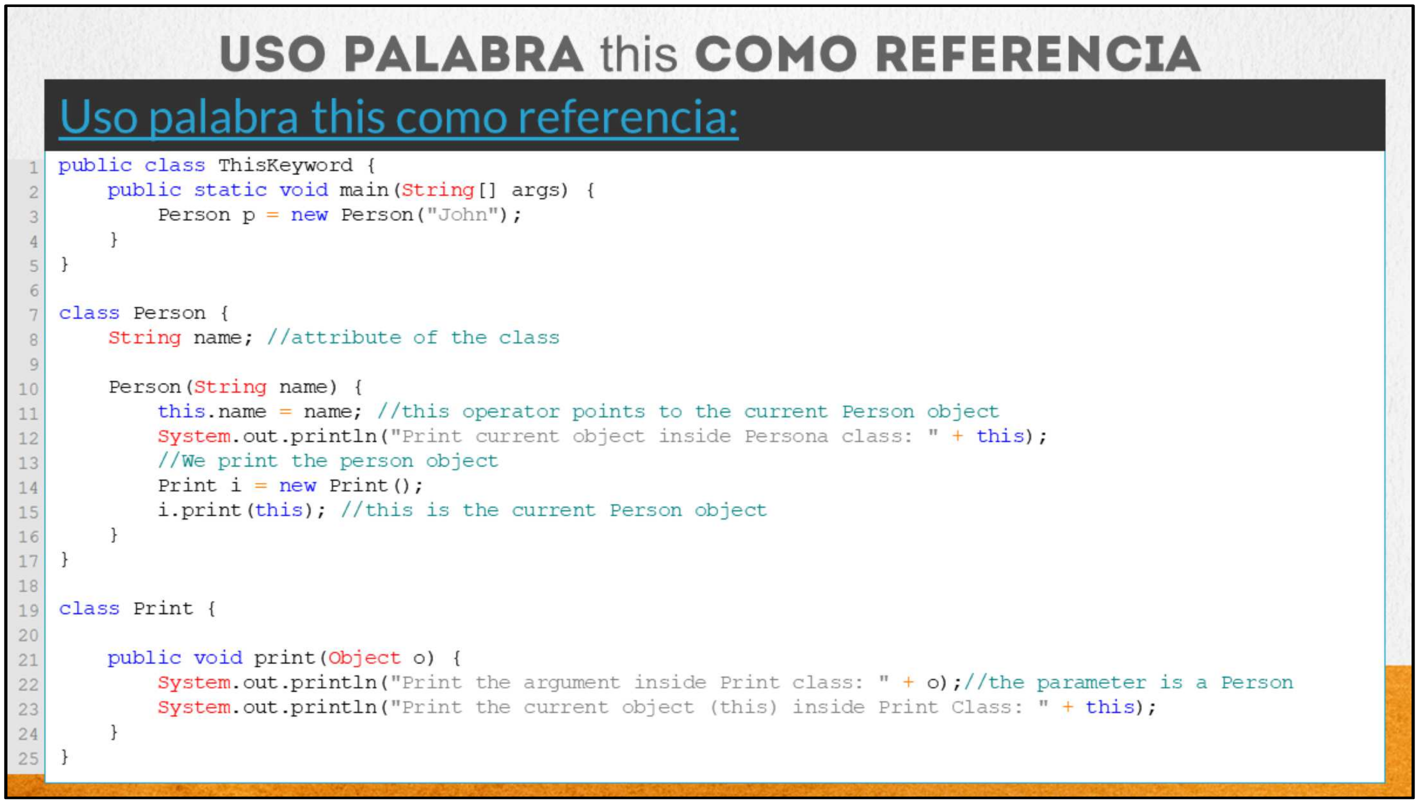
```
class Person{
    String name;

    Person(String name){
        this.name = name;
    }
}
```

```
public static void main(String[] args) {
        Person p = new Person("John");
}
```

Using the most common uses of the **this** operator, as we have already mentioned, is to avoid ambiguity between the attributes of the classes and the local variables or arguments of a method.

In the example, we can see how we are declaring a class called Person, which has a constructor that receives an attribute called name, which is identical to the name of the attribute of the class. To make the difference between both names, we can use the **this** operator which refers to the attribute of the class, in this way we can name the arguments of a method identically and differentiate the attributes of the class by means of the **this** operator.

In the figure, we can see how both the variable p and the **this** operator point to the same object at the time the class constructor is called. Once the constructor of the class finishes executing the **this** operator points to another object, and if a method of the Person class is again executed, the **this** operator will point to the created object, that is, the **this** operator is always pointing to the object that is currently executing.



USO PALABRA this COMO REFERENCIA

Uso palabra this como referencia:

```
 1 public class ThisKeyword {
 2     public static void main(String[] args) {
 3         Person p = new Person("John");
 4     }
 5 }
 6
 7 class Person {
 8     String name; //attribute of the class
 9
10     Person(String name) {
11         this.name = name; //this operator points to the current Person object
12         System.out.println("Print current object inside Persona class: " + this);
13         //We print the person object
14         Print i = new Print();
15         i.print(this); //this is the current Person object
16     }
17 }
18
19 class Print {
20
21     public void print(Object o) {
22         System.out.println("Print the argument inside Print class: " + o);//the parameter is a Person
23         System.out.println("Print the current object (this) inside Print Class: " + this);
24     }
25 }
```

In this example, we can observe in more detail the use of the **this** operator. We are creating several classes and several objects so that we can observe how the **this** operator is changing according to the current object that is being executed.

In line 3 we create an object of type Person. Up to this point the this pointer has not been created or pointed at any object. It is only until line 3 is executed that it creates the person type object and then this pointer is created and points to the newly created Person object.

We know that the immediate step in the creation of an object in Java is to call the constructor of the class, so the next step is executed line 10, in which we receive in the local variable of type name the value of John, but to differentiate between the argument and the attribute of the class we use the **this** operator for the first time, in this way we make a reference to the attribute of the class.

Later, we create a new object of the Print class, but it is not until the print method is executed that the **this** pointer will change from pointing to the Person object to the Print object. For that reason in line 15 we pass as parameter the current Person object using the **this** operator, and when line 21 is executed two important things happen, on the one hand the **this** operator stops pointing to the person object and the argument that receives the print method is of type Person since **this** on line 15 still pointed to the person object.

All classes in Java inherit from the Object class, and although we will see it in another topic, the Object class serves as a wildcard class to be able to receive any Java type as an argument, in this case the created Object variable stores the reference of the class Person previously created, and whose reference was sent on line 15 through the **this** pointer.

For that reason on line 22 an object of type Person is printed, and on line 23 an object of type Print is printed, since we remember that **this** is now pointing to the object that is currently executing, which is of type Print.

There are more things we could do to improve this code, however we need more lessons to be able to improve this code, for this reason we are going to leave this code as simple as possible, in order to understand how the this operator changes depending on the execution of our code.

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