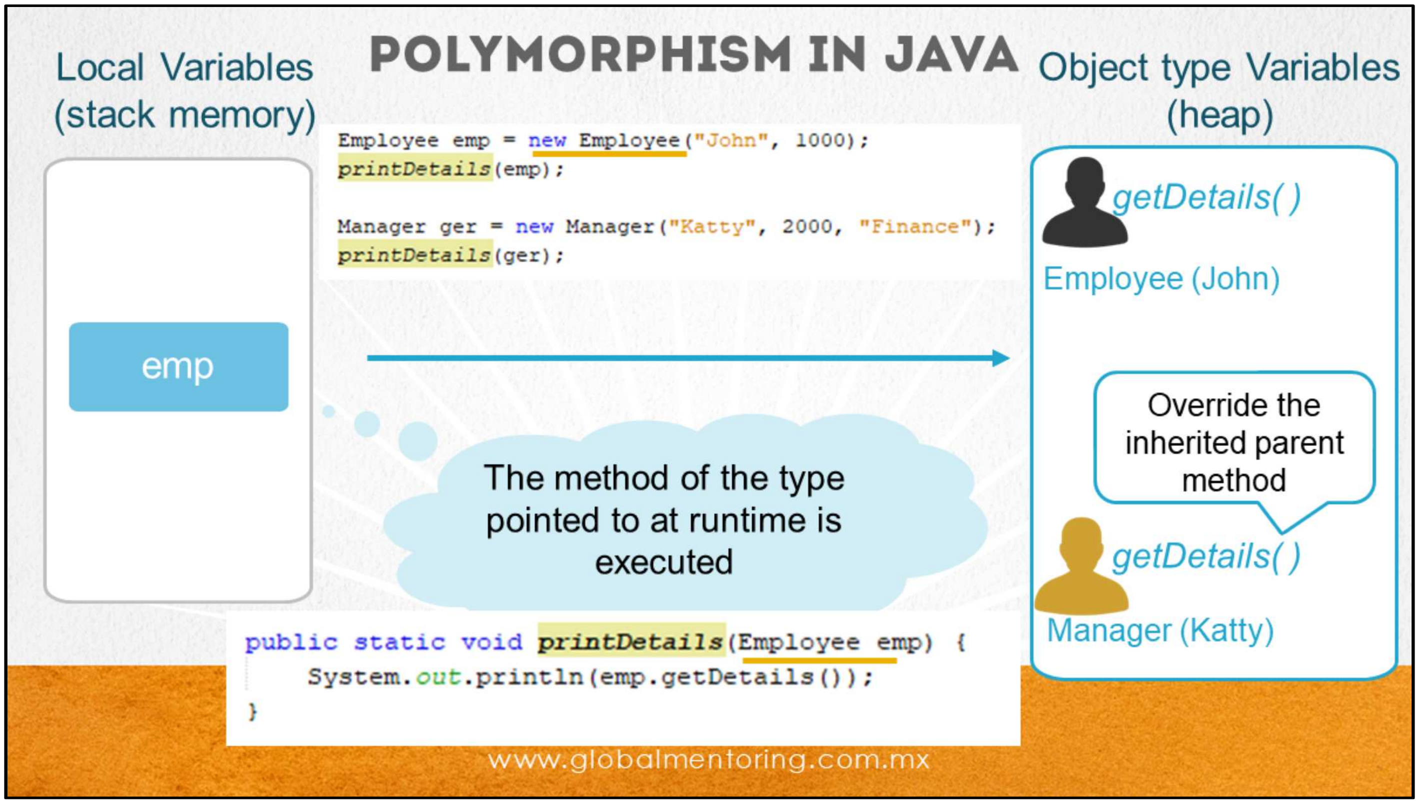
JAVA PROGRAMMING COURSE
POLYMORPHISM IN JAVA
By the expert: Ubaldo Acosta
Eng. Ubaldo Acosta
Global Mentoring
Experiencia y Conocimiento para tu vida
JAVA UNIVERSITY
JAVA PROGRAMMING COURSE
www.globalmentoring.com.mx

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

We are going to study the subject of polymorphism in Java.

Are you ready? OK let's go!


POLYMORPHISM IN JAVA
Local Variables (stack memory)
Object type Variables (heap)
emp
getDetails( )
Employee (John)
Override the inherited parent method
getDetails( )
Manager (Katty)
The method of the type pointed to at runtime is executed
www.globalmentoring.com.mx

```
Employee emp = new Employee("John", 1000);
printDetails(emp);

Manager ger = new Manager("Katty", 2000, "Finance");
printDetails(ger);
```

```
public static void printDetails(Employee emp) {
    System.out.println(emp.getDetails());
}
```

Polymorphism is the ability to execute syntactically equal methods in different types. Let's review this concept of object-oriented programming with the following example.

An object (created in the heap memory) only has one form and one type and this never changes during the whole life of the created object. However, a variable of one type can refer to objects of different types, as long as there is a relationship between these types, such as an inheritance relationship (Parent Class or Child Class).

A type variable of a parent class can store references of child classes or of the same type of the parent class, and send the methods it has in common using polymorphism, that is, it executes the methods of the child class, with the variable of type of the parent class. This is precisely the concept of polymorphism.

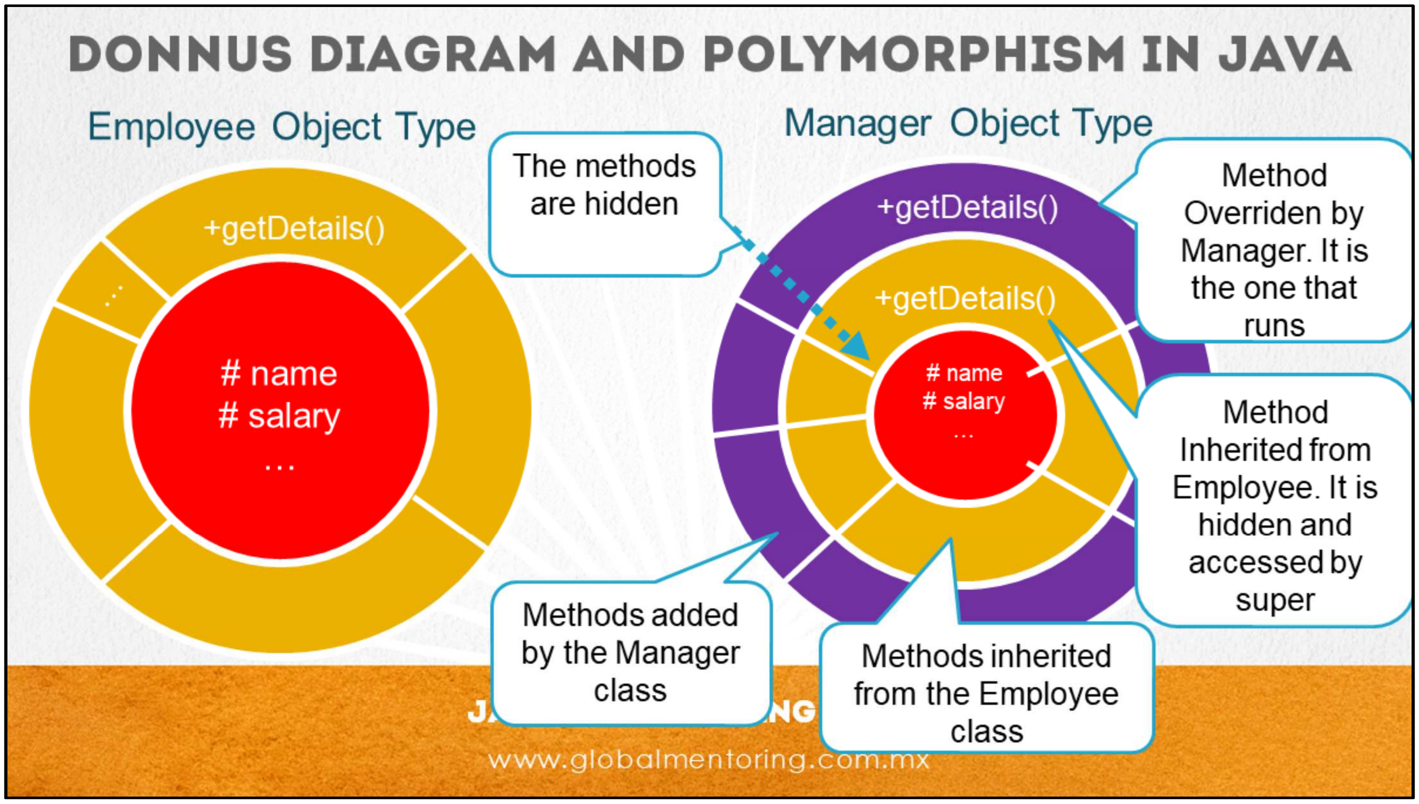
The importance of polymorphism is that we can generalize a method that receives different types in the defined class hierarchy (parent and child classes), for example a method that prints out the details of each class, regardless of whether it is an object of type parent or of child type, and all this, with a variable of type father.

In the example shown, we observe that in the code a variable of type Employee (parent type), named emp, is created. This variable is stored in the stack memory, since it is a local variable. And the variable emp, is assigned a reference of a parent object, that is, of type Employee, with the attribute named John. And when the method getDetails() with the variable emp is called, the method that is executed is that of the Employee class.

So far there is nothing new, since we are calling a method of the same type of the Father class. However later with the same variable of type father, we assign a reference of type child, that is to say the class Manager, with the attribute of name Katty, and later we call the method getDetails(). This is where the doubt may arise, what is the method that is sent to call ?, that of the Employee class because the variable emp is of the Employee type, or that of the Manager method, since the reference to which it points the variable emp is of an object of type Manager?

The answer to this question has to do precisely with the concept of polymorphism, which says that the method that is executed will be of the object whose reference is pointing in time of execution, therefore we can conclude that in this last case, the method that Execute is the method getDetails() of the Manager class, since it is the type to which the variable emp refers. So, regardless of whether the variable emp is of the type used, the method that will eventually be executed will be of the type to which it refers.

The variable emp can store a reference of type Manager because the Manager class extends from the Employee class, otherwise it would not be possible to store this reference, for example, the variable emp can not store a reference of a String type, since the String class does not extend the Employee class, and therefore they do not have anything in common, so to speak of polymorphism, this will also necessarily deal with the issue of inheritance and some kind of relationship between the classes that will execute the polymorphism.


DONNUS DIAGRAM AND POLYMORPHISM IN JAVA
Employee Object Type
+getDetails()
# name
# salary
...
The methods are hidden
Manager Object Type
+getDetails()
+getDetails()
# name
# salary
...
Method Overriden by Manager. It is the one that runs
Method Inherited from Employee. It is hidden and accessed by super
Methods added by the Manager class
Methods inherited from the Employee class
www.globalmentoring.com.mx

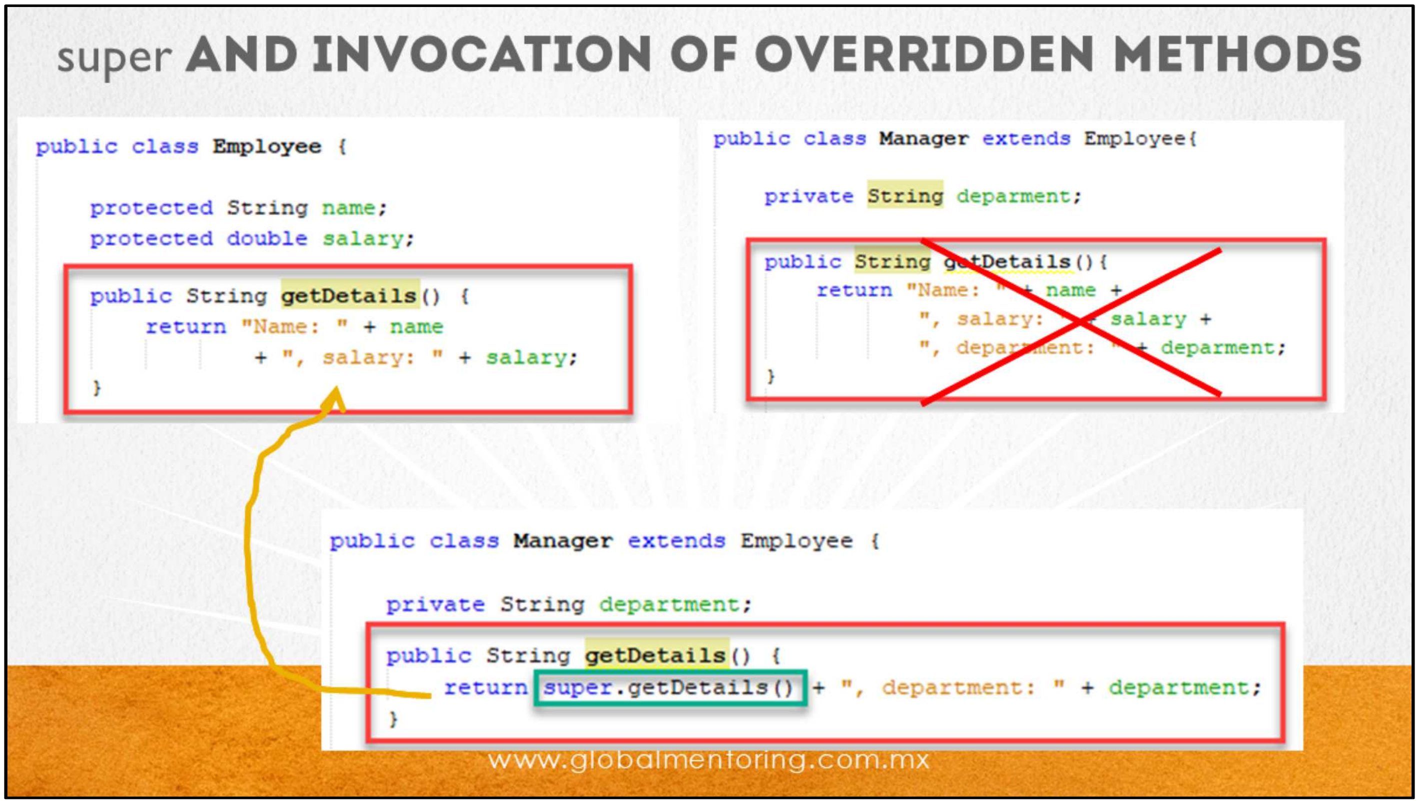
We can summarize the concept of polymorphism with the following donut diagram.

We observe two cases, an object of type Employee and another object of type Manager, and observing this diagram we can easily know which method is the one that is being executed in the object and the reason why the method we indicate is executed.

In the case of the Employee type, a variable that executes the method GetDetails () will have no doubt that the method defined in this class will be executed, since it is the parent class.

But in the case of the Manager type, as we have seen, there may be doubt as to which method is executed, whether that of the child class or the parent class. However, as we can see the donut diagram of the Manager type object, the most external methods are those defined in the Manager class (purple color), and the most internal are the methods inherited from the Employee class (yellow color). So we can say that the method that will be executed is the most external method, and the one that will be hidden will be the inherited method, or what is the same, the most internal method. If we want to access the hidden method we can use the word super followed by the hidden method. Next we will see an example of this.

We have added some notes about the donation diagram so that we can observe the order in which they are accessed and hiding the methods. In addition, the methods that are not hidden, that is, that have not been overwritten, are accessed directly as if they had been defined directly in the Manager class, since as we know that is the concept of inheritance. But at the end, you must remember that is just on object, not two.



super AND INVOCATION OF OVERRIDDEN METHODS

```java
public class Employee {

    protected String name;
    protected double salary;

    public String getDetails() {
        return "Name: " + name
                + ", salary: " + salary;
    }
```

```java
public class Manager extends Employee{

    private String deparment;

    public String getDetails(){
        return "Name: " + name +
               ", salary: " + salary +
               ", deparment: " + deparment;
    }
```

```java
public class Manager extends Employee {

    private String department;

    public String getDetails() {
        return super.getDetails() + ", department: " + department;
    }
```

www.globalmentoring.com.mx

A subclass can invoke a method of the parent class by means of the "super“ keyword. The word "super" is like the operator "this" but refers to the parent class.

The "super" keyword can reference attributes and methods of the parent class. The method that is invoked by means of super, should not necessarily be declared in the parent class, but could be declared in a class higher up in the class hierarchy.

As we can see in the figure, one of the advantages of using super is the reuse of code. So instead of duplicating the code of the Employee class, we can take advantage of what this method already does and only add to the method overwritten by the Manager class what we need to complete this method, that is, add the department attribute. In this way we can reuse the code of the Parent class if applicable.

Let's see below an exercise to apply all these concepts.

ONLINE COURSE

JAVA PROGRAMMING

By: Eng. Ubaldo Acosta

JAVA PROGRAMMING COURSE

www.globalmentoring.com.mx