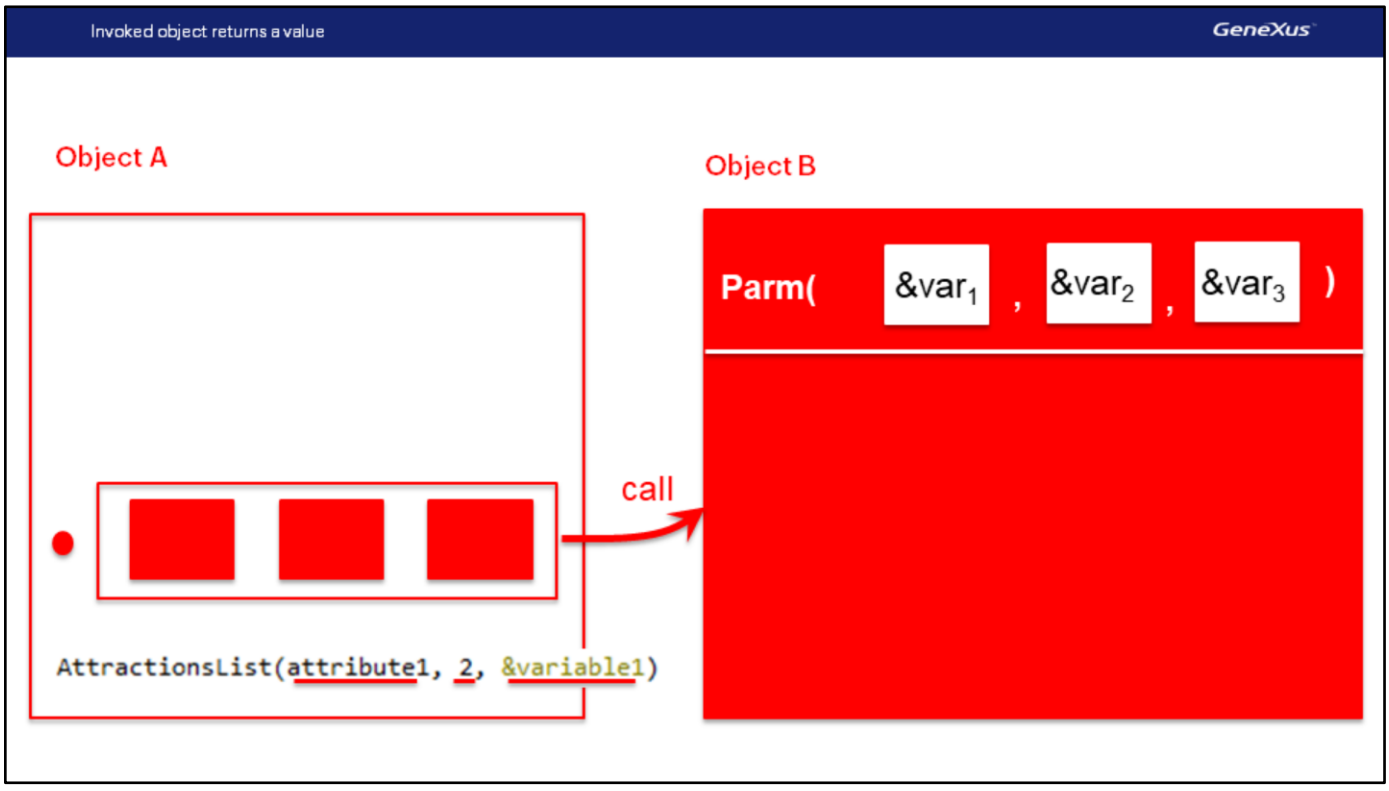
When the invoked object returns a value

GeneXus 16

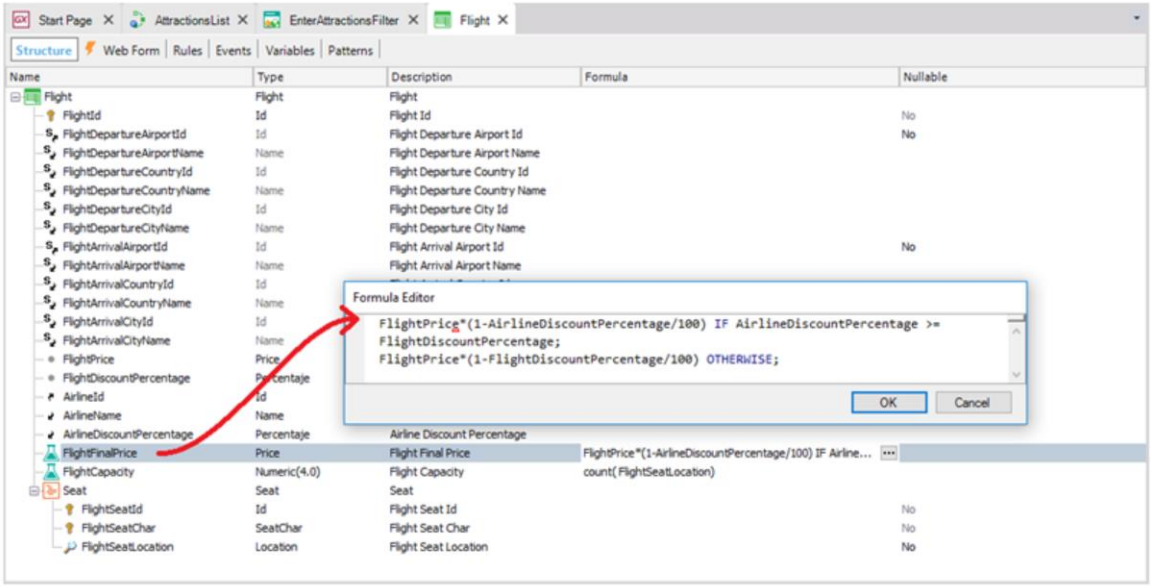
When the invoked object returns a value



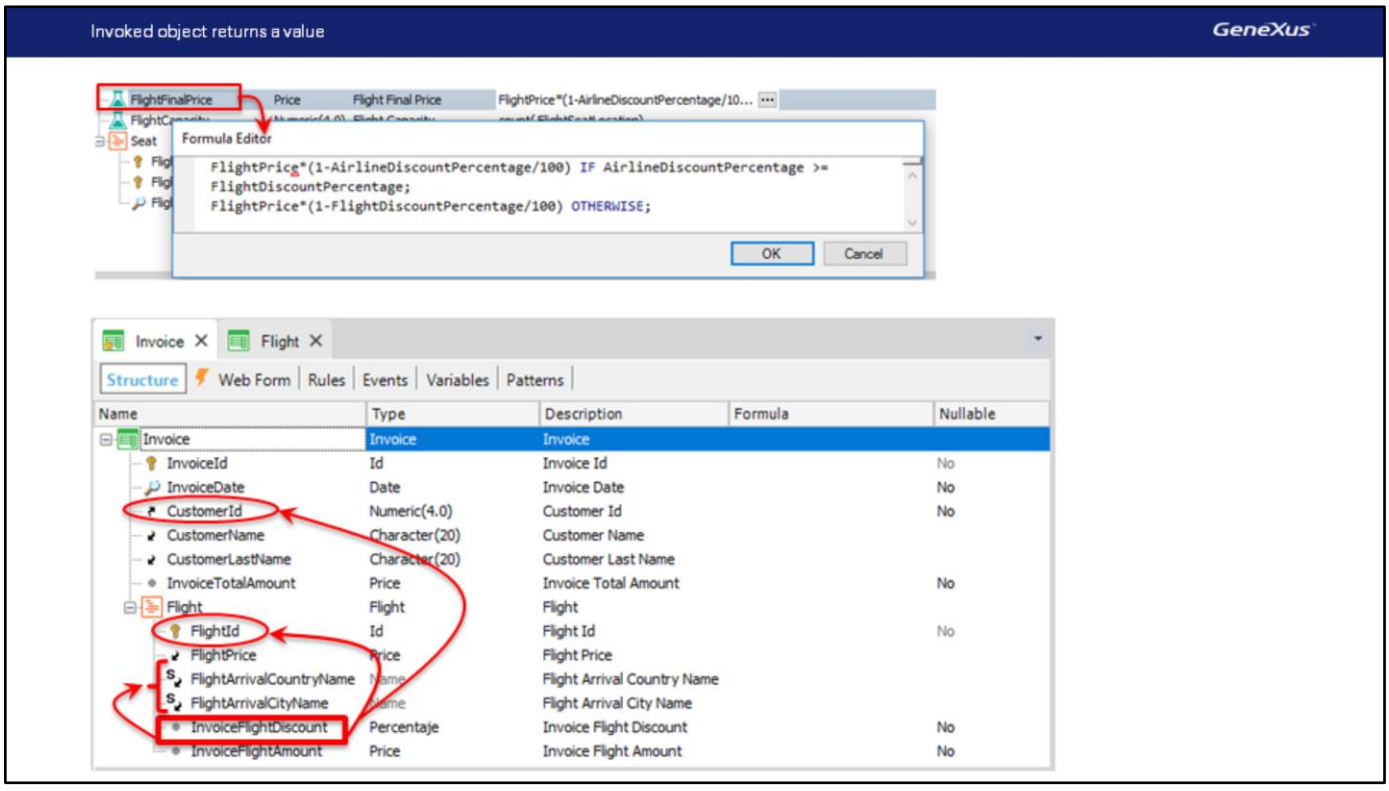
We have seen how to state parameters in an object to allow it to receive data from another object and perform the corresponding actions according to this data. To this end we used the Parm rules and variables. The examples we saw involved input parameters; that is to say, parameters only received by the object.

In this way, if object B has a Parm rule stated with three variables, to invoke object B, any object will have to send it three values that, as we've seen, may be saved in attributes, be an expression (as in the case of a fixed value), or be saved in variables.

Now we'll see what happens when object B must **return** a value to the caller object, when it ends running.

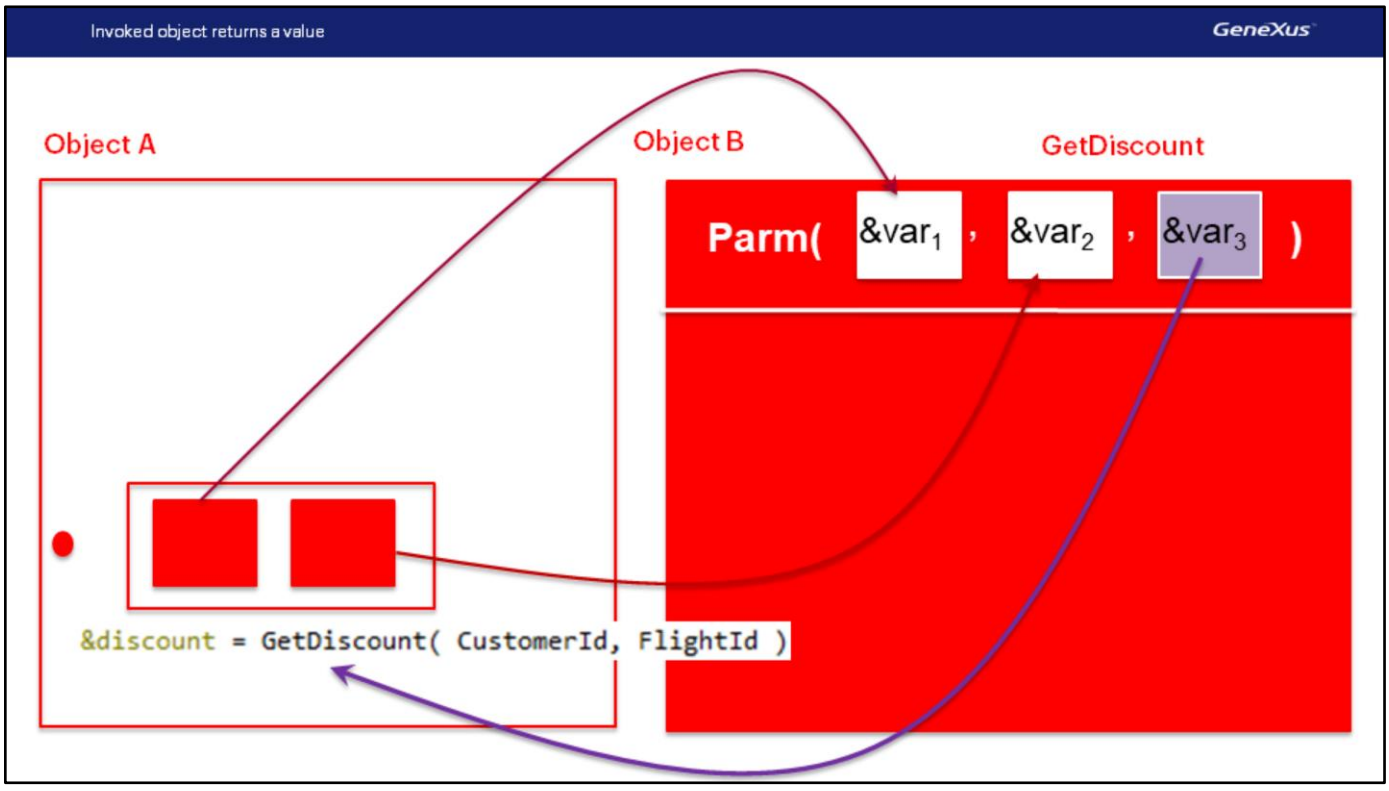


In the Flight transaction we had a formula that calculated the price of a flight according to the discount percentage offered by the airline and the percentage indicated for the flight itself. It selected the biggest discount and applied it.



Suppose we're creating a transaction to record the invoices issued to customers when they purchase flight tickets.

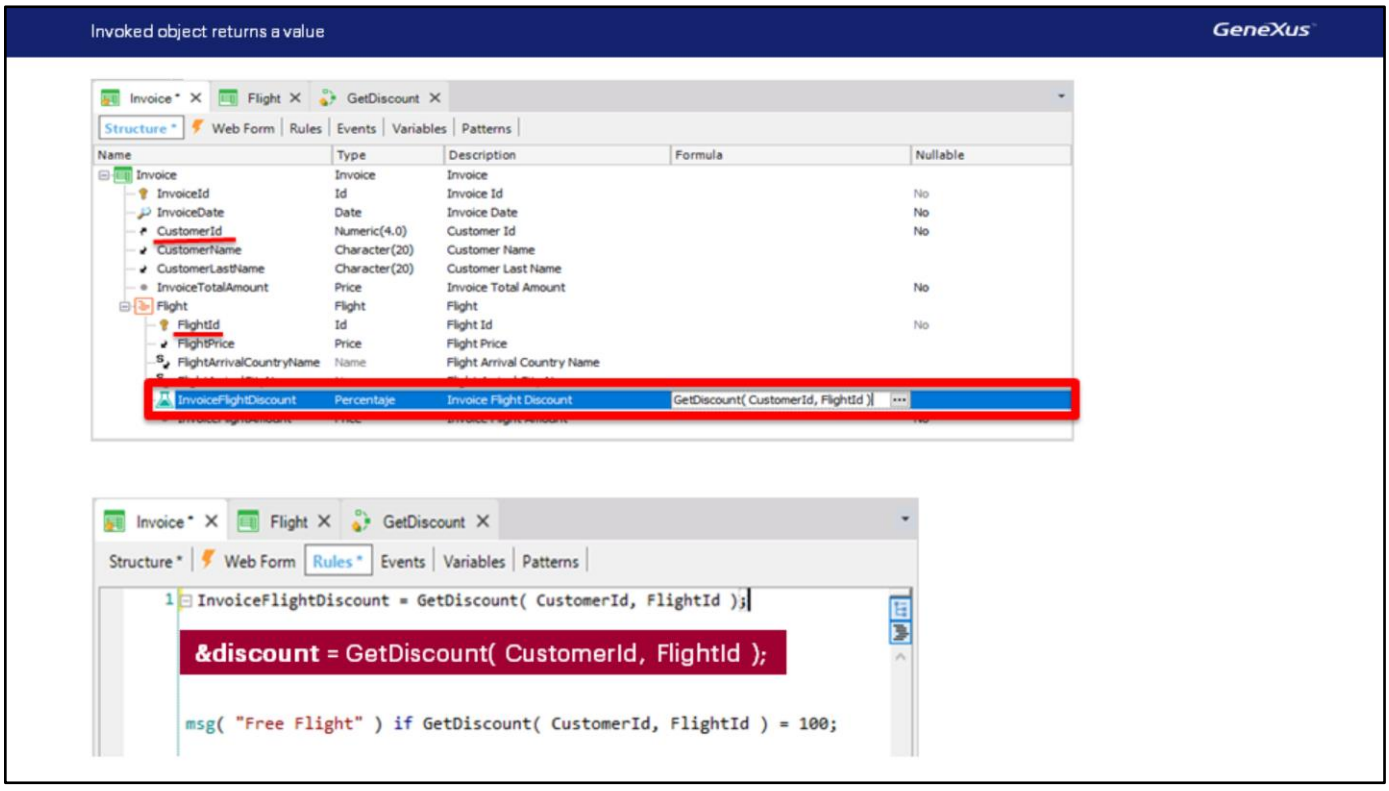
Also, suppose that the discount is a more complex calculation that implies not only the flight, but also some condition related to the customer who is purchasing a flight ticket. For example, the number of tickets that he has purchased in the past, if he is a recurring customer, and if a destination is offered at a discount. The discount percentage is determined according to these more complex conditions.



For cases such as this, we may need to implement a procedure that makes these calculations and returns the resulting value to the caller.
For example, we could call this procedure GetDiscount.

The procedure will have to receive the customer and flight as input parameters. It will return the resulting discount.

The first question is how this result is received by the object that needs the procedure result. It has to be considered as a function that is called in order to perform an action with the result returned.



One possibility is to assign the result to an attribute. For example, we could define the FlightDiscount attribute in the Invoice transaction structure as a formula that is calculated by invoking GetDiscount

In this way, the formula will be evaluated in every object where the InvoiceFlightDiscount attribute is mentioned. The GetDiscount procedure will be invoked and executed, and when it ends running, the returned result will be shown as the formula attribute value.

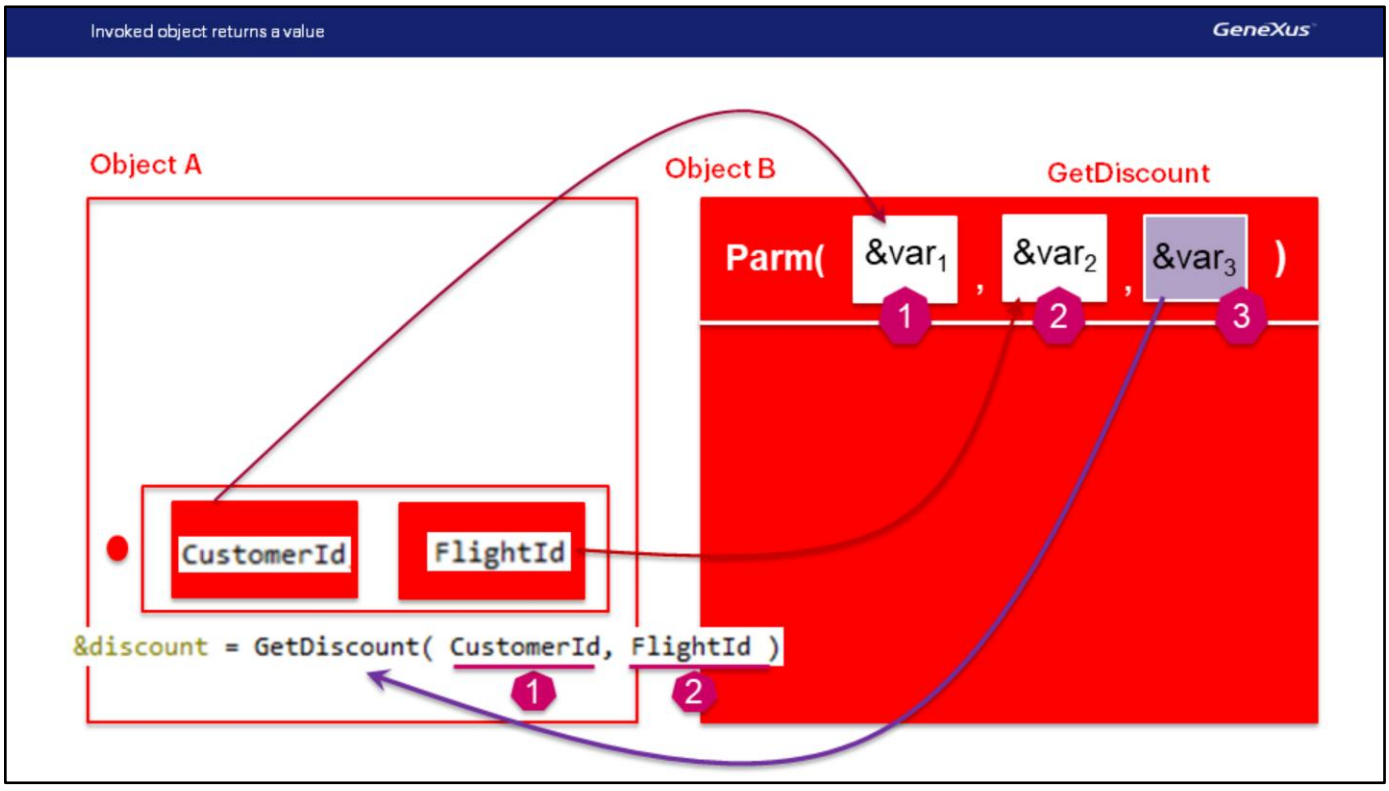
If we don't want to set this attribute as a formula, but rather we want it to be an attribute stored in the corresponding table, and have it stored with the procedure result only when the transaction is executed, we type in the rules the first assignation we could see above.

In addition, the result of the procedure's execution could be assigned to a variable.
Also, it may not be assigned, but used in an expression instead. For example, to condition the triggering of a rule.

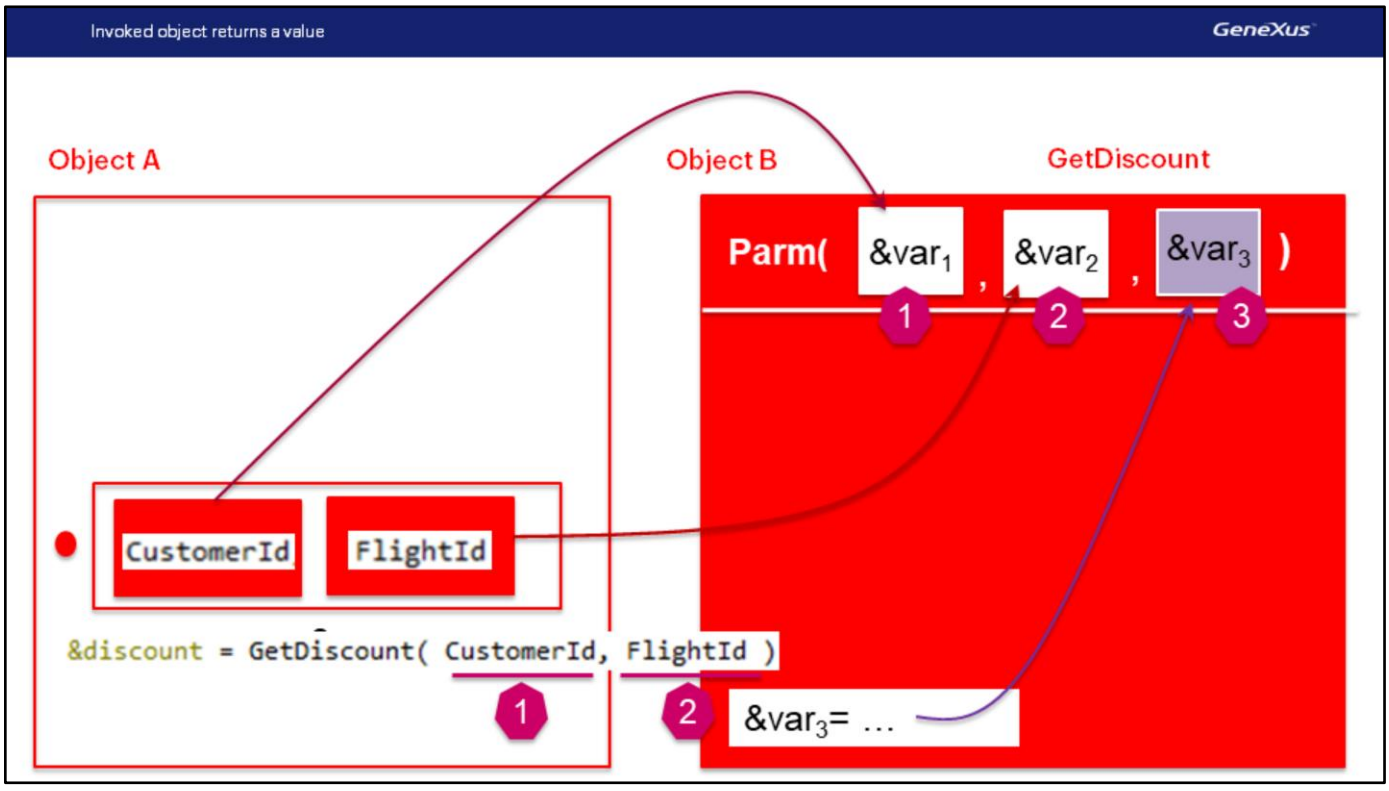
Or of instructions in a procedure or an event:

```
If GetDiscount( CustomerId, FlightId) > 10
    ...
Endif
```

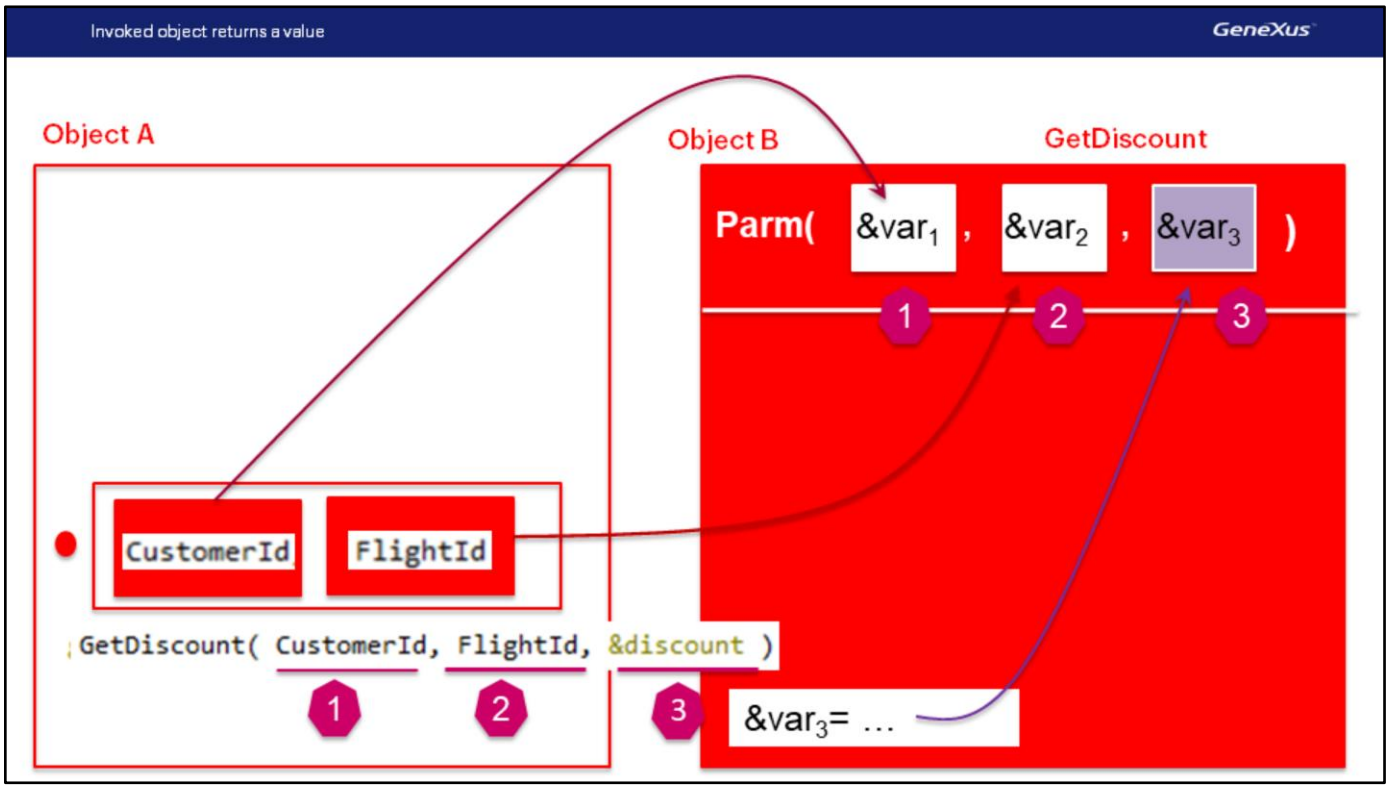
We will not talk about how to implement the GetDiscount procedure, because it isn't relevant for what we're studying now. However, we must see how the **parm rule** is stated in the called object when the call syntax assumes that the object returns a value, as in the examples that we've just mentioned.



In the rules section of the GetDiscount procedure we must state the Parm rule with the number of parameters described in the call... plus one at the end...

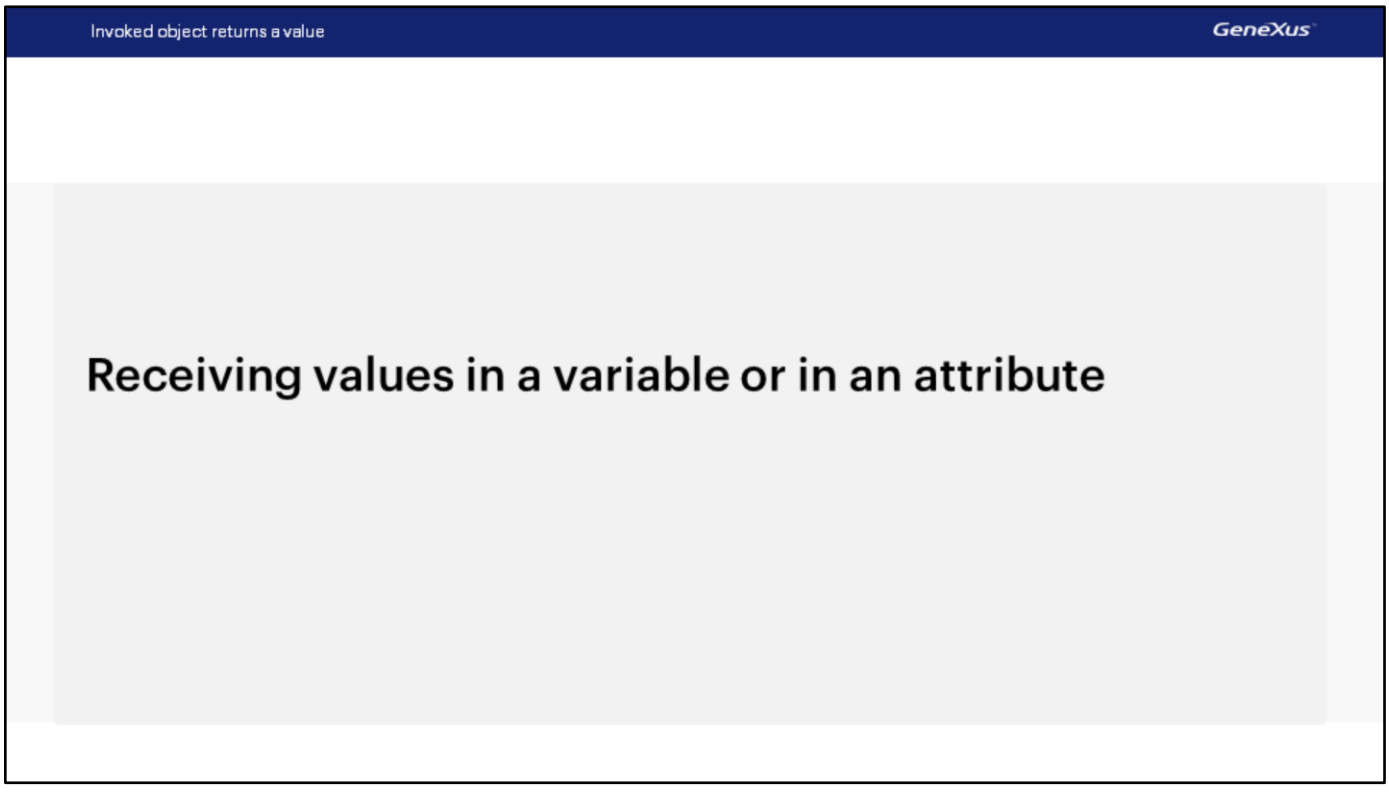


...that will have to be a variable whose value is loaded in the object code (and in our case, in the procedure Source). The value taken by the variable when the code ends running will be the value returned to the object that called it.

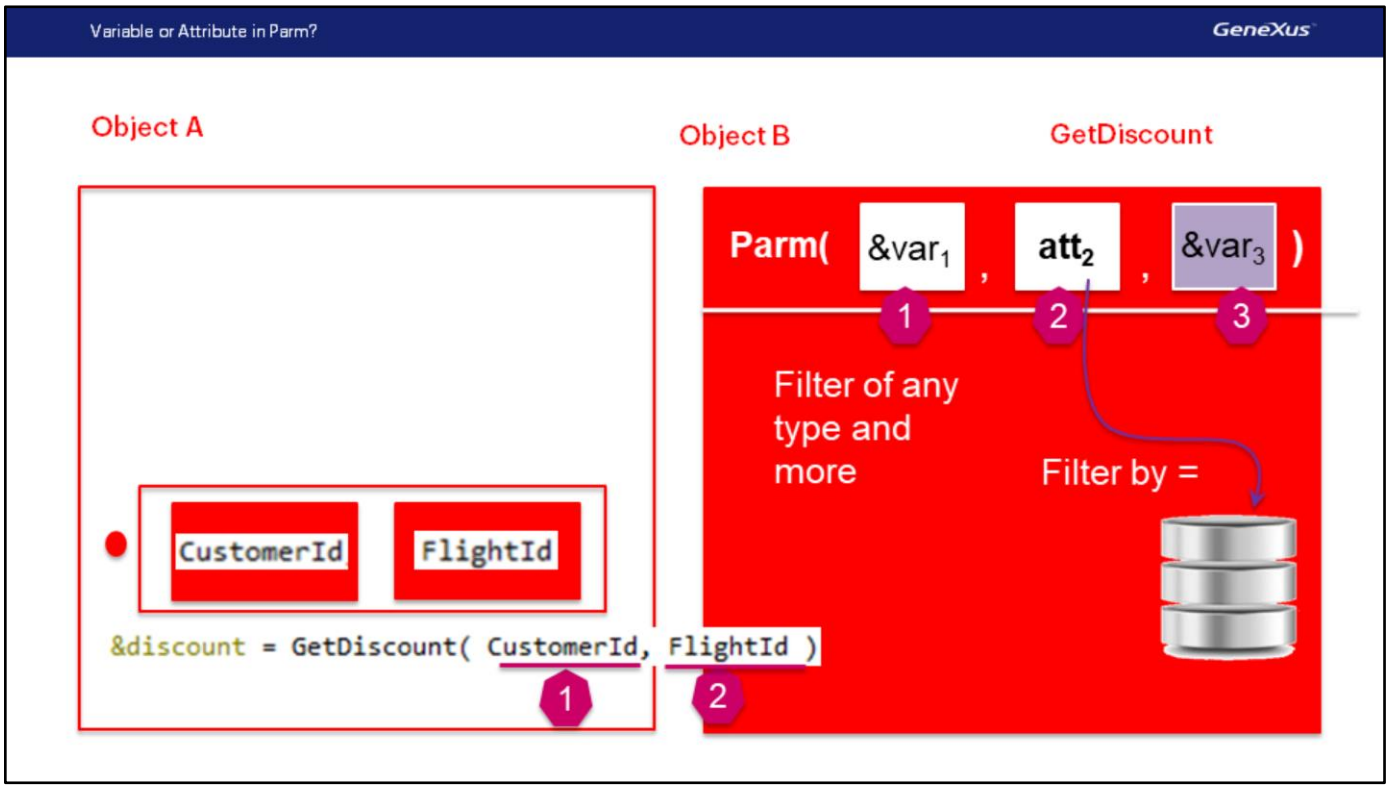


For this same Object B we could have made the invocation as we did before:
where the first two parameters are input parameters and the last one is an output parameter.

The only issue is that in this way the invocation doesn't clearly indicate that &discount will return loaded. In other words, it isn't made clear that the invoked object will return a value. In these cases, using the invocation syntax that clearly indicates it is recommended.



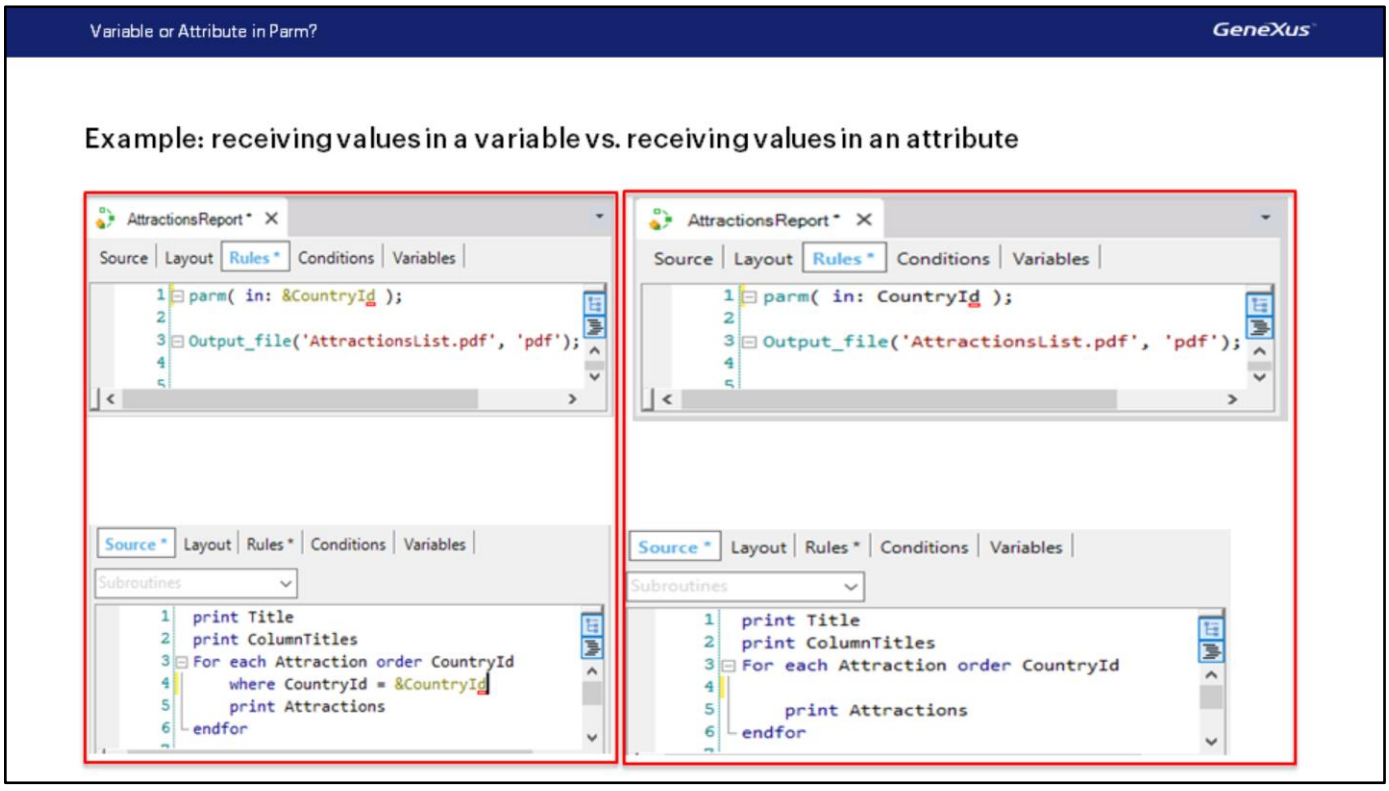
Lastly, let's look at the case where a parameter of the Parm rule is an attribute instead of a variable.



What's the difference between using a variable or an attribute in the **Parm** rule of the invoked object?

If the value is received in a **variable**, it can be freely used in the programming: it can be used as a filtering condition for filters such as equality, higher than, higher than or equal to, and so on... also, it can be used for an arithmetic operation, or for whatever is necessary. It's a space in memory with a name that we use within the object through explicit instructions, to do what we want.

If, on the other hand, the value is received in an attribute, this is fixed, determined, and implicit. We receive values in an attribute when we access the database from inside the object. In particular, a table in whose extended table this attribute is stored. So, when a value is received in that attribute through a parameter, an equality filter will be applied. Only the records that have that value for the attribute will be considered.
Let's see this with an example.



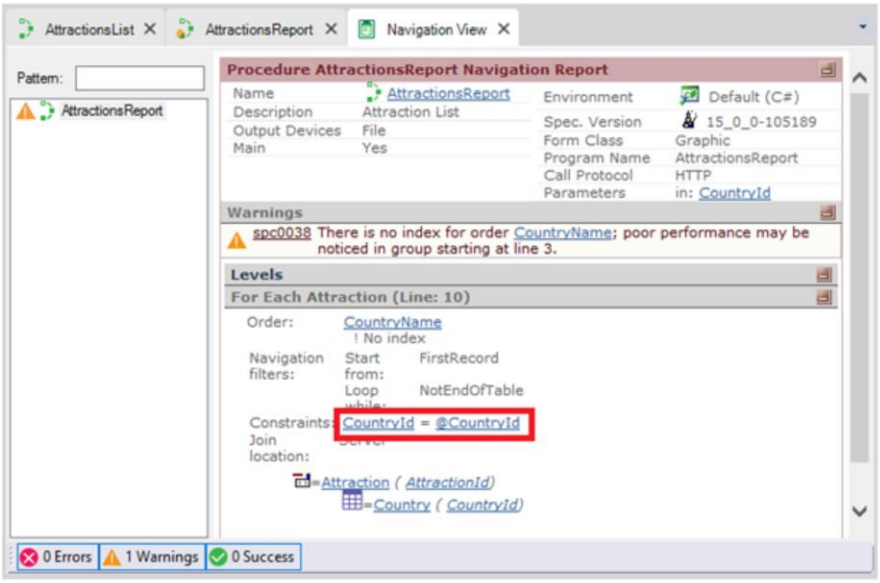
We make a copy of the AttractionsList procedure with the AttractionsReport name.

Remember that the procedure on which it is based uses a variable as a parameter: &CountryId. It used it to filter the attractions in the Attraction table by country.

As we can see, it is implementing an equality filter: it will list only those attractions whose CountryId matches the value of the &CountryId variable received in a parameter.
We could have implemented exactly the same without explicitly indicating that filter.
How? By receiving the value directly in the CountryId attribute.

When we receive the value in an attribute in the Parm rule, GeneXus uses an equality filter; that is to say, only those records that have the same country identifier are accessed.
If we look at this object's navigation list...

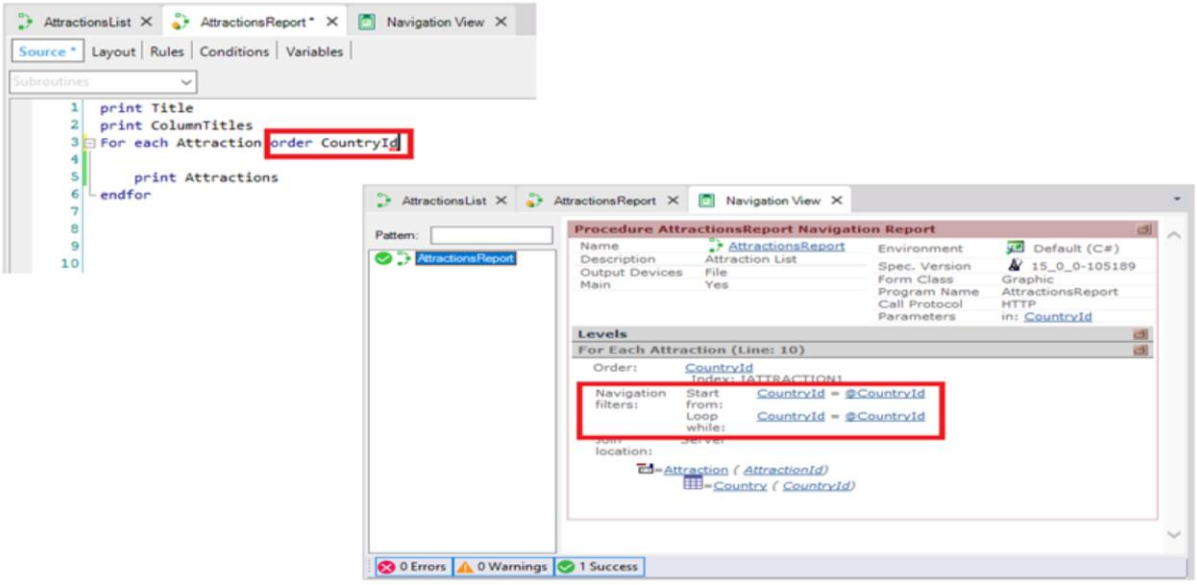
Navigation List



...we see that the filter is applied even if the Where clause is not written.

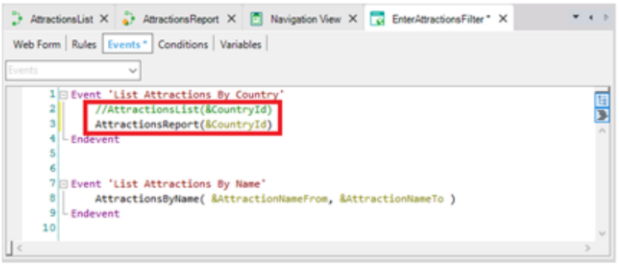
It's interesting to note that since the For Each command is being run through ordered by CountryName, the entire table has to be run through to filter by the CountryId values corresponding to the parameter.

Navigation List



On the other hand, if we order by the attribute used to filter, we see in the navigation list that the entire table is no longer run through.

DEMO

- 

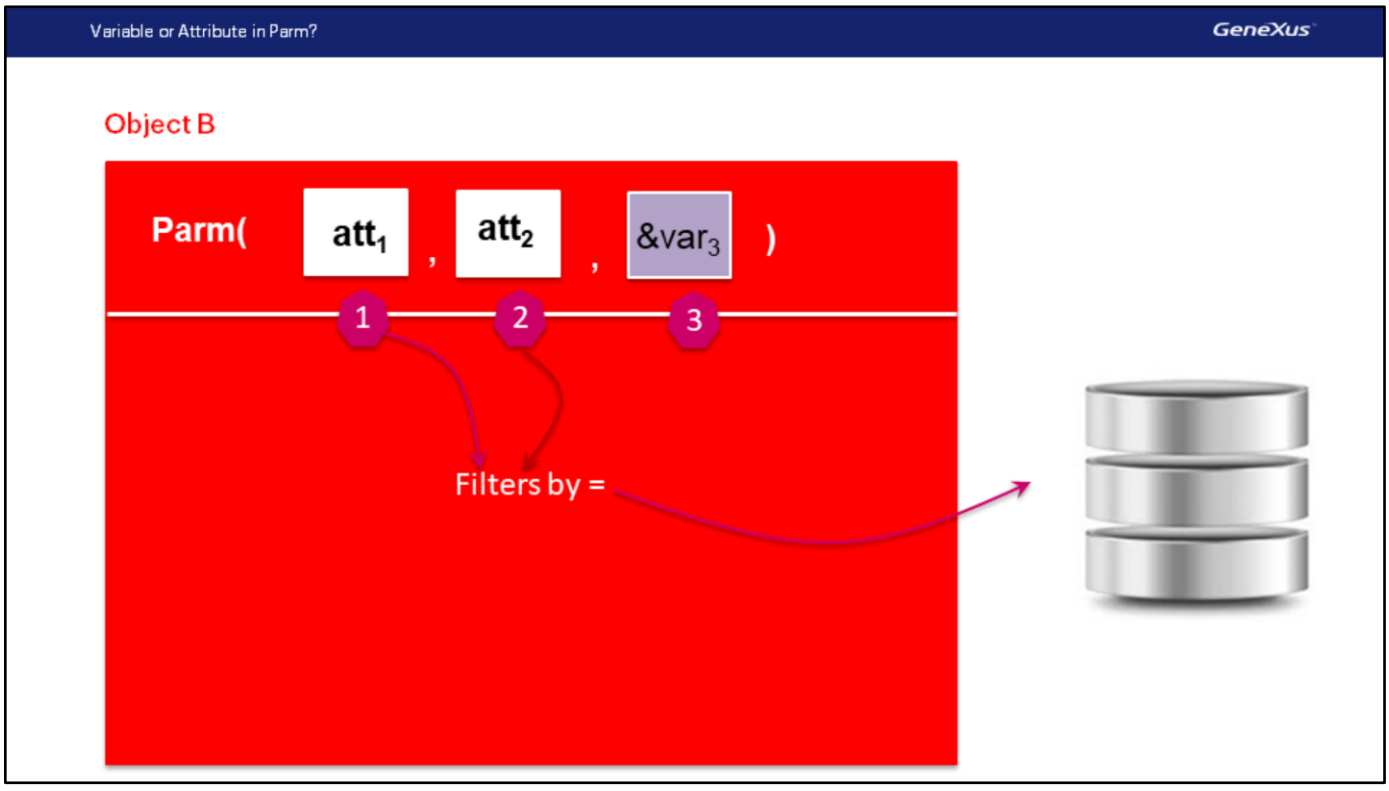
- F5 → run Web panel, select France

DEMO

Let's replace the invocation in the web panel that called AttractionList with an invocation to this other procedure: What comes after a double slash is considered a comment; that is to say, it isn't interpreted by GeneXus as an instruction. In this way, we comment the first invocation and type the new one:

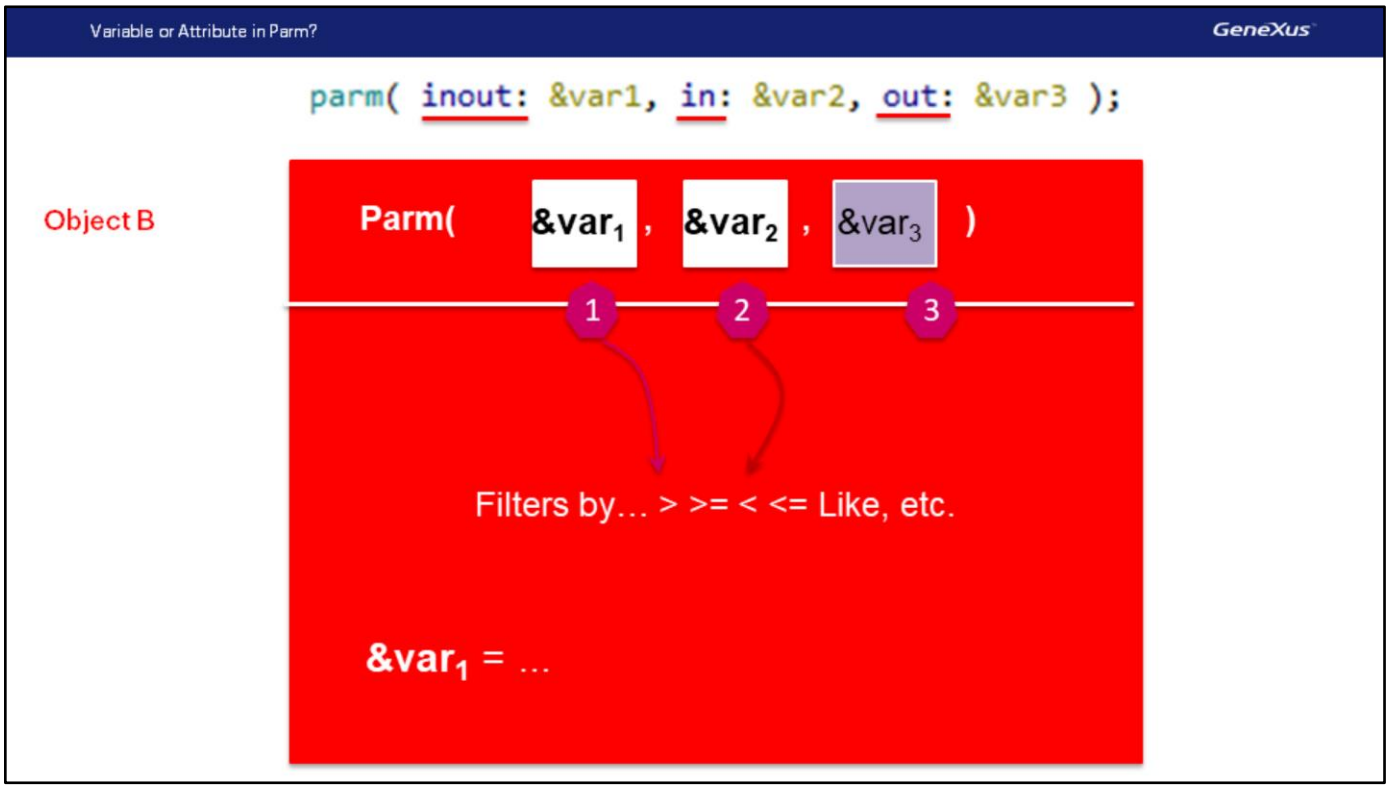
We run it...
We select France and list it...
As we can see, the list contains the tourist attractions of France.

Lastly, upload everything to GeneXus Server.



If we received more than one value using attributes to receive them, only the records that have the same value as each attribute received would be accessed.

Also, we can't change these attribute values.



If our objective is not to receive values to apply an equality filter, the solution will be to receive values in variables instead of using attributes. In addition, they can be freely used in the programming, for example, to assign other values to them if necessary.

In this sense, we must also say that not only the last variable of the Parm rule can be used as an output parameter; that is to say, it will be returned to the caller. All variables can be input, output or input/output variables. In the example, &variable 1, that as we indicated changes its value inside the object code, could be an output, or input/output variable.

This can be indicated by typing “in”, “inout” or “out” before the variable names, as we can see here.

Communication between objects is essential for any GeneXus application, because it enables an object to start running another object and send or receive information to and from it.

GeneXus™

The power of doing.

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certifications	training.genexus.com/certifications