

Checking for null values and initializing them is a necessary step in the implementation of a ruleset, because input objects might be null or contain null values.

For any Java™ application, rules developers must implement checks for null values on the objects used in the application (within rules in the case of JRules) to avoid potential NullPointerExceptions (NPEs) such as the following:

```
Exception in thread "main" An
```

The rule developer must check that none of the input parameters are null, unless they are of a Java primitive type.

Indeed, if a parameter 'customer', verbalized as 'the customer' and of type test.Customer, is null and a rule author writes the following condition:

if the age of

It translates in IRL (ILOG Rules Language) into the following statement, and produces an NPE on 'customer.age' because 'customer' is null:

```
evaluate (customer.age > 2
```

On the other hand, null objects are not inserted in the working memory, so they will never be bound in the rules.

Therefore, the rule developer does not have to check if the working memory objects themselves are null.

Whether the object used in a rule is a parameter or a working memory object, the rule developer must check for nulls on its attributes and method return values of the following types:

- Java wrapper types (i.e. `java.lang.Integer`, `java.lang.BigDecimal`...)
- `java.lang.String`
- Custom types

Note that even if an attribute is found on the right-hand side of a comparison, it might generate an NPE if it is null. This can be due to the automatic unboxing of Java wrapper types, see the documentation for JRules 7.1 at [Rule Studio > Optimizing execution > Run-time efficiency > Autoboxing](#).

For example, the following condition:

```
if      the age of
```

Is translated as follows in IRL:

```
evaluate (person.age > exa
```

Where 'ageLimit' is a java.lang.Integer implicitly unboxed to its Java primitive type 'int' to allow the comparison to 'age' which is of type 'int'. This is the same mechanism used in Java 5 when comparing a primitive type and its wrapper type.

The check for null values can be performed before any rule is executed or while the rules are being executed.

In any case, because the notion of null value is a technical concept, **the task of checking for nulls must remain in the hands of the rule application developer.**

It should not be the responsibility of rule authors when they are writing rules, as they are likely to be business analysts for whom the notion of null values is abstract.

Therefore, the check performed during the rule execution must be done implicitly: rule authors write the phrases that make sense to them from a business point of view and have checks for nulls performed in the background at execution time.


"is null"/"is not null" operators are available in BAL rules, but are not valid or available for all types. Again, business users might not know programming concepts such as null values and the order of test evaluations to use them.

Checking for null values

Sist oppdatert: 2021-03-03

You can include a condition to check that an entity is null, or not null, in your rules.

Use the `if <entity> is null then...` construct to test whether an entity is null. For example, the following rule checks if the relationship to the `customer` entity does not exist:

```
when a purchase event occurs , call   
if  
    the customer of 'the purchase'  
then  
    print "There is no customer!";
```