

# A New Semantics for JML Signals Clauses

*Kristof Mertens*

*Nele Smeets*

*Marko van Dooren*

*Jan Dockx*

*Eric Steegmans*

*Report CW 343, June 2002*



Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# A New Semantics for JML Signals Clauses

*Kristof Mertens*

*Nele Smeets*

*Marko van Dooren*

*Jan Dockx*

*Eric Steegmans*

*Report CW 343, June 2002*

Department of Computer Science, K.U.Leuven

## Abstract

JML, which stands for "Java Modeling Language", is a behavioral interface specification language designed to specify Java classes and interfaces. JML uses syntactic sugar to make specifications easier to read and write. In this paper, we reveal that the current JML desugaring of signals clauses has a somewhat counterintuitive meaning. A new semantics is proposed that is more intuitive. Special attention is given to the desugaring of signals clauses that are spread over different behavior blocks connected by the keywords **also** or **and**. The described desugaring ensures behavioral subtyping. The consequences of the new semantics are worked out.

**Keywords :** exceptions, specification, formal specification, signals clause, signals, JML, BSL, desugaring, semantics

# A New Semantics for JML Signals Clauses

Kristof Mertens, Nele Smeets, Marko van Dooren, Jan Dockx, Eric Steegmans

Katholieke Universiteit Leuven, Department of Computer Science,  
Celestijnenlaan 200A, 3001 Leuven, Belgium  
{Kristof.Mertens, Nele.Smeets, Marko.vanDooren, Jan.Dockx,  
Eric.Steegmans}@cs.kuleuven.ac.be

**Abstract.** JML, which stands for "Java Modeling Language", is a behavioral interface specification language designed to specify Java classes and interfaces. JML uses syntactic sugar to make specifications easier to read and write. In this paper, we reveal that the current JML desugaring of signals clauses has a somewhat counterintuitive meaning. A new semantics is proposed that is more intuitive. Special attention is given to the desugaring of signals clauses that are spread over different behavior blocks connected by the keywords `also` or `and`. The described desugaring ensures behavioral subtyping. The consequences of the new semantics are worked out.

## 1 Introduction

JML [4], which stands for "Java Modeling Language", is a behavioral interface specification language (BISL) designed to specify Java [1][3] classes and interfaces. JML offers syntactic sugar to make specifications easier to read and write.

The current desugaring for JML method specifications is explained in [5]. In this paper, we focus our attention on the signals clause. The predicate of the signals clause describes the conditions under which exceptions may be thrown ([4], paragraph 2.2.1.5).

In section 2, we describe the methodology that is used in JML for desugaring a single signals clause in a behavior specification block. We explain why this desugaring appears strange to us.

In section 3, we suggest a new desugaring for signals clauses. In subsection 3.1, we describe how we would like to interpret a single signals clause. Subsection 3.2 gives suggestions for the desugaring of a single signals clause combined with a `requires` clause. In subsections 3.3, 3.4, and 3.5, we describe the interpretation of the combination of multiple signals clauses. Three situations have to be considered: the signals clauses can be (1) in one behavior block or in different blocks (2) connected by `also` or (3) connected by `and`. Special attention is given to the desugaring of the last two cases: the keywords `and` and `also` can be used to connect the specification of a method in a subclass with the specification given

in the superclass. In JML, behavioral subtyping is supported; our desugaring ensures that behavioral subtyping is enforced. Finally, 3.6 describes the desugaring of normal and exceptional behavior blocks. For all these situations, the formal desugaring is given, together with an informal description of the meaning of this desugaring. In section 4, some conclusions are given.

## 2 Current Semantics of JML Signals Clauses

In this section, the current JML semantics for a single signals clause in a behavior block that is not a normal or exceptional behavior block is described, together with the reasons why this semantics seems strange to us.

The current semantics for a single signals clause<sup>1</sup> in JML is as follows

```
public behavior
  signals (AE) a;
```

is desugared to ([5], Chapter 3.4)

```
public behavior
  signals (Exception e) (e instanceof AE) ==> a;
```

<sup>2</sup>This last clause means that `a` has to be satisfied when an exception of type `AE` is thrown, but it does not forbid throwing an exception of other types than `AE` even when `a` is not satisfied. This is because `false ==> something` is `true`. Although this could be a valid desugaring, it seems very confusing to us, and probably also to most readers and writers of similar specifications.

The JML desugaring also makes it very hard to specify the common situation where we want to express that an exception of a specific type can be thrown under a certain condition. For example, if we want to specify that only an instance of `AE` may be thrown and this only when `a` is true, we have to write explicitly:

```
public behavior
  signals (Exception e) (e instanceof AE) && a;
```

## 3 A New Semantics

In the previous section, the current JML desugaring of a single signals clause was described together with our remarks about this semantics. In this section, a

---

<sup>1</sup> Note that in this paper only public specifications are used, but the same applies to package accessible, protected or private specifications.

<sup>2</sup> The symbol `==>` is the logical implication.

new desugaring for signals clauses is proposed, taking into account the remarks given in the previous section.

In section 3 of [5], the different steps for desugaring a method specification are explained. We do not add or remove steps and we also keep the same order of steps. The only thing we do is suggest another desugaring for some steps<sup>3</sup>. Although most desugaring steps are the same as the current JML desugaring, we consider the meaning of all steps related to signals clauses: because the desugaring of the basic step is changed, the meaning of the more complex usages of signals clauses changes also. We describe the new interpretation in full detail.

### 3.1 A Single Signals Clause

In this subsection, the semantics of a single signals clause in a behavior block that is not a normal or exceptional behavior block is discussed.

We suggest that a signals clause like

```
public behavior
  signals (AE) a;
```

is desugared to

```
public behavior
  signals (Exception e) (e instanceof AE) && a;
```

This means that an exception `e` can only be thrown when the exception is of type `AE` and the condition `a` is satisfied.

This semantics makes it easy to specify the typical situation in which an exception of a fixed type may be thrown and this only under a specified condition.

### 3.2 Desugaring a Behavior Block Containing a Requires Clause

For a behavior block containing a requires clause and a signals clause, we suggest to keep the current JML desugaring for a requires clause ([5], Figure 14). Because the changes in the desugaring of a single signals clause the complete desugaring for a behavior block with a requires and signals clause will change.

```
public behavior
  requires r;
  signals (AE) a;
```

---

<sup>3</sup> The problem of variable capture remains also in the new semantics; the solution, renaming, is also the same. In this paper, no further attention is paid to this problem.

becomes

```
public behavior
  requires r;
  signals (Exception e)
    \old(r) ==> (e instanceof AE) && a;
```

This means that no condition is imposed if `r` is not true in the pre-state; if `r` is true in the pre-state, then an exception may be thrown, but only if it is an instance of `AE` and `a` is satisfied. This can be seen as: the contract of a `signals` clause has to be satisfied only when the predicate in the `requires` clause evaluates to true.

### 3.3 Multiple Signals Clauses in One Behavior Block

We suggest to connect multiple `signals` clauses in one behavior block with `||` (conditional or). The behavior block

```
public behavior
  signals (AE) a;
  signals (BE) b;
```

is then desugared to

```
public behavior
  signals (Exception e)
    ((e instanceof AE) && a) ||
    ((e instanceof BE) && b);
```

This means that an exception `e` can be thrown under the following conditions:

- if `BE` is the same as `AE`: `e` can be thrown when "`e` is an instance of `AE`" and "`a` or `b` is true"
- if `BE` is a subtype of `AE`: `e` can be thrown when one of the following is true<sup>4</sup>:
  - "`e` is an instance of `AE`" and "`a` is true"
  - "`e` is an instance of `BE`" and "`a` or `b` is true"
- if `BE` is not a subtype of `AE` and `AE` not a subtype of `BE`<sup>5</sup> then `e` can be thrown if one of the following is true:

<sup>4</sup> The symmetric situation where `AE` is a subtype of `BE` is not considered here because it is similar to the described case.

<sup>5</sup> We assume here that only subtypes of `Exception` may be mentioned in a `signals` clause. This is also the current JML meaning and it will probably stay so, as was said in an email from Gary Leavens on the JML mailing-list on April 17, 2002.

- "e is an instance of AE" and "a is true"
- "e is an instance of BE" and "b is true"

The interpretation given above makes it easy to handle typical situations in which a method can throw a number of different exceptions, each under one or more different conditions. This can be expressed by enumerating all exceptions and their corresponding condition(s). When an exception can be thrown under more than one condition, these conditions can be spread over different signals clauses. Using this syntactic sugar will typically make the specification more readable.

### 3.4 Multiple Signals Clauses in Different Behavior Blocks Connected With `also`

In subsections 3.4 and 3.5, the desugaring of multiple signals clauses spread over different behavior blocks, connected by `also` and `and`, is discussed. The connectors `also` or `and` are especially interesting since they can be used to connect the behavior block of a method in a subtype with the behavior block of the corresponding method in the supertype.

Since JML supports behavioral subtyping ([4] section 2.6), our desugaring is written in such a way that behavioral subtyping is ensured. Some more information about behavioral subtyping is given in the following introductory subsection.

**Behavioral Subtyping** In JML, behavioral subtyping is forced through specification inheritance [2] and the semantics that is used for interpreting multiple specification blocks connected by `also` or `and`. Therefore, it is needed to consider behavioral subtyping when defining a desugaring for `also` and `and`. Behavioral subtyping has, according to [2] section 4, two categories of constraints:

- Syntactic constraints ensure that an expression of a subtype can be used in place of an expression of its supertype without any error.
- Semantic constraints ensure that subtype objects behave like supertype objects; that is, the use of subtype objects in place of supertype objects does not produce any unexpected behavior. Expected behavior is specified by the supertype's specification.

Again, according to [2] section 4, methods in subtypes may not throw exceptions that were not mentioned in the supertype. The exceptions mentioned in the method specification of the subtype may be more specific than those in the supertype. In [2], it is not mentioned which semantic constraints are needed for exceptions, but it seems straightforward that it is unexpected behavior if exceptions can be thrown in more situations than in the supertype.

The criterion we use for the desugaring of signals clauses in different behavior blocks that are connected with `also` or `and` is therefore: a subtype may not allow exceptions that were not allowed in the supertype and the condition under which an exception may be thrown may only be strengthened.

**Desugaring** In JML, the signals clauses in different behavior blocks connected with the keyword `also` are desugared using `&&` (see [5], Figure 14). We propose keeping this desugaring. In this way, it is ensured that a subtype cannot throw exceptions that are not mentioned in the supertype. Moreover, the condition under which an exception may be thrown can only be strengthened.

The desugaring of

```
public behavior
  signals (AE) a;
also
public behavior
  signals (BE) b;
```

is then

```
public behavior
  signals (Exception e)
    ((e instanceof AE) && a) &&
    ((e instanceof BE) && b);
```

This means:

- if `BE` is the same as `AE`: `e` can be thrown when "e is an instance of `AE`" and "a and b are true"
- if `BE` is a subtype of `AE`: `e` can only be thrown when "e is an instance of `BE`" and "a and b are true"<sup>6</sup>
- if `BE` is not a subtype of `AE` and `AE` not of `BE`: `e` cannot be thrown

**The Meaning of Behavior Blocks Connected with `also` and Containing Requires Clauses** The meaning of two behavior blocks connected with `also`, both having a requires clause can be derived by combining subsections 3.2 and 3.4. The following code

```
public behavior
  requires r;
```

---

<sup>6</sup> The symmetric situation is again omitted.

```

    signals (AE) a;
also
public behavior
    requires s;
    signals (BE) b;

```

can be interpreted as

```

public behavior
    requires r || s;
    signals (Exception e)
        ( r && !s ==> (e instanceof AE) && a ) &&
        (!r && s ==> (e instanceof BE) && b) &&
        ( r && s ==>
            (e instanceof AE) && a &&
            (e instanceof BE) && b
        );

```

This means that with two behavior blocks connected by `also`, the implementation has to adhere to the signals clauses in each block whose requires clause evaluates to true in the pre-state; in general this can be the signals clauses of zero, one or both behavior blocks.

### 3.5 Multiple Signals Clauses in Different Behavior Blocks Connected With `and`

The `and` keyword, can among others be used to connect the behavior block of a method in a subtype with the behavior block of the corresponding method in the supertype. To enforce behavioral subtyping, we suggest that conjoinable specification, specification that is connected with the keyword `and` (see [5], subsection 3.3), is desugared as follows:

```

public behavior
    requires r;
    signals (AE) a;
also
public behavior
    requires s;
    signals (BE) b;

```

together with<sup>7</sup>

<sup>7</sup> Conjoinable specification has no header, so there cannot be a requires, when or measured clause (see [5], subsection 2.3).

```

and
public behavior
  signals (CE) c;

```

becomes

```

public behavior
  requires r;
  signals (Exception e)
    (e instanceof AE) && a &&
    (e instanceof CE) && c;
also
public behavior
  requires s;
  signals (Exception e)
    (e instanceof BE) && b &&
    (e instanceof CE) && c;

```

If there are multiple signals clauses in some behavior block, they first have to be combined as described in subsection 3.3. The suggested desugaring for conjoinable specification is different from the current JML desugaring ([5], Figure 10), in which the clauses from the conjoinable specification are inserted in the existing specification, but the result remains the same because the desugaring of multiple signals clauses in one behavior blocks is also different.

The meaning of this desugaring is as follows:

- If only **r** is true in the pre-state, exceptions may be thrown, but only if they are instances of both **AE** and **CE**, and **a** and **c** are satisfied.
- If only **s** is true in the pre-state, exceptions may be thrown if they are instances of both **BE** and **CE**, and **b** and **c** are satisfied.
- If both **r** and **s** are true in the pre-state, exceptions may be thrown if they are instances of **AE**, **BE** and **CE**, and **a**, **b** and **c** are true.

Again, the proposed desugaring prohibits that a method in a subtype throws more exceptions than the corresponding method in the supertype.

### 3.6 Desugaring Normal and Exceptional Behavior Blocks

For the normal and exceptional behavior blocks, we again reuse the current JML desugaring ([5], Section 3.2).

**Normal Behavior** A normal behavior block

```
public normal_behavior
  requires r;
  ensures v;
```

is desugared to

```
public behavior
  requires r;
  ensures v;
  signals (Exception e) false;
```

This means that no exception can be thrown when `r` is satisfied in the pre-state.

**Exceptional Behavior** The exceptional behavior block

```
public exceptional_behavior
  requires r;
  signals (AE) a;
```

is desugared to

```
public behavior
  requires r;
  ensures false;
  signals (AE) a;
```

This means that the method cannot return normally when `r` is satisfied in the pre-state, so it has to end by throwing an exception.

## 4 Conclusion

In this paper, we discussed the semantics of signals clauses in JML method specifications. In section 2 we explained that the current JML desugaring has some strange effect: it allows any exception to be thrown that is not mentioned in the signals clause.

In section 3, we suggested a new semantics for the signals clause. The most important change is that only the exceptions that are mentioned may be thrown and this only under the specified condition. This is done by using the `&&` operator instead of the `==>` operator between (`e instanceof SomeException`) and the given condition.

Another change is that multiple signals clauses in one behavior block are connected with `||` instead of `&&`, making them correspond to typical uses of signals clauses. For more complex usages of the signals clause (`requires`, `also`, `and`, `normal_behavior` and `exceptional_behavior`) the new desugaring is also given. Often, this desugaring is the same as the one currently used in JML, but the meanings differ because of the changed desugaring of the basic element, the single signals clause. While writing the desugaring of multiple signals clauses in different behavior blocks connected by `also` or `and`, we kept in mind the following criterion: a subtype may not allow exceptions that were not allowed in the supertype and the condition under which an exception may be thrown may only be strengthened. The desugaring given in this paper ensures that behavioral subtyping is enforced. In general, the new semantics is more intuitive and therefore easier to understand and to reason about.

## References

1. Ken Arnold, James Gosling, and David Holmes. *The Java Programming Language Third Edition*. Addison-Wesley, Reading, MA, third edition, 2000.
2. Krishna Kishore Dhara and Gary T. Leavens. Forcing behavioral subtyping through specification inheritance. Technical Report 95-20c, Department of Computer Science, Iowa State University, Ames, Iowa, 50011, December 1997. Also in Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, 1996, pp. 258–267. Available by anonymous ftp from [ftp.cs.iastate.edu](ftp://ftp.cs.iastate.edu), and by e-mail from [almanac@cs.iastate.edu](mailto:almanac@cs.iastate.edu).
3. James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification Second Edition*. The Java Series. Addison-Wesley, Boston, Mass., 2000.
4. Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary design of JML: A behavioral interface specification language for Java. Technical Report 98-06p, Iowa State University, Department of Computer Science, August 2001. See [www.cs.iastate.edu/~leavens/JML.html](http://www.cs.iastate.edu/~leavens/JML.html).
5. Arun D. Raghavan and Gary T. Leavens. Desugaring JML method specifications. Technical Report 00-03c, Iowa State University, Department of Computer Science, August 2001.