# JavaCheck: A Domain Specific Language for the static analysis of Java code

**Sara Pérez-Soler, Juan de Lara**
**Universidad Autónoma. Madrid, Spain**
**e-mail: {sara.perezs, juan.delara}@uam.es**

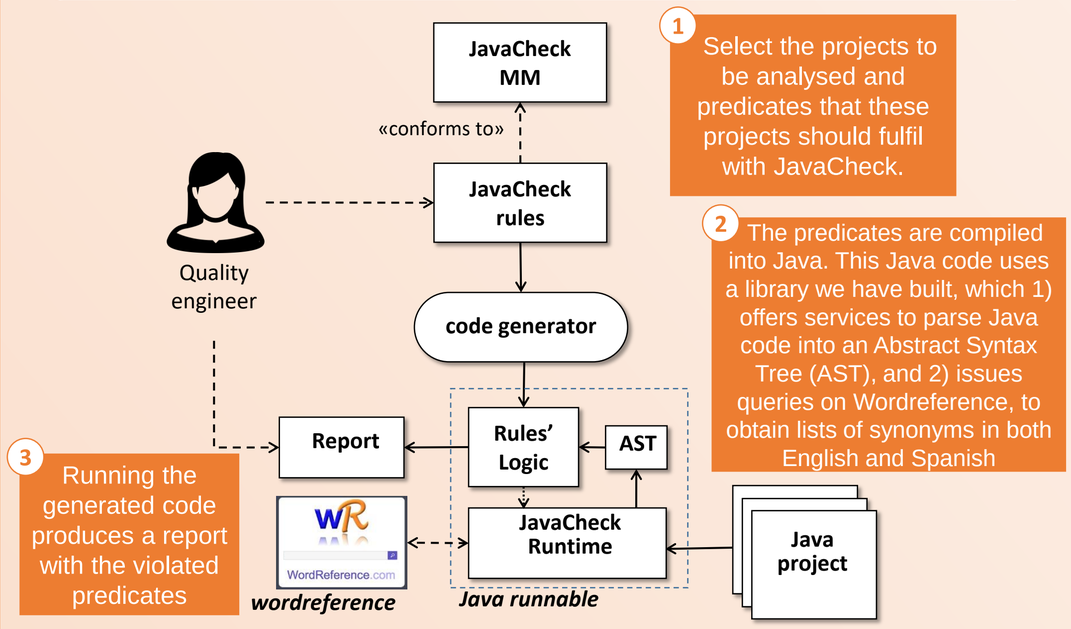
## What is JavaCheck?

- JavaCheck is a domain specific language (DSL) to express predicates to be evaluated over the source code of Java projects.
- Predicates can express general quality properties, style guidelines, project-specific guidelines, application-specific checking or smells of possible errors.
- The DSL has been created using Model-based technology (EMF and Xtext) and is integrated within Eclipse.

```
1 Equals: Method satisfy name="equals" and return type=Primitive.boolean and
2         parameter size=1 types=["Object"];
3 HashCode: Method satisfy name="hashCode" and return type=Primitive.int and parameter size=0;
4
5 all Class which have { one Method in Equals }
6     satisfy have { one Method in HashCode };
```

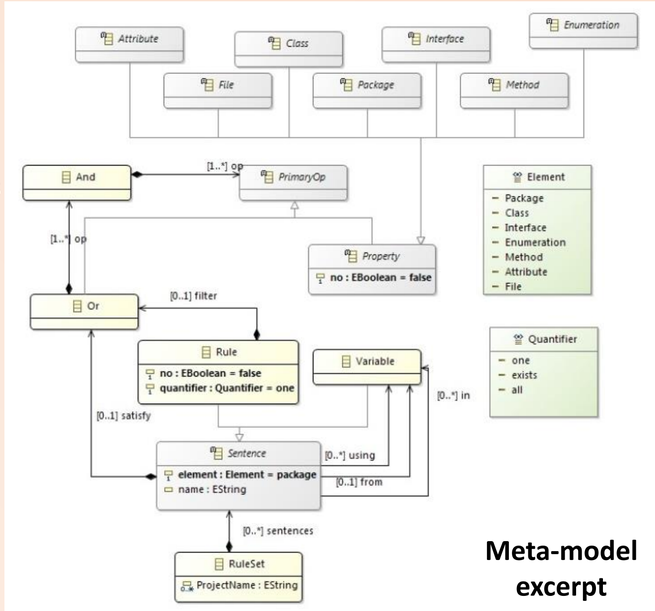
- The first sentence collects all methods named equals with a parameter of type Object and return type boolean, in a collection variable named Equals.
- *HashCode* is a collection that contains all methods named *hashCode*, without parameters and integer return type.
- Lines 5-6 show a rule that checks that all classes with one method in the Equals collection also have one method in *HashCode* collection

## Approach




## JavaCheck structure

- **RuleSet** is the root class, and contains:
  - A list of project names to be checked
  - A set of sentences to check on them.
- There are two types of sentences:
  - the rules that will be evaluated.
  - intermediate variables to store collections of elements that have some properties.
- Sentences have:
  - Type element, that can be **File**, **Package**, **Interface**, **Class**, **Enum**, **Method** or **Attribute**.
  - A clause that needs to be satisfied.
- Rules also:
  - Have a quantifier (**all**, **exist** or **one**).
  - Have a filter.
  - Can reference a collection.
- The **satisfy** and the **filter** clauses contains all properties that the element must comply with.




**Meta-model excerpt**

**Sentence structure**

Collections of objects

```
1 Equals: Method satisfy name="equals" and return type=Primitive.boolean and
2         parameter size=1 types=["Object"];
3 HashCode: Method satisfy name="hashCode" and return type=Primitive.int and parameter size=0;
4
5 all Class which have { one Method in Equals }
6     satisfy have { one Method in HashCode };
```

Rule

```
1 Equals: Method satisfy name="equals" and return type=Primitive.boolean and
2         parameter size=1 types=["Object"];
3 HashCode: Method satisfy name="hashCode" and return type=Primitive.int and parameter size=0;
4
5 all Class which have { one Method in Equals }
6     satisfy have { one Method in HashCode };
```

Type element
Quantifier
Collection reference
Satisfy clause
Filter clause

## JavaCheck: example

Figure 1 shows a JavaCheck file containing:

- The name of project(s) to be analysed (line 1). A "*" takes all projects in the workspace.
- The sentences to be checked.

```
1  Projects Name: *;
2
3  all Attribute
4      which is not modified with [static and final]
5      satisfy is modified with [private or protected];
6
7  all Method satisfy JavaDoc @parameter  @return;
8
9  one Class satisfy name like "User", English and have {
10     one Attribute satisfy name="address"
11 };
12
13 all Class which is modified with [abstract] satisfy is superclass;
```

All attributes that are not static and final (i.e., all attributes that are not constant), must be private or protected

Every method must have a JavaDoc comment with @parameter and @return tags

The project has one class named User or a synonym (in English), and this class must have an attribute named address

all abstract classes have some children class

Figure 2 shows a report produced by the last rule of Figure 1. The report is a text file showing:

- If the rule is met or not (in this case it is not).
- All the elements that pass and violate the rule.

**Figure 2**

```
1  all class which modifiers: [ (abstract) ] satisfy is superclass [1..*]
2  Checked.....ERROR
3  PASS:
4    These elements do not satisfy is superclass [1..*]:
5    – In file D:\Workspace\Evaluate\src\abstractClass\Plane.java the class Plane (line: 3)
6
7  FAIL:
8    These elements satisfy is superclass [1..*]:
9    – In file D:\Workspace\Evaluate\src\abstractClass\Element.java the class Element (line: 3)
10     is super of:
11     In file D:\Workspace\Evaluate\src\restOfClass\Point.java the class Point (line: 5)
```