http://c4j.vksi.de

VKSI Verein der Karlsruher Software-Ingenieure

Overview
Ease of use
Powerful
Six principles
Examples
Syntax reference
Inheritance of contracts
Installation
Configuration
Guidelines

Download C4J 6.0.0
C4J Eclipse Plugin

Download C4J 2.7.5

Overview

Contracts for Java (C4J) is a *Contracts* (from Design by Contract, see Wikipedia DbC definition) framework for Java 1.6 and later. The primary goal for C4J is ease of use. Contracts are about design and quality, aspects of programming that a lot programmers don't spend enough time and energy on.
Therefore a Contracts framework must be simple and painless to use. At the same time the framework must be powerful.
C4J is simple *and* powerful.

We are not going to try to convince any readers that Contracts are indeed a very powerful design and quality assurance technique, so if you are not already convinced of that, please follow the links above and you may be convinced to try this tool out! These are our favorites though:

- *Contracts* consist of *preconditions* and *postconditions*, which are systematically defined method by method.
- The *class invariant* ensures the *DRY principle (Don't Repeat Yourself)* for contracts by defining all those assertions, which must be fulfilled at any visible state of an object, only once.
- Contracts can be linked to classes and interfaces.
- Contracts are inherited by extending a class or by implementing an interface which is guarded by a contract.
- To be able to define meaningful contracts you are forced to split lengthy methods into small, well defined, methods with a single responsibility.
- As your contracts are checked at runtime, your classes are validated against the *real* usage of your application, not against some test cases that may not even be real use cases.
- If you are dealing with legacy code that you are afraid of refactoring, *external* contracts are perfect to add to existing code with no risk involved.

What is the effect of contracts in software development? Example: Implementing the `equals()` method in Java.


```
public boolean equals(Object obj)
```

Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

Applying the equals-Contract to Eclipse Source Code and launching Eclipse reveals Problems with `equals`:

13:17:04,093 ERROR [main]
de.vksi.c4j.internal.runtime.ContractErrorHandler: Contract Violation in post-condition.
java.lang.AssertionError: is consistent with hashCode (post-condition)
    at de.andrena.c4j.object.ObjectSpecContract.assertThat(ObjectSpecContract.java:65)
    at de.andrena.c4j.object.ObjectSpecContract.equalsPostConditionNonNull(ObjectSpecContract.java:59)
    at de.andrena.c4j.object.ObjectSpecContract.equals(ObjectSpecContract.java:46)
    at org.eclipse.jface.viewers.StructuredSelection.equals(StructuredSelection.java:149)

Root cause: **`equals`** has been overridden without overriding **`hashCode`**

13:17:31,747 ERROR [org.eclipse.jdt.internal.ui.text.JavaReconciler]
de.vksi.c4j.internal.runtime.ContractErrorHandler: Contract Violation in post-condition.
java.lang.AssertionError: is symmetric (post-condition)
    at de.andrena.c4j.object.ObjectSpecContract.assertThat(ObjectSpecContract.java:65)
    at de.andrena.c4j.object.ObjectSpecContract.equalsPostConditionNonNull(ObjectSpecContract.java:54)
    at de.andrena.c4j.object.ObjectSpecContract.equals(ObjectSpecContract.java:46)
    at org.eclipse.jdt.internal.launching.JREContainer$RuleEntry.equals(JREContainer.java:176)

Root cause: A container object may be equal to an array (by comparing its content entry per entry),
but an array is not equal to a container object (only to an array).

http://c4j.vksi.de

VKSI Verein der Karlsruher Software-Ingenieure

Overview
Ease of use
Powerful
Six principles
Examples
Syntax reference
Inheritance of contracts
Installation
Configuration
Guidelines

Download C4J 6.0.0
C4J Eclipse Plugin

Download C4J 2.7.5

Overview

Contracts for Java (C4J) is a *Contracts* (from Design by Contract, see Wikipedia DbC definition) framework for Java 1.6 and later. The primary goal for C4J is ease of use. Contracts are about design and quality, aspects of programming that a lot programmers don't spend enough time and energy on.
Therefore a Contracts framework must be simple and painless to use. At the same time the framework must be powerful.
C4J is simple *and* powerful.

We are not going to try to convince any readers that Contracts are indeed a very powerful design and quality assurance technique, so if you are not already convinced of that, please follow the links above and you may be convinced to try this tool out! These are our favorites though:

- *Contracts* consist of *preconditions* and *postconditions*, which are systematically defined method by method.
- The *class invariant* ensures the *DRY principle (Don't Repeat Yourself)* for contracts by defining all those assertions, which must be fulfilled at any visible state of an object, only once.
- Contracts can be linked to classes and interfaces.
- Contracts are inherited by extending a class or by implementing an interface which is guarded by a contract.
- To be able to define meaningful contracts you are forced to split lengthy methods into small, well defined, methods with a single responsibility.
- As your contracts are checked at runtime, your classes are validated against the *real* usage of your application, not against some test cases that may not even be real use cases.
- If you are dealing with legacy code that you are afraid of refactoring, *external* contracts are perfect to add to existing code with no risk involved.

Your feedback is highly appreciated!