Advances in Programming Languages
APL7: JML — The Java Modeling Language

David Aspinall
(slides by Ian Stark)

School of Informatics
The University of Edinburgh

Monday 2 February 2009
Semester 2 Week 4
This is the middle of three lectures about some techniques and tools for formal verification, specifically:

- Hoare logic
- JML: The Java Modeling Language
- ESC/Java 2: The Extended Static Checker for Java
Outline

1. Introduction
2. Samples of JML
3. JML Tools
Hoare Logic (recap)

- Hoare assertions \( \{P\} \ C \ \{Q\} \) state that if *precondition* \( P \) holds before running code \( C \) then *postcondition* \( Q \) will hold afterwards.

- Assertions \( \vdash \{P\} \ C \ \{Q\} \) can be derived using Hoare *rules*; they may also be tested against a *semantics* \( \models \{P\} \ C \ \{Q\} \).

- This allows logical reasoning about program behaviour: notably in formal *specification* and *verification*.

- Hoare assertions are widely used in tools and languages for formal methods. (e.g. Praxis SPARK Examiner)

- Assertions may be strengthened to *contracts* for code, placing obligations on both caller and called. (e.g. Eiffel)
Model-based specification

Modeling (sic) is an abstraction technique for system design and specification.

A *model* is a representation of the desired system.

A *formal model* is one that has a precise description in a formal language.

A model differs from an implementation in that it might:

- capture only some aspects of the system (e.g., interfaces);
- be partial, leaving some parts unspecified;
- not be executable.

An implementation of the system can be compared to the model. Sometimes the model is iteratively refined to give the implementation.

Sample applications of modeling in computer software development:

- **VDM** the *Vienna Development Method*.
- **B** the *B language* and *B method*.
- **Extended ML** the extension of Standard ML with specifications.
- **OCL** the *Object Constraint Language* extension of UML.
The Java Modeling Language, JML, combines model-based and contract approaches to specification.

Some design features:

The specification lives close to the code
Within the Java source, in *annotation comments /*@...@*/

Uses Java syntax and expressions
Rather than a separate specification language.

Common language for many tools and analysis
Tools add their own extensions, and ignore those of others.

Web site: jmlspecs.org
public class Account {
    private int credit;

    /** @requires credit > amount && amount > 0; */
    @ensures credit > 0 && credit == \old(credit) − amount;
    @*/
    public int withdraw(int amount) {
        ...
    }
}

JML conditions combine logical formulae (&&,==) with Java expressions (credit, amount). Expressions must be pure: no side-effects.
public class Account {
    private int credit;

    /*@ requires credit > amount && amount > 0;
        @ ensures credit > 0 && credit == \old(credit) – amount;
        @ signals (RefusedException) credit == \old(credit);
        @*/
    public int withdraw throws RefusedException (int amount) {
        ...
    }
}

Where ensures speaks about normal termination, signals specifies properties of the state after exceptional termination.
The search routine requires that array contents be sorted on entry. This would, for example, be necessary if it used binary chop to locate value.
public class IntArray {
    public int[] contents;

    /** @ invariant (\forall int i,j;
       @ 0<i && i<j && j<contents.length;
       @ contents[i] <= contents[j]);
       @*/

    /** @ ensures contents[result] == value || result == -1
       @*/
    public int search (int value) {
        ... }
}

Now contents must be sorted whenever it is visible to clients of IntArray.
An assumption may help a static analysis tool.

An assertion must always be checked.
public class IntArray {
    public int[] contents;

    /** @model int total;
        @represents total = arraySum(contents)
        @*/

    /** @ghost int cursor;
        @set cursor = contents.length / 2
        @*/

    ...
}

A *model* field represents some property of the model that does not appear explicitly in the implementation.

A *ghost* field is a local variable used only by other parts of the specification.
Specifications may refer to *model methods* and even entire *model classes* to represent and manipulate desired system properties.

JML provides specifications for the standard Java classes, as well as a library of model classes for mathematical constructions like sets, bags, integers and reals (i.e. of arbitrary size and precision).
JML tools: running and testing

JML annotations can be used to drive various runtime checks.

**jmlc** is a compiler which inserts runtime tests for every assertion; if an assertion fails, an error message provides static and dynamic information about the failure.

**jmlunit** creates test classes for JUnit based on preconditions, postconditions and invariants. These automatically exercise and test assertions made in the code.

JML annotations also provide formal documentation:

**jmldoc** generates human-readable web pages from JML specifications, extending the existing javadoc tool.
The ESC/Java 2 framework carries out a range of static checks on Java programs. These include formal verification of JML annotations using a fully-automated theorem prover. Controversially, the checker is neither sound nor complete: it warns about many potential bugs, but not all actual bugs. This is by design: the aim is to find many possible bugs, quickly.

The LOOP tool also attempts to verify JML specifications. Some can be done automatically; where this is not possible it provides proof obligations for the interactive PVS theorem prover.

The JACK tool generates proof obligations from JML annotations on Java and JavaCard programs; these can then be tackled with a variety of automatic and semi-automatic theorem provers.
More tools

*Key* is dynamic logic tool with a JML front end.

*Krakatoa* is another verification tool accepting JML.

*Jive* the Java Interactive Verification Environment, uses JML.

*Houdini* will suggest JML annotations and test them with ESC/Java.

*Daikon* analyses program runs to suggest likely JML invariants.

Finally:

*Spec#* is to C# as ESC/Java 2 is to Java.
The Java Modeling Language

- JML combines model-based and contract specification
- Annotations within code: \texttt{requires}, \texttt{ensures}, \ldots
- Provides \textit{model} fields, methods and classes.
- Common language for many tools: runtime checks, static analyses, etc.
Homework

The next lecture will be on ESC/Java 2.

Meanwhile you should try using JML.

- Install some JML tools.
- Develop a simple recipe card application using JML.
  - design a few classes for representing ingredients, amounts and recipes;
  - start specifying gradually: add simple pre-conditions to methods;
  - write tests by composing methods;
  - see where your code needs additional requires, ensures or (sometimes) assume annotations;
  - consider useful object invariants to constrain fields.