Runtime Assertion Checking Using JML

Roy Patrick Tan
Department of Computer Science
Virginia Tech
660 McBryde Hall, Mail Stop 0106
Blacksburg, VA 24061, USA
rtan@vt.edu

```
public class IntMathOps3 {

  //@ requires y >= 0;
  public static int isqrt(int y)
  {
    return (int) Math.sqrt(y);
  }
}
```

Figure 1: A simple specification requiring the parameter to be non-negative

Categories and Subject Descriptors
D.2.4 [**Software Engineering**]: Software/Program Verification—*programming by contract, assertion checkers, class invariants*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*pre- and post-conditions, invariants, assertions*; D.2.3 [**Software Engineering**]: Coding Tools and Techniques—*object-oriented programming*; D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids*; D.3.2 [**Programming Languages**]: Language Classifications—*JML*

General Terms
Languages

Keywords
JML, Java, run-time checking, design by contract

1. INTRODUCTION
JML, the Java Modeling Language, is a language that allows programmers to specify the detailed design of Java programs. A software developer can use JML to add specifications such as method preconditions and postconditions, and class invariants to clearly indicate their correct behavior.

SAVCBS '03 Helsinki, Finland

```
//@ model import org.jmlspecs.models.*;

public class IntMathOps2 {

 /*@ public normal_behavior
   @   requires y >= 0;
   @   assignable \nothing;
   @   ensures -y <= \result && \result <= y;
   @   ensures \result * \result <= y;
   @   ensures
   @        y < (Math.abs(\result) + 1)
   @            * (Math.abs(\result) + 1);
   @*/
  public static int isqrt(int y);

}
```

Figure 2: A complete specification

JML annotations are written in the form of specially commented sections of the code. Figure 1 shows a lightweight JML specification for an integer square root method, requiring that the input be non-negative [4]. While specifications can be as simple as that, JML has sophisticated features that allow programmers to write full, abstract, model-based specifications. Figure 2 shows a complete specification for the integer square root method [4].

2. THE JML COMPILER
Jmlc, the JML compiler, is one of several tools support the JML notation [2]. Jmlc takes JML annotated source files and compiles preconditions, postconditions, invariants, and history constraints into bytecode that checks these specifications at runtime, making jmlc an ideal design-by-contract tool.

Runtime monitoring of contract assertions has many well known advantages. One particularly useful feature of runtime assertion checking is that an error is likely to be found at the point of contract violation; the error does not propagate such that when it is detected, the point of failure is in correctly implemented code.

The lack of assertions in early versions of the Java language, and the lack of design-by-contract checking of preconditions, prostconditions, and invariants in the current version (JDK

1.4) has led to many runtime assertion checking tools for Java, such as iContract [3], and Jass [1].

However, JML has features not found in other runtime checkers. For example, JML's facility for specification-only fields and methods allows programmers to create specifications using an abstract model of the object's state. The JML compiler allows programmers to use formal specification as a practical tool for debugging and testing software components.

3. CONCLUSIONS

Any developer of Java components and applications may benefit from the use of JML tools. JML has an easy adoption path, since classes and methods need not have full specifications. The programmer can begin by putting the odd precondition or postcondition check and then code in more complex specifications as his proficiency in the language improves. JML has been put to practical use in industry. Particularly, nearly all the API of Java Card, a dialect of Java for use in smart cards, has been specified in JML [5].

JML was originally developed in Iowa State University by Gary Leavens and his students. It is now an open source project with developers from all over the world actively contributing to improve the tools and the language. The JML homepage can be found at http://jmlspecs.org.

Acknowlegements

We gratefully acknowledge the financial support from the National Science Foundation under the grant CCR-0113181. Any opinions, conclustions or recommendations expressed in this paper do not necessarily reflect the views of the NSF.

4. REFERENCES

[1] D. Bartetzko, C. Fischer, M. Mller, and H. Wehrheim. Jass - java with assertions. In K. Havelund and G. Rosu, editors, *Electronic Notes in Theoretical Computer Science*, volume 55. Elsevier, 2001.

[2] L. Burdy, Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G. T. Leavens, K. Rustan, M. Leino, and E. Poll. An overview of JML tools and applications. In *Eighth International Workshop on Formal Methods for Industrial Critical Systems (FMICS '03)*, volume 80, pages 73–89. Elsevier, 2003.

[3] R. Kramer. iContract — the Java design by contract tool. In *TOOLS 26: Technology of Object-Oriented Languages and Systems*, pages 295–307, 1998.

[4] G. T. Leavens, A. L. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. Technical Report 98-06v, Department of Computer Science, Iowa State University, May 2003.

[5] E. Poll, J. van den Berg, and B. Jacobs. Specification of the JavaCard API in JML. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, *Fourth Smart Card Research and Advanced Application Conference (CARDIS'2000)*, pages 135–154. Kluwer Acad. Publ., 2000.