A Process for the Specification of Core JDK Classes

Ralph Hyland - 07270861

August 25, 2009

1 Project Summary

Supervisor: Dr. Joe Kiniry

Subject Area: Reliable Software Engineering

Pre-requisite: A good understanding of object-oriented programming in Java, basic understanding of the Java Modeling Language (JML), contracts, and unit testing

Co-requisite (things that must be learned along the way): Writing specifications in JML, using the JML tool suite and ESC/Java2, advanced unit testing, Design by Contract (DBC)

Subject Coverage: Class specification and unit testing, static analysis

Project Type: Analysis, design, and implementation

Hardware and Software: PC, laptop or workstation capable of running Java and Eclipse

2 Background

To understand complex APIs like those available for Java, (e.g., the core classes in the java.lang, java.io, java.util packages etc.), one needs a precise description of the API's behavior. While natural language documentation, like that found in Sun Microsystems's Javadocs for JDK 5, has improved API documentation over the past ten years, such "specifications" are vague, imprecise, and error-prone.

The Java Modeling Language (JML) is the de facto standard for writing precise specifications of Java programs. JML is used by many courses around the world (including

several at UCD), applied at several companies, and is supported by many tools. JML is an extension of Java that permits one to write assertions, like invariants and pre- and post-conditions, about Java classes and interfaces. Various tools support using such specifications to, among other things, generate runtime assertion checks, generate Javadoc-like documentation, generate method-level unit tests, and check that method implementations fulfill their specifications.

3 The Problem

New specifications for core JDK classes (e.g., java.lang.String) are typically written "on-demand," as they are necessary for the testing and verification of new programs that exercise previously unused portions of the (very large) Java platform API. In the past, such specifications were written by experts who essentially "translated" Javadoc's English into JML.

Unfortunately, such specifications are rarely generally useful because (a) they do not target a particular use-case (e.g., run-time checking vs. verification), (b) they are not rigorously tested in any way, and (c) they are based upon erroneous data in the first place (i.e., Javadocs).

4 Addressing the Problem

Recently a new specification writing process has been introduced by Dr. David Cok. This process involves writing new specifications and complementary specification-centric unit tests that target a particular use-case. The purpose of this project is to evaluate and extend this process by incorporating existing available comprehensive unit tests suites, like those written by Sun Microsystems to certify new implementations of the Java programming language.

A new tool-assisted process is necessary to assist the JML community in:

- identifying which specifications are necessary
- understanding how to properly write such specifications
- objectively evaluating a specifications correctness through run-time and static checking
- providing a feedback mechanism from the tools and the community as to the quality and completeness of a specification
- objectively measuring a specification's utility

5 Evaluating the Solution

In order to evaluate the proposed solution, a number of goals are outlined below ranging from mandatory to exceptional. These goals attempt to portray the extent of the work required from start to finish in this project and show the various aspects involved in the area of verification-centric software engineering.

5.1 Mandatory Goals:

- Review and refine knowledge of JML already gained in Dr. Kiniry's Software Engineering modules.
- Learn the verification-centric software engineering process (the KindSoftware Process), as specified by Dr. Kiniry.
- Learn the basic use of the JML compiler, unit testing with JML-jUnit, and documentation generation with jmldoc.
- Obtain JDK unit test suites from Classpath Project and Sun.
- Setup local automated nightly or triggered runtime- and static checking-based ("hybrid unit testing") testing of JDK core classes.
- Evaluate existing JML and ESC/Java2 JDK specifications against this test suite.
- Write new JML specifications and complementary test suites for at least three JDK core classes.
- Document and refine through experience the process by which one should write such specifications incorporating hybrid unit testing.

5.2 Discretionary Goals:

- Write JML specifications and complementary test suites for a core JDK package.
- Incorporate other static analysis tools into hybrid test suite.
- Contribute to the design and development of new Java annotation to help with specification evaluation and tracing a specification's history.
- Augment the JML Hudson system to automatically generate the objective evaluations of specification correctness and utility.

5.3 Exceptional Goals:

- Co-author a paper with Dr. Kiniry on this specification-centric process.