Hi. This is an guide to software verification with the Java Modeling Language ("JML"). The goal is to provide a hands-on introduction to software verification and to get comfortable with the JML framework through toy examples.

This document is for the curious java developer that wants to dip their toes in the world of software verification and who is comfortable learning by doing, without the broader theoretical context. This document is best thought of as a stepping stone to deeper and more comprehensive resources .

What we will cover

- Implementation of motivated but unrealistically simple examples
- Basic JML syntax and keywords

What we will not cover

- General features of JML (We will only cover the minimal information required to implement our solutions. We value demonstration over explanation).
- Theoretical aspects of software verification

Prerequisites

- basic knowledge of Java (syntax, types, OOP basics)
- general knowledge of programming concepts
- OpenJML 21-0.8 installed

Background on JML

JML ("Java Modeling Language") is a specification language for java. It is written in java comments, and is used to express *semantic properties* of programs (i.e, properties about the behaviour of the programs). We can use a program verification tool, like OpenJML, to *verify* a *contract* written in JML. OpenJML will read java code annoted with JML expressions and try to prove that the code satisfies the conditions expressed in those annotations. For example, someone might write JML code to assert that a method always returns a positive integer. We will refer to JML code used to express the properties we would like to verify as a "specification". In this tutorial we will use the tool **OpenJML** to verify our specifications.

PROF

Even if you're program *does* satisfy the conditions expressed in its JML code, there's no guarantee that a program verification tool will be able to prove that it does. However, if your program verification tool *can* prove that your program satisfies those conditions, you can be absolutely certain that the program satisfies those properties.

It can be useful to compare and contrast software verification with unit testing:

Unit testing	**Software verificaion**

Unit testing	Software verificaion
(usually) involves writing many tests to verify a single property	involves writing a single specification of a property, which is then handed over to a software verifier for verification.
involves *executing* the code to be tested with selectd inputs and then inspecting outputs after execution.	involves statically verifying properties of the code (the code is not executed)
is *empricial* (i.e, you check if your code behaves correctly on *some* inputs, and conclude it behaves correctly on *all* inputs)	is *logical* (i.e, you get a logical proof that your code behaves correctly *on all possible inputs*)
is relatively simple and quick to use	is relatively complex and slow to use

Bottom line

If you can afford the extra complexity and time, and you really need to guarantee your code works correctly, software verification can be very useful. For this reason, it is mostly used in high-stakes applications: finance, information security, aerospace engineering, etc.

Organization

This document is broken into three tasks, to be done in order.

1. specify properties of methods
2. specify properties of classes in general
3. specify properties of interface implementations.

Each task will involve working towards some piece of formally verified java code. We will get there through a sequence of numbered steps. Almost all of these steps will involve writing code. The code will be presented *cumulatively*, with each step adding to what we've done so far. By the end of a task, we will have some code that we can feed into our JML verifier to prove correctness. We will explain our additions as we go. You should not expect verification to work until all the steps have been finished.

PROF

Most new vocabulary will be defined as it is introduced, but we will link to glossary entries whenever we want to use a bit of jargon without immediately defining it.

Conventions

Code

Code will be displayed using the markdown code style `like this`.

New lines of code (i.e, lines of code to be added at the current step in the task) will be highlited like this:

```
here is an old line
here is a new
here is another new line
here is another old line
```

Information boxes

Warnings will look like this.

!!! warning
This is a warning box.

Notes will look like this.

!!! note
This is just a note.

Tips will look like this

!!! tip
Helpful tip goes here.

Danger messages will look like this
!!! danger
Be careful with this!

Information messages will look like this
!!! info
Here's some info.