Assertions in Java (JDK 1.4)

Jarret Raim

updated by Glenn Blank

What is an assertion?

- An *assertion* is a statement in Java that enables you to test your assumptions about your program.
- Each assertion contains a boolean expression that you believe will be true when the assertion executes.
- By verifying that the boolean expression is indeed true, the assertion confirms your assumptions about the behavior of your program, increasing your confidence that the program is free of errors.

Why use assertions?

- Programming by contract
- Pre-conditions
  - Assert precondition as requirement of client
- Post-conditions
  - Assert post-condition as effect of client method

Simple Assertion Form

- The assertion statement has two forms
- The first is:

```
assert Expression1 ;
```

- Where *Expression1* is a boolean expression
- When the system runs the assertion, it evaluates *Expression1* and if it is false throws an AssertionError with no details

Complex Assertion Form

- The second form of the assertion statement is:

```
assert Expression1 : Expression2 ;
```

- where:
  - *Expression1* is a boolean expression
  - *Expression2* is an expression that has a value
  - It cannot invoke of a method that is declared `void`

Complex Assertion Form, cont.

- Use the second version of the assert statement to provide a detailed message for the AssertionError
- The system passes the value of *Expression2* to the appropriate AssertionError constructor, which uses the string error message
- The purpose of the message is to communicate the reason for the assertion failure
- Don’t use assertions to flag user errors—why not?

When an Assertion Fails

- Assertion failures are labeled in stack trace with the file and line number from which they were thrown
- Second form of the assertion statement should be used in preference to the first when the program has some additional information that might help diagnose the failure

Compiler Directives

- For the javac compiler to accept code with assertions, use this command-line option:

  -source 1.4

- For example:

```
javac -source 1.4 MyClass.java
```

Performance Problems

- Assertions may slow down execution—why?
- For example, if an assertion checks to see if the element to be returned is the smallest element in the list, then the assertion would have to do the same amount of work that the method would have to do
- So, assertions may be enabled and disabled
- Assertions are, by default, disabled at run-time
- In this case, the assertion has the same semantics as an empty statement

Arguments

- no arguments
  Enables or disables assertions in all classes except system classes
- *packageName*...
  Enables or disables assertions in the named package and any subpackages
- ...
  Enables or disables assertions in the unnamed package in the current working directory
- *className*
  Enables or disables assertions in the named class

Enabling Assertions

- To enable assertions at various granularities, use the -enableassertions, or -ea, switch
- To disable assertions at various granularities, use the -disableassertions, or -da, switch
- Specify the granularity with the arguments that you provide to the switc

Examples

- The following command runs a program, BatTutor, with assertions enabled in only package com.wombat.fruitbat and its subpackages:

```
java -ea:com.wombat.fruitbat... BatTutor
```

- If a single command line contains multiple instances of these switches, they are processed in order before loading any classes.
- For example, the following command runs the BatTutor program with assertions enabled in package com.wombat.fruitbat but disabled in class com.wombat.fruitbat.Brickbat:

```
java -ea:com.wombat.fruitbat...
  -da:com.wombat.fruitbat.Brickbat BatTutor
```

When to use Assertions

- Internal Invariants
  - Many programmers used comments to indicate programming assumptions.

```
if (i % 3 == 0) { ... }
else if (i % 3 == 1) { ... }
else { // We know (i % 3 == 2)
... }
```

When to use Assertions

- Now we can use assertions to guarantee the behavior.

```
if (i % 3 == 0) { ... }
else if (i % 3 == 1) { ... }
else { assert i % 3 == 2 : i; ... }
```

Control Flow

- If a program should never reach a point, then a constant false assertion may be used

```
void foo() {
        for (...) {
            if (...)
            return;
        }
        assert false; // Execution should never get here
}
```

Assertions in C++

- assert() macro
- #include <assert.h>
- assert(expression)
- If expression is 1, nothing happens
- If expression is 0, halt program, and display file name, line number and expression
- #define ndebug 0 //turns off assertion testing
- Compare assertions in C++ and Java?

More Information

- For more information, see:

http://java.sun.com/j2se/1.4/docs/guide/lang/assert.html#usage-conditions