MSc Software Testing
MSc Hugbúnaðar

Fyrirlestrar 37 og 38
Assertion facility in Java

Assertions as a test oracle in random testing before code deployment...
Case Study
Dæmisaga

Reference
http://java.sun.com/developer/Books/javaprogramming/jdk14/javapch06.PDF
Copyright 1994-2007 Sun Microsystems, Inc. All Rights Reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You acknowledge that this software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.
6.1 Assertion basics

- Java assertions provide a mechanism for making additional checks on the code.
- Assertions are like error checks, but they are turned off in deployment.
- An assertion is a conditional expression that evaluates to true (code is working correctly) or false (code has a fault).
- An assertion is easier to write than a RunTimeException.
6.1 Assertion basics

- “A good rule of thumb is that you should use an assertion for exceptional cases that you would like to forget about.”
- Bugs often involve exceptional conditions that “some programmer, somewhere, forget to handle”.
- Handling all exceptions gracefully is unrealistic.
- An assertion is the easiest way of handling an exceptional case.
6.1 Assertion basics

• **Preconditions** should be true at the start of a method.
  – Only use assertions as preconditions if the method is not a public method.

• **Postconditions** should be true at the end of a method.

• **Invariants** should always be true.
  – A speed should not be faster than the speed of light...
6.2.1 Assertion syntax

public class aClass {
    public void aMethod( int argument ) {
        Foo foo = null;
        // ... somehow get a Foo object
        // Now check to make sure we've managed to get one:
        assert foo != null;
    }
}

public class aClass {
    public void aMethod( int argument ) {
        Foo foo = null;
        // ... somehow get a Foo object
        // Now check to make sure we've managed to get one:
        if (!(foo != null)) {
            throw new AssertionError();
        }
    }
}

The two aMethods are roughly equivalent. Using assert, it is easier to write code to deal with the exceptional case.
6.2.1 Assertion syntax

```java
class aClass {
    public void aMethod( int argument ) {
        Foo foo = null;
        // ... somehow get a Foo object
        // Now check to make sure we've managed to get one:
        assert foo != null : "Can't get a Foo, argument="+argument;
    }
}
```

assert expression_1 : expression_2;
- more debugging information -

```java
class aClass {
    public void aMethod( int argument ) {
        Foo foo = null;
        // ... somehow get a Foo object
        // Now check to make sure we've managed to get one:
        if (!(foo != null)) {
            throw new AssertionError(  
                "Can't get a Foo, argument="+argument );
        }
    }
}
```

Dr Andy Brooks 8
6.2.2 Compiling with assertions

- `javac –source 1.4 *.java`
  - enables use of assertions in compilation
  - assertions were introduced from JDK 1.4
    - earlier Java code may have used the keyword `assert` as an identifier

Do not forget to set a path to `javac` in your OS. My Path environment variable under Windows includes: C:\j2sdk1.4.2_15\bin;
6.2.3 Controlling assertions from the command line

- "java –ea ..." to enable run time assertions
  - also "java –enableassertions ..."
- "java –da ..." to disable run time assertions
  - also "java –disableassertions ..."
6.2.7 Catching an assertion failure

public void method() {
    AssertionError ae = null;
    try {
        int a = anotherMethod();
        // ...
        assert i==10; // The assertion
        // ...
    } catch (AssertionError ae2) { // Trapping the assertion failure
        ae = ae2;
        StackTraceElement stes[] = ae.getStackTrace();
        if (stes.length>0) {
            StackTraceElement first = stes[0];
            System.out.println( "NOTE: Assertion failure in "+
                    first.getFileName()+" at line "+first.getLineNumber() );
        } else {
            System.out.println( "NOTE: No info available." );
        }
        throw ae; // Rethrowing the assertion failure
    }
}

Catch assertions only if it is necessary to gather additional information. Any caught assertion must then be thrown to alert the user.
6.3 Assertion examples

Pipes are used to channel the output from one thread into the input of another.

### 6.3.1 Avoiding inconsistent states

```java
import java.io.*;
public class Example {
    private PipedInputStream pin;
    private PipedOutputStream pout;
    private void initializePipe() throws IOException {
        pin = new PipedInputStream();
        pout = new PipedOutputStream( pin );
    }
    public void someMethod() {
        assert (pin==null) == (pout==null) :
            "Warning: pipe is inconsistent" ;
    }
}
```

- The assert checks that both pin and pout are null together.
- If pin is null while pout is non-null, the state is inconsistent.
6.3 Assertion examples

6.3.2 Narrowing the range of states

```java
public void runSimulation() {
    // ...
    assert
        Math.abs( velocity ) < 2000 :
        "Object way too fast! velocity="+velocity;
}
```

- A simple check that in a simulation, an object's velocity does not exceed or equal 2000.
6.3.4 More complicated consistency checks

```
public class EmployeeDatabase()
{
    private Set employees;
    private Map employeeGroups;
    private SortedMap employeeTitles;
    private Set groups;
    private Map groupMemberships;
    private Set projects;
    private List groupDeadlines;
    public void doSomething() {
        // ...
        assert isConsistent() : "Error: inconsistent state!";
    }
    private boolean isConsistent() {
        // check lots and lots of stuff here
        // ...
    }
}
```

- Sometimes it is awkward to write a consistency check as a single expression, so a private boolean method is written.

Having safety checks in one place avoids the need for multiple copies.
Rule: do not use assertions to enforce command-line usage

```java
public class Application {
    static public void main( String args[] ) {
        // BAD!!
        assert args.length == 3;
        int a = Integer.parseInt( args[0] );
        int b = Integer.parseInt( args[1] );
        int c = Integer.parseInt( args[2] );
    }
}

public class App {
    static public void main( String args[] ) {
        if (args.length != 3)
            throw new RuntimeException( "Usage: <progname> a b c" );
        int a = Integer.parseInt( args[0] );
        int b = Integer.parseInt( args[1] );
        int c = Integer.parseInt( args[2] );
    }
}
```
6.4.1 Rules of use

Rule: use assertions to enforce constraints on arguments in **private** methods

```java
private void method( Object required, Object optional ) {
    assert( required != null ) : "method(): required=null";
}
```
/**
 * Sets the refresh interval (which must correspond to a legal frame rate).
 * @param interval refresh interval in milliseconds.
 */

private void setRefreshInterval(int interval) {

    // Confirm adherence to precondition in nonpublic method
    assert interval > 0 && interval <= 1000/MAX_REFRESH_RATE : interval;

    ... // Set the refresh interval
}

http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html
Rule: do not use assertions to enforce constraints on arguments in public methods

```java
public void disconnect( Source source, sink sink ) {
    // BAD!!
    assert source.isConnected( sink ) :
    "disconnect(): not connected "+source+","+sink;
}
```

```java
public void disconnect( Source source, sink sink ) throws IOException {
    if (!source.isConnected( sink )) {
        throw new IOException(
            "disconnect(): not connected "+source+","+sink );
    }
}
```

- A public method should guarantee its interface. Assertions may or may not enforce constraints – it depends if they are enabled in the run time environment.
6.4.1 Rules of use

Rule: do not use assertions to enforce public usage patterns or protocols

public class Connection
{
    private boolean isOpen = false;
    public void open() {
        // ...
        isOpen = true;
    }
    public void close() {
        // BAD!!
        assert isOpen : "Cannot close a connection that is not open!";
        // ...
    }
}

• "... if the Connection class is used publicly, it would not be surprising to find a bug in which someone tried to close a Connection that was not open in the first place."
6.4.1 Rules of use

Rule: do not use assertions to enforce a property of a piece of user-supplied information.

```java
public void processZipCode( String zipCode ) {
    if (zipCode.length() == 5) {
        // ...
    } else if (zipCode.length() == 9) {
        // ...
    } else {
        // BAD!!
        assert false : "Only 5- and 9-digit zip codes supported";
    }
}
```

- Use assertions to enforce internal consistency, not input.

31/10/2007  Dr Andy Brooks  20
Rule: use assertions to enforce internal assumptions about aspects of data structures.

```java
private void showDate( int array[] )
{
    assert( array.length==3 );
}
```

- The private method takes an array of three integers, so an assertion is written as a safety check that the length of the array is three.
6.4.2 What to check for

**Rule: use assertions to check conditions at the end of any kind of method.** (Postconditions)

```java
public class Connection {
    private boolean isOpen = false;
    public void open() {
        // ...
        isOpen = true;
        // ...
        assert isOpen;
    }
    public void close() throws ConnectionException {
        if (!isOpen) {
            throw new ConnectionException(
                "Cannot close a connection that is not open!" );
        }
        // ...
        isOpen = false;
        // ...
        assert !isOpen;
    }
}
```

The asserts might seem redundant, but there is no telling what may happen to the code in the future.
Suppose a class implements a balanced tree data structure. Each public method and constructor should contain “assert balanced;” immediately prior to its return. – e.g. after item insert, delete, ...

// Returns true if this tree is properly balanced
private boolean balanced() {
    ...
}

class invariant from java.sun.com
http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html
Rule: use assertions to check for conditional cases that should never happen.

```java
private int getValue() {
    if (/* something */) {
        return 0;
    } else if (/* something else */) {
        return 1;
    } else {
        return 2;
    }
}
```

```java
public void method() {
    int a = getValue(); // returns 0, 1, or 2
    if (a==0) {
        // deal with 0 ...
    } else if (a==1) {
        // deal with 1 ...
    } else if (a==2) {
        // deal with 2 ...
    } else {
        assert false : "Impossible: a is out of range";
    }
}
```

The public method makes no promises about handling values other than 0, 1, or 2 so it is OK here to use an assert.
public void method() {
    int a = getValue(); // returns 0, 1, or 2
    if (a==0) {
        // deal with 0 ...
    } else if (a==1) {
        // deal with 1 ...
    } else {
        assert a==2 : "Impossible: a is out of range";
        // deal with 2 ...
    }
}
Equivalent implementation 2

```java
public void method() {
    int a = getValue(); // returns 0, 1, or 2
    switch( a ) {
        case 0:
            // deal with 0 ...
            break;
        case 1:
            // deal with 1 ...
            break;
        case 2:
            // deal with 2 ...
            break;
        default:
            assert false : "Impossible: a is out of range";
            break;
    }
}
```
switch examples from java.sun.com

http://java.sun.com/j2se/1.4.2/docs/guide/lang/assert.html

switch(suit) {
    case Suit.CLUBS:
    ...
    break;
    case Suit.DIAMONDS:
    ...
    break;
    case Suit.HEARTS:
    ...
    break;
    case Suit.SPADES:
    ...
    break;
    default:
        assert false : suit;
}

or

switch(suit) {
    case Suit.CLUBS:
    ...
    break;
    case Suit.DIAMONDS:
    ...
    break;
    case Suit.HEARTS:
    ...
    break;
    case Suit.SPADES:
    ...
    break;
    default:
        throw new AssertionError(suit);
}

Offers protection even if assertions are disabled.
Rule: use assertions to check for conditional cases that should never happen, even if you’re really sure they can never happen.

```java
public void method() {
    int a = getValue(); // returns 0, 1, or 2
    assert a>=0 && a<=2 : "Impossible: a is out of range";
    // ...
    if (a==0) {
        // deal with 0 ...
    } else if (a==1) {
        // deal with 1 ...
    } else {
        assert a==2;
        // deal with 2 ...
    }
}
```

Safety checks have been done twice and one of the asserts appears redundant, but watch what happens when the code changes...
After code changes...

```java
public void method() {
    int a = getValue(); // returns 0, 1, or 2
    assert a>=0 && a<=2 : "Impossible: a is out of range";
    // ...
    boolean shouldPromote = shouldPromote( b, c, d );
    if (shouldPromote && somethingElse)
        a++;
    a = modifyMaybe( a );
    // ...
    if (a==0) {
        // deal with 0 ...
    } else if (a==1) {
        // deal with 1 ...
    } else {
        assert a==2;
        // deal with 2 ...
    }
}
```

The inserted code is between the first and second assertion and changes the value of the variable `a`. The second assertion is no longer redundant.
Programming tip by Greg Travis

It is good programming discipline to always have a final else { } case for any conditional. If you add one, but you know that it should never be reached, add an assert false;.
6.4.2 What to check for

Rule: use assertions to check related conditions at the start of any method.

```java
public void processZipCode( String zipCode )
{
    assert zipCodeMapIsValid();
    // ...
}
```

- The assertion checks that a valid map for zip codes has been loaded so that the zip code can be processed correctly.
- A map of zip codes identifies geographical boundaries and is typically used for marketing purposes (real estate,...).
Rule: use assertions to check things in the middle of a long-lived loop.

```java
public class Server {
    private ServerSocket serverSocket;
    public void acceptConnections() {
        while (true) {
            Socket socket = serverSocket.accept(); //Long pause here
            assert socketListIsValid(); //It’s good to check some stuff
            // deal with new connection ...
        }
    }
}
```

accept(): “Listens for a connection to be made to this socket over a network and accepts it.” This code could run for some time.

- Place safety checks in the middle of long-lived loops to ensure that assumptions made at the start of the loop still hold after the passage of some time.
6.4.3 Miscellaneous rules

Rule: do not use an assertion as a shorthand for “if (something) error ();”

```java
public class Server {
    private int port;
    public void listen() {
        // BAD!!
        assert port >= 1024 : "No permission to listen on port " + port;
        // ...
    }
}
```

```java
public void listen() {
    if (port < 1024) {
        throw new RuntimeException("No permission to listen on port "+port);
    }
    // ...
}
```

- Here, the programmer was lazy and forgot that assertions typically do not run in the deployed software. The code should have thrown a proper exception.
6.4.3 Miscellaneous rules

**Rule: do not use an assertion as an externally controllable conditional.**

```java
public class Application {
    static private boolean turnLoggingOn() {
        // Turn logging on
        // ...
        return true;
    }
    static public void main( String args[] ) throws Exception {
        // ...
        // BAD!!
        assert turnLoggingOn();
    }
}
```

The semantics of `da` and `ea` are changed. Using **da** means logging is not turned on. Using **ea** means logging is turned on. End users should not have to worry about changing semantics like this.

```
java -da Application
java -ea Application
```
6.4.3 Miscellaneous rules

Rule: do not use assertions to check the correctness of your compiler, operating system, ..., unless you are debugging it.

```java
public void method() {
    int a = 10;
    // REDUNDANT!!
    assert a==10;
}
```

- This assertion will be triggered only if there is a serious problem with your system.
- If you suspect that your compiler has a bug, then it is okay to use assertions like this, but otherwise such assertions will simply confuse anyone reading the code.
  - Why is this assertion here? I don’t understand...
6.4.3 Miscellaneous rules

Final rule: any assertion is better than nothing.

- Assertions are easy to add.
- So if you think you are making an assumption about something, add an assertion.

"An assertion that is never triggered is far better than one that would have been triggered but wasn’t there."

Greg Travis