Assertions

Program Correctness

Outline

- Assertions
- Using Exception Handling or Assertions?
- Conclusion

Assertions

- Java statement
  - enables you to assert an assumption about your program.
  - An assertion contains a Boolean expression that should be true during program execution.
  - Assertions can be used to assure program correctness and avoid logic errors.

Declaring Assertions

An *assertion* is declared using the new Java keyword assert in JDK 1.4 as follows:

```
assert booleanExpressionIsFalse;
    if (booleanExpressionIsFalse) throw new AssertionError();

assert booleanExpressionIsFalse : detailMessage;
    if (booleanExpressionIsFalse) throw new AssertionError(detailMessage);
```

where *assertion* is a Boolean expression and *detailMessage* is a primitive-type or an Object value.

Which constructor is used?

- assert *assertion*;
  - the no-arg constructor of AssertionError is used.
- assert *assertion* : *detailMessage*;
  - an appropriate AssertionError constructor is used to match the data type of the message.
  - Since AssertionError is a subclass of Error, when an assertion becomes false, the program displays a message on the console and exits.

Executing Assertions Example

```
public class AssertionDemo {

    public static void main(String[] args) {
        int i; int sum = 0;
        for (i=0; i < 10; i++) {
            sum += i;
        }

        assert i == 10;
        assert sum > 10 && sum < 5 * 10 : "sum is " + sum;

        /*System.out.println("i = " + i);
        System.out.println("sum = " + sum); */
    }

}
```

Compiling Programs with Assertions

Since assert is a new Java keyword introduced in JDK 1.4, you have to compile the program using a JDK 1.4 compiler. Furthermore, you need to include the switch –source 1.4 in the compiler command as follows:

javac –source 1.4 AssertionDemo.java

Executing Assertions

- When Java executes the assertion statement:
  - the boolean assertion is evaluated
  - If it is false, an AssertionError will be thrown.
  - AssertionError class constructors:
    - no-arg constructor
    - seven overloaded single-argument constructors of type
      - int,
      - long,
      - float,
      - double,
      - boolean,
      - char,
      - and Object.

Constructor Summary

AssertionError()
Constructs an AssertionError with no detail message.

AssertionError(boolean detailMessage)
Constructs an AssertionError with its detail message derived from the specified boolean, which is converted to a string as defined in *The Java Language Specification, Second Edition*, Section 15.18.1.1.

AssertionError(char detailMessage)
Constructs an AssertionError with its detail message derived from the specified char, which is converted to a string as defined in *The Java Language Specification, Second Edition*, Section 15.18.1.1.

AssertionError(double detailMessage)
Constructs an AssertionError with its detail message derived from the specified double, which is converted to a string as defined in *The Java Language Specification, Second Edition*, Section 15.18.1.1.

AssertionError(float detailMessage)
Constructs an AssertionError with its detail message derived from the specified float, which is converted to a string as defined in *The Java Language Specification, Second Edition*, Section 15.18.1.1.

AssertionError(int detailMessage)
Constructs an AssertionError with its detail message derived from the specified int, which is converted to a string as defined in *The Java Language Specification, Second Edition*, Section 15.18.1.1.

AssertionError(long detailMessage)
Constructs an AssertionError with its detail message derived from the specified long, which is converted to a string as defined in *The Java Language Specification, Second Edition*, Section 15.18.1.1.

AssertionError(Object detailMessage)
Constructs an AssertionError with its detail message derived from the specified object, which is converted to a string as defined in *The Java Language Specification, Second Edition*, Section 15.18.1.1.

Running Programs with Assertions

By default, the assertions are disabled at runtime. To enable it, use the switch –enableassertions, or –ea for short, as follows:

java –ea AssertionDemo

Assertions can be selectively enabled or disabled at class level or package level. The disable switch is –disableassertions or –da for short. For example, the following command enables assertions in package package1 and disables assertions in class Class1.

java –ea:package1 –da:Class1 AssertionDemo

If assertions are enabled and the boolean expression is true, nothing happens.
If assertions are enabled and the boolean expression is false, an AssertionError is thrown.
If assertions are disabled, no matter the outcome of the boolean expression, assertions are ignored.

Enable Assertion (single instance)

Assertions are better used to validate parameters, properties, preconditions, postconditions, application flow of control (for points of the code that should never be reached), and in general, error checking that otherwise, you would have to comment or disabled when the code is ready for production.

Assertions can also be enabled for one class:

```
java –ea:ClassName MainClass
```

Or for all classes in a package:

```
java –ea:com.example MainClass
```

For enabling assertions in the unnamed package (when you don't use a package statement):

```
java –ea:... MainClass
```

Or in the rare case you'd want to enable assertions for the system classes:

```
java –esa MainClass
```

Or

```
java –enablesystemsassertions MainClass
```

Enable Assertion (multiple instance)

If a single command line contains multiple instances of these switches, they are processed in order before loading any classes. For example, to run a program with assertions enabled only in package `com.wombat.fruitbat` (and any subpackages), the following command could be used:

```
java -ea:com.wombat.fruitbat... java -ea:com.wombat.fruitbat... <Main class>
```

- *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

For example, 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
```

Disable Assertion

There's also a command to disable assertions:

```
java –da MainClass
```

Or

```
java –disableassertions MainClass
```

With the same options that `-ea`. To disable for one class:

```
java –da:ClassName MainClass
```

For disabling assertions in a package:

```
java –da:com.example MainClass
```

For enabling assertion in the unnamed package:

```
java –ea:... MainClass
```

For the system classes:

```
java –dsa MainClass
```

Or

```
java –disablesystemassertions MainClass
```

This option exists because there will be times when you want to use commands like:

```
java –ea -da:ClassOne MainClass
```

Using Exception Handling or Assertions?

- Assertion should not be used to replace exception handling.
- Exception handling deals with unusual circumstances during program execution.
- Assertions are to assure the correctness of the program.
- Exception handling addresses robustness and assertion addresses correctness.
- Like exception handling, assertions are not used for normal tests, but for internal consistency and validity checks. Assertions are checked at runtime and can be turned on or off at startup time.

Using Exception Handling or Assertions?

- *Do not use assertions for argument checking in public methods*.
  - Valid arguments that may be passed to a public method are considered to be part of the method's contract.
  - The contract must always be obeyed whether assertions are enabled or disabled.

```
public void setRadius(double newRadius) {
  assert newRadius >= 0;
  radius =  newRadius;
}// improper


public void setRadius(double newRadius) {
if (newRadius >=0)
      radius = newRadius;
else
      throw new IllegalArgumentException(
        "Radius cannot be negative');
}
```

Using Exception Handling or Assertions?

- Use assertions to reaffirm assumptions.
  - This gives you more confidence to assure correctness of the program.
  - A common use of assertions is to replace assumptions with assertions in the code.

Using Exception Handling or Assertions?

- Another good use of assertions is place assertions in a switch statement without a default case.

```
switch (month) {
  case 1: ... ; break;
  case 2: ... ; break;
  ...
  case 12: ... ; break;
  default: assert false : "Invalid month: " + month
}
```

Conclusion

- An assertion is a statement used to check if something is true and helps you to detect errors in a program.
- Assertion should not be used to replace exception handling.
- An assert statement has the following syntax:

  assert booleanExpressionIfFalse;

  assert booleanExpressionIfFalse: "Message about the error";
- If the boolean expression of the assert statement evaluates to false, the program will throw a java.lang.AssertionError and terminate.
- By default, assertions are disabled. You can use the command-line arguments –ea (for enabling assertions) and –da (for disabling assertions) with other options for classes, packages, and system classes when running the program.