Assertion is a new feature in JDK 1.4. You can read this supplement after Chapter 13.

13.8 Assertions

An *assertion* is a Java statement that 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.

13.8.1 Declaring Assertions

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

assert *assertion*; or

assert *assertion* : *detailMessage*;

where *assertion* is a Boolean expression and *detailMessage* is a primitive-type or an Object value. When an assertion statement is executed, Java evaluates the assertion. If it is false, an AssertionError will be thrown. The AssertionError class has a default constructor and seven overloaded single-argument constructors of type int, long, float, double, boolean, char, and Object. For the first assert statement with no detail message, the default constructor of AssertionError is used. For the second assert statement with a detail message, an appropriate AssertionError constructor is used to match the data type of the message. Since AssertionError is a subclass of Error, when a assertion becomes false, the program displays a message on the console and terminates.

Here is an example of using assertions.

******PD: Please add line numbers in the following code******

```
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;
  }
}
```

The statement assert i == 10 asserts that i is 10 when the statement is executed. If i is not 10 an AssertionError is thrown. The statement assert sum < 5 * 10 : "sum is " + sum asserts that sum < 5 * 10. If false, an AssertionError with message "sum is " + sum is thrown.

Suppose you typed i < 100 instead of i < 10 by mistake in Line 4, the following AssertionError would be thrown:

```
Exception in thread "main" java.lang.AssertionError
        at AssertionDemo.main(AssertionDemo.java:7)
```

Suppose you typed sum += i instead of sum += 1 by mistake in Line 5, the following AssertionError would be thrown:

```
Exception in thread "main" java.lang.AssertionError: sum is 10
        at AssertionDemo.main(AssertionDemo.java:7)
```

13.8.2 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
```

NOTE: In JBuilder, check *Enable assert keyword* in the General tab of the Project properties dialog to compile and run programs with assertions.

13.8.3 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
```

13.8.4 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.

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. For example, the following code should be rewritten using exception handling as shown in Lines 28-35 in Circle.java in Example 13.1.

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

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. For example, the following code

```
if (even) {
  ...
}
else { // even is false
  ...
}
```

can be replaced by

```
if (even) {
  ...
}
else {
  assert !even;
  ...
}
```

The following code

```
if (numOfDollars > 1) {
  ...
}
else if (numOfDollars == 1) {
  ...
}
```

can be replaced by

```
if (numOfDollars > 1) {
  ...
}
else if (numOfDollars == 1) {
  ...
}
else
  assert false : numOfDollars;
```

Another good use of assertions is place assertions in a switch statement without a default case. For example,

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