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 ensure program correctness and avoid logic errors.

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 no-arg constructor and seven overloaded single-parameter constructors of type `int`, `long`, `float`, `double`, `boolean`, `char`, and `Object`. For the first `assert` statement with no detailed message, the no-arg constructor of `AssertionError` is used. For the second `assert` statement with a detailed message, an appropriate `AssertionError` constructor is used to match the data type of the message. `AssertionError` is a subclass of `Error`, so when an assertion becomes false, the program displays a message on the console and exits.

Here is an example of using assertions:

```java
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 > 10 && sum < 5 * 10 : "sum is " + sum` asserts that `sum > 10` and `sum < 5 * 10`. If false, an `AssertionError` with the 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 += 1` instead of `sum += i` 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:8)
```

2 Running Programs with Assertions
By default, assertions are disabled at runtime. To enable them, use the switch `-enableassertions`, or `-ea` for short, as follows:

```
java -ea AssertionDemo
```

Assertions can be selectively enabled or disabled at the class level or the 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
```

3 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 intended to ensure the correctness of the program. Exception handling addresses robustness, whereas 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 in (a) should be rewritten using exception handling, as shown in (b).

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

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

Use assertions to reaffirm assumptions. This will increase your confidence in the program’s correctness. A common use of assertions is to replace assumptions with assertions in the code. For example, the following code in (a) can be replaced by (b).

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

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

Similarly, the following code in (a) can also be replaced by (b).

```java
if (numberOfDollars > 1) {
    ...
} else if (numberOfDollars == 1) {
    ...
}
```

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

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

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