# **Programming with Assertions**

Source: <https://docs.oracle.com/javase/6/docs/technotes/guides/language/assert.html>

An assertion is a statement in the JavaTM programming language that enables you to test your assumptions about your program. For example, if you write a method that calculates the speed of a particle, you might assert that the calculated speed is less than the speed of light.

Each assertion contains a boolean expression that you believe will be true when the assertion executes. If it is not true, the system will throw an error. 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.

# **When to Use Assertions**

Use Assertions to verify

* Internal Invariants
* Control-Flow Invariants
* ~~Preconditions~~(but only for private methods), **Postconditions**, and Class Invariants

DO NOT use Assertions

* for argument checking in public methods.

Argument checking is typically part of the published specifications (or contract) of a method, and these specifications must be obeyed whether assertions are enabled or disabled. Another problem with using assertions for argument checking is that erroneous arguments should result in an appropriate runtime exception (such as IllegalArgumentException, IndexOutOfBoundsException, or NullPointerException). An assertion failure will not throw an appropriate exception.

* Do not use assertions to do any work that your application requires for correct operation.

Because assertions may be disabled, programs must not assume that the boolean expression contained in an assertion will be evaluated. Violating this rule has dire consequences. For example, suppose you wanted to remove all of the null elements from a list names, and knew that the list contained one or more nulls. It would be wrong to do this:

// Broken! - action is contained in assertion

assert names.remove(null);

// Fixed - action precedes assertion

boolean nullsRemoved = names.remove(null);

assert nullsRemoved; // Runs whether or not asserts are enabled

As a rule, the expressions contained in assertions should be free of side effects.

**Look at example: AssertDemo01.java**

# **Preconditions and Postconditions**

Source: <https://www.infoworld.com/article/3543239/how-to-use-assertions-in-java.html>

Assertions test a program’s assumptions by verifying that its various preconditions and postconditions aren’t violated, alerting the developer when a violation occurs:

* A precondition is a condition that must evaluate to true before the execution of some code sequence. Preconditions ensure that callers keep their contracts with callees.
* A postcondition is a condition that must evaluate to true after the execution of some code sequence. Postconditions ensure that callees keep their contracts with callers.

You can enforce preconditions on public constructors and methods by making explicit checks and throwing exceptions when necessary. For private helper methods, you can enforce preconditions by specifying assertions.

**Look at example: AssertDemo02.java**

The PNG class is the minimal beginning of a library for reading and decoding PNG (portable network graphics) image files. The constructor explicitly compares filespec with null, throwing NullPointerException when this parameter contains null. The point is to enforce the precondition that filespec not contain null.

It’s not appropriate to specify assert filespec != null; because the precondition mentioned in the constructor’s Javadoc would not (technically) be honored when assertions were disabled. (In fact, it would be honored because FileInputStream() would throw NullPointerException, but you shouldn’t depend on undocumented behavior.)

However, assert is appropriate in the context of the private readHeader() helper method, which will be completed eventually to read and decode a PNG file’s 8-byte header. The precondition that is always be passed a non-null value will always hold.

**Look at example: AssertDemo03.java**

Postconditions are typically specified via assertions, regardless of whether or not the method (or constructor) is public.

AssertDemo03 demonstrates that assertions are typically expensive to execute. For this reason, assertions are usually disabled in production code.

# **Assertions versus Exceptions**

Source (but cleaned up by Neal Holtschulte): <https://stackoverflow.com/questions/1957645/when-to-use-an-assertion-and-when-to-use-an-exception>

Assertions should be used to check something that should never happen, while an exception should be used to check something that might happen.

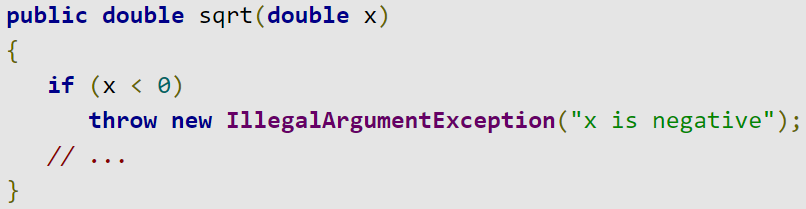
An assertion stops the program from running, but an exception can let the program continue running.

* Use exceptions when checking parameters passed to public or protected methods and constructors.
* Use exceptions when interacting with the user or when you expect the client code to recover from an exceptional situation.
* Use exceptions to address problems that might occur
* Use assertions when checking pre-conditions, post-conditions and invariants of private/internal code.
* Use assertions to provide feedback to yourself or your developer team.
* Use assertions when checking for things that are very unlikely to happen, but if it does happen it means that there is a serious ﬂaw in your application.
* Use assertions to state things that you (supposedly) know to be true.

In other words, **exceptions address the robustness of your application while assertions address its correctness**.

Assertions are designed to be cheap to write. You can use them almost everywhere. When debugging a program that does not behave the right way, you will surely check the more obvious failure possibilities. Then you will check for problems that just cannot happen: this is exactly when assertions help a lot and save time.

Correct usage:



Incorrect usage:

