Exceptions and Assertions

Oracle® Certified Professional, Java® SE 7 Programmer

Module 6

Module 6 - Objectives

- Use throw and throws statements.
- Use the try statement with multi-catch, and finally clauses.
- Auto-close resources with a try-with-resources statement.
- Create custom exceptions.
- Test invariants by using assertions.

Exceptions and Errors

- All exceptions and errors are sub-types of Throwable.
- Developers are not required to handle errors, or RuntimeException(s).
- A "checked" exception requires handling by the implementer.
- If you want to catch an Error, but are unsure of the specific error; you may catch Throwable

Exceptions and Errors

- A printStackTrace() prints out the last method called, and all previous methods down the call stack (unwinding).
- Checked exceptions must be handled in try…catch…finally (optional) blocks, or explicitly thrown.
- A try…catch block is used by the programmer to gracefully handle the exception to try and recover.

Exceptions and Errors

- finally blocks are used for cleanup.
- You should generally try to make a best effort to handle the exception in a try…catch, if not you should throw it.
- Java 7 introduced try with multi-catch. This allows a single multi-catch statement to process the exception in one place. This reduces the need for repetitious code.
  - try {… } catch(FileNotFoundException | IllegalArgumentException e) {…}

try…catch…finally

- Every handled "checked" exception must be in a try…catch block.
- The finally block is optional.
- The finally block is always called. It is always called after the try…catch block.
- The finally block is always called, even if there is a return statement in the try…catch block. It is called before the method returns.

try…catch…finally (cont.)

- A try must have a catch, finally, or both.

Propagating Exceptions

- If you are not going to handle the exception, add a throws keyword followed by the exception to your method.
- If you catch an exception in a try..catch…finally block, and do not handle it, you should re-throw the original exception, and include any additional information.

Exceptions

- When catching exceptions, you must go from most specific to least specific in your catch blocks.
- Super types of an exception before its sub-types will cause a compiler error.
- If a method is called which throws an exception, you must either handle it, or declare that you throw the exception.

Exceptions

- Exceptions are either JVM, or programmatic.
- NoClassDefFoundError, and NullPointerException are very common. The first is usally a classpath issue, and the second is usually the result of an empty initialization, or parameter.
- You should not catch runtime (unchecked) exceptions.

throw and throws

- When you want to report an exception you “throw” the exception.
- When a method has a checked exception which is not handled, it must indicate this by using “throws” .

Assertions

- Assertions have been in Java since 1.4.
- Assertions help to validate our expectations in our code.
- Assertions have been replaced in a number of places by frameworks like jUnit, testNG, etc.
- Assertions check validity via boolean expression.

Assertions (cont.)

- Assertions use the keyword assert.

```
assert (/*condition to validate*/);

assert(/*condition to validate*/) : value;
```

- If the assertion fails, an AssertionError is thrown.
- Assertions should not be used in flow control.

Assertions (cont.)

- Assertions are automatically "turned-off" in your code. You must enable them by passing the -ea parameter.
- The -da parameter disables assertions on a package basis.
- Assertions can return a value to be passed to the AssertionError to provide more detail.

Assertions (cont.)

- Do not use assertions to validate parameters to public methods.
- Use them to validate parameters to private methods.
- Use assertions in cases where something is supposed to never happen.
- Don't use assertions where the results produce side affects.

creative commons

Attribution-NonCommercial-ShareAlike 3.0 Unported