Java Object-Oriented Concepts Unit

Lesson 5: Exceptions

The Learning House

427 S 4th Street #300

Louisville KY 40202

Lesson 5: Exceptions

Overview

In this lesson, we'll move past the "happy path" and look at handling error conditions in our code. Error conditions in Java are represented by **exceptions**. The Java mechanisms for handling exceptions are the try/catch/finally construct and the throws keyword.

Definition

Exception is short for 'exceptional event'. An exception (as defined by the Oracle Java documentation) is:

> **An exceptional event which occurs in the normal execution of a program that disrupts the normal flow of the program's instructions.**

Catch or Specify Requirement

If your code (or code that your code calls) can cause an exception to be thrown, you must either **catch** the exception or **specify** that your code might cause that error to occur. Catching the exception means that you have written code either to try to recover from the error or to simply report that the error occurred. Specifying the exception means marking your code to indicate that it may cause this error. In this case, you are not trying to recover from the error or report it — your code simply throws the error and lets the caller try to handle it.

Exception Types

There are two major categories of exceptions: checked and unchecked. Checked exceptions are always subject to the "catch or specify" requirement whereas uncheck exceptions are not. You are still free to handle (i.e. try to recover from) unchecked exceptions in your code, but you are not required to. There are two types of unchecked exceptions: **errors** and **runtime exceptions**. Errors are abnormal conditions from which most programs should not attempt to recover, but there may be conditions under which you would want to attempt to recover from a runtime exception.

Handling (Catching) Exceptions

As mentioned above, we use the try/catch/finally construct to handle exceptions in Java. In this section, we'll look at each piece.

Try Block

The try block must surround the code that might cause the exception. This block can contain lines of the code that might cause several different exception types as well as lines of code that cannot cause an exception to occur.

Catch Block

Each try block must be accompanied by at least one catch block. A catch block contains code that either attempts to recover from the exception or simply reports the error in some way (for example, writes to a log file). If the try block contains code that throws more than one type of exception, you can have a separate catch block for each exception type so that you can respond to each error in a different way. A catch block can also

handle an entire class or family of exceptions, which is useful when you want to respond to all exceptions in the same way.

Finally Block

Each try/catch block combination may optionally be accompanied by a finally block. Code in the finally block will **always** run after the try/catch combination, whether an exception occurred or not. Finally blocks are well suited for code that cleans up resources — without the finally block, the cleanup code would have to appear in both the try block and the catch block(s).

```
try {
    //code that can cause an exception
    // goes here
} catch (<exception type> identifier) {
    // code to handle this type
    // of exception
} catch (<another exception type> identifier) {
    // code to handle this type
    // of exception
} finally {
    // code that runs whether an exception
    // occurred or not
}
```

Specifying Exceptions

If you decide that your code should not attempt to catch (i.e. handle) the exceptions that may be thrown with the try/catch/finally construct, you must then specify that your code can cause those exceptions. This is done using the **throws** keyword as part of you method definition. You simply add the throws keyword followed by a comma-delimited list of exception types that could be thrown to the method definition. For example, the following method throws an IOException:

```
public static void myMethod()  throws IOException {
    // method code goes here
}
```

Wrap-up

In this lesson we covered the basics of exception handling in Java:

- The definition of exception as it pertains to Java
- The catch or specify requirement
- The main categories of exceptions
- How to handle exceptions
- How to specify exceptions