11.8 Chained Exceptions

9

Sometimes a method responds to an exception by throwing a different exception type that’s specific to the current application. If a `catch` block throws a new exception, the original exception’s information and stack trace are *lost*. Earlier Java versions provided no mechanism to wrap the original exception information with the new exception’s information to provide a complete stack trace showing where the original problem occurred. This made debugging such problems particularly difficult. **Chained exceptions** enable an exception object to maintain the complete stack-trace information from the original exception. Figure 11.7 demonstrates chained exceptions.

```
1   // Fig. 11.7: UsingChainedExceptions.java
2   // Chained exceptions.
3
4   public class UsingChainedExceptions {
5      public static void main(String[] args) {
6         try {
7            method1();
8         }
9         catch (Exception exception) { // exception
10           exception.printStackTrace();
11        }
12     }
13
```

```
14      // call method2; throw exceptions back to mai
15      public static void method1() throws Exception
                16          try {
             17             method2();
                   18         }
19         catch (Exception exception) { // exception
20            throw new Exception("Exception thrown i
                   21         }
                     22      }
                         23
24      // call method3; throw exceptions back to met
25      public static void method2() throws Exception
                26          try {
             27             method3();
                   28         }
29         catch (Exception exception) { // exception
30            throw new Exception("Exception thrown i
                   31         }
                     32      }
                         33
    34      // throw Exception back to method2
35      public static void method3() throws Exception
36         throw new Exception("Exception thrown in m
                   37      }
                     38   }
```

```
java.lang.Exception: Exception thrown in method1
        at UsingChainedExceptions.method1(UsingChain
        at UsingChainedExceptions.main(UsingChainedE
Caused by: java.lang.Exception: Exception thrown in m
        at UsingChainedExceptions.method2(UsingChain
        at UsingChainedExceptions.method1(UsingChain
                        … 1 more
Caused by: java.lang.Exception: Exception thrown in m
        at UsingChainedExceptions.method3(UsingChain
        at UsingChainedExceptions.method2(UsingChain
                        … 2 more
```

Fig. 11.7

Chained exceptions.

Program Flow of Control

The program has four methods—`main` (lines 5–12), `method1` (lines 15–22), `method2` (lines 25–32) and `method3` (lines 35–37). Line 7 in `main`'s `try` block calls `method1`. Line 17 in `method1`'s `try` block calls `method2`. Line 27 in `method2`'s `try` block calls `method3`. In `method3`, line 36 throws a new `Exception`. Because line 36 is not in a `try` block, `method3` terminates, and the exception is returned to the calling method (`method2`) at line 27. This statement *is* in a `try` block; therefore, the `try` block terminates and the exception is caught at lines 29–31. Line 30 in the `catch` block throws a new exception. We call the `Exception` constructor with *two* arguments—the second represents the exception that was the original cause of the problem. In this program, that exception occurred at line 36. Because an exception is thrown from the `catch` block, `method2` terminates and returns the new exception to method1 at line 17. Once again, this statement is in a `try` block, so the `try` block terminates and the exception is caught at lines 19–21. Line 20 in the `catch` block throws a new exception and uses the exception that was caught as the second argument to `Exception`'s constructor. Because an exception is thrown from the `catch` block, `method1` terminates and

returns the new exception to `main` at line 7. The `try` block in `main` terminates, and the exception is caught at lines 9–11. Line 10 prints a stack trace.

`Throwable` Method `getCause`

For any chained exception, you can get the `Throwable` that initially caused that exception by calling `Throwable` method `getCause`.

Program Output

Notice in the program output that the first three lines show the most recent exception that was thrown (i.e., the one from `method1` at line 20). The next four lines indicate the exception that was thrown from `method2` at line 27. Finally, the last four lines represent the exception that was thrown from `method3` at line 36. Also notice that, as you read the output in reverse, it shows how many more chained exceptions remain.