25.9 Exceptions

[*Note:* This section may be read after studying Chapter 7 and the preceding sections of Chapter 25.]

In Section 7.5, we introduced Java's exception-handling mechanism, showing how to catch an exception that occurred when we attempted to use an out-of-bounds array index. In JShell, catching exceptions is not required—it automatically catches each exception and displays information about it, then displays the next JShell prompt, so you can continue your session. This is particularly important for *checked exceptions* (Section 11.5) that are required to be caught in regular Java programs—as you know, catching an exception requires wrapping the code in a `try`…`catch` statement. By automatically, catching all exceptions, JShell makes it easier for you to *experiment* with methods that throw checked exceptions.

In the following new JShell session, we declare an array of `int` values, then demonstrate both valid and invalid array-access expressions:

```
jshell> int[] values = {10, 20, 30}
values ==> int[3] { 10, 20, 30 }

jshell> values[1]
$2 ==> 20
```

```
jshell> values[10]
|  java.lang.ArrayIndexOutOfBoundsException thrown: 1
|        at (#3:1)

jshell>
```

The snippet `values[10]` attempts to access an out-of-bounds element—recall that this results in an `ArrayIndexOutOfBoundsException`. Even though we did not wrap the code in a `try`…`catch`, JShell catches the exception and displays the its `String` representation. This includes the exception’s type and an error message (in this case, the invalid index `10`), followed by a so-called stack trace indicating where the problem occurred. The notation

```
| at (#3:1)
```

indicates that the exception occurred at line 1 of the code snippet with the ID 3. Section 6.6 discussed the method-call stack. A stack trace indicates the methods that were on the method-call stack at the time the exception occurred. A typical stack trace contains several `"at"` lines like the one shown here—one per stack frame. After displaying the stack trace, JShell shows the next `jshell>` prompt. Chapter 11 discusses stack traces in detail.