7.5 Exception Handling: Processing the Incorrect Response

An **exception** indicates a problem that occurs while a program executes. **Exception handling** helps you create **fault-tolerant programs** that can resolve (or handle) exceptions. In some cases, this allows a program to continue executing as if no problems were encountered. For example, the `StudentPoll` application still displays results (Fig. 7.8), even though one of the responses was out of range. More severe problems might prevent a program from continuing normal execution, instead requiring it to notify the user of the problem, then terminate. When the JVM or a method detects a problem, such as an invalid array index or an invalid method argument, it **throws** an exception—that is, an exception occurs. Methods in your own classes can also throw exceptions, as you’ll learn in Chapter 8.

7.5.1 The `try` Statement

To handle an exception, place any code that might throw an exception in a `try` **statement** (lines 14–21 of Fig. 7.8). The `try` **block** (lines 14–16) contains the code that might *throw* an exception, and the `catch` **block** (lines 17–21) contains the code that *handles* the exception if one occurs. You can have *many* `catch`es to handle different *types* of exceptions that might be thrown in the corresponding `try` block. When line 15 correctly increments a `frequency` array element, lines 17–21 are ignored. The braces that delimit the bodies of the `try` and `catch` blocks are required.

7.5.2 Executing the `catch` Block

When the program encounters the invalid value `14` in the `responses` array, it attempts to add `1` to `frequency[14]`, which is *outside* the bounds of the array—the `frequency` array has only six elements (with indexes 0–5). Because array bounds checking is performed at execution time, the JVM generates an *exception*—specifically line 15 throws an `ArrayIndexOutOfBoundsException` to notify the program of this problem. At this point the `try` block terminates and the `catch` block begins executing—if you declared any local variables in the `try` block, they're now *out of scope* (and no longer exist), so they're not accessible in the `catch` block.

The `catch` block declares an exception parameter (`e`) of type `ArrayIndexOutOfBoundsException`. The `catch` block can handle exceptions of the specified type. Inside the `catch` block, you can use the parameter's identifier to interact with a caught exception object.

Error-Prevention Tip 7.1

When writing code to access an array element, ensure that the array index remains greater than or equal to 0 and less than the length of the array. This will prevent `ArrayIndexOutOfBoundsException`*s if your program is correct.*

Software Engineering Observation 7.1

Systems in industry that have undergone extensive testing are still likely to contain bugs. Our preference for industrial-strength systems is to catch and deal with runtime exceptions, such as `ArrayIndexOutOfBoundsException`*s, to ensure that a system either stays up and running or degrades gracefully, and to inform the system's developers of the problem.*

7.5.3 `toString` Method of the Exception Parameter

When lines 17–21 *catch* the exception, the program displays a message indicating the problem that occurred. Line 18 *implicitly* calls the exception object's `toString` method to get the error message that's implicitly stored in the exception object and display it. Once the message is displayed in this example, the exception is considered *handled* and the program continues with the next statement after the `catch` block's closing brace. In this example, the end of the `for` statement is reached (line 22), so the program continues with the increment of the control variable in line 13. We discuss exception handling again in Chapter 8, and more deeply in Chapter 11.