11.11 Assertions

When implementing and debugging a class, it’s sometimes useful to state conditions that should be true at a particular point in a method. These conditions, called **assertions**, help ensure a program’s validity by catching potential bugs and identifying possible logic errors during development. Preconditions and postconditions are two types of assertions. Preconditions are assertions about a program’s state when a method is invoked, and postconditions are assertions about its state after a method finishes.

While assertions can be stated as comments to guide you during program development, Java includes two versions of the `assert` statement for validating assertions programatically. The `assert` statement evaluates a `boolean` expression and, if `false`, throws an `AssertionError` (a subclass of `Error`). The first form of the `assert` statement is

```
assert expression;
```

which throws an `AssertionError` if *expression* is `false`. The second form is

```
assert expression1 : expression2;
```

which evaluates *expression1* and throws an `AssertionError` with *expression2* as the error message if *expression1* is `false`.

You can use assertions to implement *preconditions* and *postconditions* programmatically or to verify any other *intermediate* states that help you ensure that your code is working correctly. Figure 11.8 demonstrates the `assert` statement. Line 9 prompts the user to enter a number between 0 and 10, then line 10 reads the number. Line 13 determines whether the user entered a number within the valid range. If the number is out of range, the `assert` statement reports an error; otherwise, the program proceeds normally.

```
 1   // Fig. 11.8: AssertTest.java
 2   // Checking with assert that a value is within r
 3   import java.util.Scanner;
 4
 5   public class AssertTest {
 6      public static void main(String[] args) {
 7         Scanner input = new Scanner(System.in);
 8
 9         System.out.print("Enter a number between 0
10         int number = input.nextInt();
11
12         // assert that the value is >= 0 and <= 10
13         assert (number >= 0 && number <= 10) : "ba
14
15         System.out.printf("You entered %d%n", numb
16      }
17   }
```

```
Enter a number between 0 and 10: 5
You entered 5
```

```
Enter a number between 0 and 10: 50
Exception in thread "main" java.lang.AssertionError:
        at AssertTest.main(AssertTest.java:13)
```

Fig. 11.8

Checking with `assert` that a value is within range.

You use assertions primarily for debugging and identifying logic errors in an application. You must explicitly enable assertions when executing a program, because they reduce performance and are unnecessary for the program's user. To do so, use the java command's `-ea` command-line option, as in

```
java -ea AssertTest
```

Software Engineering Observation 11.16

Users shouldn't encounter `AssertionError`*s—these* *should be used only during program development. For this reason, you shouldn't catch* `AssertionError`*s. Instead, allow the program to terminate, so you can see the error message, then locate and fix the source of the problem. You should not use* `assert` *to indicate runtime problems in production code (as we did in Fig. 11.8 for demonstration purposes)—use the exception mechanism for this purpose.*