(table 9-4) *continued*

Summary of methods in the `File` class

Method	Description
`boolean isDirectory()`	Determines whether the path specifies a directory.
`long length()`	Gets the number of characters in the file.
`boolean mkDir()`	Makes the directory represented by this `File`. Returns `true` if successful.

9.4 Interacting with Users

Without input from users, most programs are not worth writing. A word processor that always typed the same essay, regardless of what the user wanted to say, would be worthless. A game program that didn't react to the game player's decisions would be boring.

This section and the next will discuss how to interact with the user of your program to modify how the program behaves. As an example, we will modify the program in Listing 9-6, which currently processes a server log, printing only those records resulting from serving a file larger than or equal to 25,000 bytes. Our new program will ask the user which log file to open and the minimum served file size to report. In particular, we want to implement the following pseudocode:

```
String fileName = ask the user for the log file to open
int minSize = ask the user for the minimum served file size
open the log file named in fileName
while (log file has another line)
{ ServerRecord sr = new ServerRecord(log file);
  if (sr.getSize() >= minSize)
  { print the record
  }
}
close the log file
```

9.4.1 Reading from the Console

Fortunately, the techniques we learned to read from a file can also be used to read information from the console. We still use the `Scanner` class, but we construct the `Scanner` object slightly differently, as follows:

```
Scanner cin = new Scanner(System.in);
```

`System.in` is an object similar to `System.out`. `Scanner` uses it to read from the console. Unlike opening a file, we are not required to catch any exceptions.

Before we implement the pseudocode discussed earlier, consider the sample program shown in Listing 9-7. It illustrates the important elements of reading from the console. The result of running this program is shown in Figure 9-4.

FIND THE CODE

ch09/readConsole/

Listing 9-7: *A short program demonstrating reading from the console*

```
 1  import java.util.Scanner;
 2
 3  public class ReadConsole
 4  {
 5    public static void main(String[] args)
 6    { Scanner cin = new Scanner(System.in);
 7
 8      System.out.print("Enter an integer: ");
 9      int a = cin.nextInt();
10      System.out.print("Enter an integer: ");
11      int b = cin.nextInt();
12
13      System.out.println(a + "*" + b + " = " + a * b);
14    }
15  }
```

The program begins by creating a new `Scanner` object used to read from the console. It's named `cin`, short for "console input."

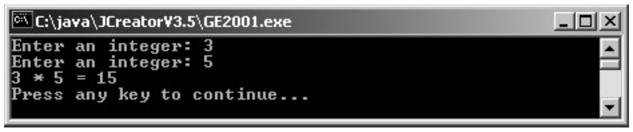
KEY IDEA

Prompt the user when input is expected.

At lines 8 and 10, the program prints a **prompt** for the user. The prompt informs the user that input is expected. In Figure 9-4, the user responded to the first prompt with 3↵. That is, the user entered the digit 3 and the Enter key. The Enter key is the user's cue to the program that it should read the input and process it. The program waits to read the input until Enter is pressed. The online documentation uses the term **block**, which means to wait for input.

(figure 9-4)

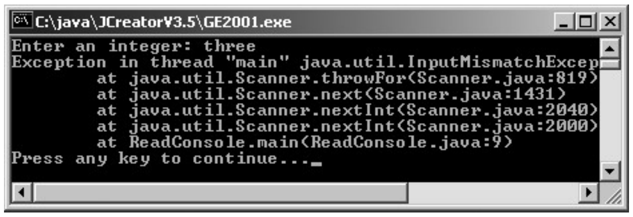
Result of running the program shown in Listing 9-7



```
Enter an integer: 3
Enter an integer: 5
3 * 5 = 15
Press any key to continue...
```

9.4.2 Checking Input for Errors

Unfortunately, if the user misreads the prompt and enters `three`↵, the program will throw an exception, as shown in Figure 9-5.



```
Enter an integer: three
Exception in thread "main" java.util.InputMismatchExcep
        at java.util.Scanner.throwFor(Scanner.java:819)
        at java.util.Scanner.next(Scanner.java:1431)
        at java.util.Scanner.nextInt(Scanner.java:2040)
        at java.util.Scanner.nextInt(Scanner.java:2000)
        at ReadConsole.main(ReadConsole.java:9)
Press any key to continue..._
```

(figure 9-5)

Result of entering data with an inappropriate type

The program can be protected from such errors with the code shown in Listing 9-8. The loop in lines 9–19 verifies that the next token is an integer. If it is not, the program reads it and displays a helpful message. This action gives the user the opportunity to try again. When an integer is entered, it is read in line 12, and the loop ends with the `break` in line 14.

Listing 9-8: *Rewriting Listing 9-7 to check for input errors*

```
 1  import java.util.Scanner;
 2
 3  public class ReadConsoleChecked
 4  {
 5    public static void main(String[] args)
 6    { Scanner cin = new Scanner(System.in);
 7
 8      int a = 0;
 9      while (true)
10      { System.out.print("Enter an integer: ");
11        if (cin.hasNextInt())
12        { a = cin.nextInt();
13          cin.nextLine();                  // consume remaining input
14          break;
15        } else
16        { String next = cin.nextLine();    // consume the error
17          System.out.println(next + " is not an integer such as 10 or -3.");
18        }
19      }
20      // Repeat lines 8–19, but read the data entered into variable b.
33
34      System.out.println(a + " * " + b + " = " + a * b);
35    }
36  }
```

FIND THE CODE

ch09/readConsole/

PATTERN

Error-Checked Input

The need to repeat essentially identical code to read the second integer suggests that a method should be written. Such a method does not rely on any instance variables and can therefore be a class method. In fact, a whole set of similar methods will be needed.

LOOKING BACK

Class methods were discussed in Section 7.5.2

We can place them in a class named `Prompt` and call them as shown in the following example:

```
int a = Prompt.forInt("Enter an integer: ");
```

KEY IDEA

The `Prompt` class must be used for all of the console input—or none of it.

Listing 9-9 shows the beginning of the class. Notice that line 9 declares a `static` (class) variable used to read from the console. One consequence of this decision is that *all* input from the console must be obtained with the methods in this class. Using more than one `Scanner` object to read from the same source (that is, `System.in`) will not work reliably.

LOOKING AHEAD

The problem set will ask you to add to this library.

Listing 9-9 also includes the methods `forInputFile` and `forInputScanner`. The first method uses the `File` class to verify that a string entered by the user specifies a file that exists and can be read by this program. The second method uses the first to open the specified file using `Scanner`. Putting this code in its own class has the following advantages:

- We can avoid writing it anew for each program that asks the user for a file or integer to process.
- We can put the `try-catch` statement here, rather than cluttering the main program with it.
- If the methods need enhancing or debugging, there is only one place that requires attention.

FIND THE CODE

ch09/userIO/

PATTERN

Error-Checked Input

Listing 9-9: *A class providing error-checked reading of an integer and a filename*

```
1  import java.util.Scanner;
2  import java.io.*;
3  
4  /** A set of useful static methods for interacting with a user via the console.
5   *
6   *  @author Byron Weber Becker */
7  public class Prompt extends Object
8  {
9    private static final Scanner in = new Scanner(System.in);
10 
11   /** Prompt the user to enter an integer.
12    *  @param prompt The prompting message for the user.
13    *  @return The integer entered by the user. */
14   public static int forInt(String prompt)
15   { while (true)
16     { System.out.print(prompt);
17       if (Prompt.in.hasNextInt())
18       { int answer = Prompt.in.nextInt();
19         Prompt.in.nextLine(); // consume remaining input
```

Listing 9-9: *A class providing error-checked reading of an integer and a filename* (continued)

```
          return answer;
        } else
        { String input = Prompt.in.nextLine();
          System.out.println("Error: " + input
                + " not recognized as an integer such as '10' or '-3'.");
        }
      }
    }

    /** Prompt the user for a file to use as input.
    *   @param prompt The prompting message for the user.
    *   @return A File object representing a file that exists and is readable. */
    public static File forInputFile(String prompt)
    { while (true)
      { System.out.print(prompt);
        String name = in.nextLine().trim();
        File f = new File(name);
        if (!f.exists())
        { System.out.println("Error: " + name + " does not exist.");
        } else if (f.isDirectory())
        { System.out.println("Error: " + name + " is a directory.");
        } else if (!f.canRead())
        { System.out.println("Error: " + name + " is not readable.");
        } else
        { return f;
        }
      }
    }

    /** Prompt the user for a file to use as input.
    *   @param prompt The prompting message for the user.
    *   @return A Scanner object ready to read the file specified by the user. */
    public static Scanner forInputScanner(String prompt)
    { try
      { return new Scanner(Prompt.forInputFile(prompt));
      } catch (FileNotFoundException ex)
      { // Shouldn't happen, given the work we do in forInputFile.
        System.out.println(ex.getMessage());
        System.exit(1);
      }
      return null;  // for the compiler
    }
  }
```

Using `Prompt`

The completed program for interacting with the user to ask for a specific Web server log file to process and the minimum size of returned page to print in a report is shown in Listing 9-10. Notice that it uses the `Prompt` class in lines 11 and 14.

FIND THE CODE

ch09/userIO/

Listing 9-10: *A program that processes a Web server log based on user input*

```
 1  import java.util.Scanner;
 2
 3  /** List files in a user-specified Web server log that meet a minimum size criteria.
 4   *   Report the number of files that are printed.
 5   *
 6   *   @author Byron Weber Becker */
 7  public class ListFilesBySize
 8  {
 9    public static void main(String[] args)
10    { // Prompt for the file to process.
11      Scanner in = Prompt.forInputScanner("Web server log name: ");
12
13      // Get the minimum size from the user.
14      int minSize = Prompt.forInt("Minimum served file size: ");
15
16      // Process the files.
17      int count = 0;
18      while (in.hasNextLine())
19      { ServerRecord sr = new ServerRecord(in);
20        if (sr.getSize() >= minSize)
21        { System.out.println(sr.toString());
22          count++;
23        }
24      }
25
26      // Close the input file and report the count.
27      in.close();
28      System.out.println(count + " files served were at least "
29                                 + minSize + " bytes.");
30    }
31  }
```