15.2 Files and Streams

Java views each file as a sequential **stream of bytes** (Fig. 15.1).[1] Every operating system provides a mechanism to determine the end of a file, such as an **end-of-file marker** or a count of the total bytes in the file that's recorded in a system-maintained administrative data structure. A Java program processing a stream of bytes simply receives an indication from the operating system when it reaches the end of the stream—the program does *not* need to know how the underlying platform represents files or streams. In some cases, the end-of-file indication occurs as an exception. In others, the indication is a return value from a method invoked on a stream-processing object.

[1]. Java's NIO APIs also include classes and interfaces that implement so-called channel-based architecture for high-performance I/O. These topics are beyond the scope of this book.

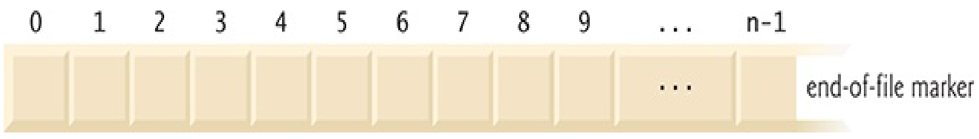

Fig. 15.1

Java's view of a file of *n* bytes.

Byte-Based and Character-Based Streams

File streams can be used to input and output data as bytes or characters.

- **Byte-based streams** output and input data in its *binary* format—a `char` is two bytes, an `int` is four bytes, a `double` is eight bytes, etc.
- **Character-based streams** output and input data as a *sequence of characters* in which every character is two bytes—the number of bytes for a given value depends on the number of characters in that value. For example, the value `2000000000` requires 20 bytes (10 characters at two bytes per character) but the value `7` requires only two bytes (1 character at two bytes per character).

Files created using byte-based streams are **binary files**, while files created using character-based streams are **text files**. Text files can be read by text editors, while binary files are read by programs that understand the file’s specific content and its ordering. A numeric value in a binary file can be used in calculations, whereas the character `5` is simply a character that can be used in a string of text, as in `"Sarah Miller is 15 years old"`.

Standard Input, Standard Output and Standard Error Streams

A Java program **opens** a file by creating an object and associating a stream of bytes or characters with it. The object’s constructor interacts with the operating system to *open* the file. Java can also associate streams with different devices. When a Java program begins executing, it creates three stream objects that are associated with devices—`System.in`, `System.out` and `System.err`. The `System.in` (standard input stream) object normally enables a program to input bytes from the keyboard. The `System.out` (standard output stream) object normally enables a program to output character data to the screen. The `System.err` (standard error stream) object normally enables a program to output character-based error messages to the screen. Each stream can be **redirected**. For `System.in`, this capability enables the program to read bytes from a different source. For `System.out` and `System.err`, it enables the output to be sent to a different location, such as a file on disk. Class `System` provides methods `setIn`, `setOut` and `setErr` to redirected the standard input, output and error streams, respectively.

The `java.io` and `java.nio` Packages

Java programs perform stream-based processing with classes and interfaces from package `java.io` and the subpackages of `java.nio`—Java’s New I/O APIs that were first introduced in Java SE 6 and have been enhanced since. There are also other packages throughout the Java APIs containing classes and interfaces based on those in the `java.io` and `java.nio` packages.

Character-based input and output can be performed with classes `Scanner` and `Formatter`, as you’ll see in Section 15.4. You’ve used class `Scanner` extensively to input data from the keyboard. `Scanner` also can read data from a file. Class `Formatter` enables formatted data to be output to any text-based stream in a manner similar to method `System.out.printf`. Appendix I presents the details of formatted output with `printf`. All these features can be used to format text files as well. In Chapter 28, we use stream classes to implement networking applications.

Java SE 8 Adds Another Type of Stream

8

Chapter 17, Lambdas and Streams, introduces a new type of stream that’s used to process collections of elements (like arrays and `ArrayList`s), rather than the streams of bytes we discuss in this chapter’s file-processing examples. In Section 17.13, we use the `Files` method `lines` to create one of these new streams containing the lines of text in a file.