Files, Streams and Object Serialization
Part 1:4

INTRODUCTION

- Data stored in variables and arrays is temporary
  - It's lost when a local variable goes out of scope or when the program terminates
- For long-term retention of data, computers use **files.**
- Computers store files on **secondary storage devices**
  - hard disks, optical disks, flash drives and magnetic tapes.
- Data maintained in files is **persistent data** because it exists beyond the duration of program execution.

Files and Streams

- Java views each file as a sequential **stream of bytes** (Fig. 17.1).
- 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 is recorded in a system-maintained administrative data structure.
- A Java program simply receives an indication from the operating system when it reaches the end of the stream

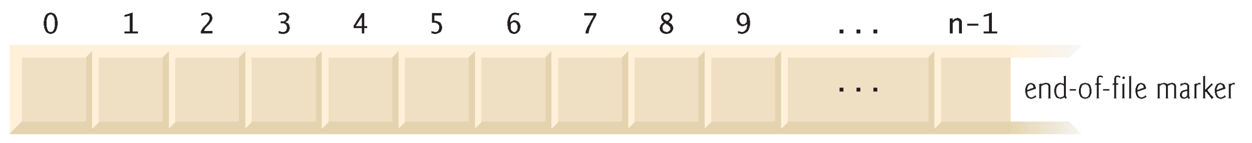

Fig. 17.1 | Java's view of a file of *n* bytes.

Files and Streams (cont.)

- File streams can be used to input and output data as bytes or characters.
- Streams that input and output bytes are known as **byte-based streams**, representing data in its binary format.
- Streams that input and output characters are known as **character-based streams**, representing data as a sequence of characters.
- Files that are created using byte-based streams are referred to as **binary files**.
- Files created using character-based streams are referred to as **text files**. Text files can be read by text editors.
- Binary files are read by programs that understand the specific content of the file and the ordering of that content.

Files and Streams (cont.)

- A Java program **opens** a file by creating an object and associating a stream of bytes or characters with it.
  - Can also associate streams with different devices.
- Java creates three stream objects when a program begins executing
  - `System.in` (the standard input stream object) normally inputs bytes from the keyboard
  - `System.out` (the standard output stream object) normally outputs character data to the screen
  - `System.err` (the standard error stream object) normally outputs character-based error messages to the screen.
- Class `System` provides methods **`setIn`**, **`setOut`** and **`setErr`** to **redirect** the standard input, output and error streams, respectively.

Files and Streams (cont.)

- Java programs perform file processing by using classes from package **`java.io`**.
- Includes definitions for stream classes
  - **`FileInputStream`** (for byte-based input from a file)
  - **`FileOutputStream`** (for byte-based output to a file)
  - **`FileReader`** (for character-based input from a file)
  - **`FileWriter`** (for character-based output to a file)
- You open a file by creating an object of one these stream classes. The object's constructor opens the file.

Files and Streams (cont.)

- Can perform input and output of objects or variables of primitive data types without having to worry about the details of converting such values to byte format.
- To perform such input and output, objects of classes `ObjectInputStream` and `ObjectOutputStream` can be used together with the byte-based file stream classes `FileInputStream` and `FileOutputStream`.
- The complete hierarchy of classes in package `java.io` can be viewed in the online documentation at
- http://download.oracle.com/javase/6/docs/api/java/io/package-tree.html

Files and Streams (cont.)

- Class **`File`** provides information about files and directories.
- Character-based input and output can be performed with classes `Scanner` and **`Formatter`**.
  - Class `Scanner` is used extensively to input data from the keyboard. This class can also 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`.

Class File

- Class `File` provides four constructors.
- The one with a `String` argument specifies the `name` of a file or directory to associate with the `File` object.
  - The `name` can contain **path information** as well as a file or directory name.
  - A file or directory's path specifies its location on disk.
  - An **absolute path** contains all the directories, starting with the **root directory**, that lead to a specific file or directory.
  - A **relative path** normally starts from the directory in which the application began executing and is therefore "relative" to the current directory.

CLASS FILE (CONT.)

- The constructor with two `String` arguments specifies an absolute or relative path and the file or directory to associate with the `File` object.
- The constructor with `File` and `String` arguments uses an existing `File` object that specifies the parent directory of the file or directory specified by the `String` argument.
- The fourth constructor uses a `URI` object to locate the file.
  - A **Uniform Resource Identifier (URI)** is a more general form of the **Uniform Resource Locators (URLs)** that are used to locate websites.
- Figure 17.2 lists some common `File` methods. The
- http://download.oracle.com/javase/6/docs/api/java/io/File.html

Method	Description
`boolean canRead()`	Returns `true` if a file is readable by the current application; `false` otherwise.
`boolean canWrite()`	Returns `true` if a file is writable by the current application; `false` otherwise.
`boolean exists()`	Returns `true` if the file or directory represented by the `File` object exists; `false` otherwise.
`boolean isFile()`	Returns `true` if the name specified as the argument to the `File` constructor is a file; `false` otherwise.
`boolean isDirectory()`	Returns `true` if the name specified as the argument to the `File` constructor is a directory; `false` otherwise.
`boolean isAbsolute()`	Returns `true` if the arguments specified to the `File` constructor indicate an absolute path to a file or directory; `false` otherwise.

Fig. 17.2 | `File` methods. (Part 1 of 2.)

Method	Description
`String getAbsolutePath()`	Returns a `String` with the absolute path of the file or directory.
`String getName()`	Returns a `String` with the name of the file or directory.
`String getPath()`	Returns a `String` with the path of the file or directory.
`String getParent()`	Returns a `String` with the parent directory of the file or directory (i.e., the directory in which the file or directory is located).
`long length()`	Returns the length of the file, in bytes. If the `File` object represents a directory, an unspecified value is returned.
`long lastModified()`	Returns a platform-dependent representation of the time at which the file or directory was last modified. The value returned is useful only for comparison with other values returned by this method.
`String[] list()`	Returns an array of `String`s representing a directory's contents. Returns `null` if the `File` object does not represent a directory.

Fig. 17.2 | `File` methods. (Part 2 of 2.)

```java
// Fig. 17.3: FileDemonstration.java
// File class used to obtain file and directory information.
import java.io.File;
import java.util.Scanner;

public class FileDemonstration
{
   public static void main( String[] args )
   {
      Scanner input = new Scanner( System.in );

      System.out.print( "Enter file or directory name: " );
      analyzePath( input.nextLine() );
   } // end main

```

Fig. 17.3 | `File` class used to obtain file and directory information. (Part 1 of 5.)

```
    // display information about file user specifies
    public static void analyzePath( String path )
    {
       // create File object based on user input
       File name = new File( path );

       if ( name.exists() ) // if name exists, output information about it
       {
          // display file (or directory) information
          System.out.printf(
             "%s%s\n%s\n%s\n%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s",
             name.getName(), " exists",
             ( name.isFile() ? "is a file" : "is not a file" ),
             ( name.isDirectory() ? "is a directory" :
                "is not a directory" ),
             ( name.isAbsolute() ? "is absolute path" :
                "is not absolute path" ), "Last modified: ",
             name.lastModified(), "Length: ", name.length(),
             "Path: ", name.getPath(), "Absolute path: ",
             name.getAbsolutePath(), "Parent: ", name.getParent() );

```

Fig. 17.3 | `File` class used to obtain file and directory information. (Part 2 of 5.)

```
            if ( name.isDirectory() ) // output directory listing
            {
               String[] directory = name.list();
               System.out.println( "\n\nDirectory contents:\n" );

               for ( String directoryName : directory )
                  System.out.println( directoryName );
            } // end if
         } // end outer if
         else // not file or directory, output error message
         {
            System.out.printf( "%s %s", path, "does not exist." );
         } // end else
      } // end method analyzePath
   } // end class FileDemonstration
```

Fig. 17.3 | `File` class used to obtain file and directory information. (Part 3 of 5.)

```
Enter file or directory name: E:\Program Files\Java\jdk1.6.0_11\demo\jfc
jfc exists
is not a file
is a directory
is absolute path
Last modified: 1228404395024
Length: 4096
Path: E:\Program Files\Java\jdk1.6.0_11\demo\jfc
Absolute path: E:\Program Files\Java\jdk1.6.0_11\demo\jfc
Parent: E:\Program Files\Java\jdk1.6.0_11\demo

Directory contents:

CodePointIM
FileChooserDemo
Font2DTest
Java2D
Laffy
Metalworks
Notepad
SampleTree
Stylepad
SwingApplet
SwingSet2
SwingSet3
```

Fig. 17.3 | `File` class used to obtain file and directory information. (Part 4 of 5.)

```
Enter file or directory name: C:\Program Files\Java\jdk1.6.0_11\demo\jfc
\Java2D\README.txt
README.txt exists
is a file
is not a directory
is absolute path
Last modified: 1228404384270
Length: 7518
Path: E:\Program Files\Java\jdk1.6.0_11\demo\jfc\Java2D\README.txt
Absolute path: E:\Program Files\Java\jdk1.6.0_11\demo\jfc\Java2D\RE-
ADME.txt
Parent: E:\Program Files\Java\jdk1.6.0_11\demo\jfc\Java2D
```

Fig. 17.3 | `File` class used to obtain file and directory information. (Part 5 of 5.)

Class File (cont.)

- A **separator character** is used to separate directories and files in the path.
- On Windows, the separator character is a backslash (\).
- On Linux/UNIX, it's a forward slash (/).
- Java processes both characters identically in a path name.
- When building `String`s that represent path information, use `File.separator` to obtain the local computer's proper separator.
  - This constant returns a `String` consisting of one character—the proper separator for the system.

Thank you

☺