```
      catch (Exception ex) {
        System.out.println("Exception in main");
      }
    }

    static void method() throws Exception {
      try {
        Circle c1 = new Circle(1);
        c1.setRadius(-1);
        System.out.println(c1.getRadius());
      }
      catch (RuntimeException ex) {
        System.out.println("RuntimeException in method()");
      }
      catch (Exception ex) {
        System.out.println("Exception in method()");
        throw ex;
      }
    }
  }
```

12.10 The File Class

The **`File`** *class contains the methods for obtaining the properties of a file/directory, and for renaming and deleting a file/directory.*

Key Point

Having learned exception handling, you are ready to step into file processing. Data stored in the program are temporary; they are lost when the program terminates. To permanently store the data created in a program, you need to save them in a file on a disk or other permanent storage device. The file can then be transported and read later by other programs. Since data are stored in files, this section introduces how to use the `File` class to obtain file/directory properties, to delete and rename files/directories, and to create directories. The next section introduces how to read/write data from/to text files.

why file?

Every file is placed in a directory in the file system. An *absolute file name* (or *full name*) contains a file name with its complete path and drive letter. For example, **c:\book\Welcome.java** is the absolute file name for the file **Welcome.java** on the Windows operating system. Here, **c:\book** is referred to as the *directory path* for the file. Absolute file names are machine dependent. On the UNIX platform, the absolute file name may be **/home/liang/book/Welcome.java**, where **/home/liang/book** is the directory path for the file **Welcome.java**.

absolute file name

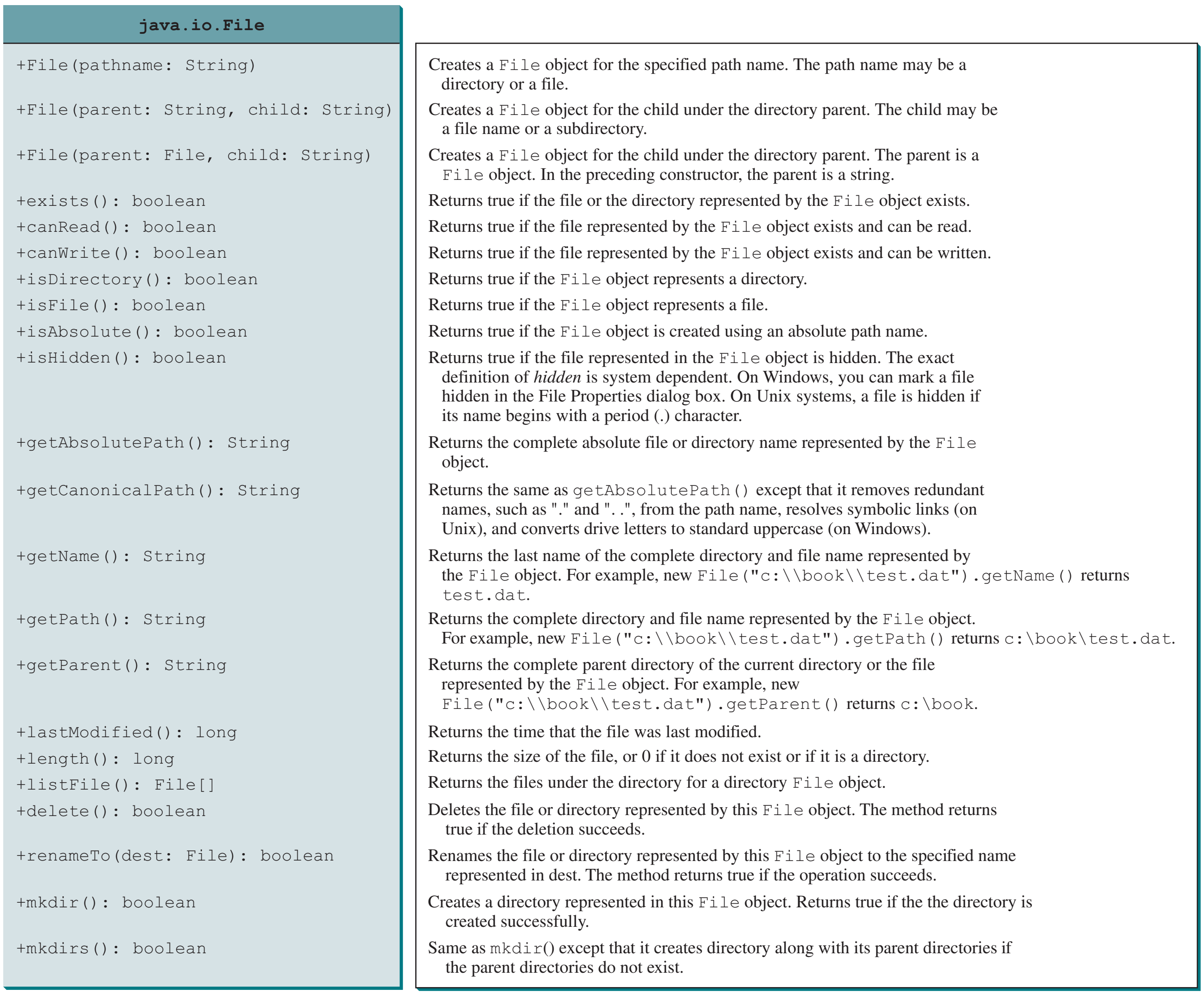
directory path

A *relative file name* is in relation to the current working directory. The complete directory path for a relative file name is omitted. For example, **Welcome.java** is a relative file name. If the current working directory is **c:\book**, the absolute file name would be **c:\book\Welcome.java**.

relative file name

The `File` class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion. The `File` class contains the methods for obtaining file and directory properties, and for renaming and deleting files and directories, as shown in Figure 12.6. However, *the* `File` *class does not contain the methods for reading and writing file contents*.

The file name is a string. The `File` class is a wrapper class for the file name and its directory path. For example, `new File("c:\\book")` creates a `File` object for the directory **c:\book** and `new File("c:\\book\\test.dat")` creates a `File` object for the file **c:\book\test.dat**, both on Windows. You can use the `File` class's `isDirectory()` method to check whether the object represents a directory, and the `isFile()` method to check whether the object represents a file.



java.io.File	
+File(pathname: String)	Creates a File object for the specified path name. The path name may be a directory or a file.
+File(parent: String, child: String)	Creates a File object for the child under the directory parent. The child may be a file name or a subdirectory.
+File(parent: File, child: String)	Creates a File object for the child under the directory parent. The parent is a File object. In the preceding constructor, the parent is a string.
+exists(): boolean	Returns true if the file or the directory represented by the File object exists.
+canRead(): boolean	Returns true if the file represented by the File object exists and can be read.
+canWrite(): boolean	Returns true if the file represented by the File object exists and can be written.
+isDirectory(): boolean	Returns true if the File object represents a directory.
+isFile(): boolean	Returns true if the File object represents a file.
+isAbsolute(): boolean	Returns true if the File object is created using an absolute path name.
+isHidden(): boolean	Returns true if the file represented in the File object is hidden. The exact definition of *hidden* is system dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period (.) character.
+getAbsolutePath(): String	Returns the complete absolute file or directory name represented by the File object.
+getCanonicalPath(): String	Returns the same as getAbsolutePath() except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
+getName(): String	Returns the last name of the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getName() returns test.dat.
+getPath(): String	Returns the complete directory and file name represented by the File object. For example, new File("c:\\book\\test.dat").getPath() returns c:\book\test.dat.
+getParent(): String	Returns the complete parent directory of the current directory or the file represented by the File object. For example, new File("c:\\book\\test.dat").getParent() returns c:\book.
+lastModified(): long	Returns the time that the file was last modified.
+length(): long	Returns the size of the file, or 0 if it does not exist or if it is a directory.
+listFile(): File[]	Returns the files under the directory for a directory File object.
+delete(): boolean	Deletes the file or directory represented by this File object. The method returns true if the deletion succeeds.
+renameTo(dest: File): boolean	Renames the file or directory represented by this File object to the specified name represented in dest. The method returns true if the operation succeeds.
+mkdir(): boolean	Creates a directory represented in this File object. Returns true if the the directory is created successfully.
+mkdirs(): boolean	Same as mkdir() except that it creates directory along with its parent directories if the parent directories do not exist.

FIGURE 12.6 The **`File`** class can be used to obtain file and directory properties, to delete and rename files and directories, and to create directories.

\ in file names

Caution

The directory separator for Windows is a backslash (`\`). The backslash is a special character in Java and should be written as `\\` in a string literal (see Table 4.5).

Note

*Constructing a **`File`** instance does not create a file on the machine.* You can create a **`File`** instance for any file name regardless of whether it exists or not. You can invoke the **`exists()`** method on a **`File`** instance to check whether the file exists.

relative file name

Java directory separator (/)

Do not use absolute file names in your program. If you use a file name such as **`c:\\book\\Welcome.java`**, it will work on Windows but not on other platforms. You should use a file name relative to the current directory. For example, you may create a **`File`** object using **`new File("Welcome.java")`** for the file **Welcome.java** in the current directory. You may create a **`File`** object using **`new File("image/us.gif")`** for the file **us.gif** under the **image** directory in the current directory. The forward slash (**`/`**) is the Java directory separator, which is the same as on UNIX. The statement `new File("image/us.gif")` works on Windows, UNIX, and any other platform.

Listing 12.12 demonstrates how to create a `File` object and use the methods in the `File` class to obtain its properties. The program creates a `File` object for the file **us.gif**. This file is stored under the **image** directory in the current directory.

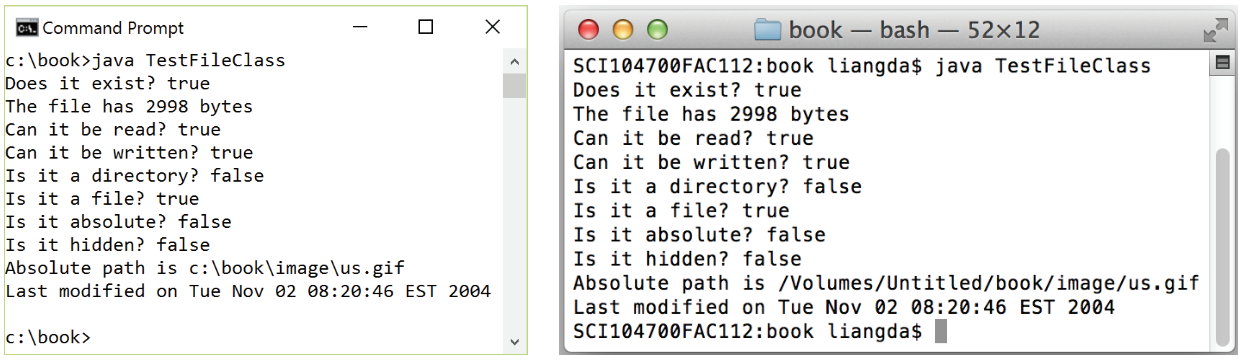
LISTING 12.12 `TestFileClass.java`

```
 1  public class TestFileClass {
 2    public static void main(String[] args) {
 3      java.io.File file = new java.io.File("image/us.gif");
 4      System.out.println("Does it exist? " + file.exists());
 5      System.out.println("The file has " + file.length() + " bytes");
 6      System.out.println("Can it be read? " + file.canRead());
 7      System.out.println("Can it be written? " + file.canWrite());
 8      System.out.println("Is it a directory? " + file.isDirectory());
 9      System.out.println("Is it a file? " + file.isFile());
10      System.out.println("Is it absolute? " + file.isAbsolute());
11      System.out.println("Is it hidden? " + file.isHidden());
12      System.out.println("Absolute path is " +
13        file.getAbsolutePath());
14      System.out.println("Last modified on " +
15        new java.util.Date(file.lastModified()));
16    }
17  }
```

create a `File`
`exists()`
`length()`
`canRead()`
`canWrite()`
`isDirectory()`
`isFile()`
`isAbsolute()`
`isHidden()`
`getAbsolutePath()`
`lastModified()`

The `lastModified()` method returns the date and time when the file was last modified, measured in milliseconds since the beginning of UNIX time (00:00:00 GMT, January 1, 1970). The `Date` class is used to display it in a readable format in lines 14 and 15.

Figure 12.7a shows a sample run of the program on Windows and Figure 12.7b, a sample run on UNIX. As shown in the figures, the path-naming conventions on Windows are different from those on UNIX.



```
c:\book>java TestFileClass
Does it exist? true
The file has 2998 bytes
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
Absolute path is c:\book\image\us.gif
Last modified on Tue Nov 02 08:20:46 EST 2004

c:\book>
```

(a) On Windows

```
SCI104700FAC112:book liangda$ java TestFileClass
Does it exist? true
The file has 2998 bytes
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
Absolute path is /Volumes/Untitled/book/image/us.gif
Last modified on Tue Nov 02 08:20:46 EST 2004
SCI104700FAC112:book liangda$
```

(b) On UNIX

FIGURE 12.7 The program creates a `File` object and displays file properties.

Check Point

12.10.1 What is wrong about creating a `File` object using the following statement?
`new File("c:\book\test.dat");`

12.10.2 How do you check whether a file already exists? How do you delete a file? How do you rename a file? Can you find the file size (the number of bytes) using the `File` class? How do you create a directory?

12.10.3 Can you use the `File` class for I/O? Does creating a `File` object create a file on the disk?