JAVA NOTES

DATA STRUCTURES AND ALGORITHMS

Terry Marris August 2001

17 THE FILE CLASS

17.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

- distinguish between *volatile* and *persistent* data storage structures
- explain the role of the *File* class
- use *File* methods to delete and rename files
- implement the *FilenameFilter* interface

17.2 INTRODUCTION

In chapters one through 16 inclusive, we studied the properties of volatile data structures, data structures that hold their data in memory and loose their data when the computer system is shut down. We now turn our attention to persistent data structures, which preserve their data between program runs and computer shut downs. We look at files held on disks.

17.3 THE FILE CLASS

The file class provides methods that manage the storage of a file on disk. It allows you to remove or rename a file and to determine whether a file with a given name exists.

A *File* constructor associates a *File* object with a file on disk.

```
File aFile = new File("FileTest.java");
```

The file on disk is named *FileTest.java*. Now that we have a *File* instance connected with an actual file we can

- see if *FileTest.java* actually exists on disk with *if aFile.exists() ...*
- rename *FileTest.java* to *FileTest.back* with *aFile.renameTo(new File("FileTest.back");*
- erase *FileTest.java* with *aFile.delete();*

We can also supply a full file name that includes the drive and directory path as an argument value to the constructor. We use *File.separator* to represent the directory separator symbol, \. So, to associate a *File* instance with *c:\JavaNotesDataStructures\FileTest.java* we write

```
File aFile = new File("c:" + File.separator +
                      "JavaNotes" + File.separator +
                      "DataStructures" + File.separator +
                      "FileTest.java");
```

java.io.File

Fields	
static final String separator	represents \ in the Windows filing system

Constructors	
File(String name)	Initialises a new File instance with the given name. If the full pathname to the file is not included, the current working directory is assumed. A file with the given name must exist. A file is created using a stream class constructor - see the next two chapters.

Methods		
static File	createTempFile(String prefix, String suffix, File directory)	creates a new temporary file in the given directory, using the given prefix and suffix to generate the file name. (Must conform to Windows rules for naming files.) Usually used in tandem with deleteOnExit().
boolean	delete()	removes this file (or directory, which must be empty). Returns true if successful, false otherwise.
void	deleteOnExit()	removes this file (or directory) when the program run terminates.
boolean	exists()	returns true if this file (or directory) exists.
boolean	isFile()	returns true if this File object represents a file (rather than a device or a directory).
String[]	list()	returns an array of strings containing the files (and directories) contained in this directory, in no particular order.
String[]	list(FilenameFilter f)	returns an array of strings of filenames that match the give filter, in no particular order.
boolean	renameTo(String newName)	renames this file as the given new name. Returns false if a file with the given new name already exists.

17.4 FILENAME FILTER INTERFACE

A *FilenameFilter* object enables you to list a restricted set of files e.g. all files that end in *.java*.

To implement the *FilenameFilter* interface we just implement the *boolean accept(File, String)* method.

```
class ExtensionFilter implements FilenameFilter {
  private String extension;

  public ExtensionFilter(String ext)
  {
    extension = "." + ext;
  }


  public boolean accept(File dir, String name)
  {
    return name.endsWith(extension);
  }
}
```

To use it we create a *File* directory object.

```
     File dir = new File(".");
```

The "." denotes the current working directory.

Then we create the *ExtensionFilter* object that selects just the *.java* files.

```
String[] fileList = dir.list(new ExtensionFilter("java"));
```

java.io.FilenameFilter

Methods		
boolean	accept(File dir, String name)	returns true if the file matches the filter criterion.

17.5 EXAMPLES

The first example program renames a file named *Four* to *Four.java*. The file named *Four* must exist. Perhaps the easiest way to create it is to use your normal program text editor. The file could contain just one line:

```
/* Four */
```

You could check the disk directory with the usual *dos dir* command to confirm that the rename has actually taken place. Of course, *rename()* will fail if a file with the new name already exists.

```
/* FileTest.java
   Terry Marris  6 August 2001
*/

import java.io.*;

public class FileTest {
  public static void main(String[] s)
  {
    File fileFour = new File("Four");
    if (fileFour.exists())
      fileFour.renameTo(new File("Four.java"));
  }
}
```

The second example program lists all the *.java* files in the current working directory.

```
/* ListJavaFiles.java
   Terry Marris  6 August 2001
*/

import java.io.*;

class ExtensionFilter implements FilenameFilter {
  private String extension;

  public ExtensionFilter(String ext)
  {
    extension = "." + ext;
  }


  public boolean accept(File dir, String name)
  {
    return name.endsWith(extension);
  }
}


public class ListJavaFiles {
  public static void main(String[] s) throws IOException
  {
    File dir = new File(".");
    String[] fileList = dir.list(new ExtensionFilter("java"));

    for (int i = 0; i < fileList.length; i++) {
      System.out.print(fileList[i]);
      if (i + 1 < fileList.length)
        System.out.print(", ");
    }
  }
}
```

Sample Output

```
Four.java, ListJavaFiles.java, FileTest.java
```

Some *File* methods throw an *IOException.* File handling is error prone. For example, files or directories with a given name may not be found because they have previously been deleted or a pathname may have been miss-spelled. Any method that deals with files is obliged to either let the system deal with file handling errors by including a *throws IOException* clause in the method signature or implementing a *try ... catch ...* routine - see later lessons for details and examples.

17.6 REVIEW

17.7 FURTHER READING

HORSTMANN & CORNELL *Core Java 2 Volume 1* pp 689

17.8 EXERCISES

1 Explain the meaning of the terms *volatile* and *persistent* with respect to data storage.

2 Explain the role of the *File* class.

3 Write fragments of Java code to accomplish the following

(a) create a *File* instance associated with the file *employee.data* in the current working directory
(b) create a *File* instance associated with the file *employeeData.back* in the current working directory
(c) check whether *employeeData.back* exists - if so, delete it.
(d) rename *employee.data* as *employeeData.back.*

4 Try out the example programs described in § 17.7 above.