26 File Management

1 File information: File

The file system can be manipulated using either class `File` which is simple, or class `Files` which is complex but more general and flexible. Here we use class `File`.

Files have properties, such as a name, whether or not it is readable and/or writable, whether it is a regular file or a folder (directory), and so on. Whether a file is readable or writable has nothing to do with the contents or the file or its structure or any error situations, but is a property of the file as an entity in the system. For example, a file may be unwritable if it exists on a write-once CD, or it may be unreadable because the user attempting to read it does not have permission to do so. Java provides the class **`File`** for discovering and modifying properties of files. It has a simple constructor:

```
File(String)
```

`new File(s)` creates an object of type `File` pertaining to a file known to the operating system as `s`. For example, the declaration

```
File myInfo = new File("diskData");
```

creates and assigns to variable `myInfo` an object of type `File` pertaining to file `diskData`. A path name can be supplied in place of a simple file name. It is possible that a file called `diskData` does not exist; that is allowable, but a new file of that name is *not* created. Class `File` resides in the `java.io` package which must be imported

`File` includes the following methods for discovering the status of a file:

```
boolean exists()
boolean isDirectory()
boolean isFile()
boolean canRead()
boolean canWrite()
String getName()
File[] listFiles()
```

`f.exists()` returns a boolean indicating whether the file associated with `f` exists. **`f.isDirectory()`**, **`f.isFile()`**, **`f.canRead()`**, and **`f.canWrite()`**, return booleans indicating whether the file is a directory, a regular file, is readable, and is writable, respectively. A *regular* file is, for all practical purposes, a file that it is not a directory. All these methods can be invoked without error whether or not the file exists. **`f.getName()`** returns the name of the file. **`f.listFiles()`** returns an array of `File` objects, one for each file or directory in the directory identified by `f`; `null` is returned if `f` does not identify a directory or if an i/o error occurs. The following methods of `File` are used to change the status of files:

```
boolean renameTo(File)
boolean delete()
```

`f.renameTo(fnew)` changes the name of the file associated with `f` to the name associated with `fnew`. Note that the new name must be supplied within an object `fnew` of type `File`. For example, `myFile.renameTo(new File("newFile"))` changes the name associated with `myFile` to `newFile`. **`f.delete()`** deletes the file associated with `f`. Both methods returns a boolean indicating whether they completed successfully.

Example 1: deleting files

The following program deletes files whose names are keyed in by the user. It takes care not to delete directories, and reports on the success or otherwise of each deletion.

```
import java.io.*;
class DeleteFiles {
   public static void main(String[] args) {
    System.out.print("Delete file: ");  // prompt user for file name or end of input
    while (!Console.endOfFile()) {
        String fileName = Console.readString(); // read name of file to be deleted
        File file = new File(fileName);
        if (!file.exists())
            System.out.println("Cannot find file " + fileName);
        else if (file.isDirectory())
            System.out.println("Cannot delete directory " + fileName);
        else {
            boolean ok = file.delete();
            if (ok) System.out.println(fileName + " deleted");
            else System.out.println("Cannot delete file " + fileName);
```

```
            }
            System.out.print("Delete file: "); // prompt for file name or end of input
        }
    }
}
```

Example 2: generating a unique file name

The following piece of code generates a file name that is guaranteed to be different from that of any other file in the directory.

```
String tempName = "Temp" + (int)(Math.random()*1000000);
File theFile = new File(tempName);
while (theFile.exists()) { // bad luck -- try again
    tempName = "Temp" + (int)(Math.random()*1000000);
    theFile = new File(tempName);
}
```

Example 3: listing the files in a directory

The following code segment lists the names of all files in a given directory.

```
String dirName = "C:\Java\MyProgs";
File theDirectory = new File(dirName);
File[] files = theDirectory.listFiles();
if (files==null)  // dirName not a valid directory name
    System.out.println(dirName+" is not a directory ");
else {
    for (File file: files)
        System.out.println(file.getName());
}
```