Chapter 12 Exception Handling and Text IO

Objectives

- To discover file/directory properties, to delete and rename files/directories, and to create directories using the **File** class
- To write data to a file using the **PrintWriter** class To use try-with-resources to ensure that the resources are closed automatically
- To read data from a file using the **Scanner** class
- To understand how data is read using a **Scanner**
- To develop a program that replaces text in a file

The File Class

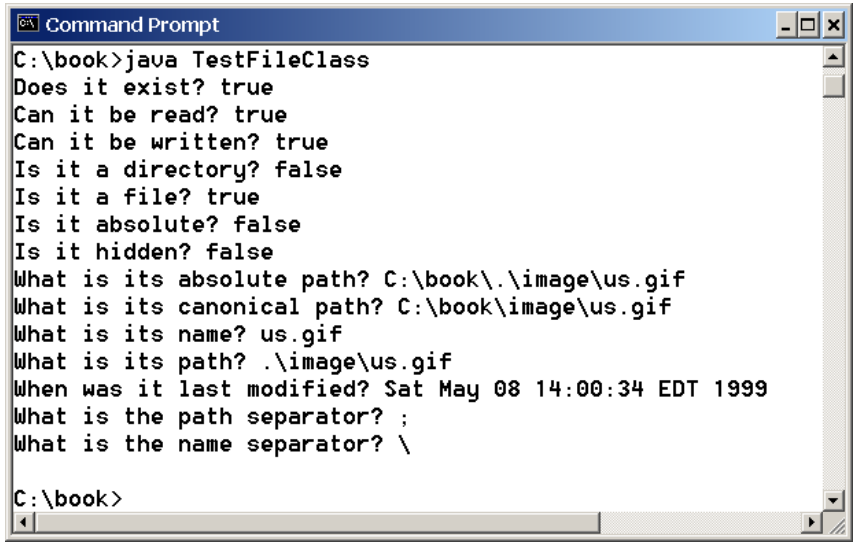
The File class is intended to provide an abstraction that deals with most of the machine-dependent complexities of files and path names in a machine-independent fashion. The filename is a string. The File class is a wrapper class for the file name and its directory path.

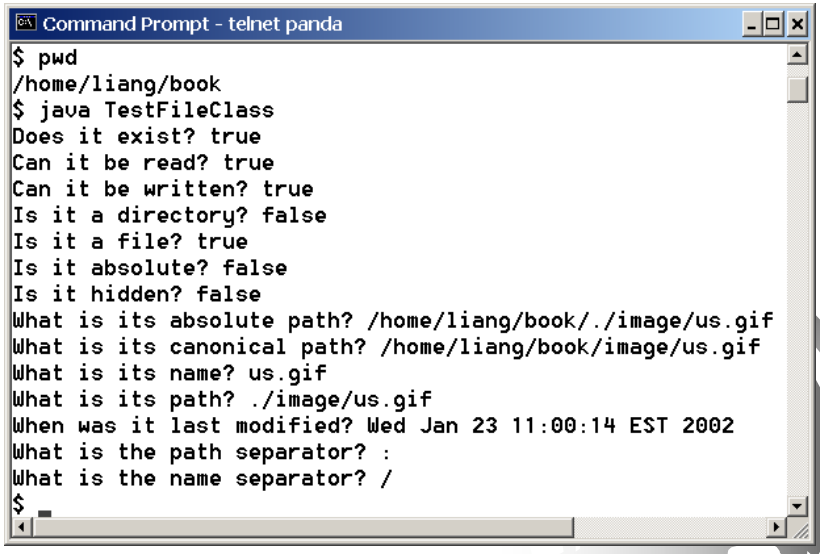
Obtaining file properties and manipulating file

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.

Problem: Explore File Properties

Objective: Write a program that demonstrates how to create files in a platform-independent way and use the methods in the File class to obtain their properties. The following figures show a sample run of the program on Windows and on Unix.





TestFileClass Run

Example to Explore File Properties

```
Import java.io.File
import java.util.Date;
public class TestFileClass {
   public static void main(String[] args)
   {
        File myFile = new File("test/my.txt");
        // Create image folder in your project folder & copy us.gif there
        System.out.println("Does it exist? " + myFile.exists());
        System.out.println("The file has " + myFile.length() + " bytes");
        System.out.println("Can it be read? " + myFile.canRead());
        System.out.println("Can it be written? " + myFile.canWrite());
        System.out.println("Is it a directory? "
                                    + myFile e.isDirectory());
        System.out.println("Is it a file? " + myFile.isFile());
        System.out.println("Is it absolute? " + myFile.isAbsolute());
        System.out.println("Is it hidden? " + myFile.isHidden());
        System.out.println("Absolute path is "
                                    + myFile.getAbsolutePath());
        Date d= new Date(myFile.lastModified());
        System.out.println("Last modified on "+ d);
        }
}
```

Text I/O

A File object encapsulates the properties of a file or a path, but does not contain the methods for reading/writing data from/to a file. In order to perform I/O, you need to create objects using appropriate Java I/O classes. The objects contain the methods for reading/writing data from/to a file. This section introduces how to read/write strings and numeric values from/to a text file using the Scanner and PrintWriter classes.

Writing Data Using PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §4.6, "Formatting Console Output and Strings."

WriteData Run

Example Writing Data Using PrintWriter

```
Import java.io.PrintWriter;
public class WriteDataToFile {
public static void main(String[] args) throws java.io.IOException
        {
        java.io.File myFile = new java.io.File("scores.txt");
        if (myFile.exists()) {// if you don't want to overwrite the file
                System.out.println("File already exists");
                System.exit(0);
               }
        // Create a output object to write to the file
        PrintWriter output = new PrintWriter(myFile);
        // Write formatted output to the file
        output.print("John Thomas Smith ");
        output.println(90);
        output.print("Eric Kan Jones ");
        output.println(85);
         // Must Close the file
        output.close();
    }
}
```

Try-with-resources

Programmers often forget to close the file. JDK 7 provides the followings new try-with-resources syntax that automatically closes the files.

```
try (declare and create resources) {
  Use the resource to process the file;
}
```

WriteDataWithAutoClose Run

Example Writing Data Using PrintWriter with **try** automatically closes the file

```
public class WriteDataWithAutoClose {
public static void main(String[] args) throws java.io.IOException
        {
        java.io.File file = new java.io.File("scores.txt");
        if (file.exists()) { // if you don't want to overwrite the file
                System.out.println("File already exists");
                System.exit(0);
               }
        try( // try is used to automatically close the file
             // Create a output object to write to a file
             java.io.PrintWriter output = new java.io.PrintWriter(file);
          )
           {
               // Write formatted output to the file
               output.print("John T Smith ");
               output.println(90);
               output.print("Eric K Jones ");
               output.println(85);
                // Must Close the file
               output.close();
            } // end of try body
    }
}
```

Reading Data Using Scanner

java.util.Scanner	
+Scanner(source: File)	Creates a Scanner object to read data from the specified file.
+Scanner(source: String)	Creates a Scanner object to read data from the specified string.
+close()	Closes this scanner.
+hasNext(): boolean	Returns true if this scanner has another token in its input.
+next(): String	Returns next token as a string.
+nextByte(): byte	Returns next token as a byte.
+nextShort(): short	Returns next token as a short.
+nextInt(): int	Returns next token as an int.
+nextLong(): long	Returns next token as a long.
+nextFloat(): float	Returns next token as a float.
+nextDouble(): double	Returns next token as a double.
+useDelimiter(pattern: String): Scanner	Sets this scanner's delimiting pattern.

ReadData Run

Example Reading Data from file Using Scanner and display on Output Screen

```
public class ReadDataFromFile {
public static void main(String[] args) throws java.io.IOException
    {
        java.io.File file = new java.io.File("scores.txt");
        // note score.text was created in the previous example inside
        // your project folder.
        // Create a Scanner for reading data from the file
        java.util.Scanner input = new java.util.Scanner(file);
        // Read data from a file
        while (input.hasNext())
        {
          String firstName = input.next();
          String middleName = input.next();
          String lastName = input.next();
           int score = input.nextInt();
            System.out.println( firstName + " " + middleName +
                " " + lastName + " " + score);
        }
         // Must Close the file
        input.close();
    }
}
```

```java
package ch12files;
import java.io.*;
import java.util.Scanner;
public class FilesSlide14_ReadFromFileAndWriteToFile {
 public static void main(String[] args) throws Exception {
    // Create a File instance to refer to the score.text
    File inputfile = new File("scores.txt");
    File outputfile = new File("scores1.txt");
    // Create a Scanner for the file
    Scanner input = new Scanner(inputfile);
    PrintWriter output = new PrintWriter(outputfile);
    // Read data from a file
    while (input.hasNext()) {
      String firstName = input.next();
      String middleName = input.next();
      String lastName = input.next();
      int score = input.nextInt();
      //Now write same data to other file
      output.print(firstName+" ");
      output.print(middleName+" ");
      output.print(lastName+" ");
      output.println(score);
     }
    // Close both files
    output.close();
    input.close();
  }
}
```

Example Reading Data from a file and writing to a other file

Any Questions