21. Text I/O

Java

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

- File Class
- Writing Data Using PrintWriter
- Reading Data Using Scanner
- Example: Replacing Text
- References

File Class

File Class

- Data stored in variables, arrays, and objects is temporary and is 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.
- The file can be transported and can be read later by other programs.

File Class

- Every file is placed in a directory in the file system.
- **Absolute file 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
  - Here c:\book is referred to as the **directory path** for the file.

File Class

- java.io.File
  - a class that helps you that examines and manipulates files and directories.
- The File class **does not** contain the methods for reading and writing file contents.
- File instances represent file names, not files.
- The file corresponding to the file name might not even exist.

File Class

- Why create a File object for a file that doesn't exist?
  - The file can be created by passing the File object to the constructor of some classes, such as FileWriter.
- If the file **does exist**, a program can examine its attributes and perform various operations on the file, such as **renaming** it, **deleting** it, or **changing** its permissions.

File Class

- For example:

  File a = new File(“test.dat");

  - creates a File object for the file test.dat

  File a = new File("c:\\book")

  - creates a File object for the directory c:\book

  File a = new File("c:\\book\\test.dat")

  - creates a File object for the file c:\\book\\test.dat

File Class Methods

- exists(): boolean
  - Returns true if the file or the directory represented by the File object exists.
- 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.
- canRead(): boolean
  - Returns true if the file represented by the File object exists and can be read.

File Class Methods

- isHidden(): boolean
  - Returns true if the file represented in the File object is hidden.
- lastModified(): long
  - Returns the time that file was last modified, measured in milliseconds since the time (00:00:00 GMT, January 1, 1970).
- getAbsolutePath(): String
  - Returns the complete absolute file or directory name represented by the File object.

TestFileClass.java

- Example:
  - TestFileClass.java
- The output:

```
Does it exist? true
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? true
Is it hidden? false
Absolute path is d:\Test\test.dat
Last modified on Sat Sep 20 01:11:54 IRDT 2008
```

Writing Data Using PrintWriter Class

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 write strings and numeric values to a text file using the PrintWriter class.

Writing Data Using PrintWriter

- The java.io.PrintWriter class can be used to write data to a text file.
- First, you have to create a PrintWriter object for a text file as follows:

```
PrintWriter output = new PrintWriter(filename);
```

- Then, you can invoke the print, println, and printf methods on the PrintWriter object to write data to a file.

PrintWriter Methods

- **PrintWriter(file: File)**
  - Creates a PrintWriter object 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.

PrintWriter Methods

- **print(f: float): void**
  - Writes a float value.
- **print(d: double): void**
  - Writes a double value.
- **print(b: boolean): void**
  - Writes a boolean value.
- **close(): void**
  - Close the file
- Also contains the overloaded println & printf methods.
- A println method acts like a print method; additionally it prints a line separator.

WriteData.java

- This program gives an example that creates an instance of PrintWriter and writes two lines to the file "scores.txt".
- Each line consists of first name (a string), middle name initial (a character), last name (a string), and score (an integer).
- Program:
  - WriteData.java

WriteData.java

- Invoking the constructor new PrintWriter(String filename) may throw an I/O exception. For example if the filename exists.
- Java forces you to write the code to deal with this type of exception.
- For now, simply declare throws Exception in the method declaration
- You will learn how to handle exceptions (run time errors) later.

WriteData.java

- The content of scores.txt:

  John T Smith 90

  Eric K Jones 85

Reading Data Using Scanner

Reading Data Using Scanner

- The java.util.Scanner class is used to read from a file
- To create a Scanner to read data from a file, you have to use the java.io.File class to create an instance of the File using the constructor new File(filename)
- Then use new Scanner (File) to create a Scanner for the file as follows:

```
Scanner input = new Scanner(new File(filename));
```

Scanner Methods

- **Scanner(source: File)**
  - Creates a Scanner that produces values scanned from the specified file.
- **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.

Scanner Methods

- **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.

ReadData.java

- Invoking the constructor new Scanner(File) may throw an I/O exception. So the main method declares throws Exception
- The program:
  - ReadData.java
- The output:

  John T Smith 90

  Eric K Jones 85

Reading Data Using Scanner

- **useDelimiter(pattern: String): Scanner**
  - Sets this scanner's delimiting pattern.
  - By default, the delimiters for separating tokens in a Scanner are whitespace.
  - You can use the useDelimiter(String) method to set a new pattern for delimiters.

Example: Replacing Text

Example: ReplacingText.java

- Write a method named replaceText that replaces a string in a text file with a new string.
- The filename and strings are passed as arguments as follows:

  replaceText (sourceFile, targetFile, oldString, newString)
- The program:
  - ReplacingText.java
- The content of scoresNew.txt:

  John T Smith 90

  Eric K Keaton 85

References

References

- Y. Daniel Liang, **Introduction to Java Programming**, Sixth Edition, Pearson Education, 2007. (Chapter 8)

The End