Keenan Knaur
Adjunct Lecturer

California State University, Los Angeles
Computer Science Department

Text I/O

CS2012: Introduction to Programming II

The File Class

File I/O

- File I/O (input / output) is very common in programming.

- Files allow you to store data permanently
  - all data stored in variables is lost once the program exits.
  - files can be backed up and transferred from computer to computer easily.

Absolute File Paths

- ***absolute file name***:
  - contains a file name with its complete path and driver letter
- example:
  - windows:
    - `c:\user\documents\games.txt`
  - UNIX/LINUX:
    - `/home/user/documents/games.txt`
- these names are machine dependent
  - different machines may have different file systems, and folder structures
  - absolute file names are generally not good for dynamic programs.

Relative File Paths

- ***relative file name:***
  - name is relative to the current working directory
  - the complete directory path is usually omitted

- example: games.txt

- In Eclipse:
  - Create a folder in your **project directory** called **resources**. Place your .txt files in this folder.
  - The relative path would be:
    - resources/fileName.txt

The File Class

- Java provides a `File` class to easily work with files stored on a computer.

- This class is machine independent.

- Contains basic operations for files:
  - obtaining a file or directory and its properties
  - creating or deleting a file or directory
  - renaming files or directories
  - etc...

- The File class API will be very important to study for learning how to use this object type.

The File Class

- This class ONLY creates an object to interact with a file on your computer.
- This class does not have any methods for reading or writing TO the file, we will need to use other classes for this.
- Caution: Be careful, if you delete a file in your program, it will actually remove it from your computer.

Caution with Backslash

- The directory separator in Windows is a backslash (\)
- If you want to write a back slash in a string you need to escape it by writing a double backslash \\
- If you use a file name such as: c:\\book\\Welcome.java it will only work on a Windows machine and not on any others.
- you should always just use a forward slash (/) because this will work on any machine.

File I/O

File Input and Output

- For file I/O you need to use either the `Scanner` or `PrintWriter` classes.
  - `Scanner` can be used to read data from the file.
  - `PrintWriter` can be use to write to the file.

- These classes are generally used for text i/o
  - All data is read and written as text.

- Another way to read data is by using Binary I/O.
  - All data is kept as binary data instead of being converted to text.
  - We will look at binary I/O much later in the semester.

Writing Data to a File

- **PrintWriter** is used to write data.
  - To create an instance of PrintWriter, you will need to use a try / catch to deal with the FileNotFound exception.
- The constructor to PrintWriter can accept a String or a File object.
  - The String must be a path to a file.
- Create a PrintWriter object and use the **print** or **println** methods to write the data to the file.
- Caution: Don't forget to close the PrintWriter object, otherwise all of the data may not be written to the file.
- See Example:

Automatically Closing Resources

- A good practice is to always close your input / output streams as soon as you are finished with them.
  - This will free up memory as input / output streams will use up resources until they are closed.
- You can close your i/o stream using the .close() function.
- JDK 7 also added a new try / catch called the "try-with-resources" to automatically close files.

try-with-resources

```
try(PrinterWriter out = new PrintWriter(file)) {
   //Code for file writing
}
catch (FileNotFoundException ex) {
   //Code for exception handling
}
```

Closing Resources Automatically

- The resource is declared and created in the try clause.
  - Be sure to enclose the resources in ( ).

- For the try-with-resources to work, the resource must be a subtype of `AutoCloseable` and have a .close() method.
  - `PrintWriter` and Scanner are both `AutoCloseable`.

Reading Data with Scanner

- In addition to reading data from the console, Scanner can also be used to read data from a file.

- When you construct the Scanner object, simply replace System.in with the file object you want to read from.
  - `Scanner in = new Scanner(file)`

- token-reading methods:
  - `nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), next()`
  - methods which read tokens separated by delimiters
  - by default, the delimiters are whitespace characters (new-line character, tabs, spaces, etc)
  - you can use the `useDelimiter(String regex)` method to set a new pattern for delimiters

- each method skips any delimiters and then reads the next "token" converting it to the correct datatype (except for next() which requires no conversion).

next() and nextLine()

- Both of these methods read strings
  - next() reads up to the next whitespace character
  - nextLine() reads up to the next line separator character.

- Note about line-separators:
  - the line-separator string is defined by the system
  - Windows is "\r\n"
  - Unix based systems it is "\n"
  - to get the line separator for a particular system use:
    - String lineSeparator = System.getProperty("line.separator");
  - when you enter input from the keyboard, the line ends with the enter key press which is the \n character.

Reading Data from a CSV File

- A .csv file is a "comma-separated values" file.
  - Usually you can open these files in a program like Excel or LibreOffice Calc.
- Each value in the file is separated by a comma and the comma can be used a delimiter with Scanner.
- You can also use the split() method of the String class to automatically split an entire line on each comma and each value gets placed into an array.
- See Example.

References

- Liang, Chapter 12: Exception Handling and Text I/O
- Liang, Chapter 13: Abstract Classes and Interfaces