Structural Programming and Data Structures

Winter 2000

CMPUT 102: File Input/Output

Dr. Osmar R. Zaïane

University of Alberta

Course Content

- Introduction
- Objects
- Methods
- Tracing Programs
- Object State
- Sharing resources
- Selection
- Repetition
- Vectors
- Testing/Debugging
- Arrays
- Searching
- **Files I/O**
- Sorting
- Inheritance
- Recursion

Objectives of Lecture 22
File Input/Output

- Introduce the concept of a file in Java;
- Learn how to write data to files;
- Learn how to read data from files.

Outline of Lecture 22

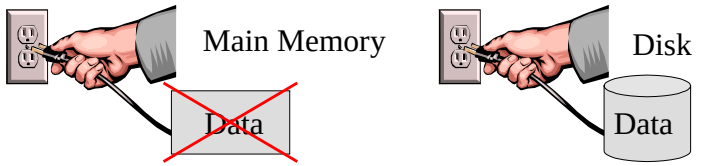
- Files and basic operations on files
- Writing to a file (File output)
- Reading from a file (File input)

User - input/output

- A program can input data from the user, process the data and output some results to the screen. The data is stored and manipulated in main memory.
- However, if you run the program once, obtain the data, exit the program and then start the program again, all of the data is lost.
- Main memory is volatile, in the sense that data stays in memory only as long as the program is still in execution and the computer is on.

Main Memory versus Disk

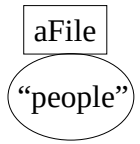
- In order to store data for a long period of time, it is better to store the data on disk.
- Data stored on disk can be accessed even after the program has been terminated and restarted again.



File - input/output

- A program that outputs data to a disk file, exits and then re-starts, can access the data again, by inputting it from the disk file.
- Java has a class called File whose instances are used to **represent** disk files.
- You can create a File object that represents a disk file called “people”:

```
File aFile;
aFile = new File(“people”);
```



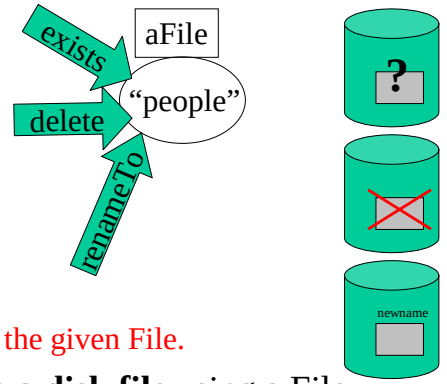
The File Class

- Here are some messages you can send to a File object:

boolean exists()
// returns true if the file exists

void delete()
// deletes the file.

void renameTo(File aFile)
// rename this file to the name of the given File.



- However, you cannot **create a disk file** using a File object, or read or write data to a disk file directly!

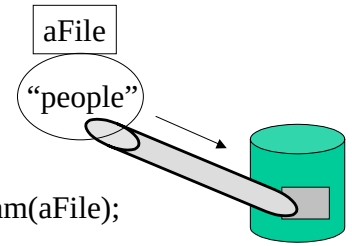
Outline of Lecture 22

- Files and basic operations on files
- Writing to a file (File output)
- Reading from a file (File input)

FileOutputStream Class

- To create a disk file and to write to it, you need to create an instance of FileOutputStream "on" the File object:

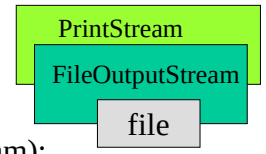
```
File              aFile;
FileOutputStream  outputStream;
aFile = new File("people");
outputStream = new FileOutputStream(aFile);
```



- If a disk file named "people" did not exist then creating the FileOutputStream creates it.
- If the file already existed, it is emptied.

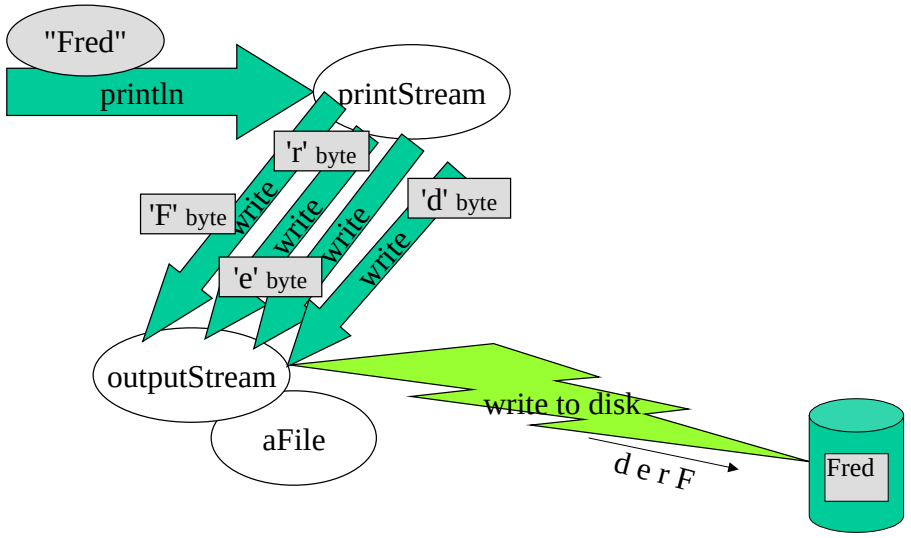
PrintStream Class

- Unfortunately, the only thing you can do with a FileOutputStream is to output bytes onto it.
- We usually want to output Strings or ints or other more interesting objects and values.
- To do this we create a PrintStream object "on" the FileOutputStream:




```
PrintStream       aPrintStream;
aPrintStream = new PrintStream(outputStream);
```


Output Objects



PrintStream class (con't)

- We have already used an instance of PrintStream in this course, but it was not created on a FileStream.
- The object reference, "*System.out*" is bound to a PrintStream on the screen.
- When we send the message:

  System.out.println("hello");

  we are sending a message to an instance of PrintStream.

File Output Example

```
import java.io.*;
public class FileOutput {
// This program is an example of file output.
public static void main(String args[]) throws Exception {

    File             aFile;
    FileOutputStream outputStream;
    PrintStream      aPrintStream;
    aFile = new File("people");
    outputStream = new FileOutputStream(aFile);
    aPrintStream = new PrintStream(outputStream);
    aPrintStream.println("Flinstone, Fred");
    aPrintStream.close();
    }
}
```

necessary "magic" when creating a FileOutputStream

Closing the stream is necessary

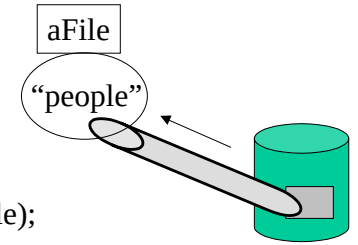
Outline of Lecture 22

- Files and basic operations on files
- Writing to a file (File output)
- Reading from a file (File input)

FileInputStream class

- To read a disk file, you need to create an instance of FileInputStream "on" the File object:

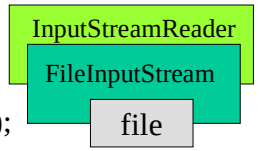
```
File            aFile;
FileInputStream aFileStream;
aFile = new File("people");
aFileStream = new FileInputStream(aFile);
```



- If a disk file named "people" exists, the FileInputStream is ready to read it.
- If not, an "exception" will occur and your program will terminate.

InputStreamReader class

- Unfortunately, the only thing you can do with a FileInputStream is to input bytes.
- We usually want to input each line of a file as a String or int or some other object or value.
- To do this we first create an InputStreamReader object "on" the FileInputStream that reads **characters**:

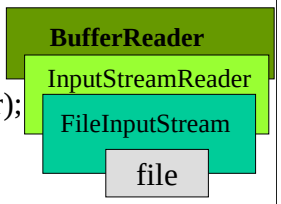



```
InputStreamReader     aReader;
aReader = new InputStreamReader(aFileStream);
```


BufferedReader class

- To read **lines** as Strings instead of reading characters, we need to construct a BufferedReader "on" the InputStreamReader:

```
BufferedReader      aBufferedReader;
String              aString;

aBufferedReader = new BufferedReader(aReader);
aString = aBufferedReader.readLine()
```



File Input Example

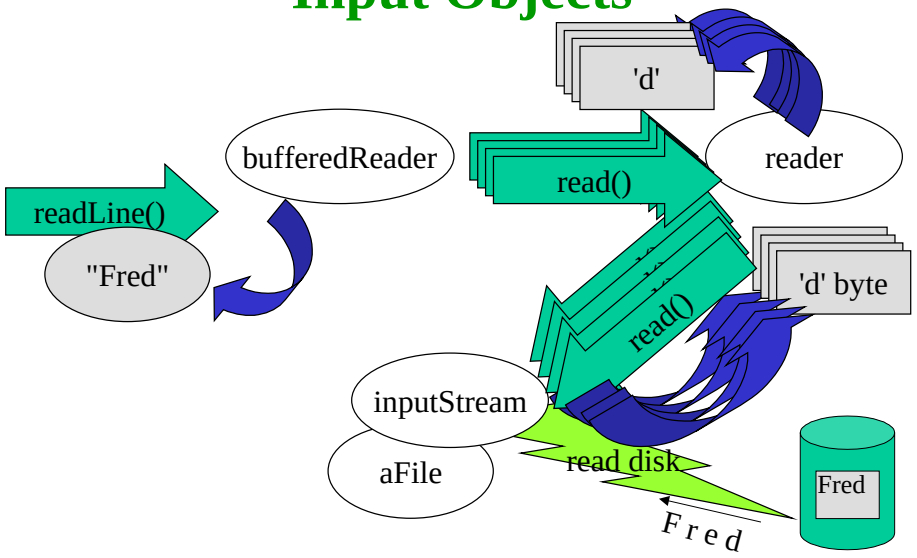
necessary "magic" when creating a FileInputStream

```
import java.io.*;
public class FileOutput {
// This program is an example of file input.

public static void main(String args[]) throws Exception {

    File                aFile;
    FileInputStream     inputStream;
    InputStreamReader   aReader;
    BufferedReader      aBufferedReader;
    String              aString;
```


File Input Example (con't)

```
    aFile = new File("people");
    inputStream = new FileInputStream(aFile);
    aReader = new InputStreamReader(inputStream);
    aBufferedReader = new BufferedReader(aReader);
    aString = aBufferedReader.readLine();
    System.out.println(aString);
     aBufferReader.close();
    }
}
```

Input Objects



Questions

- How to store a collection of objects in a file with objects having different instance variables? (records with many attributes)
- How to update a file? (delete, add, and change records)
- How to sequentially access a list of objects in a file, one after the other?
- How to randomly access an object in a file at a given position?

Summary

- Declare a file stream
- Open file
- Write to file
- Read from file
- Close file

In Java

- Files can only be used with applications (not applets);
- Create a file abstraction object;
  - The file objects recognizes some messages exist(), length(), renameTo(), delete()…;
- Create stream object on the file object;
  - The stream object is either for input or for output ;
  - It receives or sends bytes from/to the file;
- Other streams can be created on top to facilitate operations with strings and numbers;
- A close() message should be sent to the upper stream to close the I/O stream when it is not needed anymore.