FILE IO (Input & Output)

What to do?

- become familiar with the concept of an I/O stream
- understand the difference between binary files and text files
- learn how to save data in a file
- learn how to read data from a file

Outline

- Overview of Streams and File I/O
- Text-File I/O
- Using the `File` Class

I/O Overview

- *I/O* = Input/Output
- In this context it is input to and output from programs
- Input can be from keyboard or a file
- Output can be to display (screen) or a file
- Advantages of file I/O
    - permanent copy
    - output from one program can be input to another
    - input can be automated (rather than entered manually)

Streams

- ***Stream***: an object that either delivers data to its destination (screen, file, etc.) or that takes data from a source (keyboard, file, etc.)
  - it acts as a buffer between the data source and destination
- ***Input stream***: a stream that provides input to a program
  - `System.in` is an input stream
- ***Output stream***: a stream that accepts output from a program
  - `System.out` is an output stream
- A stream connects a program to an I/O object
  - `System.out` connects a program to the screen
  - `System.in` connects a program to the keyboard

Binary vs Text Files

- *All* data and programs are ultimately just zeros and ones
    - each digit can have one of two values, hence *binary*
    - *bit* is one binary digit
    - *byte* is a group of eight bits
- *Text files*: the bits represent printable characters
    - one byte per character for ASCII, the most common code
    - for example, Java source files are text files
    - so is any file created with a "text editor"
- *Binary files*: the bits represent other types of encoded information, such as executable instructions or numeric data
    - these files are easily read by the computer but not humans
    - they are *not* "printable" files
        - actually, you *can* print them, but they will be unintelligible
        - "printable" means "easily readable by humans when printed"

Java: Text versus Binary Files

- Text files are more readable by humans
- Binary files are more efficient
  - computers read and write binary files more easily than text
- Java binary files are portable
  - they can be used by Java on different machines
  - Reading and writing binary files is normally done by a program
  - text files are used only to communicate with humans

Text File I/O

- Important classes for text file **output** (to the file)
  - **`PrintWriter`**
  - **`FileOutputStream`** [or **`FileWriter`**]
- Important classes for text file **input** (from the file):
  - **`BufferedReader`**
  - **`FileReader`**
- **`FileOutputStream`** and **`FileReader`** take file names as arguments.
- **`PrintWriter`** and **`BufferedReader`** provide useful methods for easier writing and reading.
- Usually need a combination of two classes
- To use these classes your program needs a line like the following:

```
import java.io.*;
```

Text File Output

- To open a text file for output: connect a text file to a stream for writing

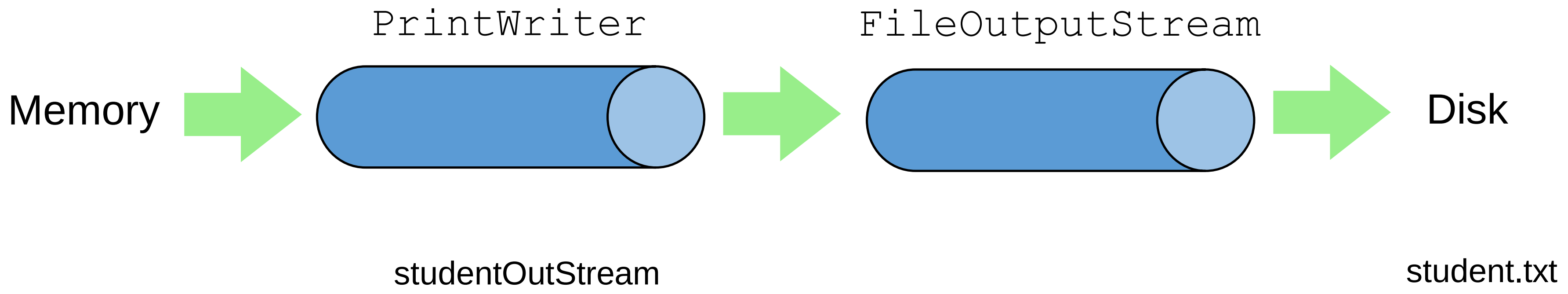
```
PrintWriter outputStream =
  new PrintWriter(new FileOutputStream("out.txt"));
```

- Similar to the long way:

```
FileOutputStream s = new FileOutputStream("out.txt");
PrintWriter outputStream = new PrintWriter(s);
```

- Goal: create a `PrintWriter` object
  - which uses `FileOutputStream` to open a text file
- `FileOutputStream` "connects" `PrintWriter` to a text file.

Output File Streams



```
PrintWriter studentOutStream = new PrintWriter( new FileOutputStream(“student.txt”) );
```

Methods for `PrintWriter`

- Similar to methods for `System.out`
- `println`

```
outputStream.println(count + " " + line);
```

- `print`
- `format`
- `flush`: write buffered output to disk
- `close`: close the `PrintWriter` stream (and file)

TextFileOutputDemo Part 1

```
public static void main(String[]
{
    PrintWriter outputStream =
    try
    {
        outputStream =
             new PrintWriter(new FileOutputStream(“student.txt"));
    }
    catch(FileNotFoundException e)
    {
        System.out.println("Error opening the file student.txt. “
                                   + e.getMessage());
        System.exit(0);
    }
```

A `try`-block is a block: `outputStream` would not be accessible to the rest of the method if it were declared inside the `try`-block

Opening the file

Creating a file can cause the `FileNotFoundException` if the new file cannot be made.

TextFileOutputDemo Part 2

```
System.out.println("Enter three lines of text:");
String line = null;
int count;
    for (count = 1; count <= 3; count++)
    {
        line = keyboard.nextLine();
        outputStream.println(count + " " + line);
    }
    outputStream.close();
    System.out.println("... written to out.txt.");
}
```

Writing to the file

Closing the file

The `println` method is used with two different streams: `outputStream` and `System.out`

Overwriting a File

- Opening an output file creates an empty file
- Opening an output file creates a new file if it does not already exist
- Opening an output file that already exists eliminates the old file and creates a new, empty one
  - data in the original file is lost
- To see how to check for existence of a file, see the section of the text that discusses the `File` class

Java Tip: Appending to a Text File

- To add/append to a file instead of replacing it, use a different constructor for **`FileOutputStream`**:

```
outputStream =
  new PrintWriter(new FileOutputStream(“student.txt", true));
```

- Second parameter: append to the end of the file if it exists?
- Sample code for letting user tell whether to replace or append:

```
System.out.println("A for append or N for new file:");
char ans = keyboard.next().charAt(0);
boolean append = (ans == 'A' || ans == 'a');
outputStream = new PrintWriter(
      new FileOutputStream("out.txt", append));
```

Closing a File

- An output file should be closed when you are done writing to it (and an input file should be closed when you are done reading from it).

- Use the `close` method of the class `PrintWriter (BufferedReader` also has a `close` method`).`

- For example, to close the file opened in the previous example:

```
outputStream.close();
```

- If a program ends normally it will close any files that are open.

Why Bother to Close a File?

If a program automatically closes files when it ends normally, why close them with explicit calls to `close`?

<u>Two reasons:</u>

1. To make sure it is closed if a program ends abnormally (it could get damaged if it is left open).

2. A file opened for writing must be closed before it can be opened for reading.
    - Although Java does have a class that opens a file for both reading and writing, it is not used in this text.

Text File Input

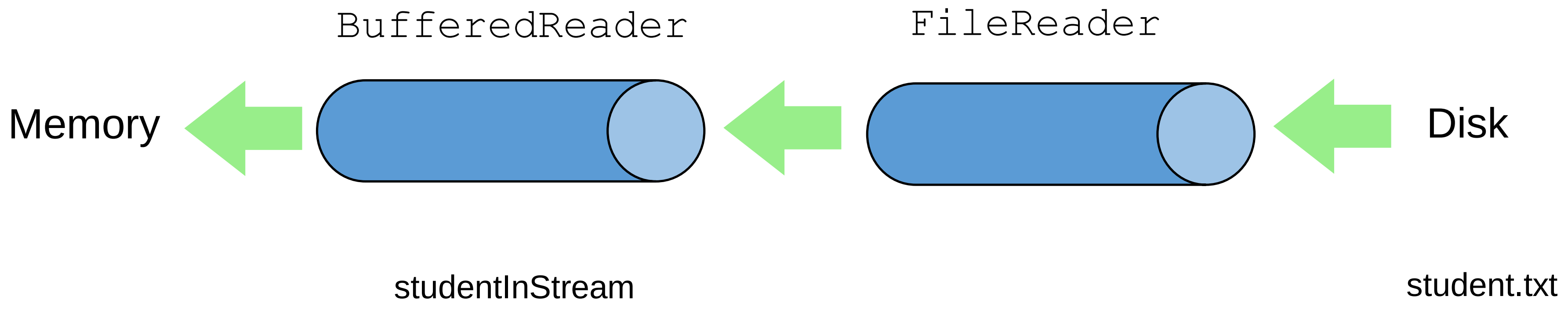
- To open a text file for input: connect a text file to a stream for reading
  - Goal: a `BufferedReader` object,
    - which uses `FileReader` to open a text file
  - `FileReader` “connects” `BufferedReader` to the text file
- For example:

```
BufferedReader studentInStream =
  new BufferedReader(new FileReader(“student.txt"));
```

- Similarly, the long way :

```
FileReader s = new FileReader(“student.txt");
BufferedReader studentInStream = new BufferedReader(s);
```

Input File Streams



```
BufferedReader studentInStream = new BufferedReader( new FileReader("student.txt") );
```

Methods for **`BufferedReader`**

- `readLine`: read a line into a `String`
- no methods to read numbers directly, so read numbers as `Strings` and then convert them (`StringTokenizer` later)
- `read`: read a `char` at a time
- `close`: close `BufferedReader` stream

Exception Handling with File I/O

<u>Catching IOExceptions</u>

- `IOException` is a predefined class
- File I/O might throw an `IOException`
- catch the exception in a catch block that at least prints an error message and ends the program
- `FileNotFoundException` is derived from `IOException`
  - therefor any catch block that catches `IOException`s also catches `FileNotFoundException`s
  - put the more specific one first (the derived one) so it catches specifically file-not-found exceptions
  - then you will know that an I/O error is something other than file-not-found