

Java IO

CIS 226
Intro to Java – Final Lecture

12/13/2002

1

Preliminary Stuff

- ◆ Final Next Week – 901B - 12/18
 - Same time as class
- ◆ Assignment 8 due today
 - However, I'll accept it next week with a print out of the source for NewQuizDialog and QuizGUI.
 - If submitted next week, my grading starts at 90 points, instead of 100.

12/13/2002

2

Java io

- ◆ Provides methods for accessing file, text data, object serialization and internationalization
- ◆ Sequential and Random access
- ◆ Reading and writing of primitive values
- ◆ Applications and applets are provided with three streams automatically
 - System.out (output stream)
 - System.in (input stream)
 - System.err (error stream)

12/13/2002

3

Streams

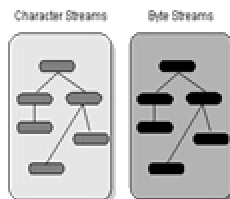
- ◆ Java uses streams to process file data,
- ◆ A sequence of data that has no defined ending.
- ◆ Flowing data

12/13/2002

4

Java.io

- ◆ Contains two types of stream classes
 - Bytes
 - Chars
- ◆ All applications using input and output functionality need to import java.io.
- ◆ Supports random and sequential access to data



12/13/2002

5

Processing Streams

- ◆ While loop
- ◆ End-of-File or some special character
- ◆ Windows Control-Z (EOF)
- ◆ Unix Control-D (EOF)
- ◆ Data comes fast and slow

12/13/2002

6

Character Streams

- Two flavors
 - Reader
 - Writer
- Reader and Writer are abstract classes
- Most programs should use readers and writers to read and write textual information



12/13/2002

7

Byte Streams

- To read and write, programs should use the byte streams, descendants of InputStream and OutputStream
- These streams are typically used to read and write binary data such as images and sounds



12/13/2002

8

More on Streams

`System.out` is an `OutputStream`; specifically it's a `PrintStream`

There's a corresponding `System.in` which is an `InputStream` used to read data from the console.

Data for streams can also come from files. Later you'll see how to use the `File` class and the `FileInputStream` and `FileOutputStream` classes to read and write data from files.

Streams can also be used to access data from an ftp server or a web server

12/13/2002

9

Echo Example

- Most io methods throw `IOException`
- `IOExceptions` occur if the disk or file cannot be opened or read.

12/13/2002

10

File Processing

- File processing with classes in package `java.io`
 - `FileInputStream` for byte-based input from a file
 - `FileOutputStream` for byte-based output to a file
 - `FileReader` for character-based input from a file
 - `FileWriter` for character-based output to a file

12/13/2002

11

Buffering

- Buffering
 - Improves performance of I/O
 - Copies each output to a region of memory called a buffer
 - Entire buffer output to disk at once
 - One long disk access takes less time than many smaller ones
 - `BufferedReader` buffers file output
 - `BufferedOutputStream` buffers file input

12/13/2002

12

Reading

- The basic read() method reads a byte at a time. This is less than perfectly efficient. The following two overloaded variants read multiple bytes into an array of bytes.
- public int read(byte[] data) throws IOException
- public int read(byte[] data, int offset, int length) throws IOException
- The first method tries to read enough bytes to fill the array b. The second method reads length bytes from the input stream and stores them into the array b starting at position offset.
- These methods then return the number of bytes actually read. You should not assume that the array will be filled or that length bytes will actually have been read. If the end of stream is encountered, -1 is returned

12/13/2002

13

Writing

- abstract void **close()**
Close the stream, flushing it
- abstract void **flush()**
Flush the stream. void **write(char[] cbuf)**
Write an array of characters.
- abstract void **write(char[] cbuf, int off, int len)**
Write a portion of an array of characters.
- void **write(int c)**
Write a single character. void **write(String str)**
Write a string. void **write(String str, int off, int len)**
Write a portion of a string.

12/13/2002

14

java.io.File

- This is the main class used for file and directory manipulation in a platform independent way.
- User interfaces and operating systems use system-dependent *pathname strings* to name files and directories. This class presents an abstract, system-independent view of hierarchical pathnames.

12/13/2002

15

Reading a File – Type Ex.

- Uses FileInputStream
- Gets filename from the command-line
- Retrieves the size of the file and the data
- You create a new FileInputStream object by passing the name of the file to the constructor, like this:

```
FileInputStream fis = new FileInputStream("14.html");
```

12/13/2002

16

Writing to a File – MultiType Ex.

- Uses FileOutputStream
- Gets filename from the command-line to write to
- Uses
 - public void write(byte[] data, int offset, int length) throws IOException
 - b - the data.
 - off - the start offset in the data.
 - len - the number of bytes to write
- If the file exists in the current directory, it will be overwritten by the new data. If the file doesn't exist it will be created in the current directory.
- You create a new FileOutputStream object by passing the name of the file to the constructor, like this:

```
FileOutputStream fos = new FileOutputStream("16.html");
```

12/13/2002

17

Appending data

- It's often useful to be able to append data to an existing file rather than overwriting it. To do this just pass the boolean value true as the second argument to the FileOutputStream() constructor. For example,

```
FileOutputStream fos = new FileOutputStream("16.html", true);
```

12/13/2002

18

Buffered Input and Output

- `BufferedInputStream` and `BufferedOutputStream`
 - These classes buffer reads and writes by reading data first into a buffer (an internal array of bytes). Thus an application can read bytes from the stream without necessarily calling the underlying native method. The data is read from or written into the buffer in blocks; subsequent accesses go straight to the buffer.
 - More efficient way of handling data reads and writes
- `public BufferedInputStream(InputStream in)`
- `public BufferedInputStream(InputStream in, int size)`
- `public BufferedOutputStream(OutputStream out)`
- `public BufferedOutputStream(OutputStream out, int size)`

12/13/2002

19

File Class

- `java.io.File`
- Main class for manipulating files and directories
- System Independent fashion
 - Path names are specific to a particular Operating System
 - Ex. `C:\projects\myfile.txt` – windows
 - `/home/projects/myfile.txt` – unix
- A `File` object is not a file handle. Just because you have a `File` object does not mean that the equivalent file actually exists on the disk. There are methods you can use to determine whether a `File` object refers to a real file or not (specifically `exists()`).

12/13/2002

20

File Example

- File Info
 - Uses methods from the `File` class to retrieve information about a particular file.

12/13/2002

21

Reading Files

- The `FileReader` class reads text files using the platform's default character encoding and the buffer size. If you need to change these values, construct an `InputStreamReader` on a `FileInputStream` instead.

```
public FileReader(String fileName) throws FileNotFoundException
public FileReader(File file) throws FileNotFoundException
public FileReader(FileDescriptor fd)
```

- Only the constructors are declared in this class. For example,
- `FileReader fr = new FileReader("36.html");`

12/13/2002

22

Writing to text files

- The `java.io.FileWriter` class writes text files using the platform's default character encoding and the buffer size. If you need to change these values, construct an `OutputStreamWriter` on a `FileOutputStream` instead.

```
public FileWriter(String fileName) throws IOException
public FileWriter(String fileName, boolean append) throws IOException
public FileWriter(File file) throws IOException
public FileWriter(FileDescriptor fd)
```

Only the constructors are declared in this class. For example,

- `FileWriter fw = new FileWriter("36.html");`

12/13/2002

23

Line Numbering – Line Number Ex.

- The `java.io.LineNumberReader` class is a subclass of `java.io.BufferedReader` that keeps track of which line you're currently reading. It has all the methods of `BufferedReader` including `readLine()`. It also has two constructors, `getLineNumber()`, and `setLineNumber()` methods:

```
public LineNumberReader(Reader in)
public LineNumberReader(Reader in, int size)
public void setLineNumber(int lineNumber)
public int getLineNumber()
```

- The `setLineNumber()` method does not change the file pointer. It just changes the value `getLineNumber()` returns.
- For example, it would allow you to start counting from -5 if you knew there were six lines of header data you didn't want to count.

12/13/2002

24

Final Exam – Next Week

- 901B
- 7pm
- You will have to implement a java application with a JFrame , JMenuBar, JMenus, JMenuItem, JTextFields, JButtons, and appropriate event listeners
- Program will be required to do specific tasks
- Program will be written from scratch by YOU
- Less complex than the last homework

12/13/2002

25

That's it

- **Questions**
- **Pleasure having you in my class.**
- **See you next week**
- **No department surveys**
- **Student Surveys can be submitted at**
 - <http://www.whototake.com>
 - **Who to take!**

12/13/2002

26