Java I/O

Reading, Writing, and stuff

Announcement

- Office hour Wednesday…
  - Moved to 1-2 rather than 2-3.

Project Announcements

- CS Labs will NOT be open during break.
  - Machines may be available remotely.
    - Still waiting on confirmation.
- New due dates:
  - Minimum submission: Sunday, January 11th
  - Final submission: Sunday, January 18th

Project Announcement

- You'll need to create an "empty Customer" class in order to test CustomerQueue
  - Constructor
  - toString();

Project Announcements

- Project Grade
  - Clock – 10 points
  - CustomerQueue – 30 points
  - Customer – 30 points
  - Register – 30 points

Java I/O

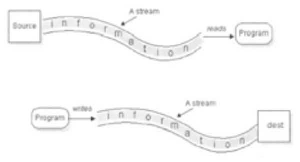
- For the next couple of classes we will be talking about Java I/O
  - This class: basics and low level I/O
  - Next class: "wrappers" and high level I/O
- All Java I/O classes are defined in the `java.io` package.

Java I/O

- Low level vs high level
  - Low level: can only read/write a character or byte at a time
  - High level: can read/write strings that represent different data types
    - Ex. read/write an int, float,

Streams

- Basic low level mechanism for I/O in Java is the stream



Streams

- Reading from a stream
  - Open a stream
  - While more info
    - Read data
  - Close the stream
- Writing to a stream
  - Open a stream
  - While more info
    - Write data
  - Close the stream

Data and Streams

- Types of data that can be read from/written to streams
  - Bytes (8-bits / bytes)
    - Raw data
  - Characters (16-bits / bytes)
    - Text data
- Basic stream operations
  - Read
  - Write

The 4 base Java I/O classes

	READ	**WRITE**
CHAR	`Reader`	`Writer`
BYTE	`InputStream`	`OutputStream`

Each of these are abstract classes

A look at `Reader`

- `public abstract class Reader {`
  - `public int read() throws IOException`
    - Returns int in range (0 – 65535)
  - `public int read(char[] cbuf) throws IOException`
  - `public abstract int read(char[] cbuf, int off, int len) throws IOException`
    - Returns number of chars read

    `}`
- Returns –1 at end of stream

A look at `Reader`

- Also contains functions for
  - Marking a location in a stream
  - Skipping input
  - Resetting current position
  - Close the stream

A look at `InputStream`

- `public abstract class InputStream{`
  - `public int read() throws IOException`
    - Returns int in range (0 – 255)
  - `public int read(byte[] cbuf) throws IOException`
  - `public abstract int read(byte[] cbuf, int off, int len) throws IOException`
    - Returns number of bytes read
      `}`
- Returns –1 at end of stream

A look at `InputStream`

- Also contains functions for
  - Marking a location in a stream
  - Skipping input
  - Resetting current position
  - Close the stream

A look at `Writer`

- `public abstract class Writer{`
  - `public void write(int c) throws IOException`
    - Only low order 16 bits are written
  - `public void write(char[] cbuf) throws IOException`
  - `public abstract void write(char[] cbuf, int off, int len) throws IOException`

A look at `Writer`

- Also contains functions for
  - Writing strings
  - Flushing the stream
  - Close the stream

A look at `OutputStream`

- `public abstract class OutputStream{`
  - `public void write(int b) throws IOException`
    - Only low order 8 bits are read
  - `public void write(byte[] cbuf) throws IOException`
  - `public abstract void write(byte[] cbuf, int off, int len) throws IOException`
- Also contains functions for
  - Flushing the stream
  - Close the stream

Observations

- Almost every operation will throw an IOException if something goes wrong
- These classes are abstract!
  - Don't indicate how a read/write is to be done
  - Don't indicate where the data is coming from or going to.
  - These details will be filled in by subclasses.

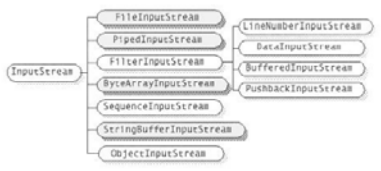
Files

- File Object
  - abstract representation of file and directories.
  - Encapsulates all details of files and how they are named.
  - Create a File object by providing the filename to the File constructor
    - `File F = new File (“tmp/input.txt”);`

Pipes

- Streams where the output of one process becomes the input of another
  - In UNIX: `ls –l | more`
- In Java, you can have independent processes running. Each is called a thread.
  - Pipes are used to let the output of one thread be the input of another
- More when you get to CS3

Subclassing InputStream



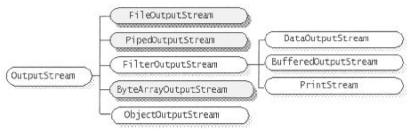
Subclassing InputStream

- Based on where data is coming from
  - File
    - `FileInputStream`
  - In Memory
    - `ByteArrayInputStream`
    - `StringBufferInputStream` (going away)
  - Pipe
    - `PipedInputStream`

Look at the `FileInputStream`

- `public FileInputStream(String name)throws FileNotFoundException`
- `public FileInputStream(File file) throws FileNotFoundException`
- Implements methods of InputStream class.

Subclassing OutputStream



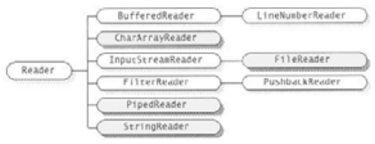
Subclassing OutputStream

- Based on where data is going
  - File
    - `FileOutputStream`
  - In Memory
    - `ByteArrayOutputStream`
  - Pipe
    - `PipedOutputStream`

Look at the FileOutputStream

- `public FileOutputStream(String name, boolean append)throws FileNotFoundException`
- `public FileOutputStream(File file, boolean append)  throws FileNotFoundException`
- Implements methods of OutputStream class.

Subclassing Reader



Subclassing Reader

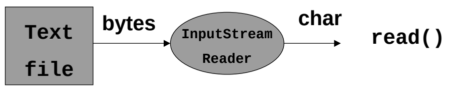
- Based on where data is coming from
  - File
    - `FileReader`
  - In Memory
    - `CharArrayReader`
    - `StringReader`
  - Pipe
    - `PipedReader`

Look at FileReader

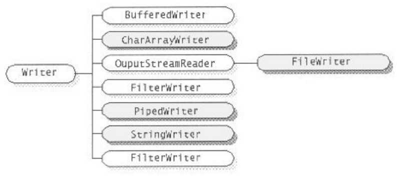
- `public FileReader(String name)throws FileNotFoundException`
- `public FileReader(File file) throws FileNotFoundException`
- Implements methods of Reader class.
- Actually a subclass of `InputStreamReader`

InputStreamReader

- Converts read bytes to characters



Subclassing Writer



Subclassing Writer

- Based on where data is going
  - File
    - `FileWriter`
  - In Memory
    - `CharArrayWriter`
    - `StringWriter`
  - Pipe
    - `PipedWriter`

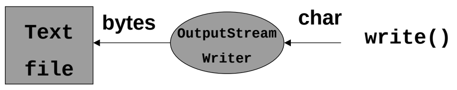
Look at FileWriter

- `public FileWriter(String name, boolean append)throws FileNotFoundException`
- `public FileWriter(File file, boolean append)  throws FileNotFoundException`

- Implements methods of `Writer` class.
- Actually a subclass of `OutputStreamWriter`

OutputStreamWriter

- Converts characters to write to bytes



Let’s look at some code

SimpleCopy – does copying of binary files

SimpleTextCopy – does copying of text files.

SimpleArrayCopy – like SimpleCopy but uses arrays

SimpleTextArrayCopy – like SimpleTextCopy but uses arrays

SimpleCopy

```
try {
   FileInputStream fin = new FileInputStream (args[0]);
   FileOutputStream fout = new FileOutputStream
  (args[1]);

   int n=0;
   while ((n = fin.read()) != -1)
        fout.write (n);

   fin.close();
   fout.close();
}
catch (IOException E){
     System.out.println ("Problem: " + E.getMessage());
}
```

SimpleTextCopy

```
try {
   FileReader fin = new FileReader (args[0]);
   FileWriter fout = new FileWriter (args[1]);

   int n=0;
   while ((n = fin.read()) != -1)
        fout.write (n);

  fin.close();
  fout.close();
}
catch (IOException E){
    System.out.println ("Problem: " + E.getMessage());
}
```

SimpleArrayCopy

```
try {
   FileInputStream fin = new FileInputStream (args[0]);
    FileOutputStream fout = new FileOutputStream (args[1]);

   int buflen = 80;
   byte buffer[] = new byte[buflen];
   int n;
   while ((n = fin.read(buffer)) != -1)
      fout.write (buffer,0,n);
   fin.close();
   fout.close();
}
catch (IOException E){
   System.out.println ("Problem: " + E.getMessage());
}
```

SimpleTextArrayCopy

```
try {
   FileReader fin = new FileReader (args[0]);
   FileWriter fout = new FileWriter (args[1]);

   int buflen = 80;
   char buffer[] = new char[buflen];
   int n;
   while ((n = fin.read(buffer)) != -1)
        fout.write (buffer,0,n);
   fin.close();
   fout.close();
}
catch (IOException E){
   System.out.println ("Problem: " + E.getMessage());
}
```

Summary

- Basic I/O mechanism is streams
- Streams for read / write
- Streams to chars / bytes
- Reader, Writer, InputStream, OutputStream
- File Object
- Subclassing based on source / destination.
- IOExceptions

Tomorrow

- “wrapping” a class
- Higher level I/O classes.