Topic #10: Java Input / Output

CSE 413, Autumn 2004

Programming Languages

http://www.cs.washington.edu/education/courses/413/04au/

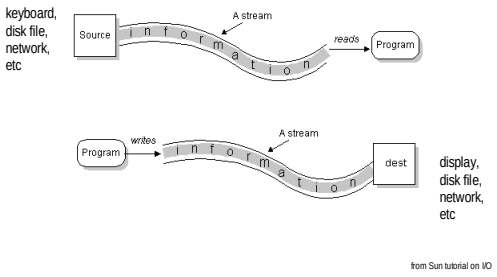
Readings and References

- Reading
- Other References
  - » Section "I/O" of the Java tutorial
  - » http://java.sun.com/docs/books/tutorial/essential/io/index.html

Input & Output

- Program input can come from a variety of places:
  - » the mouse, keyboard, disk, network…
- Program output can go to a variety of places:
  - » the screen, speakers, disk, network, printer…

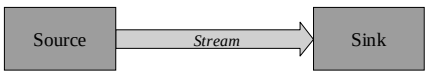
"Streams" are the basic I/O objects



from Sun tutorial on I/O

The stream model

- The stream model views all data as coming from a source and going to a sink



- Sources and sinks can be files, memory, the console, network ports, serial ports, etc

Streams

- Getting data from source to sink is the job of an object of a *stream* class
- Use different streams for doing different jobs
- Streams appear in many packages
  - » java.io - basic stream functionality, files
  - » java.net - network sockets
  - » javax.comm - serial ports
  - » java.util.zip - zip files

Streams are *layered* classes

- Inheritance and composition both play key roles in defining the various types of streams
- Each layer adds a little bit of functionality
- The nice thing about this design is that many programs don't need to know exactly what kind of stream they are working with

7

Classes of Streams

1. Byte streams
   - » *InputStream* and *Output Stream*
   - » Binary data: sounds, images, etc.
   - » Use this for binary data or primitive objects
2. Character-based streams
   - » *Reader* and *Writer*
   - » Use this if working with text

Mismatch?

- » If you get an InputStream or OutputStream from somewhere else, you can convert it to a Reader or a Writer as needed by wrapping it with an InputStreamReader or OutputStreamWriter

8

OutputStream

- An OutputStream sends bytes to a sink
  - » OutputStream is an abstract class
  - » the actual "write" method depends on the device being written to
- Key methods:

```
abstract void write(int b) throws IOException
void write(byte[] b) throws IOException
void close() throws IOException
```

9

OutputStream subclasses

- Subclasses differ in how they implement write() and in what kind of sink they deal with:
  - » FileOutputStream: sink is a file on disk
  - » ByteArrayOutputStream: sink is an array of bytes
  - » PipedOutputStream: sink is a pipe to another thread
- Other subclasses process output streams
  - » FilterOutputStream: process the stream in transit
  - » ObjectOutputStream: primitives and objects to a sink

10

FilterOutputStream

- Constructor takes an instance of OutputStream
- Resulting object is also instance of OutputStream
- These classes *decorate* the basic OutputStream implementations with extra functionality
- Subclasses of FilterOutputStream in java.io:
  - » BufferedOutputStream: adds buffering for efficiency
  - » PrintStream: supports display of data in text form (using the default encoding only)
  - » DataOutputStream: write primitive data types and Strings (in binary form)

11

InputStream

- An InputStream gets bytes from a source
  - » InputStream is an abstract class
  - » The actual "read" method depends on the source being read from
  - » Key methods:

```
abstract int read() throws IOException
int read(byte[] b) throws IOException
void close() throws IOException
```

12

InputStream subclasses

- Subclasses differ in how they implement read() and in what kind of source they deal with:
  - » FileInputStream: source is a file on disk
  - » ByteArrayInputStream: source is an array of byte
  - » PipedInputStream: source is pipe from another thread
- Other subclasses process input streams
  - » FilterInputStream: process the stream in transit
  - » ObjectInputStream: primitives and objects from a source

13

FilterInputStream

- Constructor takes an instance of InputStream
- Resulting object is also instance of InputStream
- These classes "decorate" the basic InputStream implementations with extra functionality
- Some useful subclasses
  - » BufferedInputStream: adds buffering for efficiency
  - » ZipInputStream: read zip files
  - » DataInputStream: read primitive data types and Strings (in binary form)

14

Reader and Writer

- Reader and Writer are abstract classes that are Unicode aware and can use a specified encoding to translate Unicode to/from bytes
- Subclasses implement most of the functionality
  - » BufferedReader, BufferedWriter
    - add buffering for efficiency
  - » StringReader, StringWriter
  - » PipedReader, PipedWriter
  - » InputStreamReader, OutputStreamWriter

15

System.in, System.out

- System.in is a predefined InputStream
- Can convert to a BufferedReader:

```
BufferedReader r =
       new BufferedReader(new InputStreamReader(System.in));
```

- System.out is a predefined OutputStream (a PrintStream)
- Can convert to a PrintWriter like this:

```
PrintWriter w =
       new PrintWriter(new OutputStreamWriter(System.out),true);
```

16

Read a String from the console

```
/* ask for the names we were not given */

BufferedReader console =
       new BufferedReader(new InputStreamReader(System.in));

for (int i=count; i<3; i++) {
       System.out.print("name "+i+"? ");
       String petName = console.readLine();
       if (petName == null) {
              petName = "<blank>";
       }
       names.add(petName);
}
```

17

Sources and Sinks - Console

- When reading from the console
  - » Source:
  - » Sink:
- When writing to the console
  - » Source:
  - » Sink:

18

Sources and Sinks - Files

- When reading from a file
  - » Source:
  - » Sink:
- When writing to a file
  - » Source:
  - » Sink:

19

FileInputStream and FileOutputStream

- The file streams read or write from a file on the native file system
  - » FileInputStream
    - retrieve bytes from a file and provide them to the program
  - » FileOutputStream
    - send bytes to a file from your program
- If used by themselves, FileInputStream and FileOutputStream are for binary I/O
  - » just plain bytes in and out with no interpretation as characters or anything else

20

"bytes from a file" plus "bytes as text"

- Create new FileInputStream and connect it to a specific file
- "decorate" the stream with an InputStreamReader that will do Unicode translation for you

```
FileInputStream(String name)

InputStreamReader(InputStream in)

InputStreamReader(InputStream in, String enc)
```

21

"bytes from a file as text"

- Shortcut: Create FileReader and connect it to a file
  - » Uses default encoding and buffer sizes.

```
FileReader(File file)

FileReader(FileDescriptor fd)

FileReader(String fileName)
```

22

prepare to read a file

```
public TextRead(String fn) throws IOException {
        InputStream in;
        in = new FileInputStream(fn);
        textReader = new BufferedReader(new InputStreamReader(in));
}
```

23

readline()

- Read one line from a BufferedReader

```
/**
* Read one line from the text file and return it as a String to the caller.
* Note that the line might be null (at end of file), empty (0 characters) or
* blank (all whitespace).  Of course, it might also be a non-blank String with
* some useful characters in it.
* @return a String containing the next line or null if
* we are at the end of the file
*/
private String getNextLine() throws IOException {
        return textReader.readLine();
}
```

24

Detecting end of file

- End of file is expected when using readline()
- So the method returns **`null`** if we are end of file

```
String myLine = tr.getNextLine();
while (myLine != null) {
        System.out.println(">> "+myLine);
        myLine = tr.getNextLine();
}
```

25

close when done

- After reading through the file, you should close the stream, since an open file takes up system resources and prevents other programs from using the file

```
/**
* Close the stream.
*/
public void close() throws IOException {
        textReader.close();
}
```

26

"bytes to a file as text"

- Create new PrintWriter and connect it to a file using a FileWriter
  - » PrintWriter provides the text formatting capabilities
  - » FileWriter provides the connection between the PrintWriter and the actual file
  - » FileWriter is a convenience class like FileReader
    - could use OutputStreamWriter with a FileOutputStream

`PrintWriter(Writer out)`
Create a new PrintWriter, without automatic line flushing.

`FileWriter(String fileName)`
Constructs a FileWriter object given a file name.

27

prepare to write a file

```
public TextRW(String fn) throws IOException {
    File sink = new File(fn);
    sink.createNewFile();
    System.out.println("Created "+sink.getAbsolutePath());
    textWriter = new PrintWriter(new BufferedWriter(new FileWriter(sink)));
}
```

28

println(...)

- Print formatted representations of objects and primitive type to a text-output stream

```
/**
* Write one line on the output file.
* @param line the line of text to write out
*/
public void writeOneLine(String s) {
        textWriter.println(s);
}
```

29

close when done

- After writing the file, you should close the stream

```
/**
* Close the stream.
*/
public void close() throws IOException {
    textWriter.close();
}
```

- Why?

30

The File class

- Manages an entry in a directory (a pathname)
- Several constructors are available
  - » File(String pathname)
    - pathname string
  - » File(String parent, String child)
    - parent pathname string and a child pathname string.
  - » File(File parent, String child)
    - parent abstract pathname and a child pathname string.
- The File() constructors create a pathname object in memory, NOT a new file on disk

31

File class examples

```
File f = new File("c:\autoexec.bat");

File app = new File("c:\apps\JPadPro","JPadPro.exe");

File jppDir = new File("c:\apps\JPadPro");
File jppApp = new File(jppDir, "JPadPro.exe");
```

32

File class methods

- Create, rename, delete a file
  - » createNewFile(), createTempFile(), renameTo(), delete()
- Determine whether a file exists and access limitations
  - » exists(), canRead(), canWrite()
- Get file info
  - » getParent(), getCanonicalPath(), length(), lastModified()
- Create and get directory info
  - » mkdirs(), list(), listFiles(), getParent()
- Etc, etc

33

APPENDIX - Writing output to the console

- Java provides standard PrintStream System.out
  - » has methods to print text to the console window
- Some operations:
  - System.out.println(<expression>);
  - System.out.print(<expression>);
- expression can be
  - » primitive type: an int, double, char, boolean
  - » or an object of any class type

34

Printing objects on System.out

- Any object can be printed on System.out
  - Rectangle rect = new Rectangle(30,50,100,150,Color.blue,true);
  - System.out.println(rect);
- Can be very useful for debugging
  - » Put System.out.print or println method calls in your code to display a message when that place is reached during execution
  - » Particularly useful if the string version of the object has useful information in a readable format

35

Object Representation on System.out

- What actually happens when an object is printed?
  - » The toString() method belonging to the object provides the string to be printed
  - » All classes have a default toString( ), the one defined by the Object class (not very descriptive)

```
public String toString() {
return getClass().getName()+"@"+Integer.toHexString(hashCode());
}
```

  - » But you can provide a custom version of toString() in any of your classes very easily

36