Input / Output Streams

CSE 142, Summer 2003

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/03su/

Readings and References

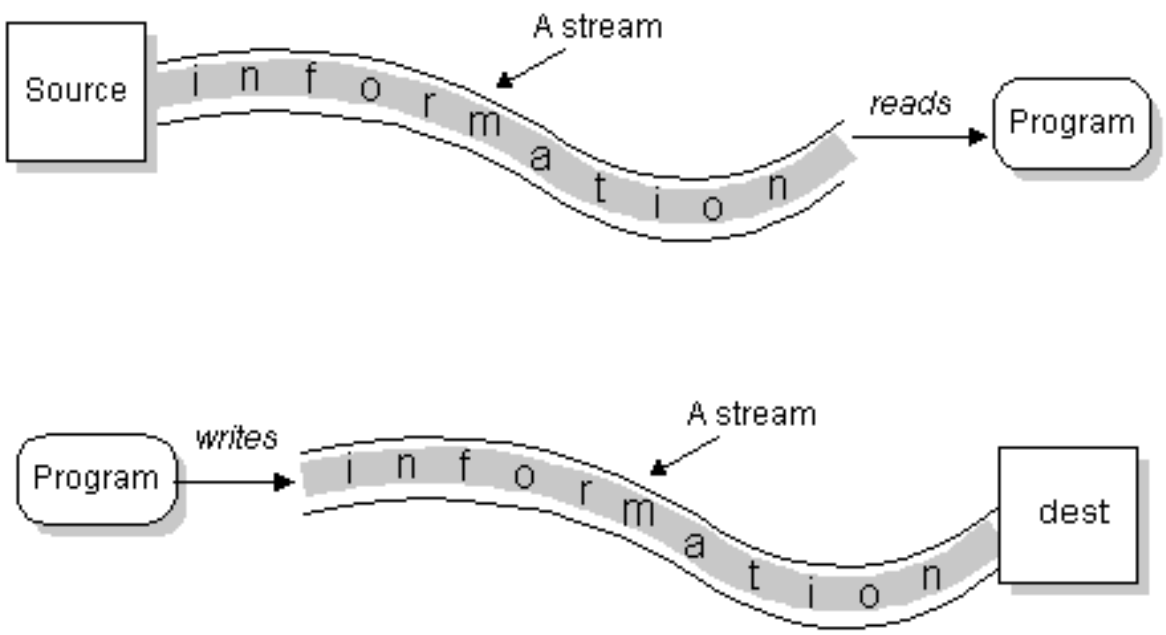
- Reading
  - » Appendix A, *Intro to Programming and Object-Oriented Design Using Java*, Niño and Hosch
- Other References
  - » Section "I/O" of the Java tutorial
  - » http://java.sun.com/docs/books/tutorial/essential/io/index.html

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

keyboard,
disk file,
network,
etc

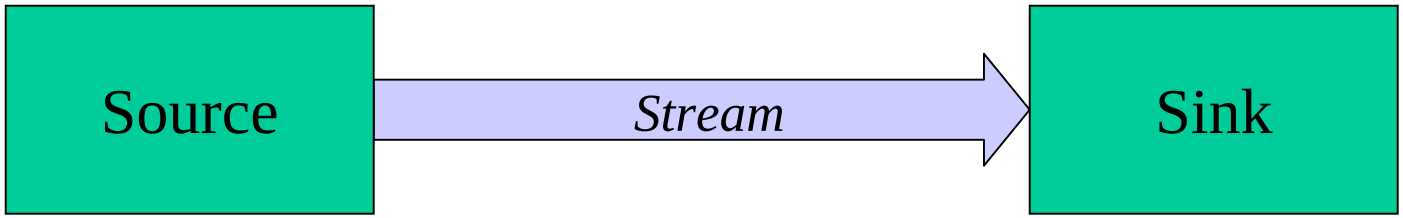


display,
disk file,
network,
etc

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

- Each layer adds a little bit of functionality
  - » in some ways, this is similar to how the ArrayList class adds functionality to the simple array
- One nice thing about this design is that many programs don't need to know exactly what kind of stream they are working with
  - » one OutputStream is as good as another in many situations, as long as it knows how to move data

`OutputStream`

- An **`OutputStream`** sends data to a sink
  - » **`OutputStream`** is an *abstract class*
    - implements some but not all of a group of methods
  - » subclasses *extend* this abstract class and implement the actual **`write`** method depending on the device
- Key methods:

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

Inheritance and Subclasses

- We can extend a class in order to add new methods to the basic class
  - » This is the fundamental concept of *inheritance*
- The basic class might be complete in itself
  - » for example, Object
  - » extend it to define a new class and add new methods
- The basic class might be incomplete
  - » for example, OutputStream
  - » extend it to define a new class and add missing and new methods

java.io

Class OutputStream

OutputStream is an abstract class

java.lang.Object
  |
  +--**java.io.OutputStream**

Direct Known Subclasses:

ByteArrayOutputStream, FileOutputStream, FilterOutputStream, ObjectOutputStream, OutputStream, PipedOutputStream

---

public abstract class **OutputStream**
extends Object

This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

java.io

Class FileOutputStream

FileOutputStream *extends* OutputStream to write to disk files

java.lang.Object
  |
  +--java.io.OutputStream
        |
        +--**java.io.FileOutputStream**

---

public class **FileOutputStream**
extends OutputStream

A file output stream is an output stream for writing data to a `File` or to a `FileDescriptor`.

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

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)

buffer picture

`InputStream`

- An **`InputStream`** gets bytes from a source
  - » **`InputStream`** is an abstract class
    - implements some but not all of a group of methods
  - » subclasses *extend* this abstract class and implement the actual **`read`** method depending on the device
- Key methods:

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

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

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)

```
InputStream in = new FileInputStream(fn);
textReader = new BufferedReader(new InputStreamReader(in));
```

from ex21\TextFileRead.java

Sources and Sinks - Console

- When reading from the console
  - » the keyboard is the source
  - » a data structure in your application is the sink
- When writing to the console
  - » a data structure in your application is the source
  - » the monitor (terminal window) is the sink

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
  - » InputStreamReader, OutputStreamWriter
    - rely on the underlying streams to actually move bytes
  - » BufferedReader, BufferedWriter
    - add buffering for efficiency
  - » StringReader, StringWriter
  - » PipedReader, PipedWriter

Reader and Writer guidelines

- In general:
  - » If you're working with text (Strings and chars), use Readers and Writers
  - » If you're working with primitive data types, use InputStreams and OutputStreams
  - » 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

System.in, System.out

- System.in is a predefined InputStream
- You can convert to a BufferedReader like this:

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

- System.out is a predefined OutputStream
- You can convert to a PrintWriter like this:

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