Input and Output

CSE 142, Summer 2002

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/02su/

Readings and References

- Reading
  - » Section A.2, *An Introduction to Programming and Object Oriented Design using Java*, by Niño and Hosch
  - » Chapter 5 (end of chapter), *Introduction to Programming in Java*, Dugan
- Other References
  - » Section "I/O" of the Java tutorial
  - » http://java.sun.com/docs/books/tutorial/essential/io/index.html

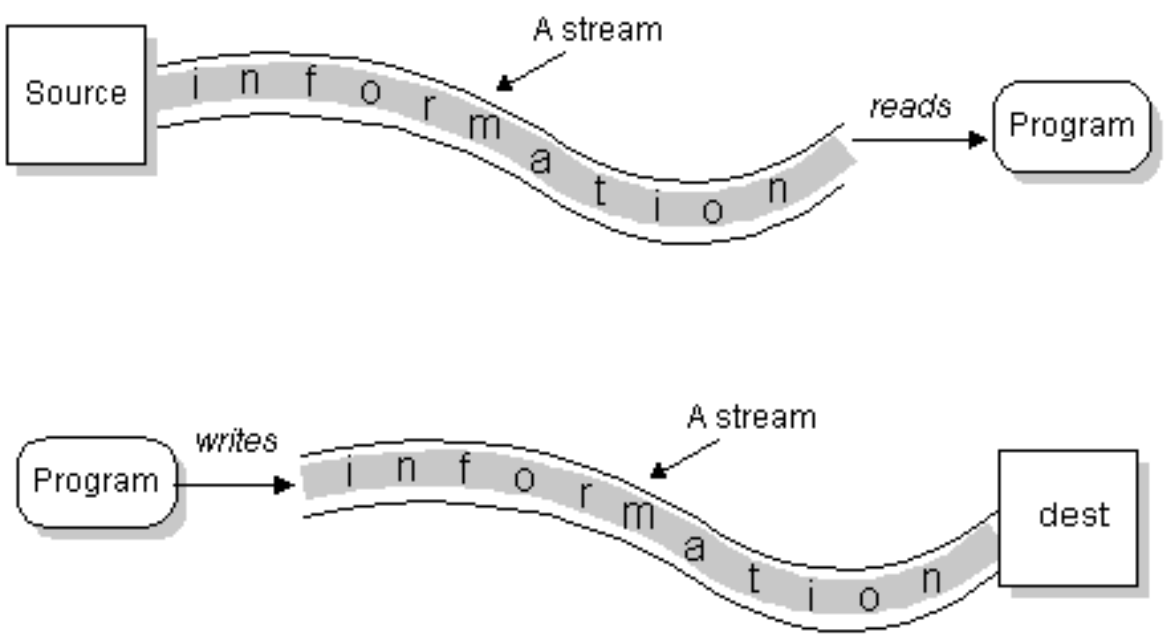
Input & Output

- Most programs perform both *input* and *output*
  - » Think about a video game with no display
  - » Or an ATM that doesn't let you enter your PIN!

    or worse yet, doesn't give you any money ...
- Output can go to a variety of places:
  - » the screen, speakers, disk, network, printer…
- Input can come from a variety of places:
  - » the mouse, keyboard, disk, network…

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

Sources and Sinks - Files

- When reading from a file
  - » the file is the source
  - » a data structure in your application is the sink
- When writing to a file
  - » a data structure in your application is the source
  - » the file is the 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 a *stream*
- 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
  - » one OutputStream is as good as another in many situations, as long as it knows how to move bytes

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

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
  - » FilterOutputStream: process the stream in transit
- There are many more subclasses
  - » ObjectOutputStream: primitives and objects to a sink
  - » ByteArrayOutputStream: sink is an array of bytes
  - » PipedOutputStream: sink is a pipe to another thread

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 in java.io:
  - » PrintStream: supports display of data (in text form)
  - » BufferedOutputStream: adds buffering for efficiency
  - » DataOutputStream: supports writing primitive data types and Strings (in binary form)

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

Printing primitives on System.out

- System.out is a PrintStream object
- PrintStream defines a whole bunch of print(…) methods, one for each type

```
void print(boolean b)
void print(char c)
void print(char[] s)
void print(double d)
void print(float f)
void print(int i)
void print(long l)

void print(Object obj)
void print(String s)
```

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

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

toString() in Vehicle & LocatedVehicle

```
/**
* Provide a String representation of this Vehicle.
* @return a String describing this Vehicle
*/
public String toString() {
       return this.getClass().getName()+" "+vin;
}
```

```
/**
* Provide a String representation of this LocatedVehicle.
* @return a String describing this LocatedVehicle
*/
public String toString() {
       return super.toString()+" at "+location.toString();
}
```

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
```

InputStream subclasses

- Subclasses differ in how they implement read() and in what kind of source they deal with:
  - » FilterInputStream: process the stream in transit
  - » FileInputStream: source is a file on disk
- There are many more subclasses
  - » ByteArrayInputStream: source is an array of byte
  - » PipedInputStream: source is pipe from another thread
  - » 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
  - » ObjectInputStream: read primitive data types and objects
  - » ZipInputStream: read zip files

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 functionality
  - » InputStreamReader, OutputStreamWriter
  - » StringReader, StringWriter
  - » PipedReader, PipedWriter
  - » BufferedReader, BufferedWriter

Reader and Writer guidelines

- In general:
  - » If you're working with text (Strings and chars), use Reader and Writer
  - » If you're working with primitive data types, use InputStream and OutputStream
  - » If you get an InputStream or OutputStream from somewhere else, you can convert to Reader/Writer if needed

System.in, System.out

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

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

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

```
Writer w = new OutputStreamWriter(System.out));
```

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);
}
```

this is from PetSetB.java, ex142\lect12