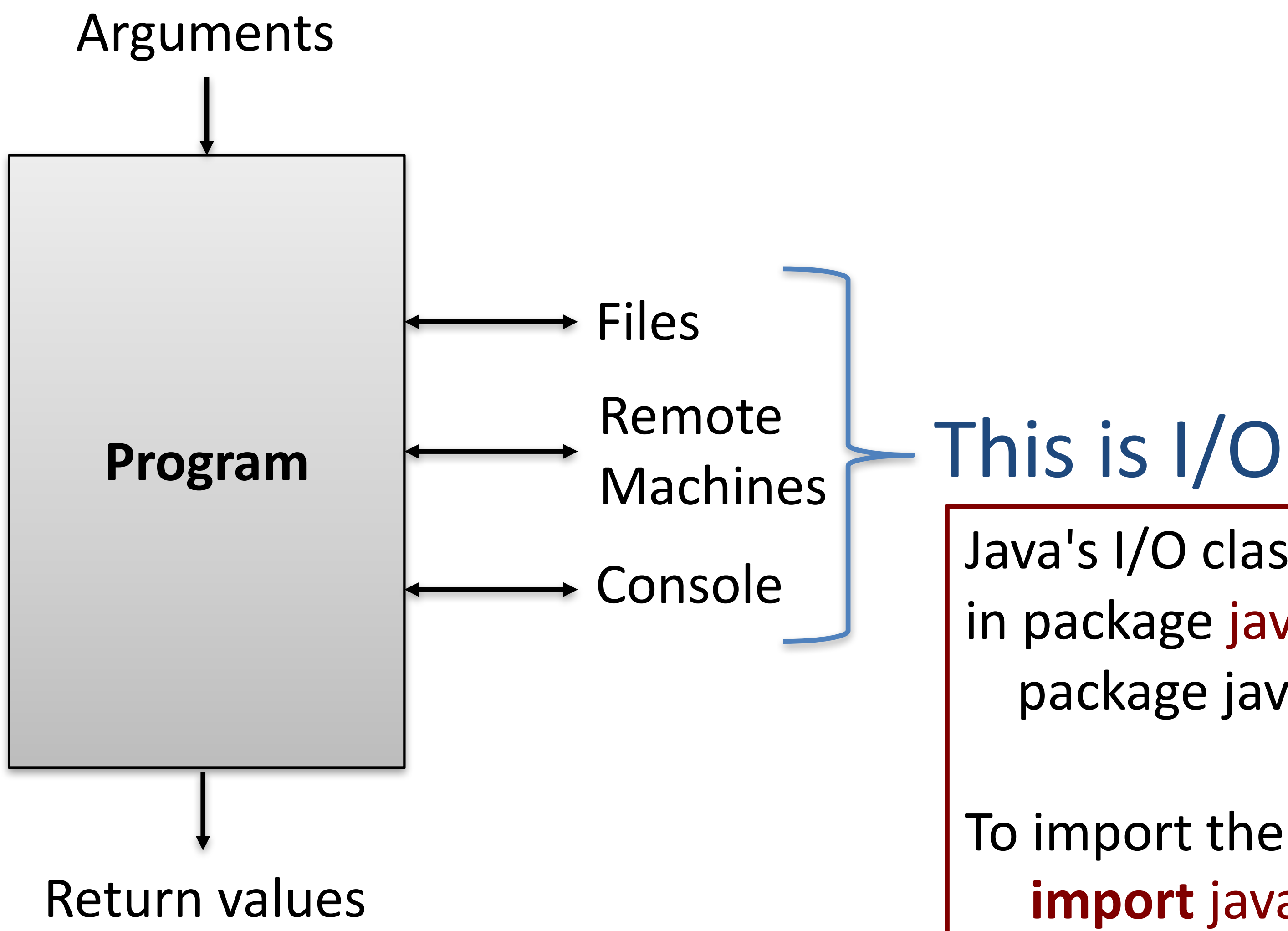
Program Input/Output (I/O)

CS2110

Recitation 8

Program Input/Output



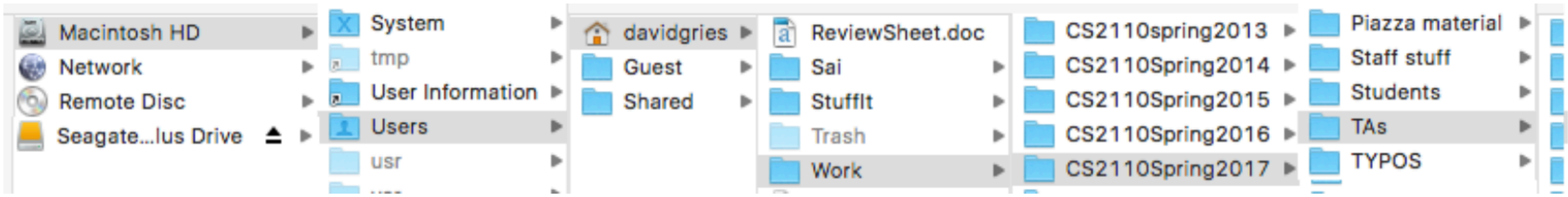
Java's I/O classes are in package java.io
package java.nio

To import the classes:

```
import java.io.*;
import java.nio.*;
```

Files

- Files (and directories) are identified by paths



- File system on a hard disk is structured as a tree
  - leaves are files (or empty directories)
  - Internal nodes are directories (aka folders)

Interface Path

An object of type Path contains the path name to a file or directory.

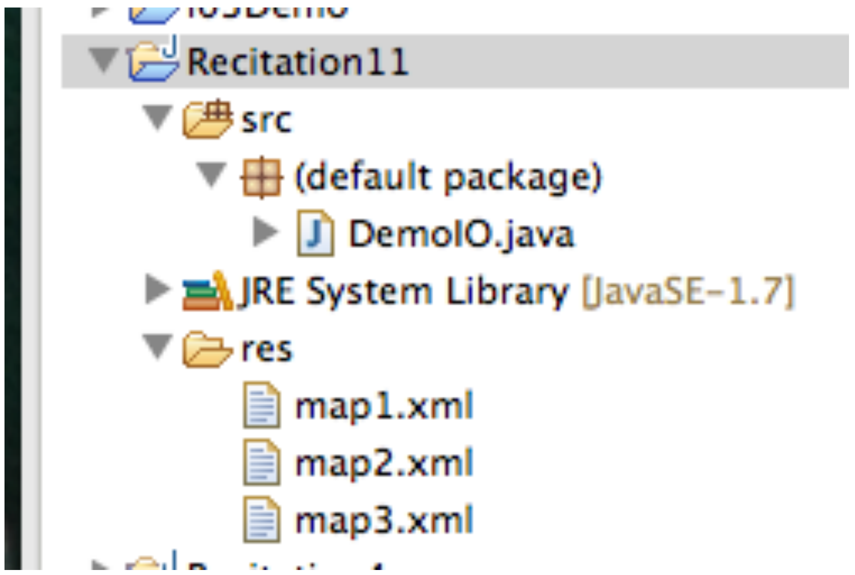
- Path is an interface because different operating systems handle files differently.
  - For each OS, there is a class that implements Path
  - To find out which class your OS uses, try p.getClas()
- A path can be absolute or relative.
  - Absolute paths give the full path of the file. To find out what absolute paths look like on your machine, try p.toAbsolutepath()
  - Relative paths define the location relative to some default location (in Java, the package directory)
  - You should always use relative paths (otherwise your code won't work on other machines)

Class Paths

An object of type Path contains the path name to a file or directory.

Class Paths contains static methods for creating Path objects

```
Path p = Paths.get("res","map1.xml");
```



Paths.get can take any number of arguments.

Arguments define a path relative to the package in which the class resides. (e.g., res/map1.xml)

Class Files

Class Files contains static methods to operate on the file/directory given by a path object. Class Files has lots of methods, e.g.

exists(Path p)	isReadable(Path p)	createFile(Path p)
delete(Path p)	isWritable(Path p)	
size(Path p)		… (lots more) …

javax.swing.JFileChoooser

Want to ask the user to navigate to select a file to read?

```
JFileChooser jd= new JFileChooser();
jd.setDialogTitle("Choose input file");
int returnVal= jd.showOpenDialog(null);
```

```
File f= jd.getSelectedFile();
```

returnVal is one of

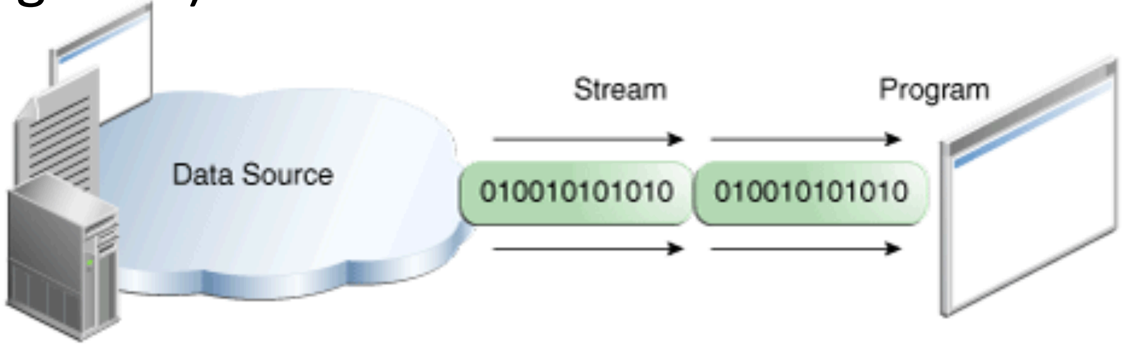
```
JFileChooser.CANCEL_OPTION
JFileChooser.APPROVE_OPTION
JFileChooser.ERROR_OPTION
```

```
jd.showOpenDialog("/Volumes/Work15A/webpage/ccgb/");
```

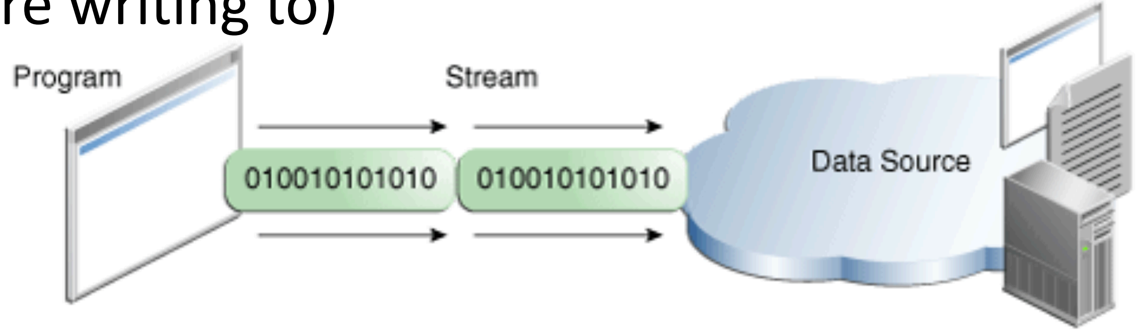
Starting always from the user's directory can be a pain for the user. User can give an argument that is the path where the navigation should start

Java I/O uses Streams

- **Stream**: a sequence of data values that is processed—either read or written—from beginning to end.
- Input streams represent an input source (e.g., a file you are reading from)



- Output streams represent an output destination (e.g., a file you are writing to)



A metaphor

- Streams are like conveyor belts in a factory or warehouse
- Input streams: take each item (e.g., a line from a file) off the conveyor belt and deal with it

- Output streams: generate each item (e.g., a line in a file) and then put it on the conveyor belt

Types of Streams

- Lots of different types of streams

Byte Streams

Raw Streams

Blocking Streams

Character Streams

Buffered Streams

NIO streams

Object Streams

Input Streams

- `InputStream` and `OutputStream` are byte I/O streams that can be used for File I/O
- Read input stream for a file is by creating an instance of class InputStream:

```
InputStream is= Files.newInputStream(p);

is.read()   // get next byte of file
```

Too low-level! Don't want to do byte by byte.
Instead, use a buffered stream to read line by line

Buffered Streams

Class BufferedReader creates a buffered stream from a raw stream (e.g., a InputStream object). You can also create a BufferedReader directly from a path. BufferedReader provides a method for reading one line at a time.

```
InputStream is= Files.newInputStream(p);
BufferedReader br= new BufferedReader(is);

                OR

BufferedReader br= Files.newBufferedReader(p);


String s= br.readLine(); // Store next line of file in s
                         // (null if none)

br.close();              // close stream when done
```

Pattern to read a file

Always use this pattern to read a file!

```
line= first line;
while (line != null) {
    Process line;
    line= next line;
}
```

```
line= br.readLine();
while (line != null) {
      Process line
      line= br.readLine();
}
```

Example: counting lines in a file

```
/** Return number of lines in file at path p.
   Throw IO Exception if problems encountered when reading
*/
public static int getSize(Path p) throws IOException {
   BufferedReader br= Files.newBufferedReader(p);
   int n= 0;  // number of lines read so far
   String line= br.readLine();

   while (line != null) {
       n= n+1;
       line= br.readLine();
   }
   br.close();
   return n;
}
```

Don't forget!

(write as while loop)

Always use this pattern to read a file!

```
line= first line;
while (line != null) {
    Process line;
    line= next line;
}
```

Output Streams

Writing a file is similar. First, get a BufferedWriter:

```
BufferedWriter bw= Files.newBufferedWriter(p);
```

Default: create file if it doesn't exist, overwrite old files

Can override defaults using options from Class **StandardOpenOption**

Then use

```
    bw.write("…");
```

to write a String to the file.

```
bw.close();    // Don't forget to close!
```

Recommended: use a PrintWriter to write non-String objects and to access additional methods (e.g., println)

```
Printwriter pw = new PrintWriter(Files.newBufferedWriter(p));
pw.println(6);
```

Standard Streams

- Standard streams are operating system features that read input from the keyboard and write output to the display
- Java supports these

```
System.out
System.in
```

You've probably already used this! It's just an output stream.

- System.out is a PrintWriter
- System.in is an InputStream

Reading Remote Files

Class URL in package java.net:

URL url= **new** URL(“http://www. … …. /links.html);

A URL (Universal Resource Locator) describes a resource on the web, like a web page, a jpg file, a gif file

The “protocol” can be:

http	(HyperText Transfer Protocol)
https	
ftp	(File Transfer Protocol)

Reading from an html web page

Given is URL url= **new** URL("http://www. … …. /links.html);

To read lines from that webpage, do this:

1. Create an InputStreamReader:

```
InputStreamReader isr=
        new InputStreamReader(url.openStream());
```

Have to open the stream

2. Create a Buffered Reader:

```
BufferedReader br=   new BufferedReader(isr);
```

3. Read lines, as before, using br.readLine()