

Java File I/O

- Follows *the* Java I/O model.
 - you can't escape from:
 - `InputStream` and `OutputStream`
 - read/write bytes
 - `Reader` and `Writer`
 - read/write characters
 - `BufferedReader` and `BufferedWriter`
 - read/write strings
 - There are specialized versions of these object types that are used with files.

Before we look at File I/O

- `File`: a class that deals with the properties of files (not the contents of files).
 - file name and directory
 - some *generic* attributes (java is OS independent!)
 - the file *length* (number of bytes in the file).
- A `File` object is used to get information about files (and directories/folders), but not to read from or write to files.

Some File Methods

`File(String pathname):` constructor

`canRead()`, `canWrite()`: checks permissions

`createNewFile()`, `delete()`

`isDirectory()`, `isFile()`

`length()`

`listFiles()`, `list()`

Sample Code: [Listfiles.java](#)

Writing to a File

- `FileOutputStream`: can write bytes.

```
FileOutputStream f = new FileOutputStream("foo");  
    f.write(22);  
    f.close();
```

- `FileWriter`: can write characters and strings

```
FileWriter f = new FileWriter("foo");  
    f.write("Hello World\n");  
    f.close();
```

Sample Code: [TextIOWrite.java](#)

Reading from a File

- `FileInputStream`: can read bytes.

```
FileInputStream f = new FileInputStream("foo");  
    int x = f.read();  
    f.close();
```

- `FileReader`: can read characters

```
FileReader f = new FileReader("foo");  
    char buf[100];  
    f.read(buf, 0, 100);
```

Reading strings from a file

- You need a `BufferedReader`
 - provides method `readLine()`

```
BufferedReader fin =  
    new BufferedReader( new FileReader("foo") );  
String s = fin.readLine();
```

Sample Code: [TextIOWrite.java](#)

Object Streams

- *Any serializable* object can be written to and read from an *object stream*
 - can be used to save objects in a file for later use
 - can also be used to exchange Java objects over a network (Interprocess Communication).
- Recall that to make your own class serializable, you need to implement the interface *Serializable*.
 - unless you don't want a deep copy, you don't need to do anything special, just state that the class implements `Serializable`.

Object Stream Classes

- `ObjectInputStream`: allows you to read objects (serialized objects) from any byte stream.

```
ObjectInputStream ois = new ObjectInputStream(  
    new FileInputStream("foo"));
```

```
Object huh = ois.readObject();
```

- `ObjectOutputStream`: allows you to write objects to any byte stream.

```
ObjectOutputStream oos = new ObjectOutputStream(  
    new FileOutputStream("foo"));
```

```
oos.writeObject(someobject);
```

Sample Code

- `SerializeArgs`: reads/writes an array of String (whatever it gets in args).
- `SerializePlay`: defines it's own serializable class and saves/restores an array of these objects.
- `SerializePoly`: defines a base class and some derived classes. Creates an array of these objects and serializes it saving to a file.



HW3!

Poly-Poly-Poly-Polymorphism