20-21. Streams and File I/O

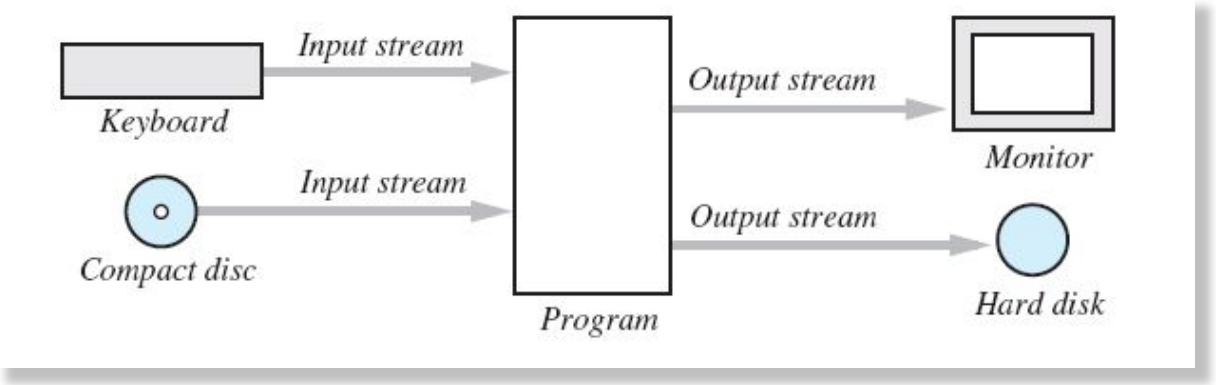
[ECE200016/ITP20003] Java Programming

Agenda

- An Overview of Streams and File I/O
- Text File I/O
- Techniques for Any File
- Binary File I/O
- File I/O With Objects and Arrays

The Concept of a Stream

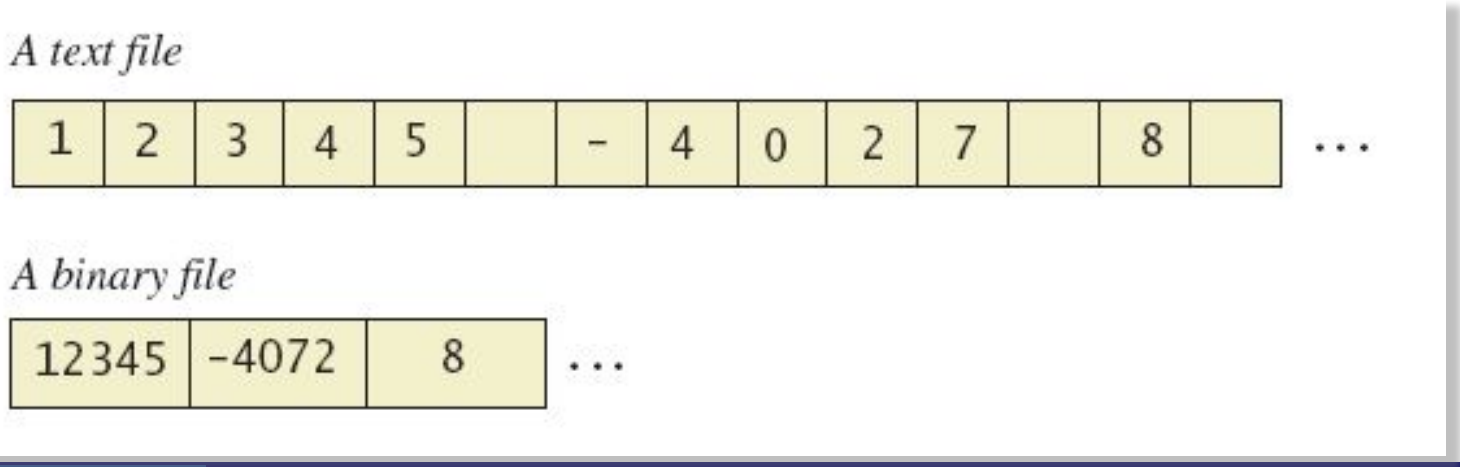
- A stream is a flow of input or output data
  - Characters, numbers, and bytes



- Streams are implemented as objects of special stream classes
  - Class Scanner
  - Object System.out

Text Files and Binary Files

- Files treated as sequence of characters called text files
  - Java program source code
  - Can be viewed, edited with text editor
- All other files are called binary files
  - Movie, music files
  - Access requires specialized program



Creating a Text File

- Class PrintWriter defines methods needed to create and write to a text file
  - Must import package java.io
- To open the file
  - Declare stream variable for referencing the stream
  - Invoke PrintWriter constructor, pass file name as argument
  - Requires try and catch blocks

https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/PrintWriterDemo.java

```
String fileName = "out.txt";//Could read file name from user
PrintWriter outputStream = null;
try
{
    outputStream = new PrintWriter(fileName);
}
catch(FileNotFoundException e)
{
    System.out.println("Error opening the file " + fileName);
    System.exit(0);
}
```

Creating a Text File

- Opening, writing to file overwrites pre-existing file in directory
- File is empty initially
  - May now be written to with method println
- Data goes initially to memory buffer
  - When buffer full, goes to file
- Closing file empties buffer, disconnects from stream
  - Flushes content of the memory buffer to file.

Creating a Text File

https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/TextFileOutputDemo.java

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class TextFileOutputDemo      // Listing 10.1
{
   public static void main (String [] args)
   {
      String fileName = "out.txt"; //The name could be read from the keyboard.
      PrintWriter outputStream = null;
      try {
         outputStream = new PrintWriter (fileName);
      } catch (FileNotFoundException e) {
         System.out.println ("Error opening the file " + fileName);
         System.exit (0);
      }
      System.out.println ("Enter three lines of text:");
      Scanner keyboard = new Scanner (System.in);
      for (int count = 1 ; count <= 3 ; count++) {
         String line = keyboard.nextLine ();
         outputStream.println (count + " " + line);
      }
      outputStream.close();
      System.out.println ("Those lines were written to " + fileName);
   }
}
```

Creating a Text File

- Result

```
Enter three lines of text:
A tall tree
in a short forest is like
a big fish in a small pond.
Those lines were written to out.txt
```

Resulting File

```
1 A tall tree
2 in a short forest is like
3 a big fish in a small pond.
```

You can use a text editor to read this file.

Appending to a Text File

https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/TextFileOutputAppendDemo.java

- Opening a file new begins with an empty file
  - If already exists, will be overwritten
- Some situations require appending data to existing file
- Command could be

  outputStream = new PrintWriter(
  new FileOutputstream(fileName, true));

  See. FileOutputStream

  See. FileOutputStream(File file, boolean append)
  - For more, see http://java.oracle.com
- Method println would append data at end

Reading from a Text File

- class TextFileInputDemo
  - Reads text from file, displays on screen
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/TextFileInputDemo.java

- Note
  - Statement which opens the file
  - Use of Scanner object
  - Boolean statement which reads the file and terminates reading loop

Reading from a Text File

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class TextFileInputDemo
{
    public static void main (String [] args)
    {
        String fileName = "out.txt";
        Scanner inputStream = null;
        System.out.println ("The file " + fileName + "\ncontains the following lines:\n");
        try {
            inputStream = new Scanner (new File (fileName));
        }  catch (FileNotFoundException e) {
            System.out.println ("Error opening the file " + fileName);
            System.exit (0);
        }
        while (inputStream.hasNextLine ()) {
            String line = inputStream.nextLine ();
            System.out.println (line);
        }
        inputStream.close ();
    }
}
```

Reading from a Text File

- Result

```
The file out.txt
contains the following lines:

1 A tall tree
2 in a short forest is like
3 a big fish in a small pond.
```

Reading from a Text File

- Additional methods in class Scanner
  - For System.in, they always returns true.

Scannner_Object_Name.hasNext() Returns true if more input data is available to be read by the method next.
Scannner_Object_Name.hasNextDouble() Returns true if more input data is available to be read by the method nextDouble.
Scannner_Object_Name.hasNextInt() Returns true if more input data is available to be read by the method nextInt.
Scannner_Object_Name.hasNextLine() Returns true if more input data is available to be read by the method nextLine.

Agenda

- An Overview of Streams and File I/O
- Text File I/O
- **Techniques for Any File**
- Binary File I/O
- File I/O With Objects and Arrays

The Class *File*

- A File object represents the name of a file

  Ex) new File ("treasure.txt");
  - Not simply a string
  - It is an object that knows it is supposed to name a file

Programming Example

https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/TextFileInputWithUserFileDemo.java

- Reading a file name from the keyboard

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
public class TextFileInputDemo2
{
   public static void main (String [] args)
   {
      System.out.print ("Enter file name: ");
      Scanner keyboard = new Scanner (System.in);
      String fileName = keyboard.next ();
      Scanner inputStream = null;
      System.out.println ("The file " + fileName + "\n" + "contains the following lines:\n");
      try {
         inputStream = new Scanner (new File (fileName));
      } catch (FileNotFoundException e) {
         System.out.println ("Error opening the file " + fileName ");
         System.exit (0);
      }

      …
   }
}
```

```
Enter file name: out.txt
The file out.txt
contains the following lines:

1 A tall tree
2 in a short forest is like
3 a big fish in a small pond.
```

Using Path Names

- Files opened in our examples assumed to be in same folder as where program run
- Possible to specify path names
  - Full path name
  - Relative path name
- Be aware of differences of pathname styles in different operating systems
  - UNIX style: /usr/bin/ls
  - Windows style: C:\Windows\system32\cmd.exe

Methods of the Class *File*

- Recall that a File object is a system-independent abstraction of file's path name
  - File(File parent, String child)
    - Creates a new File instance from a parent abstract pathname and a child pathname string.
  - File(String parent, String child)
    - Creates a new File instance from a parent pathname string and a child pathname string.

- Class File has methods to access information about a path and the files in it
  - Whether the file exists
  - Whether it is specified as readable or not
  - Etc.

Methods of the Class *File*

- Some methods in class File

`public boolean canRead()` Tests whether the program can read from the file.
`public boolean canWrite()` Tests whether the program can write to the file.
`public boolean delete()` Tries to delete the file. Returns true if it was able to delete the file.
`public boolean exists()` Tests whether an existing file has the name used as an argument to the constructor when the `File` object was created.
`public String getName()` Returns the name of the file. (Note that this name is not a path name, just a simple file name.)
`public String getPath()` Returns the path name of the file.
`public long length()` Returns the length of the file, in bytes.

Defining a Method to Open a Stream

- Method will have a String parameter
  - The file name
- Method will return the stream object
- Will throw exceptions
  - If file not found
  - If some other I/O problem arises

Defining a Method to Open a Stream

- Should be invoked inside a try block and have appropriate catch block

Ex)

```
public static PrintWriter openOutputTextFile(String fileName)
                    throws FileNotFoundException, IOException
{
    PrintWriter toFile = new PrintWriter(fileName);
    return toFile;
}
```

Ex)

```
PrintWriter outputStream = null;
try
{
    outputStream = openOutputTextFile("data.txt");
}
< appropriate catch block(s) >
```

Case Study: Processing a Comma-Separated Values File

- A comma-separated values (CSV) file is a simple text format used to store a list of records

- Example from log of a cash register's transactions for the day:

```
SKU,Quantity,Price,Description
4039,50,0.99,SODA
9100,5,9.50,T-SHIRT
1949,30,110.00,JAVA PROGRAMMING TEXTBOOK
5199,25,1.50,COOKIE
```

  - SKU(stock keeping unit): a distinct item, such as a product or service.

Example Processing a CSV File

- class TransactionReader
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/TransactionReader.java

- Uses the split method which puts strings separated by a delimiter into an array

```
String line = "4039,50,0.99,SODA"
String[] ary = line.split(",");
System.out.println(ary[0]);          // Outputs 4039
System.out.println(ary[1]);          // Outputs 50
System.out.println(ary[2]);          // Outputs 0.99
System.out.println(ary[3]);          // Outputs SODA
```

Various file reading approaches

- Lab15
- JC's FileUtil
  - https://github.com/lifove/JCTools/blob/master/src/main/java/net/lifove/research/utils/FileUtil.java

Agenda

- An Overview of Streams and File I/O
- Text File I/O
- Techniques for Any File
- **Binary File I/O**
- File I/O With Objects and Arrays

Why Binary?

- Efficient!
  - Writing '1234567890' in a text file or a binary file???
  - Memory use for primitive types (roughly)
    - int
      - 4 byte + additional 6 bytes = 10 bytes
    - char (ASCII)
      - 1 byte * 10 + additional 3 bytes = 13 bytes
    - 20 integers vs 20 strings
      - int: 4 * 20 + 6 = 86 bytes
      - char: 1 * 10 * 20 + 3 = 203 bytes
    - https://docs.google.com/presentation/d/1eJLyN9dHnEZRPdgb9R9iJDJs0DAgR-UFZEvh5x6fAaw/edit#slide=id.p12

Creating a Binary File

- Stream class ObjectOutputStream allows files which can store values of primitive types, strings, and other objects

- Creating a binary file

  Ex) ObjectOutputStream outputStream =
  new ObjectOutputStream (new FileOutputStream (fileName));

  - Constructor for ObjectOutputStream cannot take a String parameter
  - Constructor for FileOutputSream can take a String parameter

- Writing an integer value into a binary file

  Ex) outputStream.writeInt(anInteger);

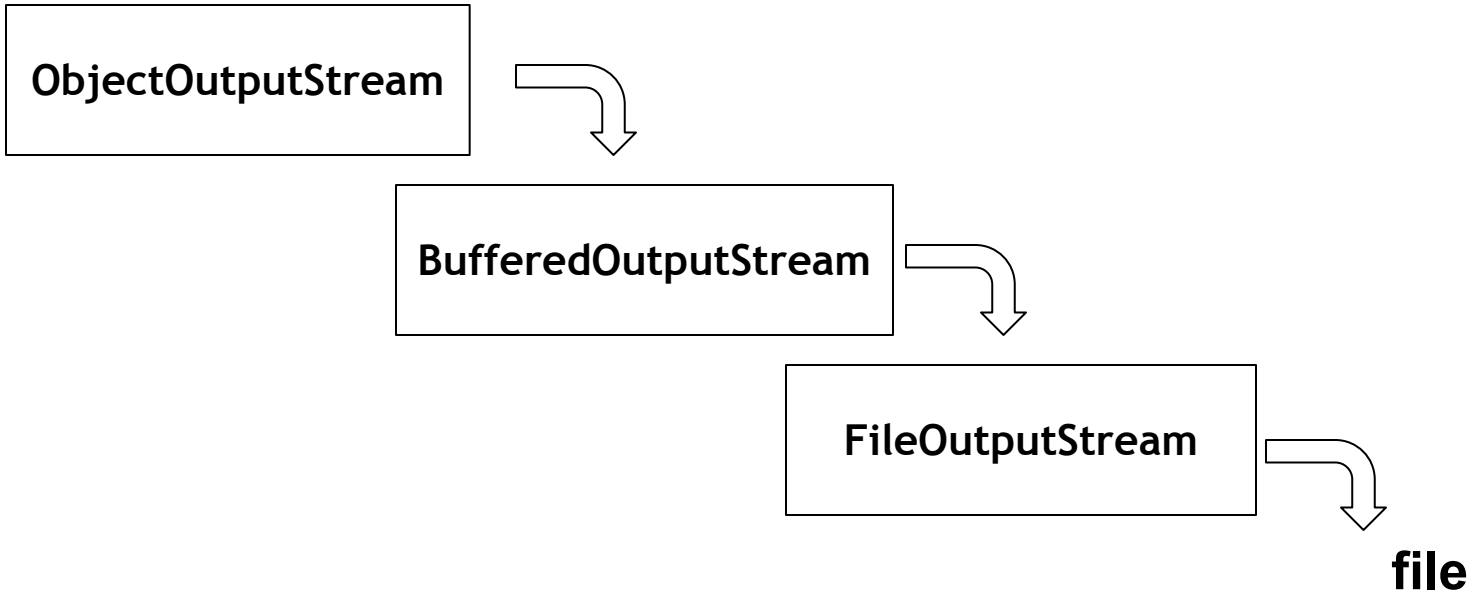
- Closing a binary file

  Ex) outputStream.close ();

Filters (== Decorators)

- Example

```
ObjectOutputStream outputStream =
                        new ObjectOutputStream(
                        new BufferedOutputStream(
outputStream            new FileOutputStream(fileName)));
```



Creating a Binary File

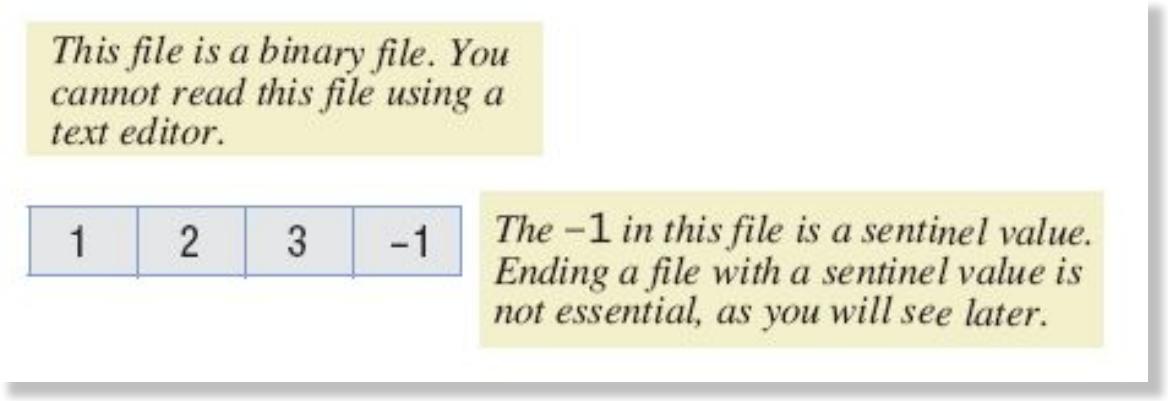
- class BinaryOutputDemo
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/binarydemo/BinaryOutputDemo.java

- Result

```
Enter nonnegative integers.
Place a negative number at the end.
1 2 3 -1
Numbers and sentinel value
written to the file numbers.dat.
```

Writing Primitive Values to a Binary File

- Method println() is not available
  - Instead, use writeInt() method
- Binary file stores numbers in binary form
  - A sequence of bytes one immediately after another



Writing Primitive Values to a Binary File

- Some methods in class ObjectOutputStream

`public ObjectOutputStream(OutputStream streamObject)`

Creates an output stream that is connected to the specified binary file. There is no constructor that takes a file name as an argument. If you want to create a stream by using a file name, you write either

```
new ObjectOutputStream(new FileOutputStream(File_Name))
```

or, using an object of the class `File`,

```
new ObjectOutputStream(new FileOutputStream(
                       new File(File_Name)))
```

Either statement creates a blank file. If there already is a file named *File_Name*, the old contents of the file are lost.

The constructor for `FileOutputStream` can throw a `FileNotFoundException`. If it does not, the constructor for `ObjectOutputStream` can throw an `IOException`.

`public void writeInt(int n) throws IOException`

Writes the `int` value `n` to the output stream.

`public void writeLong(long n) throws IOException`

Writes the `long` value `n` to the output stream.

Writing Primitive Values to a Binary File

- Some methods in class ObjectOutputStream

`public void writeDouble(double x) throws IOException`
Writes the `double` value `x` to the output stream.

`public void writeFloat(float x) throws IOException`
Writes the `float` value `x` to the output stream.

`public void writeChar(int c) throws IOException`
Writes a `char` value to the output stream. Note that the parameter type of `c` is `int`. However, Java will automatically convert a `char` value to an `int` value for you. So the following is an acceptable invocation of `writeChar`:

```
outputStream.writeChar('A');
```

`public void writeBoolean(boolean b) throws IOException`
Writes the `boolean` value `b` to the output stream.

`public void writeUTF(String aString) throws IOException`
Writes the string `aString` to the output stream. UTF refers to a particular method of encoding the string. To read the string back from the file, you should use the method `readUTF` of the class `ObjectInputStream`. These topics are discussed in the next section.

Writing Primitive Values to a Binary File

- Some methods in class ObjectOutputStream

```
public void writeObject(Object anObject) throws IOException,
                NotSerializableException, InvalidClassException
```

Writes `anObject` to the output stream. The argument should be an object of a serializable class, a concept discussed later in this chapter. Throws a `NotSerializableException` if the class of `anObject` is not serializable. Throws an `InvalidClassException` if there is something wrong with the serialization. The method `writeObject` is covered later in this chapter.

```
public void close() throws IOException
```

Closes the stream s connection to a file.

Writing Strings to a Binary File

- Use method writeUTF

  Ex) outputStream.writeUTF("Hi Mom");

  - UTF stands for Unicode Text Format
  - If you want to write ASCII string, it is recommended to use the *PrintWriter* class
- Uses a varying number of bytes to store different strings
  - Depends on length of string
  - Contrast to writeInt which uses same for each

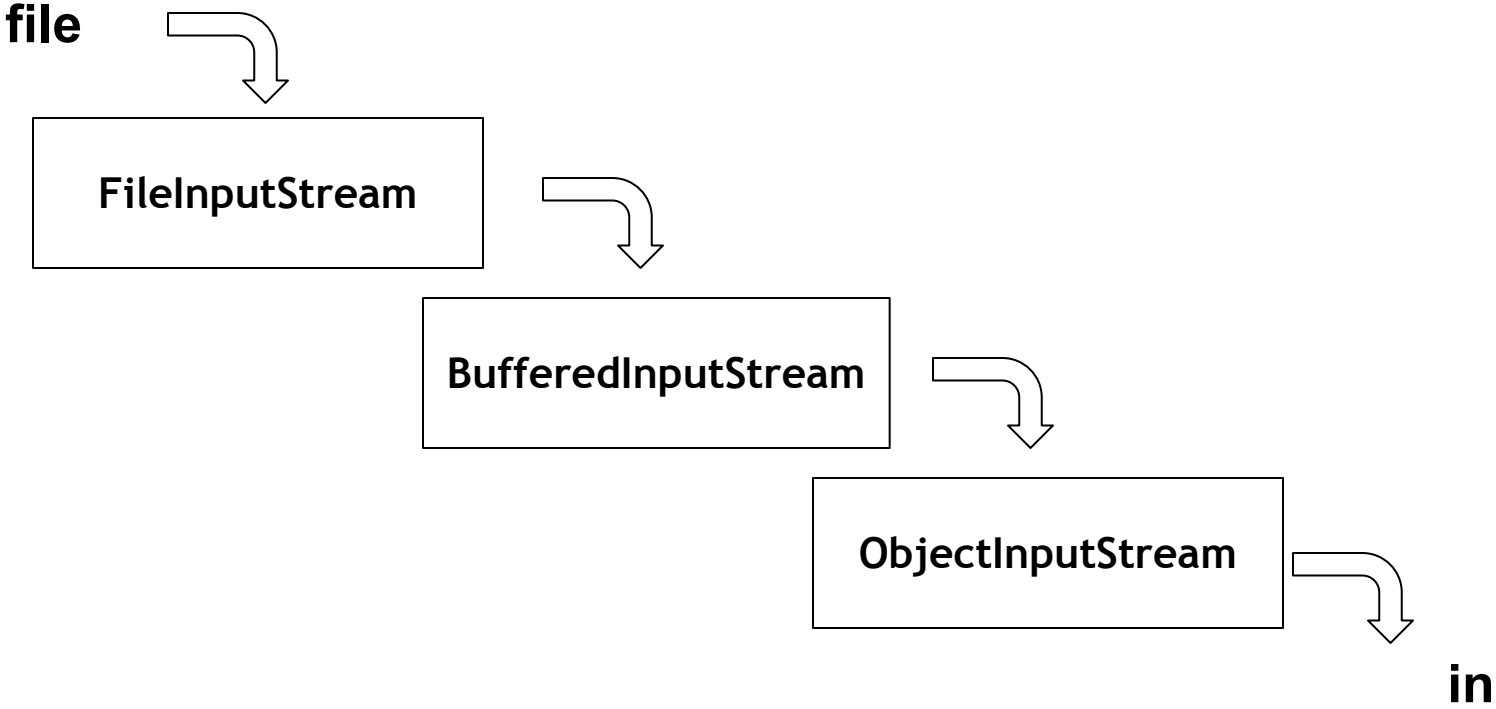
Reading from a Binary File

- File must be opened as an ObjectInputStream
- Read from binary file using methods which correspond to write methods
  - Integer written with writeInt will be read with readInt
- Be careful to read same type as was written

Opening Files

- Example

```
ObjectInputStream in = new ObjectInputStream(
                       new BufferedInputStream(
                       new FileInputStream(file_name)));
```



Reading from a Binary File

- Some methods of class ObjectInputStream

`ObjectInputStream(InputStream streamObject)`

Creates an input stream that is connected to the specified binary file. There is no constructor that takes a file name as an argument. If you want to create a stream by using a file name, you use either

```
new ObjectInputStream(new FileInputStream(File_Name))
```

or, using an object of the class `File`,

```
new ObjectInputStream(new FileInputStream(
                      new File(File_Name)))
```

The constructor for `FileInputStream` can throw a `FileNotFoundException`. If it does not, the constructor for `ObjectInputStream` can throw an `IOException`.

`public int readInt() throws EOFException, IOException`

Reads an `int` value from the input stream and returns that `int` value. If `readInt` tries to read a value from the file that was not written by the method `writeInt` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

Reading from a Binary File

- Some methods of class ObjectInputStream

`public long readLong() throws EOFException, IOException`

Reads a `long` value from the input stream and returns that `long` value. If `readLong` tries to read a value from the file that was not written by the method `writeLong` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

Note that you cannot write an integer using `writeLong` and later read the same integer using `readInt`, or to write an integer using `writeInt` and later read it using `readLong`. Doing so will cause unpredictable results.

`public double readDouble() throws EOFException, IOException`

Reads a `double` value from the input stream and returns that `double` value. If `readDouble` tries to read a value from the file that was not written by the method `writeDouble` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

Reading from a Binary File

- Some methods of class ObjectInputStream

`public float readFloat() throws EOFException, IOException`

Reads a `float` value from the input stream and returns that `float` value. If `readFloat` tries to read a value from the file that was not written by the method `writeFloat` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

Note that you cannot write a floating-point number using `writeDouble` and later read the same number using `readFloat`, or write a floating-point number using `writeFloat` and later read it using `readDouble`. Doing so will cause unpredictable results, as will other type mismatches, such as writing with `writeInt` and then reading with `readFloat` or `readDouble`.

Reading from a Binary File

- Some methods of class ObjectInputStream

`public char readChar() throws EOFException, IOException`
Reads a `char` value from the input stream and returns that `char` value. If `readChar` tries to read a value from the file that was not written by the method `writeChar` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

`public boolean readBoolean() throws EOFException, IOException`
Reads a `boolean` value from the input stream and returns that `boolean` value. If `readBoolean` tries to read a value from the file that was not written by the method `writeBoolean` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. If the read goes beyond the end of the file, an `EOFException` is thrown.

Reading from a Binary File

- Some methods of class ObjectInputStream

```
public String readUTF() throws IOException,
                               UTFDataFormatException
```

Reads a `String` value from the input stream and returns that `String` value. If `readUTF` tries to read a value from the file that was not written by the method `writeUTF` of the class `ObjectOutputStream` (or was not written in some equivalent way), problems will occur. One of the exceptions `UTFDataFormatException` or `IOException` can be thrown.

```
Object readObject() throws ClassNotFoundException,
    InvalidClassException, OptionalDataException, IOException
```

Reads an object from the input stream. Throws a `ClassNotFoundException` if the class of a serialized object cannot be found. Throws an `InvalidClassException` if something is wrong with the serializable class. Throws an `OptionalDataException` if a primitive data item, instead of an object, was found in the stream. Throws an `IOException` if there is some other I/O problem. The method `readObject` is covered in Section 10.5.

```
public void close() throws IOException
```

Closes the stream's connection to a file.

Reading from a Binary File

- class BinaryInputDemo
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/binarydemo/BinaryInputDemo.java

```
Reading the nonnegative integers
in the file numbers.dat.
1
2
3
End of reading from file.
```

The Class *EOFException*

- Many methods that read from a binary file will throw an EOFException
  - Can be used to test for end of file
  - Thus it can end a reading loop
- class BinaryInputEOFDemo
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/binarydemo/BinaryInputEOFDemo.java

```
try {
    while (true) {
        int anInteger = inputStream.readInt ();
        System.out.println (anInteger);
    }
} catch (EOFException e) {
    System.out.println ("Reached end of the file.");
}
```

The Class *EOFException*

- Note the -1 formerly needed as a sentinel value is now also read

```
Reading ALL the integers
in the file numbers.dat.
1
2
3
-1
End of reading from file.
```

- Always a good idea to check for end of file even if you have a sentinel value

Programming Example

- Processing a file of binary data
  - Asks user for 2 file names
  - Reads numbers in input file
  - Doubles them
  - Writes them to output file

- class Doubler
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/binarydemo/Doubler.java

Agenda

- An Overview of Streams and File I/O
- Text File I/O
- Techniques for Any File
- Binary File I/O
- **File I/O With Objects and Arrays**

Binary-File I/O with Class Objects

- Consider the need to write/read objects other than Strings
  - Possible to write the individual instance variable values
  - Then reconstruct the object when file is read

- A better way is provided by Java
  - Object serialization – represent an object as a sequence of bytes to be written/read
  - Possible for any class implementing Serializable

Binary-File I/O with Class Objects

- Interface Serializable is an empty interface
  - No need to implement additional methods
  - Tells Java to make the class serializable (class objects convertible to sequence of bytes)

- class Species implements Serializable
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/serializable/Species.java

Binary-File I/O with Class Objects

- Once we have a class that is specified as Serializable we can
  - Write objects to a binary file with method writeObject

    Ex) outputStream.writeObject (califCondor);

  - Read objects with method readObject

    Ex) readOne = (Species) inputStream.readObject ();

    - Also required to use typecast of the object

- class ObjectIODemo
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/serializable/ClassObjectIODemo.java

Binary-File I/O with Class Objects

- Result

```
Records sent to file species.record.
Now let's reopen the file and echo the records.
The following were read
from the file species.record:
Name = Calif. Condor
Population = 27
Growth rate = 0.02%

Name = Black Rhino
Population = 100
Growth rate = 1.0%
End of program.
```

Some Details of Serialization

- Requirements for a class to be serializable
  - Implements interface Serializable
  - Any instance variables of a class type are also objects of a serializable class
    - Exception: transient variables
  - Class's direct superclass (if any) is either serializable or defines a default constructor

Some Details of Serialization

- Effects of making a class serializable
  - Affects how Java performs I/O with class objects
  - Java assigns a serial number to each object of the class that it writes to the ObjectOutputStream
  - If same object written to stream multiple times, only the serial number written after first time
    - Exception: *reset*() method

Array Objects in Binary Files

- Since an array is an object, possible to use writeObject with entire array
- Similarly use readObject to read entire array
- readObject()/writeObject() cannot read/write an array of class objects that are not Serializable.
- class ArrayIODemo
  - https://github.com/lifove/FileIO/blob/master/src/main/java/edu/handong/csee/java/example/serializable/ArrayIODemo.java

Array Objects in Binary Files

- Result

```
Array written to file array.dat and file is closed.
Open the file for input and echo the array.
The following were read from the file array.dat:
Name = Calif. Condor
Population = 27
Growth rate = 0.02%

Name = Black Rhino
Population = 100
Growth rate = 1.0%

End of program.
```