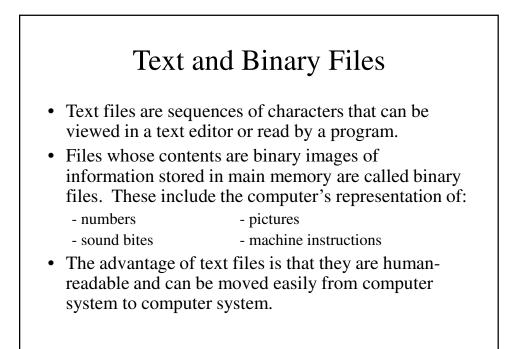
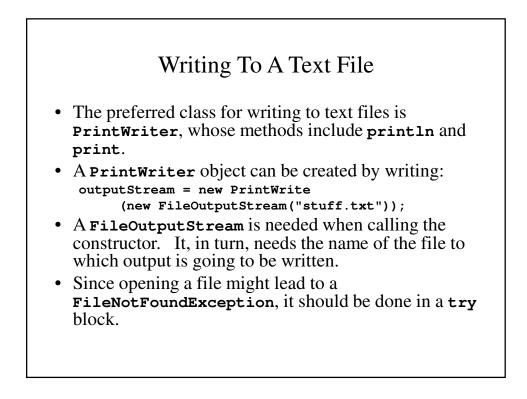
# Intermediate Programming

Lecture #11 – File I/O

## Streams

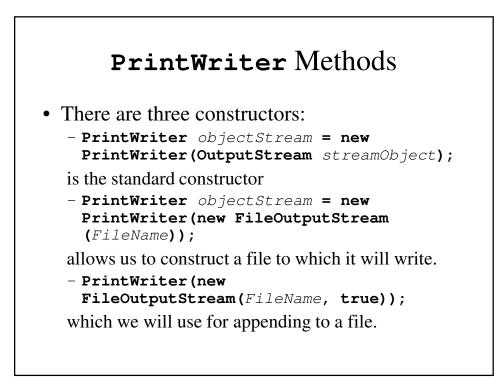
- A stream is an object that allows for the flow of data between a program and some I/O device (or a file).
  - If the flow is into a program it's an input stream.
  - If the flow is out of a program, it's an output stream.
- System. in and System. out are examples of input and output streams respectively.

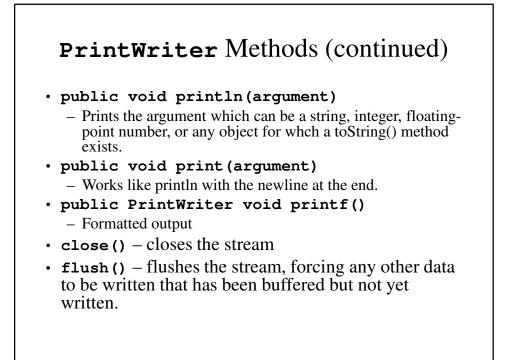




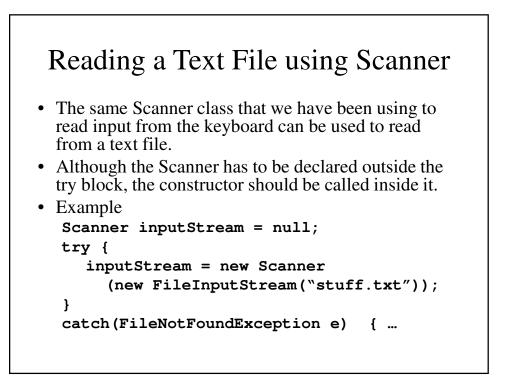
# Every File Has 2 Names

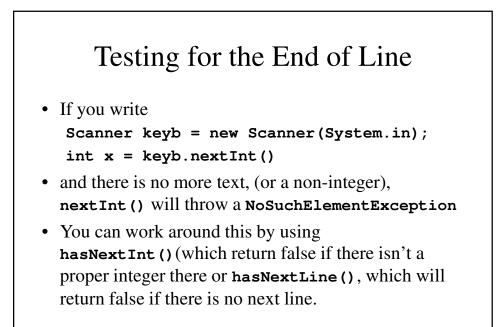
• Every file has 2 names: the name by which it is known the operating system of its computer and the name of the stream connect to the file (by which the program knows it).

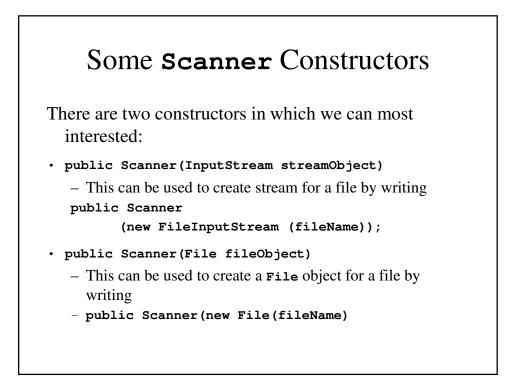


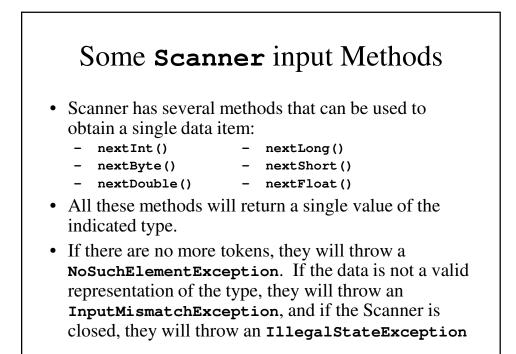


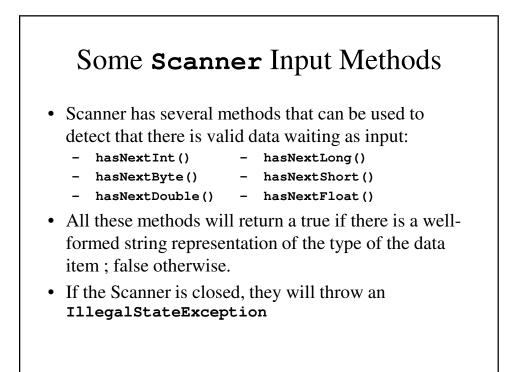
TextFileOutputDemo.java
<pre>import java.io.PrintWriter;</pre>
<pre>import java.io.FileOutputStream;</pre>
<pre>import java.io.FileNotFoundException;</pre>
<pre>public class TextFileOutputDemo {    public static void main(String[] args) {       PrintWriter outputStream = null;</pre>
try {
outputStream = new PrintWriter
<pre>(new FileOutputStream("stuff.txt"));</pre>
}
<pre>catch (FileNotFoundException e) {    System.out.println</pre>
<pre>("\"Error opening the file stuff.txt\"."); System.exit(0);</pre>
}











## Some Other Scanner Methods

- Scanner has two methods that can be used to detect that there is valid data waiting as input:
  - next() hasNext()
  - nextLine() nextLine()
- **next()** and **nextLine()** return the next token and line respectively.
- **hasNext()** and **hasNextLine()** return true if there is another token or line respectively; otherwise false.
- These methods throw the appropriate exceptions.

```
TextFileScannerDemo.java
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
public class TextFileScannerDemo {
   public static void main(String[] args) {
     System.out.println
        ("I will read three nunbers and a line");
     System.out.println
        ("of text from the file morestuff.txt");
     Scanner inputStream = null;
     try {
        inputStream = new Scanner
               (new FileInputStream("morestuff.txt"));
     }
```

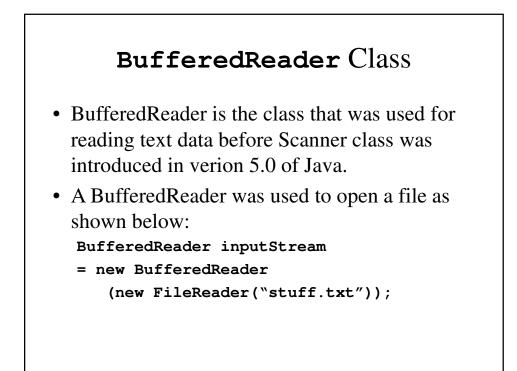
```
System.out.println
                ("The line read from the file is:");
System.out.println(line);
inputStream.close();
}
```

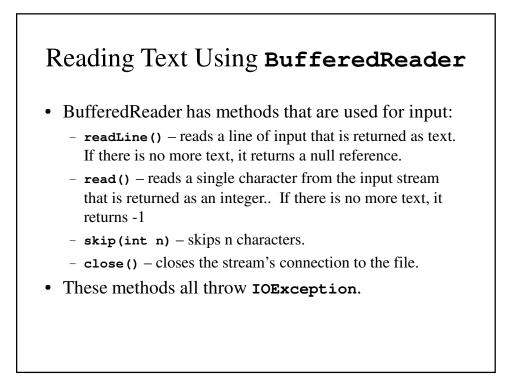
## HasNextLineDemo.java

```
catch (FileNotFoundException e) {
   System.out.println("Problem opening files.");
   System.exit(0);
   String line = null;
   int count = 0;
   while (inputStream.hasNextLine()) {
      line= inputStream.nextLine();
      count++;
      outputStream.println(count + " " + line);
      }
   inputStream.close();
   outputStream.close();
   }
}
```

#### HasNextIntDemo.java

```
catch (FileNotFoundException e) {
      System.out.println
               ("File data.txt was not found");
      System.out.println("or could not be opened.");
      System.exit(0);
    }
    int next, sum = 0;
    while (inputStream.hasNextInt()) {
     next= inputStream.nextInt();
      sum += next;
    }
    inputStream.close();
    System.out.println("The sum of the numbers is "
                      + sum);
 }
}
```





## TextFileInputDemo.java

```
line= inputStream.readLine();
      System.out.println
         ("The second line read from the file is:");
      System.out.println(line);
      inputStream.close();
    }
    catch(FileNotFoundException e) {
      System.out.println
            ("File morestuff2.txt was not found");
      System.out.println("or could not be opened.");
    }
    catch (IOException e) {
      System.out.println
             ("Error reading from morestuff2.txt");
    }
  }
}
```

## TextEOFDemo.java

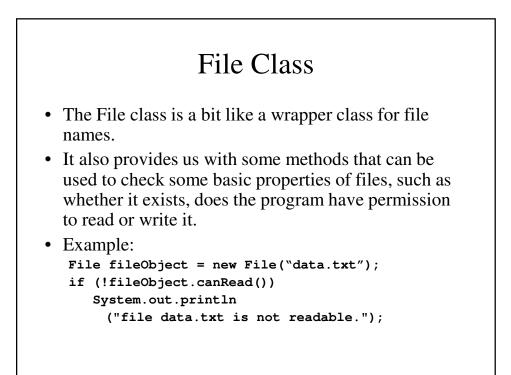
```
int count= 0;
      String line= inputStream.readLine();
      while (line != null) {
        count++;
        outputStream.println(count + " " + line);
        line = inputStream.readLine();
      }
      inputStream.close();
      outputStream.close();
    }
    catch (FileNotFoundException e) {
      System.out.println("Problem opening file.");
    }
    catch (IOException e) {
      System.out.println
          ("Error reading from original.txt");
    }
  }
}
```

## **Standard Streams**

• All Java programs are assumed to have at least three open streams: System.in, System.out and System.err (the last one allows for error message and output to be redirects separately).

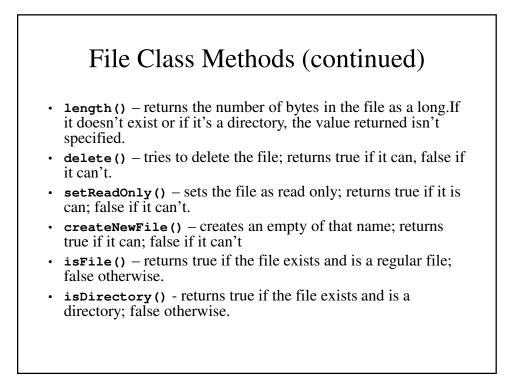
#### RedirectionDemo.java import java.io.PrintStream; import java.io.FileOutputStream; import java.io.FileNotFoundException; public class RedirectionDemo { public static void main(String[] args) { PrintStream errStream = null; try { errStream = new PrintStream(new FileOutputStream("errormessage.txt")); } catch (FileNotFoundException e) { System.out.println ("Error opening file with FileOutputSteam."); System.exit(0); }

```
System.setErr(errStream);
System.err.println("Hello fgom System.err.");
System.err.println("Hello from System.out.");
System.err.println("Hello again from
System.err.");
errStream.close();
}
```



# File Class Methods

- exists() returns true if it exists; false otherwise
- **canRead()** returns true if the program can read data from the file; false otherwise.
- **canWrite()** returns true if the program can write data from the file; false otherwise.
- **isFile()** returns true if the file exists and is a regular file; false otherwise.
- **isDirectory()** returns true if the file exists and is a directory; false otherwise.



## File Class Methods (continued)

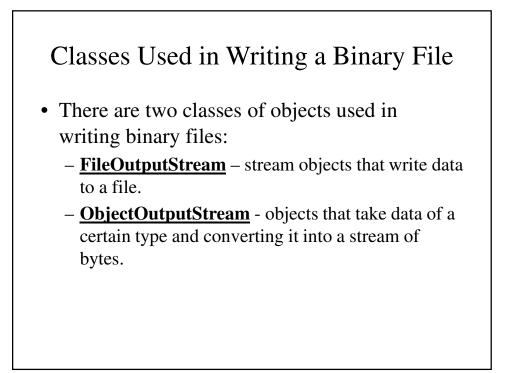
- getName() return the simple name (the last part of the path).
- getPath() returns the abstract (absolute or relative) path name or an empty string if the path name is an empty string.
- renameTo (newName) renames the file to a name represented by the file object newName; returns true if it is can; false if it can't.
- **createNewFile()** creates an empty of that name; returns true if it can; false if it can't
- mkdir() makes the directory named by the abstract path name. Will not create parent directories; returns true if the file exists and is a regular file; false otherwise.
- mkdirs() makes the directory named by the abstract path name. Will necessary but nonexistent create parent directories; returns true if the file exists and is a regular file; false otherwise.

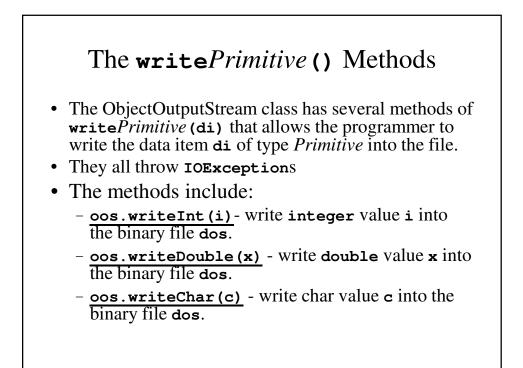
FileClassDemo.java		
import	java.util.Scanner;	
import	java.io.File;	
import	java.io.PrintWriter;	
import	java.io.FileOutputStream;	
import	java.io.FileNotFoundException;	
publ: Sca	<pre>class FileClassDemo {     c static void main(String[] args) {     anner keyb= new Scanner(System.in);     ring line = null;</pre>	
	ring fileName = null;	
Sys	stem.out.println	
	("I will store a line of text for you.");	
Sys	stem.out.println("Enter the line of text:");	
1	<pre>ne = keyb.nextLine();</pre>	

```
try {
      outputStream
           = new PrintWriter(new
                      FileOutputStream(fileName));
    }
    catch (FileNotFoundException e) {
      System.out.println("Error opening the file.");
      System.exit(0);
    }
    System.out.println("writing \"" + line + "\"");
    System.out.println("to the file " + fileName);
    outputStream.println(line);
   outputStream.close();
   System.out.println("Writing completed.");
 }
}
```

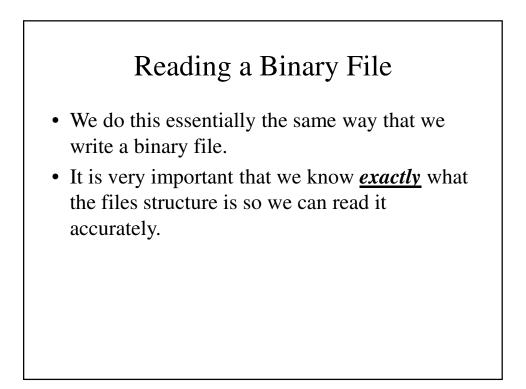
# **Binary Files**

- Text files save data in the same form as they appear on the screen or printed page. This requires the computer to do some conversion when reading and writing them.
- Binary files save data in their internal form.
  - Strings have their unused bytes saved
  - Numbers are stored in binary form.





BinaryOutputDemo.java
<pre>import java.io.ObjectOutputStream; import java.io.FileOutputStream; import java.io.IOException;</pre>
<pre>public class BinaryOutputDemo {    public static void main(String[] args) {      try {         ObjectOutputStream outputStream</pre>
<pre>int i; for (i = 0; i &lt; 5; i++)    outputStream.writeInt(i);</pre>

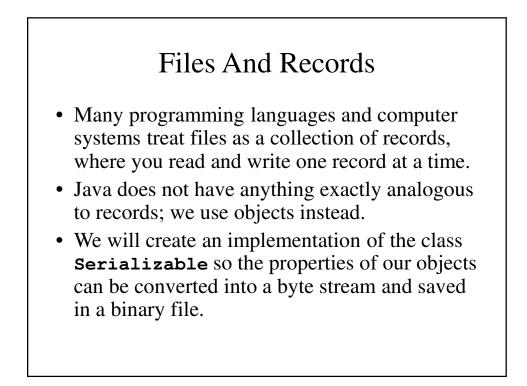


# The readPrimitive () Methods The ObjectInputStream class has several methods of readPrimitive (di) that allows the programmer to read the data item di of type Primitive from the file. The methods include: ois.readInt(i) read integer value i from the

- binary file dos.
  <u>ois.readDouble(x)</u> read double value x from the binary file dos.
- ois.readChar(c) read char value c from the binary file dos.

BinaryInputDemo.java
<pre>import java.io.ObjectInputStream;</pre>
<pre>import java.io.FileInputStream;</pre>
<pre>import java.io.IOException;</pre>
<pre>import java.io.FileNotFoundException;</pre>
<pre>public class BinaryInputDemo {</pre>
<pre>public static void main(String[] args) {     try {</pre>
ObjectInputStream inputStream = new ObjectInputStream(new
<pre>FileInputStream("numbers.dat"));</pre>
System.out.println
("Reading the file \"numbers.dat\"");
<pre>int n1 = inputStream.readInt();</pre>
<pre>int n2 = inputStream.readInt();</pre>

```
System.out.println("Numbers read from file:");
      System.out.println(n1);
      System.out.println(n2);
      inputStream.close();
    }
    catch(FileNotFoundException e) {
      System.out.println
              ("Cannot find file \"numbers.dat\"");
    }
    catch(IOException e)
                         {
      System.out.println
       ("Problem with input from \"numbers.dat\".");
    }
  }
}
```



```
someClass.java
import java.io.Serializable;
public class SomeClass implements Serializable {
    private int number;
    private char letter;

    public SomeClass() {
        number = 0;
        letter = 'A';
    }
```

```
public SomeClass(int theNumber, char theLetter) {
   number = theNumber;
   letter = theLetter;
  }
  public String toString() {
   return "Number = " + number +
      " Letter " + letter;
  }
}
```

## ObjectIODemo.java

```
try {
 ObjectInputStream inputStream
      = new ObjectInputStream(new
             FileInputStream("datafile"));
  SomeClass readOne
        = (SomeClass) inputStream.readObject();
  SomeClass readTwo
        = (SomeClass) inputStream.readObject();
  System.out.println
     ("The following were read from the file:");
 System.out.println(readOne);
 System.out.println(readTwo);
}
catch (FileNotFoundException e) {
System.out.println("Cannot find datafile.");
}
```

```
catch(ClassNotFoundException e) {
   System.out.println
        ("Problems with file input.");
   }
   catch(IOException e) {
    System.out.println
        ("Problems with file input.");
   }
   System.out.println("End of program.");
  }
}
```

## ArrayIODemo.java

```
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
public class ArrayIODemo {
   public static void main(String[] args) {
      SomeClass[] a = new SomeClass[2];
      a[0] = new SomeClass(1, 'A');
      a[1] = new SomeClass(2, 'B');
```

```
SomeClass[] b = null;
try {
 ObjectInputStream inputStream
      = new ObjectInputStream(
             new FileInputStream("arrayfile"));
 b = (SomeClass[])inputStream.readObject();
  inputStream.close();
}
catch (FileNotFoundException e) {
 System.out.println
             ("Cannot find file arrayfile.");
 System.exit(0);
}
catch (ClassNotFoundException e) {
  System.out.println
           ("Problems with file input.");
 System.exit(0);
}
```

## Random Access Files

- We usually read files from the beginning to the end. Files that are always accessed in this fashion as called sequential access files.
- Sometimes we need to read files from any particular point to which we may need access. These are called random access files.
- These have two methods that other files do not have: getFilePointer() (which takes us to the current location from which we are reading) and seek() (which moves us to another location in the file).

	RandomAccessDemo.java
import	java.io.RandomAccessFile;
import	java.io.IOException;
import	java.io.FileNotFoundException;
public	class RandomAccessDemo {
publi	ic static void main(String[] args) {
try	7 {
I	RandomAccessFile ioStream
	<pre>= new RandomAccessFile("bytedata", "rw");</pre>
5	System.out.println
	("Writing 3 bytes to the file bytedata.");
i	ioStream.writeByte(1);
i	ioStream.writeByte(2);
	LoStream.writeByte(3);
5	System.out.println
	("the length of the file is now = "
	+ ioStream.length());

```
System.out.println
    ("The file pointer location is "
                 + ioStream.getFilePointer());
System.out.println
   ("Moving the file pointer to location 1.");
ioStream.seek(1);
byte oneByte = ioStream.readByte();
System.out.println
   ("The value at location 1is " + oneByte);
oneByte = ioStream.readByte();
System.out.println
   ("The value at the next location is "
                     + oneByte);
System.out.println
("Now we can move the file pointer back to ");
System.out.println
         ("location 1, and change the byte.");
```

```
ioStream.seek(1);
ioStream.writeByte(9);
ioStream.seek(1);
oneByte = ioStream.readByte();
System.out.println
      ("The value at location 1 is now "
                                 + oneByte);
System.out.println
     ("Now we can go to the end of the file");
System.out.println
                   ("and write a double.");
ioStream.seek(ioStream.length());
ioStream.writeDouble(41.99);
System.out.println
    ("the length of the fiule is now = "
                     + ioStream.length());
System.out.println
           ("Returning to location 3, ");
```

```
System.out.println
                    ("where we wrote the double.");
      ioStream.seek(3);
      double oneDouble = ioStream.readDouble();
      System.out.println
           ("The double version of location is "
                           + oneDouble);
      ioStream.close();
    }
    catch(FileNotFoundException e) {
      System.out.println("Problem opening file.");
    }
    catch (IOException e) {
        System.out.println
                         ("Problems with file I/O");
    }
   System.out.println("End of program.");
  }
}
```