

## File I/O - Chapter 10

- A Java stream is a sequence of bytes.
- An `InputStream` can read from
  - a file
  - the console (`System.in`)
  - a network socket
  - an array of bytes in memory
  - a `StringBuffer`
  - a pipe, which is an `OutputStream`
- An `OutputStream` can write to
  - a file, etc.

1

---

---

---

---

---

---

---

---

## Many Stream Classes

- Java has many different stream classes.
- The classes differ in the methods they provide for converting "regular" Java values to/from sequences of bytes.
- The classes also differ in how they are constructed. For example, can you specify the name of a file as a `String` when constructing a particular stream class.

2

---

---

---

---

---

---

---

---

## Text Files vs Binary Files

- text files contain "text", usually in ASCII
- everything else is a binary file
- 1234 vs "1234"

byte 0	byte 1	byte 2	byte 3
00000000	00000000	00000100	11010010
0	0	4	210
49	50	51	52

---

---

---

---

---

---

---

---

## Writing Text Files

```
import java.io.*;

class HelloFile {
    public static void main(String[] args)
        throws java.io.IOException
    {
        PrintWriter out =
            new PrintWriter(new FileWriter("hello.txt"));
        out.println("Hello, file system!");
        out.close();
    }
}
```

String, Object, or  
primitive type → **PrintWriter** → char  
stream → **FileWriter** → byte  
stream → **File**

4

---

---

---

---

---

---

---

---

## Reading Text Files

```
import tio.*;

class FileReadInput {
    public static void main(String[] args)
    {
        ReadInput in = new ReadInput("sample.txt");
        int x = in.readInt();
        double y = in.readDouble();
        String s = in.readWord();
        System.out.println("x = "+x);
        System.out.println("y = "+y);
        System.out.println("s = "+s);
    }
}
```

5

---

---

---

---

---

---

---

---

## EOF with hasMoreElements()

```
//WordCount.java - count the words in a file
import tio.*;

class WordCount {
    public static void main(String[] args)
    {
        ReadInput in = new ReadInput("sample.txt");
        int count = 0;
        while (in.hasMoreElements()){
            String word = in.readWord();
            count++;
        }
        System.out.println("There were "+count+" words.");
    }
}
```

6

---

---

---

---

---

---

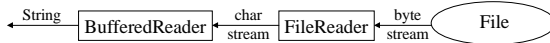
---

---

## Reading Text Files without tio

```
import java.io.*;

class HelloFileRead2 {
    public static void main(String[] args)
        throws IOException
    {
        BufferedReader in = new BufferedReader(
            new FileReader("hello.txt"));
        System.out.println(in.readLine());
    }
}
```



7

---

---

---

---

---

---

---

---

## Formatting Text Output

- C uses printf(...)
- C++ uses streams  
cout << setw(10) << i << j << k
- For tio we followed the C++ model

8

---

---

---

---

---

---

---

---

```
//FormattedOutput.java - demo of formatted text output
import tio.*;
class FormattedOutput {
    public static void main(String[] args)
        throws java.io.IOException
    {
        int x = 1, y = 22;
        double z = 4.5678E-20;
        FormattedWriter out = new FormattedWriter(System.out);
        out.setWidth(10);
        out.setDigits(2);
        out.printf("col1");
        out.printf("col2");
        out.printfln("col3");
        out.printf(x);
        out.printf(y);
        out.printfln(z);
    }
}
```

9

---

---

---

---

---

---

---

---

```
package tio;
import java.io.*;
import java.text.*;

public class FormattedWriter extends PrintWriter {
    // constants for specifying justification
    public static final int LEFT = 1;
    public static final int RIGHT = 2;

    public FormattedWriter(OutputStream os) {
        super(os, true); // make default auto-flushing
    }
    public FormattedWriter(FileWriter writer) {
        super(writer, true);
    }
    public FormattedWriter(String filename)
        throws java.io.IOException
    {
        this(new FileWriter(filename));
    }
}
```

10

---

---

---

---

---

---

---

---

```
public void setWidth(int width) {
    if (width < 0)
        this.width = 0;
    else if (width > MAX_WIDTH)
        this.width = MAX_WIDTH;
    else
        this.width = width;
}

public void setDigits(int places) {
    decimalPlaces = places;
    form.setMaximumFractionDigits(decimalPlaces);
}
```

11

---

---

---

---

---

---

---

---

```
public void setJustify(int leftOrRight) {
    if (leftOrRight != LEFT && leftOrRight != RIGHT)
        throw new IllegalArgumentException(
            "use FormattedWriter.LEFT or " +
            "FormattedWriter.RIGHT");
    justify = leftOrRight;
}

public void setPadChar(char pad) {
    if (pad == ' ')
        padding = spaces;
    else if (pad == '0')
        padding = zeros;
    else
        padding = buildPadding(MAX_WIDTH, pad);
}
```

12

---

---

---

---

---

---

---

---

```
public void printf(String s) {
    if (s.length() >= width)
        super.print(s);
    else if (justify == LEFT)
        super.print(s +
            padding.substring(0, width - s.length()));
    else
        super.print(
            padding.substring(0, width - s.length()) + s);
}

public void printf(boolean value) {
    printf(String.valueOf(value));
}

. . .
```

13

---

---

---

---

---

---

---

---

## Binary I/O

- DataInputStream/DataOutputStream
- readBoolean(), readInt(), readFloat(),..., readUTF()
- NOT the same as readInt() from tio which reads TEXT and parses it into an int.

14

---

---

---

---

---

---

---

---

```
import java.io.*;

class BinIOtest {
    public static void main(String[] args)
        throws IOException
    {
        DataOutputStream out = new DataOutputStream(
            new FileOutputStream("test.bin"));
        double x = 1.0;
        int count = 10;
        out.writeInt(count);
        for (int i = 0; i < count; i++){
            out.writeDouble(x);
            x = x / 3.0;
        }
        out.close();
    }
}
```

15

---

---

---

---

---

---

---

---

```
DataInputStream in = new DataInputStream(
    new FileInputStream("test.bin"));
count = in.readInt();
for (int i = 0; i < count; i++){
    System.out.println(in.readDouble());
}
}
```

16

---

---

---

---

---

---

---

---

```
import java.io.*;

class ObjectIO {
    public static void main(String[] args)
        throws Exception
    {
        int[] someArray = {1,2,3};
        Stack stack = new Stack();
        stack.push(new Integer(1));
        stack.push("two");
        stack.push(someArray);

        ObjectOutputStream out =
            new ObjectOutputStream(new FileOutputStream("outfile"));
        out.writeObject(stack);
        out.close();
    }
}
```

17

---

---

---

---

---

---

---

---

```
stack = null;
someArray = null;
ObjectInputStream in = new
    ObjectInputStream(new FileInputStream("outfile"));
stack = (Stack)in.readObject();
someArray = (int[])stack.pop();
System.out.println(someArray[1]);
System.out.println(stack.pop());
System.out.println(stack.pop());
}
}
```

18

---

---

---

---

---

---

---

---

```
//Stack.java - a generic stack
class Stack implements java.io.Serializable {
    Object top() {
        . . .
    }
    void push(Object value) {
        . . .
    }
    Object pop() {
        . . .
    }
    boolean empty() { return top == null; }
    private ListElement top = null;
}
```

19

---

---

---

---

---

---

---

---

```
//ListElement.java - a generic list element
class ListElement implements java.io.Serializable {
    ListElement(Object value) {
        data = value;
    }
    ListElement next;
    Object data;
}
```

20

---

---

---

---

---

---

---

---

## EOF

- input method returns a special value
- input method throws an exception
- special method used to check for more input

21

---

---

---

---

---

---

---

---

## EOF Summary

Class	Method	EOF detected by
tio.ReadInput	all	hasMoreElements()
	readLine()	returns null
	readChar()	returns -1
java.io.DataInputStream	readPrimitive()	EOFException
	readLine()	returns null
	read()	returns -1
java.io.BufferedReader	readLine()	returns null
	read()	returns -1

22

---

---

---

---

---

---

---

---

## Another Example of ObjectIO

- The following example reinforces that an `ObjectOutputStream` can handle recursive data structures.

23

---

---

---

---

---

---

---

---

```
import java.io.*;
import java.util.Vector;

class Graph implements Serializable {
    Node root;
    void print() {
        root.print();
        root.reset();
    }
}
```

24

---

---

---

---

---

---

---

---



```
class Node implements Serializable {
    int value;
    boolean visited;
    Vector children = new Vector();
    Node(int v) { value = v;}
    void print() {
        if (visited)
            return;
        System.out.print(value + ":");
        for (int i = 0; i < children.size(); i++)
            System.out.print(
                ((Node)children.elementAt(i)).value + " ");
        System.out.println();
        visited = true;
        for (int i = 0; i < children.size(); i++)
            ((Node)children.elementAt(i)).print();
    }
}
```

25

---

---

---

---

---

---

---

---

```
void reset() {
    if (!visited)
        return;
    visited = false;
    for (int i = 0; i < children.size(); i++)
        ((Node)children.elementAt(i)).reset();
}
}
```

26

---

---

---

---

---

---

---

---

```
class GraphTest {
    public static void main(String[] args)
        throws IOException, ClassNotFoundException
    {
        Graph theGraph = new Graph();
        Node node1 = new Node(1);
        Node node2 = new Node(2);
        Node node3 = new Node(3);
        Node node4 = new Node(4);

        theGraph.root = node1;
        node1.children.add(node2);
        node1.children.add(node3);
        node2.children.add(node4);
        node3.children.add(node4);
        node4.children.add(node1);
        theGraph.print();
    }
}
```

27

---

---

---

---

---

---

---

---

```
ObjectOutputStream out = new ObjectOutputStream(
    new FileOutputStream("outfile"));
out.writeObject(theGraph);
out.close();
}
}
```

28

---

---

---

---

---

---

---

---

```
class GraphRestore {
    public static void main(String[] args)
        throws IOException, ClassNotFoundException
    {
        Graph theGraph;

        ObjectInputStream in = new
            ObjectInputStream(new FileInputStream("outfile"));
        theGraph = (Graph)in.readObject();
        System.out.println("resurrected");
        theGraph.print();
    }
}
```

29

---

---

---

---

---

---

---

---

### More about Serializable

- So what can't be serialized automatically?
- There are many things. One example is an open file stream.
- In general, anything that involves state that is outside of the Java Virtual Machine, such as the file system state for an open file stream.

30

---

---

---

---

---

---

---

---

## Special Handling of Objects

- If you want to serialize an object that requires special handling, have the class implement

```
private void  
writeObject(java.io.ObjectOutputStream out)  
    throws IOException;
```

and

```
private void  
readObject(java.io.ObjectInputStream in)  
    throws IOException, ClassNotFoundException;
```

31

---

---

---

---

---

---

---

---

```
class Graph implements Serializable {  
    Node root;  
    void print() {  
        root.print();  
        root.reset();  
    }  
    private void writeObject(ObjectOutputStream out)  
        throws IOException  
    {  
        System.out.println("Writing");  
        out.defaultWriteObject();  
    }  
    private void readObject(ObjectInputStream in)  
        throws IOException, ClassNotFoundException  
    {  
        System.out.println("reading");  
        in.defaultReadObject();  
    }  
}
```

32

---

---

---

---

---

---

---

---