core WEB programming

Java Input/Output

Agenda

- Handling files and directories through the File class
- Understanding which streams to use for character-based or byte-based streams
- Character File input and output
- Formatting output
- Reading data from the console
- Binary File input and output

File Class

- A **File** object can refer to either a file or a directory

```
File file1 = new File("data.txt");
File file1 = new File("C:\java");
```

  – To obtain the path to the current working directory use

```
System.getProperty("user.dir");
```

  – **To obtain the file or path separator use**

```
System.getProperty("file.separator");
System.getProperty("path.separator");
```

or

```
File.separator()
File.pathSeparator()
```


Useful File Methods

- **isFile/isDirectory**
- **canRead/canWrite**
- **length**
  – Length of the file in bytes (`long`) or `0` if nonexistant
- **list**
  – If the `File` object is a directory, returns a `String` array of all the files and directories contained in the directory; otherwise, `null`
- **mkdir**
  – Creates a new subdirectory
- **delete**
  – Deletes the directory and returns `true` if successful
- **toURL**
  – Converts the file path to a URL object

Directory Listing, Example

```
import java.io.*;

public class DirListing {
  public static void main(String[] args) {

    File dir = new File(System.getProperty("user.dir"));

    if(dir.isDirectory()){
      System.out.println("Directory of " + dir);
      String[] listing = dir.list();
      for(int i=0; i<listing.length; i++) {
        System.out.println("\t" + listing[i]);
      }
    }
  }
}
```


DirectoryListing, Result

```
> java DirListing

Directory of C:\java\
        DirListing.class
        DirListing.java
        test
        TryCatchExample.class
        TryCatchExample.java
        XslTransformer.class
        XslTransformer.java
```

Input/Output

- **The `java.io` package provides over 60 input/output classes (streams)**
- **Streams are combined (piped together) to create a desired data source or sink**
- **Streams are either byte-oriented or character-oriented**
  - Use DataStreams for byte-oriented I/O
  - Use Readers and Writers for character-based I/O
    - Character I/O uses an encoding scheme
- **Note: An `IOException` may occur during any I/O operation**

Character File Output

Desired …	Methods	Construction
Character File Ouput	**FileWriter** write(int char) write(byte[] buffer) write(String str)	File file = new File("filename"); FileWriter fout = new FileWriter(file); *or* FileWriter fout = new FileWriter("filename");
Buffered Character File Output	**BufferedWriter** write(int char) write(char[] buffer) write(String str) newLine()	File file = new File("filename"); FileWriter fout = new FileWriter(file); BufferedWriter bout = new BufferedWriter(fout); *or* BufferedWriter bout = new BufferedWriter(new FileWriter(new File("filename")));

Character File Output, cont.

Desired …	Methods	Construction

Character Output	**PrintWriter**
	write(int char)
	write(char[] buffer)
	writer(String str)
	print(…)
	println(…)
		FileWriter fout = new FileWriter("filename");
		PrintWriter pout = new PrintWriter(fout);
		or
		PrintWriter pout = new PrintWriter(
		new FileWriter("filename"));
		or
		PrintWriter pout = new PrintWriter(
		new BufferedWriter(
		new FileWriter("filename")));

FileWriter

- **Constructors**
  - FileWriter(String filename)/FileWriter(File file)
    - Creates a output stream using the default encoding
  - FileWriter(String filename, boolean append)
    - Creates a new output stream or appends to the existing output stream (append = true)
- **Useful Methods**
  - write(String str)/write(char[] buffer)
    - Writes string or array of chars to the file
  - write(int char)
    - Writes a character (int) to the file
  - flush
    - Writes any buffered characters to the file
  - close
    - Closes the file stream after performing a flush
  - getEncoding
    - Returns the character encoding used by the file stream

CharacterFileOutput, Example

```
import java.io.*;

public class CharacterFileOutput {
  public static void main(String[] args) {
    FileWriter out = null;

    try {
      out = new FileWriter("book.txt");
      System.out.println("Encoding: " + out.getEncoding());
      out.write("Core Web Programming");
      out.close();
      out = null;
    } catch(IOException ioe) {
      System.out.println("IO problem: " + ioe);
      ioe.printStackTrace();
      try {
        if (out != null) {
          out.close();
        }
      } catch(IOException ioe2) { }
    }
  }
}
```


CharacterFileOutput, Result

```
> java CharacterFileOutput
Encoding: Cp1252

> type book.txt
Core Web Programming
```

- **Note: Cp1252 is Windows Western Europe / Latin-1**
  - To change the system default encoding use
    `System.setProperty("file.encoding", "encoding");`
  - To specify the encoding when creating the output steam, use an `OutputStreamWriter`

```
OutputStreamWriter out =
    new OutputStreamWriter(
      new FileOutputStream("book.txt", "8859_1"));
```

Formatting Output

- **Use `DecimalFormat` to control spacing and formatting**
  – Java has no `printf` method

- **Approach**
  1. Create a `DecimalFormat` object describing the formatting

```
DecimalFormat formatter =
   new DecimalFormat("#,###.##");
```

  2. Then use the `format` method to convert values into formatted strings

```
formatter.format(24.99);
```


Formatting Characters

Symbol	Meaning
0	Placeholder for a digit.
#	Placeholder for a digit. If the digit is leading or trailing zero, then don't display.
.	Location of decimal point.
,	Display comma at this location.
-	Minus sign.
E	Scientific notation. Indicates the location to separate the mattissa from the exponent.
%	Multipy the value by 100 and display as a percent.

NumFormat, Example

```
import java.text.*;

public class NumFormat {
  public static void main (String[] args) {
    DecimalFormat science = new DecimalFormat("0.000E0");
    DecimalFormat plain = new DecimalFormat("0.0000");

    for(double d=100.0; d<140.0; d*=1.10) {
      System.out.println("Scientific: " + science.format(d) +
                         " and Plain: " + plain.format(d));
    }
  }
}
```


NumFormat, Result

```
> java NumFormat

Scientific: 1.000E2 and Plain: 100.0000
Scientific: 1.100E2 and Plain: 110.0000
Scientific: 1.210E2 and Plain: 121.0000
Scientific: 1.331E2 and Plain: 133.1000
```

Character File Input

Desired …	Methods	Construction
Character File Input	**FileReader** read() read(char[] buffer)	File file = new File("filename"); FileReader fin = new FileReader(file); *or* FileReader fin = new FileReader("filename");
Buffered Character File Input	**BufferedReader** read() read(char[] buffer) readLine()	File file = new File("filename"); FileReader fin = new FileReader(file); BufferedReader bin = new BufferedReader(fin); *or* BufferedReader bin = new BufferedReader(new FileReader(new File("filename")));

FileReader

- **Constructors**
  - FileReader(String filename)/FileReader(File file)
    - Creates a input stream using the default encoding
- **Useful Methods**
  - read/read(char[] buffer)
    - Reads a single character or array of characters
    - Returns –1 if the end of the steam is reached
  - reset
    - Moves to beginning of stream (file)
  - skip
    - Advances the number of characters
- **Note: Wrap a BufferedReader around the FileReader to read full lines of text using `readLine`**

CharacterFileInput, Example

```
import java.io.*;

public class CharacterFileInput {
  public static void main(String[] args) {

    File file = new File("book.txt");
    FileReader in = null;

    if(file.exists()) {
      try {
        in = new FileReader(file);
        System.out.println("Encoding: " + in.getEncoding());
        char[] buffer = new char[(int)file.length()];
        in.read(buffer);
        System.out.println(buffer);
        in.close();
      } catch(IOException ioe) {
        System.out.println("IO problem: " + ioe);
        ioe.printStackTrace();
        ...
      }
    }
  }
}
```


CharacterFileInput, Result

```
> java CharacterFileInput

Encoding: Cp1252
Core Web Programming
```

- **Alternatively, could read file one line at a time:**

```
BufferedReader in =
  new BufferedReader(new FileReader(file));
String lineIn;
while ((lineIn = in.readLine()) != null) {
  System.out.println(lineIn);
}
```

Console Input

- **To read input from the console, a stream must be associated with the standard input, `System.in`**

```
import java.io.*;

public class IOInput{
  public static void main(String[] args) {
    BufferedReader keyboard;
    String line;
    try {
      System.out.print("Enter value: ");
      System.out.flush();
      keyboard = new BufferedReader(
                    new InputStreamReader(System.in));
      line = keyboard.readLine();
    } catch(IOException e) {
      System.out.println("Error reading input!");   }
    }
  }
}
```


Binary File Input and Output

- **Handle byte-based I/O using a DataInputStream or DataOutputStream**

DataType	DataInputStream	DataOutputStream
byte	readByte	writeByte
short	readShort	writeShort
int	readInt	writeInt
long	readLong	writeLong
float	readFloat	writeFloat
double	readDouble	writeDouble
boolean	readBoolean	writeBoolean
char	readChar	writeChar
String	readUTF	writeUTF
byte[]	readFully	

  - The `readFully` method blocks until all bytes are read or an EOF occurs
  - Values are written in big-endian fashion regardless of computer platform

UCS Transformation Format – UTF-8

- **UTF encoding represents a 2-byte Unicode character in 1-3 bytes**
  - Benefit of backward compatibility with existing ASCII data (one-byte over two-byte Unicode)
  - Disadvantage of different byte sizes for character representation

UTF Encoding	
Bit Pattern	**Representation**
0xxxxxxx	ASCII (0x0000 - 0x007F)
10xxxxxx	Second or third byte
110xxxxx	First byte in a 2-byte sequence (0x0080 - 0x07FF)
1110xxxx	First byte in a 3-byte sequence (0x0800 - 0xFFFF)

Binary File Output

Desired …	Methods	Construction
Binary File Output bytes	**FileOutputStream** write(byte) write(byte[] buffer)	File file = new File("filename"); FileOutputStream fout = new FileOutputStream(file); *or* FileOutputStream fout = new FileOutputStream("filename");
Binary File Output byte short int long float double char boolean	**DataOutputStream** writeByte(byte) writeShort(short) writeInt(int) writeLong(long) writeFloat(float) writeDouble(double) writechar(char) writeBoolean(boolean) writeUTF(string) writeBytes(string) writeChars(string)	File file = new File("filename"); FileOutputStream fout = new FileOutputStream(file); DataOutputStream dout = new DataOutputStream(fout); *or* DataOutputStream dout = new DataOutputStream(new FileOutputStream(new File("filename")));

Binary File Output, cont.

Desired …	Methods	Construction
Buffered Binary File Output	**BufferedOutputStream** flush() write(byte) write(byte[] buffer, int off, int len)	File file = new File("filename"); FileOutputStream fout = new FileOutputStream(file); BufferedOutputStream bout = new BufferedOutputStream(fout); DataOutputStream dout = new DataOutputStream(bout); *or* DataOutputStream dout = new DataOutputStream(new BufferedOutputStream(new FileOutputStream(new File("filename"))));

BinaryFileOutput, Example

```
import java.io.*;

public class BinaryFileOutput {

  public static void main(String[] args) {
    int[] primes = { 1, 2, 3, 5, 11, 17, 19, 23 };
    DataOutputStream out = null;

    try {
      out = new DataOutputStream(
              new FileOutputStream("primes.bin"));

      for(int i=0; i<primes.length; i++) {
        out.writeInt(primes[i]);
      }
      out.close();
    } catch(IOException ioe) {
       System.out.println("IO problem: " + ioe);
       ioe.printStackTrace();
    }
  }
}
```

Binary File Input

Desired …	Methods	Construction
Binary File Input bytes	**FileInputStream** read() read(byte[] buffer)	File file = new File("filename"); FileInputStream fin = new FileInputStream(file); *or* FileInputStream fin = new FileInputStream("filename");
Binary File Input byte short int long float double char boolean	**DataInputStream** readByte() readShort() readInt() readLong() readFloat() readDouble() readchar() readBoolean() readUTF() readFully(byte[] buffer)	File file = new File("filename"); FileInputStream fin = new FileInputStream(file); DataInputStream din = new DataInputStream(fin); *or* DataInputStream din = new DataInputStream(new FileInputStream(new File("filename")));

Binary File Input, cont.

Desired …	Methods	Construction
Bufferred Binary File Input	**BufferedInputStream** read() read(byte[] buffer, int off, int len) skip(long)	File file = new File("filename"); FileInputStream fin = new FileInputStream(file); BufferedInputStream bin = new BufferedInputStream(fin); DataInputStream din = new DataInputStream(bin); *or* DataInputStream din = new DataInputStream(new BufferedInputStream(new FileInputStream(new File("filename"))));

BinaryFileInput, Example

```
import java.io.*;

public class BinaryFileInput {
  public static void main(String[] args) {

    DataInputStream in = null;
    File file = new File("primes.bin");
    try {
      in = new DataInputStream(
               new FileInputStream(file));
      int prime;
      long size = file.length()/4;  // 4 bytes per int
      for(long i=0; i<size; i++) {
        prime = in.readInt();
        System.out.println(prime);
      }
      in.close();
    } catch(IOException ioe) {
       System.out.println("IO problem: " + ioe);
       ioe.printStackTrace();
    }
  }
}
```


Summary

- **A `File` can refer to either a file or a directory**
- **Use Readers and Writers for character-based I/O**
  - A BufferedReader is required for readLine
  - Java provides no printf; use DecimalFormat for formatted output
- **Use DataStreams for byte-based I/O**
  - Chain a FileOutputStream to a DataOutputStream for binary file output
  - Chain a FileInputStream to a DataInputStream for binary file input

core WEB programming

Questions?