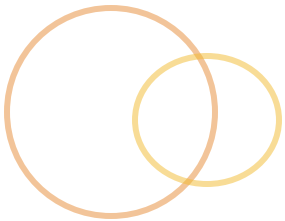
Java File IO

Objectives



At the end of this module you should be able to:

- Describe the File class
- Use data and character streams to files
- Understand the File Input and File Output Streams
- Work with file navigation
- Use buffered I/O and the PrintWriter

© DevelopIntelligence http://www.DevelopIntelligence.com

File Systems

- Describe Java was designed with the knowledge that no two file systems are the same.
- One of the big issues with file systems is how data is stored.
- File systems treat memory as a big array of data.
- When a computer wants to access specific bytes in the array, it uses an index that points to the starting position of the information.
- An address that accesses a byte is called byte-addressable in computer architecture terms.

© DevelopIntelligence http://www.DevelopIntelligence.com

10 Little Endians

- File systems tend to represent bytes in two different ways, big endian and little endian.
- A file system that stores bytes in a big endian way stores the most significant bit in the smallest address space
- In a big endian system the hex value would be stored as:

Address	Value
1000	9F
1001	A6
1002	1E
1003	CD

© DevelopIntelligence http://www.DevelopIntelligence.com

Little Endian

- In a little endian system the same number would be stored as:

Address	Value
1000	CD
1001	1E
1002	A6
1003	9F

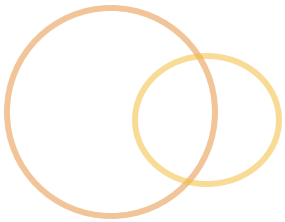
© DevelopIntelligence http://www.DevelopIntelligence.com

What's the Big Deal?

- Endianess is a problem when writing data to files, or sending data over a network.
- Suppose our program writes a float to a file from a computer that uses big endian storage, e.g., a Sun server.
- Then transfers that file to a server that uses little endian addresses like an IBM server.
- Suddenly our float, which started out as 2,678,464,205, is now 3,441,338,015.

© DevelopIntelligence http://www.DevelopIntelligence.com

Bytes and Chars



- Java file IO is broken into two categories:
  - bytes
  - characters
- Java provides classes that will read or write byte arrays, and classes that will read or write character arrays or Strings.
- Java File IO also allows the manipulation of file and directory attributes.

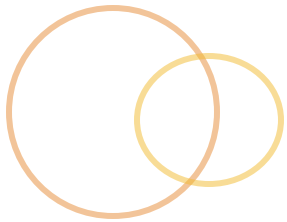
© DevelopIntelligence http://www.DevelopIntelligence.com

Byte Streams

- Byte stream classes all descend from InputStream or OutputStream which read or write 8-bit bytes.
- When reading or writing byte arrays from files, we will use the FileInputStream or the FileOutputStream classes.
- There are other Java classes that are considered byte stream classes, but we are going to concentrate only on the file streams.

© DevelopIntelligence http://www.DevelopIntelligence.com

Character Streams



- Character stream classes store values using Unicode conventions.
- Character streams automatically convert from Unicode to the local representation of character sets.
- In Western locales, the stored character set will be in 8-bit ASCII.
- All character stream classes descend from the Reader and Writer classes. We will be concentrating on the FileReader, FileWriter, and PrintWriter classes.

© DevelopIntelligence http://www.DevelopIntelligence.com

File Interactions

- Java provides classes to work with underlying file systems in a platform-independent manner.
- The `File` class allows you to create an object representation for a file.
- A `File` instance is not the file itself, but allows you to:
  - Find out information about the underlying file
  - Delete, rename, move the underlying file
  - Check permissions
  - Etc.

© DevelopIntelligence http://www.DevelopIntelligence.com

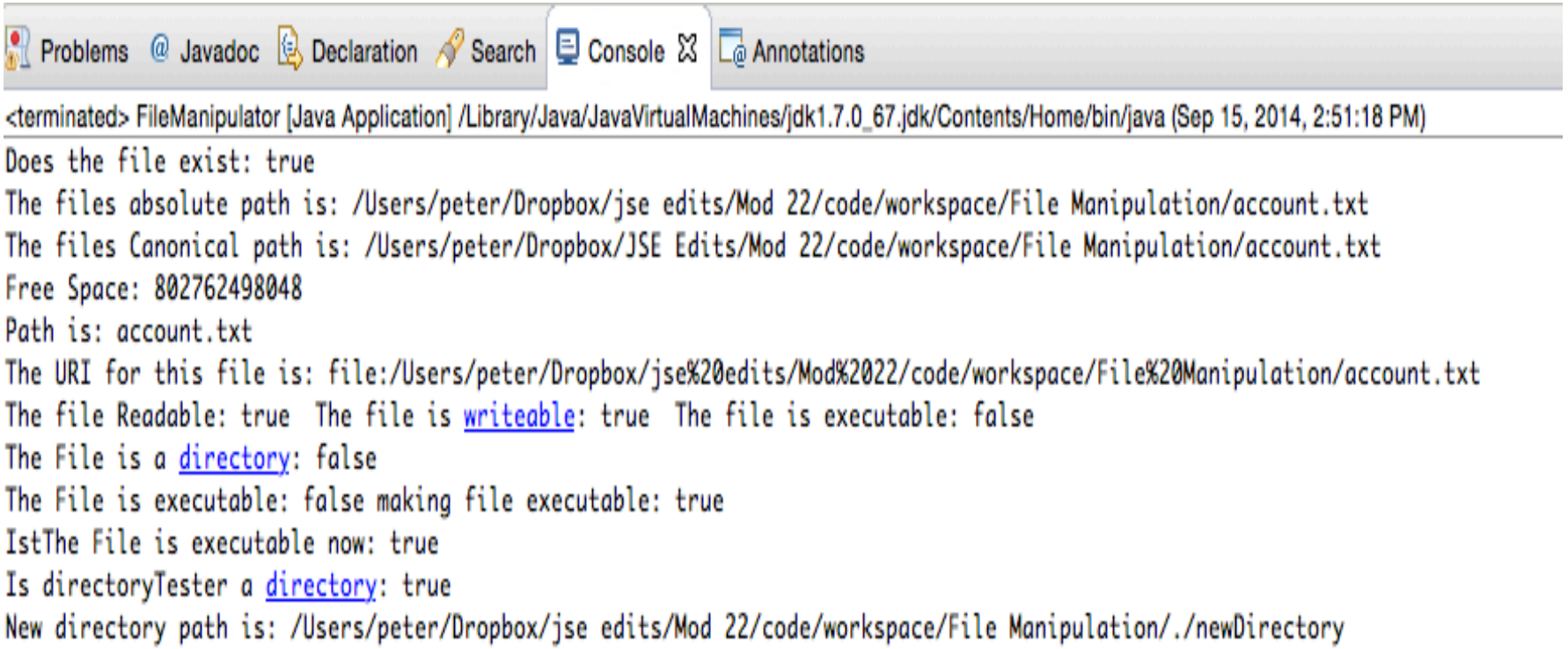
47 Methods in The File Class

There ar 47 methods, here are a few:

```
File fileTester = new File("account.txt");
File directoryTester = new File("./newDirectory");
    try {
        System.out.println(fileTester.exists());
        if(!fileTester.exists()){
          System.out.println("New File Was created: "+
                                    fileTester.createNewFile());
        }
System.out.println("The files absolute path is: “
                                    +fileTester.getAbsolutePath()+
        "\nThe files Canonical path is: ”
                                    +fileTester.getCanonicalPath()+
        "\nFree Space: ”           +fileTester.getFreeSpace()+
        "\nPath is: "+fileTester.getPath()+
        "  The file is writeable: "+
                                fileTester.canWrite()+"  The file is executable: "+
```

© DevelopIntelligence http://www.DevelopIntelligence.com

The File Class



Output from example

© DevelopIntelligence http://www.DevelopIntelligence.com

File Interactions

- Java provides classes to work with underlying file systems in a platform-independent manner.
- The `File` class allows you to create an object representation for a file.
- A `File` instance is not the file itself, but allows you to:
  - Find out information about the underlying file
  - Delete, rename, move the underlying file
  - Check permissions
  - Etc.

© DevelopIntelligence http://www.DevelopIntelligence.com

Byte Stream Classes

- Java provides classes to work with underlying file systems in a platform-independent manner.
- The `File` class allows you to create an object representation for a file.
- A `File` instance is not the file itself, but allows you to:
  - Find out information about the underlying file
  - Delete, rename, move the underlying file
  - Check permissions
  - Etc.

© DevelopIntelligence http://www.DevelopIntelligence.com

File Interactions

- Java provides classes to work with underlying file systems in a platform-independent manner.
- The `File` class allows you to create an object representation for a file.
- A `File` instance is not the file itself, but allows you to:
  - Find out information about the underlying file
  - Delete, rename, move the underlying file
  - Check permissions
  - Etc.

© DevelopIntelligence http://www.DevelopIntelligence.com

File Interactions

- The FileInputStream and FileOutputStream are designed to read streams of raw bytes from a file.
- Images and binary data files are good candidates for using these classes.

© DevelopIntelligence http://www.DevelopIntelligence.com

FileInputStream

```
FileInputStream fis = null;
int amountOfData = 0;
int runningTally = 0;
byte[] actualData = new byte[8];
File imageFile = new File("./images/imageFile.png");
try {
        // create new file input stream
        fis = new FileInputStream(imageFile);
        while (data != -1) {
          // read bytes to the buffer
          data = fis.read(bs);
         // count the bytes
         runningTally += 8;
        }
        System.out.print("Bytes read: " + runningTally
                                                / 1000 + "K");
        } catch (Exception ex) { //Catch and close blocks
```

© DevelopIntelligence http://www.DevelopIntelligence.com

FileOutputStream

```
public static void main(String[] args) throws IOException {
        File duke = new File("./images/duke.png");
        FileOutputStream dukeOut = new FileOutputStream(
                                "./images/copy_of_duke.jpg");
        FileInputStream dukeIn = new FileInputStream(duke);
        int data = 0;
        // Create a byte array to hold image
        byte[] image = new byte[(int) duke.length()];
        while (data != -1) {
          // read bytes to the buffer
          data = dukeIn.read(image);
        }
        dukeOut.write(image);
        dukeOut.flush();
}
```

© DevelopIntelligence http://www.DevelopIntelligence.com

Buffered Streams

- Streams read a byte of data at a time.
- Buffered streams aim to speed up this process by allowing the JVM to read or write large blocks of data all at once.
- It is much faster to read and write a large block of data all at once and then iterate through that data as you need it.
- Buffered streams also provide positional awareness within the data.

© DevelopIntelligence http://www.DevelopIntelligence.com

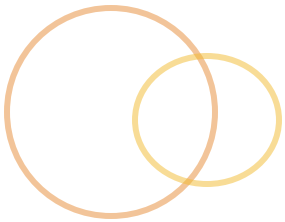
Chaining Streams

- Using one stream as an input for another stream is sometimes called chaining.
- Chaining increases speed, provides better positional awareness, and provides filtering capabilities

```
File milkyWay = new File("./images/milkyWay.jpg");
File copyOfMilkyWay = new File("./images/copy_milkyway.jpg");
FileOutputStream copyMWOut = new FileOutputStream(
                                        copyOfMilkyWay, false);
FileInputStream milkyWayIn = new FileInputStream(milkyWay);
BufferedInputStream fasterMWIn = new BufferedInputStream(milkyWayIn,
                                        (int) milkyWay.length());
BufferedOutputStream bufferedMWOut= new BufferedOutputStream(copyMWOut);
```

© DevelopIntelligence http://www.DevelopIntelligence.com

Go With the Flow



- Streams are simply flows of data that we are either sending to a sink or reading from a source.
- These input and output flows have no positional awareness within the flow.
- They are simply rivers of data moving through channels.
- There is no way to index the data while it is in the flow; it is sequential.

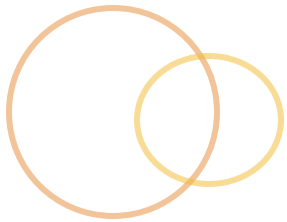
© DevelopIntelligence http://www.DevelopIntelligence.com

Character Readers

- Readers are interpreters, they read bytes of data and then the reader will translate into the language of your choice.
- Readers use specific charsets to determine what language to read.
- A charset is a mapping between Java's 16-bit Unicode representation of characters and a sequence of bytes.
- The class reading or writing the characters defines the decoders or encoders for retrieving specific charsets.

© DevelopIntelligence http://www.DevelopIntelligence.com

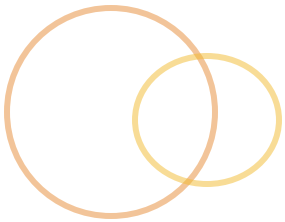
Character Readers



```
public class MainBranchFileReader {
        public static void main(String[] args) throws IOException {
                FileReader fr = new FileReader("TextExample.txt");
                BufferedReader br = new BufferedReader(fr);
                String characters;
                while ((characters = br.readLine()) != null) {
                        System.out.println(characters);
                }
                fr.close();
        }
}
```

© DevelopIntelligence http://www.DevelopIntelligence.com

Character Wrtiters



- FileWriters work in much the same way as a FileReader.
- The FileWriter does not allow character encoding when writing the file; it always uses the system's default encoding.
- In order to specify the specific encoding, we have to wrap the FileReader into a BufferedReader and then change the encoding.

© DevelopIntelligence http://www.DevelopIntelligence.com

Character Encoding

```
public static void main(String args[]) throws Exception {
 File charFile = new File("TextExample.txt");
 File charOutFile = new File("accounts.txt");
 char[] charBuffer =
 MainBranchWriter.getCharacters(charFile);

 BufferedWriter encodedOut = new BufferedWriter(
    new OutputStreamWriter( new FileOutputStream(charOutFile),
                                                    "KOI8-R"));
 encodedOut.write(charBuffer);
 encodedOut.close();
}
```

© DevelopIntelligence http://www.DevelopIntelligence.com

PrintWriter

- The PrintWriter is a one-sided stream;
  - there is not a PrintReader stream.
- The PrintWriter class shares the same methods that are in the PrintStream class that we so often use with System.out.println.
- The PrintWriter prints text-formatted representations of objects to a stream and will never throw an IO error after it has been created.

```
PrintWriter console = new PrintWriter(new BufferedWriter(
                new OutputStreamWriter(System.out)), true);
console.println("Time to check the Vault");
```

© DevelopIntelligence http://www.DevelopIntelligence.com

RandomAccessFile (RAF)

- The RandomAccessFile class has methods that allow you to search for a position in the file.
- RAF allows reading or writing from that position.
- The methods we use to manipulate the file pointer are:
  - **getFilePointer()** – returns the current position of the pointer
  - **seek(int)** – sets the pointer to a new position in the file
  - **read(byte[])** – reads data from the file from the current position up to byte[].length
  - **write(byte[])** – writes data from the byte[] starting at the current pointer position until current position + byet[].length.

© DevelopIntelligence http://www.DevelopIntelligence.com

The New IO

- The java.nio package was introduced in Java 4.
- It is intended to offer new capabilities for intensive I/O operations.
- Java 7 added the NIO2 or java.nio.file packages that include classes that break a file into a path object and the file object.
- The Java 7 version of NIO2 is much more complete and adds a file attribute package, the ability to watch a directory for changes that are event-driven, and many other capabilities.
- The java.nio package added some interesting tools worth taking a look at.

© DevelopIntelligence http://www.DevelopIntelligence.com

The ByteBuffer Class

- The ByteBuffer class is one of the cornerstones of the java.nio package.
- When a ByteBuffer is allocated, a call to **allocateDirect(size)** instructs the JVM to treat the ByteBuffer as a RAM-based buffer, allowing the native OS to read directly into native memory.
- This eliminates the middleman and prevents copying the contents of the stream into an intermediate buffer before or after calling a native OS IO operation.

© DevelopIntelligence http://www.DevelopIntelligence.com

File Channels

- The FileChannel is a channel that is connected directly to a file, allowing a program to read and write data to a file.
- FileChannels are flexible, allowing different channels to interact and transfer data between them.
- FileChannels provide a mechanism to lock a file that is being shared between threads.
- FileChannels use ByteBuffers for better performance.

© DevelopIntelligence http://www.DevelopIntelligence.com

File Channels

```
File file = new File(source);
File oFile = new File(sink);
FileInputStream is = new FileInputStream(file);
FileOutputStream fos = new FileOutputStream(oFile);
FileChannel reader = is.getChannel();
FileChannel writer = fos.getChannel();
ByteBuffer buf = new ThreadLocalByteBuffer(
                                ByteBuffer.allocateDirect(64 * 1024)).get();
long len = 0;
while((len = reader.read(buf)) != -1) {
buf.flip();
writer.write(buf);
buf.clear();
}
```

© DevelopIntelligence http://www.DevelopIntelligence.com

Summary

In this module, we covered how to:

- Describe the architecture of the `java.io` API
- Describe the streams model
- Use implementation streams
- Use filter streams
- Describe the difference between streams, readers and writers
- Use data streams and files
- Use buffered I/O and the `PrintWriter`
- Describe the `File` class

© DevelopIntelligence http://www.DevelopIntelligence.com