UCL

# COMP1008 Overview of File Handling





UCL

## Streams

- Have seen FileInput and FileOutput classes.
  - They are wrappers for the library file handling classes.
- Now want to look at Java library classes for file I/O.
- A stream is a sequence of values with a source and destination:
  - write/output: Program Data -> File.
  - read/input: File -> Program data.
- Java libraries define a number of stream classes.
  - Reader/Writer for dealing with character formatted data (e.g., unicode characters).
  - InputStream/OutputStream for dealing with unformatted data (bytes).






UCL

## Class File

- Provides a representation for file/directory *pathnames*.
  - Not the actual files/directories.
- Also provides methods to operate on the files/directories named.
- Creating a File object specifies name/path only:
  - File myFile = new File("data.txt");
  - File myDirectory = new File("/users/person/directory);
- See the Javadoc for full details.






UCL

## Manipulating Files/Directories

- File class contains number of methods to manipulate files:
  - exists
  - isFile
  - delete
  - createNewFile
  - renameTo
- For directories can use:
  - mkdir
  - listFiles
  - isDirectory






UCL

## PathNames

- Different operating systems represent paths with different separators:
  - Unix: /cs/students/fred/coursework
  - Windows: C:\cs\students\fred\coursework
- File class will attempt to translate given path to style supported on current machine
  - but c:\ is a problem...
  - also note that for \ you need to use an escape:
    - c:\\cs\\students\\fred\\coursework
- Can use File.separator instead of \ or /.






UCL

## Listing files

- Like the ls command

```
import java.io.*;
import java.util.*;
public class LS {
  public static void main(String[] args) {
    File currentDirectory = new File(".");
    String[] contents = currentDirectory.list();
    Arrays.sort(contents);
    for (int i = 0; i < contents.length; i++)
      { System.out.println(contents[i]); }
  }
}
```

"." stands for current directory - where program is run

## FileReader

- FileReader opens file for reading:
  - throws exception if open fails
  - FileReader reader = new FileReader("filename");
  - FileReader reader = new FileReader(fileObject);
- Provides basic set of read methods:
  - read character (mapped to character set).
  - read array of characters.
- Also has close method to close stream.






## FileWriter

- FileWriter opens file for writing:
  - throws exception if open fails
  - FileWriter writer = new FileWriter("filename");
  - FileWriter writer = new FileWriter(fileObject);
- Provides basic set of write methods:
  - write character.
  - write array of characters.
  - write String.
- Also has close method to close stream.
  - Important to close file, otherwise some data may not be written to file.






## FileWriter (2)

- Opening an existing file for writing, *deletes* existing contents,
- Unless append mode is selected:
  - FileWriter writer = new FileWriter(fileObject,true);
- Used for writing character based data.

All FileReader/Writer methods throw exceptions. Must use try/catch blocks or write methods with a throws declaration.






## Copying a text file

```
import java.io.*; // Note the import
public class Copy {
   public static void main(String[] args) throws IOException {
      File inputFile = new File(args[0]);
      File outputFile = new File(args[1]);
      FileReader in = new FileReader(inputFile);
      FileWriter out = new FileWriter(outputFile);
      int c;
      while ((c = in.read()) != -1) { out.write(c); }
      in.close();
      out.close();
   }
}
```

Note exception

Copy one character at a time. Read returns -1 when no more data.

java Copy file1 file2






## File Input/Output Streams

- Streams read/write byte data.
  - Raw data.
  - Use for *binary* data.
- FileInputStream
- FileOutputStream






## Copy any file as bytes

```
import java.io.*;
public class CopyBytes {
   public static void main(String[] args) throws IOException {
      File inputFile = new File(arg[0]);
      File outputFile = new File(arg[1]);
      FileInputStream in = new FileInputStream(inputFile);
      FileOutputStream out = new FileOutputStream(outputFile);
      int c;
      while ((c = in.read()) != -1) { out.write(c); }
      in.close();
      out.close();
   }
}
```

Copy one byte at a time. Type int used to store byte.

## PrintWriter

- A kind of Writer that reads its input (source), formats it, and writes to an output.
- System.out is actually a PrintWriter.
  - Provides character-based formatted output of primitive types and Strings.
- Can chain together Writer/Stream objects:
- PrintWriter pw = new PrintWriter(new FileWriter(...));
  - Can also create a PrintWriter directly on to a file using a File object.
  - Data -> PrintWriter (format) -> FileWriter -> text file
  - Use print/println methods.






## BufferedReader

- Reader that reads an entire line of text into a buffer and provides a readLine method to read complete line into a String

  BufferedReader in

  = new BufferedReader(new FileReader("data.txt"));
- String s = in.readLine();
- String can then be converted to other types (int, double, etc.)
  - This is what FileInput does.






## Counting Words

```
import java.io.*;
import java.util.*;
public class WordCount {
  public static void main(String args[]) throws IOException {
    Map<String,Integer> counts = new HashMap<String,Integer>();
    BufferedReader br = new BufferedReader(new FileReader(args[0]));
    String line;
    while ((line = br.readLine()) != null) {
      countWords(line, counts);
    }
    String[] keys = counts.keySet().toArray(new String[0]);
    Arrays.sort(keys);
    for (String word : keys) {
      System.out.println("Word: " + word + " count: " + counts.get(word));
    }
    br.close();
  }
```

Use a Map to store word counts. Then display map.






## Counting Words (2)

```
public static void countWords(String line, Map<String,Integer> counts) {
  StringTokenizer st = new StringTokenizer(line);
  while (st.hasMoreTokens()) {
    incrementCount(counts, st.nextToken());
  }
}

public static void incrementCount(Map<String,Integer> counts, String word) {
  Integer count = counts.get(word);
  if (count == null) {
    counts.put(word, 1);
  } else {
    counts.put(word, count + 1);
  }
}
```






## Scanner

- Class that "scans" input and translates to required type.
  - Library class that does what FileInput does.

```
import java.io.*;
import java.util.*;
public class ScanFor {
   public static void main(String[] args) throws IOException {
      Scanner s = new Scanner(new BufferedReader(new FileReader(arg[0])));
      while (s.hasNext()) {
         System.out.println(s.next());
      }
      s.close();
   }
}
```

hasNext returns true if there is more to scan. next returns next token (word separated by whitespace).






## Reading Doubles

```
import java.io.*;
import java.util.*;
public class SumFile {
   public static void main(String[] args) throws IOException {
      Scanner s = new Scanner(new BufferedReader(new FileReader(arg[0])));
      double sum = 0;
      while (s.hasNext()) {
        if (s.hasNextDouble() {
         sum += s.nextDouble();
        }
        else { next(); }
      }
      s.close();
      System.out.println(sum);
   }
}
```

Read double or skip if input cannot be converted to a double.

## Other Reader/Writers

- StringReader, StringWriter
  - read/write to/from strings rather than files or writers.
- InputStreamReader
  - Convert from stream to reader.
- OutputStreamWriter
  - Convert from writer to stream.
- Plus a family of InputStream and OutputStream classes.






## Summary

- Just an overview
  - Readers/Writers
  - Input/Output Streams
- Family of classes, providing wide range of features.
- See http://java.sun.com/docs/books/tutorial/essential/io/index.html for more information.