COMPSCI 105

Principles of Computer Science

Reading from Files

Contents

`java.io.File`

Information about the properties of a file

- Size
- Modification date
- Location of the file in the file system (i.e. the path)
- Access permissions
- Rename or delete a File

- Does not represent an actual file on disk
- Rather information about a potential file
- The actual file may (or may not) exist

Creating a `java.io.File`

Creating a new File

- Pass the pathname of the File to the constructor

```
File f = new File(pathname);
```

Absolute path name

- Specifies the drive letter
- Starts with the root directory
- Lists all subdirectories in the path
- Finishes with the file name

Relative path name

- List subdirectories from the current directories

Examples

- Note the use of \ in a Windows system pathname.

```
File f = new File("d:\\COMPSCI105\\A2\\testFile.txt");
File g = new File("\\A2\\testFile.txt");
File h = new File("simpleFile.txt");
```

File class - instance methods

`String[] list()`
Returns an array of strings naming the files and directories in the directory.

`boolean exists()`
Tests whether the file or directory exists.Tests whether the file or directory exists.

`long length()`
Returns the length of the file.

`boolean delete()`
Deletes the file or directory. Deletes the file or directory.

`boolean IsDirectory()`
Tests whether the file denoted by this abstract pathname is a directory.

`boolean isFile()`
Tests whether the file denoted by this abstract pathname is a normal file.

`String getAbsolutePath()`
Returns the absolute path String of the file or directory.

`String getName()`
Returns the name of the file or directory.

Example - File 01

Using the File class to find out the size of a file.

```
private void usingFileClassEx01() {
        File f = new File( "folder2/file1.txt" );
        if ( f.exists() && f.isFile() ) {
             System.out.println( f.length() + " bytes" );
        }
}
```

```
60 bytes
```


Example - File 02

Using the File class to find out the path.

```
private void usingFileClassEx02() {
      File f = new File( "Test" );
      if ( f.exists() && f.isDirectory() ) {
            System.out.print( "The directory, " );
            System.out.print( f.getName() );
            System.out.print( ", has the path: " );
            System.out.println( f.getAbsolutePath() );
      }
}
```

```
The directory, Test, has the path: S:\2007\L05\Test
```

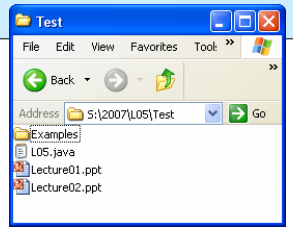

Directories

A directory (folder) is a special file that contains a list of other files.

- If a File object refers to a directory, then isDirectory() will return true
- In this case, list() will obtain a list of the files in the directory.

```
File f = new File( "Test" );
String[] fileNames;
if ( f.exists() && f.isDirectory() ) {
      System.out.println( f.getName() +" contains:" );
      fileNames = f.list();
      for( int i=0; i<fileNames.length; i++ ) {
            System.out.println( fileNames[i] );
      }
}
```

```
Test contains:
Lecture02.ppt
Lecture01.ppt
L05.java
Examples
```



File - Exercise 01

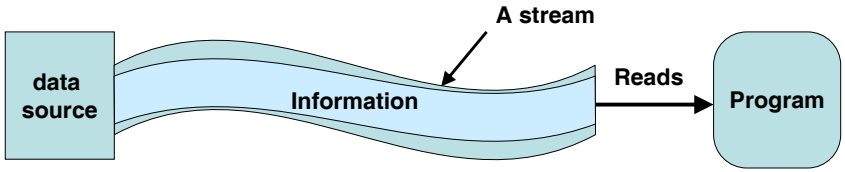
Complete the code below so that it prints out the name each text file in the directory "Test". Note that the `.endsWith()` method of the String class might prove useful.

```
File f = new File( "Test" );
String[] fileNames;
if ( f.exists() && f.isDirectory() ) {
      //Complete the code here









}
```


Input Stream

Stream of input data

- Keyboard
- File
- Network connection



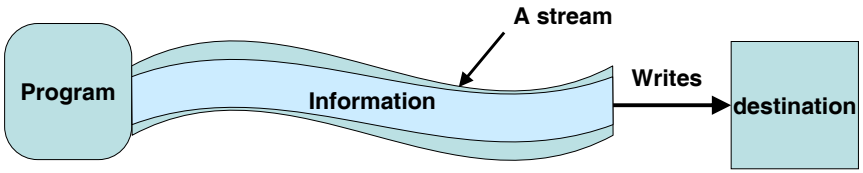
Algorithm

- open a stream
- while there is more data to read
  - read the next piece of data
- close the stream

Output Stream

Stream of output data

- Console
- File
- Network connection



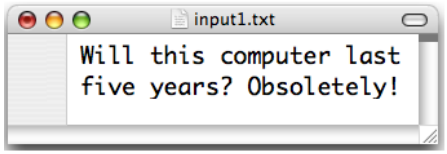
Algorithm

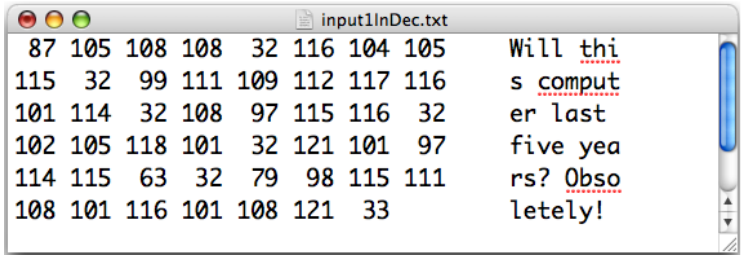
- open a stream
- while there is more data to write
  - write the next piece of data
- close the stream

Binary files and Text files

All data is stored in binary.

- We interpret the numbers in different ways
- Text is encoded using ASCII, Unicode, UTF-8
- Each letter stored using a number





java.io package

java.io package contains classes that do input and output

- Character Streams
  - Reading and writing text
- Byte Streams
  - Reading and writing raw data (bytes)

Java classes we will discuss

- See Java API for more information
- File
- FileReader
- FileWriter
- BufferedReader
- BufferedWriter
- PrintWriter

Text files

Many different ways to encode characters

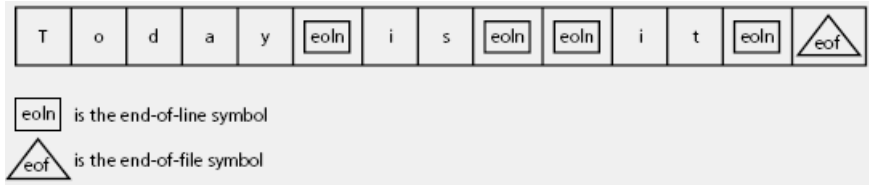
Internally, the Java char type uses Unicode
When writing to a character stream, Java uses UTF-8
Compatible with ASCII

End-of-line symbol

Creates the illusion that a text file contains lines

End-of-file symbol

Follows the last component in a file



FileReader

The FileReader class is used to read a file

- Automatically translate from internal encoding to external
- Makes it easy to read characters from disk

Creating a FileReader

- Using a File object
- Using the name of a file

```
FileReader( File file ) throws FileNotFoundException

FileReader( String fileName ) throws FileNotFoundException
```


Checked Exceptions

Checked exceptions

- Programmer is forced to explicitly deal with the exception
- Methods that throw unchecked exceptions must be called within a try…c atch block

```
FileReader( String filename ) throws FileNotFoundException
```

```
FileReader fr;
try {
       fr = new FileReader("Test");
} catch (FileNotFoundException) {
       System.out.println("Sorry, file does not exist");
}
```

FileReader - read()

`public int read() throws IOException`

- Read a single character.
- This method will block until a character is available, an I/O error occurs, or the end of the stream is reached.

Returns

- The character read, as an integer in the range 0 to 65535 (0x00-0xffff)
- -1 if the end of the stream has been reached

Throws:

- IOException - If an I/O error occurs

FileReader - close()

`public void close() throws IOException`

- Close the stream.
- Closing a stream that is already closed has no effect.
- Once a stream has been closed, further read() invocations will throw an IOException.

Throws:

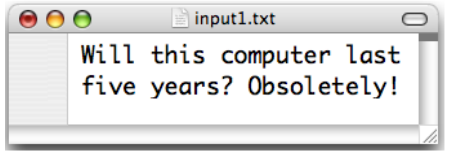
- IOException - If an I/O error occurs

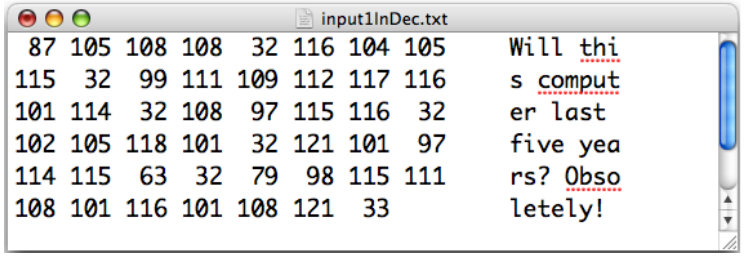
FileReader - example 1

```
private void fileReaderEx01() {
       FileReader fr;
       int num = 0;
       try {
              fr = new FileReader( "input1.txt" );
              num = fr.read();
              System.out.println( num );
              System.out.println( fr.read() );
              System.out.println( fr.read() );
              System.out.println( fr.read() );
              System.out.println( fr.read() );
              fr.close();
        }  catch( IOException e ) {
              System.out.println("Error: " + e);
        }
}
```

```
87
105
108
108
32
```


FileReader - reading characters





BufferedReader

Allows us to read entire String at a time

- Can only create once we have an existing Reader object
- Must pass a Reader object to the constructor of BufferedReader

```
FileReader fr;
BufferedReader br;
try {
      fr = new FileReader("Test");
      br = new BufferedReader( fr );
} catch (FileNotFoundException) {
      System.out.println("Sorry, file does not exist");
}
```

The program reads from the `BufferedReader`, which reads from the `FileReader` which in turn reads from file.

BufferedReader - readLine()

`public String readLine() throws IOException`

- Read a line of text.
- A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.

Returns

- String containing the contents of the line not including any line-termination characters.
- `null` if the end of the stream has been reached

Throws:

- IOException - If an I/O error occurs

BufferedReader - example 1

```
private void bufferedReaderEx01() {
      FileReader fr = null;
      BufferedReader br = null;
      String message = "";
      try {
            fr = new FileReader( "input.txt" );
            br = new BufferedReader( fr );

            String input = br.readLine();
            while( input != null ) {
                  message += input;
                  input = br.readLine();
            }
      } catch( IOException e ) {
            System.out.println( e );
      }
      System.out.println( message );
}
```


BufferedReader - example 2

```
private void bufferedReaderEx02() {
      FileReader fr = null;
      BufferedReader br = null;
      String message = "";
      try {
            fr = new FileReader( "input.txt" );
            br = new BufferedReader( fr );

            String input = br.readLine();
            while( input != null ) {
                  message += input + "\n";
                  input = br.readLine();
            }
      } catch( IOException e ) {
            System.out.println( e );
      }
      System.out.println( message );
}
```

BufferedReader - example 3

```
private void bufferedReaderEx02() {
	try {
		BufferedReader br = new BufferedReader(
				new FileReader("myFile.txt"));
		String line;
		while ((line = br.readLine()) != null) {
			System.out.println(line);
		}
		br.close();
	} catch (IOException e) {
		System.out.println(e);
	}
}
```


BufferedReader - Exercise 01

What is the problem with this code?

```
private void bufferedReaderProblem01() {
	FileReader fr = new FileReader( "input.txt" );
	BufferedReader br = null;
	String message = "", input;
	try {
		br = new BufferedReader( fr );
		while( (input = br.readLine) != null ) {
			message += input + "\n";
		}
	} catch( IOException e ) {
		System.out.println( e );
	}
	System.out.println( message );
}
```


BufferedReader - Exercise 02

Write a method that reads in the file "input.txt" and prints out the contents of the file in reverse.

For example, if the text file contained the text:
Hello,
this is a
simple file.

The output from your method should be:
.elif elpmis
a si siht
,olleH