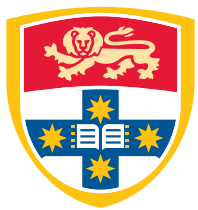
INFO1103: Introduction to Programming

School of Information Technologies, University of Sydney



Lecture 12: File Input/Output

Open, Read/Write, Close

Files

Files are an idea that makes information storage simple for users.

What kinds of file are these?

- HelloWorld.java
- family.jpg
- addresses.db
- birthdaylist.txt

Files contain information

There are no rules about what information is stored in a file. Text, images, binary data

The file name suffix[1] is there for the operating system to *identify* which program should be associated when opening the file.

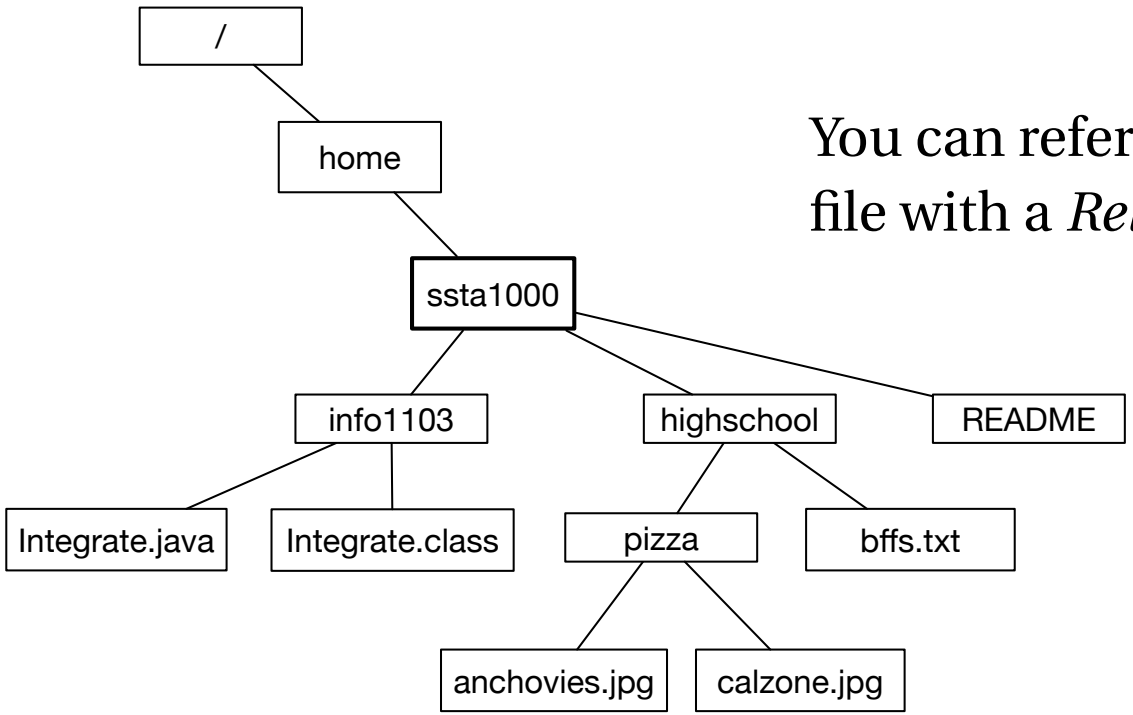
A unix/linux tool called `file` can scan the contents of a file and determine its type

```
~> file HelloWorld.java
HelloWorld.java: ASCII C++ program text
~> file HelloWorld.class
HelloWorld.class: compiled Java class data, version 50.0 (Java 1.6)
~> file runButton.png
runButton.png: PNG image, 30 x 24, 8-bit/color RGBA, non-interlaced
```

[1] those final letters after the full stop

Location of Files

There is a *path* associated with files



You can refer to the
file with a *Relative* path, or *Absolute* path

Where is calzone.jpg?

File I/O

To read from or write to from a file you need several things

1. The file has to be there
2. The file has to be available — it must be opened
3. You must have access to it
4. You must know what to read/write

Once you've finished with a file you should *always* close it.

Creating a `File` Object

First we need to create an File Object that will represent the pathname

`File`: *An abstract representation of file and directory pathnames.*

Consider the example where we open the file called README

```
1 import java.io.File; // symbol "File" found here
2
3 public class FileHandle {
4    public static void main(String [] args) {
5       File infile = new File("README");
6    }
7 }
```

What can go wrong here?
Will this compile?
Will this run?

Accessing the `File`

Next we will need some kind of access to it. There are *many* different ways to access a file, but the easiest is the `Scanner`

`Scanner`: A simple text scanner which can parse primitive types and strings

```
1  import java.io.File; // symbol "File" found here
2
3  public class FileHandle {
4      public static void main(String [] args) {
5          File infile = new File("README");
6          Scanner scan = new Scanner(infile);
7      }
8  }
```

Accessing the `File` (cont.)

Sadly, this won't compile!

```
   ~> javac FileHandle.java
FileHandle.java:6: error: cannot find symbol
      Scanner scan = new Scanner(infile);
      ^
  symbol:   class Scanner
  location: class FileHandle
FileHandle.java:6: error: cannot find symbol
      Scanner scan = new Scanner(infile);
                         ^
  symbol:   class Scanner
  location: class FileHandle
```

```
   ~> javac FileHandle.java
   FileHandle.java:10: unreported exception java.io.FileNotFoundException;
         Scanner scan = new Scanner(infile);
                        ^
   1 error
```

Accessing the `File` (cont.)

This error is telling us that there's an *uncaught exception* that "must be caught" or "declared".

This is where the idea of *exceptions* are important.

Suppose you have to read integers from a file called numbers.txt and print them to console

```
12
7
314
```

What could go wrong with the following code?

Reading Integers from File

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.Scanner;
4
5  public class FileReaderInt {
6     public static void main(String [] args) {
7        File infile = new File(args[0]);
8        try {
9          Scanner scan = new Scanner(infile);
10         while ( scan.hasNext() ) {
11             System.out.println( scan.nextInt() );
12         }
13       } catch (FileNotFoundException e) {
14             System.err.println( "file not found" );
15       }
16    }
17 }
```

Compiles? Works?

Reading Integers from File

Array bounds checking

Always expect integer

Didn't close file

Writing to File

Scanner only does reading, another object is needed to write to files.

`PrintWriter`: Prints formatted representations of objects to a text-output stream

But...

```
public PrintWriter(File file)
```

`file` - *The name of the file to use as the destination of this writer. If the file exists then it will be truncated to zero size; otherwise, a new file will be created.*

Potential data loss

Writing Doubles to File

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.io.PrintWriter;
4
5  public class FileWriterDouble {
6     public static void main(String [] args) {
7        if (args.length < 1) {
8           return;
9        }
10       File outfile = new File(args[0]);
11       try {
12         PrintWriter output = new PrintWriter(outfile);
13         output.println( 1.0 );
14         output.println( 3.14 );
15         output.println( Math.sqrt(2) );
16         output.println( 14.0/0.0 );
17         output.close(); // don't forget!
18       } catch (FileNotFoundException e) {
19             System.err.println( "file not found" );
20       }
21    }
22 }
```

Reading Text Files

Often a file contains many different parts. These need to be loaded into memory for the program to do useful work.

Example: read a file and separate the numerical data from text

The following file is "points.txt", it contains 2D point data of exactly 20 locations

```
4, 12
5, 3
18, 19
43, 27
82, 71
57, 45
...
140, 0
```

You are to extract the coordinates and store them an array of type `Point`

Reading Point data

```
1  import java.io.File;
2  import java.util.Scanner;
3  import java.awt.Point;
4
5  public class PointFileReader {
6     public static void main(String [] args) {
7        if (args.length < 1)
8           return;
9        File infile = new File(args[0]);
10       try {
11         Scanner scan = new Scanner(infile);
12         int location = 0;
13         Point [] points = new Point[20];
14         while ( scan.hasNextLine() ) {
15              String line = scan.nextLine();
16              // ???
17         }
18         scan.close(); // don't forget!
19       } catch (FileNotFoundException e) {
20              System.out.err( "file not found" );
21       }
22    }
23 }
```

Reading Point data

```
12  // extract tokens from line
13  String[] tokens = line.split(",");
14  if (tokens.length < 2) { // bad line, skip to next line
15      System.err.println("not enough tokens");
16      continue;
17  }
18
19  // parse integers from 1st and 2nd tokens
20  try {
21      int x = Integer.parseInt(tokens[0].trim());
22      int y = Integer.parseInt(tokens[1].trim());
23
24      // create a new point with data
25      Point newPoint = new Point();
26      newPoint.setLocation(x, y);
27
28      // update the array
29      points[location] = newPoint;
30      location = location + 1;
31
32  } catch ( NumberFormatException nfe ) { // bad number, skip to next
33      System.err.println("bad string conversion");
34      continue;
35  }
```

Reading Point data

What is the output when using the previous text file points2d.txt:

```
~> javac PointFileReader
~> java PointFileReader points2d.txt
```

If we change the input file points2d.txt

```
1, 2
3,
4  4
a , 5
          8              ,    7
1600, , 14 ...

9, 14, 32, 57
# frivolous comments!
```

What is the output?