CSY2030 Systems Design & Development

File Handling

File Handling

- Re-entering data all the time could get tedious for the user
- The data can be saved to a file
  - Files can be input files or output files.
- Files:
  - Files have to be opened.
  - Data is then written to the file or read from it
  - The file must be closed prior to program termination.
- In general, there are two types of files:
  - Text
  - Binary

Writing text to a text file

- Text files are readable
  - They can be used to store text e.g strings
- To write to a text file you need to import the ***java.io.**** package i.e have the following at the top of your java program:
  - **import java.io.*;**
- To open a file for text output you create an instance of the *PrintWriter* class e.g

```
PrintWriter outputFile = new PrintWriter(“StudentData.txt");
```

Pass the name of the file that you wish to open as an argument to the PrintWriter constructor.
Note you could do following to write to file in C:\ drive:

PrintWriter outputFile = new PrintWriter(“C:\\StudentData.txt");

Warning: if the file already exists, it will be erased and replaced with a new file.

The PrintWriter Class

- The *PrintWriter* class allows you to write data to a file using the *print* and *println* methods, as you have been using to display data on the screen.
- Just as with the *System.out* object, the *println* method of the *PrintWriter* class will place a newline character after the written data.
- The *print* method writes data without writing the newline character.

The PrintWriter Class

```
import java.io.*;

public class test {
    public static void main(String args[]) throws IOException
    {
        PrintWriter outputFile = new PrintWriter(“Names.txt");
        outputFile.println("Chris");
        outputFile.println("Kathryn");
        outputFile.println("Jean");
        outputFile.close();
    }
}
```

Import the package for reading and writing files

Throw exception if no file found

Open the file for writing

Write 3 strings to the file

Close the file

Names.txt file would have the following contents on the same folder as your java file
Chris
Kathryn
Jean

Appending Text to a File

- To avoid erasing a file that already exists, create a *FileWriter* object in this manner:
  - *FileWriter fw = new FileWriter(“Names.txt", true);*
- Then, create a *PrintWriter* object in this manner:
  - *PrintWriter outputFile = new PrintWriter(fw);*

Appending Text to a File

```
import java.io.*;

public class test {
	public static void main(String args[]) throws IOException
	{
		FileWriter fw = new FileWriter("Names.txt", true);
		PrintWriter outputFile  = new PrintWriter(fw);
		outputFile.println("Bob");
		outputFile.println("Jimmy");
		outputFile.close();
	 }
}
```

Open the file for appending

Pass object to PrintWriter object

Append 2 strings to the end of the file

Close the file

Names.txt file would now have the following contents:
Chris
Kathryn
Jean
Bob
Jimmy

Reading data from a file

- You use the *File* class and the *Scanner* class to read data from a file:

```
File myFile = new File(“Names.txt");
Scanner inputFile = new Scanner(myFile);
```

Pass the name of the file as an argument to the File class constructor.

Pass the File object as an argument to the Scanner class constructor.

- Once an instance of Scanner is created, data can be read using the same methods that you have used to read keyboard input (*nextLine*, *nextInt*, *nextDouble*, etc).

```
File file = new File("Names.txt");
Scanner inputFile = new Scanner(file);
String str = inputFile.nextLine();
inputFile.close();
```

Detecting The End of a Text File

- The *Scanner* class's *hasNext()* method will return true if another item can be read from the file.

```
import java.io.*;
import  java.util.*;

public class test {
            public static void main(String args[]) throws IOException
            {
                  File file = new File("Names.txt");
                  Scanner inputFile = new Scanner(file);
                  while (inputFile.hasNext())
                  {
                        String str = inputFile.nextLine();
                        System.out.println(str);
                  }
                  inputFile.close();
                  }
}
```

Binary Files

- Binary files cannot be opened in a text editor such as Notepad.
  - Binary files can be used to store objects
- To write data to a binary file you must create objects from the following classes:
  - *FileOutputStream* - allows you to open a file for writing binary data.
    - It provides only basic functionality for writing bytes to the file.
  - *DataOutputStream* - allows you to write data of any primitive type or String objects to a binary file. Cannot directly access a file.
    - It is used in conjunction with a *FileOutputStream* object that has a connection to a file.

Binary Files

- A *DataOutputStream* object is wrapped around a *FileOutputStream* object to write data to a binary file.

```
FileOutputStream fstream = new FileOutputStream("MyInfo.dat");
DataOutputStream outputFile = new DataOutputStream(fstream);
```

- These 2 statements can be combined into one i.e

```
DataOutputStream outputFile = new
        DataOutputStream(new FileOutputStream("MyInfo.dat"));
```

- If the file that you are opening with the *FileOutputStream* object already exists, it will be erased and an empty file by the same name will be created.

Appending Data to Binary Files

- The *FileOutputStream* constructor takes an optional second argument which must be a boolean value.
  - If the second argument is true, the file will not be erased if it exists; new data will be written to the end of the file.
  - If the second argument is false, the file will be erased if it already exists.

```
FileOutputStream fstream = new FileOutputStream("MyInfo.dat", true);
DataOutputStream outputFile = new DataOutputStream(fstream);
```

Appending Data to Binary Files

- When writing objects to a binary file the associated class must be serialised
  - This makes an object represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.
  - After a serialised object has been written into a file, it can be read from the file and deserialised that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.

Reading and Writing Objects Example

```
import java.io.Serializable;

public class student implements Serializable {
        private String name;
        private String address;

        public student(String name, String address) {
                this.name = name;
                this.address = address;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getAddress() {
                return address;
        }

        public void setAddress(String address) {
                this.address = address;
        }
}
```

Import the Serializable package so objects can be represented as a sequence of bytes

Make class Serializable so its objects can be represented as a sequence of bytes to be Written to and read from file

```
import java.io.*;

public class test {
        public static void main(String[] args) throws Exception {
        try {
                student fred = new student("fred", "Northampton");
                student bob = new student("bob", "Glasgow");
                FileOutputStream fos = new FileOutputStream("students.dat");
                ObjectOutputStream oos = new ObjectOutputStream(fos);
                oos.writeObject(fred);
                oos.writeObject(bob);
                oos.close();

                FileInputStream fis = new FileInputStream("students.dat");
                ObjectInputStream ois = new ObjectInputStream(fis);
                student obj = null;

                while ((obj=(student)ois.readObject())!=null) {
                                System.out.println("Name:" + obj.getName() + ", Address:" + obj.getAddress());
                }
                 ois.close();

         } catch (EOFException ex) {  //This exception will be caught when EOF is reached
                 System.out.println("End of file reached.");
            } catch (ClassNotFoundException ex) {
                 ex.printStackTrace();
            } catch (FileNotFoundException ex) {
                 ex.printStackTrace();
            } catch (IOException ex) {
                 ex.printStackTrace();
            }
        }
}
```

Create 2 student objects

Set up binary file called *students.dat* for writing

Write the 2 student objects to binary file

Close file

Set up binary file called *students.dat* for reading

Create a null student object which will hold Student objects read in from file

Check for end of file i.e make check if an object has been read in

Display details of the object read in

Close file

Do exception handling