Worksheet 8: File I/O

Updated: 12th May, 2020

- Use an Array of Objects
- Implement and understand how to read in a CSV file
- Implement and understand how to write to a CSV file
- Use arrays in conjunction with file IO

Note: You must submit this practical worksheet by:

Sunday, 17th May 2020, 5pm (local time)

And have it signed off during your next **registered** practical session.

Submissions must be done **electronically** through Blackboard. You are only required to submit once for this practical (your **entire** `P08` directory in **`.tar.gz`** format). You must submit everything you have completed in this practical, both in class and at home.

You must also not modify the files in `~/Documents/PDI/P08` until after your submission has been marked. During your practical your tutor will check the modified dates of the files using `ls -l` (el) before marking your work. To create a gzipped tarball use the following command from your **`PDI`** folder:

```
[user@pc]$ tar -cvzf <studentID>_P08.tar.gz P08
```

For more information on what each argument of the above command does use:

```
[user@pc]$ man tar
```

To get your **`.bash_history`** into a submittable format, first close all terminals down, using <ctrl>-d. Then open a new terminal, and type this command from anywhere:

```
[user@pc]$ history >~/Documents/PDI/P08/BashHistoryP08.txt
```

Your submission will be marked during your next **registered** practical session via an online interview (in `Blackboard Collaborate Ultra`) from your tutor, comprised of a few questions.

Please note that the questions asked in the interview may cover the entirety of the worksheet, not just the material in your Assignment Task. Your submitted work will be assessed and marks will be allocated accordingly.

1. Student Class

The university has asked you to design a "**Student**" Class that will be able to store a students name, as well as 1 of their Test Marks. You are required to store:

- **String name** – It is considered valid if not blank or `null`
- **Integer studentID** – It is considered valid if it is between `10000000` and `99999999`
- **Real mark** – It is considered valid if it is between `0.0` and `100.0` (inclusive)

While writing this class, as well as having all of the required methods from the lecture slides (3 Constructors, Accessors for each Classfield, Mutators for each Classfield, **toString**, **equals** and **clone**) you are required to have the following:

- **String getGrade()** – Returns the "grade" equivalent of the **mark**
- **String toFileString()** – Returns the object reconstructed into its CSV equivalent. e.g., `"Mark Upston,10000001,100.0"`

Once written in pseudocode, convert it to Java and test it fully.

2. Reading in Students

You are required to design an algorithm that reads in a list of Student Names, ID's and Test Marks from a CSV file. The format of the file is as follows:

```
<NAME1>,<ID1>,<TESTMARK1>
<NAME2>,<ID2>,<TESTMARK2>
```

If any element is invalid then your program should reject the entire line but also keep reading the file. There will not be any spaces after a comma, if there are you may assume the input is invalid. You must import **java.io.*;** for this Worksheet.

```
Select an option:
1. Import Student's
2. Calculate assessment average
3. View all Student's
0. Exit
```

Import Student's should prompt the user for the name of a file to read. Then proceed to read that file and import it to the program.

An example of a valid file:

```
Christopher Lee,12345678,54.0
Mark Upston,10000001,100.0
Oliver Stewart,18649532,75.8
Jessica Chang,14976542,34.65
```

- An empty file is not considered valid.
- If the file is empty your program should display a message indicating so (exceptions).
- You must store the contents of the file into one **Student** Array.

The maximum number of **`Student`**'s that can be stored is 20. You should create a **`Student`** array of size 20. You can then populate this array from the file. If a file has more than 20 entries, read the first 20 entries and then print an error message to the user specifying that the storage capacity has been exceeded and only the first 20 entries will be searched.

A file is not guaranteed to have 20 lines, so you should also store an integer count that keeps track of how many valid **`Student`**'s have been stored. When searching through the arrays you should be using this count as your end point, not the actual array size.

In order to convert a mark from the String input you get from the file to an actual real number you can validate, you will need to "parse" it. Look up the Double class in the Java API (yes the capital D is important), can you find a method that converts a string to a Real number?

> **Note:** A line is not guaranteed to be complete, a line may be blank, may be missing a mark, or may not have a valid number as the mark. In each of these cases your program will need to discard the invalid line and **continue reading** the rest of the file.

`Calculate assessment average` should loop through the marks for each student and calculate an average for that assessment.

> **Note:** You may write an "**`average`**`(double[])`" method in your **`PDIMath`** Class that calculates the average of an array of Reals for your convenience. Although it may seem like extra overhead for this instance, it may come in handy later.

`View all Student's` should do as it says in the name, and print out all of the Students currently stored in the program. e.g., If only `"Mark Upston"` was stored it would display: `"Mark Upston (10000001) scored 100.0%`

`Exit` should exit the program.

Once written in pseudocode, convert it to Java and test it fully. You will need to write your own valid file and also test with invalid cases to ensure it works fully.

3. Adding a Student

You now should give the ability for the user to input the values for a student and then construct and object of it and add it to the next position in the array.

Your menu should look like this:

```
Select an option:
1. Import Student's
2. Calculate assessment average
3. View all Student's
4. Add a Student
0. Exit
```

Ensure you are validating everything that will be gathered from the user and adding the new student to the next available spot in the array. If the array is full, the user shouldn't be prompted to enter in anything.

Once written in pseudocode, convert it to Java and test it fully.

4. Writing Students to a File

Once you have processed the input from the file you should give the option for the user to export to a file.

Your menu should look like this:

```
Select an option:
1. Import Student's
2. Calculate assessment average
3. View all Student's
4. Add a Student
5. Export Student's
0. Exit
```

You should write the contents of the array to a file gathered from the user. The output file should be the format shown below.

```
<NAME>,<ID>,<TESTMARK>
<NAME>,<ID>,<TESTMARK>
```

Have a look at the lecture slides on how to implement writing to a file. Feel free to use the method you wrote for your **Student** Class **toFileString()**

The output should look identical to the file you read in plus any Student's you added.

Once written in pseudocode, convert it to Java and test it fully.

5. Assignment Task Part 1: FileIO Class

Your second last task for this practical is to complete the following task **in your own time**.

> **Warning:** This question goes towards your portfolio/assignment mark and thus any collusion will be dealt with as per university policy.

Create a subdirectory in this week's `P08` directory and call it **`AssignmentTask`** your files should be called **`FileIO.txt`** and **`FileIO.java`** respectively.

Now you need to create a Static class that is designed to handle the File IO for your Assignment so far. It will be tasked to read in 2D arrays from a file and returning the reconstructed array to the caller. It will also be tasked with writing them back to a file.

These arrays need to be a rectangle, so if there is a ragged/staggered array then it is considered invalid. You are also required to accept any length and width `>0`. These could be greater than (`1000,1000`) so ensure your algorithm doesn't hardcode anything.

An example of an array could be:

```
10,10,10,0,0,0
10,10,10,0,0,0
10,10,10,0,0,0
10,10,10,0,0,0
10,10,10,0,0,0
10,10,10,0,0,0
```

```
12,37,53,45,96,16
54,26,81,94,34,78
02,46,81,24,46,73
96,65,76,46,25,64
34,38,95,18,12,05
46,98,13,34,84,54
```

The methods required in this class:

- `int[][] readFile(String fileName)` – Attempts to open the file and read its contents into a 2D array and return that to the caller.
- `void writeFile(String fileName, int[][] writeArray)` – Attempts to write a 2D array to a file.
- `int[][] readPNG(String fileName)` – Attempts to open a `.png` file and construct a 2D array to return to the caller.
- `void writePNG(String fileName, int[][] writeArray)` – Attempts write a 2D array to a `.png` file.

Plus any other `private` methods that you deem necessary to produce this. The lecture slides have many examples to use.

> **Note:** The last 2 methods in Java have been provided for you. You will need to import:
>
> ```
> import java.io.*;
> import java.awt.*;
> import java.awt.image.*;
> import javax.imageio.*;
> ```

Note: You must not use the **ImageIO**, **File**, **Color** or **BufferedImage** class in any other methods other than the ones below.

```
public static int[][] readPNG(String fileName)
{
    BufferedImage img;
    File inputFile;
    int[][] image = null;

    try
    {
        // Open the file
        inputFile = new File(fileName);

        // Turn file into an Image
        img = ImageIO.read(inputFile);

        // Construct array to hold image
        image = new int[img.getHeight()][img.getWidth()];

        // Loop through each pixel
        for (int y = 0; y < img.getHeight(); y++)
        {
            for (int x = 0; x < img.getWidth(); x++)
            {
                // Turn the pixel into a Color object.
                Color pixel = new Color(img.getRGB(x, y), true);

                // Converts each pixel to a grayscale equivalent
                //    using weightings on each colour.
                image[y][x] = (int)((pixel.getRed() * 0.299) +
                                    (pixel.getBlue() * 0.587) +
                                    (pixel.getGreen() * 0.114));
            }
        }
    }
    catch(IOException e)
    {
        UserInterface.displayError("Error with .png reading: "
                                   + e.getMessage());
        // Alternatively you could rethrow an IllegalArgumentException
    }
    return image;
}
```

```java
public static void writePNG(String fileName, int[][] writeArray)
{
    // The following is very Java specific and is implemented in a way to
    // reconstruct a colour image from a set of 8bit colours.
    BufferedImage theImage;
    File outputfile;

    try
    {
        // Open the file
        outputfile = new File(fileName);

        // Construct a BufferedImage, with dimensions and of type RGB
        theImage = new BufferedImage(writeArray[0].length,
                                     writeArray.length,
                                     BufferedImage.TYPE_INT_RGB);

        // This will step through each element of our "writeArray"
        for(int y = 0; y < writeArray.length; y++)
        {
            for (int x = 0; x < writeArray[0].length; x++)
            {
                // This will ensure that we are only putting a value into
                //     our png, between 0 and 255. (8bit colour depth)
                int value = Math.abs(writeArray[y][x] % 256);

                // Turns the greyscale pixel to a "colour" representation
                Color newColor = new Color(value, value, value);

                // This will set the value of the pixel within the .png
                theImage.setRGB(x, y, newColor.getRGB());
            }
        }

        // Write the image to a .png
        ImageIO.write(theImage,"png",outputfile);
    }
    catch(IOException e)
    {
        UserInterface.displayError("Error with .png reading: "
                                   + e.getMessage());
        // Alternatively you could rethrow an IllegalArgumentException
    }
}
```

6. Assignment Task Part 2: DetectEdges Class

Your last task for this practical is to complete the following task **in your own time**.

> **Warning:** This question goes towards your portfolio/assignment mark and thus any collusion will be dealt with as per university policy.

Your files should be called **`DetectEdges.txt`** and **`DetectEdges.java`** respectively.

Your task is to write a program using **Part 1** that prompts the user to import a kernel in a CSV format, then import an image either in CSV format or in **`.png`** format.

Once imported and validated you must perform a convolution operation on the image imported along with the kernel. Then export the image as a **`.png`**. The file name should be the filename of the original image, with "**`_Converted.png`**" concatenated on the end.

An example of your program running could look like this:

```
[user@pc]$ java DetectEdges

Please enter the filename of the kernel: VerticalKernel.csv
Would you like to perform on (C)SV or (P)NG: p
Please enter the filename of the PNG: MyImage.png
File (MyImage.png_Converted.png) written. Goodbye!
```

Some **`.png`** and **`.csv`** files have been provided for you on Blackboard to use in the archive: **`Supplementary_Files.tar.gz`**. However, feel free to use your own.

Ensure you test your algorithm. Obviously it will be hard to test the **`.png`** output, but try supply it an image/array that clearly has a line in it and test to see if it is detectable with the corresponding kernel.

End of Worksheet