Problem

Consider the following learning set:

1	7	-2	4.5
2	3	-5	2.3
1	6	-2	1.2
2	5	-2	4.5
1	6	-5	2.3

Requirements:

a) Read the learning set from a file saved on your local drive (in.txt) The values on each line will be separated by space

b) Transform the learning set to the normalized form:
$x_{ij} = (x_{ij}-x_{jmin})/(x_{jmax}-x_{jmin})$

c) Write the normalized form to a .csv file (comma separated values)

Steps & Hints

1. Open Eclipse and create new Java Project: rf

2. Create new package: ro.usv.rf

Package names *are written in all lower case to avoid conflict with the* ***names*** *of classes or interfaces. Companies use their reversed Internet domain* ***name*** *to begin their* ***package names****—for example, com.example.mypackage for a* ***package*** *named mypackage created by a programmer at example.com*

3. Create new class: MainClass with public static void main() method

4. Create class FileUtils (will be used to read and write to files)

5. In FileUtils.java, create a protected static method to read from file

Java static method

- *A static method belongs to the class rather than object of a class.*
- *A static method can be invoked without the need for creating an instance of a class.*
- *static method can access static data member and can change the value of it.*

Reading from files before Java8:

```
try {
     File file = new File("c:\\test.txt");
     FileReader fileReader = new FileReader(file);
     BufferedReader bufferedReader = new  BufferedReader(fileReader);
              StringBuffer stringBuffer = new StringBuffer();
              String line;
              while ((line = bufferedReader.readLine()) != null) {
                       stringBuffer.append(line);
                       stringBuffer.append("\n");
              }
              fileReader.close();
              System.out.println("Contents of file:");
              System.out.println(stringBuffer.toString());
   }
  catch (IOException e) {
              e.printStackTrace();
   }
```

Reading from files with Java8:

```
String fileName = "c:\\test.txt";

//read file into stream, try-with-resources

try (Stream<String> stream = Files.lines(Paths.get(fileName))) {

         stream.forEach(System.out::println);

} catch (IOException e) {

         e.printStackTrace();

}
```

The Java 8 API has a new abstraction called Stream *that lets you process data in a declarative way. More details: http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html*

How to read learning set from file:

***Option 1) use bi-dimensional array**: private double[][] learningSet = new double[5][5];*

Pros: simplicity

***Cons:** we need to know the exact size of matrix when we initialize it. In order to do this, we would have to read the file twice (first to get the sizes (n and p) and second time to populate the matrix)*

Option 2) – recommended:

Start with an `ArrayList<ArrayList<Double>>`, and then as soon as you've finished reading the file, turn it into an `double[][]` for performance

ArrayList** is a class that implements **List.

***ArrayList** supports dynamic arrays that can grow as needed. Standard **Java** arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold*

https://docs.oracle.com/javase/tutorial/collections/implementations/list.html

6. In FileUtils, create a protected static method to write to a double[][] data structure in a .csv file (comma separated values)

Example output (out.csv):

1,2,3,4,5

6,7,8,9,10

2.1,3.2,3,6.7,1

```
package ro.usv.rf;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class FileUtils
{
	private static final String inputFileValuesSeparator = " ";
	private static final String outputFileValuesSeparator = ",";

	protected static double[][] readLearningSetFromFile(String fileName)
	{
		//Start with an ArrayList<ArrayList<Double>>
		List<ArrayList<Double>> learningSet = new ArrayList<ArrayList<Double>>();
		// read file into stream, try-with-resources
		try (Stream<String> stream = Files.lines(Paths.get(fileName))) {
			learningSet = stream.map(FileUtils::convertLineToLearningSetRow).collect(Collectors.toList());
		} catch (IOException e) {
			e.printStackTrace();
		}
		//  convert ArrayList<ArrayList<Double>> to double[][] for performance
		return convertToBiDimensionalArray(learningSet);
	}

	private static double[][] convertToBiDimensionalArray(List<ArrayList<Double>> learningSet) {

		double[][] learningSetArray = new double[learningSet.size()][];

		for (int n = 0; n < learningSet.size(); n++) {
			ArrayList<Double> rowListEntry = learningSet.get(n);

			// get each row double values
			double[] rowArray = new double[learningSet.get(n).size()];

			for (int p = 0; p < learningSet.get(n).size(); p++)
			{
				rowArray[p] = rowListEntry.get(p);
			}
			learningSetArray[n] = rowArray;

		}
		return learningSetArray;
	}

	private static ArrayList<Double> convertLineToLearningSetRow(String line)
	{
		ArrayList<Double> learningSetRow = new ArrayList<Double>();
		String[] stringValues = line.split(inputFileValuesSeparator);
		//we need to convert from string to double
		for (int p = 0; p < stringValues.length; p++)
		{
			learningSetRow.add(Double.valueOf(stringValues[p]));
		}
		return learningSetRow;
	}

	protected static void writeLearningSetToFile(String fileName, double[][] normalizedSet)
	{
		// first create the output to be written
		StringBuilder stringBuilder = new StringBuilder();
		for(int n = 0; n < normalizedSet.length; n++) //for each row
```

```
                {
                        //for each column
                        for(int p = 0; p < normalizedSet[n].length; p++)
                        {
                                //append to the output string
                                stringBuilder.append(normalizedSet[n][p]+"");
                                //if this is not the last row element
                            if(p < normalizedSet[n].length - 1)
                            {
                                    //then add separator
                                    stringBuilder.append(outputFileValuesSeparator);
                            }
                        }
                        //append new line at the end of the row
                        stringBuilder.append("\n");
                }
                try {
                        Files.write(Paths.get(fileName), stringBuilder.toString().getBytes());
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

}
```

7. Create method to normalize the learning set

```
private static double[][] normalizeLearningSet(double[][] learningSet)
{
        double[][] normalizedLearningSet = new double[learningSet.length][];
        //.. enter your code here
        return normalizedLearningSet;
}
```

8. Run your main method

```
public static void main(String[] args) {
    double[][] learningSet = FileUtils.readLearningSetFromFile("e://in.txt");
    FileUtils.writeLearningSetToFile("e://out.csv",
normalizeLearningSet(learningSet));

}
```

Discussion points:

Encapsulation, Statics, Data structures

Primitives vs Wrappers (e.g double vs Double)

Java 8 Streams

StringBuilders
