Swinburne University of Technology
Faculty of Information and Communication Technologies

Programming Hurdle

List Processing

Team size: To be completed individually
Due Date: Final deadline: 5:30pm, Monday 11-Aug-2014
Marks: This assignment will contribute a Pass or Fail result to your final assessment.
You must pass this assignment to pass the subject.

Overview

Develop a simple utility in both Java and C# that can process data files and produce some metrics of the data the files contain.

The objective of this assignment is to revise object-oriented structuring of programs and ensure that you have sufficient programming competency to undertake this subject.

Note: It is understood that not all students have used both the Java and C# languages. However, a student enrolled in this subject is expected to be capable of learning a new programming language and completing this hurdle task as an indication that they are ready to approach the rest of the subject.

If you feel that you are unable to do this, you are strongly advised to reconsider your enrolment in the subject while you still can without penalty.

Instructions

Complete the implementations of the programs and submit a single ***zip*** *compressed file* containing all items specified for assessment. **Do not submit any redundant items**. Ensure the file names are consistent with the script for compiling/running the program given below.

Submit your work to ESP before the deadline (esp.ict.swin.edu.au).

Submission Instructions

For this assignment you will need to submit the following artefacts:

- A Java application
  a. Completed source code
  b. Ant script to build the program
    i. Use **compile** target within the build.xml
- A C# application
  c. Completed source code
  d. A **MSBuild script**, **batch file** or **makefile** that builds the project
    i. If you supply a batch file it must be called **build.bat**
    ii. If you supply a **makefile** it must have an **all** target
- A UML class diagram which corresponds to the design of your programs.

Do not include compiled programs in your zip file submission as the marker will compile and run your programs using your configuration files. If the marker cannot compile and run your code as specified, you will be awarded a Fail for the assignment. It is your responsibility to ensure your scripts and build files are portable.

Ranking by Total Value Program

Develop a program which processes several files containing a list of vectors each. The file has metadata which contains a field *algo_name* to identify the group of vectors which are listed in the file. The algorithm has produced these vectors as solutions to a problem. (The problem itself is irrelevant.) The entries in each vector express the solution's quality with respect to a criterion. The values for the different criteria do not have the same scale. Therefore, the values have to be standardised before we can add them up to obtain an overall comparison for the solutions.
Given the standardised total, the program determines which three of the vectors for each algorithm rank best.

The input txt file format is provided below. The program must calculate the total standardised quality of each vector and display the best three vectors' total for each algorithm in the console.

Your program must be implemented in **both** Java and C#, resulting in a total of two programs within the submission. It is expected that the object-oriented design will be similar across both languages. Provide a UML class diagram that accurately reflects the design in both languages.

Background

Multicriteria optimisation finds solutions to problems which have several criteria. For example, if you buy a car, you will be interested in its price, its condition, how many litres of petrol it consumes, etc. Solutions to such a problem can be expressed as vectors which contain the quality of each solution according to each of these criteria. Each vector entry (dimension) represents the value of the solution according to a criterion, such as price, running cost, insurance premium, etc. Vectors ($x= [x_1, x_2, .. x_k]$), where x_1 is the value of a solution according to one criterion, can have any number of dimensions *k*, but at least two will be present.

When we use algorithms to produce solutions to such problems, we obtain several solutions which are optimal in different criteria. Therefore, to compare the performance of algorithms, we have to find a way of deciding which solutions are best when all criteria are considered.

Since the scales of the criteria vary, we cannot simply add the values of the dimensions of a vector. We can only do this after we have standardised the dimension values. Standardisation expresses the value as a factor of the standard deviation of all solutions' values in that dimension.

This means we have to add all solutions' values in a single dimension to obtain the average, then the standard deviation. If we have *k* dimensions, we have *k* averages μ and *k* standard deviations σ.

$$\text{standardised total} = \sum_{i=1}^{k} \frac{x_i - \mu_i}{\sigma_i}$$

When each solution has standardised total value, we can determine which algorithm produced the best solutions and how much better or worse the solutions are. The three best solutions' number, standardised values and the algorithm that produced it are shown as output on the console.

Program: Results Comparison

Your program must be able to calculate the standardised values for each vector.

Input data format
The program must be able to read this file format. There is plenty of metadata, of which you only need the value of the algo_name field. The actual list of vectors starts after the line "Complete Population". Each vector occupies a single line in the file. There can be any number of vectors, but you do not have to cater for the case where they do not all fit into memory.
The program must be able to read and process a varying number of files (no more than 10) and varying numbers of vector dimensions.

```
#Tue May 14 08:27:25 EST 2013
mating_pool=80
mutation_dist=3
algo_name=ARMO
stopping_criterion=1000000
comparator=multiObjective.RankingComparator
sol_type=solver.mosolver.MOGASolution
algo_type=solver.mosolver.RankingMOGA
logging=false
rand_seed_for_pop=888
range_subdivisions=10
desired_solutions=5,10,15
problemfile=50fac5to20obj\\\\50fac5obj.txt
o=2
HV_sample=1000
runs=30
k=10
rand_seed=clock
problemtype=MOQAP
tournament_size=2
vads=true
mutation_prob=0.1
reweighting_interval=100
c=8
crossover_prob=0.01
delta=10
population=100
Used function evaluations: 1000000

Complete Population
8.78792396E8 7.45689508E8 8.37899916E8 9.52778502E8 8.47061622E8
8.80017166E8 7.50224432E8 8.23658404E8 9.51664198E8 8.49145008E8
8.85724416E8 7.48191542E8 7.61295532E8 1.00892758E9 8.52389824E8
8.96069156E8 7.11234404E8 7.68007126E8 9.7238065E8 8.5759227E8
8.96193522E8 7.11177522E8 7.67777526E8 9.72449466E8 8.5763106E8
8.95546766E8 7.1112849E8 7.68311754E8 9.71998374E8 8.57960886E8
8.95480802E8 7.11023308E8 7.68223532E8 9.72097758E8 8.5803376E8
8.9549393E8 7.11015392E8 7.68194136E8 9.72079838E8 8.5804897E8
8.95467666E8 7.11364074E8 7.68318732E8 9.7189094E8 8.58053462E8
…..
```

Use this field to identify the group of vectors

Group of vectors to rank. Each line is one vector, values separated by a space. This example has 5 dimensions. The number of vectors/dimensions can vary.

Invocation format

The java program will be run using:

java ResultsComparison <file1.txt,file2.txt,file3.txt>

The C# program will be run using:

ResultsComparison <file1.txt,file2.txt,file3.txt>

Note: At least two file names have to be provided as arguments. If there are insufficient arguments, show an error message.

Usage

Usage illustrated with Java execution, the same results are expected from the .NET version.

```
$ java ResultsComparison result_5_a.txt,result_5_b.txt,result_5_c.txt
Number of solutions compared: 9000
ARMO
[8], 8.46
[22], 7.34
[13], 7.18
Hype
[97], 3.77
```

[19], 2.91
[35], 2.11
IBEA_eps
[82], 5.64
[48], 3.17
[7], 2.88

(Number of solutions depends on the number of input files. Round the average rank to the nearest integer.)

Output format
The application must write the result to the console as demonstrated above. Each algorithm is followed by the number of the solution (according to the ordering in the file) and its standardised total quality.

Object-Oriented Design

This is one of the main reasons students fail the assignment. A solution must be represented by an object, which also specifies the algorithm that produced it. Two-dimensional arrays and a separate array for names (where entries are matched using the index) are not acceptable.
Input/output is specialised functionality that is taken care of by specialised objects. E.g. solution objects DO NOT write to the console.

Additional requirements

- Only use standard Java/.NET libraries.
- Do not use packages/namespaces. All files **must be in the root folder** of your zip file.
- All classes must have a comment that contains the names of the author and the date the code was originally .
- A UML class diagram has to be provided. The program must be structured into at least 3 classes, and the OO design has to be meaningful.
- Methods must not be too long (if it is more than 30 statements it is too long)
- Your program **should not make assumptions** where the input/output files are located (the full path will be given in the arguments when the program is invoked).
- Any clarifications will be given on the Discussion board (**not e-mail**).
- **Java applications must be compiled using ANT, and not just in the IDE (e.g. Eclipse)**
- **C# applications must be compiled using a makefile or MSBuild script – not in Visual Studio.**

Marking Scheme

Your assignment will be awarded either 1 or 0.

To be awarded 1 your submission must meet **all** of the following requirements:

1. Software compiles using supplied scripts
    a. Java programs compile using Ant
    b. .NET programs compile using batch file, makefile, or MSBuild script
2. Software is fully functional, all requirements are met and the software is capable of handling errors with data entry appropriately.
    a. All requirements from this document have been met.
    b. All clarifications from the discussion board have been met.
    c. Coding style, comments, overall structure
        i. Class level comments have been provided
        ii. Methods are not too long or too complex
        iii. A consistent coding style has been followed
3. Your software is structured using an appropriate **object-oriented design**. You have provided a UML diagram which corresponds to the design of your program.
    a. The method signatures have been included in the class diagram.
4. Your software produces output exactly as specified.
5. Your software can be run (invoked) exactly as specified.