Department of Computer Science and Information Technology

La Trobe University

CSE1OOF / CSE4OOF Semester 2, 2019
Assignment Part D

Due Date: Tuesday, 22 October 2019, at 10.00 a.m.

First and Final date for SUBMISSION Tuesday, 22 October 2019, at 10.00 a.m.

Delays caused by computer downtime cannot be accepted as a valid reason for a late submission. Students must plan their work to allow for both scheduled and unscheduled downtime.
There are NO EXTENSIONS on this assignment as execution test marking will begin on Wednesday, 23 October – in your normal lab (Week 12)
This is an individual assignment. You are NOT permitted to work as a group when writing this assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Information Technology treats academic misconduct seriously. When it is detected, penalties are strictly imposed. Refer to the unit guide for further information and strategies you can use to avoid a charge of academic misconduct. ***All submissions will be electronically checked for plagiarism.***

Assessment Objectives:
- ◊ to design programs that conform to given specifications
- ◊ to practise combining multiple classes and methods into a whole program
- ◊ to implement programs in Java.

Submission Details: Full instructions on how to submit electronic copies of your source code files from your latcs8 account are given at the end. *If you have not been able to complete a program that compiles and executes containing all functionality, then you should submit a program that compiles and executes with as much functionality as you have completed. (You may comment out code that does not compile.)*

Note you must submit electronic copies of your source code files using the **`submit`** command on latcs8. Ensure you submit all required files, one file at a time. **For example**, the file **`Vector.java`** would be submitted with the command:

```
> submit OOF Vector.java
```

Do **NOT** use the LMS or EMAIL to submit your files, use latcs8 only

PLEASE NOTE*: While you are free to develop the code for this progress check on any operating system, your solution must run on the latcs8 system.*

Marking Scheme: *This assignment is worth 10% of your final mark in this subject.*
Implementation (Execution of code) 100%
You may be required to attend a viva voce (verbal explanation of your code) assessment (to be used as a weighting factor on the final mark).

Return of Mark sheets:
The face to face execution test marking in the lab (Week 12) constitutes a return of the assignment mark.

Please note carefully:

The submit server will close at 10:00 am on Tuesday, 22 October 2019.

After the submit server has closed, NO assignments can be accepted.

Please make sure that you have submitted **all** your assignment files before the submit server closes.

There can be **NO** extensions or exceptions, unless a request is made before the deadline using the following link, together with a medical certificate that certifies a medical condition.
https://www.latrobe.edu.au/students/admin/forms/request-an-extension/request

Your assignment will be marked in your normal lab from Wednesday, 23 October 2019.

You should attend your assigned lab; however, any other lab will be ok. You may have the assignment marked later, as it is marked from the submit server, not from your student account.

All OOF assignments are marked from the submit server, never from a student's account. Please make sure that you submit you assignment on time.

If you need to arrange a time other than your normal lab, please email the lecturer Dr Tian Feng (t.feng@latrobe.edu.au).

Using code not taught in OOF - READ THIS

Please also note carefully that whilst we encourage innovation and exploring java beyond what has been presented in the subject to date, **above all, we encourage understanding**.

The assignment that follows can be solved using techniques that have been presented in lectures, lecture / workshops and labs so far.

These are the techniques and knowledge that we will later be examining in the Real Time Test (20 marks) and the exam (50 marks).

Code and techniques that are outside the material presented will not be examined, of course.

You are free to implement the program below in any way, with one condition.

Any assignment that uses code that is outside what has been presented to this point **must be fully explained at the marking execution test**. Not being able to **fully** explain code outside what has been presented in the subject so far will **result in the assignment being awarded a mark of 0**, regardless of the correctness of the program.

Submitting an assignment with code outside what has been presented so far and not attending the marking execution test will result in an automatic mark of 0, regardless of the correctness of the submission.

Access Modifiers in objects

Unless stated, all attributes must have **`private`** as their access modifier. Any classes that have any non-private object attributes will result in the whole assignment mark being heavily reduced, up to and including, being awarded 0, regardless of the correctness of the program. **This includes omitting access modifiers,** which means that the object attributes have the java default access modifier **`package`**.

Task 1: Vector

In mathematics and physics, a vector is an element of a vector space. If the space is N dimensional, such a vector consists of N scalars. For example, the vector (0, 0) represents the origin point of a two-dimensional Euclidean space.

Task 1 requires you to write a Java class for a simplified vector that handle integer scalars. To enable rapid development of the program, the following Java file is provided:

`Vector.java`

An object of Vector class has the following attributes:

- **values** This is an **integer array** that stores a collection of input integers passed in a constructor**.** Note its access mode is **protected**.
- **size** This is an **integer** that represents the size of *values* attribute, i.e. the number of input integers. Note its access mode is **protected**.

Vector class has the following **two** overloaded constructors:

- The first constructor has **two** parameters: an **integer array** that stores input integers passed by users, and an **integer** that represents a default value. It takes following actions:
    1. If the integer array parameter is **null** or contains less than **two** elements, *values* attribute is initialised of length 2, and both elements inside are set to the default value parameter; otherwise, *values* attribute is initialised of the same length as the integer array parameter, and all elements in the integer array parameter are **copied** to *values* attribute. **Note it is NOT allowed to assign the value of the integer array parameter to *values* attribute.** (Recall that an array is an object, and its corresponding object variable holds a memory address)
    2. *size* attribute is set to the actual length of *integer* attribute.

- The second constructor has **any number of integers** as parameters for *values* attribute. It takes similar actions to what the first constructor does, with the default value parameter of **zero (0**). **Note the second constructor is required to reuse the first constructor**. (Recall what you have learned on the **this** keyword and **varargs**)

Vector class has the following **four** methods:

- **toString**

  This is a **public** overrided method, which has no parameters but returns a String to represent the information about the current object's *values* attribute. For example, if the *values* attribute has elements {2, 2, 2, 2}, the returned String is "[2 2 2 2]".

- **getValues**

  This is a **public** method, which has no parameters but returns an integer array. Specially, it returns a **copy** of all elements in *values* attribute. **Note it is NOT allowed to return values attribute here.** (Recall private leakage)

- **getSize**

This is a **public** method, which has no parameters but returns an integer. Specially, it returns the value of *size* attribute.

- **main**

  This is a **public static** method to test the class and has been provided already. **Note it is NOT allowed to make any change on this method.** If the class is correctly written, it is supposed to output following information:

```
[ 0 0 ]
[ 0 0 ]
[ 1 2 ]
[ 1 1 ]
[ 4 5 6 ]
[ 4 5 6 ]
[ 4 5 6 ]
```

Marking Requirement

Task 1 is worth **40** mark in this assignment. **Any of the following actions will lead to a deduction of 10 mark**:

1. The program does not compile or run after at most **three** minor fixes;
2. The first constructor is **NOT** reused in the second constructor;
3. The second constructor does **NOT** support variable parameters;
4. Any change is made on the main method.

If the mark for Task 1 is **more than 0 mark** after checking the above requirements, the tutor will continue to compile and run your code (if possible) and mark according to the marking sheet.

Task 2: RangedVector

Task 2 requires you to write a Java class for a more complicated vector compared to the previous one. It ensures all elements in *values* attribute in a specific range. To enable rapid development of the program, the following Java file and test data are provided:

```
RangedVector.java
v1.dat
v2.dat
v3.dat
```

RangedVector class **inherits** Vector class in Task 1. (Recall the lectures on inheritance in Week 10, especially the **extends** keyword)

Beside the attributes inherited from Vector class, an object of RangedVector class has the following attributes:

- **lowerbound** This is an **integer** that specifies the smallest value stored in *values* attribute. i.e. any element in values attribute is larger than or equal to it.
- **upperbound** This is an **integer** that specifies the largest value stored in *values* attribute. i.e. any element in values attribute is smaller than or equal to it.

RangedVector class has **one** constructor. It has **three** parameters: an **integer array** that stores input integers passed by users, an **integer** that represents the lowerbound, and another **integer** that represents the upperbound. The constructor takes following actions:

1. It calls the first constructor of bass class, i.e. Vector class, and pass the integer array parameter and the lowerbound integer parameter (as the default value) to that constructor. (Recall the lectures on inheritance in Week 10, especially the **super** keyword)
2. It sets *upperbound* and *lowerbound* attributes to the corresponding parameters. It is **assumed** that lowerbound parameter is always **smaller than or equal to** upperbound parameter.
3. It checks each element in *values* attirbute to ensure if it is in the range of [lowerbound, upperbound] (inclusive). Specifically, if an element is smaller than *lowerbound* attribute, it is set to *lowerbound* attribute; if an element is larger than *upperbound* attribute, it is set to *upperbound* attribute.

RangedVector class has the following **four** methods:

- **getDistance**

  This is a **public** method that has **one** parameter, i.e. another object of RangedVector class, and returns the Euclidean distance between the current vector and the parameter vector **in double**. Specifically, it takes the following actions,

  1. If the parameter object is **null** or has a **difference size of values** from the current one, it returns **-1.0**.

  2. Otherwise, as mentioned above, it calculates the Euclidean distance between the two vector objects. Suppose the current object's values is **x** = {x1, x2, …, xn} and the other's values is y = {y1, y2, …, yn}, it returns a double *d* as follows:

$$d = \sqrt[2]{(x1 - y1)^2 + (x2 - y2)^2 + \cdots + (xn - yn)^2}$$

- **add**

  This is a **public** method that has **one** parameter, i.e. another object of RangedVector class, and returns an object of **Vector class**. Specifically, it takes following actions:

  1. If the parameter object is **null** or has **a different size** to the current object, it returns a **null**.

  2. Otherwise, it adds the *values* attributes of the current object and the parameter object and returns an object of **Vector class** by using its first constructor with the added integer array and the default value of 0 as parameters. For example, if the two RangedArray objects have *values* attributes {1, 2, 3} and {4, 5, 6}, the method creates an object of Vector class with an integer array {5, 7, 9} and 0 as the constructor's parameters and then returns it.

- **toString**

  This is a **public** overrided method, which has no parameters but returns a String to represent the information about the current object. Specifically, it takes following actions:

  1. If *lowerbound* and *upperbound* attributes are same, it calls *toString* method of base class.

  2. Otherwise, it returns a String about the *values* attribute, e.g. {1, 2, 3, 4}, the lowerbound attribute, e.g. 1, and the upperbound attribute, e.g. 4, as
     "[1 2 3 4] in range of [1, 4]"

- **main**

  This is a **public static** method to test the class and has been provided already. **Regarding change on main method, you are ONLY allowed to add code between the two following comments:**

  ```
  //---CHANGE ON main METHOD STARTS HERE

  //---CHANGE ON main METHOD ENDS HERE
  ```

  You are required to add code in the abovementioned area to enable the following actions:

  1. The program takes as command line arguments the paths to **three** .dat files as command arguments. For example:
     ```
     java RangedVector v1.dat v2.dat v3.dat
     ```

  2. If command line arguments are less than or more than three, it prints the following information:
     ```
     Error: The program requires as input 3 .dat files.
     ```

  3. Otherwise, it creates an object of RangedVector class for data read from each .dat file and stores all objects in an RangedVector array named **rv**.

  4. Each .dat file, no matter the one provided for testing or marking, consists **one** line of integers separated by white space. The first integer is lowerbound, the second is upperbound, and the rest are used as values in an object of RangedVector class. For example, v1.dat contains the following line:
     ```
     3 7 1 3 5 7 9
     ```

Hence, the program takes 3 as *lowerbound* parameter, 7 as *upperbound* parameter, and {1, 3, 5, 7, 9} as the interger array parameter to the constructor and then creates an object of RangedVector class.

Any .dat test file for this assignment contains at most 16 integers.

5. If the class is correctly written, it is supposed to output following information with v1.dat, v2.dat and v3.dat as command line arguments:

```
RV 0: [ 3 3 5 7 7 ] in the range of [ 3, 7 ]
RV 1: [ 4 4 6 8 8 ] in the range of [ 4, 8 ]
RV 2: [ 2 2 2 2 ]
--->
Euclidean distance between RV 0 and RV 1: 2.24
Addition of RV 0 and RV 1: [ 7 7 11 15 15 ]
Euclidean distance between RV 0 and RV 2: -1.00
Addition of RV 0 and RV 2: Invalid!
Euclidean distance between RV 1 and RV 2: -1.00
Addition of RV 1 and RV 2: Invalid!
```

Marking Requirement

Task 2 is worth **60** mark in this assignment. **Any of the following actions will lead to a deduction of 15 mark**:

1. The program does not compile or run after at most **three** minor fixes;
2. The constructor does **NOT** call the constructor of base class;
3. *toString* method does **NOT** call *toString* method of base class;
4. Any change is made on the main method **out of the allowed area**.

If the mark for Task 2 is **more than 0 mark** after checking the above requirements, the tutor will continue to compile and run your code (if possible) and mark according to the marking sheet.

How the assignment is marked

Please note that the assignment is marked, face to face, in your lab. You are required to run your program. This means that you need to have code that compiles, runs and displays to the screen. Code that does not compile will achieve a mark of 0. We cannot award marks for non-compiling, non-running or non-displaying code, that is, look through your code and award marks on what might have happened if you could have gotten your code to run and/or display.

It is essential that your code displays to the screen the outcome of the actions that you have coded. Without this display, we have no way of checking that your assignment is in fact meeting the requirements. Please refer to the paragraph above.

The smallest amount of code that compiles, runs and displays an outcome will get more marks than a whole assignment that does not compile, does not run or does not display anything to the screen.

Final Notes: - transferring files between Windows and Unix

Be very careful transferring files from Windows to Unix. If you do transfer a file from Windows to Unix open the file, in Unix, using vi. For example, if you transferred a file named **`b.txt`** from Windows to Unix, open the file in Unix with the command

```
vi -b b.txt
```

you will (might) see a lot of **`^M`**'s at the end of each line.

These MUST be removed using the command shown below or else your input file will have too many newline characters and will not translate properly. That is, your code will not correctly read the input file(s).

Your code will work on Windows but NOT on Unix. Still in vi, in command mode (press the Esc key first) do the following

```
:%s/ctrl-v ctrl-m//g
```

`ctrl-v ctrl-m` means hold down the control key and with the control key down press v then press m.

Final, final notes

Don't let anyone look at your code and definitely don't give anyone a copy of your code. The plagiarism checker will pick up that the assignments are the same and you will both get 0 (it doesn't matter if you can explain all your code).

There will be consultation sessions for the assignment, the times will be posted on LMS. If you have problems come to consultation.

And a final, final, final note, "eat the dragon in little bits". Do a little bit every night; before you know it you will be finished. The assignment is marked with running code, so you are better to have 2 or 3 parts completed that actually compile and run, rather than a whole lot of code that doesn't compile.

The execution test is done on latcs8 so please make sure that your code runs on latcs8 before you submit.