School of Computing & Mathematics
Programming II: Data Structures & Algorithms
CSC-20037 (P2)

Coursework Assessment
List-based Queues with GUI & Data Structure Report

This task is set as part of the formal assessment for CSC-20037 and will count for 100% of the overall module mark. Accordingly, you should take note of the following points:

1. Your work *must* be accessible and readable using the School's computer systems.
2. You may be asked to demonstrate/explain your submitted materials to a designated member of staff, so you should keep any documents safely.
3. This assignment should be **your own work** and may need to be completed in your own time: just like the independent study time that you spend working on items of coursework for your other taught modules back at the University. You should not seek or request assistance from any code-sharing or code-development websites or companies.
4. Of course you are encouraged to refer to, and reuse as appropriate, any of the course materials that you have been provided with or developed earlier in your degree studies. However, if you make use of the published work of other scholars, authors or software developers you should ensure that this usage is clearly stated and properly acknowledged in your submitted material. You are responsible for ensuring that any discussion of ideas with other students does not transgress the guidelines provided in the School Handbook or the University's guidelines. In particular, you should neither supply material to other students, or request it from others. Remember that you are responsible for keeping your work securely.

Help & Guidance
Remember that, while this is an individual element of assessment, you are strongly encouraged to obtain any technical support, help and guidance through the usual channels including: your CSC-20037 lecturers, tutors, teaching assistants and demonstrators. **Absolutely no penalty is incurred by doing so.**

The assignment is in two parts.

Part 1 (70%) involves the design, implementation and testing of a list-based Queues with a suitable graphical user interface (GUI).

Part 2 (30%) requires a short written report that briefly justifies the choice of the most suitable data structure for a given set of problem scenarios. (**Part 2 requires no coding**.)

Part 1 – list-based Queue Application with GUI (70%)

Design and code a Java application that uses a singly-linked list-based implementation of a **Queue** abstract data type (ADT). The application should possess a suitable graphical user interface (GUI) that will allow the end-user to maintain two separate queues of *personal contact* information consisting of:

Q1. a small set of **BirthdayReminder** objects each comprised of the following data items
`{String name, String birthDate}` and

Q2. a short list of **PhoneNoInfo** objects each comprised of the following data items
`{String name, int phoneNumber}`.

Part 1 - Marking Scheme

The application should:

(a)	Provide a singly-linked list-based implementation of a Queue ADT	(20 Marks)
(b)	Allows the user to select between working with either Q1 or Q2	(5 Marks)
(c)	Allows values to be loaded from a specified text file and inserted into the selected data structure	(10 Marks)
(d)	Allows additional values to be inserted and removed through the GUI	(10 Marks)
(e)	Depicts the selected structure graphically (updated after each change)	(20 Marks)
(f)	Displays appropriate messages in the message area of the GUI	(5 Marks)

For each of the mark break-downs (a) to (g) above: 50% of the marks will be available for functionality; 30% for the quality of coding; and 20% for coding presentation and comments. Marks will also be awarded in accordance with the university's generic undergraduate marking scheme.

All of the coding should be your own work save for use of the Java AWT, SWING and I/O libraries. However, you may 'dismantle and/or reassemble', as you see fit, the drawing program constructed during the CSC-20037 practicals, as a starting point for your application.

A few more detailed requirements are presented below:

1. The Queue abstract data type (ADT) should be implemented as a singly-linked data structure in accordance with the ADT implementations discussed in lectures.
2. A toggle button of some form should permit the user to work with either Q1 or Q2.
3. The currently selected data structure's contents should be displayed graphically in the GUI. This does not need to be elaborate. The display should be updated on any change to the contents of the selected data structure.

4. The program should be able to load a file of either **BirthdayReminder** data or **PhoneNoInfo** data from the application directory when requested to do so. In each case the data file will be a text file containing comma delimited, string representations of the set of data items for each data structure entry, one entry per line. (N.B. For marking purposes, two small files, each with 5 lines will be used.) It should be possible to load a file from the GUI: emptying the currently selected data structure first. Failure to find or load the file should be handled gracefully within the program with feedback for the user displayed in a message area.
5. Values entered and removed via the GUI should result in an update of the graphical display when a suitable button is pressed.
6. In addition to displaying the current state of the entire data structure after each end-user gesture/interaction, information should also be clearly displayed to show how many underlying data structure interactions (e.g. the number of list-pointer traversals/pointer-updates) took place in order to accomplish the last end-user interaction. (Whilst not strictly required for the proper operation of the data structure, this ancillary information will enable the end-user to see, first-hand, how the amount of computation required varies with the current size of the data structure for the various operations carried out.)
7. Another button should be provided to allow the emptying of the currently selected queue, item by item, with the values removed displayed within the message area.
8. The message area can be employed to indicate error states and other appropriate messages to the user.

Note that a Java Graphics object has a `setFont()` method to select a font and a `drawString()` to display text. The `getFontMetrics()` method returns a **FontMetrics** instance that can be used to calculate the dimensions of a string to be drawn to the display.

If you fail to implement any of the above GUI elements, you should make the functionality of your data structures clear in some other manner: e.g. by outputting suitable console-directed character-string displays of the Q1 and Q2 contents.

In Part 2 of this coursework (see below) you are asked to produce a short report. As a preamble to that report there should be just a few lines of instructions for the assessors about how to run the software you have developed for Part 1. In this preamble you should state whether or not your program will be providing evidence of the user's interactions via a GUI or just as console-directed character strings. This preamble should also include a small results table that summarises the algorithmic performance of the principal data structure operations under various test conditions: e.g. when the amount of data currently stored is low (perhaps two or three data items) and then when it is significantly larger (perhaps 10 or 12 data items). The information displayed by completing requirement 6., above, should prove useful here.

Part 2 – Data Structure Selection Report (30%)

Real-world objects and their properties can be represented by data structures that can be accessed and manipulated by a computer program. Consider the scenarios listed in below, and some of the important operations that they involve.

(a) A supermarket stock control system needs to keep track of how long a perishable line of produce (e.g. a type of bar-coded fruit or vegetable) has been on the shelves. The stock control system will direct the supermarket's shelf-stacking staff to ensure that they systematically remove the oldest items from the shelves, leaving fresher items to remain on display.

[15%]

(b) A phone company wants to store information about each of its new subscribers using subscriber surname as the key field. They want to be able to add new subscribers quickly and to efficiently access existing subscriber's details. Frequently the company will also need to produce complete phonebook listings sorted by subscriber surname.

[15%]

Explain which of the following data structures you think would be most suitable for use in software written to support each of the above scenarios:

- a priority queue
- a B-tree
- a queue
- a binary search tree

Choose only one data structure for each scenario. Your choice should be justified by briefly discussing how some of the most important data structure operations provide a good fit for each of the problem scenarios described above. In particular, your explanation should make reference to the algorithmic complexity of some of the principal operations that would be carried out using your chosen data structure.

What to hand-in & submit

You should submit a zip-file containing copy of your NetBeans project containing all of your code for Part 1 and a separate PDF version of your coursework report for Part2 (including the *preamble* described above).

As usual, your Part 2 report should be a spell-checked, word-processed document. You may use any word-processing software that is provided in the School's computer laboratories, but remember that the document you upload as your submitted report should be in the form of a pdf. You should submit your zip-file **by 12noon on Monday 3rd December 2018** using the KLE drop-box that has been provided on the CSC-20037 KLE pages.