Tutorial Letter 101/3/2020

Advanced Programming

COS3711

Semesters 1 and 2

Computer Science Department

This tutorial letter contains important information about your module.

BARCODE

CONTENTS

Dear Student

As part of this tutorial letter, we wish to inform you that Unisa has implemented a transformation charter based on five pillars and eight dimensions. In response to this charter, we have also placed curriculum transformation high on the agenda. For your information, curriculum transformation includes the following pillars: student-centred scholarship, the pedagogical renewal of teaching and assessment practices, the scholarship of teaching and learning, and the infusion of African epistemologies and philosophies. These pillars and their principles will be integrated at both the programme and module levels, as a phased-in approach. You will notice the implementation thereof in your modules, and we encourage you to fully embrace these changes during your studies at Unisa.

1 INTRODUCTION

Welcome to the Advanced Programming module (COS3711). We trust that you will find this module stimulating and interesting, and wish you a successful semester. The focus of this module is on advanced object-oriented programming aspects using C++ as the implementation language and the Qt 5 framework for developing object-oriented applications.

Do not hesitate to contact your lecturer (by email or telephone) if you are experiencing problems with the content of this tutorial letter or any aspect of the module. We sincerely hope that you find this module, as well as your online learning experience, interesting and rewarding and trust that you will complete the module successfully.

1.1 Blended module

Please note that this module is offered in a blended format (which means that though all the material will be available online, some material will be printed and some will only be available online). You thus need to use the COS3711 page on *my*Unisa (https://www.unisa.ac.za/sites/myunisa/default/) regularly as part of your studies for this module.

All study material for this module will be available on *my*Unisa. It is thus very important that you register on *my*Unisa and access the module site on a regular basis. You must be registered on *my*Unisa to be able to access your learning material, submit your assignments, gain access to various learning resources, “chat” to your lecturer/e-tutor and fellow students about your studies and the challenges that you might encounter, and to participate in online discussion forums. Importantly, *my*Unisa contains the *Lessons* pages from which you will be able to access the study material for this module.

1.2 Printed materials to support the blended module

Because we want you to be successful in this online module, we also provide you with some of the study materials in printed format as well as in PDF format for downloading from the *Additional Resources* page. This will allow you to read the study materials, even if you are not online. Do not wait for the printed support materials to arrive to start studying.

You can find a copy of the online lessons on *my*Unisa in PDF format in the *Additional Resources* (TL104). While the PDF materials may appear slightly different from the online study materials, they are exactly the same. Note that most tutorial matter will not be printed (such as tutorial letters 102 and 103, and

assignment solutions that are issued as tutorial letters 201 and 202), and these will only be available in Additional Resources.

Please activate your *my*Life email address as well as obtain access to the *my*Unisa module site. Remember that the University will use your *my*Life email account to contact you, and we will also use it to send you important updates to material in COS3711. It may be a good idea to set up forwarding of emails sent to your *my*Life account to your primary email account or some other account that you access regularly.

1.3 e-Tutors

Once you have been registered for this module, you will be allocated to a group of students under the support of an e-tutor who will be your tutorial facilitator. We strongly encourage you to use your e-tutor – do the exercises that they post online, email them when you have problems, and discuss the module content on the e-tutor discussion forums. The point of the e-tutor is to help you, and it would be a pity if you were not to use this valuable resource. Of course, you can still contact the module lecturer if you need to.

We wish you success on your journey!

2 PURPOSE AND OUTCOMES

You can find more detail on the purpose and outcomes of this module in Lesson 0 on *my*Unisa. Also, the assumed background knowledge can be found on the home page of the *my*Unisa site for this module.

3 LECTURER(S) AND CONTACT DETAILS

3.1 Lecturer

You are welcome to contact the COS3711 lecturer. The names and telephone numbers of the lecturers will be supplied in a COSALL tutorial letter, and can also be found on the module site on *my*Unisa. You should also check the home page of the COS3711 site on *my*Unisa to see if there have been any changes to the lecturing staff for this module.

When you contact the lecturers, please do not forget to include your student number and module code. This will help the lecturers to assist you.

3.2 Department

Should you have difficulty in contacting your lecturers, you may phone the general number of the School of Computing at 011 670 9200. Your message will then be conveyed to the relevant lecturer. Remember to provide your student number together with the relevant module code.

3.3 University

Visit http://www.unisa.ac.za and click on the Contact us menu button (top right) to obtain information on how to communicate with the University. Click on the Student enquiries link to obtain a list of contact details

(including e-mails) of various departments in the University. The Contact us page also contains Information about Unisa's regional centres and campus maps and directions.

4 RESOURCES

4.1 Prescribed book

The prescribed book for COS3711 is:
Ezust, A. and Ezust, P. 2012. *An Introduction to Design Patterns in C++ with Qt*. Second edition. Prentice Hall.

You can find more information on the prescribed book on the *Orientation* page on *my*Unisa. Here you will also find information on the two recommended books, as well as how to access some of these, and other Qt books, online.

4.2 Prescribed software

Details of the software prescribed can also be found on the *Orientation* page on *my*Unisa.

The software for this module is open-source. In other words, you are free to download, install, copy and distribute it under the relevant open-source license. Open-source software for modules in the School of Computing is available for download from the school's Osprey server: http://osprey.unisa.ac.za/. From the home page, click on the SoC Registered Students link. Then, if you haven't done so already, select your modules and click on the Submit button. You should then see a link to the Open-source software repository under Resources. For more up-to-date versions, visit https://www.qt.io/download.

4.3 Free computer and internet access

Unisa has entered into partnerships with establishments (referred to as Digital Access Centres) in various locations across South Africa to enable you (as a Unisa student) free access to computers and the Internet. This access enables you to conduct the following academic related activities: registration; online submission of assignments; engaging in e-tutoring activities and signature courses; etc. Please note that any other activity outside of these is for your own cost, for example, printing, photocopying, etc. For more information on the Digital Access Centre nearest to you, please visit www.unisa.ac.za/telecentres.

4.4 Library services and resources information

The Unisa Library offers a range of information services and resources:

- For brief information go to: https://www.unisa.ac.za/library/libatglance.
- For more detailed Library information, go to http://www.unisa.ac.za/sites/corporate/default/Library.
- For research support and services (e.g. personal librarians and literature search services), go to http://www.unisa.ac.za/sites/corporate/default/Library/Library-services/Research-support.

The Library has created numerous Library guides: http://libguides.unisa.ac.za
Recommended guides:

- Request and find library material/download recommended material: http://libguides.unisa.ac.za/request/request.

- Postgraduate information services: http://libguides.unisa.ac.za/request/postgrad.
- Finding and using library resources and tools: http://libguides.unisa.ac.za/Research_skills.
- Frequently asked questions about the Library: http://libguides.unisa.ac.za/ask.
- Services to students living with disabilities: http://libguides.unisa.ac.za/disability.

Important contact information:

- https://libguides.unisa.ac.za/ask – ask a Librarian
- Lib-help@unisa.ac.za – technical problems accessing library online services
- Library-enquiries@unisa.ac.za – general library related queries
- Library-fines@unisa.ac.za – for queries related to library fines and payments

4.5 Student support services

Study@Unisa, available on myUnisa (www.unisa.ac.za/brochures/studies), has all the tips and information you need to succeed at distance learning and, specifically, at Unisa.

5 STUDY PLAN

Tuition Week	Dates	Work to do	Assignments due
1	3-9 Feb *3-9 Aug*	Chapter 7 (Libraries and Design Patterns) Read and work through tutorial letter 102 available on *my*Unisa as well as Lesson 1 of the Learning Units.	
2	10-16 Feb *10-16 Aug*	Chapter 12 (Meta Objects) Refer to the Notes in the Lessons for additional material on Chapter 12.	
3	17-23 Feb *17-23 Aug*	Chapter 13 (Models and Views) Refer to the Notes in the Lessons for additional material on Chapter 13.	
4	24 Feb - 1 Mar *24-30 Aug*	Chapter 14 (Validation and Regex) Refer to the Notes in the Lessons for additional material on Chapter 14.	
5	2-8 Mar *31 Aug - 6 Sep*	Complete assignment 1	Due 9 Mar *4 Sep*
6	9-15 Mar *7-13 Sep*	Chapter 15 (Parsing XML) Refer to the Notes in the Lessons for additional material on Chapter 15.	
7	16-22 Mar *14-20 Sep*	Chapter 16 (More Design Patterns) Refer to the Notes in the Lessons for additional material on Chapter 16.	
8	23-29 Mar *21-27 Sep*	Chapter 17 (Concurrency) Refer to the Notes in the Lessons for additional material on Chapter 17.	
9	30 Mar - 5 Apr *28 Sep - 4 Oct*	Networking and the Web Study tutorial letter 103 which you can find under the Official Study Material.	

10	6-12 Apr *5-11 Oct*	Complete assignment 2	Due 14 Apr † *8 Oct* †
11	13 Apr onwards *12 Oct onwards*	Revision: work through assignments 1 and 2 again, work through all the tutorial letters, extra notes and the textbook, as well as the past exams.	
	4 May *19 Oct*	Exams commence	

† Please note that there will be **no extension of the due date** for assignment 2 (in both semesters) as there will not be sufficient time to mark the assignments before all marks need to be submitted.

The study plan above should help you get through all the work that needs to be covered this semester. Note that the dates in normal text refer to the first semester, and those in italics refer to the second semester.

6 ASSESSMENT

6.1 Assessment plan

Below is a summary of the formal assignments as they occur in each semester. You can find our policy on extensions and plagiarism in the *Orientation* page on *my*Unisa.

Semester	Assignment	Due date	Unique assignment number
1	1	9 March 2020	712697
	2	14 April 2020 †	635926
2	1	4 September 2020	826078
	2	8 October 2020 †	813178

† Please note that there will be **no extension of the due date** for assignment 2 (in both semesters) as there will not be sufficient time to mark the assignments before all marks need to be submitted.

The two assignments each count 50% towards the semester mark. Together they count 20% towards the final module mark. The exam counts the remaining 80% to the final module mark.

The assignment questions can be found in section 7 below.

6.2 Submission of assignments

You should submit your assignments electronically via *my*Unisa. If this is not possible, then you may submit it on a CD via post.

The two assignments each have two parts. Part A should be submitted for marking, but Part B should not. Part B is for self-assessment purposes only (and concepts from it will be included in the exams). Tutorial letters 201 and 202, which will be posted on *my*Unisa, will contain model solutions to both parts of the respective assignments. Electronic solutions will be posted to the Additional Resources page. Both parts

of each assignment are equally important, and we strongly recommend that you tackle all the questions of both parts of all the assignments to gain the full benefit from them.

As the assignments will be marked from running versions of your answers, for assignments 1 and 2 you have to submit a `.zip` file containing all the code (`.h` and `.cpp` files), the project file (`.pro`), and any text or other files (if applicable). The `.zip` file should not contain any `.exe` or `.o` files (so do not include the build-desktop folder), and you can delete the `.pro.user` file as well. If there is additional information that you feel the lecturer needs to take note of, please include a `readme.txt` in the `.zip` file. Please submit each question in a separate folder.

Also, your project will only be tested in the prescribed software of this module. Hence you should make sure that your project runs in the prescribed software. Ensure that your submission is virus free!

When marking your assignments, your `.zip` file will be unzipped, the project will be built and the functionality of your application will be tested. Note that there are often marks awarded for submitting an assignment answer that builds and runs. Thus, even if you cannot complete all sections of a question, make sure that you have done what you can that still results in a running version of the program (even if it has limited functionality). No marks will be awarded for a project that does not build successfully.

Electronic submission (preferred)
1. Create a `.zip` file of your assignment submission (as explained above).
2. Upload your single `.zip` file on *my*Unisa.

Postal or assignment-box submission
1. Create a CD with your assignment submission (in the format explained above) and place it in an assignment cover included in your study package.
2. Place your assignment in an envelope and submit it using the postal mail or one of Unisa's assignment mailboxes.

7 ASSIGNMENT QUESTIONS

7.1 Semester 1 – Assignment 1

Chapters 7, 12, 13, and 14 are covered in this assignment.

PART A (To be submitted)

Question 1
For this question you need to ensure that you design the appropriate classes needed to address the specification below.

Create a console application to handle vehicle details. Considering the required functionality that is required and OOP design principles (avoiding anti-patterns and having minimal redundant code), create and implement the appropriate classes necessary to achieve the following.

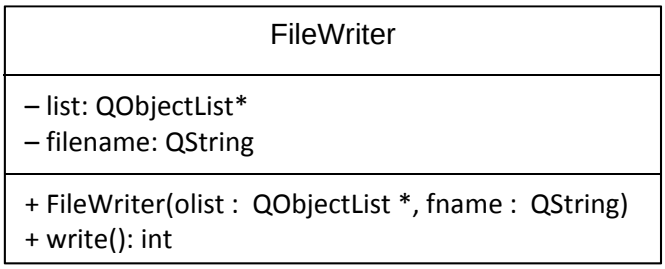
- A vehicle needs to have a model and a year (which should be a reasonable value). There are basically 2 types of vehicles: passenger vehicles (that can carry a specified number of passengers) and

transport vehicles (that have a set carrying capacity expressed in kilograms). Vehicles that are created without the necessary details should have default values.
- Ensure all appropriate getters and setters are included.
- Include a function that can output the details of the vehicle. toString()
- Use Qt's parent-child facility to implement a list of vehicles. QList()
- Test your solution by creating some passenger and transport vehicles (at least one of which is created using the default constructor), adding them to the list, and then outputting the values in the list to the console.

Question 2

This question tests the concepts of reflection and meta-objects.

Extend the application you wrote in question 1 by adding a class named `FileWriter` as described in the UML class diagram below:



The `write()` function should use the meta-object of the items in `list` to write each item's type of vehicle, properties and their values to a text file named in `filename`. You should use reflective programming techniques to do this, and should not use the getters or the `toString()` functions to extract the type of vehicle and the data from each item in the list. Thus you cannot assume that you know what properties an item holds. The function should return the number of items that were written to file.

From `main()`, create an instance of the `FileWriter` class, passing the list of vehicles (as a `QObjectList`) and a file name, and then write the list to file. Display the number of records printed to the console.

Question 3

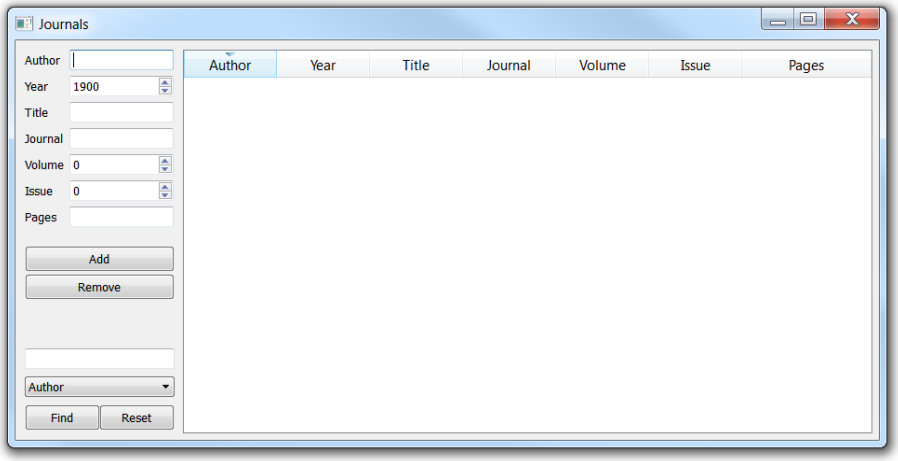
This question focuses on model-view programming.

A reference for a journal article needs the following data: the author, year of publication, article title, journal name, volume and issue numbers, and the pages on which the article can be found. Using a `QStandardItemModel` and an appropriate view, create a database for journal articles. The user should be able to do the following.
- Sort on any column in view.
- Add data to the database.
- Remove data from the database.
- Filter the database on any of the fields (except the page numbers). The user should be able to provide a wildcard filter and select which field to filter the database on. Then only fields that meet the requirements should be displayed. There should also be an option to clear the filter so that all records are displayed again.

Additionally, a year value later than the current year should not be allowed (and you can decide how you will deal with situations where this is attempted). Also, if a reference is older than 10 years, highlight the row in red; if within the last 5 years, highlight in green. Note that you cannot assume that the current year is 2020.

Note further that if the year should be changed in the view, it should not be allowed to be later than the current year, and the highlight colour of the row should still change appropriately.

Below is an example of a possible interface.



Question 4

This question looks at validating user input.

Extend the application developed in question 3 so that only valid input is allowed in the cases where strings have been used to input data. Remember that such validation should take place both when data is input and when it is edited.

Note the following.

- An author's name could be names like "RJ Nkomo-Smythe". You should assume that no full stops will be used with author initials, and that no other special characters should be allowed.
- Assume (for this exercise) that article and journal titles can allow letters of the alphabet ("Journal", for example), numbers, and spaces. However, do not allow a word in the title to contain both alphabetic characters and numbers in the same word – that is, 3D should not be allowed.
- The page numbers should be of the form "12-15" or "121 - 155".

PART B (For self-assessment; not to be submitted)

Question 5

What is the difference between a library and a framework?

Question 6

Create a simple console application that instantiates a `Blank` object. This `Blank` object has no data members or member functions. Then add two dynamic properties to this instance (for example, colour:

blue and size:12.12). Finally, query this instance for its dynamic properties and display the property name and value to the console.

Question 7

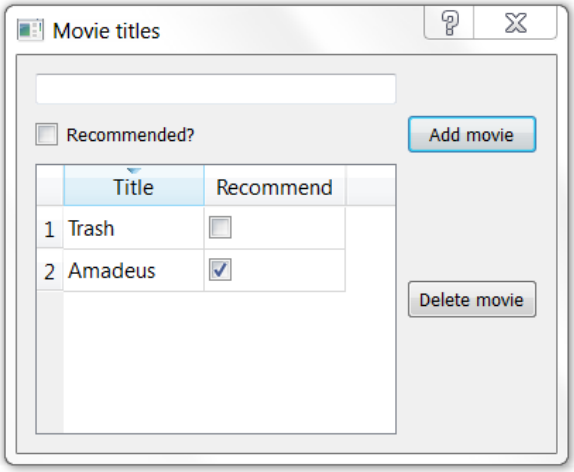
Write a GUI application that makes use of Qt's item view convenience class, `QTableWidget`, instead of the full model and view classes, to manage a list of recommendations of movie titles.

The application should support the following.

- The user should be able to add a (non-empty) movie title via a text entry field and use a check box to indicate if the movie is recommended or not. The movie should only be added to the list if it is not already in the list. Warn the user if data is not inserted into the model.
- The user should be able to delete a movie that is selected in the view (ensuring that the user does want to delete the title).
- The list of move titles is maintained in alphabetical order. This should be true when data is edited in the view as well.

Ensure that you understand the different approaches (convenience class versus model-view classes) and when/why you would use one or other approach.

Below is an example interface.



Question 8

Write regular expressions to accept the following date formats:

- YYYY-MM-DD
- YYYY/MM/DD
- YYYY MM DD

Write a simple graphical user interface application to test the regular expressions using an appropriate input widget. Note that the first separator character should be the same as the second one.

7.2 Semester 1 – Assignment 2

Chapters 15, 16, 17, and TL103 are covered in this assignment.

Please note that there can be **no extension of the due date** for this assignment as there will not be sufficient time to mark the assignments before all marks need to be submitted.

PART A (To be submitted)

Question 1

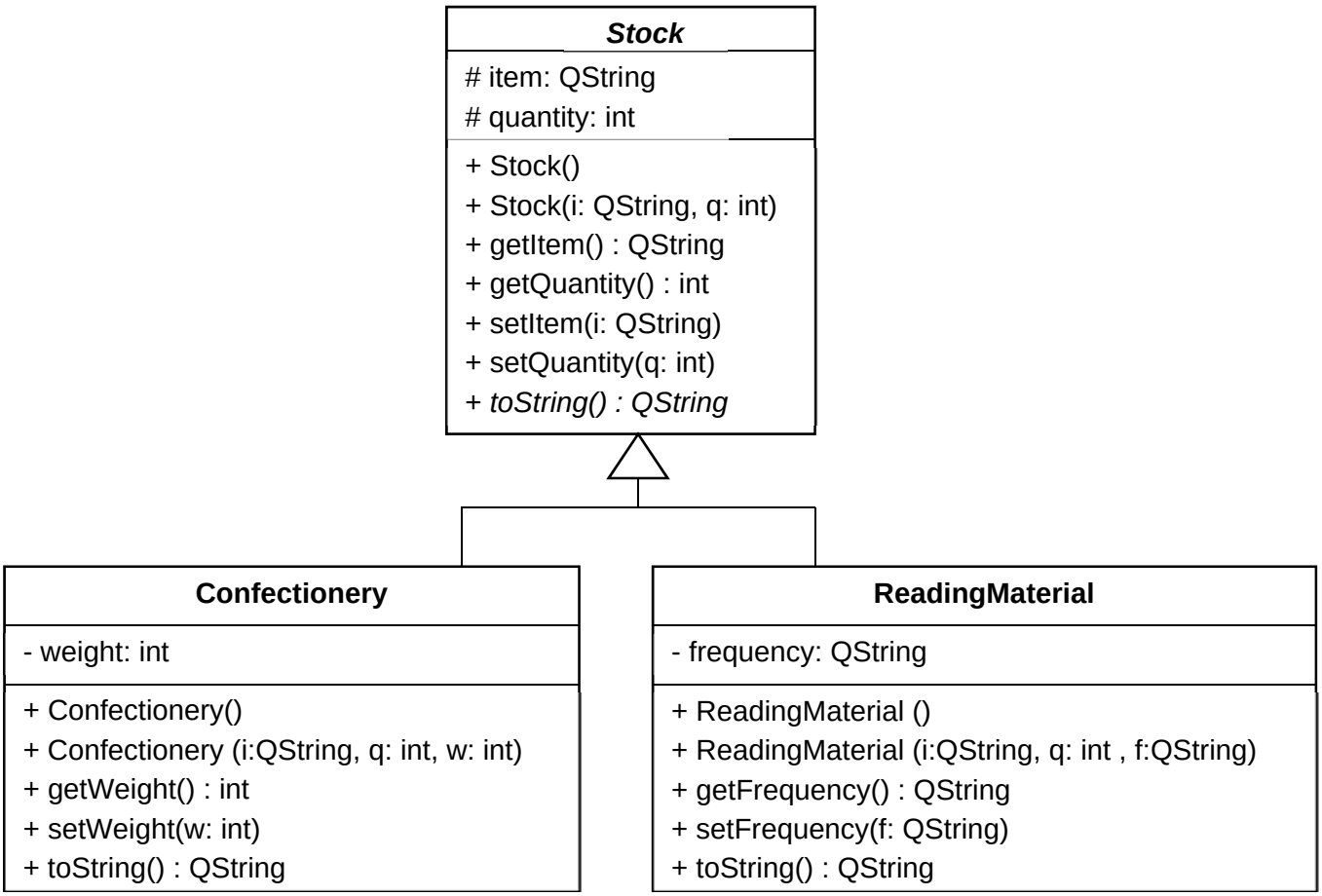
Write a console application that implements the classes that will be used to keep stock for a small corner store that sells only confectionery and reading material. For each item you will track 2 things: what it is, and how many items you have in stock. The derived classes will add one appropriate data member of their own. Use the UML diagram below to assist you. Note that you should not be able to instantiate the `Stock` class, that is, you should not be able to create an object/instance of the `Stock` class.

Use the Factory Method design pattern to return a pointer to either a `Confectionery` or `ReadingMaterial` object. The factory method should take a `QString` as an argument indicating what type of item is required, as well as the data used to construct the specific object. Note that the concrete factory should have only one `createStock()` method; however, the objects that it has to create, although they both have three arguments, are made up of different data types. Solve this problem in the simplest possible way.

You should then be able to do something similar to the following (this is not entirely correct as the two signatures of the `createStock()` method differ – they should be the same).

```
StockFactory sf;
Stock* c1 = sf.createStock("Confectionery", "Kit Kat", 12, 65);
Stock* rm1 = sf.createStock("ReadingMaterial", "The Star", 100, "Daily");
```

Create two or three objects for each of the `Confectionery` and `ReadingMaterial` classes and display these objects.



Question 2

Implement a stock list based on a Singleton design pattern. The only instance of the stock list should have two lists: one for confectionery, and one for reading material. There should only be one `addStock()` function (which would need to check what is being added so that it knows which list to add the item to – achieve this using `QMetaObject`).

You should thus be able to do the following.

```
StockList* stockList = StockList::getInstance();
stockList->addStock(c1);
stockList->addStock(rm1);
```

If you then do the following

```
QList<Stock*>* cList = stockList->returnList("Confectionery");
for (int i=0; i<cList->size(); i++)
    cout << cList->at(i)->toString() << endl;
```

you should get a list of the confectionery items that you have added to the list using `addStock()`.

Write code to test the requirements stated in this question.

Question 3

Use `QXMLStreamWriter` to write the stock list implemented in Question 2 to an XML file. Use a single file to write both the lists managed by the stock list.

The XML should be written in the following format.

```
<stockList>
    <stockItem type="Confectionery">
        <item>Kit Kat</item>
        <quantity>12</quantity>
        <weight>65</weight>
    </stockItem>
    <stockItem type="ReadingMaterial">
        <item>The Star</item>
        <quantity>100</quantity>
        <frequency>Daily</frequency>
    </stockItem>
</stockList>
```

Question 4

Extend Question 2 to allow an XML file to be read in using SAX. Use the same file that was output from Question 3 as the input to Question 4.

This program should parse the file and add all stock items read in to the Singleton stock list. Test that the data has been read in correctly by displaying both `Confectionery` and `ReadingMaterial` lists to the console.

PART B (For self-assessment; not to be submitted)

Question 5

Write a console program `LottoNumberGenerator` that generates a random set of six numbers from the number set 1 to 49.

Using the Qt class `QProcess`, create another program (console or GUI) that will call `LottoNumberGenerator` as a child process and display the six numbers returned by `LottoNumberGenerator`.

Question 6

What similarities and differences are there between the Serializer and Memento patterns?

Question 7

Write a class named `Counter`, capable of running in a thread, to implement counting. The class `Counter` should allow four data members; to specify the start and end numbers, increment value and delay between increments.

Write a graphical user interface (GUI) application, which allows the user to run at most two `Counter` threads simultaneously. For each thread there should be a facility on the interface to enter the beginning and end numbers, increment values and the desired delay. The increments calculated by the threads should be displayed on the interface.

You can design the graphical user interface as you wish as long as it satisfies the above requirements. Note that you have to implement this solution without using `QtConcurrent`.

Question 8
Describe how `QString` employs a Façade design pattern.

7.3 Semester 2 – Assignment 1

Chapters 7, 12, 13, and 14 are covered in this assignment.

PART A (To be submitted)

Question 1
For this question you need to ensure that you design the appropriate classes needed to address the specification below.

Create a console application to handle graph details. Considering the required functionality that is required and OOP design principles (avoiding anti-patterns and having minimal redundant code), create and implement the appropriate classes necessary to achieve the following.

- A node in the graph would be made up of a label (the name of the node), an (x, y) coordinate indicating its position, and an indication of its shape. Provide all the necessary getters and setters.
- The application should include a list of nodes.
- The application should also have a list of paths where the user can add paths between labelled nodes indicating the type of path (for example, `--`, `->`, `<->`, or `<-`). It is clearly necessary to check that the labels exist before adding a path between them.
- The application should be able to output to the console a list of paths in the following format:
  `a [Square at (1, 1)] -> b [Circle at (5, 5)]`
- Test your solution by creating some nodes, adding them to the list of nodes, and then adding some paths (some acceptable, some not). Finally, display the path list.

Question 2
This question focusses on the concepts of reflection and meta-objects.

Extend the application you wrote in question 1 by fully serialising the node and path lists. Note the following requirements.
- Both writer and reader parts of the Serializer should be implemented, serialising to a single file.
- Any text-based file format may be used.
- When writing, use reflection and the meta-objects as far as possible.
- When reading, the various lists should re-created and the data displayed again (to confirm that the data was successfully serialised).

Question 3
This question focusses on the use of Qt's model-view approach to programming.

Write a file system browser that allows a user to do the following:
- Use the `QFileSystemModel` to model the directory structure on your computer.
- Browse the file/folder structure (using a `QTreeView`)
- Display the following information on any file/folder clicked on: name, path, size (for files), and whether it is a file or a folder.

- Create a folder (using a ‘New folder’ button) as a sub-directory in the selected directory
- Rename an existing file/directory (using the inherent ability of the model class)
- Delete a file/directory (using a ‘Delete’ button) – ensure that you check whether the file/folder should be deleted, and also ensure that the information displayed remains current once the file/folder has been deleted.

Question 4
This question tests the use of validation techniques.

Rewrite the `InputForm` class in Chapter 14 to include an additional `QLineEdit` to enter a student number. A `QRegExpValidator` should be attached to this `QLineEdit` so that it only allows a user to enter a single, uppercase alphabetic character, followed by a 4 or 5 digit number that does not begin with a 0. The result of the computation should be displayed in a `QMessageBox` with the student number (Example: ‘Total Pay for the student S5666 is 183.75’) instead of displaying the message on the input interface.

PART B (For self-assessment; not to be submitted)

Question 5
Discuss how the classic MVC pattern differs from the way in which Qt implements model-view programming. Which would you consider to be the better approach?

Question 6
Rework question 2 by adding dynamic properties to some (not all) of the Nodes that will be added to the list. These dynamic properties need to be written to the file along with the rest of a Node’s properties.

Note that these properties do not need to be displayed to the console when the paths are displayed, and that you can remove the functionality to read the file again to recreate the lists.

Question 7
Use Qt’s model/view framework to store and display music CD information. The following information should be stored in the model (a `QStandardItemModel`):

- composer,
- album name,
- replacement value, and
- a rating (out of 100).

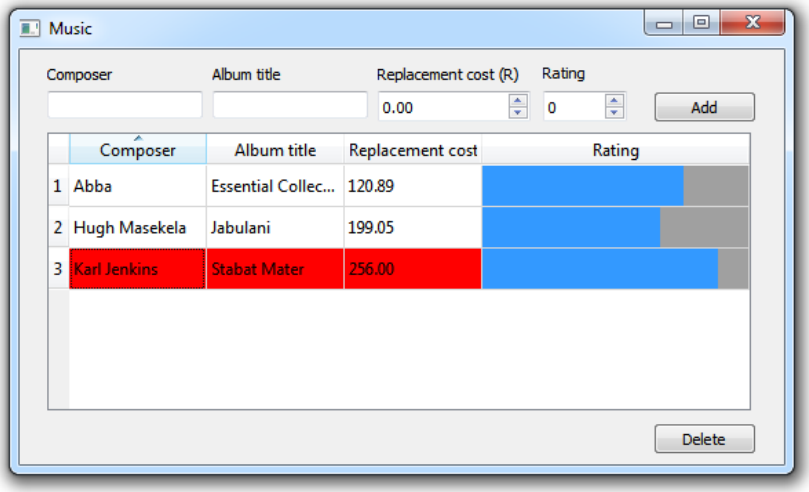
The information should be displayed in a table.

The following functionality should be included.

- There should be a header row.
- The user should be able to add rows of data to the model.
- The replacement value should always display the cents (even as .00).
- Use a delegate to display the rating as a horizontal bar. This column should take up all the remaining space available.
- The user should be able to sort the data by clicking on the column header on which the sorting should be implemented.

- Where the replacement value is greater than or equal to R200, the row should be made a different colour. Note that this colour should also change if the user edits a replacement value (and it should return to the normal colour if the value drops below R200).
- The user should be able to delete a row of data.

Here is an example of the interface.



Question 8

Write an application that can check passwords (using regular expressions) and indicate if they are acceptable or not acceptable according to the following rules.

- It must be at least 5 characters in length.
- It must contain at least one capital/uppercase letter.
- It must contain at least one lowercase letter.
- It must contain at least one number/digit.
- It may not contain any repeating characters (that is, “ele” is acceptable, but “eel” is not).

The application should indicate as you type whether the text provided is acceptable, and should not rely on the user having to press a button to check acceptability.

7.4 Semester 2 – Assignment 2

Chapters 15, 16, 17, and TL103 are covered in this assignment.

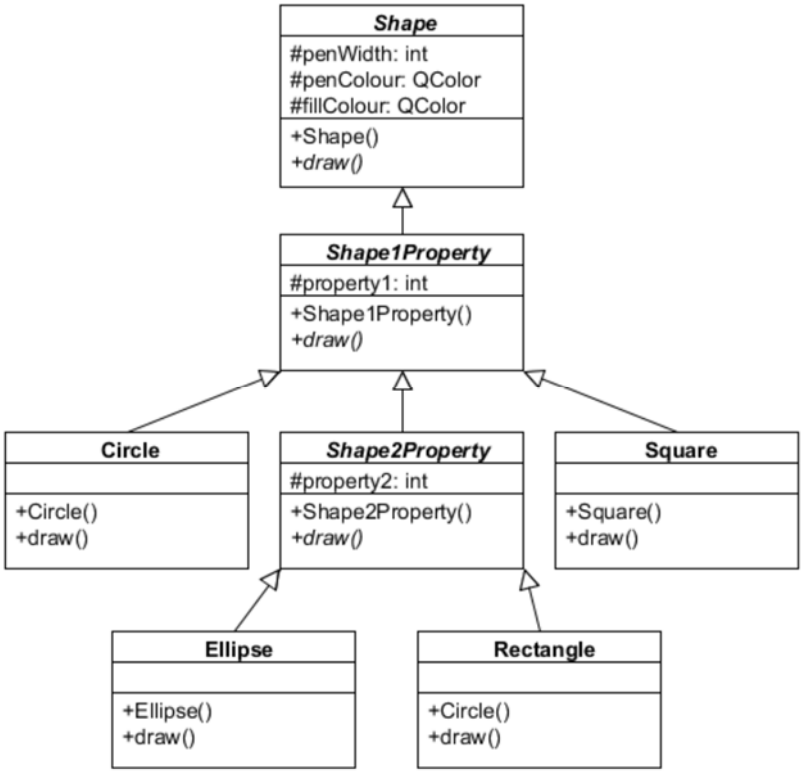
Please note that there can be **no extension of the due date** for this assignment as there will not be sufficient time to mark the assignments before all marks need to be submitted.

PART A (To be submitted)

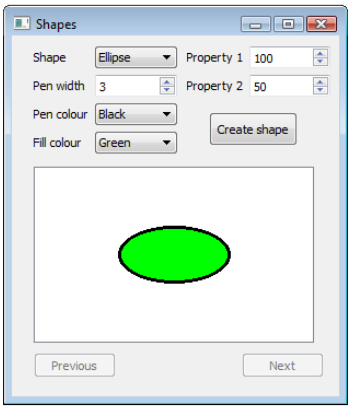
Question 1

One way to draw a set of shapes (squares, circles, rectangles, and ellipses) is to use an inheritance hierarchy similar to the one below. The base class holds basic pen and brush values, and subclasses hold a number of additional properties (squares and circles need one property – the radius of a circle for example, whereas rectangles and ellipses need an additional property – length and width of the two sides of a rectangle, for example). The `Shape`, `Shape1Property` (for circles and squares), and

`Shape2Property` (for ellipses and rectangles) classes are abstract classes (the `draw()` function being the pure virtual function in all of them).



Use this structure to create an application that will draw the required shape in a GUI window. The UML diagram gives only the basic structure, and you will need to add getter and setters, other constructors, and helper functions as you need them.



Note that if you cannot get the image to appear, you should at least display the shape properties in a GUI window so that the following three questions can also be completed.

Question 2

Extend the application in Question 1 by adding a class that will maintain a list of shapes. You will want only one list in the application, so use an appropriate design pattern to ensure this.

As items are added to the list, allow the user to move backwards and forwards through the list.

Question 3
Using the XML file below, import the shapes contained in it when the application starts up (using DOM), and display the first one. You should then be able to navigate through the 4 shapes and add new ones to the list.

```
<shapeList>
   <shape type="Square" pw="1" pc="Red" fc="Black" p1="110" p2="" />
   <shape type="Circle" pw="2" pc="Green" fc="Blue" p1="75" p2="" />
   <shape type="Ellipse" pw="3" pc="Black" fc="Red" p1="140" p2="55" />
   <shape type="Rectangle" pw="4" pc="Blue" fc="Green" p1="75" p2="120" />
</shapeList>
```

Ensure that all necessary file checks are implemented.

Question 4
Extend the application in Question 3 using a Memento pattern to enable a backup/restore facility. Allow the user to keep the state of the application at a user-indicated point, and return the state of the application to that point (again, when the user chooses), and display the first shape in the list. You need only allow for a single backup point.

PART B (For self-assessment; not to be submitted)

Question 5
Write a console program `LottoNumbers` that generates a random set of six numbers from the number set 1 to 49.

Using the Qt class `QProcess`, create another program that will call `LottoNumbers` as a child process and display the six numbers returned by `LottoNumbers`.

Question 6
What similarities and differences are there between the Serializer and Memento patterns?

Question 7
Write a class named `Counter`, capable of running in a thread, to implement counting. The class `Counter` should allow four data members; to specify the start and end numbers, increment value and delay between increments.

Write a graphical user interface (GUI) application, which allows the user to run at most two `Counter` threads simultaneously. For each thread there should be a facility on the interface to enter the beginning and end numbers, increment values and the desired delay. The increments calculated by the threads should be displayed on the interface.

You can design the graphical user interface as you wish as long as it satisfies the above requirements. Note that you have to implement this solution without using `QtConcurrent`.

Question 8
Describe how `QString` employs a Façade design pattern.

8 IN CLOSING

Do not hesitate to contact your lecturer by email if you are experiencing problems with the content of this tutorial letter or any aspect of the module.

We wish you a fascinating and satisfying journey through the learning material and trust that you will complete the module successfully.

Enjoy the journey!
COS3711 lecturers