Tutorial Letter 101/3/2019

Advanced Programming
COS3711

Semesters 1 and 2

School of Computing

This tutorial letter contains important information
about your module.

BARCODE

Define tomorrow.

UNISA university of south africa

CONTENTS

1 INTRODUCTION

Dear Student,

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.

Because this is a blended module (that is, it has some printed and some online material), you need to use myUnisa to study and complete the learning activities for this module. You need to visit the website on myUnisa for COS3711 frequently. The website for your module is COS3711-19-S1/S2.

1.1 To get started…

Because some of the learning material is online, you need to go online to see your study materials and read what to do for the module. Go to the website here: https://my.unisa.ac.za and login with your student number and password. You will see COS3711-19-S1/S2 in the row of modules in the orange blocks across the top of the webpage. Remember to check in the -more- tab if you cannot find it in the orange blocks. Click on the module you want to open.

1.2 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 be available only online).

All study material for this module will be available on myUnisa. It is thus very important that you register on myUnisa and access the module site on a regular basis. You must be registered on myUnisa 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, myUnisa contains the **Learning Units** tool from which you will only be able to access the study material for this module if you have registered and have access to myUnisa.

1.3 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.

You can find a copy of the online study materials from myUnisa in PDF format in the Additional Resources (MO001). 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). Remember, the PDF support materials are a back-up to everything that is found online, on myUnisa. There are no extra things there**. In other words, you should NOT wait for the printed support materials to arrive to start studying.**

Please activate your myLife email address as well as obtain access to the myUnisa module site. Remember that the University will use your myLife 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 myLife account to your primary email account or some other account that you access regularly.

1.4 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 Unit 0 in the Learning Units on myUnisa. Also, the assumed background knowledge can be found on the home page of the myUnisa site for this module.

3 LECTURER(S) AND CONTACT DETAILS

3.1 Lecturer(s)

You are welcome to contact a 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 myUnisa. You should also check the home page of the COS3711 site on myUnisa 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 www.unisa.ac.za and follow the link Contact us to obtain information on how to communicate with the University. Information about Unisa's regional centres is also listed here. Visit http://www.unisa.ac.za/sites/corporate/default/Contact-us and click on the link Student enquiries to obtain a list of contact details (including e-mails) of various departments in the University.

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 4*. Second edition. Prentice Hall.

You can find more information on the prescribed book in Unit 0 of the Learning Units on myUnisa. 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 in Unit 0 of the Learning Units on myUnisa. You can get a copy of it on the Osprey web server of the School of Computing at http://osprey.unisa.ac.za/download/Disk/.

4.3 Free computer and internet access

Unisa has entered into partnerships with establishments (referred to as Telecentres) 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 Telecentre nearest to you, please visit www.unisa.ac.za/telecentres.

5 STUDY PLAN

Tuition Week	Dates	Work to do	Assignments due
1	28 Jan - 3 Feb *15-21 Jul*	Chapter 7 (Libraries and Design Patterns) Read and work through tutorial letter 102 available on *my*Unisa as well as Unit 1 of the Learning Units.	
2	4-10 Feb *22-28 Jul*	Chapter 12 (Meta Objects) Refer to the Notes in the Learning Units for additional material on Chapter 12.	
3	11-17 Feb *29 Jul - 4 Aug*	Chapter 13 (Models and Views) Refer to the Notes in the Learning Units for additional material on Chapter 13.	
4	18-24 Feb *5-11 Aug*	Chapter 14 (Validation and Regex) Refer to the Notes in the Learning Units for additional material on Chapter 14.	
5	25 Feb - 3 Mar *12-18 Aug*	Complete assignment 1	Due 4 Mar *19 Aug* †
6	4 -10 Mar *19-25 Aug*	Chapter 15 (Parsing XML) Refer to the Notes in the Learning Units for additional material on Chapter 15.	
7	11-17 Mar *26 Aug - 1 Sep*	Chapter 16 (More Design Patterns) Refer to the Notes in the Learning Units for additional material on Chapter 16.	
8	18-24 Mar *2-8 Sep*	Chapter 17 (Concurrency) Refer to the Notes in the Learning Units for additional material on Chapter 17.	
9	25 Mar - 31 Mar *9-15 Sep*	Networking and the Web Study tutorial letter 103 which you can find under the Official Study Material.	
10	1-7 Apr *16-22 Sep*	Complete assignment 2	Due 8 Apr *23 Sep* †
11	8-14 Apr *23-29 Sep*	Check comments from assignments 1 and 2 as well as tutorial letters 201 and 202.	
12	15 Apr onwards *30 Sept 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.	
	May *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 Unit 0 of the Learning Units on myUnisa.

Semester	Assignment	Due date	Unique assignment number	% Contribution towards semester mark
1	1	4 March 2019	788288	50
	2	8 April 2019 [†]	730071	50
2	1	19 August 2019	657169	50
	2	23 September 2019 [†]	845108	50

[†] 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 semester and exam marks contribute 20% and 80% towards the final mark.

The assignment questions can be found in section 7 below.

6.2 Submission of assignments

You should submit your assignments electronically via myUnisa. 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 may be included in the exams). Tutorial letters 201 and 202, which will be posted on myUnisa, 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

SEMESTER 1

Semester 1 – Assignment 1

Chapters 7, 12, 13, and 14 are covered in this assignment.
Total marks: 100

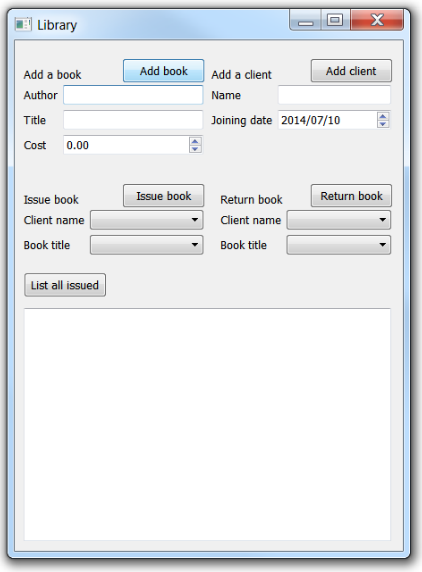
PART A (To be submitted)

Question 1 (33 marks)

Create an application to handle a simple library. Considering the required functionality that is required, and OOP design principles (avoiding anti-patterns), create the appropriate classes necessary to achieve the following:

- The user should be able to add books to a list of books in the library. Each book should have an author, title, and cost.
- The user should be able to add clients to a list of clients of the library. Each client should have a name and a joining date.
- The user should be able to issue books to clients. There should be combo boxes that provide the user with an alphabetically sorted list of clients and an alphabetically sorted list of books to choose from (and these should be updated automatically when a book or client is added). You can assume that there are multiple copies of books, and there is no need to remove a book from the list once it has been issued to someone. You can also assume that there is no limit to the number of books that may be issued to a client.
- The application should maintain a list of issued books (storing just the client name and book title). Use a map container for this. Note that the application should allow a client to take out several books.
- The user should be able to return books. There should be combo boxes to display which clients (sorted alphabetically) have taken out books, and which books (also sorted alphabetically) have been issued to the chosen client. These combo boxes should be updated automatically as books are issued and returned. Note that if a client has taken out more than one book, his or her name should appear only once in the combo box list.
- The user should be able to view a list of all books issued.

See below for an example of the interface:



Joining date 2014/07/10
Cost 0.00

Question 2 (33 marks)

Extend the application you wrote in Question 1, by adding functionality that allows the client and book lists to be written to file (in any format); allow the user to choose which list should be written to file. You should use reflective programming techniques to do this, and should aim to have one function that receives either a book or a client, and is able to determine its properties so that they can be written to file. Thus, you cannot assume that you know what properties a book or client may have.

Question 3 (34 marks)

Write an application that will manage a list of registration numbers for South African vehicles.

3.1 You need to use a model view programming approach, subclassing your model from `QAbstractListModel` and using an appropriate view class. The user should be able to add registration numbers to the list, as well as edit or delete a selected registration number.

3.2 Further, use a subclass of QValidator to ensure that only one of the following registration number formats is accepted (see http://en.wikipedia.org/wiki/ Vehicle_registration_plates_of_South_Africa for more information). Note that all alphabetic characters are in uppercase.

- Three letters, three numbers, and a province indicator (e.g. GP, MP, EC, FS, NC, NW, L). No vowels are allowed.
- Newer GP numbers taking the form two letters, two numbers, two letters, followed by GP. Again, no vowels are allowed.
- Western Cape and KwaZulu-Natal numbers that take the form C or N (respectively) followed by one or two further letters and then a number in the range 1 to 999999.
- Personalised plates that are made up of A-Z or 0-9, and have at least one character, with a further 5 optional characters, followed by one of the provincial indicators (GP, MP, EC, FS, NC, NW, L, WP, ZN).

Try to keep the validation rules to a minimum. Also, make sure that if the registration numbers are edited in the view, that only acceptable ones are used.

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

Question 4

Extend Question 2 so that the full Serializer design pattern is implemented. This means that you have to add to your application the ability to read data into the application. You may assume that you know the structure of the files that hold the book and client information.

Question 5

Extend the application in Question 2, so that a dynamic property can also be added to each client in the list. This means that 2 clients can have different properties added, or none at all. When the client list is written to file, this property name and value should also be written.

Question 6

Using Qt's `QStandardItemModel` and an appropriate view, store and display book information. The following information should be stored in the model:

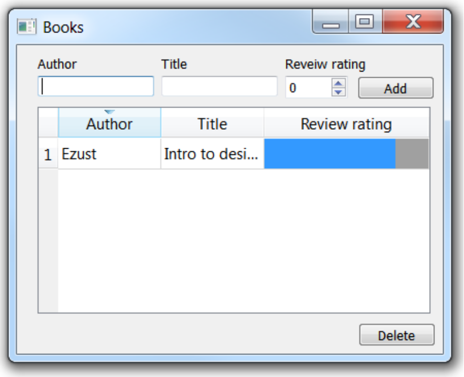
- author,
- title, and
- a review rating (out of 10).

Use a table to display the information.

The following functionality should be included.

- The view should have a header row.
- The user should be able to add data to the model.
- The rating should be displayed as a delegate (displayed 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.
- The user should be able to delete a row of data.

Here is an example of the interface:



	Author	Title	Review rating
1	Ezust	Intro to desi...	

Question 7

Write an application (that uses regular expressions) that will anonymise debit and credit card numbers (as can be seen in the transaction slips that you get when you use a debit or credit card to pay for goods). The application should show both the original text (in one window) and the text where the numbers have been anonymised (in a second window) – this so that you can check that the process has worked correctly. Note the following:

- The user should be able to select the file that is to be processed, via a standard open file dialog.
- Only 16-digit card numbers need to be processed.
- The credit card numbers could come in one of two formats:

  - 1111 1111 1111 1111 (16 digits in 4 groups of 4 digits), or
  - 1111111111111111 (one group of 16 digits with no spaces)
- The credit card number should be anonymised by replacing the middle 8 character with asterisks:
  - 1111 **** **** 1111, or
  - 1111********1111.

Semester 1 – Assignment 2

Chapters 15, 16, 17, and TI103 are covered in this assignment.
Total marks: 100

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 (30 marks)

Study the `GuestHouse` project, a console application provided on *my*Unisa under Additional Resources.

Modify `GuestHouse` so that it becomes a graphical user interface (GUI) application, where a user can add a booking, check whether there is a vacancy in the guest house during a specified time period and find out how many rooms are available on a given date. Thus the user interface should have appropriate widgets to enable the user to achieve these tasks.

In order to support serialization of bookings, expand the `GuestHouse` GUI project so that the list of `Booking`s managed by `BookingList` can be written to a file in an XML format and a `BookingList` object can be re-created with a list of `Booking`s read from an XML file. DOM should be used for creating an XML format of `BookingList` and SAX should be used to parse an XML file of bookings to create an instance of `BookingList`.

Given below is an XML format that can be used for saving a `BookingList` object. The XML format should have the same number of elements for `booking` and attributes for `booking`, `contact`, `guest` but the order of the elements and attributes can be different from the format given below:

```
<bookinglist>
 <booking type="sharing">
  <contact telnumber="082 280 2882" name="A Balone"
  email="ab@agent.com"/>
  <arrivaldate>2015/07/08</arrivaldate>
  <departuredate>2015/07/12</departuredate>
  <guest telnumber="073 527 3492" name="B Eater"
  email="be@holliday.com"/>
  <guest telnumber="012 345 6789" name="C Locanth"
  email="cl@holliday.com"/>
 </booking>
 <booking type="single">
  <contactperson telnumber="082 280 2882" name="A Balone"
  email="ab@agent.com"/>
  <arrivaldate>2015/07/08</arrivaldate>
  <departuredate>2015/07/12</departuredate>
  <guest telnumber="082 280 2882" name="A Balone"
  email="ab@agent.com"/>
</bookinglist>
```

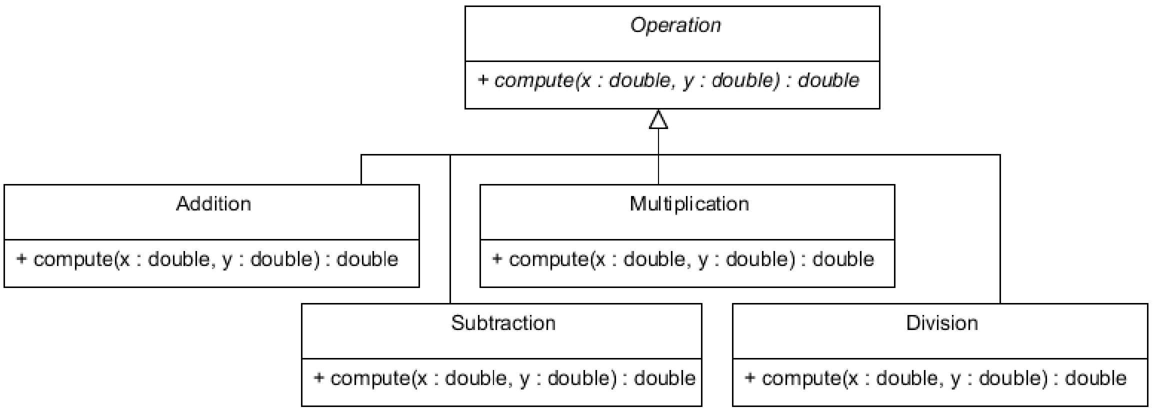
The `BookingList` object should be saved automatically when the GUI is closed, to a file chosen by the user. Similarly when the application starts it should give the user an option to read from an XML file. Make use of the standard file dialog to assist the user in selecting files.

You are allowed to modify the classes in the given project. You should also be adding new classes to answer this question. You are expected to follow good programming design and principles. You can decide on the design of the GUI and it can be done manually or using Qt Designer.

Question 2 (20 marks)

In this exercise, you will design a simple calculator with four basic operations, namely addition, subtraction, multiplication and division. Implementation of this simple calculator should separate the code for user interface and mathematical operations.

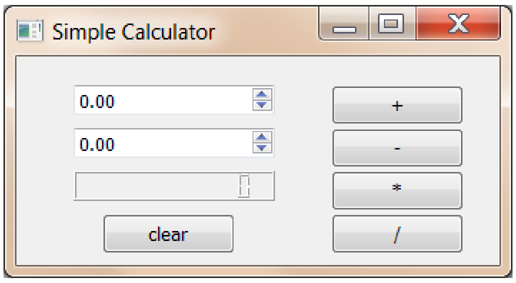
The code for performing mathematical operations should be designed using the following class hierarchy:


Operation
+ compute(x : double, y : double) : double
Addition
+ compute(x : double, y : double) : double
Multiplication
+ compute(x : double, y : double) : double
Subtraction
+ compute(x : double, y : double) : double
Division
+ compute(x : double, y : double) : double

The `Operation` class hierarchy contains the logic for performing various mathematical operations. A concrete class in this hierarchy represents one mathematical operation, which is performed in its `compute()` function that returns the result of an operation.

Implement the Factory method design pattern using one factory class named `OperationFactory` so that an instance of a class in the `Operation` class hierarchy can be created based on the requested operation (`+`, `-`, `*`, `/`). Additionally, `OperationFactory` should also be an implementation of Singleton.

The user interface for the calculator should make use of five `QPushbutton`s, two `QDoubleSpinBox`es and one `QLCDNumber` as shown below:



`QDoubleSpinBox`es allow user to enter numbers and `QLCDNumber` is for displaying the result of the calculation. To use the calculator, the user should first enter two numbers and then select one of the four (`+`, `-`, `*`, `/`) buttons. The result of the operation is displayed in `QLCDNumber`. The button `clear` is used to clear `QDoubleSpinBox`es and `QLCDNumber`.

Note the following:

- The signals emitted by the four buttons (`+`, `-`, `*`, `/`) should be handled in a single slot.
- The user interface code should make use of the `Operation` class hierarchy to get the result of the requested operation.
- Handle overflow in `QLCDNumber` display appropriately.
- Handle division by 0 appropriately.
- GUI can be done manually or using Qt Designer.

Question 3 (25 marks)

Write a class, `FileFinder`, which is capable of running in a thread that can find files in a given directory. If there are files in a directory, the file names are emitted as signals one at a time with uniform pauses in-between by an object of `FileFinder`.

The above class should be utilized within a GUI application with a dedicated GUI class, which allows the user to choose a directory (using a file dialog) and the files in the directory are displayed in the GUI. The GUI class should use `FileFinder` to obtain file names in the selected directory.

You are expected to implement threading using the technique explained in learning unit 7.

Develop a GUI either manually or using the Qt Designer for this application.

Question 4 (25 marks)

Expand the solution in question 2 so that the simple calculator can be used as a plugin, which can be embedded in a web page. Test it by embedding it into a web page and view this web page in a `QWebView`.

Refer to Section 4 of TL103 to obtain support with this question.

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

Question 5

Modify the solution to question 1 so that the writing of XML and reading from XML is done using `QXMLStreamWriter` and `QXMLStreamReader`.

Question 6

It is said that the pseudo-random number generator of Qt `qrand()`generates a deterministic sequence of numbers based on the seed value passed to the `qsrand()` function (refer to `QtGlobal`). In other words you can generate the same sequence of numbers if the seed value is the same.

Write a class named `RandomNumberGenerator` that has a data member `seed` that is used to seed the function `qsrand()`. The value of `seed` is set and managed by the class itself and it cannot be accessed outside the class. It also has a member function, `generateRandomNumbers()`, to generate and return *n* random numbers within a specified range of numbers.

Implement a memento class for `RandomNumberGenerator` that so that the value of the `seed` can be saved at a given time.

Implement a `Caretaker` that is in charge of requesting the list of random numbers from a `RandomNumberGenerator` instance as well as requesting and maintaining a number of mementos of `RandomNumberGenerator`. Of course, the `Caretaker` can request the instance of `RandomNumberGenerator` to roll back to a previous seed value using a memento.

To make it fun use `RandomNumberGenerator` to simulate the rolling of a dice *n* number of times. The values of a dice are between 1 and 6. The `Caretaker` should present the values of the dice (for every roll) to the user graphically as well as let the user choose a previous sequence to demonstrate that the function `generateRandomNumbers()` produces the same sequence of numbers for a given seed value passed to `qsrand()` using the mementos saved in the `Caretaker`.

Note that you have the flexibility in designing the classes taking into account the requirements stated above. You can also decide how you want to represent the values of the dice graphically.

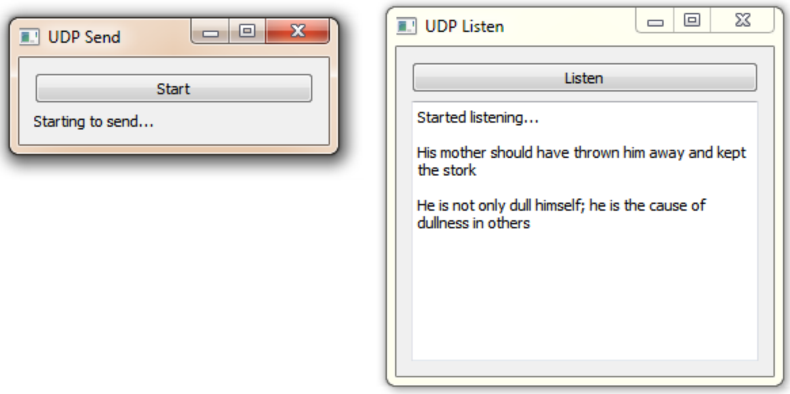
Question 7

Write a program that simulates a random walk in a Cartesian plane, where the object walks one point at a time in one of the four directions; North, South, East or West. The program takes a starting point (x, y) and number of steps as the inputs and as the output it produces the coordinates of all the points of the random walk for the specified number of steps.

Create a simple console application that starts the above program as a separate process and displays the points of the random walk on the console. Allow the user to enter the starting point and the number of steps on the console.

Question 8

Write two applications, a listener and a sender application, that communicate via UDP. The listener application should start listening on a port when a "Listen" button is clicked. The sender should broadcast randomly selected messages at random time intervals (where the messages can simply be stored in a `QStringList` in the sending application) when a "start" button is clicked. As messages are broadcast from the sender, they should be picked up by the listener and displayed on the listener interface/window.



Starting to send...

Started listening...

His mother should have thrown him away and kept the stork

He is not only dull himself; he is the cause of dullness in others

SEMESTER 2

Semester 2 – Assignment 1

Chapters 7, 12, 13, and 14 are covered in this assignment.
Total marks: 100

PART A (To be submitted)

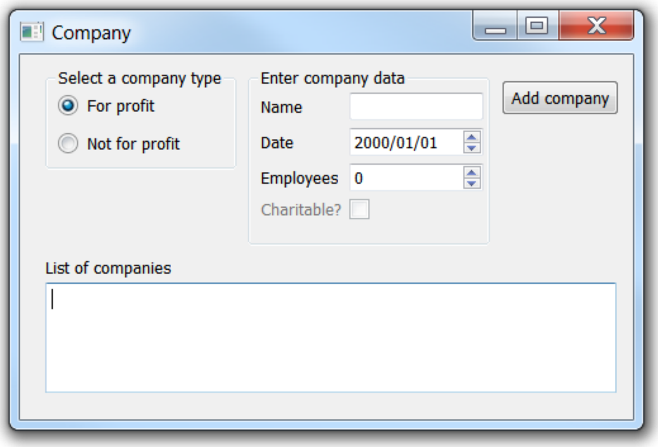
Question 1 (25 marks)

Consider the following scenario. You need to maintain a list of companies, where each company has a name and a date on which it was formed. However, there are two basic types of companies, those that are "for profit" (which have a data member indicating the number of employees in the company), and those that are "not for profit" (which have a flag indicating whether they are a charitable organisation or not).

Design a solution (using appropriate classes) so that the following can be achieved. Ensure that you avoid using any anti-patterns in your solution:
A user should be able to input data for the two types of companies. If data for one type of company is being added, the user should not be able to add data for the other type. For example, if data is being added for a "for profit" company, the user should not be able to indicate its charitable status. You can use any method to achieve this restriction.
You should maintain a list of companies that contains companies of both types. Companies that are added via the GUI should be automatically added to the list.
Display the list, as data is added, in alphabetic order (by company name). The data can be displayed in any format.
Serialize the list.
Data from both lists should be saved to a single file (named backup.txt) when the application is closed. You may use any format to store data in this file.
Data should be added to the list from this same file (if it exists) when the application is started.

HINT: Consider using a class hierarchy with an abstract base class.

See below for an example of the interface:



Question 2 (25 marks)

You need to change the solution to Question 1, so that it follows new specifications.

- You need to change the way the list is handled.
  - The company list should be maintained in a separate class (if this has not been done already).
  - There should be two lists: one for "for profit" companies, and one for "not for profit" companies.
  - This means that functions that were written to access the contents of the previous list will also need to be adapted. These lists should be accessed independently/separately.
  - When displaying the lists on the GUI, they should be displayed separately (not in one, mixed list as before), both in alphabetic order.
  - There should still be only one function to add data to the list. This function needs to determine what type of object is being passed to it (using its meta-object), and then add it to the correct list.
- You need to change the way data is written to, and read from, file. Ensure the following.
  - When writing to file
    - You may not use public functions to access the values of data members in the lists. You will still need a way to access the objects in the lists, though.
    - You may not assume that you know what properties need to be written to file.
    - You may use the meta-object to access company properties and their values.
  - When reading from file
    - You may assume that you know what the file contains.

- You may use either one or two files to save the data.

Question 3 (25 marks)

Write a GUI application that makes use of Qt's `QStringListModel` and `QListView` to manage a list of book titles. The application should support the following.

- The user should be able to add a book title via a text entry field and button. The book 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 book that is selected in the view (ensuring that the user does want to delete the title). Add a button to the GUI to achieve this.
- The list of book titles is maintained in alphabetical order. This should be true when data is edited in the view as well.

Question 4 (25 marks)

Write an application that uses regular expressions to check dates in a text file provided by the user. The following functionality should be included:

- The user should be able to select the file to check, using a standard file open dialog box.
- The file should be displayed in the GUI.
- Discovered dates should be highlighted/emphasised in some way in the text.
- You do not need to check that the dates are valid.

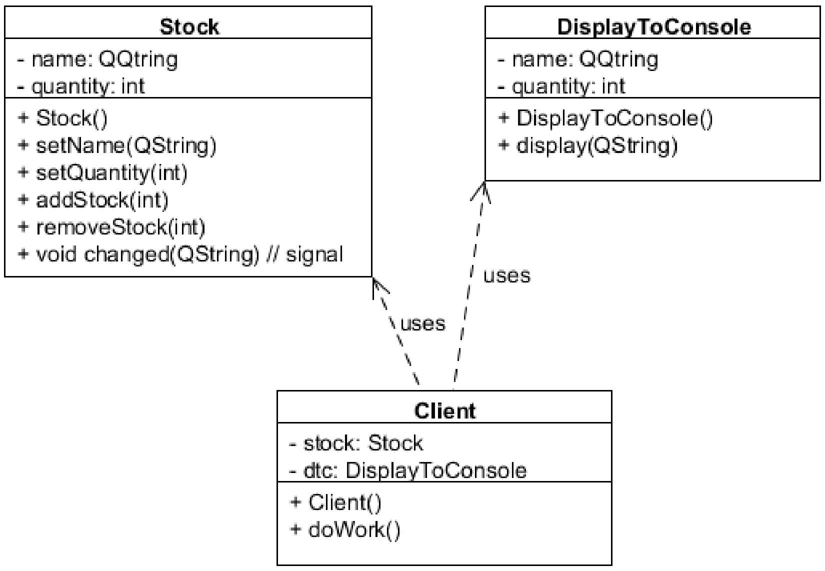
A date could take any one of the following forms:

- 1 Dec 2000
- 1 December 2000
- 01/12/2000
- 2000/12/01

where the '/' character in numerical dates could also be a space, a dot or a dash. The year range to check for would be from 1900 to 2099.

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

Write a console application that implements the UML diagram below:



For the `Stock` class, the functions work as follows:

- `setName()` sets the Stock object's name.
- `setQuantity()` sets the amount of stock on hand.

For the `DisplayToConsole` class, the function works as follows:

- `display()` simply display its argument to the console.

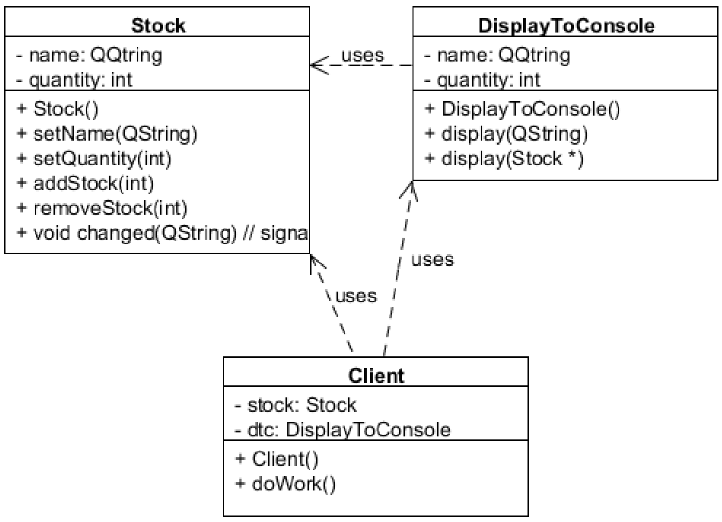
For the `Client` class, the function works as follows:

- `doWork()` should set up values for a `Stock` item and then add and remove stock from that item by invoking `addStock()` and `removeStock()` respectively.

As changes are made to the stock quantity, the new quantity should be displayed to the console. Use signals and slots to ensure that you do not couple the `Stock` and `DisplayToConsole` classes in your design. Thus, when any change to the quantity of the `Stock` item is made, a signal should be emitted that is connected to the `DisplayToConsole` object.

Question 6

Consider the updated UML diagram below:



Change the application in Question 5, so that the following additional functionality can be achieved:

- The `doWork()` function adds two dynamic properties to a `Stock` object: a property named *expiry* with some date value, and a property named *price* with some float value.
- Pass this object to the `DisplayToConsole::display(Stock *)` function where the object's dynamic properties are found and displayed to the console. You may not assume in this function that you know what the dynamic properties are.

Question 7

Instead of using Qt's model-view approach, it is possible to use its item view convenience classes. Rework Question 3, using a `QListWidget` instead of the full model and view classes.

Ensure that you understand the different approaches and when/why you would use one or other approach.

Question 8

Write an application that uses regular expressions to find telephone numbers in a text file provided by a user. The following functionality should be included:

- The user should be able to select the file to check, using a standard file open dialog box.
- The file should be displayed in the GUI.
- The telephone numbers should be highlighted in yellow in the text.

Telephone numbers could be from any of the following formats

- 000-000-0000

- 000-0000000
- 0000000000
- (000)000-0000
- (000)0000000

and should all be converted to the first format (000-000-0000) in the final display.

Semester 2 – Assignment 2

Chapters 15, 16, 17, and TI103 are covered in this assignment.
Total marks: 100

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 (30 marks)

Study the `Contacts` project, a console application provided on *my*Unisa under Additional Resources.

Modify the `Contacts` project to be a GUI application, which allows the user to add and delete contacts from the list instead of using `ContactFactory`. The GUI should allow the user to view the entire contact list, telephone lists and address lists based on a category as allowed by `ContactList`. You can decide on the design of the GUI and it can be done manually or using Qt Designer.

Expand this project by adding appropriate classes to write a `ContactList` object in an XML format. Use the `QObjectWriter` class in the Ezust library `dataobjects` to create an XML format of a `ContactList` object. The XML format should then be written to a file. You have to modify the classes in the `Contacts` project to utilize `QObjectWriter`. However, you are expected to make use of `QObjectWriter` as it is. The `dataobjects` library should be included in the `Contacts` project as explained in the section *Using the libraries* of TL102.

Expand the `Contacts` project, to re-create a `ContactList` instance with `Contact` instances from the XML file. Reading of the XML file of a contact list should be achieved using SAX.

The `ContactList` object should be saved automatically when the GUI is closed, to a file chosen by the user. Similarly when the application starts it should give the user an option to read from an XML file. Make use of the standard file dialog to assist the user in selecting files.

You are expected to introduce new classes and follow good programming design principles.

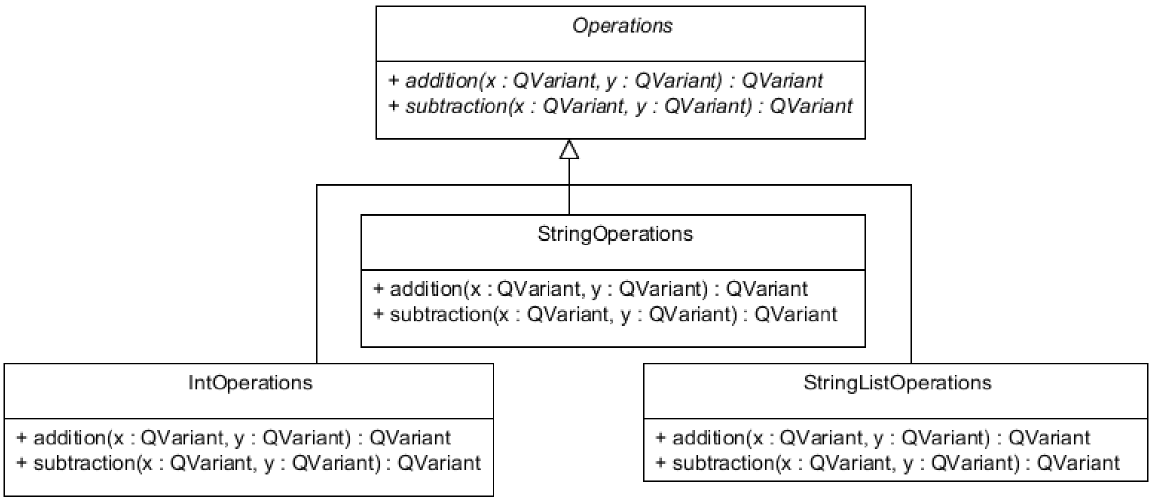
Question 2 (20 marks)

In this exercise, you will create two class hierarchies; one representing multiple interpretations of two operations, addition and subtraction, for different data types and one representing widgets for these different data types. We use these two class hierarchies to demonstrate how these different widgets and different operations can be interchangeably utilized using factories.

A basic design for the operations hierarchy is given on the next page.

`IntOperations` represents operations for `int`. Hence addition and subtraction of two `int`s follow the generally accepted mathematical interpretation of addition and subtraction. Examples: 5 + 4 = 9 and 2 – 1 = 1

`StringOperations` represents operations for `QString`. Addition of two `QString`s in this context means appending the second `QString` to the first one. For example if `QString s1("A"); QString s2("B");` then `s1+s2` should produce `AB`. Subtraction of a `QString` from another `QString` means all occurrences of the latter string is removed from the former string. Example: `QString s1("ABCCC"); QString s2("C");` then `s1-s2` should produce `AB`. You may decide how you want to deal with case sensitivity.



Operations
+ addition(x : QVariant, y : QVariant) : QVariant
+ subtraction(x : QVariant, y : QVariant) : QVariant
StringOperations
+ addition(x : QVariant, y : QVariant) : QVariant
+ subtraction(x : QVariant, y : QVariant) : QVariant
IntOperations
+ addition(x : QVariant, y : QVariant) : QVariant
+ subtraction(x : QVariant, y : QVariant) : QVariant
StringListOperations
+ addition(x : QVariant, y : QVariant) : QVariant
+ subtraction(x : QVariant, y : QVariant) : QVariant

`StringListOperations` represents operations for `QStringList`. Addition and subtraction in this context correspond to the set operations union and difference respectively. When two `QStringList`s are added, the result is a union of the set of `QString`s represented by these two lists. For example if `list1` contains `"A"`, `"B"` and `"C"` and `list2` contains `"C"`, `"D"` and `"E"` then `list1+list2` should produce `"A"`, `"B"`, `"C"`, `"D"` and `"E"` (Note that `"C"` is not duplicated). When one `QStringList` is subtracted from another, the result is a set difference. For example if `list1` contains `"A"`, `"B"` and `"C"` and `list2` contains `"C"`, `"D"` and `"E"` then `list1-list2` should produce `"A"` and `"B"`. Again you may decide how you want to deal with case sensitivity.

Note that you are expected to reuse functions from relevant Qt classes to realize various operations for various data types as described above.

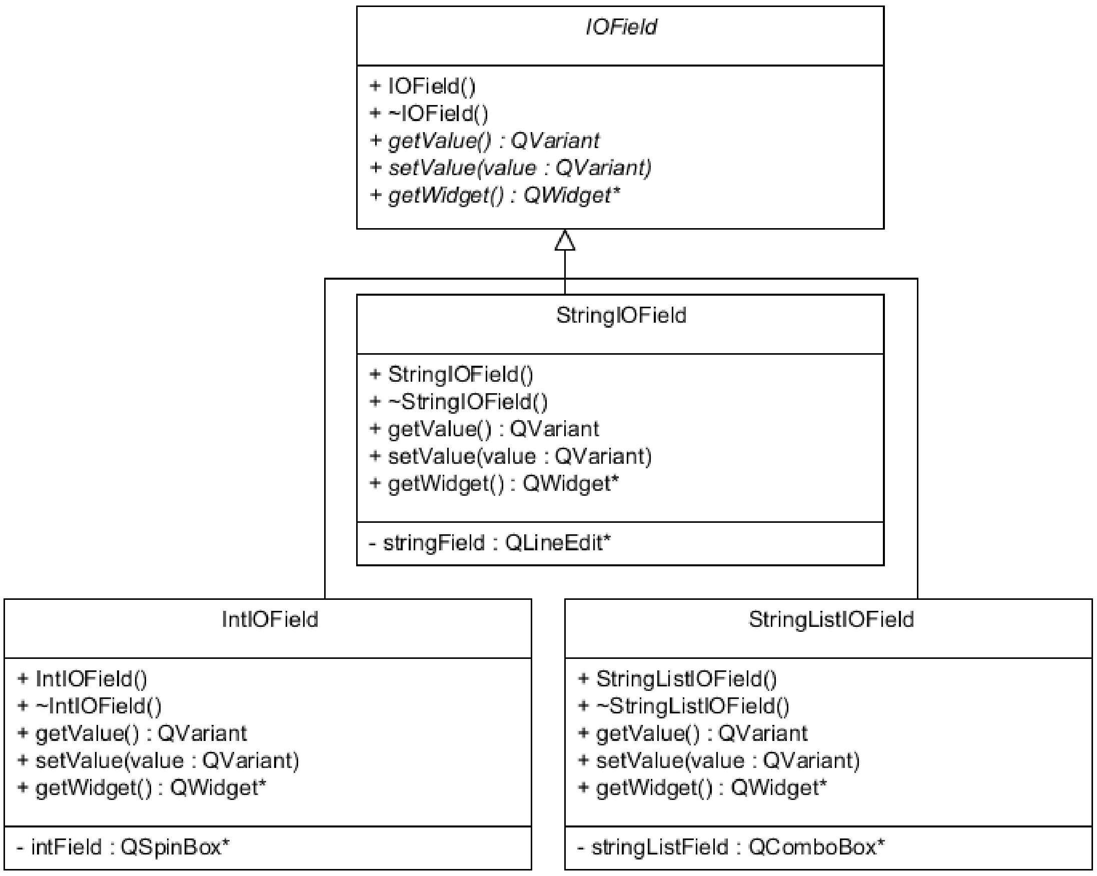
A basic class hierarchy (`IOField`) that can be used for input and output of the data types in the `Operations` class hierarchy is given on the next page.

The idea with the `IOField` class hierarchy is that different widgets can be treated uniformly. Each concrete class of `IOField` has an appropriate data member (a type of `QWidget`) suitable for input and output of a data type. `getValue()` returns the value in the widget and `setValue()` allows value to be set in the widget. `getWidget()` simply returns the `QWidget` data member of the respective class.

Implement two separate factories that implement the Factory method design pattern: one to generate objects of the `Operations` class hierarchy and another factory to generate objects of the `IOField` class hierarchy. Both factories should make use of appropriate input parameters in their respective factory methods. Make both factories implementations of Singleton.

Include a graphical user interface in the application, where the user can choose between int, string or string list. Based on the user's chosen data type, he/she is allowed to enter data in two different `QSpinBox`es or two different `QLineEdit`s or two different `QComboBox`es respectively. Then the user can choose between two operations (addition or subtraction), which will utilize the appropriate function implementations in the `Operations` class hierarchy and the result of the operation is displayed on a third appropriate `IOField`.

You can decide on the design of the GUI and it can be done manually or using Qt Designer. You are expected to follow good programming principles.



Question 3 (25 marks)

Write a class `InvestmentCalculator` that is capable of running in a thread to simulate yearly investment growth using compound interest for a given amount, for a given interest rate and for a given number of years. This class calculates the growth in the investment yearly, then emits the new investment amount in the investment as a signal and pauses for a small period of time before it continues to calculate the growth in the investment for the next year.

The above class should be utilized within a GUI application with a dedicated GUI class, which allows the user to specify the initial investment amount, interest rate and the total number of years of investment. The GUI class should use `InvestmentCalculator` to obtain the growth in investment and the result produced by `InvestmentCalculator` should be displayed on the GUI.

You are expected to implement threading using the technique explained in learning unit 7. Develop a GUI either manually or using the Qt Designer for this application.

Question 4 (25 marks)

Expand the solution in question 3 so that the investment calculator can be used as a plugin, which can be embedded in a web page. Test it by embedding it into a web page and view this web page in a `QWebView`.

Refer to Section 4 of TL103 to obtain support with this question.

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

Question 5

Modify the solution to question 1 so that the writing of XML and reading from XML is done using `QXMLStreamWriter` and `QXMLStreamReader`.

Question 6

It is said that the pseudo-random number generator of Qt `qrand()`generates a deterministic sequence of numbers based on the seed value passed to the `qsrand()` function (refer to `QtGlobal`). In other words you can generate the same sequence of numbers if the seed value is the same.

Write a class named `RandomNumberGenerator` that has a data member `seed` that is used to seed the function `qsrand()`. The value of `seed` is set and managed by the class itself and it cannot be accessed outside the class. It also has a member function, `generateRandomNumbers()`, to generate and return *n* random numbers within a specified range of numbers.

Implement a memento class for `RandomNumberGenerator` that so that the value of the `seed` can be saved at a given time.

Implement a `Caretaker` that is in charge of requesting the list of random numbers from a `RandomNumberGenerator` instance as well as requesting and maintaining a number of mementos of `RandomNumberGenerator`. Of course, the `Caretaker` can request the instance of `RandomNumberGenerator` to roll back to a previous seed value using a memento.

To make it fun use `RandomNumberGenerator` to simulate the rolling of a dice *n* number of times. The values of a dice are between 1 and 6. The `Caretaker` should present the values of the dice (for every roll) to the user graphically as well as let the user choose a previous sequence to demonstrate that the function `generateRandomNumbers()` produces the same sequence of numbers for a given seed value passed to `qsrand()` using the mementos saved in the `Caretaker`.

Note that you have the flexibility in designing the classes taking into account the requirements stated above. You can also decide how you want to represent the values of the dice graphically.

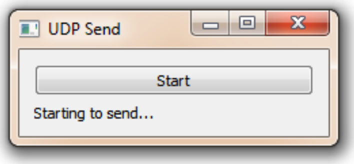
Question 7

Write a program that simulates a random walk in a Cartesian plane, where the object walks one point at a time in one of the four directions; North, South, East or West. The program takes a starting point (x, y) and number of steps as the inputs and as the output it produces the coordinates of all the points of the random walk for the specified number of steps.

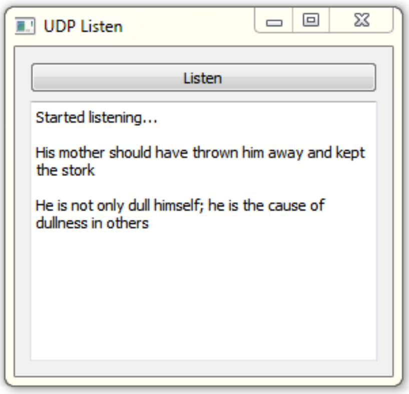
Create a simple console application that starts the above program as a separate process and displays the points of the random walk on the console. Allow the user to enter the starting point and the number of steps on the console.

Question 8

Write two applications, a listener and a sender application, that communicate via UDP. The listener application should start listening on a port when a “Listen” button is clicked. The sender should broadcast randomly selected messages at random time intervals (where the messages can simply be stored in a `QStringList` in the sending application) when a “start” button is clicked. As messages are broadcast from the sender, they should be picked up by the listener and displayed on the listener interface/window.



```
Starting to send...
```



```
Started listening...

His mother should have thrown him away and kept
the stork

He is not only dull himself; he is the cause of
dullness in others
```

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