Tutorial Letter 101/3/2018

Programming: Contemporary Concepts
COS2614

Semesters 1 and 2

School of Computing

General module information and assignments

BARCODE

CONTENTS

Page

IMPORTANT INFORMATION

Please activate your *my*Unisa account and your *my*Life email address and ensure that you have regular access to the *my*Unisa module site COS2614-2018-S1/S2.

This is a blended online module, and therefore your module is available on *my*Unisa. However, in order to support you in your learning process, you will also receive some study material in printed format.

1 INTRODUCTION

Dear Student

Welcome to COS2614, Programming: Contemporary Concepts. We trust that you will find this module stimulating and interesting, and wish you a successful semester of study. The focus of this module is on object oriented programming (OOP) using C++ as the implementation language and the Qt framework for developing object oriented programs.

Do not hesitate to contact your lecturer (on *my*Unisa, by email, or by telephone) if you are experiencing problems with the content of this tutorial letter or any aspect of the module.

2 MODULE FORMAT

2.1 Blended module

Please note that this module is offered in a blended format which means that although 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 *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 and to participate in online discussion forums.

Because this is an online module, you need to go online to see your study material 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 COS2614-18-S1/S2 in the row of modules in the orange blocks across the top of the webpage. Remember to also check in the -more- tab if you cannot find it in the orange blocks. Click on the module you want to open.

2.2 Printed materials to support the online module

We will also provide you with some of the study material in printed format. You will receive a copy of this tutorial letter.

Note, however, that other tutorial matter will not be printed (such as tutorial letters 102 and 103, and assignment solutions that are issued as tutorial letters 201, 202 and 203). You will be able to download them as PDFs from *my*Unisa.

You can access all the study material on *my*Unisa**. You should NOT wait for the printed documents to arrive to start studying.**

Please consult the *Study @ Unisa* publication for more information on the activation of your *my*Life email address as well as obtaining access to the *my*Unisa module site.

3 LECTURER(S) AND CONTACT DETAILS

The details of the lecturers will be provided on the home page of the COS2614 site on *my*Unisa.

The details of the lecturers will also be communicated in a COSALL tutorial letter.

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

To contact the University, you should follow the instructions in the *Study @ Unisa* publication. Remember to have your student number available when you contact the University.

4 RESOURCES

4.1 E-tutor

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.

4.2 Telecentres

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 ASSESSMENT

Assessment for COS2614 is in the form of three assignments and an exam.

5.1 Formative assessment (Assignments)

Below is a break-down of the assignments as they occur in the respective semesters.

Semester 1

Assignment	Due date	Weight	Unique assignment number
1	05 March 2018	20%	747209
2	26 March 2018	50%	785835
3	9 April 2018	30%	894440

Semester 2

Assignment	Due date	Weight	Unique assignment number
1	20 August 2018	20%	874957
2	10 September 2018	50%	738653
3	25 September 2018	30%	806313

The assignment questions are given in section 7 of this tutorial letter.

5.2 Summative assessment (exam)

A 2 hour exam for this course will be scheduled for the end of the semester. The Examination Section will inform you of the time, date and venue. You can also obtain the examination date from *my*Unisa.

Please note that you will be admitted to the examination if-and-only-if you submit at least one assignment before the relevant due date on or before 1 April 2018 (in semester 1) or 11 September 2018 (in semester 2).

The exam will assess your mastery of the outcomes of the course as covered in the assignments. After the due date of the final (third) assignment, additional information about the exam will be included in TL103 (which will be made available under Additional Resources).

5.3 Final mark

The final mark is made up of the assignment marks and the exam mark as follows.

Component	Weight
Assignments	20%
Exam	80%

6 SUBMISSION OF ASSIGNMENTS

6.1 Completing and submitting assignments

There are three assignments that you need to complete. These assignments consist of practical work using the Qt Creator software development environment.

6.2 What to submit

All three assignments have two parts. Part A must be submitted (as explained below) but Part B should not be submitted (it is for self-assessment). Solutions to both parts of the respective assignments will be placed under Additional Resources and discussions of these solutions will be included in TL201, TL202 and TL203 (under Additional Resources). 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.

The questions in Part A are in the form of programs that you have to write and get to work using the prescribed software.

For each assignment, you must submit a single zip file containing the code for these programs. This single zip file should contain your solutions to each question in separate folders. Each folder should contain .pro, .h and .cpp files for that specific question. If the project for a question uses .ui files or any other files, include them in the submission folder as well.

DO NOT include the following in your submission:

- pro.user files
- directories containing the (release or debug) builds of your programs. (These directories contain the compiled .o and .exe files for your project.)

If you do not submit an assignment in the format specified above you will be awarded 0 marks for it. Note that NO resubmissions will be allowed.

Your code must compile and run on the prescribed software for COS2614. If we cannot compile and run the code that you submit, you will be given 0 marks for the corresponding question. So rather submit a program that partially fulfils the requirements of the question but that at least compiles and runs, than attempt to do everything and submit a program that does not compile.

6.3 How do we mark your assignments?

Firstly, we simply unzip your submission folder. Then we open your project using Qt Creator and build it. If it can be compiled, then we execute and test your program. To make sure that your project will indeed build and execute on our side, test the projects in the submission folder (by making a copy of the submission folder) like we do.

6.4 Submitting Assignments

Go to the Assignments option on *my*Unisa to submit the .zip file.

Note: The only students exempted from submitting their assignments via *my*Unisa are those who are in prison. These students should still submit the source code on a CD in the same format as explained in section 6.2 so that we can build and execute the programs. Non-electronic assignments submitted by other students will not be marked and hence 0 marks will be awarded.

Please note that it is your responsibility to check that your assignment is registered on the assignment database. You can do that on the Assignments option.

6.5 Extension of due dates

The time available in a semester is limited, so please adhere to the assignment due dates. We understand that there are sometimes reasons why you cannot submit an assignment by the due date, so the following allowance has been made: If you submit Assignment 1 or 2 within one week after the due date, it will still be accepted and marked. Assignments that are submitted after the one-week extension may be marked but will not count towards your year mark. NO extension to the due date of Assignment 3 is possible because the time available will not allow it to be marked and returned before the examination.

7 ASSIGNMENTS

7.1 SEMESTER 1

7.1.1 Assignment 1 for semester 1

Due date:	5 March 2018
Unique assignment no:	747209

Study material:	Chapters 1, 2, 3, 4 and 5

Part A (To be submitted)

Question 1

Write a Qt Graphical User Interface (GUI) program to generate a username and an initial password given the full name of the user.

Given below are the rules for generating a valid username:

- It consists of 5 lower case characters.
- The user name is created by combining the first 4 characters of the first name with the first character of the surname.
- If the first name does not have 4 characters, more characters are taken from the surname to make up the user name.
- If the total number of characters in the first name and the surname is less than 5 then append sufficient number of 0s to create the username.

An initial password is generated by combining randomly selected 5 characters from the full name of the user.

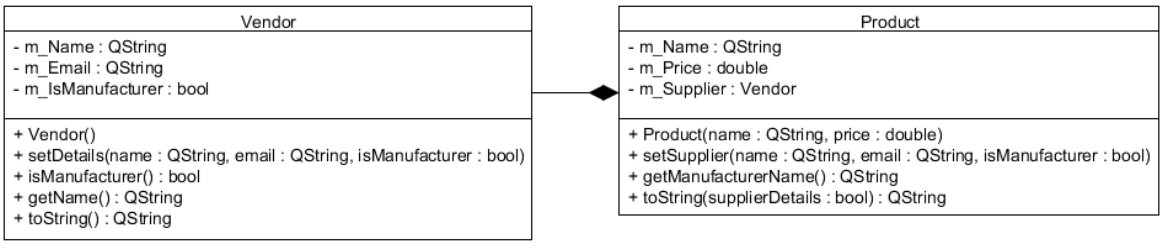
Note than no spaces are allowed in the username or password.

User input should be obtained using a `QInputDialog`. You can expect the full name as a single string where each word is separated using a space. For example: `Mike William Owen`. The output (username and password) should be displayed using a `QMessageBox`.

You need not do any verification of the user input. You can assume that the user will enter at least two words in the `QInputDialog`. Assume that the first word of the input is the first name and the last word in the name is the surname - for example, for the sample input `Mike William Owen`, `Mike` should be read as the first name and `Owen` as the surname and the generated username should be `mikeo`.

Question 2

Implement the following classes in the UML class diagram according to the description that follows:



A product is described using a name, a price and a supplier. The supplier and the manufacturer can be the same for a product.

A `Product` is initialised using a name and a price. The supplier details are set using the function `setSupplier()`, which invokes the suitable function of `Vendor` to set the details of the vendor. If one requests a product for its manufacturer, it returns the name of the vendor

stored in `m_Supplier`, if it is set as the manufacturer. Otherwise an `Unknown` string is returned. `toString()` of `Vendor` returns a string representation of the values of its data members in a readable format. `toString()` of `Product` always returns a string representation of a product. If `supplierDetails` in `toString()` of `Product` is `true`, it returns both the supplier and product details. Otherwise only product details are returned.

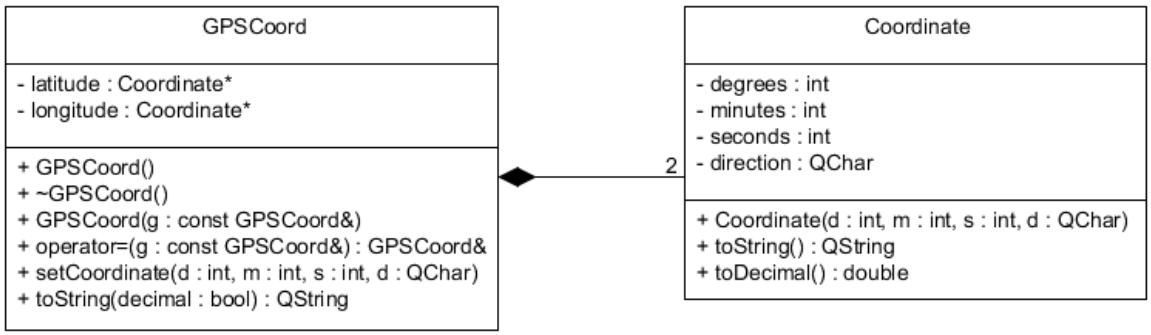
Implement it as a console application, where the product and supplier details are entered from the keyboard. Display the results of `getManufacturerName()` and `toString()`(for both `true` and `false` parameter values) on the console.

Part B (Self-assessment: Do not submit)

Question 3

The GPS coordinate system specifies a position as two geographic coordinates: one for latitude and one for longitude. A geographic coordinate consists of a number of degrees, minutes and seconds north or south (for latitude) and east or west (for longitude). For example, the GPS coordinate for Unisa's main campus in Pretoria is 25° 46' 3" S 28° 12' 3" E.

Define and implement classes to store GPS coordinates using the following UML class diagram:



The `Coordinate` class stores three integers for the degrees, minutes and seconds of a coordinate, and a character for the cardinal direction. The number of degrees should be an integer from 0 to 179, and the number of minutes and seconds should both be integers from 0 to 59. The cardinal direction should be N or S for latitude, or E or W for longitude. Note that the constructor need not check whether the values provided as parameters comply with these restrictions – it can assume that the user (or client program) provides valid values.

The `toString()` member function of the `Coordinate` class should return a string comprising the three integer values annotated with the necessary degree symbol and inverted commas to indicate degrees, minutes and seconds, and the cardinal direction (Example: 25° 46' 3" S). Use `QChar(248)` to produce the ° character.

The `toDecimal()` member function of the `Coordinate` class should convert the geographic coordinate to a decimal (double) value. To do this, the number of minutes must be divided by 60, and the number of seconds by 3600, and these fractions must be added to the number of degrees. The cardinal direction must then be used to determine whether the decimal number of degrees should be positive or negative: If the cardinal direction is N or E, the value should be positive, but if it is S or W, it should be negative. Thus for the coordinate 25° 46' 3" S, the decimal value is -25.767.Note that the `GPSCoord` class has a composition relationship with the `Coordinate` class. There is only a no-arg constructor for the `GPSCoord` class which should initialize its latitude and longitude to 0° 0' 0" N and 0° 0' 0" E, respectively. To change these

values, the `setCoordinate()` member function must be used. If the value of its fourth parameter is 'N' or 'S', the latitude coordinate must be set, but if it is 'E' or 'W', the longitude coordinate must be set.

Note that `latitude` and `longitude` are `Coordinate*`s pointing to objects of type `Coordinate` in the heap memory. Implement the destructor, copy constructor and copy assignment operator as explained in TL102. For the purpose of this exercise, you should include `qDebug()` statements in these three constructs to indicate that they are being executed.

The `toString()` member function of the `GPSCoord` class takes a single boolean value as parameter with the default value false to specify whether it should be rendered as a decimal coordinate or not. If it is false (or no argument is passed), it should return the concatenated string values of its latitude and longitude in geographic coordinates (obtained by simply calling `toString()` on each). If it is true, it should return a string comprising the decimal forms of its `latitude` and `longitude` (obtained by calling `toDecimal()` on each and converting them to strings).

Note also that the definition and implementations of these two classes must be placed in their own header and source files.
In `main.cpp` you should test this app as a console application in the following way:

- Prompt the user to enter longitude and latitude separately. You can assume that the user will enter data correctly.
- Use the data entered by the user to create a GPS coordinate by invoking the constructor and the `setCoordinate()` function.
- Display the string representation of the `GPSCoord` object in both string and decimal formats.
- Include code to invoke copy constructor and copy assignment operator.

Question 4

Chapter 1, section 1.18: Review Questions 6 and 11 (page 65).

Question 5

Chapter 2, section 2.15: Exercises 4 (page 102) and 7 (pages 104 & 105).

Question 6

Chapter 3, section 3.4: Exercise 2 (page 120).

Question 7

Chapter 4, section 4.4: Exercise 2 (pages 132 & 133).

Question 8

Chapter 5, Section 5.13: Review Questions 1, 2 and 6 (page 167).

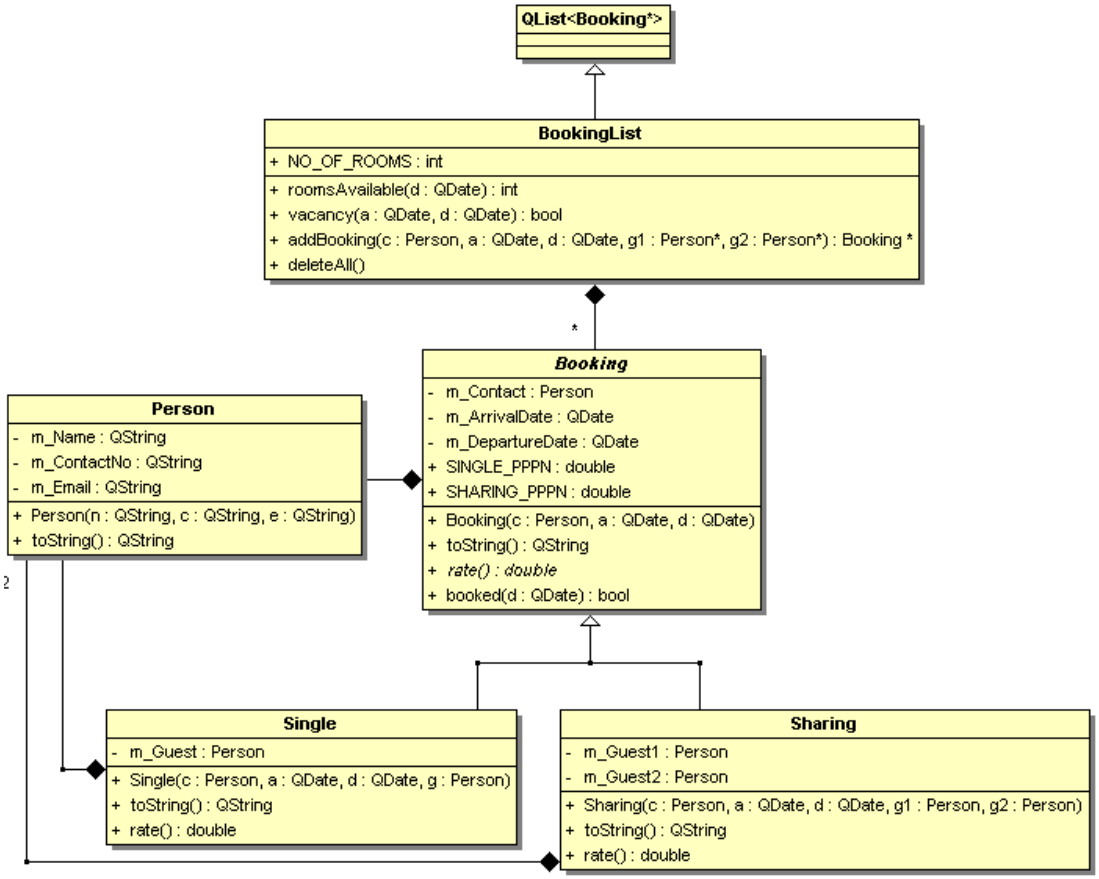
7.1.2 Assignment 2 for semester 1

Due date:	26 March 2018
Unique assignment no:	785835
Study material:	Chapters 6, 8 and 9

Part A (To be submitted)

Question 1

Consider the following UML diagram modelling the classes for an application for recording the bookings at a guest house:



The guest house has a fixed number of rooms, all with two beds in them. The rate per person per night for a single person is more than the sharing rate. (You can decide on the number of rooms and appropriate rates.)

The `BookingList` class keeps a list of (pointers to) `Single` or `Sharing` bookings. The member functions of the `BookingList` class are intended to do the following:

- `addBooking()` takes a contact person, the arrival and departure dates, and pointers to (one or two already constructed) instances of `Person` (representing the one or two guests) as parameters. It first checks whether there is a vacancy for the requested period (by calling `vacancy()`), and if so, constructs an instance of `Single` or `Sharing` (as appropriate) and adds the pointer to this booking to the list. It returns a pointer to the booking so that the client program can output the details.
- `vacancy()` tests takes two dates as parameters (intended to represent the arrival and departure dates, respectively), tests whether the two dates are valid (e.g. 32/12/2011 is not valid), tests whether the arrival date precedes the departure date, and if so, checks whether there is at least one room available on each of the dates from the arrival date to the day before the departure date (by calling `roomsAvailable()` for each of these dates).
- `roomsAvailable()` takes a date as parameter and returns the number of rooms available on that date. It calculates this by going through the entire list of bookings and counting how many of them include the given date (and subtracts this from the total number of rooms).

Hint: Use the `booked()` member function of the `Booking` class.

- `deleteAll()` calls `delete` on each of the pointers in the list. It is intended to be called at the end of the client program to prevent memory leaks.

The member functions of the `Booking` class (and its derived classes) are intended to do the following:

- The constructors of the `Booking`, `Single` and `Sharing` classes take values for their respective data members and initialise them appropriately.
- `toString()` returns a string comprising the names and values of the data members of the (respective type of) booking, formatted neatly for output purposes.
- `rate()` is abstract in the `Booking` class. It returns the single rate per person per night for a `Single` booking, and twice the sharing rate per person per night for a `Sharing` booking.
- `booked()` takes a date as parameter, and tests whether this lies between the arrival date and (the day before) the departure date.

Implement all these classes and write a short program to test them. It should add enough bookings to an instance of `BookingList` to show what happens when an attempt is made to make a booking when the guest house if fully booked for at least one of the days of the booking.

Note that you should write a console application, not a GUI app.

Question 2

Write a Graphical User Interface (GUI) application that displays randomly selected messages at specified time intervals.

The application should satisfy the following:

- When the application commences, it should allow the user to specify the time interval (in milliseconds) between messages (you may use any appropriate widget for user input - an example is `QLineEdit`).
- An error message is displayed if the user sets values for the time interval that are non-numeric or out of range (5000 to 20000 milliseconds) or you may program the input widget to ensure that it does not accept non-numeric or out-of-range values (5000 to 20000 milliseconds).
- A start button is provided on the GUI in order for the application to start displaying messages at the time interval specified.
- Messages are displayed on the same GUI (examples: `QLabel`, `QTextEdit`, etc.) from a list of messages that are randomly chosen at the specified time intervals. Use a `QTimer` to manage the time intervals.

Note that you can choose the type of messages to be displayed. Examples are quotes, advice, weather, meeting appointment reminders, etc.

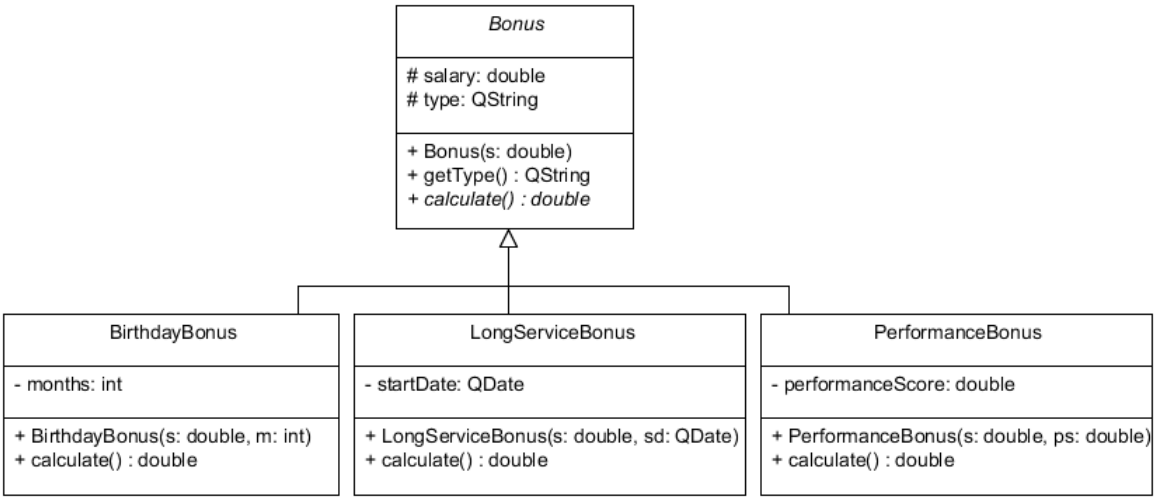
Design the application so that the model and view are separated in different classes. In other words, the logic of randomly generating messages is implemented in one class and the GUI programming is done in another class.

You are expected to do GUI programming manually without the using Qt Designer.

Part B (Self-assessment: Do not submit)

Question 3

Given below is a UML class diagram that models three types of bonuses; birthday bonus, long service bonus and performance bonus.



The class `Bonus` contains two data members to store the salary and the type of the bonus. There are three types of bonuses: birthday bonus, long service bonus and performance bonus. It has a constructor, a getter function to return the type of bonus and the `calculate()` function that has no implementation in this class.

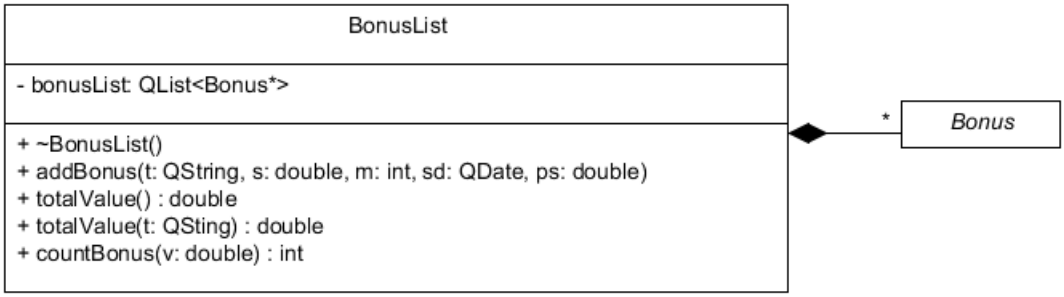
`BirthdayBonus`, a subclass of `Bonus`, has a data member to store the number of months (1 to 12). Its constructor takes two arguments: one for salary and one for months. The constructor also initializes its type correctly in the data member `type` of its superclass `Bonus`. The function `calculate()` calculates bonus on a pro-rata basis on the number of months an employee worked in the company. If the number of months stored in `months` is 12, then it simply returns the salary. Otherwise, it will be a percentage of the salary and this percentage is determined by the percentage of months the employee worked for the company. For example, if the number of months is 3, then `calculate()` will return 25% (3/12 * 100) of the salary.

`LongServiceBonus`, another subclass of `Bonus` calculates long service bonus based on the start date of employment stored in its data member. Its constructor takes two arguments: one for the salary and one for the start date of employment. The constructor also initializes its type correctly in the data member `type` of its superclass `Bonus`. Long service bonus is only awarded when the number of years of service is a multiple of 5 (for example 5, 10, 15, 20, 25, etc.). When the duration of employment is a multiple of 5, the bonus is calculated by adding a percentage of the salary to the salary and this percentage is based on the duration of employment. For example if the duration of employment is 5 years, then the long service bonus is calculated by adding 5% of the salary to the salary. When doing the calculation, make sure that the duration is based on whether the employee has completed 5 (or multiples of 5) years. When calculating the duration, take into account the days, months and years of the start and current dates.

`PerformanceBonus`, another subclass of `Bonus` calculates a performance bonus based on the performance score stored its data member. Its constructor takes two arguments: one for the salary and one for the performance score (between 1 and 5). A performance bonus is only awarded when the performance score is equal to or greater than 3.1. Similar to long service bonus, a performance bonus is calculated by adding a percentage of the salary to the salary

and this percentage is based on the performance score. For example if the performance score is 3.1, then the performance service bonus is calculated by adding 3.1% of the salary to the salary.

Given below are the details of a container class named `BonusList` that manages a list of `Bonus`es. The `Bonus` class in the diagram is the same as the `Bonus` class in the UML class diagram above.



The functions of the `BonusList` class should do the following:

- The destructor must delete all `Bonus` heap objects pointed to by `Bonus*`s in its data member `bonusList`.
- `addBonus()` takes the type of bonus, salary, number of months, start date and performance score in its arguments. Based on the type of bonus, this function creates heap objects of the relevant `Bonus` subclass and adds the pointer to these objects in its data member `bonusList`.
- One `totalValue()` calculates the total value of all bonuses stored in a bonus list and the second `totalValue()` calculates the total value of all bonuses when the type is specified.
- `countBonus()` counts the number of bonuses equal to or above a certain value specified in its argument.

Implement `Bonus`, `BirthdayBonus`, `LongServiceBonus`, `PerformanceBonus` and `BonusList` as specified above. Keep the definitions and implementations in separate header and source files.

Test all these classes by creating at least two objects of every concrete class in the `Bonus` hierarchy and by adding them to a `BonusList` object. Then test `addBonus()`, both `totalValue()` functions and `countBonus()` with appropriate arguments. Display the output for each of these function invocations on the console.

You need to develop a console application and not a GUI app.

Question 4

Chapter 8, section 8.2.2.1: Exercises 1, 2 and 3 (pages 271 & 272).

Question 5

Chapter 8, section 8.8: Exercises 1 and 2 (page 282).

Question 6

Chapter 8, section 8.9: Review Questions 7, 9, 10 and 13 (page 282).

Question 7

Chapter 9, section 9.8: Exercises 1 and 2 (pages 313 & 314).

Question 8

Chapter 9, section 9.11: Review Questions 3, 4, 7 and 8 (page 325).

7.1.3 Assignment 3 for semester 1

Due date:	9 April 2018
Unique assignment no:	894440
Study material:	Chapters 10 and 11

Part A (To be submitted)

Question 1

Implement a GUI application that has a menu `Orders` with two options `Add Order` and `View Order List` (Figure 1).

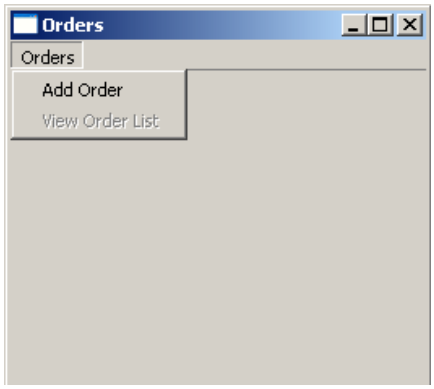


Figure 1

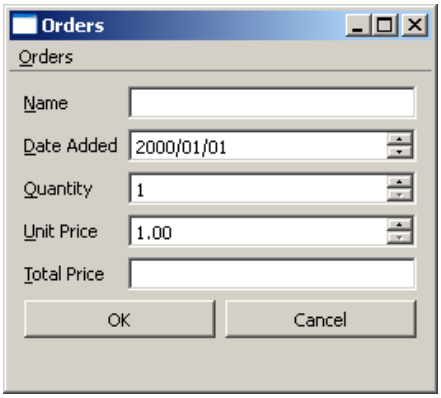
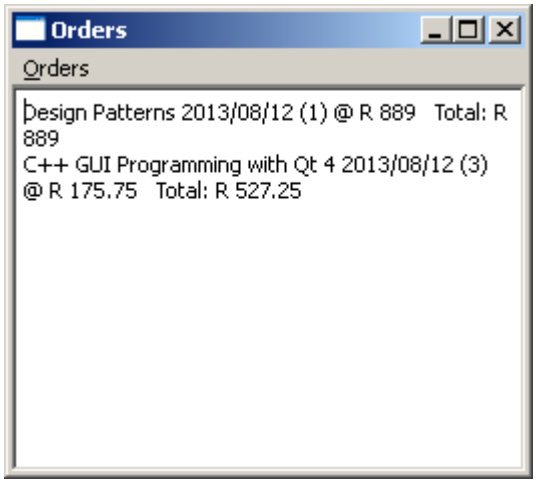


Figure 2

Selecting the menu option `Add Order` should bring up the order form as shown in Figure 2. The user should be able to enter order information in the order form. Clicking on `OK` should create an `Order` and append it to an `OrderList` object, which maintains a `QList` of `Order`s. It should also enable the menu option `View Order List` (Figure 1). Clicking on the `Cancel` button should hide the order form until the user selects `Add Order` menu option again.

Selecting the menu option `View Order List` (when enabled) should list all the `Order`s in a `QTextEdit` as shown below:



Note that the user should be able to select `Add Order` and `View Order List` any number of times so that any number of orders can be placed and the updated order list can be viewed respectively.

You may use the solution to Question 7 of Assignment 2 as a starting point to this solution. You are expected to program the GUI manually rather than using Qt Designer.

Question 2

Expand the class `TextbookMap` and `Textbook` (if required) given in Section 11.3 of Ezust to incorporate the following functions:

(1) `bool foundTextbook(QString isbn) const;` to determine whether a textbook is in the given list.

(2) `void deleteTextbook(QString isbn);` to delete a textbook entry from the list if the textbook is in the list.

(3) `QStringList textBookInfo (QString author) const;` to provide a string representation of the textbooks by the given author.

Provide code to test these new functions.

Part B (Self-assessment: Do not submit)

Question 3

Modify your solution to Question 1 of this assignment to incorporate toolbars in the GUI for the options `Add Order` and `View Order List`.

Question 4

Chapter 10, Section 10.8: Review Questions 1-12 (pages 353 & 354).

Question 5

Chapter 11, Section 11.2.1: Exercise 2 (page 364).

Question 6

Chapter 11, Section 11.7: Review Questions 1-9 (pages 376 – 377).

7.2 SEMESTER 2

7.2.1 Assignment 1 for semester 2

Due date:	20 August 2018
Unique assignment no:	874957
Study material:	Chapters 1, 2, 3, 4 and 5

Part A (To be submitted)

Question 1

Write a Qt Graphical User Interface (GUI) program to compute and display the final module mark of a student in COS2614. The program should also inform the user whether the student has passed or failed (with or without admission to a supplementary exam admission).

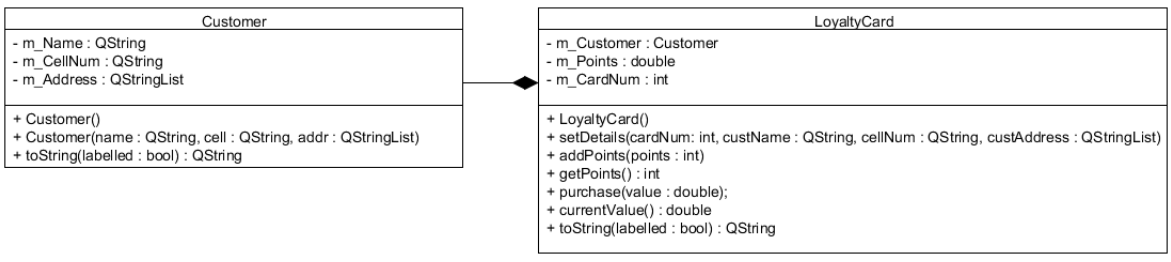
The program should take into account the following information regarding COS2614:

- There are only three assignments in a semester.
- Assignments 1, 2 and 3 contribute 20%, 50% and 30% respectively to the semester mark.
- Semester mark contributes 20% toward the final mark.
- Exam mark contributes 80% toward the final mark.
- You need to obtain a minimum of 40% in the exam for the semester mark to be included in the final mark. If you obtain less than 40% in the exam, your final mark (%) will be equal to the exam mark (%).
- One obtains a pass in COS2614 if the final mark is equal to or above 50%.
- If a student obtains less than 40% final mark, then the student fails the module.
- If a student obtains between 40% and 49% final mark, then the student gets admission to the supplementary exam.

User input should be obtained using a `QInputDialog`. You can expect the assignment and exam mark percentages as a single string separated by a comma and a space. As an example, the user input `75, 0, 55, 40` indicates that the student obtained 75, 0 and 55 percentages for assignment 1, 2 and 3 respectively. 40 is the exam mark percentage. The output should be displayed using a `QMessageBox`. You need not do any verification of user input.

Question 2

Implement the following classes in the UML class diagram according to the description that follows:



A loyalty card is linked to a customer and it is identified using a card number. A customer can accumulate points with the loyalty card when purchasing items from the shop. The number of points obtained is the same as the amount (rounded to an `int`) a customer has spent at the shop. Hence invoking `purchase()` should automatically increase the points on the cards. The value of the points in rands is 4% of the number in `m_Points`. `toString()` returns a string representation of the values of the data members in the respective classes either with or without labels (such as name, cell number, address, etc) for each values.

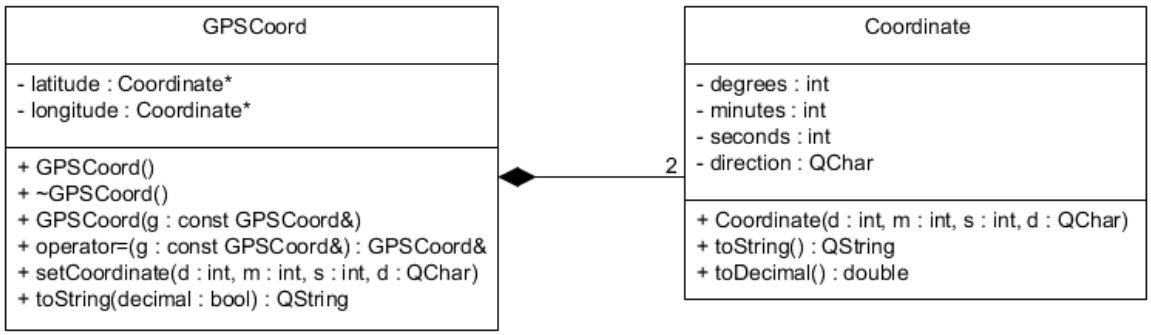
Implement it as a console application, where the customer and loyalty card details are entered from the keyboard. Allow the user to enter at least two purchase amounts. Invoke all the functions in `LoyaltyCard` and display the results of these function invocations where appropriate on the standard console.

Part B (Self-assessment: Do not submit)

Question 3

The GPS coordinate system specifies a position as two geographic coordinates: one for latitude and one for longitude. A geographic coordinate consists of a number of degrees, minutes and seconds north or south (for latitude) and east or west (for longitude). For example, the GPS coordinate for Unisa's main campus in Pretoria is 25° 46' 3" S 28° 12' 3" E.

Define and implement classes to store GPS coordinates using the following UML class diagram:



The `Coordinate` class stores three integers for the degrees, minutes and seconds of a coordinate, and a character for the cardinal direction. The number of degrees should be an integer from 0 to 179, and the number of minutes and seconds should both be integers from 0 to 59. The cardinal direction should be N or S for latitude, or E or W for longitude. Note that the constructor need not check whether the values provided as parameters comply with these restrictions – it can assume that the user (or client program) provides valid values.

The `toString()` member function of the `Coordinate` class should return a string comprising the three integer values annotated with the necessary degree symbol and inverted commas to indicate degrees, minutes and seconds, and the cardinal direction (Example: 25° 46' 3" S). Use `QChar(248)` to produce the ° character.

The `toDecimal()` member function of the `Coordinate` class should convert the geographic coordinate to a decimal (double) value. To do this, the number of minutes must be divided by 60, and the number of seconds by 3600, and these fractions must be added to the number of degrees. The cardinal direction must then be used to determine whether the decimal number of degrees should be positive or negative: If the cardinal direction is N or E, the value should be positive, but if it is S or W, it should be negative. Thus for the coordinate 25° 46' 3" S, the

decimal value is -25.767.Note that the `GPSCoord` class has a composition relationship with the `Coordinate` class. There is only a no-arg constructor for the `GPSCoord` class which should initialize its latitude and longitude to 0° 0' 0" N and 0° 0' 0" E, respectively. To change these values, the `setCoordinate()` member function must be used. If the value of its fourth parameter is 'N' or 'S', the latitude coordinate must be set, but if it is 'E' or 'W', the longitude coordinate must be set.

Note that `latitude` and `longitude` are `Coordinate*`s pointing to objects of type `Coordinate` in the heap memory. Implement the destructor, copy constructor and copy assignment operator as explained in TL102. For the purpose of this exercise, you should include `qDebug()` statements in these three constructs to indicate that they are being executed.

The `toString()` member function of the `GPSCoord` class takes a single boolean value as parameter with the default value false to specify whether it should be rendered as a decimal coordinate or not. If it is false (or no argument is passed), it should return the concatenated string values of its latitude and longitude in geographic coordinates (obtained by simply calling `toString()` on each). If it is true, it should return a string comprising the decimal forms of its `latitude` and `longitude` (obtained by calling `toDecimal()` on each and converting them to strings).

Note also that the definition and implementations of these two classes must be placed in their own header and source files.

In `main.cpp` you should test this app as a console application in the following way:
- Prompt the user to enter longitude and latitude separately. You can assume that the user will enter data correctly.
- Use the data entered by the user to create a GPS coordinate by invoking the constructor and the `setCoordinate()` function.
- Display the string representation of the `GPSCoord` object in both string and decimal formats.
- Include code to invoke copy constructor and copy assignment operator.

Question 4

Chapter 1, section 1.18: Review Questions 6 and 11 (page 65).

Question 5

Chapter 2, section 2.15: Exercises 4 (page 102) and 7 (pages 104 & 105).

Question 6

Chapter 3, section 3.4: Exercise 2 (page 120).

Question 7

Chapter 4, section 4.4: Exercise 2 (pages 132 & 133).

Question 8

Chapter 5, Section 5.13: Review Questions 1, 2 and 6 (page 167).

7.2.2 Assignment 2 for semester 2

Due date:	10 September 2018
Unique assignment no:	738653
Study material:	Chapters 6, 8 and 9

Part A (To be submitted)

Question 1

The UML class diagram on the next page specifies a number of classes that model employees and different kinds of payments.

Every employee has an ID, a first name and surname, and a means of payment.

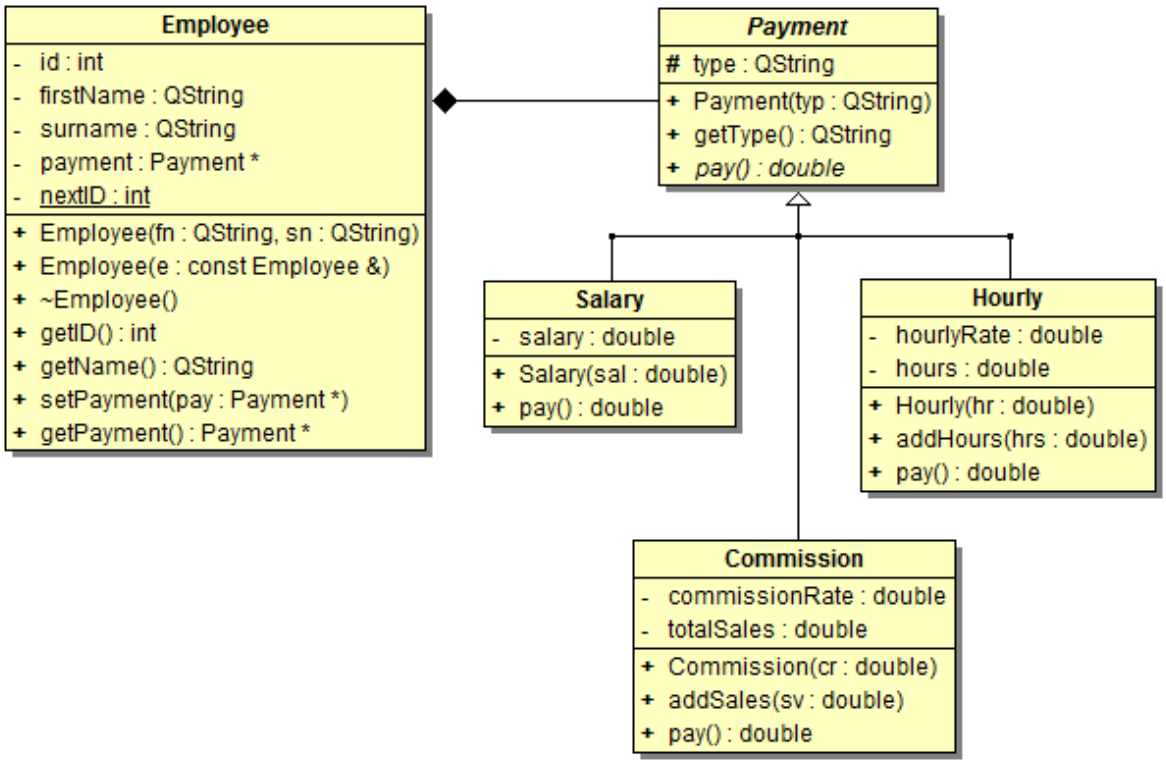
A salaried employee is paid a salary once a month.

An hourly paid employee is paid an amount calculated once a month from the number of hours worked. Whenever the employee works, the number of hours can be incremented. The amount paid is simply the total number of hours multiplied by the hourly rate.

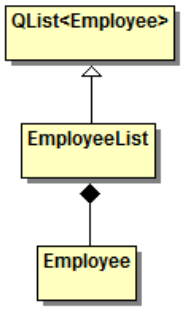
Someone employed on a commission basis is paid a commission once a month calculated from the total of all the sales they made in the month. Whenever a sale is made, the total sales can be increased. The amount paid is simply the commission rate multiplied by the total sales.

Implement the five classes `Employee`, `Payment`, `Salary`, `Hourly` and `Commission`. Note the following:

- These classes must each have their definitions and implementations in separate header and source files.
- The `Employee` class should allocate unique IDs to each new `Employee` instance, starting with the number 1001. The static data member `nextID` should therefore be initialised appropriately, assigned to the `id` of each new instance, and incremented accordingly.
- Since `Employee` instances each contain a `Payment` and access it by means of a pointer, the `Employee` class needs to implement a copy constructor (to do a deep copy) and a destructor to deallocate the memory of the `Payment`.
- The `Payment` class has an abstract member function `pay()` that must be implemented by each of its (concrete) subclasses to calculate the monthly payment as described above.
- The `type` data member of the `Payment` class is `protected` to allow it to be initialised by the constructors of the `Salary`, `Hourly` and `Commission` classes.
- You can assume that the system using these classes will call the `addHours()` and `addSales()` member functions on relevant instances of hourly and commission payments of employees as required (see below.)



Implement another class called `EmployeeList` according to the following UML diagram:



Note that the member functions and data members of the `QList` and `Employee` classes are not provided in this diagram. They are as provided in the `QList` definition in the Qt class reference, and as in the UML diagram at the beginning of this question, respectively.

Note the following about the `EmployeeList` class:

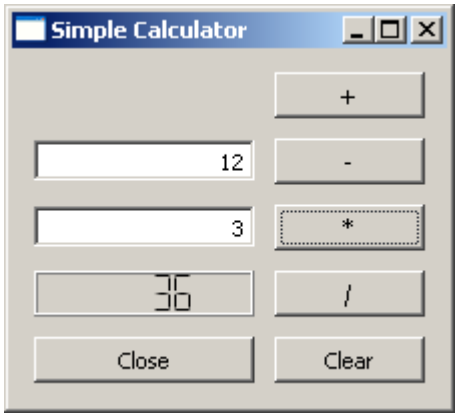
- `EmployeeList` inherits from the class template `QList`, but is not templatized itself. All its elements are `Employee`s.
- `EmployeeList` inherits all the member functions of `QList`, so it need not implement any of its own.

Write a program (a console application) to test the member functions of the `Employee` class and the `Payment` subclasses. It should do the following:

- Add at least six employees (including at least two of each of the concrete subclasses of `Payment`) to an `EmployeeList`.
- Call the `addHours()` member function on the `Hourly` payment of at least one instance of an hourly paid employee that has been stored in the list.
- Call the `addSales()` member functions on the `Commission` payment of at least one instance of a commission paid employee in the list.
- Display a report of the payments owed to each employee at the end of the month, grouped into salaried, hourly paid and commission paid employees.

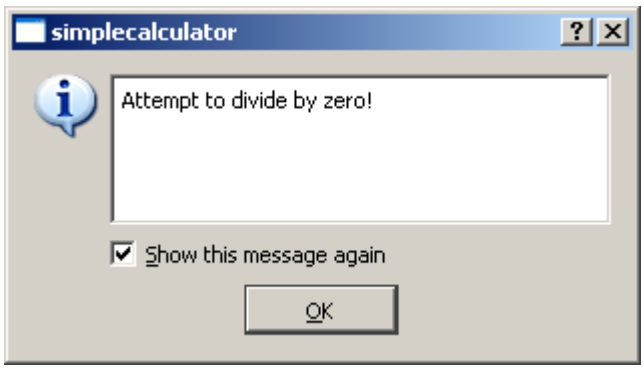
Question 2

Write an application create a simple calculator, which allows four basic arithmetic operations, namely addition, subtraction, multiplication and division. The application should provide the user with line edits to enter two numbers, buttons to select one of the four operations and an LCD display for the result.



Note that the result of the calculation should be displayed as an LCD number. Hint: Use the `QLCDNumber` class.

If the user attempts to divide by 0, an error message like the following should be displayed:



Similarly, if there is an overflow in the number of digits displayed for the result, an appropriate error message should be displayed.

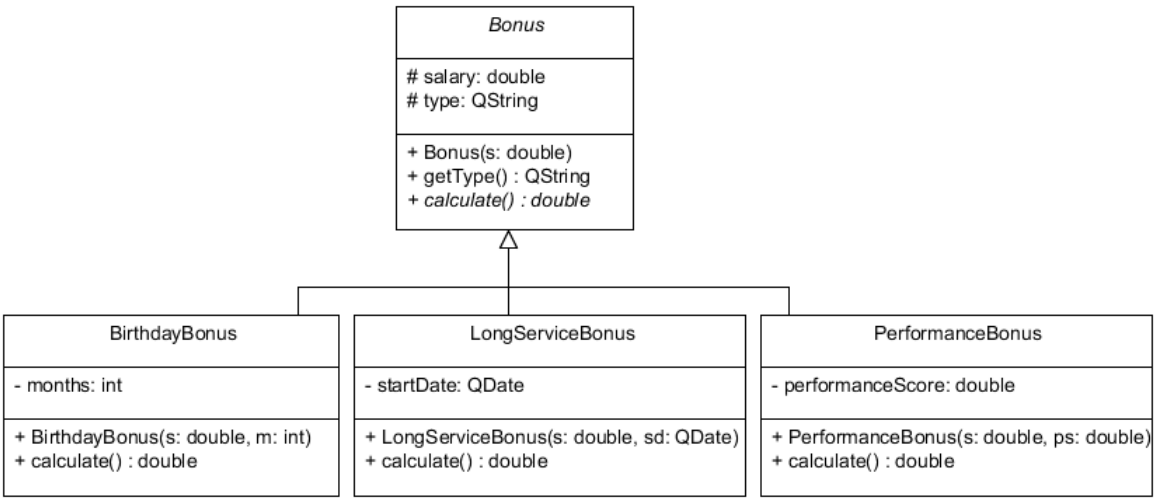
Design the application so that the model and view are separated in two different classes. In other words, calculations are implemented in one class and the GUI programming is done in another class.

You are expected to do GUI programming manually without using Qt Designer.

Part B (Self-assessment: Do not submit)

Question 3

Given below is a UML class diagram that models three types of bonuses; birthday bonus, long service bonus and performance bonus.



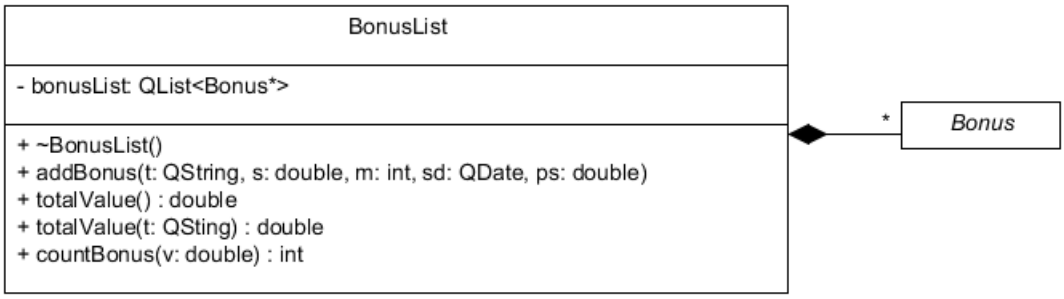
The class `Bonus` contains two data members to store the salary and the type of the bonus. There are three types of bonuses: birthday bonus, long service bonus and performance bonus. It has a constructor, a getter function to return the type of bonus and the `calculate()` function that has no implementation in this class.

`BirthdayBonus`, a subclass of `Bonus`, has a data member to store the number of months (1 to 12). Its constructor takes two arguments: one for salary and one for months. The constructor also initializes its type correctly in the data member `type` of its superclass `Bonus`. The function `calculate()` calculates bonus on a pro-rata basis on the number of months an employee worked in the company. If the number of months stored in `months` is 12, then it simply returns the salary. Otherwise, it will be a percentage of the salary and this percentage is determined by the percentage of months the employee worked for the company. For example, if the number of months is 3, then `calculate()` will return 25% (3/12 * 100) of the salary.

`LongServiceBonus`, another subclass of `Bonus` calculates long service bonus based on the start date of employment stored in its data member. Its constructor takes two arguments: one for the salary and one for the start date of employment. The constructor also initializes its type correctly in the data member `type` of its superclass `Bonus`. Long service bonus is only awarded when the number of years of service is a multiple of 5 (for example 5, 10, 15, 20, 25, etc.). When the duration of employment is a multiple of 5, the bonus is calculated by adding a percentage of the salary to the salary and this percentage is based on the duration of employment. For example if the duration of employment is 5 years, then the long service bonus is calculated by adding 5% of the salary to the salary. When doing the calculation, make sure that the duration is based on whether the employee has completed 5 (or multiples of 5) years. When calculating the duration, take into account the days, months and years of the start and current dates.

`PerformanceBonus`, another subclass of `Bonus` calculates a performance bonus based on the performance score stored its data member. Its constructor takes two arguments: one for the salary and one for the performance score (between 1 and 5). A performance bonus is only awarded when the performance score is equal to or greater than 3.1. Similar to long service bonus, a performance bonus is calculated by adding a percentage of the salary to the salary and this percentage is based on the performance score. For example if the performance score is 3.1, then the performance service bonus is calculated by adding 3.1% of the salary to the salary.

Given below are the details of a container class named `BonusList` that manages a list of `Bonus`es. The `Bonus` class in the diagram is the same as the `Bonus` class in the UML class diagram above.



The functions of the `BonusList` class should do the following:

- The destructor must delete all `Bonus` heap objects pointed to by `Bonus*`s in its data member `bonusList`.
- `addBonus()` takes the type of bonus, salary, number of months, start date and performance score in its arguments. Based on the type of bonus, this function creates heap objects of the relevant `Bonus` subclass and adds the pointer to these objects in its data member `bonusList`.
- One `totalValue()` calculates the total value of all bonuses stored in a bonus list and the second `totalValue()` calculates the total value of all bonuses when the type is specified.
- `countBonus()` counts the number of bonuses equal to or above a certain value specified in its argument.

Implement `Bonus`, `BirthdayBonus`, `LongServiceBonus`, `PerformanceBonus` and `BonusList` as specified above. Keep the definitions and implementations in separate header and source files.

Test all these classes by creating at least two objects of every concrete class in the `Bonus` hierarchy and by adding them to a `BonusList` object. Then test `addBonus()`, both `totalValue()` functions and `countBonus()` with appropriate arguments. Display the output for each of these function invocations on the console.
You need to develop a console application and not a GUI app.

Question 4

Chapter 8, section 8.2.2.1: Exercises 1, 2 and 3 (pages 271 & 272).

Question 5

Chapter 8, section 8.8: Exercises 1 and 2 (page 282).

Question 6

Chapter 8, section 8.9: Review Questions 7, 9, 10 and 13 (page 282).

Question 7

Chapter 9, section 9.8: Exercises 1 and 2 (pages 313 & 314).

Question 8

Chapter 9, section 9.11: Review Questions 3, 4, 7 and 8 (page 325).

7.2.3 Assignment 3 for semester 2

Due date:	25 September 2018
Unique assignment no:	806313
Study material:	Chapters 10 and 11

Part A (To be submitted)

Question 1

Write a simplified text editor application with a `QTextEdit` as its central widget. Provide three menu options to format the text entered in the `QTextEdit` to make the text (1) bold (2) underline (3) italics. It will be sufficient to apply these formats to the selected text in the `QTextEdit`.

Question 2

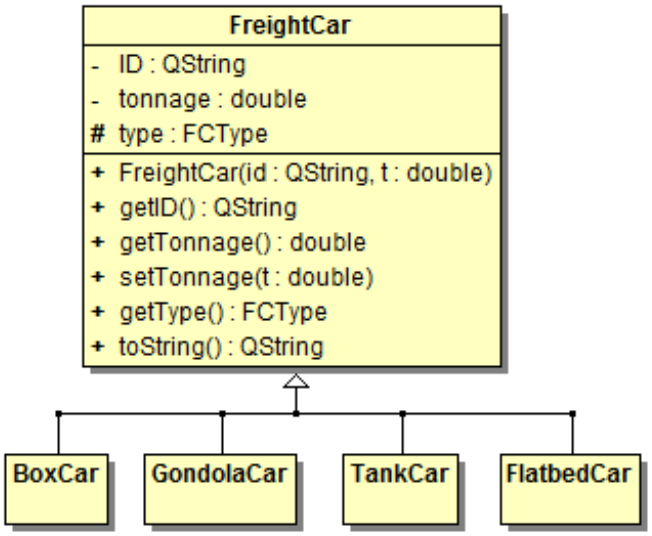
Freight trains are used to transport cargo in freight cars (i.e. goods wagons). Different types of freight cars are used for different kinds of cargo: Box cars are closed wagons that are generally used to transport packed goods, gondola cars are used to transport sand, gravel, ore, etc., tank cars are used to transport liquids, and flatbed cars are used to transport containers or other items (like motor vehicles) that can be placed or stacked on them.

An **initial** prototype system is required to model freight trains.

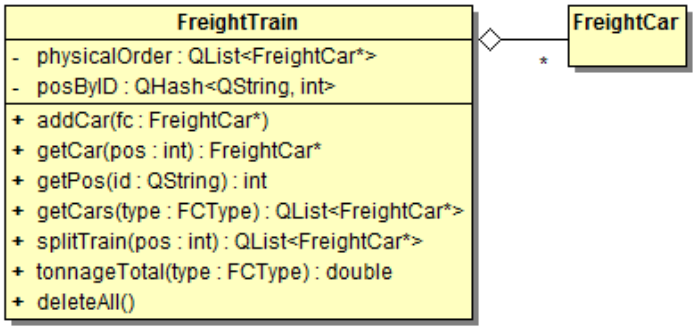
Firstly, the system must allow freight cars of different types (in particular, the four described above) to be added to a train as shown in the UML diagram on the next page.

Each freight car must have a unique ID that distinguishes it from all other freight cars. The tonnage of a freight car is the weight of the goods stored in it (measured in tonnes). At this stage of the prototype, the different types of freight cars need not have different functionality.
Implement `FreightCar`, `BoxCar`, `GondolaCar`, `TankCar` and `FlatbedCar` as specified in the UML diagram. Take note of the following:

- `FCType` is an enumeration type defined in the `FreightCar` class:
  `enum FCType {ALL, BOX, GONDOLA, TANK, FLATBED};`
- The functions `getID()`, `getTonnage()`, `setTonnage()` and `getType()` are self-explanatory.
- The `toString()` operation should return a string including the ID, tonnage and type of an instance.
- The `FreightCar` class must be defined and implemented in separate header and implementation files. The `BoxCar`, `GondolaCar`, `TankCar` and `FlatbedCar` classes, however, can be implemented simply in their respective (separate) header files.
- The constructors of the `BoxCar`, `GondolaCar`, `TankCar` and `FlatbedCar` classes should all have two parameters (like `FreightCar`) and should also set the `type` to an appropriate value. If no tonnage is supplied, a default parameter value of 0 should be used.



Secondly a freight train is modelled as follows:



A freight train must allow freight cars to be added and the positions of individual cars to be obtained by means of their IDs. It must allow a list of cars of a specified type to be obtained. It must allow the train to be split at a specified position (explained below).

Implement the `FreightTrain` class. Take note of the following:

- The `FreightTrain` class must be implemented in separate header and implementation files.
- Two lists are maintained: `physicalOrder` stores pointers to the cars in the order that they are added, and hence in the physical order that they are coupled in the train. `posByID` stores the position of cars in `physicalOrder`, allowing fast lookups according to their IDs (to prevent having to do a linear search).
- Copies of `FreightCar`s must NOT be made in any of the functions that return a pointer or a list of pointers to `FreightCar`s. The pointers returned by these functions should point to the same instances as the pointers stored in `physicalOrder`.
- The `getCar()` and `getCars()` functions should not remove `FreightCar` instances from the internal lists. `getCar()` should return a pointer to the car stored in the specified position in `physicalOrder`. `getPos()` should use `posByID` to quickly return the position of a car with the specified ID. It should return -1 if a car with the specified ID is not in the train.
- The `splitTrain()` operation, however, should remove all the `FreightCar` instances from `physicalOrder`, from the specified position up to the end of the list, and return them in a list of pointers. The function will need to regenerate `posByID` to keep it updated.
- Note the aggregation relationship between the `FreightTrain` and `FreightCar` classes. This means that the `FreightTrain` class is not responsible for the destruction of the `FreightCar` instances it is comprised of. For convenience, it provides a `deleteAll()`

function for the client program to deallocate memory for all `FreightCar`s in a `FreightTrain`.

- The `getCars()` and `tonnageTotal()` functions must return a list of cars of the specified type, and the total tonnage of all cars of the specified type, respectively. If no type is provided when these functions are called, then a default parameter value of `ALL` must be used to apply to all cars in the train.

The classes described above must be tested by a simple console application. It must create an instance of `FreightTrain` and add a number of `FreightCar`s (at least one of each type). The program must show that all the member functions of `FreightTrain` work correctly.

Part B (Self-assessment: Do not submit)

Question 3

Modify your solution to Question 1 of this assignment to incorporate toolbars in the GUI for the three edit options.

Question 4

Chapter 10, Section 10.8: Review Questions 1-12 (pages 353 & 354).

Question 5

Chapter 11, Section 11.2.1: Exercise 2 (page 364).

Question 6

Chapter 11, Section 11.7: Review Questions 1-9 (pages 376 – 377).

8 CONCLUSION

Do not hesitate to contact your lecturers 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!

COS2614 Lecturers