CSC10210 – Object-Oriented Program Development

Assignment 1 (S1 2015)

This assignment is due on Friday week 9 (17th April). You should submit your source code (C#) as a Visual C# project and UML diagrams for all parts of the assignment via e-mail. Note that marks will be deducted for poorly structured or uncommented code. All source code files submitted must include title comments that at least identify the author and the assignment part. The separate parts of the assignment are to be submitted in separate subdirectories (e.g. Part1, Part2 etc.) – submissions that ignore this instruction and leave all assignment files in one directory will be penalised.

Please note that this assignment will be due some weeks after the weekly sessions have covered the last of required materials. Do not leave this assignment to the last minute – do it while concepts are fresh in your mind. You may start Part 1 before you have studied all of the required materials. If you require an extension you must apply to your tutor before the due date to be considered. Unless an extension is approved there is a penalty (see Unit Information Guide).

Part 1 – A personal contact record class (8 marks)

This part of the assignment is for you to demonstrate your understanding C# classes by implementing a fairly simple class and a program (another class) to test it. Your task is to implement a C# class and test program that allows the programmer to save at least three contact records.

An instance of the contact record class will have the following information:

- A last name field that must not be blank
- A field to contain other names (can be blank – zero length string)
- A field to contain a company name (can be blank)
- A field to contain phone numbers as an long integer (one of phone or e-mail must contain data)
- A field to contain e-mail address (one of phone or e-mail must contain data)

The class will have:

- A method for changing each of the fields (properties are not allowed). The method should return an error status if illegal data entered (e.g. a blank last name).
- A method for returning a formatted string containing all the fields. The string should be able to be displayed on one line.
- A method for printing all of the information in the record on the Console. This method should print the information over several lines (at least two).
- A constructor should be implemented initialise the record. The constructor should check the requirements above and place the string "**error**" in bad fields, e.g. no surname or one of phone/e-mail is not present.

Your test program (class) should create an array of at least three contact records. The constructor above should be used to initialize each record. To test the methods you use at least three of the records in the array as the object reference to call the methods. All your class should be tested, i.e. every method should be called and error values tried to test the error status mentioned above.

Marks will be allocated as:

- UML class diagram for your contact record class (1 mark).
- UML Use-case diagram for whole assignment and a sequence diagram showing successful operation of part of your test program (which uses the contact record class) by adding one contact record and then changing two of the fields. (2 marks) Note that your test program will do much more than this so the sequence diagram covers only part of it.
- C# implementation of the contact record class using appropriate object-oriented structure. (4 marks)

- A test program that tests each of the above methods. In testing the methods it should demonstrate the error status of each one, for example when there is no last name or there is neither an e-mail or telephone number. (1 mark) Note that this is quite laborious but will show how well programmed your class is.

Note 1: that concepts of good testing, encapsulation, data hiding, commenting, etc. are expected to be followed and marks may be deducted if they are not.

Note 2: a comprehensive program may read data from the user to create each record instance. If you plan to do this I suggest using some hard-coded data for initial testing. It is possible to get full marks without prompting to user for data but be sure to cover all possible error conditions.

Part 2 - Multiple classes and inheritance (9 marks)

The aim of this part of the assignment is for you to demonstrate your Object-Oriented design and implementation skills using multiple classes with inheritance. Various examples were discussed in the course notes. However, doing it yourself is more difficult than reading complete designs. The assignment will only test your end design so the mental methods you use to identify your own objects for implementing your program will not be seen. The study guide has some suggested ways for approaching this, e.g. the is-a and has-a discussion in Topic 4.

The task is to develop on object-oriented program to test classes that can be used to save information about rooms in an office building. There are several types of rooms: office, kitchen, bathroom, etc. The program is to be console based.

For all rooms we will have a room identifier (string) and size in square metres (double). There will be at least three room types with the following information:

Office – number of desks and number of filing cabinets

Bathroom – number of basins and number of conveniences

Kitchen – number of people that can be seated, whether refrigerator, and/or microwave oven is present

You will need to implement a test program that adds changes and deletes room information for the three room types. The room identifier will be immutable (unchangeable) but all the other data items must have methods that can change them (even the room size). To do this your test program will need to create at least one of each type of room and call the modify methods. To make life easier, I suggest you implement a method to produce a string containing all the three main room type's data and use this method to print (dump) each record as required.

You should submit:

- One sequence diagram for creating and changing one of the room types. Note that your test program will create and change all three types so this will only cover a small part of your test program (1 mark).
- Class diagram for each of your classes (1 mark). Show the UML relationship between classes, especially the inheritance relationship. If this proves too complex for your software, show the full class diagrams and then, in a separate diagram, show the relationship using smaller class diagrams that do not show methods or all the attributes.
- Well designed object-oriented C# code that creates an instance of each of the three room classes. Demonstrate its successful use by displaying all rooms and then redisplaying after changing various attributes. (4 marks).
- Use of inheritance (2 marks) including implementation of constructors.
- Complete error-handling, including any limits you have imposed (1 mark).

Note 1: There are many ways to implement this program. You will need to identify each of the object types (classes) and then decide what attributes and methods are required. There is one obvious inheritance hierarchy here but many ways to implement it.

Note 2: There are many "good practices" through the unit study guide. You must follow them. For example, use of public/private/protected keywords will be assessed. If you use properties for this part of the assignment then you must use them properly as described in the study guide.

Part 3 - Exceptions (3 marks)

Your task here is to re-implement the contact record class from Part 1 to use exceptions. Member functions of your class and the code for testing the class will need to be re-implemented slightly. You should use exceptions as follows:

- (1 mark) Use standard (C# system) exceptions either in the class implementation or in your test program. The aim here is to correctly process standard exceptions when caught, e.g. by returning error status in a method in the contact record class or just an exception that could occur in your test program. You may have avoided exceptions in Part 1 by carefully checking of parameters. If this is the case you should remove checks and instead let the exception handling code detect the errors when they occur. Check for at least one exception, e.g. a `FormatException` could be caused if a user enters a phone number that cannot be converted to an integer.
- (2 marks) Implement an exception class called `ContactException` which will be thrown by the store class's member functions when an error occurs in the user's use of the class. The `ContactException` class should have member functions to return an integer error code (call it `code()`) and also a member function to return a string error message. The errors in field values can create two exceptions (non-blank last name, must have one of e-mail/phone number) but you may introduce other exceptions inside your contact record class (e.g. string too long, bad characters in phone numbers, bad e-mail address, etc.). However, the two error checks specified in Part 1 are a minimum and you will have to impose your own restriction to implement the other suggestions. You will need to modify your test program to catch these exceptions and print out the different error codes and associated messages. All of the errors in the use of the contact records should be reported using exceptions with different error codes and different messages.

NOTE: If you are confident in your coding you can hand in this part as an answer to Part 1 as well.