CSC10210 – Object-Oriented Program Development

Assignment 2 (S1 2015)

This assignment is due Friday 15th May. I prefer e-mail submission and you should submit your source code (C#) 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 must 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 after the suggested study timetable has covered the last of required materials. Do not to leave this assignment to the last minute. Unless an extension is approved there is a penalty (see Unit Information Guide).

Part 1 – Interfaces (7 marks)

This part of the assignment allows you to test your understanding of interfaces and their associated polymorphism.

A University has a large number of students that are identified by Student IDs to uniquely identify students. So for example, students with the same names will not be mixed up.

To allow all students to be processed by a C# program, each object representing a student needs to implement an interface with the following member functions:

```
int StudentIdentifier(); // the student ID
string FamilyName();       // return family name
string OtherNames();       // return other names (if any)
```

Write and test the following:

a. (1 marks) Write an interface called `IStudent` to allow common (polymorphic) processing of all classes that choose to implement this interface.

b. (2 mark) Implement at least two types of student objects (two classes) that implement the `IStudent` interface. For example, one may be `StudentCourse` class which contains additional information such as a course name. Others may be `StudentResult`, which contains a unit name and grade (e.g. "HD", "P", etc.) Don't make these too complicated. Just one or two attributes for each class is fine.

c. Implement and test a class called `StudentDisplay` which has the following five methods, each of which takes a parameter objects implementing the `IStudent` interface:

- (1 marks) Three methods for *printing* each of the student ID, the family name and other names.
- (1 marks) A method for *returning* all three values concatenated into a formatted string.
- (1 marks) A Boolean method with two parameters that checks if a student object (1st parameter) has the same student ID as a given integer (2nd parameter).

Part 2 – Collections and LINQ (9 marks)

This part of the assignment allows you to practice C# collections and the LINQ interface. It uses a simple object type whose class (called `Room`) is attached to this assignment. Note that this class has intentionally been made very simple and a real implementation would check things like the room identifier and sizes fields for reasonable values. You must use the from-select statements covered in the study guide. Lambda expressions like those you will find on the Internet are not permitted!

a. Implement a Dictionary collection of `Room` objects whose key will be the room identifier attribute. Initialise the Dictionary with at least 10 rooms. (You can create new room objects in your program or you may find it easier to implement code to read data from a file). (2 marks)

b. Implement a *method* to printout the details of all of the rooms in the collection (in the order they are stored). (1 marks)

c. Implement a method to prompt the program user for a room identifier and delete it from the collection if it exists. Your method should print a message as to whether the record was successfully deleted or not (due to incorrect room identifier). (1 marks)

d. Implement a method that prompts the user for a room identifier and displays all the room details for that room (or an error message if the room is not in the collection). (1 mark)

e. Implement a method to print the list of rooms in ascending identifier order. (1 mark)

f. Implement a method to list all room details in ascending size order. (1 mark)

g. Implement a method to list all rooms that seat between 10 and 20 people (inclusive) (1 mark)

h. Implement a method that lists all room `ID` and `Size` in the order of descending seating numbers (do not display the occupants field). (1 mark)

For this part of the assignment you must use LINQ to search the collection. Note that a `Dictionary<>` object has member functions for extracting the keys and data values as a separate collections so you do not have to program this yourself.

Part 3 – Operator overloading (4 marks)

This part of the assignment allows you to practice C# operator overloading.

Implement a class called `Money` for representing Australian currency in dollars and cents. Your class should have a constructor for establishing a money object specifying dollars and cents; and a second constructor specifying dollars only. For example, the follow two declarations will establish an object representing 23 dollars and 17 cents and also an object representing $100.

```
Money cost = new Money(23,17);
Money bill = new Money(100);
```

Your class should implement methods (or properties) for retrieving the dollars and cents separately. You should also implement a `ToString()` method to return a string representation of the money using the customary notation, e.g. "$23.17". This will also be useful for debugging and testing.

a. Implement the above class (1 mark). Through an `Exception` or some other notification, notify the programmer if illegal values are used in the cents parameter in the constructor, i.e. cents less than zero or greater than 99. There should be no way for the cents value stored in the class to be outside the range 0 to 99. (1 mark)

b. Overload the '`==`' and '`!=`' operators to allow two different money objects to be compared. Demonstrate both with simple examples (1 mark).

c. Overload the '`<`' and '`<=`' operators to allow two different money objects to be compared. Note that C# will also require you to overload the '`>`' and '`>=`' operators (0.5 mark). Demonstrate.

d. Overload the '+' and '-'operators to allow two `Money` objects to be added or subtracted. Note that these must make sure the cents value stays between 0 and 99. Demonstrate these operators. (1 mark)

e. Overload the "*" operator to allow a `Money` objects to be multiplied by a single integer. Make sure cents stay in range and demonstrate the operator use. (0.5 marks)

```
// simple class to represent University room data - Part 2 of Assignment

class Room
{
    private String ident;     // e.g. "MG.31"
    private String usage;     // e.g. "staff", "kitchen"
    private int size;         // size in square metres
    private int occupants;    // number of people usually accomodated.

    public Room(string id, string use, int s, int occ)
    {
        ident = id;
        usage = use;
        size = s;
        occupants = occ;
    }
    public string ID        // identifier is immutable
    {
        get
        {
            return ident;
        }
    }
    public string Usage
    {
        get
        {
            return usage;
        }
        set
        {
            usage = value;
        }
    }
    public int Size
    {
        get
        {
            return size;
        }
        set
        {
            size = value;
        }
    }
    public int Occupants
    {
        get
        {
            return occupants;
        }
        set
        {
            occupants = value;
        }
    }

    public override string ToString()
    {
        return string.Format("{0}: {1}, {2}sq. Metres, seats {3}",
                             ident, usage, size, occupants);
    }

}
```