Assessment Brief Proforma

1. Module number	*SET08119*
2. Module title	*Object Oriented Software Development*
3. Module leader	*Neil Urquhart*
4. Tutor with responsibility for this Assessment Student's first point of contact	*Neil Urquhart* E: n.urquhart@napier.ac.uk T: 0131 455 2655
5. Assessment	*Practical*
6. Weighting	*20%*
7. Size and/or time limits for assessment	*Approx 10 hours of work and a demonstration prior to submission*
8. Deadline of submission	Demonstrations must be made by the end of week 8, (the arrangements for demonstration will be confirmed by your tutor) submissions must be handed in by the end of week 8. Your attention is drawn to the penalties for late submissions
9. Arrangements for submission	*A .zip archive containing your Visual Studio solution should be uploaded into Moodle. You should also upload a .PDF file containing screenshots of your form and the C sharp listing of your person class.* *You are advised to keep your own copy of the assessment.*
10. Assessment Regulations	All assessments are subject to the University Regulations

11. The requirements for the assessment	Reassessment will be in the form of a further practical exercise.
12. Special instructions	None
13. Return of work and feedback	*Feedback will be given at the 1:1 demo session. Cohort feedback will be given after the final submission date. A further 1:1 session will be arranged for any student who requires it.*
14. Assessment criteria	*See attached,*

Object Oriented Software Development:
Coursework 1

This coursework is designed to test your ability to construct simple GUIs, using C#, to implement classes in C# and your knowledge of basic OO theory.

Module Management System – Specification

We are going to create a simple system for managing a list of students who are enrolled upon a module.

Students have the following properties:

Property	Example Data	Validation
Matric number	10001	In the range 10001 to 50000
First name	Fred	Not blank
Surname	Richardson	Not blank
Coursework Mark	15	In the range 0 to 20
Exam Mark	34	In the range 0 to 40
Date of Birth	5/10/77	Not blank

You should pick a suitable data type for each property, make sure that the property is declared as private and accessed using either C sharp properties or Java style get/set methods.

Your Student class should contain the following methods:

Method	Notes
getMark()	Should return the overall module mark as a % based on the coursework and exam components which are weighted as 0.5/0.5 EG: Coursework =22 Exam = 33 Mark = 78%

Supplied code

Along with this document you may download a Visual Studio solution that you should use to construct your coursework. If you have any problems with this solution, please contact your tutor. A basic demonstration will be given in the lecture.

The Solution contains two projects, called Presentation, which represents our Presentation Layer and Business, which represents the other layers. The projects

have been setup so that that Presentation can make use of classes contained in the Business project.

In the business project you will find two classes, Student and ModuleList. The Student class is incomplete, you will complete it as part of this coursework (see below). The ModuleList class stores Student classes –it might connect to a database or other means of persistence, but you do not need to worry about how it works.

ModuleList has the following methods/properties:

> Add(Student s) : Adds the Student object *s* to the module list
> Find(int matric) : Searches the students in the module and returns the student with a matching matric number. If the student does not exist it returns null.
> Delete(int matric)boolean: Removes the Student object s from the module list. Returns true if the student is found and deleted, else false.
> Matrics : Returns a List object that contains the matric numbers of the students enrolled upon the module. *Note you only need to use this method if you attempt the advanced tasks.*

(Note that ModuleList is not persistent, it will be empty each time you start your project.)

Presentation contains a single form, which you should use as your main window (see tasks). It contains a private instance of ModuleList named s*tore*, therefore you can write code such as this within your event handlers:

```
Student aStudent = new Student();
store.Add(aStudent);
```

Coursework Tasks

1. Add the missing properties to the Student class.

2. Appropriate controls to the form to allow all of the Student properties to be displayed/entered.

3. Create an “Add” button that creates a new Student object based on the data in the form. If the data is valid, then the Student should be added to the store. The controls on the form should then be cleared.

4. Create a “Find” button that will take the matric number entered into the form and use store.find() to see if that student exists and then display their details on the form.

5. Create a “Delete” button button that will take the matric number entered into the form and use store.delete() to remove that student from the module.

Optional Advanced tasks….

6. When a new Student is added, have the matric number generated automatically, starting at 10001 and incrementing by 1 for each student.

7. Add a list box to your form that contains the matriculation numbers of all enrolled students. Selecting a matric number should display the details of that student on the form.

8. Create a button “List all”, which opens a second window that contains details of all students including there total mark calculated using Student.getMark()

Coding standards

Your code and repository will be reviewed as part of the marking process. Please adhere to the following:

1. All classes should have comments at the top to note:
    a. Author name
    b. Description of class purpose
    c. Date last modified

2. Methods should all have a comment to describe their purpose

3. Properties should all have a comment to describe their purpose

4. All code should be indented as appropriate. For example:

```
   …
for (int counter = 1; counter <= 1000; counter++)
{
    if (counter == 10)
       break;
    Console.WriteLine(counter);
}
   …
```

5. Use descriptive variable and method names (.e.g. no use of ‘x’ or ‘y’!)

6. Make sure that your code is written in a succinct fashion (i.e. no unnecessary code.)

Demonstration

When demonstrating you must have a copy of the demonstration sheet printed out, this will be filled in during the demonstration.

Submission

A .zip archive containing your Visual Studio solution should be uploaded into Moodle. You should also upload a .PDF file containing screenshots of your form and the C sharp listing of your person class.

You are advised to keep your own copy of the assessment.

MARKING SHEET (TO BE FILLED IN DURING THE DEMONSTRATION)

NAME_____ **MATRIC** _____

Item	Mark	Notes
Layout of form	/2	1 mark all controls present, and 1 mark for neatness of layout.
Validation	/2	1 Mark partial attempt. or 2 Marks fully implemented.
Find	/2	1 Mark partial attempt. or 2 Marks fully implemented.
Delete	/2	1 Mark partial attempt. or 2 Marks fully implemented.
Auto Matric Numbers (advanced)	/2	1 Mark partial attempt. or 2 Marks fully implemented.
List names (advanced)	/4	2 Marks for list, and 2 marks for displaying details by clicking on the list.
Full class list (advanced)	/4	1 Mark presentation of list. and 2 Marks details on list. and 1 Mark students' total % displayed.

Additional Notes:

OVERALL _____/18

CODE/DESIGN REVIEW SHEET
(TO BE FILLED IN BY THE MARKER AFTER SUBMISSION)

NAME_____ **MATRIC** _____

Please note in situations where the demonstration suggests that basic functionality has not been achieved, marks in this section may be capped at the dissertation of the module leader.

Item	**Mark**	**Notes**
Appropriate data types	/2	Full marks requires date of birth to be a DateTime object
Accessors	/2	Full marks requires private types, accessed via get/set methods or properties.
Comments	/2	Full marks requires frequent comments.
Validation using try catch (advanced)	/4	Full marks requires try/catch on all properties.

OVERALL
_____/10

Total marks

From demo_____ /18
From review _____/10
Total_____ /28

Appendix 1 – Validation of properties using exceptions.

Suppose we have a property within a class called id:

```
private int m_id = -1;

        public int id
        {
            get
            {
                return m_id;
            }
            set
            {

               m_id = value;
            }
        }
```

Suppose we wish to validate it by restricting id to values of less than 100, we can do this by throwing an error:

```
private int m_id = -1;

        public int id
        {
            get
            {
                return m_id;
            }
            set
            {
                if (value > 100)
                {
                    throw new ArgumentException("Too Large!");
                }
                m_id = value;
            }
        }
```

In this case the **throw** statement generates a new error of type ArgumentException. When the throw statement is encountered the error is passed back to the calling method, *note that this means that the statement m_id = value; has not been reached and so m_id has not been changed*.

Because this property might now throw an error the code used to set it must be able to handle an error, so we use the **try{ … } catch** keywords:

```
            try
            {
                s.id = 101;
            }
            catch (Exception excep)
            {
                Console.WriteLine(excep.Message);
            }
```

If any statement within the try block causes an error (and the above example does!) then control jumps to the catch block. Note that execp.Message contains “Too Large” as we specified in the **throw** statement.