Edinburgh Napier
UNIVERSITY

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 (2nd Nov). 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 GUI (running) and the C sharp listing of any classes written.* *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.

Customer Management System – Specification

We are going to create a simple system for managing a list of customers of a business. The system is to be used to keep track of their contact details.

Customers have the following properties:

Property	**Example Data**	**Validation**
ID	10001	In the range 10001 to 50000.
First name	Fred	Not blank
Surname	Richardson	Not blank
Email address	Fred@napier.ac.uk	
Skype ID	fredSkype1978	
Telephone	+00 (44) 131 455 2655	Not blank
Preferred Contact	email or skype or tel	Must be “email”, “skype” or “tel”

You should pick a suitable data type for each property, make sure that the property is declared as private and accessed using C sharp properties.

Your Student class should contain the following methods:

Method	**Notes**
getPreferredContact()	Should return a String with either the ‘phone number, SkypeID or email depending on the setting stored within the preferred contact. The string should have “Skype:”, “Email:” or “Tel:” at the start.

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, Customer and MailingList. The Customer class is incomplete, you will complete it as part of this coursework (see below). The MailingList class stores Customer objects –it might connect to a database or other means of persistence, but you do not need to worry about how it works.

MailingList has the following methods/properties:

Add(Customer s) : Adds the Customer object s to the mailing list
Find(int id) : Searches the customers in the list and returns the Customer with a matching id number. If the Customer does not exist it returns null.
Delete(int id): Removes the Customer object from the mailing list
names : Returns a List object that contains the id numbers of the customers in the list. *Note you only need to use this method if you attempt the advanced tasks.*

(Note that Mailing List 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 MailingList named s*tore*, therefore you can write code such as this within your event handlers:

```
Customer aCustomer = new Customer();
store.Add(aCustomer);
```

Coursework Tasks

1. Add the missing properties to the Customer class.

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

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

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

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

Optional Advanced tasks….

6. When a new Custpmer is added, have the id number generated automatically, starting at 10001 and incrementing by 1 for each Customer.

7. Add a list box to your form that contains the id numbers and names of all customers. Selecting an id number should display the details of that customer on the form.

8. Create a button "List all", which opens a second window that contains details of all customer including their preferred means of contact.

9. Undertake advanced validation, email addresses must contain '@'.

Coding standards

Your code 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. Use try/catch to throw exceptions when validating the form

7. 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	/3	1 Mark partial attempt. or 2 Marks fully implemented. 3 Marks, uses exceptions
Find	/2	1 Mark partial attempt. or 2 Marks fully implemented.
Delete	/2	1 Mark partial attempt. or 2 Marks fully implemented.
Auto increment ID numbers (advanced)	/2	1 Mark partial attempt. or 2 Marks fully implemented.
List ids (advanced)	/4	2 Marks for list, and 2 marks for displaying details by clicking on the list.
Advanced validation (advanced)	/2	2 Marks for rejecting any email address that does not contain @
Full customer list on separate form(advanced)	/4	1 Mark presentation of list. and 2 Marks details on list. and 1 Mark customers' preferred contact displayed.

Additional Notes:

OVERALL ______/21

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	
Accessors	/2	Full marks requires private types, accessed via properties.
Comments	/2	Full marks requires frequent comments.
Validation using try catch (advanced)	/4	Full marks requires try/catch on all properties.

OVERALL
_____/10

<u>Total marks</u>

From demo_______ /21
From review _______/10
Total_______ /31

Appendix 1 – Validation of properties within classes 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.