***DOC\_174\_REV\_C\_ASSESMENT AND IV FRONT SHEET\_INDIVIDUAL***

***CRITERIA\_CLASSTER.docx***

**…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………**

**ASSESSMENT AND INTERNAL VERIFICATION FRONT SHEET (Individual Criteria)**

# (Note: This version is to be used for an assignment brief issued to students via Classter)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Course**  **Title** |  | **B.Sc. (Hons.) Software Development** | | Lecturer Name & Surname | **Ryan Attard**  **Laurent Azzopardi**  **David Debono** | | |
| **Unit Number & Title** | | **ITSFT-506-2006 Object Oriented Programming** | | | | | |
| **Assignment Number, Title / Type** | | **3, Syntopic / Home** | | | | | |
| **Date Set** | | 4/12/2023 | **Deadline Date** | **4/12/2023** | | | |
| **Student Name** |  | Samuel Sammut | **ID Number** | **0114105L** | | **Class / Group** | **6.1A** |

|  |  |
| --- | --- |
| **Assessment Criteria** | **Maximum Mark** |
| *KU1.1: Explain how Object-Oriented structures such as classes and objects are used in an Object-Oriented application* | 5 |
| *KU1.2: Identify proper use of attributes, behaviours and access modifiers.* | 5 |
| *KU1.3: Indicate where and how encapsulation is implemented in an object-oriented application.* | 5 |
| *KU1.5: Recognise where abstraction is required in a given scenario.* | 5 |
| *KU2.1: Distinguish between the different class relationships applied in an application.* | 5 |
| *AA2.3: Apply proper constructors to classes having different types of class relationships* | 7 |
| *AA4.1: Design an Object-Oriented application using UML to depict any implemented classes as well as any existing relationships.* | 7 |
| *SE2.2: Assess which class relationships need to be implemented for a given scenario.* | 10 |
| *AA3.3 - Use LINQ to implement persistent data storage in an object-oriented application.* | 51 |
| **Total Mark** | 49 |

|  |
| --- |
| **Notes to Students:** |
| * This assignment brief has been approved and released by the Internal Verifier through Classter.      * Assessment marks and feedback by the lecturer will be available online via Classter [(Http://mcast.classter.com)](http://mcast.classter.com/) following release by the Internal Verifier      * Students submitting their assignment on Moodle/Turnitin will be requested to confirm online the following statements:     **Student’s declaration prior to handing-in of assignment**   * + I certify that the work submitted for this assignment is my own and that I have read and understood the respective Plagiarism Policy     **Student’s declaration on assessment special arrangements**   * + I certify that adequate support was given to me during the assignment through the Institute and/or the Inclusive Education Unit.   + I declare that I refused the special support offered by the Institute. |

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ MCAST Controlled and approved document Unauthorised copying or communication strictly prohibited***

# Assignment Guidelines

Read the following instructions carefully before you start the assignment. If you do not understand any of them, ask your invigilator.

* This assignment is a synoptic (take-home) assignment.

* Fill in and print/scan the assignment sheet.

* Copying is **Strictly Prohibited** and will be penalised according to disciplinary procedures.

* Deadline: See front page

* This assignment has a total of 100 marks. Students who have a partial synoptic need to answer only the questions in Part 2. Those who have the full synoptic need to answer both Part 1 and Part 2

* Deadline is 25th February.

* Submission must be done by inputting your Git repository link or VLE https://vle.mcast.edu.mt/mod/assign/view.php?id=49532

# Part 1 (51 marks)

## AA3.3: Use LINQ to implement persistent data storage in an object-oriented application. (51 marks)

Requirements: To be able to answer Part 1, you need to create a database with two tables, create an ORM model and create a class which will use the classes representing the tables in your database.

Tables needed:

|  |  |
| --- | --- |
| Appointments |  |
| **Id** | Int, identity, not null |
| **Title** | Nvarchar(200), not null |
| **Description** | Nvarchar(max), not null |
| **Tag** | Int, foreign key, not null |
| **Date** | Datetime, not null |

|  |  |
| --- | --- |
| Tags |  |
| **Id** | Int, identity, not null |
| **Title** | Nvarchar(200), not null |

Note: The Tags table should be manually populated with 3 rows: one for work, one for school and one for life;

After implementing the above, you need to code the following methods in a class which you may call AppointmentRepository. Paste the code of each method under each task:

A close-up of a computer screen

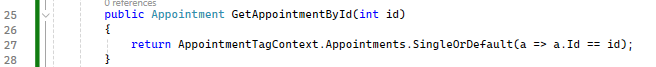
Description automatically generated

1. AddAppointment(Appointment a) *//adds appointment to the database permanently [5]*

A screenshot of a computer

Description automatically generated

1. GetAppointmentById(int id) *//gets ONE appointment by id or null if it does not exist [5] <*

**

1. GetAppointments(bool past) //gets all past appointments or upcoming ones depending on the Boolean value passed. If its in the past they should be sorted by the most recent one first, if its in the future sorted by the closest one first; [7]

### A computer screen shot of a computer code Description automatically generated

*4.* SearchAppointment(string keyword); *//searches the appointment by any word which is stored in the title or description [5]*

*A close-up of a computer screen

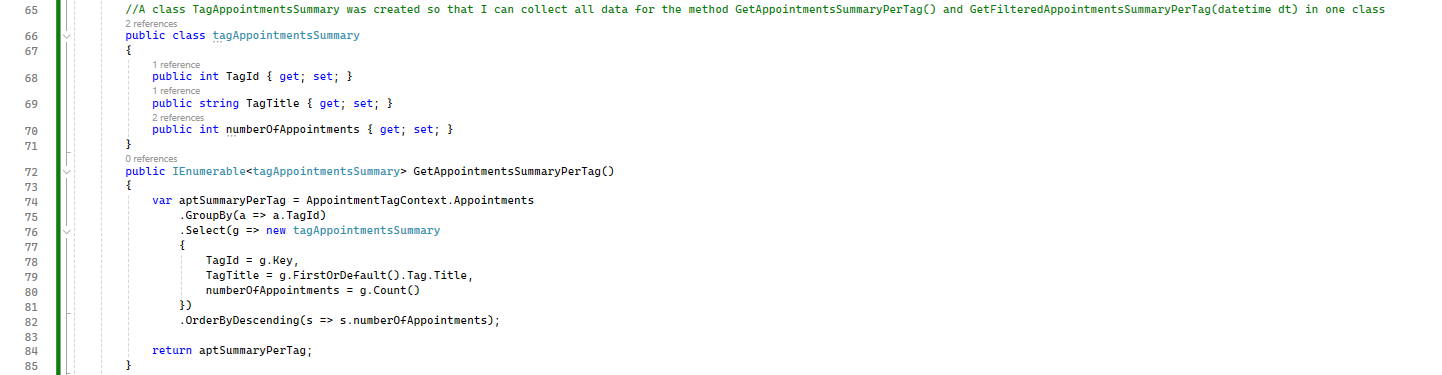
Description automatically generated*

1. *A screenshot of a computer program

   Description automatically generated*DeleteAppointment(int id); *//Deletes appointment by id permanently [5]*
2. GetAllAppointmentsByTag (int id) *//gets all the appointments by tag[5] A screenshot of a computer

   Description automatically generated*

1. GetAppointmentsSummaryPerTag() *//gets Tag Title, Number of appointment per tag, sorted by the tag which has most appointments [7]*



1. GetFilteredAppointmentsSummaryPerTag(datetime dt) *//gets Tag Title, Number of appointment per tag, sorted by the tag which has most appointments but also filtered by date only (excluding time) [7]*

A computer screen shot of a computer code

Description automatically generated

1. UpdateTagOfAppointment (int appointmentId, int newTag); *//update tag for appointment [5]*

*A computer screen shot of a computer code

Description automatically generated*

# Part 2 (49 marks)

|  |
| --- |
| KU1.1: Explain how Object-Oriented structures such as classes and objects are used in an Object-Oriented application. (5 marks) |
| KU1.2: Identify proper use of attributes, behaviours and access modifiers. (5 marks) |
| KU1.3: Indicate where and how encapsulation is implemented in an object-oriented application. (5 marks) |

KU1.1 - Create a class called *Calendar* with fields *Title (string), StartDate (date), User (string), Appointments*

*(List of Appointment class from KU2.1)* [2.5]. Create an instance of calendar *myCalendar* in *Program*.cs; [2.5];

KU1.2 - While appointments list in KU1.1 should be kept hidden to external access, add a behaviour which allows external access to allow the addition of new appointments and add a validation to check whether date of appointment is in the future [2.5]. Add a constructor to allow the assignment of a Title value [2.5];

KU1.3 Encapsulate the fields User [1], Title such that this is made read-only [2], and StartDate where StartDate has to be in the past; if not throw an Exception “Invalid Date” [2];

using System.Collections.Generic;

using System;

public class Appointment

{

private List<Appointment> appointments = new List<Appointment>();

public Appointment(string title, DateTime date)

{

Date = date;

Title = title;

}

public string Title { get; set; }

public DateTime Date { get; set; }

public int Id { get; set; }

public int TagFk { get; set; }

//Mehtod to add new Appintmens

public void addNewAppointment(Appointment a)

{

if (a.Date > DateTime.Now)

{

appointments.Add(a);

}

else

{

Console.WriteLine("You have enetered a date in the past and therefore the appointment could not be created");

}

}

}

public class Calendar

{

private string user;

private string title;

private DateTime startDate;

public string User

{

get { return user; }

private set { user = value; }

}

public string Title

{

get { return title; }

private set { title = value; }

}

public DateTime StartDate

{

get { return startDate; }

private set

{

if (value > DateTime.Now)

{

throw new Exception("Invalid Date");

}

else

{

startDate = value;

}

}

}

//Program.cs

public class Program : Calendar

{

static void Main(string[] args)

{

Calendar calendar = new Calendar();

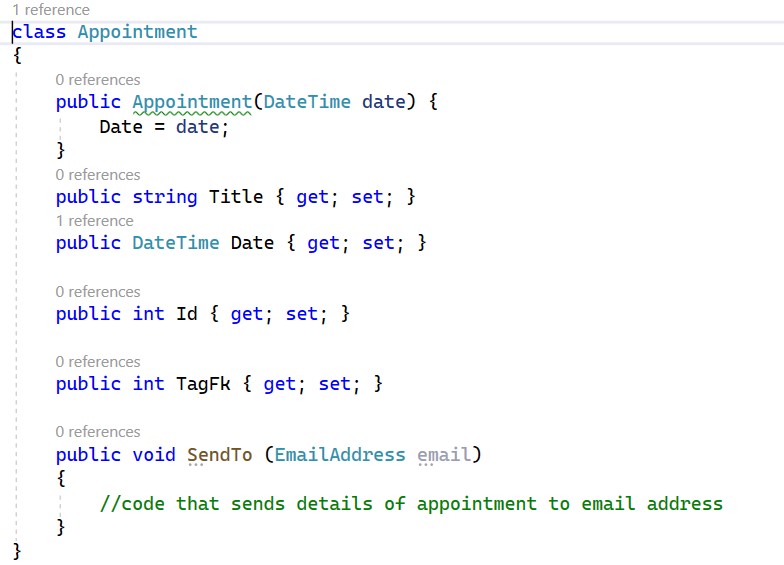
}

}

}

## KU2.1: Distinguish between the different class relationships applied in an application. (5 marks)

Look at this code and identify 2 class relationships, mentioning the class names and the type of relationship



|  |  |  |
| --- | --- | --- |
|  | Relationship name (1 each) | Classes related (1.5 each) |
| 1 | Association (Many – to – Many) | Appointment and Tag |
| 2 | Aggregation (One – to – Many) | Appointment and Email |

## SE2.2: Assess which class relationships need to be implemented for a given scenario. (10 marks)

In the example given to you as screenshots in KU2.1,

1. Change Appointment class to hold more than one Tag instance; you have to create a method that allows you to add as many tags as you want. Implement this. [3]
2. Name the relationship between Appointment and Tag that was implemented in (a):

Aggregation (one – to – many) [2]

1. Introduce and implement a new class in the example above and relate it to one of the existent classes above by means of Composition. [5] Paste the final code here.

using System;

namespace SE2.\_2

{

class Appointment

{

public static void main(string[] args) { }

public Appointment(DateTime date)

{

Date = date;

Details = new furtherAppointmentDetails();

}

public string Title { get; set; }

public DateTime Date { get; set; }

public int Id { get; set; }

public furtherAppointmentDetails Details { get; }

public void AddTag(Tag tag)

{

Tag.Add(tag);

}

}

public class furtherAppointmentDetails

{

public string Description { get; set; }

public string Location { get; set; }

}

//I added the class furtherAppointmentDetails to enable a composition relationship

}

public class Tag

{

public int ID { get; set; }

public string Title { get; set; }

}

public class EmailAddress

{

public string Title { get; set; }

public string Recipient { get; set; }

public string FullName { get; set; }

}

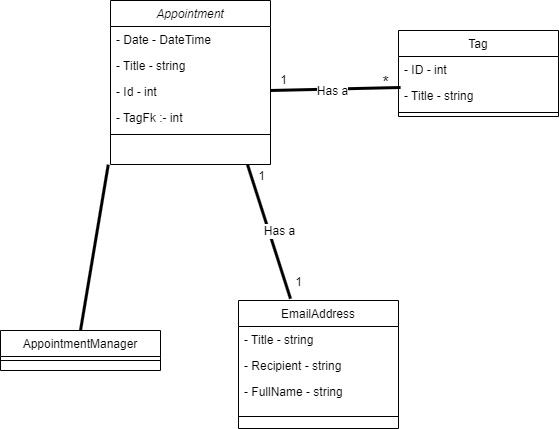
}

## AA4.1: Design an Object-Oriented application using UML to depict any implemented classes as well as any existing relationships. (7 marks)

Use Visio or www.draw.io to draw a UML class diagram and design the classes as presented in Task KU2.1 [**not** SE2.2], clearly showing the relationships. Copy and paste the image depicting your UML class diagram here. [7 marks]

Hint: you must draw 4 classes & 3 relationships

*Marking: 1 mark for each class, 1 marks for each of the relationships*



## AA2.3: Apply proper constructors to classes having different types of class relationships (7 marks)

You are asked to create an inheriting class called: *AppointmentWithNotification*. This class should inherit from Appointment however it contains an added property called Recipient of type EmailAddress. You are to come up with an efficient constructor (which has minimal code possible) where it requires the passing of a date and an instance of EmailAddress. The constructor should make sure that the date is set accordingly and effectively and also make sure that it sends the notification of the appointment immediately.

Paste code here of the entire *AppointmentWithNotification* class

[2] – for syntax and notation

1. – for constructor signature using the *base* keyword
2. – for using implementing it correctly

class notificationOfAppointment : Appointment

{

public EmailAddress Recipient { get; set; }

public notificationOfAppointment(DateTime date, EmailAddress recipient) : base(date)

{

Recipient = recipient;

sendNotification();

}

private void sendNotification()

{

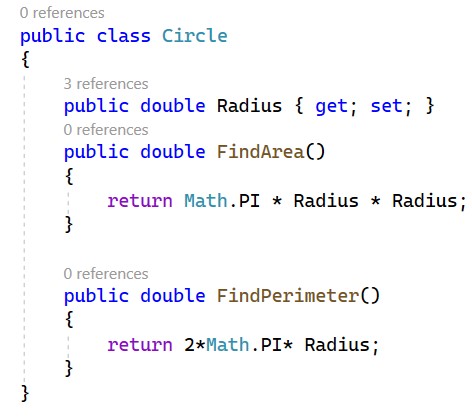
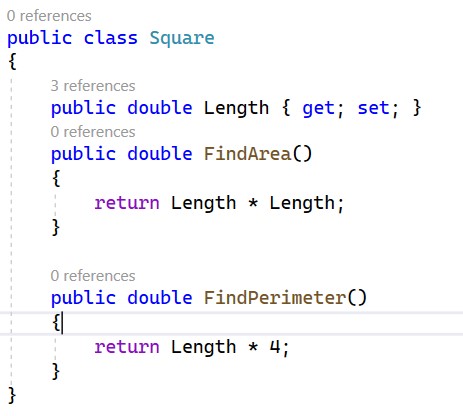
Console.WriteLine("A Notification has been sent to " +Recipient.Recipient +" for the appointment:"+Title + " which is going to be held on the " +Date);

}

}

## KU1.5: Recognise where abstraction is required in a given scenario. (5 marks)

Consider the below two classes *Circle* and *Square.* Create an abstract class that can be inherited from *Circle* and *Square*.



Paste the code of the entire abstract class below

1. – for syntax and notation
2. – per correct member of the abstract class

using System;

namespace KU

{

public abstract class Shapes

{

public abstract double FindArea();

public abstract double FindPerimeter();

}

public class Circle : Shapes

{

public double Radius { get; set; }

public override double FindArea()

{

return Math.PI \* Radius \* Radius;

}

public override double FindPerimeter()

{

return 2 \* Math.PI \* Radius;

}

}

public class Square : Shapes

{

public double Length { get; set; }

public override double FindArea()

{

return Length \* Length;

}

public override double FindPerimeter()

{

return Length + Length + Length + Length;

}

}

}

Rubric:

|  |  |  |  |
| --- | --- | --- | --- |
| **AA3.3** | • | AddAppointment(Appointment a) [5] |  |
|  | • | GetAppointmentById(int id) [5] |  |
|  | • | GetAppointments(bool past) [7] |  |
|  | • | SearchAppointment(string keyword); [5] |  |
|  | • | DeleteAppointment(int id); [5] |  |
|  | • | GetAllAppointmentsByTag (int id) [5] |  |
|  | • | GetAppointmentsSummaryPerTag() [7] |  |
|  | • | GetFilteredAppointmentsSummaryPerTag [7] |  |
|  | • | UpdateTagOfAppointment [5] |  |
| **KU1.1/**  **KU1.2/**  **KU1.3** | • | KU1.1 - Create a class called Calendar with fields Title (string), StartDate (date), User (string), Appointments (List of Appointment class from KU2.1) [2.5]. Create an instance of calendar myCalendar in Program.cs; [2.5]; |  |
|  | • | KU1.2 - While appointments list in Ku1.1 should be kept hidden to external access, add a behaviour which allows external access to allow the addition of new appointments and add a validation to check whether date of appointment is in the future [2.5]. Add a constructor to allow the assignment of a Title value [2.5]; |  |
|  | •  • | KU1.3 Encapsulate the fields User [1], Title such that this is made read-only [2], and StartDate where StartDate has to be in the past; if not throw an Exception  “Invalid Date” [2]; |  |
| **KU2.1** | • | 2 Valid Relationship Name [1, 1] |  |
|  | • | 2 Valid Class “names” in relationship mentioned in the same task [1.5,1.5] |  |
| **SE2.2** | • | Valid method that allows the addition of many Tag instances [3] |  |
|  | • | Correct relationship between Appointment and Tag [2] |  |
|  | • | Class-example showing comprehension of Composition. [5] |  |
| **AA4.1** | • | 1 mark for each class, 1 mark for each of the relationships |  |
| **AA2.3** | • | [2] – for syntax and notation |  |
|  | • | [2] – for constructor signature using the base keyword |  |
|  | • | [3] – for using implementing it correctly |  |
| **KU1.5** | • | [1] – for syntax and notation |  |
|  | • | [2] – per correct member of the abstract class |  |