

**Conestoga College - CAD Programs**  
**Programming Microsoft Web Technologies**  
**PROG8551 - 24F - Section 3**  
**Problem Assignment 3**  
**Total Marks: 100**

## **Introduction**

The goal of this assignment is to build on the skills you have been developing around database-driven ASP.NET Core MVC web apps. We will extend the model used in Problem Assignment 2, however the resulting model would be somewhat more complex. Model will include several 1-to-1, 1-to-many, and many-to-many relationships. You have some freedom to develop this ASP.NET Core MVC web app. Some highlights of this assignment are:

- Applying Data-First approach for model building
- Applying Code-First approach for model building
- Deploy a xUnit project to run unit tests against business logic
- Apply LINQ and EF Core to a rich data model
- Apply a **paging** solution by clustering records into groups

There will be some time to work on these assignments in class and, if necessary, I can offer hints if I see that you are struggling with certain parts.

## **What/How to Submit?**

Zip up your entire solution into **one zip file** and submit that file to the eConestoga dropbox for the assignment. You can submit multiple solutions but only the latest (i.e. most recent) one will be looked at and evaluated.

## **A Word of Caution - Again**

Make sure you do your own work and do not copy code from any other source. Every solution will be run through [MOSS](#) to check for academic integrity violations. There is zero-tolerance for such violations and any encountered will be dealt with in accordance with [Conestoga's policy](#).

## **The APP**

In this assignment, you are to develop an app that manage programs, students, courses, and tuition invoices in a College. It can display records in alphabetical groups, and let users add or edit records with validation requirements. It can show more details about a Student including its tuition invoices. Users would be able to add a new line item

to an invoice. For each invoice, app can show both the total and the line items of the invoice.

## Part A - Database Setup and Mapping - (20 marks)

Design an SQL database, named **CollegeDB**, by using MS SQL Manager or MS Visual Studio. Develop the database. The database must have the following tables and records. Further, establish relationships between tables as required below.

(Strictly follow the names given here for the database, tables and columns)

### Task A1 - (10 marks)

To create tables, you can use the SQL scripts given in the file "Create Queries.txt".

- Create **TABLE Provinces** with columns:
  - ID (**INT**) - auto-increment from number 1 - NOT NULL
  - Name (**NVARCHAR(30)**) - NOT NULL
  - Code (**NVARCHAR(2)**) - NOT NULL
- Create records in **TABLE Provinces** for all Canadian provinces. Use the **INSERT** statements given in the file "Insert Query Provinces.txt".
- Create **TABLE Cities** with columns:
  - ID (**INT**) - auto-increment from number 1 - NOT NULL
  - Name (**NVARCHAR(30)**) - NOT NULL
  - ProvinceID (**INT**) - NOT NULL
- Use **ProvinceID** as the foreign key to establish a one-to-many relationship between **Provinces** and **Cities**.
- Create records in **TABLE Provinces** for all Canadian cities. Use the **INSERT** statements given in the file "Insert Query Cities.txt".
- Create **TABLE StudentTypes** with columns:
  - ID (**INT**) - auto-increment from number 1 - NOT NULL
  - Type (**NVARCHAR(30)**) - NOT NULL
- Use the following **INSERT** statements to create student types:  
**INSERT INTO StudentTypes (Type) VALUES ('DOMESTIC')**  
**INSERT INTO StudentTypes (Type) VALUES ('INTERNATIONAL')**
- Create **TABLE Terms** with columns:

- ID (**INT**) - auto-increment from number 1 - NOT NULL
- Semester (**NVARCHAR(20)**) - NOT NULL
- Use the following **INSERT** statements to create records for terms:
 

```
INSERT INTO Terms (Semester) VALUES ('FALL')
INSERT INTO Terms (Semester) VALUES ('SPRING')
INSERT INTO Terms (Semester) VALUES ('WINTER')
```
- For an under-graduate program, create **TABLE UGPrograms** with columns:
  - ID (**INT**) - auto-increment from number 1 - NOT NULL
  - Code (**NVARCHAR(5)**) - NOT NULL
  - Name (**NVARCHAR(50)**) - NOT NULL
- Use the following **INSERT** statements to create 5 programs:
 

```
INSERT INTO UGPrograms (Code,Name) VALUES ('CSCDM','Cloud Management')
INSERT INTO UGPrograms (Code,Name) VALUES ('CSCYS','Cyber Security')
INSERT INTO UGPrograms (Code,Name) VALUES ('CSDAT','Data Analytics')
INSERT INTO UGPrograms (Code,Name) VALUES ('CSMAD','Application Development')
INSERT INTO UGPrograms (Code,Name) VALUES ('CSSEN','Software Engineering')
```
- Create **TABLE Courses** with columns:
  - ID (**INT**) - auto-increment from number 1 - NOT NULL
  - Code (**NVARCHAR(10)**) - NOT NULL
  - Title (**NVARCHAR(50)**) - NOT NULL
  - Section (**INT**) - NOT NULL
  - Term (**NVARCHAR(10)**) - NOT NULL
  - Year (**INT**) - NOT NULL
  - ProgramID (**INT**) - NOT NULL
- Use ProgramID as the foreign key to establish a one-to-many relationship between a **UGPrograms** and **Courses**. Such that, every **Course** must be part of exactly one **UGProgram**, while a **UGProgram** can contain any number of courses.
- Create records in **TABLE Courses**. Use the **INSERT** statements given in the file "Insert Query Courses.txt".

### Task A2 - (10 marks)

- Create a new ASP.NET Core Web App (MVC) project, named "FFF-LLL-Prob-Asst-3", where "FFF" is your first and "LLL" is your last name.
- Applying the **Database-First approach**, generate models and context using the Entity Framework, corresponding to database **CollegeDB**.
- After successful generation of models, add a new controller ((MVC with views) for

the Model Class **Course**. Use the **DbContext** class created in the previous step.

The generated view, showing all courses should look like:

https://localhost:7263

Course Management System

Home

Courses

Privacy

Index

[Create New](#)

Code	Title	Section	Term	Year	Program
CDM1544	Operating Systems – Linux	1	SPRING	2024	1
CYS1622	Internetworking	2	SPRING	2024	2
MAD1811	Interactive Application Development	1	SPRING	2024	4

- Change the Index page such that, In the coloumn "Program", full name of the program is displayed

Using **Data Annotation Validators** add validation in class **Course**, following rules:

- **StudentID** is the primary key
- **Code** is a strings of 50 charachters at most
- **Title** is a strings of 50 charachters at most
- **Section** is a single digit non-zero integer
- **Term** is a strings of 50 charachters at most
- **Year** is a 4-digit integer, whoes value is between and including the current year, and at most 4 years in future. Thus, the range is 2024-2028. However, you should implement it in a general way, so that it does not only work for 2024.
- All properties should be required

Change the Create page such that,

- The field labeled as "Term" is a drop down selection list, with values loaded from the **TABLE Terms**
- Change the label of the field "ProgramId" to "Program". Moreover, show program's full name instead of the Id.

In the generated view, the razor view page to create a new course should look like:

The screenshot shows a web browser window with the address bar displaying `https://localhost:7263/Courses/Create`. The page title is "Course Management System" with navigation links for "Home", "Courses", and "Privacy". The main heading is "Create Course". Below this, there are several input fields: "Code", "Title", "Section" (with a value of "0"), "Term", "Year" (with a value of "0"), and "ProgramId" (with a value of "1"). A blue "Create" button is positioned below the "ProgramId" field. At the bottom left, there is a blue link labeled "Back to List".

## Part B - Search, Filtering, and Pagination - (20 marks)

By implementing pagination we can divide large datasets into smaller ones, and thus more manageable ones. Pagination improves response times and reduce load. It is important that we filter data on the server instead of always retrieving all rows of the table. With filtering, users can request specific subsets of data - matching a certain search criteria. This can make data retrieval relatively more efficient.

### Task B1 - (6 marks)

- Change the Index page and the controller such that, the column headings become links that the user can click to sort by the property linked to that column.
- Change the Index page and the controller such that, clicking a column heading, should toggles record rows between ascending and descending sort order.
- Add a text link to show all (un-filtered) courses.

### Task B2 - (8 marks)

- Add filter/search capability to the Index page. For that, add a text box and a submit button to the Index page, and make corresponding changes in the controller. The text box should let user enter a string to search for a **course** by **title**.

Find Course by title:  [Search](#) [Back to Full List](#)

- Bootstrap to make sure that the text input box and the button **Search**, always appear in a single row.
- Add more filtering capability. For that, add two drop-down lists and a submit button labelled **View Courses** above the table of records in the Index page.
  - First list, labelled **Term**, gives a list of Semesters, fetched from table **Terms**.
  - Second drop-down list, labelled **Year**, gives values of years, from the current to three years in future. For instance, in 2024 it should display years 2024 to 2027.
  - Create controller method, so that on clicking the button **View Courses**, courses only offered in the selected term and year are listed in the Table.
  - If no year is selected then on clicking the button **View Courses**, all courses offered in the selected term and the current year are listed in the Table.
  - If only year is selected then on clicking the button **View Courses**, courses offered in all terms of the selected year are listed in the Table.

Term:  Year:  [View Courses](#)

- Bootstrap to make sure that the selection lists and the button **View Courses**, always appear in a single row.
- Add more filtering capability. Above the table of records in the Index page, add buttons labelled with the name of each under-graduate program. The names of the programs should not be hard-coded, and must be fetched from the database. On the click of a program button, only the courses in that program are listed.

Cloud Management

Cyber Security

Data Analytics

Application Development

Software Engineering

- Implement search and filtering in such a way that, search criteria and filtering parameters persists. For instance, if search results for "Software" are already displayed in the table when a particular term and year is selected, then only courses offered in the selected term and the selected year are listed which has the string "Software" in the title.

### Task B3 - (6 marks)

- Add paging to the Courses Index page with a page size of 10. Add paging buttons (or clickable text links) to the Courses Index page. Create two buttons, one for next page and the other for the previous page.
- Add paging to the Courses Index page with a page size of 10, by providing a clickable text link for each page of the selected 10 Courses.



- Bootstrap to make sure that the page numbers and the buttons **Previous**, and **Next** always appear in a single row.

## Part C - More Model Classes - (30 marks)

In this part, you are required to add more entities to the model, and hence more tables to the MS SQL database **CollegeDB** by using the Model-First approach.

### Task C1 - (6 marks)

- Add a new folder to the project and name it **Utilities**.
  - In the folder **Utilities**, create a new static class, **Utility**.
- In the class **Utility**, create a new static method,
 

```
public static DateTime FirstFridayOfFirstWeek(int year, int month)
```

  - Method **FirstFridayOfFirstWeek**, should return the date falling on the first Friday of the first week of the month of the given year. Here the input **int** year, is a four digit value for the year, and **int** month is the numeric value for the month (e.g. 1 for January and 12 for December).
- Similarly, create a new static method,
 

```
public static DateTime FirstMondayOfSecondWeek(int year, int month)
```

  - Method **FirstMondayOfSecondWeek**, should return the date falling on the first Monday of the second week of the month of the given year.
- Add a readonly property (type **DateTime**) **StartDate** on the **Course**, with display name **Start Date**, for the first day of classes of the **Course**. Use the following rules for the start of each term.
  - FALL: Classes start on the first Monday of the Second week of September

- WINTER: Classes start on the first Monday of the Second week of January
- SPRING: Classes start on the first Friday of the First week of May
- Add a readonly property (type `Boolean`) `IsOpenToEnroll` on the `Course`, with display name **Open**. Property `IsOpenToEnroll` is assigned the value `true`, if the `Course` is accepting students and is still open for enrollment, and `false` otherwise. To implement the business logic, follow the rules:
  - `Course` starts allowing enrollments, three months before the `StartDate`.
  - `Course` stops allowing enrollments, two weeks after the `StartDate`.
- Modify the Razor page (`Courses/Index`), which shows all courses offered, such that it now also has a column showing, if the course is open for enrollment or not. See the following snapshot for expected view.

Course Management System [Home](#) [Courses](#) [Privacy](#)

## College: Courses

[Create New](#)

Find by program name:  Search [Back to Full List](#)

Code	Title	Section	Term	Year	Open	Program	
CDM1544	Operating Systems – Linux	1	SPRING	2024	No	Cloud Management	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
CYS1622	Internetworking	1	SPRING	2024	No	Cyber Security	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
DAT1733	Database Design	1	SPRING	2024	No	Data Analytics	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
DAT1755	Statistical Methods	1	SPRING	2024	No	Data Analytics	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
MAD1811	Interactive Application Development	1	SPRING	2024	No	Mobile Application Development	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
SEN1453	Fundamentals of Software Design	1	SPRING	2024	No	Software Engineering	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
CDM1543	Data Communications and Networking	1	FALL	2024	Yes	Cloud Management	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
CDM1543	Data Communications and Networking	2	FALL	2024	Yes	Cloud Management	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
CDM1544	Operating Systems – Linux	1	FALL	2024	Yes	Cloud Management	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
CDM1544	Operating Systems – Linux	2	FALL	2024	Yes	Cloud Management	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

[Previous](#) [Next](#)

### Task C2 - (5 marks)

- Create a new class `Student`. A `Student` profile is only created in the database, if the `Student` is accepted into an existing program. A `Student` is allowed to enroll into more than one courses.
- Create the class `Student` with properties, such that after a successful migration of the model, the SQL schema for the Table **`Student`** should have:

```
[StudentID] INT IDENTITY (101100, 1) NOT NULL,
[FirstName] NVARCHAR (50) NOT NULL,
[LastName] NVARCHAR (50) NOT NULL,
[Address] NVARCHAR (100) NOT NULL,
[PostalCode] NVARCHAR (6) NOT NULL,
```



```

[Email]      NVARCHAR (50) NOT NULL,
[Type]       NVARCHAR (10) NOT NULL,
[Status]     NVARCHAR (10) NOT NULL,

```

- All properties are required. Use appropriate display labels. Use annotations to validate inputs. The `StudentID` of the first record must be equal to 101100, and subsequent records get values incremented by one. `Email` must be a valid email, and `PostalCode` must be a standard Canadian postal code. An acceptable value for `PostalCode`, to be written to the database, must be of the like, **A0A0A0**.
- You must create the class `Student` such that, navigations (reference) properties are introduced to establish the following relationships.
  - A student must be accepted in exactly one program.
  - More than one student can be accepted in a given program.
  - A student can be enrolled into none or multiple courses.
  - More than one student can be enrolled in a given course.
  - A course must be part of exactly one program.
  - More than one courses can be part of a given program.
  - A student must be living in exactly one City.
  - More than one students can live in a given City.
  - It is possible that no student lives in a given City.
- Do a migration from the Model to the Database, so that the newly added entity is created in the SQL database **CollegeDB**.

**Caution:** Be careful to update database after the creation of a migration file. Check the migration file, it might requires some clean-up. Make sure that it is only creating the new table `Student`, and that none of the other tables are created, or dropped.

### Task C3 - (6 marks)

- Add a readonly property `FullName` on the `Student` class, with display name **Student Name**, for the full name of the student in **LastName, FirstName** format.
- Add a readonly property `CourseLoad` on the `Student` class for the number of courses the student is enrolled into.
- Add a readonly property `IsFullTime` on the `Student` class. Property `IsFullTime` is assigned the value `true`, if the `Student` is a full-time student,, and `false` otherwise. To assign values, follow the rule:
  - A full-time `Student` is enrolled into at least three courses.
  - A `Student` is considered part-time if the `Student` is not full-time.

- In the static class `Utility`, create a new static method,  
`public static string ProvinceOfCity(string cityName)`
- Method `ProvinceOfCity`, returns the province name where the city `cityName` is situated (must be retrieved by joining tables `Cities` and `Provinces`).
  - If more than one provinces have a city by the name `cityName` then an empty string is returned.
- On creation of a new **Student**, field **Province** should be figured out from the field **City** by using the utility method `ProvinceOfCity`.
  - If more than one provinces have the city by the same name, the method `ProvinceOfCity` would have returned an empty string. In that case determine province based upon the first letter of the postal code. **(A table given in the end shows the letters associated with provinces.)**
  - If province can also not be determined based upon the first letter of the postal code, then an empty string is assigned to province.
- Add a readonly property `FullAddress` on the `Student` class, with display name **Student Address**, for the full address of the student in **Address, City, Province Code, Postal Code** format.
- Define an `enum` with values **ELIGIBLE**, **ENROLLED**, and **NOTELIGIBLE** representing status of a `Student`, such that:
  - The default status is **ELIGIBLE**, which is assigned when a new `Student` profile is created (and thus, must have been accepted into a program).
  - Once created, a `Student` can take any of the three statuses.
  - When `Student` enrolls into a course, the status of the `Student` changes from **ELIGIBLE** to **ENROLLED**.
  - A `Student` is allowed to enroll into a course, only if the status of the `Student` is either **ELIGIBLE** or **ENROLLED**.
  - If a `Student` is not enrolled in any course, but is still in the program then its status must be **ELIGIBLE**.
  - If for some reason (administration or academic), for example non-payment of tuition fee or college dues, a `Student` cannot enroll in any course, then the status must be **NOTELIGIBLE**.

#### Task C4 - (4 marks)

- Without creating a new `Controller` or `DbContext` classes, create a view **NewStudent** - with a form and a submit button, to create a new student profile.

- In **NewStudent**, form should have input fields for all the required properties other than **Status**. The default value, i.e. **ELIGIBLE**, must be used for **Status**.
- Input fields for **Program**, **Type**, and **City** should be drop-down menus, whose selection values are fetched from relative tables in the database **CollegeDB**.

See the following snapshot of the expected view.

The screenshot shows a web application titled "Course Management System" with navigation links for Home, Courses, Students, and Privacy. The main heading is "Create New Student Profile". The form contains the following fields: First Name, Last Name, Email, Type (with a dropdown menu showing "DOMESTIC"), Address, City (with a dropdown menu showing "Abbey"), Postal Code, Program (with a dropdown menu showing "Cloud Management"), and a blue "Create" button at the bottom.

#### Task C5 - (4 marks)

- Without creating new classes, create razor view **Course**, which will be similar to the page **ManageCourse** created in Problem Assignment 2.
- Modify the page (Courses/Index) showing all courses. Change the link [Details](#) to [Manage](#), such that on clicking [Manage](#), the page **Course** is loaded.
- Page **Course**, lists all students enrolled in the selected course.
- In the table listing all enrolled students, add a link [Drop](#) at the end of each row.
- On clicking [Drop](#) button the concerned student is dropped from the course, and the status of student changes (if necessary).

#### Task C6 - (5 marks)

- On the page **Course**, if the selected course is still open for enrollment, also

provide a single field form with a submit button (labelled Enroll Student). The lone field, a drop-down list, gives full names of all the students eligible to enroll in the selected course. Obviously, listed students must have already been accepted into the program containing the course.

- Once a student is selected, and the submit button is pressed, the selected student is enrolled into the course, and the status of student (possibly) changes.
- On the page **Course**, if the selected course is closed for enrollment, instead of providing a form with a submit button, print message "Course is closed, and is not accepting new students".

## Part D - More Model Classes - FinancialStatement (20 marks)

For every student a financial statement (sometimes called account statement) is maintained. A financial statement gives a snapshot of dues the student is supposed to pay, or have already paid to the College. The dues can be registration fee for joining a program. It can include tuition fee (calculated per course) to enroll into a course. To maintain a good status, student also has to pay some other non-tuition fee and charges. Usually, such fee depends upon student's status (i.e. domestic or international) and course load (full or part time). Academic institutes usually follow different policies for students falling into different categories.

### Task D1 - (3 marks)

- Create a new class **FinancialStatement**, with the following properties:
  - **ID**; an integer value to be automatically assigned
  - **LastChanged**; a **DateTime** value for the last time the **FinancialStatement** was changed (or created)
- Create a new class **StatementEntry**, with the following properties:
  - **ID**; an integer value to be automatically assigned
  - **Description**; a string describing the **StatementEntry**
  - **Value**; a non-negative double value
- Apply **Data Annotation Validators** for the following rules:
  - **ID**'s are the primary keys
  - All properties are required
- Introduce navigation properties to establish the following relationships.
  - A **StatementEntry** must be on exactly one **FinancialStatement**.
  - A **FinancialStatement** can have zero to many **StatementEntry**.
  - A **FinancialStatement** can only be created for an existing **Student**

- A `FinancialStatement` can only be created for a single `Student`
- A `Student` must have exactly one `FinancialStatement`.

#### Task D2 - (4 marks)

- Create a new class `FeePolicy`, with the following properties:
  - `ID`; an integer value to be automatically assigned
  - `Catagory`; a string for he catagory of the `FeePolicy`
  - `TuitionFee`; a non-negative double value
  - `RegistrationFee`; a non-negative double value
  - `FacilitiesFee`; a non-negative double value
  - `UnionFee`; a non-negative double value
- Apply **Data Annotation Validators** for the following rules:
  - `ID` is the primary key
  - All properties are required
  - Use annotations to force legal values for all fee
  - Choose appropriate (readable) display names for properties
- Introduce navigations properties to establish the following relationships.
  - A `FinancialStatement` must observe a single `FeePolicy`
  - A `FeePolicy` does not have to be assoicated to a `FinancialStatement`.
- Do a migration from the Model to the Database, so that the newly added entities are created in the SQL database **CollegeDB**.
- Create records in TABLE `FeePolicy`. Use the following sequences of (`Catagory`, `TuitionFee`, `RegistrationFee`, `FacilitiesFee`, `UnionFee`).

```
(Domestic Full-Time,      800,   100,   72.5,  97.56)
(Domestic Part-Time,      800,   100,   31.2,  20.00)
(International Full-Time, 3766,   200,  294.5, 597.56)
(International Part-Time, 3766,   200,  173.2, 420.00)
```

- Add a readonly property `Ballance` on the `Student` class, with display name **Total Amount Owed**, for the current amount owed by the student to the College.
- To calculate the amount assigned to the property `Ballance`, you should go through the connected `FinancialStatement` and read one `StatementEntry` at a time. For each `StatementEntry` add the value to the final amount. For this value, calculate the applicable tax (12.9 %) over the sum of the values of the 4 properties. The final sum returned should contain the tax.

#### Task D3 - (4 marks)

- Change the controller for the view **NewStudent** (which creates a new student

profile), such that before creating a new student:

- Create a new `FinancialStatement` with the `FeePolicy(New Student)` and with the current time and date as the value for `LastChanged`.
- Create a `StatementEntry` with the current time/date as the `DateCreated`, `Description "Registration Fee"`, and `Value` equal to the value of property `RegistrationFee`, given in the `FeePolicy(New Student)`.
- Add the `StatementEntry` to the list of entries of the above created `FinancialStatement`.
- Now create a new `Student`. Assign the newly created `FinancialStatement` to new `Student`.

#### Task D4 - (5 marks)

- Change the controller of view **Course**, where a submit button is provided to enroll Student into a course. Before enrolling the student consider the following cases:
- **Case 1:** If this is going to be the first course enrolled by the Student. Then in this case student will be considered part-time.
  - Create a new `FinancialStatement` with the `FeePolicy(Domestic Part-Time)` or `FeePolicy(International Part-Time)`, whichever applies according to `Type` of the Student.
  - Create 4 `StatementEntry` objects with the `Description` equal to the display names of the properties `TuitionFee`, `RegistrationFee`, `FacilitiesFee`, and `UnionFee`, with values equal to the amounts given in the selected `FeePolicy`, and add them to the list of entries of `FinancialStatement`.
  - Assign the newly created `FinancialStatement` to the Student.
- **Case 2:** If this is going to be the second course enrolled by the Student. Then the student is still part-time. Do not create a new `FinancialStatement`.
  - Create a new `StatementEntry` object with the `Description` equal to the display names of property `TuitionFee`, and `Value` equal to the amount given in the `FeePolicy`, and add it to the list of entries of `FinancialStatement`.
- **Case 3:** If this is going to be the third course enrolled by the Student. Then the status changes to full-time. Create a new `FinancialStatement`.
  - Choose `FeePolicy(Domestic Full-Time)` or `FeePolicy(International Full-Time)`, whichever applies according to `Type` of the Student.
  - Create 4 `StatementEntry` objects with the `Description` equal to the display names of the properties `RegistrationFee`, `FacilitiesFee`,

- UnionFee, and Value equal to the amounts given in the selected full-time student FeePolicy, and add them to the list of entries of the FinancialStatement.
  - Create three new StatementEntry objects with Description equal to the display name of TuitionFee, and Value equal to the amount given in the FeePolicy, and add to the list of entries of FinancialStatement.
  - Assign the newly created FinancialStatement to the Student.
- Case 4:** If the status of student is full-time.
  - Create a new StatementEntry object with the Description equal to the display names of property TuitionFee, and Value equal to the amount given in the FeePolicy, and add it to the list of entries of FinancialStatement.

#### Task D5 - (4 marks)

- Without creating new controller(s), create a new razor page **StudentAccount**.
- Razor View Page **Course**, lists all students enrolled in the selected course.
- In the table listing all enrolled students on Page **Course**, add a link [Account](#) at the end of each row, such that on clicking link [Account](#), Page **Course** is loaded.
- Razor View Page **Course**, must show the financial statement of the Student, showing all entries, date changed (or created) of statement, description of each entry, and the total amount owed by the student.

See an example of a financial statement below:

#### Accounts Statement

Date Changed/Created: 11/17/2024 10:35:50 AM

Student ID: 101282

Name: Mohin Das

Fee Policy: International Full-Time

Balance = \$16,653.83

-----	
Registration Fee	\$200.00
Tuition Fee(Course Code)	\$3766.00
Tuition Fee(Course Code)	\$3766.00
Tuition Fee(Course Code)	\$3766.00
Tuition Fee(Course Code)	\$3766.00
Facilities Fee	\$294.50

Student Union Fee	\$597.56
Tax(at 12.99 %)	\$2084.13
<hr/>	
<b>Total Amount Owed</b>	<b>\$18,240.19</b>
<hr/>	

- Bootstrap to make sure that the statemnet is formatted as it appears above.

## Part E - Unit Testing (Using xUnit) - (10 marks)

Implement a set of simple unit tests, such that the test(s) runs and pass.

- Add a new Unit Test Project
- Test all methods in the class `Utility`
- For method `FirstFridayOfFirstWeek`, have at least 4 tests.
  1. Test the result of `FirstFridayOfFirstWeek(2024,1)`.
  2. Test the result of `FirstFridayOfFirstWeek(2024,9)`.
  3. Test the result of `FirstFridayOfFirstWeek(2044,1)`.
  4. Test the result of `FirstFridayOfFirstWeek(2424,9)`.
- For method `FirstMondayOfSecondWeek`, have at least 4 tests.
  1. Test the result of `FirstFridayOfFirstWeek(2024,5)`.
  2. Test the result of `FirstFridayOfFirstWeek(2025,5)`.
  3. Test the result of `FirstFridayOfFirstWeek(2044,5)`.
  4. Test the result of `FirstFridayOfFirstWeek(2424,5)`.
- For testing method `ProvinceOfCity`, have the following input strings.
  1. "Armstrong". More than two provinces has this city.
  2. "Auckland". There is no city by this name in any province.
  3. "toROnto". A city by this name exists in Ontario.
  4. "toronto island". A city by this name exists in Ontario.
  5. "OTTAWA". A city by this name exists in Ontario.
  6. "". An empty string. There shpuld not be a city by this name in any province.

### What/How to Submit?

Zip up your entire solution into **one zip file** and submit that file to the eConestoga dropbox for the assignment. You can submit multiple solutions but only the latest (i.e. most recent) one will be looked at and evaluated.

### A Word of Caution - Again



Make sure you do your own work and do not copy code from any other source. Every solution will be run through [MOSS](#) to check for academic integrity violations. There is zero-tolerance for such violations and any encountered with be dealt with in accordance with [Conestoga's policy](#).

---

**First character of Postal Code and the Geographic areas:**

A	Newfoundland and Labrador
B	Nova Scotia
C	Prince Edward Island
E	New Brunswick
G	Quebec
H	Quebec
J	Quebec
K	Ontario
L	Ontario
M	Ontario
N	Ontario
P	Ontario
R	Manitoba
S	Saskatchewan
T	Alberta
V	British Columbia
X	Northwest Territories/Nunavut
Y	Yukon

---