Department of Information Systems and Technologies
CTIS479 – C# Programming in the .NET Framework
Spring 2020 - 2021
Lab Guide 1

OBJECTIVES: Indexer, operator overloading, Extension Methods, LINQ, Lambda, Delegate, Func, Action, Event
Instructors : Nese SAHIN OZCELIK

Q1. Create **Student** class according to below requirements

- Create enum **GradeLevel** with FirstYear=1, SecondYear, ThirdYear, FourthYear
- Implement properties **FirstName, LastName, Id, Year** from GradeLevel enum type
- Create **ExamScores** as a generic **List** to store exam scores of student as int.
- Override **ToString()** method with **lambda** operator which returns the student details as below

  Id: 117
  FirstName: Hanying
  LastName: Feng
  Exam scores: 93,92,80,87
  Exam average: 88

  - To find the average of exam grades implement **extension** method **Average()** for the list which find and returns the average of a list items. **Hint: extension methods can be defined in static class as static method**
- Implement **operator overload method** for **++** sign so that average of the exams of student will displayed
  - studentObject++;
  - In program.cs for a student object call ++ operator overload method

Create **StudentCollection** class according to below requirements

- Create a static generic list **students** which can store Student objects
- Create a static constructor and put the following student objects to that generic list

```
static List<Student> students = new List<Student>{
            new Student {FirstName = "Terry", LastName = "Adams", Id = 120,
                Year = GradeLevel.SecondYear,
                ExamScores = new List<int> { 99, 82, 81, 79}},
            new Student {FirstName = "Fadi", LastName = "Fakhouri", Id = 116,
                Year = GradeLevel.ThirdYear,
                ExamScores = new List<int> { 99, 86, 90, 94}},
            new Student {FirstName = "Hanying", LastName = "Feng", Id = 117,
                Year = GradeLevel.FirstYear,
                ExamScores = new List<int> { 93, 92, 80, 87}},
            new Student {FirstName = "Cesar", LastName = "Garcia", Id = 114,
                Year = GradeLevel.FourthYear,
                ExamScores = new List<int> { 97, 89, 85, 82}},
            new Student {FirstName = "Debra", LastName = "Garcia", Id = 115,
                Year = GradeLevel.ThirdYear,
                ExamScores = new List<int> { 35, 72, 91, 70}},
            new Student {FirstName = "Hugo", LastName = "Garcia", Id = 118,
                Year = GradeLevel.SecondYear,
                ExamScores = new List<int> { 92, 90, 83, 78}},
            new Student {FirstName = "Sven", LastName = "Mortensen", Id = 113,
                Year = GradeLevel.FirstYear,
                ExamScores = new List<int> { 88, 94, 65, 91}},
            new Student {FirstName = "Claire", LastName = "O'Donnell", Id = 112,
                Year = GradeLevel.FourthYear,
                ExamScores = new List<int> { 75, 84, 91, 39}},
            new Student {FirstName = "Svetlana", LastName = "Omelchenko", Id = 111,
                Year = GradeLevel.SecondYear,
                ExamScores = new List<int> { 97, 92, 81, 60}},
            new Student {FirstName = "Lance", LastName = "Tucker", Id = 119,
                Year = GradeLevel.ThirdYear,
                ExamScores = new List<int> { 68, 79, 88, 92}},
            new Student {FirstName = "Michael", LastName = "Tucker", Id = 122,
                Year = GradeLevel.FirstYear,
                ExamScores = new List<int> { 94, 92, 91, 91}},
            new Student {FirstName = "Eugene", LastName = "Zabokritski", Id = 121,
                Year = GradeLevel.FourthYear,
                ExamScores = new List<int> { 96, 85, 91, 60}}
        }
```

- Create a static method **QueryHighScores** which displays the students with the given exam number and score. Use **LINQ** to retrieve the students with the given condition.

- Example: if exam number is 1 and score is 90, students whose first exam grade greater than 90 will be displayed.
- In Program.cs call the method. Output of the method will be ;

```
Id: 117
FirstName: Hanying
LastName: Feng
Exam scores: 93,92,80,87
Exam average: 88

Id: 113
FirstName: Sven
LastName: Mortensen
Exam scores: 88,94,65,91
Exam average: 84.5

Id: 111
FirstName: Svetlana
LastName: Omelchenko
Exam scores: 97,92,81,60
Exam average: 82.5

Id: 122
FirstName: Michael
LastName: Tucker
Exam scores: 94,92,91,91
Exam average: 92
```

- Implement **indexer** which gets two int parameters. First int parameter is for student index in the list. Second int parameter is for the exam index of the student. Indexer set and return the exam grade with the given exam index of the student in the list with the given student index.
  - Create `studentCollection` object from `StudentCollection` in Program.cs as and call the **indexer** to set and display the exam grade for first student.

    ```
    studentCollection[0, 2] = 80;
    Console.WriteLine("Student exam grade "+ studentCollection[0, 2]);
    ```

    0 means first student index in the list, 2 means third exam grade, 80 is the exam grade
  - Output of the indexer will be:

    Student exam grade: 80

- Overload **indexer** which gets an int parameter as a student index in the list. Indexer sets and returns the student object in the list with the given index**.**
  - In Program.cs call the indexer as below

    `studentCollection[0]` will return first student object in the list.

    `studentCollection[0]++` will display the average of the first student in the list. Operator overload method for ++ sign is implemented in the Student class.
  - Output of the indexer and operator overload will be:

    Average of the student: 85.25

Q2. Implement console application to control the **stock amount** of **products**. If stock amount is less than the defined limit, **stock amount** and **latest purchased amount** of that product will be written to **stockDetails.txt** file. Create **LessStockEvent** event and when this event is fired on the product, product details with the latest purchased amount will be written to text file.

- Create Product class with Id, **Name**, **StockAmount** properties.
  - create a constant LIMIT

```
static readonly int LIMIT = 10;
```

- Create a delegate `LessStockEventHandler`
  - Decide that which methods can be pointed by this delegate.
  - When **LessStockEvent** event occurs current product details will be written to file so that current product object is required by the event. One of the parameter of the delegate must be Product object. Also, latest purchased amount will be written to file. So that, current purchased amount must be the second parameter of the delegate.
  - For the second parameter latest purchased amount `StockLessEventArgs class with LatestPurchasedAmount` property can be created. Implement a constructor to set the LatestPurchasedAmount. For the second parameter of delegate LatestPurchasedAmount can be send as an int value. If you use StockLessEventArgs object, many event related values can be send to event related methods.

```
public class StockLessEventArgs : EventArgs {
      public int LatestPurchasedAmount { get; set; }
      public StockLessEventArgs(int latestPurchasedAmount){
            this.LatestPurchasedAmount = latestPurchasedAmount;
      }
}
```

- Create **LessStockEvent** event from `LessStockEventHandler` delegate type.
- Invoke the **`LessStockEvent`** whenever the StockAmount of the product is changed and it is under the limit.

- In the Program.cs, create three Product objects and assign the events and methods which will be called when event is fired over the product objects.

```
Product p1 = new Product(1111, "Bag", 40);
Product p2 = new Product(2222, "Shoe", 40);
Product p3 = new Product(3333, "Belt", 40); … … …

p1.LessStockEvent += new LessStockEventHandler(ProductStockAmountless);
… … …
```

  This means that when product amount is under the limit, then event will be fired and ProductStockAmountless method will be called for the product.

- Implement **`ProductStockAmountless(Product p, StockLessEventArgs arg)`** method and append the product details and latest purchased amount to the **stockDetails.txt text file.** Use the below statements to append a text file.

```
using (var destination = File.AppendText("stockDetails.txt")){
   destination.WriteLine(p.ToString()+"Latest purchased amount "+
   arg.LatestPurchasedAmount);
}
```

- Simulate product stock amount values are decremented. When stock amount of a product is decremented and if stock amount is under the limit, event will be fired and `ProductStockAmountless` method will be called. `ProductStockAmountless` method will display messages to output screen and append the product details to text file.

```
for (int i = 0; i < 5; i++){
   p1.StockAmount -= 7;
   p2.StockAmount -= 8;
   p3.StockAmount -= 7;
}
```

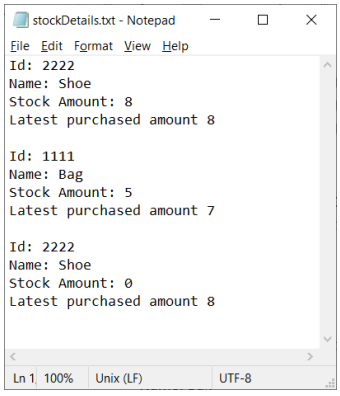
- At the end of the for loop, output will be as below

```
WARNING, Following Product is under stock
Id: 2222
Name: Shoe
Stock Amount: 8
Latest Purchased Amount 8

WARNING, Following Product is under stock
Id: 1111
Name: Bag
Stock Amount: 5
Latest Purchased Amount 7

WARNING, Following Product is under stock
Id: 2222
Name: Shoe
Stock Amount: 0
Latest Purchased Amount 8
```

- At the end of the for loop, file content will be as below.



  - File is created under **bin\Debug\netcoreapp3.1** folder of the project. To easily open the project folder in file explorer, over the project right click, select Open Folder in File Explorer,

- Modify for loop which is used to decrease the stock anount, so that Random number between 1 to 10 will be created by the usage of the **Func** delegate. Remember, Func delegate is predefined delegate which can point to methods up to 16 generic parameters with a return type. Here a delegate will be pointed to a method which returns a random number. Instead of defining a generic delegate, **Func** predefine delegate can be used and it will provide to write less statement.

```
static Func<int> GetRandomNumberDelegate;

GetRandomNumberDelegate = delegate (){ //anonymous method is assigned to delegate
   Random rnd = new Random();
   return rnd.Next(1, 10);
};


for (int i = 0; i < 5; i++){
   p1.StockAmount = GetRandomNumberDelegate ();
   p2.StockAmount = GetRandomNumberDelegate ();
   p3.StockAmount = GetRandomNumberDelegate ();
}
```

- Modify the **ProductStockAmountless** method so that Action delegate can be used to display the output. Remember, Action delegate is predefined delegate which can be point to methods which has up to 16 generic parameters with void return type. Instead of defining a generic delegate, **Action** predefine delegate can be used to write less statement.

```
Action<string> PrintActionDelegate;

PrintActionDelegate = outputToDisplay => Console.WriteLine(outputToDisplay);
```

In **ProductStockAmountless** method call **PrintActionDelegate** instead of `Console.WriteLine()`.

```
PrintActionDelegate("WARNING, Following Product is under stock" + p+"Latest
Purhased Amount "+arg.LatestPurchasedAmount+"\n");
```