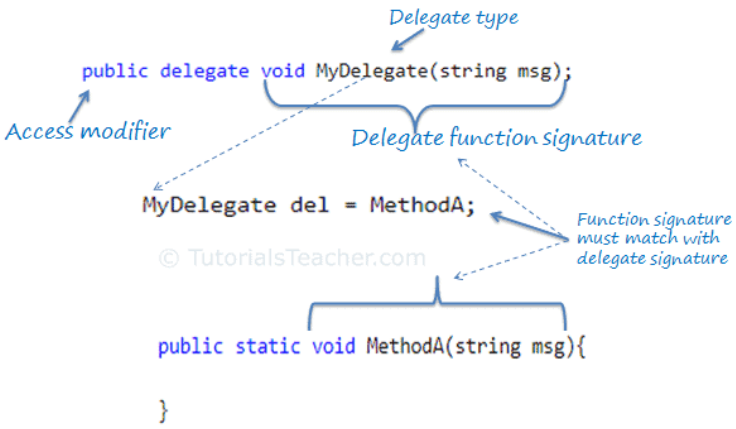
Assignment -4: Delegate, Anonymous Method, and Lambda expression

Objectives:

O[1]. To learn the delegates and apply them in real-world problem-solving.
O[2]. To learn the difference between tight coupling coding and loose coupling coding and apply them in programming.
O[3]. To apply delegate to invoke anonymous methods.
O[4]. To understand the anonymous method and lambda expression.

In this works, students are going to understand tight coupling and loose coupling coding concepts and apply them in the Asp.Net framework web form application. In addition, they will learn programming with method invocation by method name and by a delegate, anonymous method and its implementation with delegates and lambda expression will also be discussed.

A delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance. [Microsoft]



https://www.tutorialsteacher.com/csharp/csharp-delegates

Figure 1: Association of delegate instance with any method with a compatible signature and return type.

There are three(3) steps involved while working with delegates: Declare a delegate, Set a target method, and Invoke a delegate.

Tight coupling is when a group of classes are highly dependent on one another. This scenario arises when a class assumes too many responsibilities, or when one concern is spread over many classes rather than having its class.

Loose coupling is achieved using a design that promotes single-responsibility and separation of concerns. A loosely coupled class can be consumed and tested independently of other (concrete) classes.

Task1: Implement a class named Employee that contains the following members:

- A public auto-implemented property named name. Which will return string-type data.
- A public auto-implemented property named Id. Which will return int type data.
- A public auto-implemented property named salary. Which will return double type data.
- A public auto-implemented property named experience. Which will return int type data.
- A static method named IsPromotable(), which will take input a list of the employee as parameters and no return.
- This method will check the experience of all employees and print the name of the employees whose experience is greater than 4 years.

Implement a driver class with the following members:

- Create a instance of a List<T> and name it as employee list.
- Initialize the list with the following values:

```
employeeList.Add(new Employee() { name = "Merinda", Id = 1, salary = 200000.00, experience = 5 });
employeeList.Add(new Employee() { name = "Belal", Id = 2, salary = 255000.00, experience = 4 });
employeeList.Add(new Employee() { name = "Roy", Id = 3, salary = 280000.00, experience = 6 });
employeeList.Add(new Employee() { name = "Poly", Id = 4, salary = 160000.00, experience = 3 });
```

- Invoke the IsPromotable() method of the Employee class with employeeList parameter.

a) The above code is a tightly coupled code, we need to make it loosely coupled by moving the IsPromotable() method implementation in the driver class. This will help the programmer to work in framework programming and increase independency and reusability.

b) Now use delegate to invoke IsPromotable() method and create the following method into the Employee class to print the promotable employee name.

```
public static void Promotable(List<Employee> empList, IsPromotable_D promotable) {
        foreach (Employee e in empList) {
           if (promotable(e))
              Console.WriteLine(e.name + "is promotable");
        }
}
```

Task2: How to run a program with Web Form as WebApplication in .Net Framework?

a) Create a simple web form using WebApplication .Net Framework.

b) Execute the program with web form in Visual Stdio 2019 Editor.

Microsoft started to struggle to maintain frequent small components assembled and updated because of the huge monolithic framework of ASP.NET. The problem was not only a matter of release cycles. The development style also was changing. Hiding and abstracting away the complexities of HTTP and HTML markup helped a lot of WinForm developers to become web

developers, but after more than five years of experience, developers wanted more control, especially over the markup rendered on pages.
In order to solve these two problems, in 2008 the ASP.NET team developed the ASP.NET MVC framework, based on the Model-View-Controller design pattern, which was also used by many of the popular frameworks at the time. This pattern allowed a cleaner and better separation of business and presentation logic, and, by removing the server-side UI components, it gave complete control of the HTML markup to developers. Furthermore, instead of being included inside the .NET framework, it was released out of band, making faster and more frequent releases possible. Although the ASP.NET MVC framework solved most of the problems of Web Forms, it still depended on IIS and the web abstracting library System.Web. This means that it was still not possible to have a web framework that was totally independent from the larger .NET framework.

Fast-forward a few years, single page applications (SPAs) was arrived. Basically, instead of interconnected, server-generated, data-driven pages, applications were becoming mostly static pages where data was displayed interacting with the server via Ajax calls to web services or Web APIs. Also, many services started releasing APIs for mobile apps or third-party apps to interact with their data Another web framework was released to adapt better to these new scenarios: ASP.NET Web API.
The ASP.NET team also took this opportunity to build an even more modular component model, finally ditching System.Web and creating a web framework that could live its own life independently from the rest of ASP.NET and the larger .NET framework. A big role was also played by the introduction of NuGet, Microsoft's package distribution system, making it possible to deliver all these components to developers in a managed and sustainable way. One additional advantage of the break-up from System.Web was the capability to not depend on IIS anymore and to run inside custom hosts and possibly other web servers.

Task3: Take three inputs from Number1, Number2, and Option using web form of ASP.NET framework and invoke the method Add or Sub method according to the option value with Number1 and Number2 as parameters.

Implement a class named MathClass that contains the following members:

- a public static method named Add() that takes two integer arguments x and y and returns the integer value of x+y.
- a public static method named Sub() that takes two integer arguments x and y and returns the integer value of x-y.

Implement a driver class(UI) that contains the following members:

- Read a string of numbers and convert them into an integer. Assign the integer value to the integer private Number1 field.
- Read a string of numbers and convert them into an integer. Assign the integer value to the integer private Number2 field.
- Read a string of numbers and convert them into an integer. Assign the integer value to the integer private Option field.
- Now check the value of Option in the main method and call the corresponding method.

Look carefully, the classes are tightly coupled with each other. UI class needs to change for any additional methods in MathClass.

a) Make the classes loosely coupled. Shift the option checking method and place it in the MathClass. Call the method from the UI class.
b) Use delegate and return delegates to the UI class.
c) Convert the Add and Sub methods as anonymous by the delegate and check the results.
d) Convert the anonymous methods as lambda expressions and check the results.