Seanca 7

Anonymous methods

- An anonymous method is an inline method, and it does not have a name, i.e, it has body only. We can define it using a delegate

Example

```
class Program
{
    delegate int delAddtion(int num1, int num2);
    static void Main(string[] args)
    {
        delAddtion objDel= addition;
        Console.WriteLine( objDel.Invoke(4, 4).ToString());

    }
    static int addition(int num1, int num2)
    {
        return num1 + num2;
    }
}
```

```
class Program
{
      delegate int delAddtion(int num1, int num2);
      static void Main(string[] args)
      {
          delAddtion objDel= delegate (int num1, int num2)
          {
              return num1 + num2;
          };

         Console.WriteLine( objDel.Invoke(4, 4).ToString());

      }
      static int addition(int num1, int num2)
      {
          return num1 + num2;
      }
}
```

Advantages

- ✔ Performance

Anonymous methods take less time to execute

- ✔ Less code

Lambda Expressions

- Lambda expressions are anonymous methods

- No access modifier
- No name
- No return statement

- Less code

- Code is more readable

Syntax

- Lambdas take the form: ***(args[]) => { statements; }***
- Formally, => (lambda operator) reads as **'[Left] goes to [Right]'**
- Lambdas accept a void parameter list.

  () => statement(s);

- Lambdas do not require arguments to refer to external variables

  Employee employee= listEmployees.Find(Emp => Emp.ID==102)

- **`return`** isn't required to get the result of a lambda, unless the value or reference of an internal variable is being returned.

  Employee employee=listEmployees.Find((Employee Emp) =>Emp.ID==102)

Lambda expressions and delegates

- Lambdas can be mapped to the first-class function (**`Func`**`<[types]>`) generic type
  - The last type in the generic is the return type; all types before that are argument types.
  - Provides a means of storing or returning Lambdas with strong typing
- Lambdas can also be mapped to **`Action`**`<[types]>`
  - All parameters are inputs
  - Actions have a void return type.

Using the built in Delegates

- If you want a *delegate* type that doesn't return a value, you can use the ***System.Action*** types. They can also take 0 to 16 parameters, but they don't return a value.

  static Action<string> logMessage = (message) => Console.WriteLine(message);

- The ***Func<…>*** types can be found in the *System* namespace and they represent *delegates* that return a type and take 0 to 16 parameters.

  Func<int,int,int> add = (a, b) => a + b;

- The **Predicate** built in delegate type lets you create code that takes a value of a particular type and returns true or false.
- All those types inherit from ***System.MulticastDelegate*** so you can add multiple methods to the invocation list.

  Predicate<int> dividesByThree = (i) => i % 3 == 0;

Example

- Finding the square of a number using methods vs using lambda expressions

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(Square(5));
    }

    static int Square(int number)
    {
        return number*number;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        // args => expression
        //number => number*number;

        Func<int, int> square = number => number*number;

        Console.WriteLine(square(5));
    }
}
```

Example

Modify the delegates assignment in the last lecture using lambda expressions

```
static void Main()
{
    string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George" };

    List<string> namesLessThanFiveChars = ExtractStrings(names, i => i.Length < 5);
    List<string> namesMoreThanFiveChars = ExtractStrings(names, i => i.Length > 5);
    List<string> namesExactlyFiveChars = ExtractStrings(names, i => i.Length == 5);

    Console.WriteLine("All names: " + string.Join(", ", names));
    Console.WriteLine(new string('-', 40));
    Console.WriteLine("Names less than five chars: " + string.Join(", ", namesLessThanFiveChars));
    Console.WriteLine("Names more than five chars: " + string.Join(", ", n
    Console.WriteLine("Names exactly five chars: " + string.Join(", ", namesExactlyFiveChars));
    Console.WriteLine(new string('-', 40));
}
```

Anonymous Methods and Lambda Expressions

- We can create anonymous methods using lambda expressions.
- In the example below we create an anonymous method using lambda expressions that finds the sum of two integers

```
Func<int, int, bool> checkIntegers = (int i, int j) => i < 8 + j;

Action printSomething = () => Console.WriteLine("Printing");

printSomething();

Action<int, int> sumOfTwoNumbers = (i, j) =>
{
    Console.WriteLine("The i number is: " + i);
    Console.WriteLine("The j number is: " + j);
    Console.WriteLine("The sum of i + j is: " + (i + j));
};

sumOfTwoNumbers(1, 2);
```

Example

Func delegate that takes another Func delegate as input

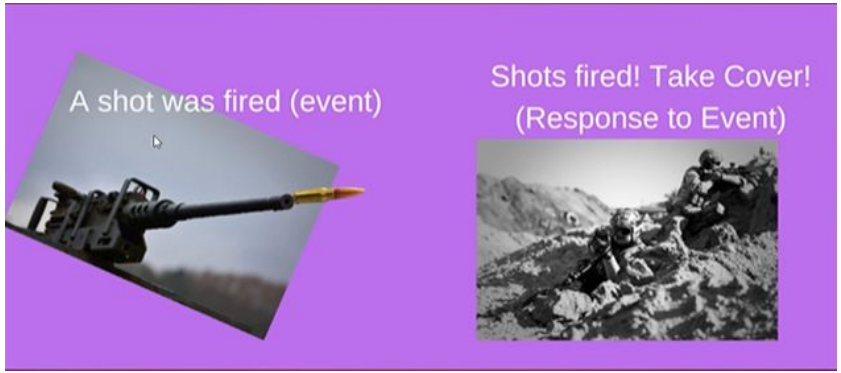
```
string[] names = { "Alice", "John", "Bobby", "Kyle", "Scott", "Tod", "Sharon", "Armin", "George" };

Func<string[], Func<string, bool>, List<string>> extractStrings = (arr, filter) =>
{
    List<string> result = new List<string>();

    for (int i = 0; i < arr.Length; i++)
    {
        if (filter(arr[i]))
        {
            result.Add(arr[i]);
        }
    }

    return result;
};
```

Assignment

- ✔ Create a class BookRepository that has a method that returns a list of Book objects. A Book has an ISBN, title, and price.
- ✔ Return all the books cheaper than 10 dollars using a delegate method
- ✔ Use a lambda expression for the solution

Events

- A mechanism for communication between objects
- Used in building **Loosely Coupled Applications**
- Helps extending applications
- Publishers are those who fire the event
- Subscribers responds to event



Creating Events

Publisher

- Delegate matching the Event signature
- Event of the same type as the Delegate
- Raise the event at some point

Subscriber

- A method with matching signature
- Subscribed to the event

Example

- We create a class Shooter with a shoot event, which is the Publisher.

1. Delegate matching the Event signature
   public delegate void KillingHandler(object sender, EventArgs e);
2. Event of the same type as the Delegate
   public event KillingHandler ShotsFired;

3. Raise the event
   ShotsFired.Invoke(this, EventArgs.Empty);

Example

4. A method with matching signature

```
static void KilledEnemy(object source, EventArgs e)
 {
    Console.WriteLine($"I killed an enemy ");
 }
```

5. Subscribed to the event

```
shooter.ShotsFired += KilledEnemy;
shooter.OnShoot();
```

EventHandler

- Instead of the KillingHandler delegate we created we can simply use the EventHandler built in delegate

public event EventHandler ShotsFired;

Extension Methods

- Allows us to add methods to an existing class without

- Changing the source code
- Creating a new class that inherits from it

Example

- We want to create an extension method for String class that returns n number of characters from the string

```
namespace ExtensionMethods
{
    public static class StringExtensions
    {
        public static string Shorten(this String str, int numberOfWords)
        {
            if (numberOfWords < 0)
                throw new ArgumentOutOfRangeException("numberOfWords should be greater than or equal to 0.");

            if (numberOfWords == 0)
                return "";

            var words = str.Split(' ');

            if (words.Length <= numberOfWords)
                return str;

            return string.Join(" ", words.Take(numberOfWords)) + "...";
        }
    }
}
```

Extension methods

- Extension methods exist in the same namespace with the class that created them
- Extension methods are static methods
- Use extension methods only when you have to.
- **IEnumerable** interface contains extension methods to use with LINQ

Assignment

- ✔ Add an extension method that sorts and array and use it to sort an array with int values.
- ✔ Add a new boolean parameter to the extension method reverse, that will reverse the array if set to true.

References

- Wouter de Kort, Exam Ref 70-483: Programming in C#, Microsoft Press, Inc., 2013
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-operator
- https://www.tutorialsteacher.com/linq/linq-lambda-expression

- https://www.c-sharpcorner.com/article/anonymous-methods-in-c-sharp/

QUESTIONS