Shortcuts

Sunday, August 29, 2021 4:17 PM

Shift + Ctrl + Alt + B
- Get to the implementation of a method

CTRL+SPACE (JAVA)
- ALT+ENTER

CTRL + F5
- Run Application

F5
- Run application in debug mode

SHIFT + B
- Build application

View > Object Browser
- Look at all the objects in the current application

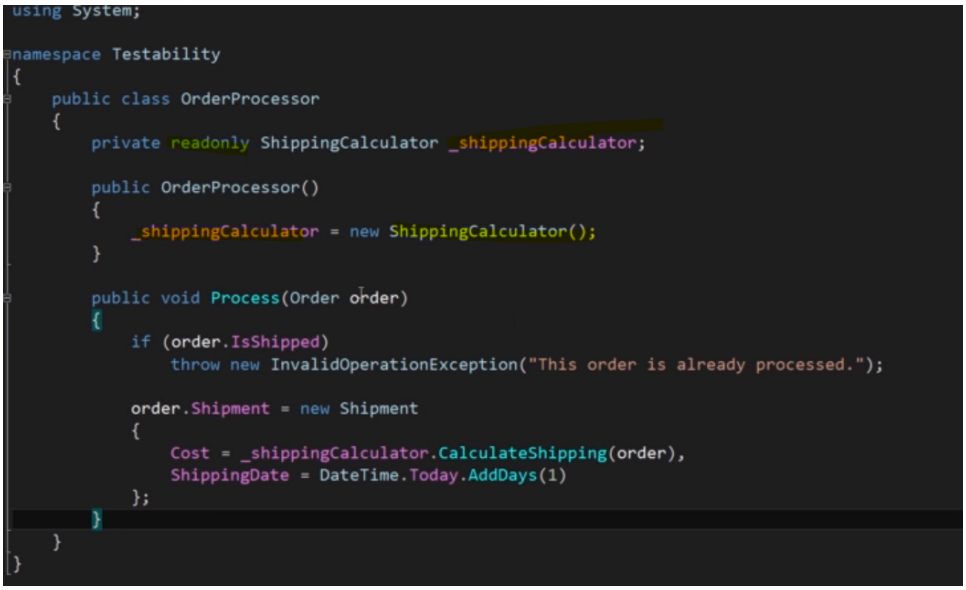
CTRL+D
- Creates a duplicate of line above

CTRL + SHIFT + N
- Go everywhere in the application by taping name of file

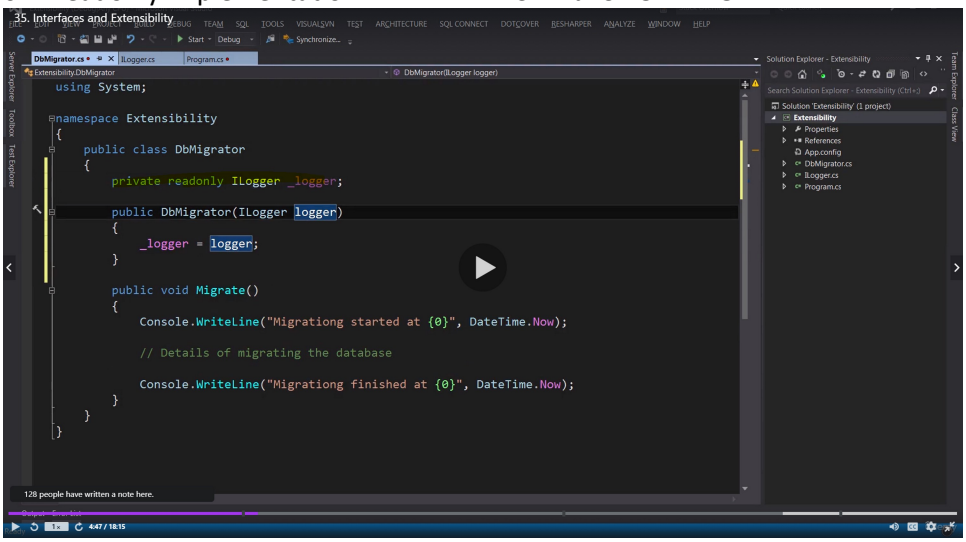
General

Sunday, August 29, 2021 4:21 PM

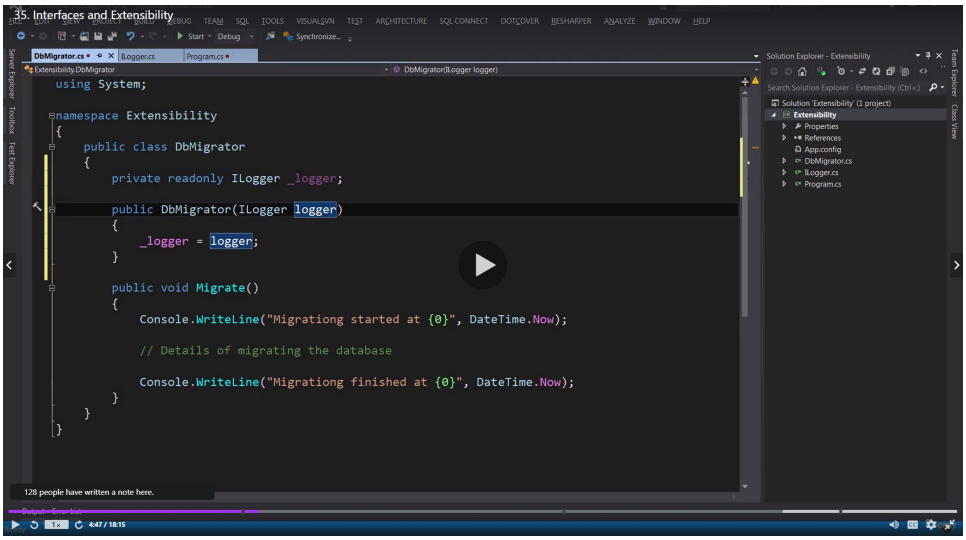
00 - readonly implementation.PNG



01 - readonly implementation - DEPENDENCY INJECTION.PNG

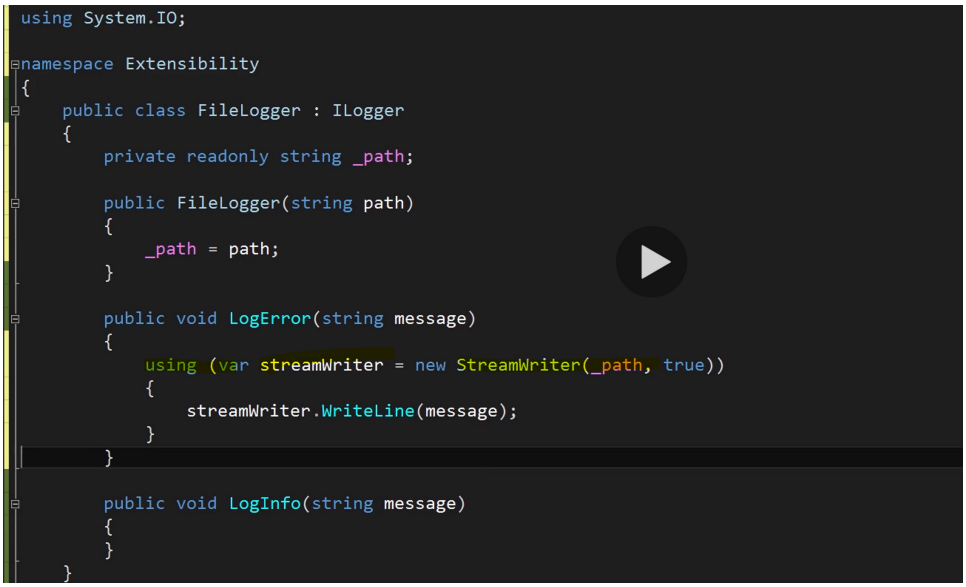


01 - 1 - readonly implementation.PNG



02 - Applying USING keyword

- To dispose of object that are not necessary after the usage to avoid memory leakage
- It is implemented here to destroy the new object StreamWriter after usage
- disposes the method
- calls method.dispose



03 - Avoid repeating code.PNG

```
public void LogError(string message)
{
    Log(message, "ERROR");
}

public void LogInfo(string message)
{
    Log(message, "INFO");
}

private void Log(string message, string messageType)
{
    using (var streamWriter = new StreamWriter(_path, true))
    {
        streamWriter.WriteLine(messageType + ": " + message);
    }
}
```

04 - Pass Multiple Interfaces

- It is possible to pass multiple interfaces in C#

Inheritance - Reusability

Interfaces - Let other method implement an interface

- Interfaces are contracts that need to pass a specific method and implment it
- You can't pass fields
- Interfaces provide polymorphism

```
using System;

namespace MultipleInheritance
{
    public class TextBox : UiControl, IDraggable, IDroppable
    {
        public void Drag()
        {
            throw new NotImplementedException();
        }

        public void Drop()
        {
            throw new NotImplementedException();
        }
    }

    public class UiControl
    {
        public string Id { get; set; }
        public Size Size { get; set; }
        public Position TopLeft { get; set; }

        public virtual void Draw()
        {
        }
```

05 - Add elements to private readonly

- add elements to the field by opening a a public method that passes interface variable to the application

```
namespace Polymorphism
{
    public class VideoEncoder
    {
        private readonly IList<INotificationChannel> _notificationChannels;

        public VideoEncoder()
        {
            _notificationChannels = new List<INotificationChannel>();
        }

        public void Encode(Video video)
        {
            // Video encoding logic
            // ...

            foreach (var channel in _notificationChannels)
                channel.Send(new Message());
        }

        public void RegisterNotificationChannel(INotificationChannel channel)
        {
            _notificationChannels.Add(channel);
        }
    }
}
```

06 - Protect list readonly

- Protect the implementation by using IEnumerable so the user does not go directly to the list that is being implemented

```
public class Workflow : IWorkflow
{
    private readonly List<ITask> _tasks;

    public Workflow()
    {
        _tasks = new List<ITask>();
    }

    public void Add(ITask task)
    {
        _tasks.Add(task);
    }

    public void Remove(ITask task)
    {
        _tasks.Remove(task);
    }

    public IEnumerable<ITask> GetTasks()
    {
        return _tasks;
    }
}
```

PASS VARIABLE ARGUMENTS IN ARRAY

- In java is known as varargs (variable arguments)
    - It is passed as (int ... num)
- In C# is also nown as varargs
    - It is passed as
    - (params int[] num)

MAIN CLASSES IN C#

- System.Collections.--Generic.

Primitves in C# are not nullable by default

- To make them nullable you must use '?' next to it
- Int?

```
0 references
private int? CallCalculator(T getProduct)
{

    return null;
}
```

Argument of a method is the name to pass the parameter

PREDICATES:

- Use for collections
- Delegates which point of a method get to a book and returns a boolean if a given condition was satisfied

```
static bool IsCheaperThan10Dollars(Book book)
{
    return book.Price < 10;
}
```

PIC: Predicate Usage

```
static void Main(string[] args)
{
    var books = new BookRepository().GetBooks();

    books.FindAll(IsCheaperThan10Dollars)


}

static bool IsCheaperThan10Dollars(Book book)
{
    return book.Price < 10;
}
        (parameter) Book book
```

- FindAll is a predicate that will go through the method and return true if the condition is satisfied
- It will iterate over all the objects of type book and each object will pass through the IsCheaperThan10Dollars method

- If the condition is satisfied, it will return that specific object

PIC: Iterate to get all the books under 10 dolalrs

```
static void Main(string[] args)
{
    var books = new BookRepository().GetBooks();

    var cheapBooks = books.FindAll(IsCheaperThan10Dollars);

    foreach (var book in cheapBooks)
    {
        Console.WriteLine(book.Title);
    }

}
```

String Class

- Ienumerable is an array and many extensions are derived from this class
- I Enumerable<> takes from any collection or array of primitives and works like Iterator<> in Java
- String.Split (Split)
    - Separates an string using a separator that is present in the string
- StringArray.Take (Take)
    - From Ienumerable<>
    - Returns a specified number of continues elements from the start of a

sequence
    - EX: Take(4) if the array has a length of 6, it will only return an Ienumerable array of 4 elements from that string
- String.Join
    - Takes a StringArray and joins them together in 1 string with the stablished separator

IENUMERABLE <T>

- Like iterator<> in java
- Use to iterate over a collection
- Can use linq with it
- Best approach is to set everything into a list and apply the Ienumerable to that list

```
public class BookRepository
{
    public IEnumerable<Book> GetBooks()
    {
        return new List<Book>
        {
            new Book() {Title = "ADO.Net Step by Step", Price = 5 },
            new Book() {Title = "ASP.NET MVC", Price = 9.99f },
            new Book() {Title = "ASP.NET Web API", Price = 12 },
            new Book() {Title = "C# Advanced Topics", Price = 7 },
            new Book() {Title = "C# Advanced Topics", Price = 9 }
        };
    }
}
```

Struct Vs Classes

Struct	Classes
reference type such as collections	value types such as primitives, pass by value
You do not need the new keyword, It is optional	You NEED the "new" keyword to create an Instance of the object to use their methods
Each variable contains its own copy Of the data, an operation on one variable Does not affect the other A = 5 B = A (B does not affect A)	More than 1 variable can contain the reference Of the same object, the operation on 1 variable Will affect the copy as well List<int> a = new List<int>(); List<int> b = a; (b affects a and viceversa)
Do NOT have a default constructor	Does have a default constructor
Cannot be a base class	Can be a base class
Have a value in memory, usually alocated	Have a reference to an instance in memory,

In Stack	Usually alocated in the heap, so 'new' is mandatory To alocate a new space in memory
In STACK	IN HEAP

Tertiary Operators

- Create a condition in the construct of a value
  (If Condition) ? TRUE (do something) : FALSE (do something)

  DateTime date = (date != null) ? Date.GetValueorDefault() : DateTime.Today;

Disposable

- Idisposable interface
- Has method dispose()
- FileReader implements disposable interface
- Always make sure to dispose of the object to clean space in the finally block of any try catch block

Partial Classes

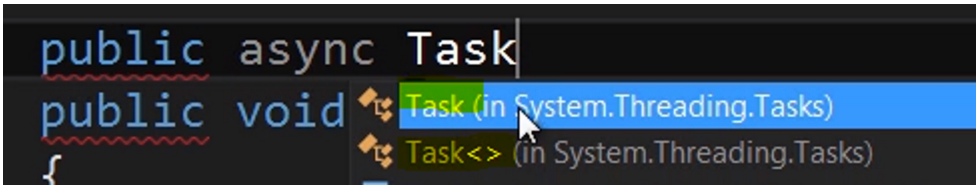
- As the name implies, it indicates the same class name in the same namespace
- It is use to split the class so multiple developers can work on it
- You can set different definitions (methods) of the class on multiple files that will be put together and run time
- The classes that are partial must have the "partial" keyword attached and they all must have the same accesibility

```
public partial class Employee
{
    public void DoWork()
    {
    }
}

public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

- The partial modifier can only appear immediately before the keywords class, struct, or interface.

```
public partial class Coords
{
    private int x;
    private int y;

    public Coords(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

public partial class Coords
{
    public void PrintCoords()
    {
        Console.WriteLine("Coords: {0},{1}", x, y);
    }
}

class TestCoords
{
    static void Main()
    {
        Coords myCoords = new Coords(10, 15);
        myCoords.PrintCoords();

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
// Output: Coords: 10,15
```

Generic and Non Generic Form of a method in C#



- Generic (Task<>)
    - If the method returns something like a string
- Non-Generic (Task)
    - If the method is void, you need to use the non-generic version of the method

Keywords

Monday, August 30, 2021 8:57 PM

Is and as

- Used to make sure a downcasting of an object is safe
- Return null if it is not

Return default()

- Default next to return
- Returns a defaulted value if the condition is met
- Use default if the value is not set
- Use in Generics

```
public T GetValueOrDefault()
{
    if (HasValue)
        return (T)_value;

    return default(T);
}
```

Need to set a nullable to pass the value as default

```
class Program
{
    static void Main(string[] args)
    {
        var number = new Nullable<int>(5);
        Console.WriteLine("Has Value ?" + number.HasValue);
        Console.WriteLine("Value: " + number.GetValueOrDefault());
    }
}
```

- Part of System.Nullable

Advanced

Sunday, August 29, 2021 4:48 PM

SECTION 2: C# Advanced Topics

06 - Generics

- Use to access any var type without having to box and unbox every time an object

- You can implement <T> on class or method level

- After that the implementation is the same as java

- You pass the elements at runtime therefore no boxing or unboxing

PIC: GENERIC Class

```
4 references
class GenericsPractice <T> //Need to pass the T for generics at the class level
{
    private readonly List<T> _elementList; //pass any T type

    2 references
    public GenericsPractice()
    {
        _elementList = new List<T>();
    }
    0 references
    public GenericsPractice(List<T> elementList)
    {
        _elementList = elementList;
    }

    2 references
    public void Add(T value)
    {
        _elementList.Add(value);
    }

    2 references
    public T this[int index] //Return anything that is stored in the given index
    {
        get => _elementList[index];
    }
}
```

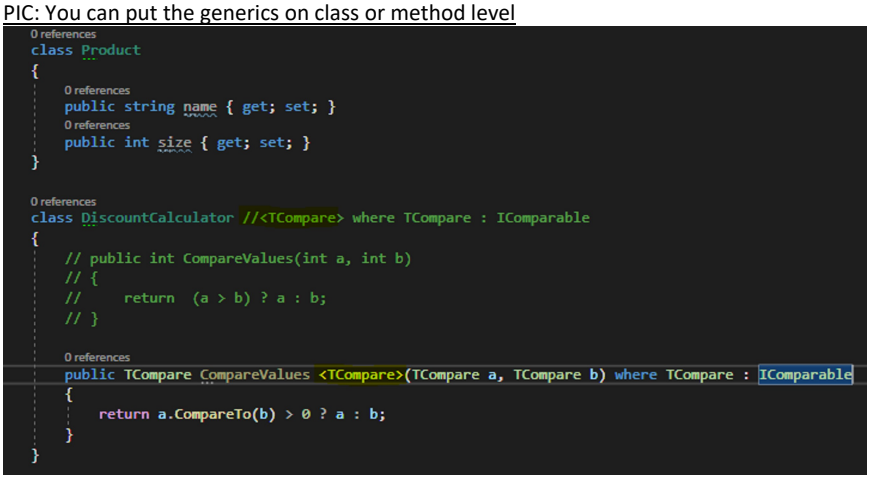
PIC: Implementation in MAIN

```
0 references
static void Main(string[] args)
{
    var numbers = new GenericsPractice<int>(); // need to define the object that is being passed in the
       instantiation
    numbers.Add(10);

    var book = new GenericsPractice<string>(); // passing string
    book.Add("One Book");

    Console.WriteLine(numbers[0]);
    Console.WriteLine(book[0]);

    var list = new List<int>() {1, 2, 3, 4, 5};
    Console.WriteLine("List Object 1: {0}, 2: {1}", list[0], list[1]);


}
```

PIC: Passing a dictionary generics

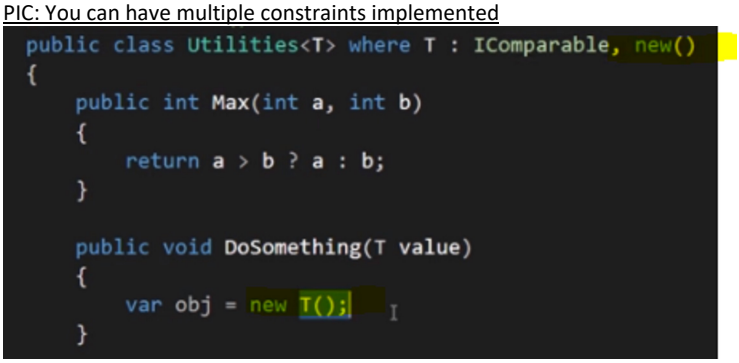
```
namespace Generics
{
    public class BookList
    {
        public void Add(Book book)
        {
            throw new NotImplementedException();
        }

        public Book this[int index]
        {
            get { throw new NotImplementedException(); }
        }
    }

    public class GenericDictionary<TKey, TValue>
    {
        public void Add(TKey key, TValue value)
        {

        }
    }

    public class GenericList<T>
    {
        public void Add(T value)
        {
```

PIC: You can put the generics on class or method level



```
0 references
class Product
{
    0 references
    public string name { get; set; }
    0 references
    public int size { get; set; }
}

0 references
class DiscountCalculator //<TCompare> where TCompare : IComparable
{
    // public int CompareValues(int a, int b)
    // {
    //     return  (a > b) ? a : b;
    // }

    0 references
    public TCompare CompareValues <TCompare>(TCompare a, TCompare b) where TCompare : IComparable
    {
        return a.CompareTo(b) > 0 ? a : b;
    }
}
```

PIC: You can have multiple constraints implemented



```
public class Utilities<T> where T : IComparable, new()
{
    public int Max(int a, int b)
    {
        return a > b ? a : b;
    }

    public void DoSomething(T value)
    {
        var obj = new T();
    }
```

Putting a constraint on T as Icomparable to restrict T to the interface IComparable

- Where T : Icomparable

```
public class Utilities
{
    public int Max(int a, int b)
    {
        return a > b ? a : b;
    }

    public T Max<T>(T a, T b) where T : IComparable
    {
        a.
    }
}
```

CompareTo (object obj):int
Equals
GetHashCode
GetType
ToString
Compares the current instance with another object of the same type and
the sort order as the other object.

Allows to use Comparable methods to make the modifications for generic parameters
With Genetics is better to use comparable or comparator to compare elements based on object and not on primitives

- You do not need a geneic class to make the implementation

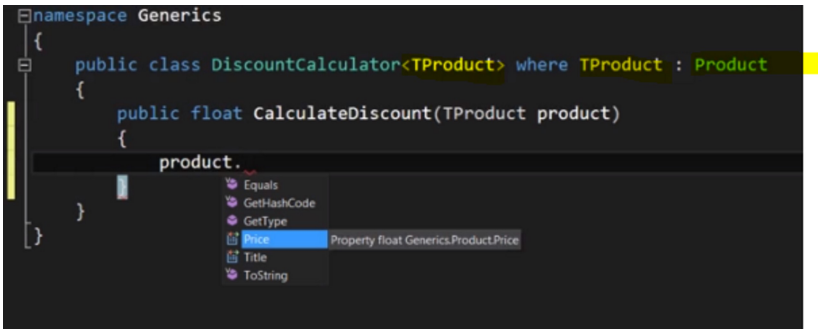
```
public class Utilities
{
    public int Max(int a, int b)
    {
        return a > b ? a : b;
    }

    public T Max<T>(T a, T b) where T : IComparable
    {

        return a.CompareTo(b) > 0 ? a : b);
    }
}
```

- If you put the generics on the class level you do not need o do it on the method level
- Generics always have <T> next to the first implementation
    - This <T> refer to a new parameter in the expression, like defining a new variable

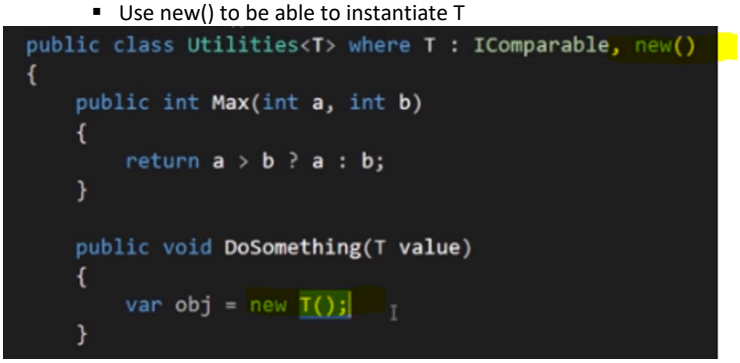
```
using System;

namespace Generics
{
    public class Utilities<T> where T : IComparable
    {
        public int Max(int a, int b)
        {
            return a > b ? a : b;
        }

        public T Max(T a, T b)
        {
            return a.CompareTo(b) > 0 ? a : b;
        }
    }
}
```

CONSTRAINTS

- Contraints allows to place restrictions on generics
- It allows access to the methods and properties of the target of the constraint
- It only allows the target to use the values of the contraint
- Types of Constraints:
    - Where T: Icomparable
        - Constraint to an interface
    - Where T : Product (Class Level)



    - Where T : struct (primitive)
        - Allows to pass data constructrs to nullables
    - Where T : new()
        - Use new() to be able to instantiate T



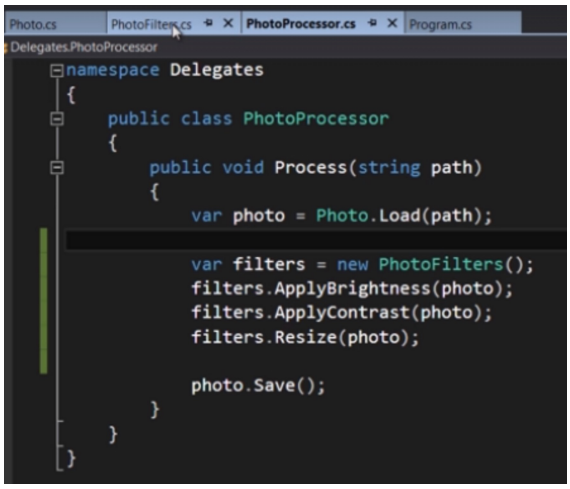
        - You can apply multiple constraints

07 - Delegates

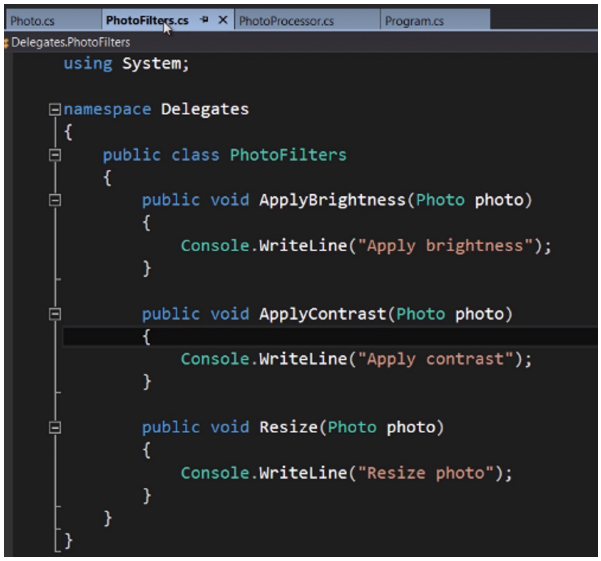
- Use to call methods
- Allows the creation of flexible applications
- Delegates are pointers with a method signature
- Methods that are applied by a delegate must contain the same method signature
- An object that knows how to call a method (or a group of methods)
- A reference to a function
- System.Action<T> and System.Func<T> belong to delegate
    - Func allows for lambda expressions

- The following framework is not extensible so we need to apply
  Delegates so the developer can create a new filter without having to

- Delegator must have the same parameter so you can define the same instructions on different
  Methods that have the same signature (access modifier)



Delete the filters
And pass delegate
instead



- We need to define a delegate as a method with the type of the method
  Included

```
public class PhotoProcessor
{
    public delegate void PhotoFilterHandler(Photo photo);

    public void Process(string path, PhotoFilterHandler filterHandler)
    {
        var photo = Photo.Load(path);

        filterHandler(photo);

        photo.Save();
    }
}
```

Wrapping the variables
With the delegate param

- Here we let the developer pass the element PhotoFiler, same as with
  An interface
- You don't need to recompile the program with delegate like interface

- Delegate methods are public static final

```
class Program
{
    static void Main(string[] args)
    {
        var processor = new PhotoProcessor();
        var filters = new PhotoFilters();
        PhotoProcessor.PhotoFilterHandler filterHandler = filters.ApplyBrightness;
        filterHandler += filters.ApplyContrast;

        processor.Process("photo.jpg", filterHandler);
    }
```

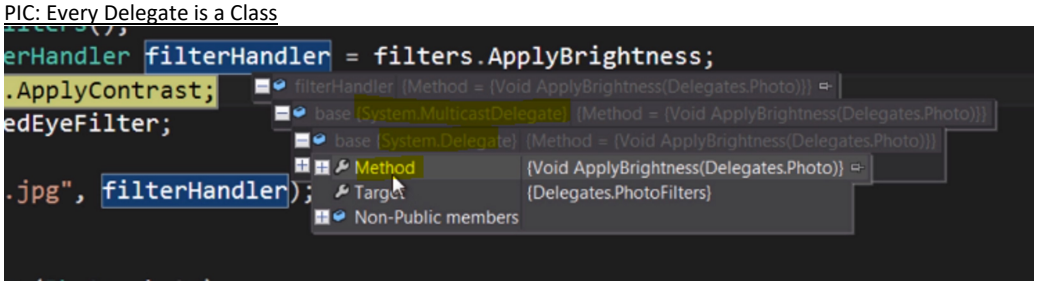
- Adding more filters through the hanlder
- You can store multiple filters in 1 variable

PIC: Creating new Filter for delegate "RemoveRedEyes"

- Delegate is pointing to the new method as it contains the same signature
  - (Photo photo)

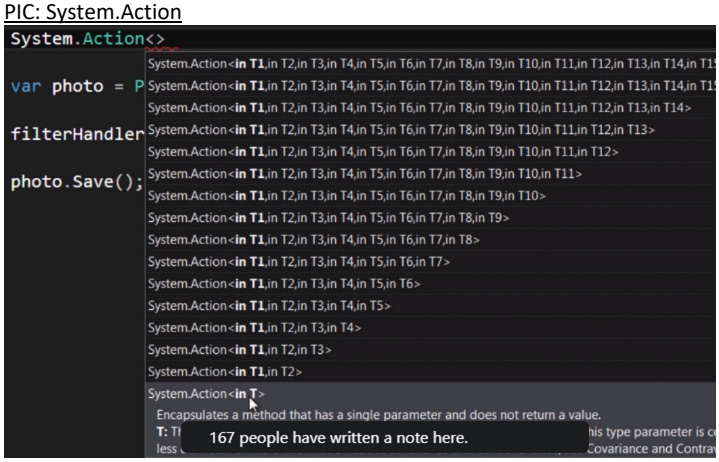
```
class Program
{
    static void Main(string[] args)
    {
        var processor = new PhotoProcessor();
        var filters = new PhotoFilters();
        PhotoProcessor.PhotoFilterHandler filterHandler = filters.ApplyBrightness;
        filterHandler += filters.ApplyContrast;
        filterHandler += RemoveRedEyeFilter;

        processor.Process("photo.jpg", filterHandler);
    }

    static void RemoveRedEyeFilter(Photo photo)
    {
        Console.WriteLine("Apply RemoveRedEye");

    }
}
```

PIC: Every Delegate is a Class

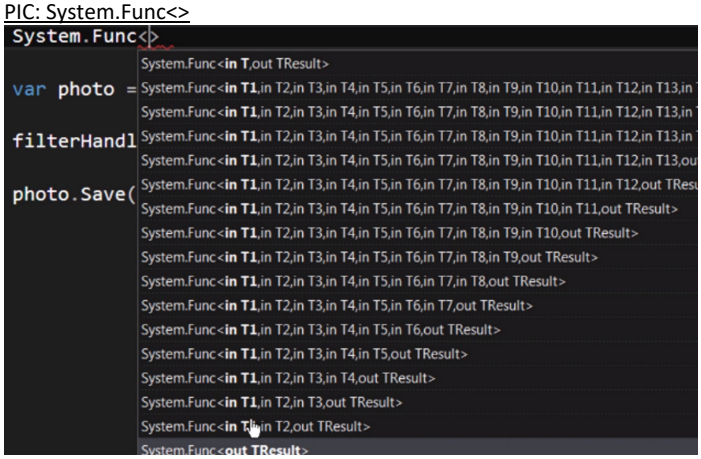


- It passes from System.MultiCastDelegate
  - It can pass multiple objects
  - Multiple function pointers
  - Delegate: It contains only 1 pointer
  - MultiDelegate: Multiple pointer

PIC: System.Action



PIC: System.Func<>



PIC: Passing Action to replace custom delegate

```
public class PhotoProcessor
{
    public void Process(string path, Action<Photo> filterHandler)
    {
        var photo = Photo.Load(path);

        filterHandler(photo);

        photo.Save();
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        var processor = new PhotoProcessor();
        var filters = new PhotoFilters();
        Action<Photo> filterHandler = filters.ApplyBrightness;
        filterHandler += filters.ApplyContrast;
        filterHandler += RemoveRedEyeFilter;

        processor.Process("photo.jpg", filterHandler);
    }

    static void RemoveRedEyeFilter(Photo photo)
    {
        Console.WriteLine("Apply RemoveRedEye");

    }
}
```

We Use Delegates for:

- Extensibility and flexibility
- Best for designing a framework
- Same as interface

Interface or Delegates:

- Personal preference
- Delegates for eventing design patern
    - Or if the caller does need access to other properties or methods

08 - Lambdas Expression

- It is like a method
- BUT I DOES NOT HAVE:
    - Access Modifier
    - Name
    - Return statement

- Use for convenience
- SYNTAX:
    - Args => expression
    - AS argument GOES TO to expression
    - In Lamba is common to just use one letter to refer to the arguments

- NOTE:
    - You don't need to define a functional interface to use lambdas,
        - It is usually understood by the compiler

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

            Console.WriteLine(Square(5));
        }

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

PIC: Passing delegate foor the method using lambda

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

PIC: Types of lambda argument passing

```
// args => expression

// () => ...
// x => ...
// (x, y, z) => ...
```

- If there are no arguments then pass ()
- If there is 1 argument just pass it without ()
- For multiple arguments, pass it in parenthesis separated by ,

PIC: Passing lambda through the scope of the variable

```
const int factor = 5;

Func<int, int> multipler = n => n*factor;

var result = multipler(10);

Console.WriteLine(result);
```

- Creating a function with lambda
- Passing argument into that function to print the result
- Use as any other method

PIC: lambda expression equivalent

- This reads as "As book goes to book price is less than 10"

```
var cheapBooks = books.FindAll(book => book.Price < 10);
```

- The var cheapBook is equivalent to :

```
static void Main(string[] args)
{
    var books = new BookRepository().GetBooks();

    var cheapBooks = books.FindAll(IsCheaperThan10Dollars);

    foreach (var book in cheapBooks)
    {
        Console.WriteLine(book.Title);
    }

}

static bool IsCheaperThan10Dollars(Book book)
{
    return book.Price < 10;
}
```

09 - Events and Delegates

It is used to let a method know of an incoming action based of a trigger

- Events are instances of a delegate to perform certain actions when it gets to that section of the code
- You don't need to use a delegate specifically, it can be done through the class
    - EventHandler<VideoEventArgs>
- We invoke a method using a delegate to indicate which method to use with the especific signature

```
namespace EventsAndDelegates
{
    public class VideoEncoder
    {
        // 1- Define a delegate
        // 2- Define an event based on that delegate
        // 3- Raise the event

        public delegate void VideoEncodedEventHandler(object source, EventArgs args);

        public event VideoEncodedEventHandler VideoEncoded;

        public void Encode(Video video)
        {
            Console.WriteLine("Encoding Video...");
            Thread.Sleep(3000);

            OnVideoEncoded();
        }

        protected virtual void OnVideoEncoded()
        {
            if (VideoEncoded != null)
                VideoEncoded(this, EventArgs.Empty);
        }
    }
```

- PIC: passing an event handler through the new .net framework

```
public delegate void VideoEncodedEventHandler(object source, VideoEventArgs args);

// EventHandler
// EventHandler<TEventArgs>

public event VideoEncodedEventHandler VideoEncoded;

public void Encode(Video video)
{
    Console.WriteLine("Encoding Video...");
    Thread.Sleep(3000);

    OnVideoEncoded(video);
}
```

```
public delegate void VideoEncodedEventHandler(object source, VideoEventArgs args);

// EventHandler
// EventHandler<TEventArgs>

public event EventHandler VideoEncoded;

public void
{
    Console
```

EventHandler
EventHandler<>
AssemblyLoadEventHandler
ConsoleCancelEventHandler
ResolveEventHandler

Delegate void System.EventHandler
Represents the method that will handle an event that has no event data.

- PIC: No need to creata a delegate ; EventHandler Takes care of the delegate

```
public delegate void VideoEncodedEventHandler(object source, VideoEventArgs args);

// EventHandler
// EventHandler<TEventArgs>

public event EventHandler<VideoEventArgs> VideoEncoded;
```

The rest of the class is the same

```
public class VideoEncoder
{
    // 1- Define a delegate
    // 2- Define an event based on that delegate
    // 3- Raise the event

    // EventHandler
    // EventHandler<TEventArgs>

    public event EventHandler<VideoEventArgs> VideoEncoded;

    public void Encode(Video video)
    {
        Console.WriteLine("Encoding Video...");
        Thread.Sleep(3000);

        OnVideoEncoded(video);
    }

    protected virtual void OnVideoEncoded(Video video)
    {
        if (VideoEncoded != null)
            VideoEncoded(this, new VideoEventArgs() { Video = video });
    }
}
```

PIC: VIDEO EVENT ARGS

```
namespace EventsAndDelegates
{
    public class VideoEventArgs : EventArgs
    {
        public Video Video { get; set; }
    }

    public class VideoEncoder
    {
        // 1- Define a delegate
        // 2- Define an event based on that delegate
        // 3- Raise the event

        public delegate void VideoEncodedEventHandler(object source, VideoEventArgs args);

        public event VideoEncodedEventHandler VideoEncoded;

        public void Encode(Video video)
        {
            Console.WriteLine("Encoding Video...");
            Thread.Sleep(3000);

            OnVideoEncoded();
        }
```

- **SELF NOTES**
  - Events are wrappers for delegates, delegates are pointers of a method and events are triggers for this points when certain conditions are met or when they are invoked
  - New .Net Framework have implement an EventHandler class to trigger an event without using a delegate and the rest is the same

```
/*

public delegate void VideoEncoderEventHandler(Object source, VideoEventArgs args);
public event VideoEncoderEventHandler VideoEncoded;

*/

//Alternatively you can pass an Event Hanlder insetead of a delegate
/*
     You can avoid placing a delegate every time you want to pass an event
    by simply passing an EventHandler<TypeOFEvent> instead
     */
public EventHandler<VideoEventArgs> VideoEncoded;
```

  - The signature for the event is void (Objec source, EventArgs args)
    - Event args can be extended by other classes to pass another objects and us instead, such as VideoEncoderHandler
  - You need to pass the event in the publisher method to trigger as any other method
  - Events are invoked like delegates, in the left hand side must be the event followed by a += and then the name of the new class that is receiving the event like the MailService

```
class ProgramEvent
{
    0 references
    static void MainProgram(string[] args)
    {
        Encoder encoder = new Encoder(); // Publisher
        MailService mailService = new MailService(); //Subscriber

        encoder.VideoEncoded += mailService.MailServiceSubscriber;

        encoder.VideoEncode(new Video() { Title = "Any Movie" });

    }
}
```

    - The MailServiceSubscriber must have the same signature of void (Objec source, EventArgs args)

```
namespace Advanced
{
    // Subscriber
    2 references
    class MailService
    {
        1 reference
        public void MailServiceSubscriber(Object source, VideoEventArgs args)
        {
            Console.WriteLine("Sending to the MailService Class: {0}", args.Video.Title);
        }
    }
}
```

- OnVideoEncoded - The method that passes the event
  - It is the VideoEnconded method with an on at the beginning to indicate when the event will be triggered
  - The method must have an access modifyer of restricted, so it can be implemented in the same class and subclasses

```
public void VideoEncode(Video video)
{
    Console.WriteLine("Encoding a video: " + video.Title);
    Thread.Sleep(millisecondsTimeout: 3000);
    OnVideoEncoded(video);
}

// Use the event VideoEncoded LN:26
1 reference
protected virtual void OnVideoEncoded(Video video) //Use the same name as the event with "On" at the beggining
{
    if(VideoEncoded != null) //Passing the event
        VideoEncoded(sender: this, e: new VideoEventArgs(){Video = video}); // pass this object (Encoder)
}
```

10 - Extension Methods

Extensions are used to provide extra functionality to sealed classes without changing the code
They are extrimely useful so you don't have to change a class
It is beter to implement extension methods instead of changing the source code of the application

For extensions you need the following

1. Set the class of the extension as static

```
public static class  Extension
```

2. Set the method of the class as satic and the signature must contain 'this' followed by the name of the class you want to extend and give it a name

```
1 reference
public static string Shorten(this string phrase, int numberOfWords)
{
```

   a. String takes the string property and loads the data, that might be why we are using "this string" to make a reference to the data that is being taken.
   b. The "phrase" name is reference to "String phrase = "Something" "

3. In the main method or method caller use the name of the main class followed by the name of the new extended method, the icon will be different for extensions

```
try
{
    var phrase = "Another thing";
    Console.WriteLine(phrase.Shorten(1));
}
catch (Exception e)
{
    Console.WriteLine(e.Message);

}
```

4. The new extension must be included in the method caller package(namespace)

```
using System;
using System.Collections.Generic;
using System.IO.Enumeration;
using System.Security.Cryptography.X509Certificates;
using System.Threading.Channels;
using Advanced.Delegator;
using Advanced.Extensions;
```

BONUS:

PIC: Extension Class

```
namespace Advanced.Extensions
{
    0 references
    public static class  Extension
    {
        2 references
        public static string Shorten(this string phrase, int numberOfWords)
        {
            if (phrase == null || phrase.Equals(""))
                throw new ArgumentNullException(paramName: "It is empty");


            var phraseSizeArray :string[] = phrase.Split(separator: ' ');


            if (phraseSizeArray.Length > numberOfWords)
                return string.Join(separator: " ", values: phraseSizeArray.Take(numberOfWords))+"...";

            return phrase;
        }
    }
}
```

PIC: Main Class

```
using System;
using System.Collections.Generic;
using System.IO.Enumeration;
using System.Security.Cryptography.X509Certificates;
using System.Threading.Channels;
using Advanced.Delegator;
using Advanced.Extensions;

namespace Advanced
{
    0 references
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            try
            {
                var phrase = "Another thing";
                Console.WriteLine(phrase.Shorten(1));
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);

            }

        }
    }
}
```

11 - LINQ

- Language Integrated Query
- Natively query objects in C#
- For
    - Collections (LINQ to Objects)
    - Databases (LINGQ to Entities) -> Entity Framework

- XMS (LINQ to XML)
- ADO.NET Data Sets (LINQ to Data Sets)

- LINQ goes hand in hand with Func<> Predicates
  - Func<> is a delegate that points to a method
  - Delegates are like interfaces
  - Predicates are boolean methods that return true if a condition has been fullfilled
  - Func allows the use of lambdas which are anonymos methods with no name of access modifyer
    - () => {}; //For any argument that goes in any condition

- PIC: LINQ - Where uses a predicate Function and extends from Ienumerable

```
static void Main(string[] args)
{
    var books = new BookRepository().GetBooks();
    var cheapBooks = books.Where(b => b.Price < 10);

    foreach (var book in cheapBooks)
        Console.WriteLine(book.Title + " " + book.Price);
}
```

- Useful LINQ methods
  - Where(lambda)
    - Filter a list of books that match a given condition
  - Single(lambda)
    - Same as where but only for 1 object

```
var book = books.Single(b => b.Title == "ASP.NET MVC");
```

  - SingleOrDefault(lambda)
    - Use if you are not sure if the element exists so it uses a default value of null if it does not exist instead of throwing an exception
  - First(lambda - predicate)
    - First element that meets the condition using a predicate
  - FirstOrDefault(lambda - predicate)
    - Return null instead of throwing an exception
  - Skip(numberOfSkips)
    - Skips elements that meet the condition
  - Take(numberOfTakes)
    - Takes the elements in the parenthesis of the method
    - Skip and take go hand in hand
  - Count()
    - Count the number of objects
  - Max(lambda - delegate)
    - Return the max count using a certain condition whitin the object
  - Order by() - lambda
  - Select() - lambda
    - Makes the var (cheappBooks) into an Enumerable<out T> where T is a string, thus, you don't need to define book.Title and book.Price. "Select" does this automatically
    - You only need to use select if you want to specify a property within the objects, otherwise you do not need it

  - NOTE: You can chain all this methods and make it look like SQL

- LINQ Syntax (They are all the same)
  - LINQ Extension Methods

```
// LINQ Extension Methods
var cheapBooks = books
                    .Where(b => b.Price < 10)
                    .OrderBy(b => b.Title)
                    .Select(b => b.Title);
```

  - LINQ Query Operators

```
// LINQ Query Operators
var cheaperBooks =
    from b in books
    where b.Price < 10
    orderby b.Title
    select b.Title;
```

- LINQ Extension Methods is more Powerful and versatile
- LINQ Query Operators is more user friendly but there are just enough keywords in the language

NOTE:

- Objects used in the LINQ topic
  - Book Object

```
namespace Linq
{
    public class Book
    {
        public string Title { get; set; }
        public float Price { get; set; }
    }
}
```

  - Methods that has an Ienumerable type and returns a list

```
public class BookRepository
{
    public IEnumerable<Book> GetBooks()
    {
        return new List<Book>
        {
            new Book() {Title = "ADO.Net Step by Step", Price = 5 },
            new Book() {Title = "ASP.NET MVC", Price = 9.99f },
            new Book() {Title = "ASP.NET Web API", Price = 12 },
            new Book() {Title = "C# Advanced Topics", Price = 7 },
            new Book() {Title = "C# Advanced Topics", Price = 9 }
        };
    }
}
```

  - Main Using an instance of BookRepository to call the GetBook() method

```
static void Main(string[] args)
{
    var books = new BookRepository().GetBooks();

}
```

NOTE2:

- This only works for arrays

```
0 references
static void Main(string[] args)
{
    string[] Names = new[] { "Bob", "Carl", "Betsy", "Malinda" };


    foreach(var name in Names.Where(n => n.Contains("B")).OrderBy(n=>n))
    {
        Console.WriteLine(name);
    }

    Console.ReadLine();
}
```

  - Unless you set it as an Ienmuerable<T> type and use the ones above

```
static void Main(string[] args)
{
    var books = new BookRepository().GetBooks();
    var cheapBooks = books.Where(b => b.Price < 10);

    foreach (var book in cheapBooks)
        Console.WriteLine(book.Title + " " + book.Price);
}
```

- For collections you have to use other keywords

```
static void Main(string[] args)
{
    List<string> names = new List<string>{ "Bob", "Carl", "Betsy", "Malinda" };

    names.FindAll(n => n.Contains("B"))
         .OrderBy(n=>n).ToList()
         .ForEach(n => Console.WriteLine(n));

}
```

12 - Nullable Types

- It is defined with a System wrapper Nullable<Type>

```
Nullable<DateTime> date = null;
```

- You can also use the shorthand '?' next to the type

```
DateTime? date = null;
```

- Methods of the Nullable Class
    - GetValueOrDefault
        - Gives the value if not null or a defaulted one generated by the system
    - HasValue
        - Boolean that throws true or false if it has value
    - Value
        - Provides value of the variable
        - If it null, it throws an argument exception

```
DateTime? date = null;

date.
```

Equals
GetHashCode
GetType
GetValueOrDefault (<no paramet
HasValue Retrieves the
ToString (DateTime de
Value

- Null Coalesce Operator
    - Works like tertiary operator (condition)? True : false

```
DateTime date3 = (date != null) ? date.GetValueOrDefault() : DateTime.Today;
```

- (hasValue) ?? (doesNotHasValue)

```
DateTime? date = null;
DateTime date2 = date ?? DateTime.Today;
```

- BONUS:
  - No tertiary or Coalesce operators

```
DateTime? date = null;
DateTime date2;

if (date != null)
    date2 = date.GetValueOrDefault();
else
    date2 = DateTime.Today;
```

13 - Dynamic

- Replaces Reflections
  - Reflection is imilar to Java to access the properties of an object
  - Use method "Optimize" to invoke the element
- Dynamic keyword
  - Allows the variable to resolved at run-time instead of compile-time
  - Dynamic can be anything until it gets to run-time
  - At runtime dynamic will be changed to the appropriate type
  - You can convert from dynamic to any time at compile-time

```
dynamic a = 10;
dynamic b = 5;
var c = a + b;
```

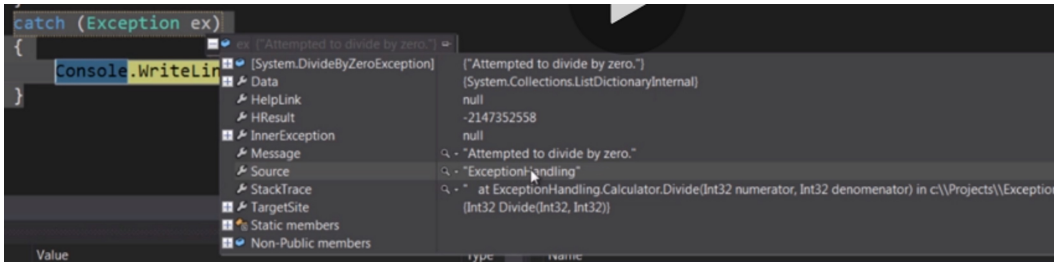
14 - Exception Handling

- When you run the application, the console shows the stack trace, which is the sequence of events where the error occurred
- You handle exceptions with the try cath block
  - Catch is only executed if an error is going to happen
  - In catch, throw; displays the error so it is optional

```
try
{
    var calculator = new Calculator();
    var result = calculator.Divide(5, 0);
}
catch (Exception)
{
    Console.WriteLine("Sorry, an unexpected error occurred.");
}
```

- You should always have a global try catch block to avoid the application from crashing, the ideal place is the closest to the main method as possible

- PIC: Exception stack trace



  - Data: When there is multiple data types
  - Message: The message associated with the exception
  - Source: usually the .dll if there are multiple files in the application
  - StackTrace: Stack in reversed order
    - Click on magnifier to open the text visualizer of the stacktrace
  - TargetSize: Method where the exception happened

- <u>Multiple Try/Catch Blocks</u>
  - You can have multiple catch blocks, they must be organized from most specific to most generic
  - Check Object Browser for a particular exception class to see the levels of exceptions
  - EX: DevideByZeroException -> ArithmeticException

```
try
{
    var calculator = new Calculator();
    var result = calculator.Divide(5, 0);
}
catch (DivideByZeroException ex)
{

}
catch (ArithmeticException ex)
{

}
catch (Exception ex)
{
    Con
}
```

class System.Exception
Represents errors that occur during application execution.

```
:ted error occurred.");
```

- <u>Try Catch Finally Block</u>
  - Finally block runs rgardless of how the execution when
  - It is use for manually cleanup
  - Idisposable interface to dispose of allocated resources
  - Use it for the class that unmanages resources

```
static void Main(string[] args)
{
    var streamReader = new StreamReader(@"c:\file.zip");
    try
    {
        var content = streamReader.ReadToEnd();
    }
    catch (Exception ex)
    {
        Console.WriteLine("Sorry, an unexpected error occurred.");
    }
    finally
    {
        streamReader.Dispose();
    }

}
```

- Everytime you open a new connection or open a new file, make sure to dispose of the object in the finally block regardless of the exception

- The "using" keyword already takes care of the finally .dispose() method
  - Using already calls a finally block
  - The try catch should always be present regadless to handle the exception, using only takes care of the disposal

```
try
{
    using (var streamReader = new StreamReader(@"c:\file.zip"))
    {
        var content = streamReader.ReadToEnd();
    }
}
catch (Exception ex)
{
    Console.WriteLine("Sorry, an unexpected error occurred.");
}
```

- <u>Creating Custom Exceptions</u>
  - Extend the Exception class
  - Create a new constructor and map it to the parent using the base keyword
  - We pass the innerException in the custom exception for debugging purposes

```
public class YouTubeException : Exception
{
    public YouTubeException(string message, Exception innerException)
        : base(message, innerException)
    {
    }
}
```

  - Passing the Exception in Main

```
try
{
    // Access YouTube web service
    // Read the data
    // Create a list of Video objects
}
catch (Exception ex)
{
    // Log

    throw new YouTubeException("Could not fetch the videos from YouTube.", ex);
}

return new List<Video>();
```

15 - Asynchronous Programming with Async/Await

- Synchronous Program Execution
  - Program is executed line by line, one at a time
  - When a function is called, program execution has to wait until the function returns.

- Asynchronous Program Execution
  - Program executes without waiting
  - It runs everything and then continues for the piece of code that is missing at the end witout delays
  - EX: Web Browser, Windows Media Player
    - Without asyncronous the UI will freeze when reading from videos

- When Do we use Asynchronous Programming?
  - Accessing the Web
  - Working with files and databases
  - Working with images

- How to Implement Asynchronous Programming?
  - Traditionally by:
    - Multi-threading
    - Callbacks
      - Tradiional methods are complex and requiere more knowledge
  - New Approach since .NET 4.5
    - Async/Wait
      - Task base asynchronous model

- Example of Syncrhonous Programming
  - Just normal programming practices of reading one line at a time

```
public class BrowserMethods
{
    1 reference
    public void DownloadHtml(string url)
    {
        WebClient webClient = new WebClient();
        var html = webClient.DownloadString(url);

        string filePath = @"D:\Users\stepb\Desktop\VisualStudio Projects\00-UDEMY\AsynchronousProgramming\Readers\";
        string fileName = "result.html";

        using (StreamWriter sw = new StreamWriter(new FileStream(filePath+fileName,FileMode.CreateNew)))
        {
            sw.Write(html);
        }
    }
}
```

- Example of Asynchronous Porgramming
  - The method should contain the "async" keyword and the naming should have a "Async" suffix at the end, the declaration will remain the same.
  - The method caller because a Task now instead of void, or Task<> if it has a retun type (string)

```
public async Task DownloadHtmlAsync(string url)
{
```

  - Change the methods that is being implemented in this async method to async as well.
    - In .NET 4.5 the methods that are related to Async behavior have an async overload, they end with Async as well

```
public async Task DownloadHtmlAsync(string url)
{
    var webClient = new WebClient();
    var html = webClient.DownloadString(url);

    using (var streamWr
    {
```

DownloadString (Uri address):stri
DownloadStringTaskAsync Downloads the
DownloadStringAsync (string address):
DownloadStringCompleted

- □ Normal "Async" is an old version which has nothing to do with the asyn model for this method (maybe others too), use "TaskAsync"

```
var html = webClient.DownloadString(url);

using (var streamWr
```

DownloadString
DownloadStringTaskAsync (Uri address):Task<string>

- Any Async methods that involve a task must have the "await" keyword atached

```
var html = await webClient.DownloadStringTaskAsync(url);
```

- □ "await" Keyword:
  - ® Await is a marker for the compiler
  - ® Indicates this operation is costly and takes time
  - ® It will allow control back to the caller inmediatly while the task is processed
  - ® The thread will not be blocked by the compiler
  - ® It indicates at some point the task will be completed and he runtime will be aware of it because of the await
  - ® The compiler goes back to the function after it gives back control to the caller - DownloadHtml
  - ® Let the compiler know that the method is going to take some time
  - ® If the data is a "Task<string>" with the "await" kword it becomes a final "string", otherwise it will remain as a "task" instead of a string
  - ® Await can only exist in a asyn method
  - ® Await only means is waiting for execution so the control is always given back to the caller

```
1 reference
private async void button1_Click(object sender, EventArgs e)
{
    BrowserMethods browser = new BrowserMethods();

    var getHtmlTask = browser.GetHtmlAsync
        (url: "https://stackoverflow.com/questions/18547862/print-textbox-contents-in-c-sharp");
    var html :string = await getHtmlTask;
    MessageBox.Show(text: html.Substring(startIndex: 0, length: 50000));
}
```

    - ◇ This is possible because when you give the control from a task you get the string, so it is the same as putting async in the button caller

```
1 reference
private async void button1_Click(object sender, EventArgs e)
{
    BrowserMethods browser = new BrowserMethods();

    var html :string = await browser.GetHtmlAsync(url: "https://stac
    MessageBox.Show(text: html.Substring(startIndex: 0, length: 50000));
}
```

  - ® You can do some work in between await operation that is not dependent of the await operations

```
/*
 * Returning string - Asynchronous
 */
var getHtmlTask = browser.GetHtmlAsync(url: "https://stackoverflow.com/ques

//You can include multiple boxes while the tasks awaits execution
MessageBox.Show(text: "Awaiting for the task to complete");

var html :string = await getHtmlTask;
MessageBox.Show(text: html.Substring(startIndex: 0, length: 50000));
```

  - ® You can use await for method that cannot wait to be executed until the result is ready, in that way you can use the thread to its fullest potential

- PIC: Example of async method that returns nothing

```
/**
 * For Asynchronous programming you have to follow certain convention
 * 1. create a method with the async keyword,
 *    the method should have the name ending with "Async" suffix
 * 2. it should have he same parameters, just changing the declaration of the method
 */
// Method with "async" keyword + Task + "Async" sufix
1 reference
public async Task DownloadHtmlAsync(string url)
{
    WebClient webClient = new WebClient();
    //method that is being implemented to download the string with "await" keyword and Async version of the method
    var html :string = await webClient.DownloadStringTaskAsync(url);

    string filePath = @"D:\Users\stepb\Desktop\VisualStudio Projects\00-UDEMY\AsynchronousProgramming\Readers\";
    string fileName = "resultAsync.html";

    using (StreamWriter sw = new StreamWriter(new FileStream( path: filePath + fileName, FileMode.CreateNew)))
    {
        // await + Async version of the Write method in the reader
        await sw.WriteAsync(html);
    }
}
```

- Returning anything instead of void
  - Get the method as Task<string> (anything)
  - You still need to away the async task

```
public async Task<string> GetHtmlAsync(string url)
{
    var webclient = new WebClient();
    return await webclient.DownloadStringTaskAsync(url);
}
```

  - Task<String> does not contain a Substring method as it is not the final result, just a task
    - In this case you need to set the variable that is calling the method as "await" and place the parent method (caller) as async

```
var html = await GetHtmlAsync("http://msdn.microsoft.com");
MessageBox.
```
The 'await' operator can only be used in a method or lambda marked with the 'async' modifier

```
1 reference
private async void button1_Click(object sender, EventArgs e)
{
```

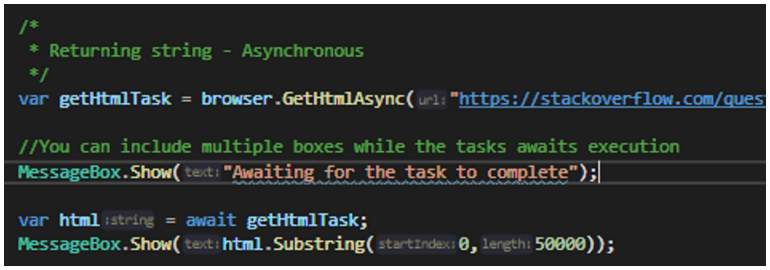
    - Final Result In the button_click method

```
1 reference
private async void button1_Click(object sender, EventArgs e)
{
    BrowserMethods browser = new BrowserMethods();

    var html :string = await browser.GetHtmlAsync( url: "https://stackoverflow.com/questions/18547862/print-textbox-contents-in-c-sharp");
    MessageBox.Show( text: html.Substring( startIndex: 0, length: 50000));
}
```

    - You can only use the "await" keyword in a "async" method
    - Therefore it is also posible to change the call in another vairable to use it as async variable

```
1 reference
private async void button1_Click(object sender, EventArgs e)
{
    BrowserMethods browser = new BrowserMethods();

    var getHtmlTask = browser.GetHtmlAsync
        ( url: "https://stackoverflow.com/questions/18547862/print-textbox-contents-in-c-sharp");
    var html :string = await getHtmlTask;
    MessageBox.Show( text: html.Substring( startIndex: 0, length: 50000));
}
```

  - PIC: Having multiple messages for the user while it awaits for the task to complete

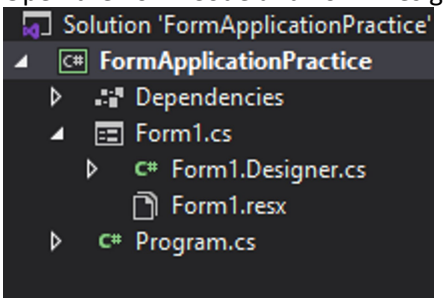

/*
* Returning string - Asynchronous
*/
var getHtmlTask = browser.GetHtmlAsync(url: "https://stackoverflow.com/ques
//You can include multiple boxes while the tasks awaits execution
MessageBox.Show(text: "Awaiting for the task to complete");
var html :string = await getHtmlTask;
MessageBox.Show(text: html.Substring(startIndex: 0, length: 50000));

Forms Apps Basics

Thursday, September 9, 2021 5:06 PM

- Forms are used to create desktop applications in VS

- Open the Form Code and Form Designer

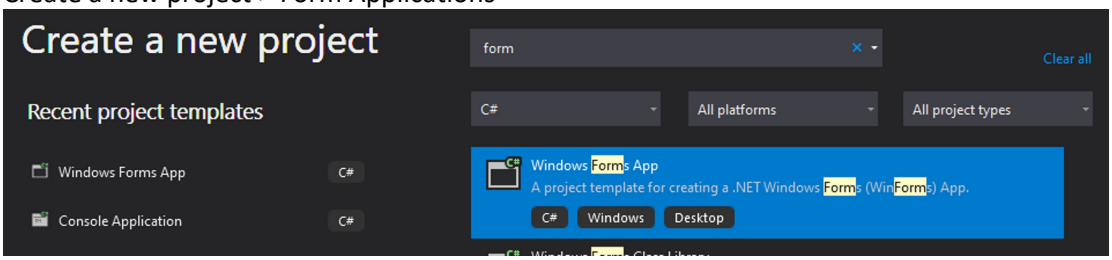


  - Right Click on the designer and select either open code or open designer
    - View Code (F7)
    - View Designer (Shift + F7)
  - In the designer you can see your form blank page and add objects as needed
  - In the designer you double click to open the normal Form.cs to make the necessary changes to the particular objects within the form

VS FormApplications

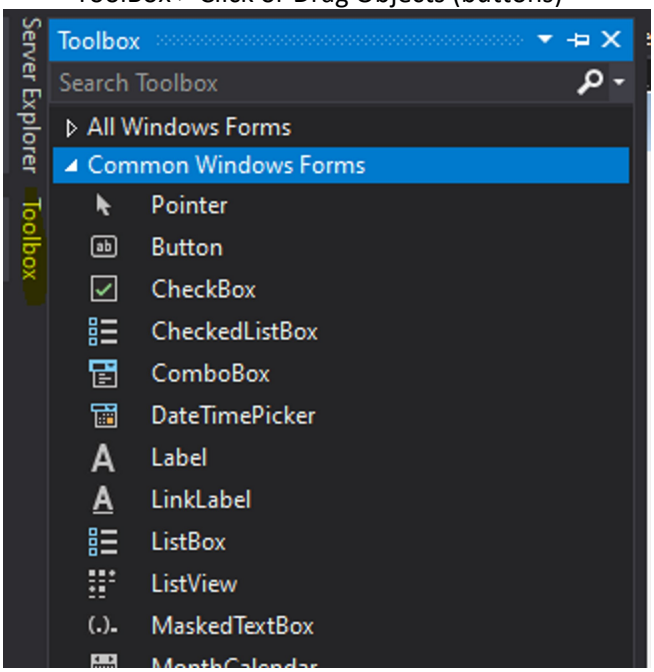
Thursday, September 9, 2021 4:59 PM

Create a new project > Form Applications

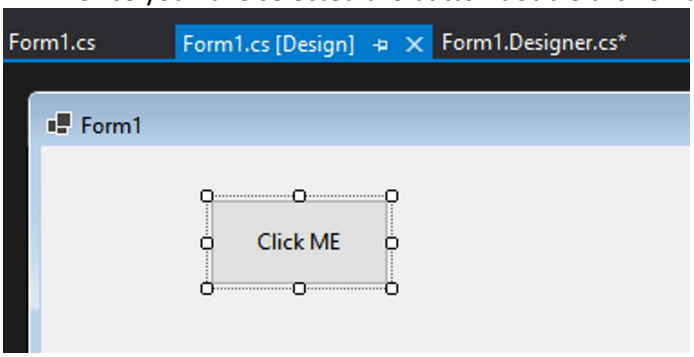


A new form is populated by default

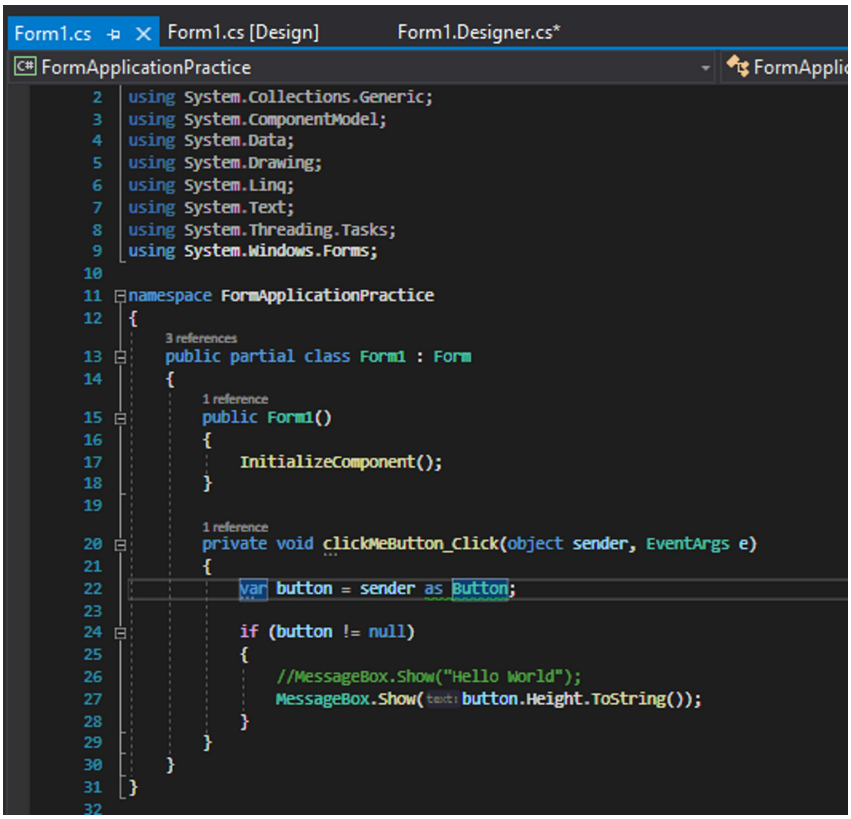
- ToolBox > Click or Drag Objects (buttons)



- Once you have selected the button double click on the buton in the Form



- This will open a new Form.cs where you can make changes to the behavior of the object


Form1.cs
Form1.cs [Design]
Form1.Designer.cs*
FormApplicationPractice
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace FormApplicationPractice
{
3 references
public partial class Form1 : Form
{
1 reference
public Form1()
{
InitializeComponent();
}
1 reference
private void clickMeButton_Click(object sender, EventArgs e)
{
var button = sender as Button;
if (button != null)
{
//MessageBox.Show("Hello World");
MessageBox.Show(button.Height.ToString());
}
}
}
}