

# Classes



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# Introduction to Classes



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# Agenda

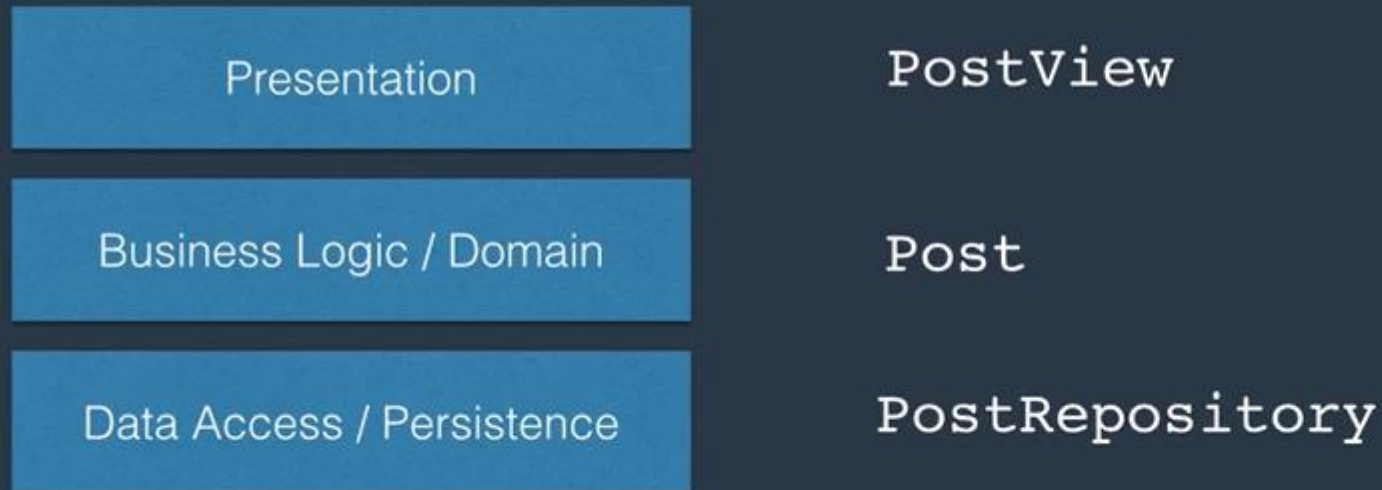
- What is a class
- Real-world example of classes
- What is an object
- Static members



# Application



# Real World Example



# Anatomy of a Class

- Data (represented by fields)
- Behaviour (represented by methods / functions)



Person
Name: string Age: byte Height: float Weight: byte
Walk() Talk() Eat() Sleep()



## Post

```
Title: string  
Description: string  
DateTime: DateTime
```

```
Publish()  
Like()  
Comment(message)
```





# Object

- An instance of a class.



## Class

<b>Person</b>
Name: string Age: byte Height: float Weight: byte
Walk() Talk() Eat() Sleep()

## Objects

<b>John</b>
<b>Mary</b>
<b>Scott</b>



# Declaring Classes

```
public class Person  
{  
}
```



# Naming Conventions

- Pascal Case
- camel Case



# Declaring Classes

```
public class Person
{
    public string Name;

    public void Introduce()
    {
        Console.WriteLine("Hi, my name is " + Name);
    }
}
```



# Creating Objects

```
Person person = new Person();
```

```
var person = new Person();
```



# Using Objects

```
var person = new Person();  
person.Name = "Mosh";  
person.Introduce();
```



# Class Members

- Instance: accessible from an object.

```
var person = new Person();  
person.Introduce();
```

- Static: accessible from the class.

```
Console.WriteLine();
```





# Why use static members?

- To represent concepts that are singleton.
- `DateTime.Now`
- `Console.WriteLine()`



# Declaring Static Members

```
public class Person
{
    public static int PeopleCount = 0;
}
```



# Demo Classes



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Program.cs

Classes.Program

Main(string[] args)

```
using System;
```

```
namespace Classes
```

```
{
```

```
    public class Person
```

```
    {
```

```
        public string Name;
```

```
        public void Introduce(string to)
```

```
        {
```

```
            Console.WriteLine("Hi {0}, I am {1}", to, Name);
```

```
        }
```

```
    }
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            var person = new Person();
```

```
            person.Name = "John";
```

```
            person.Introduce("Mosh");
```

```
        }
```

```
    }
```

```
}
```

Solution Explorer - Classes

Search Solution Explorer - Classes (Ctrl+Shift+F)

Solution 'Classes' (1 project)

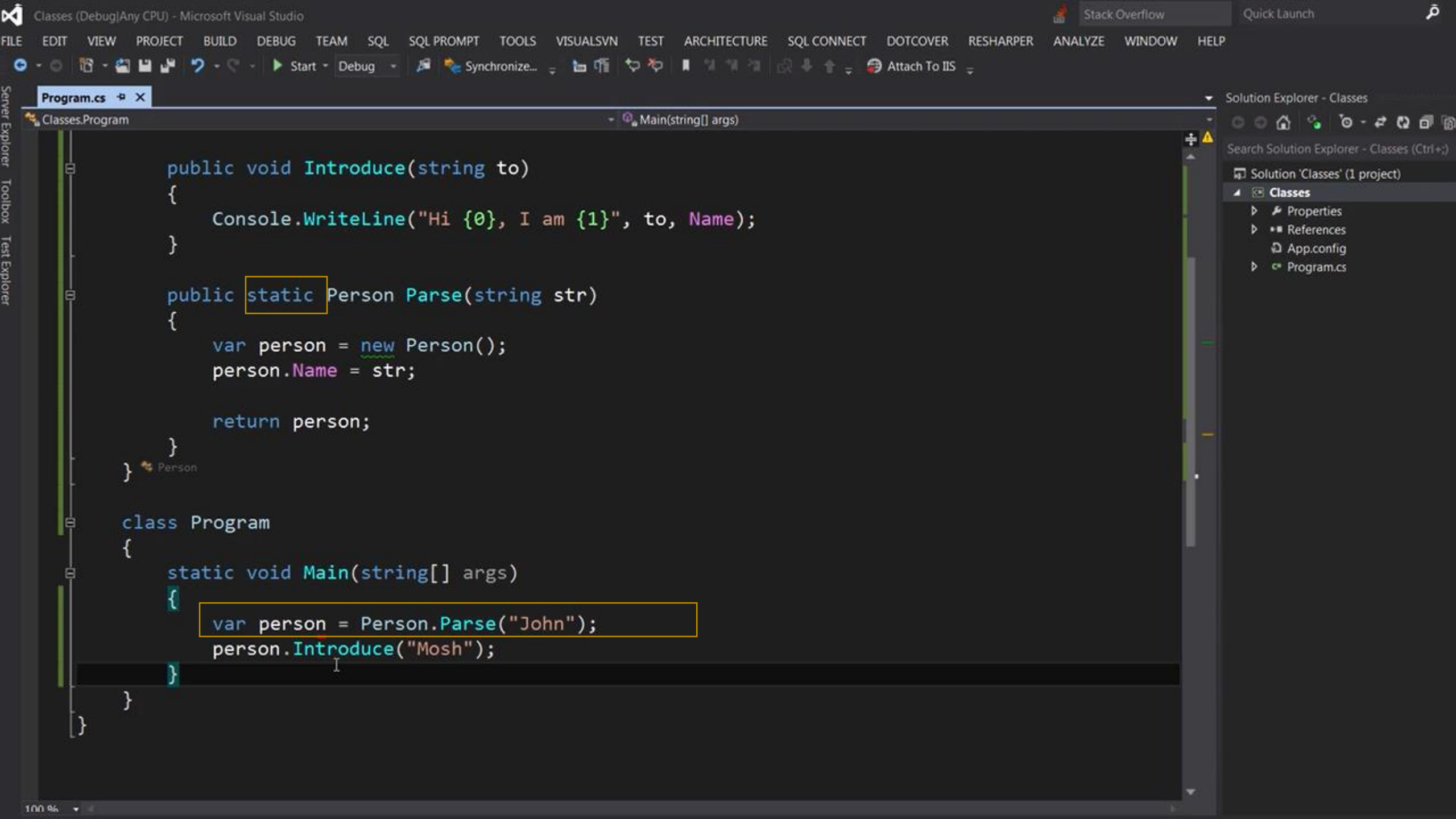
Classes

Properties

References

App.config

Program.cs



# Constructors



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# What?

A method that is called when an instance of a class is created.



# Why?

To put an object in an early state.





# How?

```
public class Customer
{
    public Customer()
    {
    }
}
```



# Constructors

```
public class Customer
{
    public string Name;

    public Customer(string name)
    {
        this.Name = name;
    }
}
```



# Constructors

```
public class Customer
{
    public string Name;

    public Customer(string name)
    {
        this.Name = name;
    }
}
```



# Constructors

```
var customer = new Customer("John");
```



# Constructor Overloading

```
public class Customer
{
    public Customer() { ... }

    public Customer(string name) { ... }

    public Customer(int id, string name) { ... }
}
```

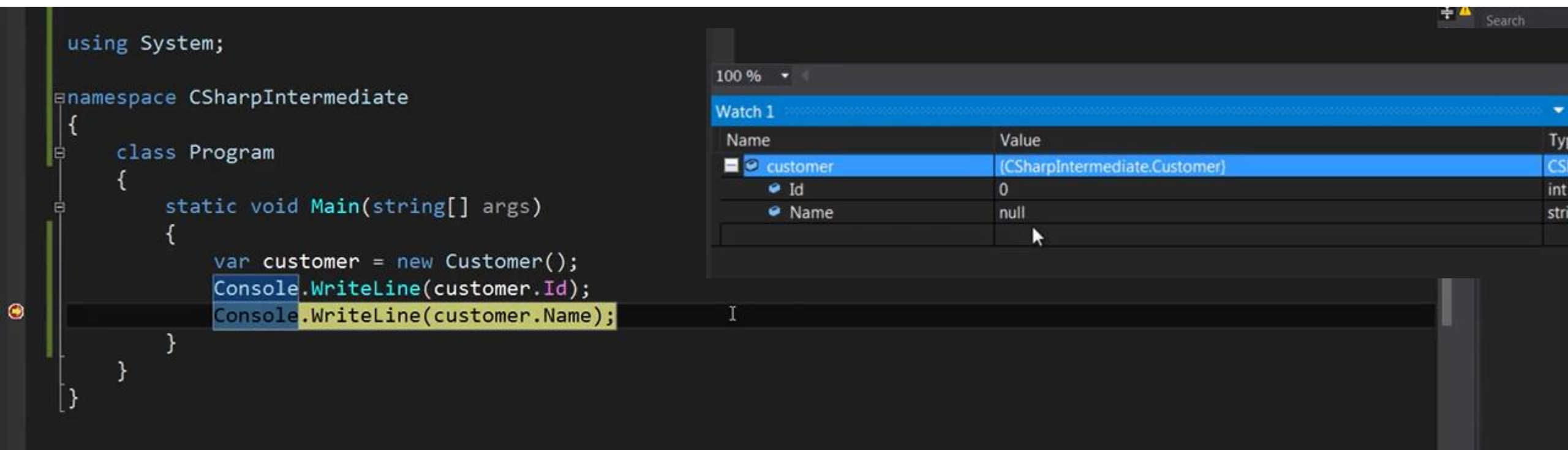
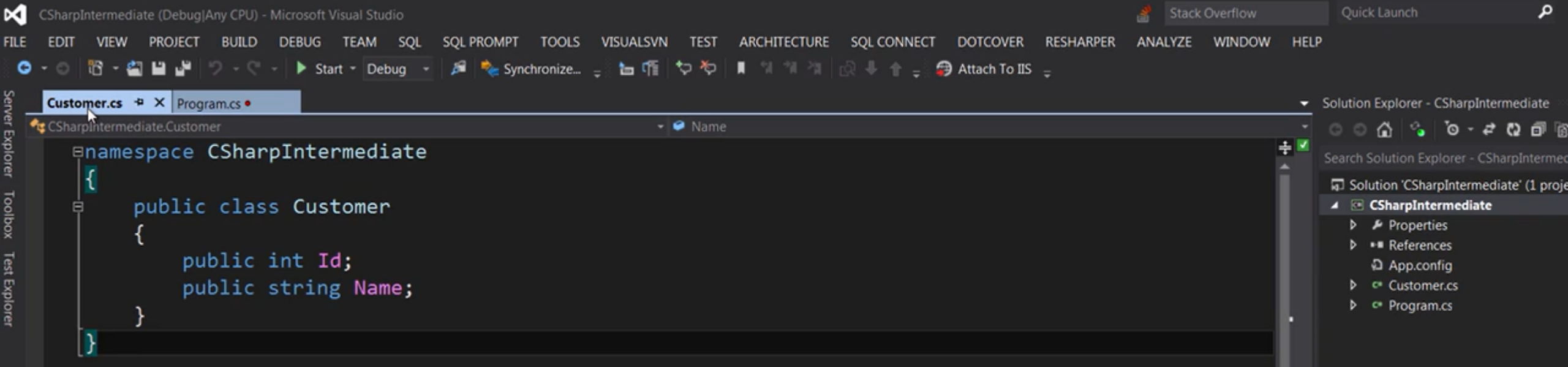


# Demo Constructors



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular



```
public class Customer
```

```
{
```

```
    public int Id;
```

```
    public string Name;
```

```
    public Customer(int id)
```

```
    {
```

```
        this.Id = id;
```

```
    }
```

```
    public Customer(int id, string name)
```

```
    {
```

```
        this.Id = id;
```

```
        this.Name = name;
```

```
    }
```

```
}
```

```
}
```

CSharpIntermediate

Properties

References

App.config

Customer.cs

Program.cs

```
static void Main(string[] args)
```

```
{
```

```
    var customer = new Customer();
```

```
    Console.WriteLine(customer.
```

```
    Console.WriteLine(customer.
```

```
}
```

Cannot resolve constructor 'Customer()', candidates are:

Customer(int) (in class Customer)

Customer(int, string) (in class Customer)



```
public class Customer
```

```
{  
    public int Id;  
    public string Name;
```

```
    public Customer(int id)  
    {  
        this.Id = id;  
    }
```

```
    public Customer(int id, string name)  
    {  
        this.Id = id;  
        this.Name = name;  
    }
```

```
}
```

CSharpIntermediate

Properties

References

App.config

Customer.cs

Program.cs

```
static void Main(string[] args)
```

```
{
```

```
    var customer = new Customer();
```

```
    Console.WriteLine(customer.
```

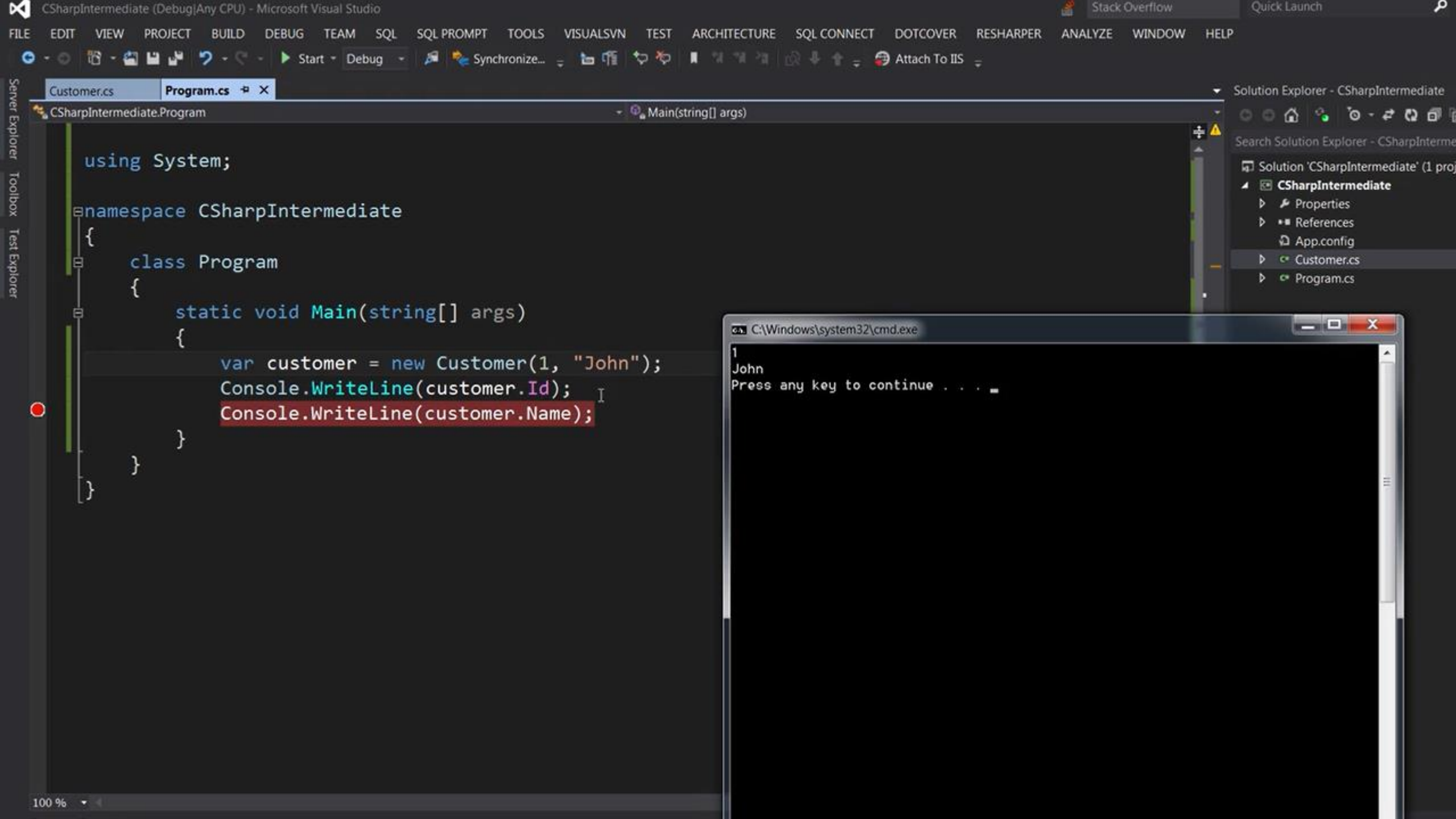
```
    Console.WriteLine(customer.
```

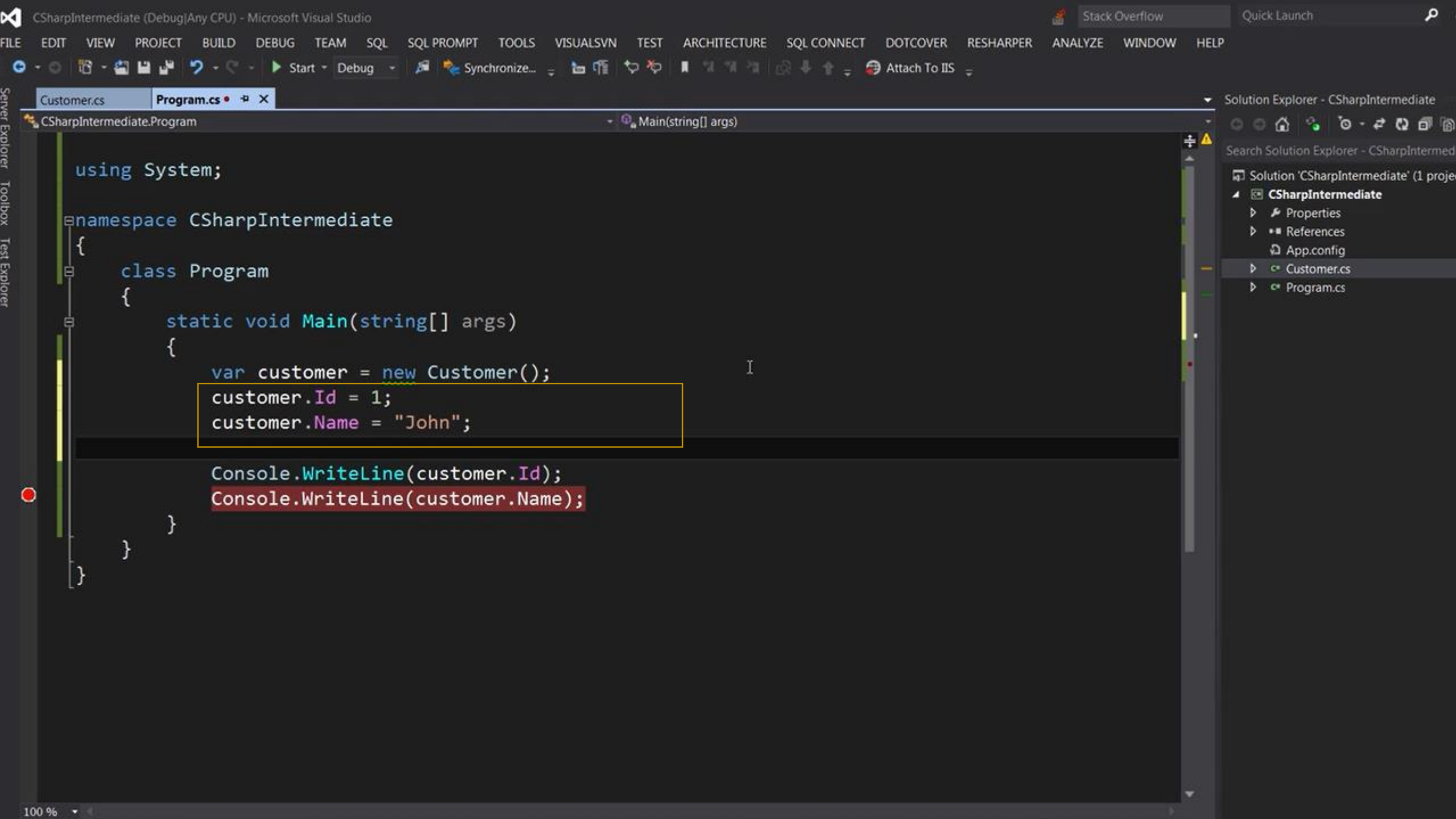
```
}
```

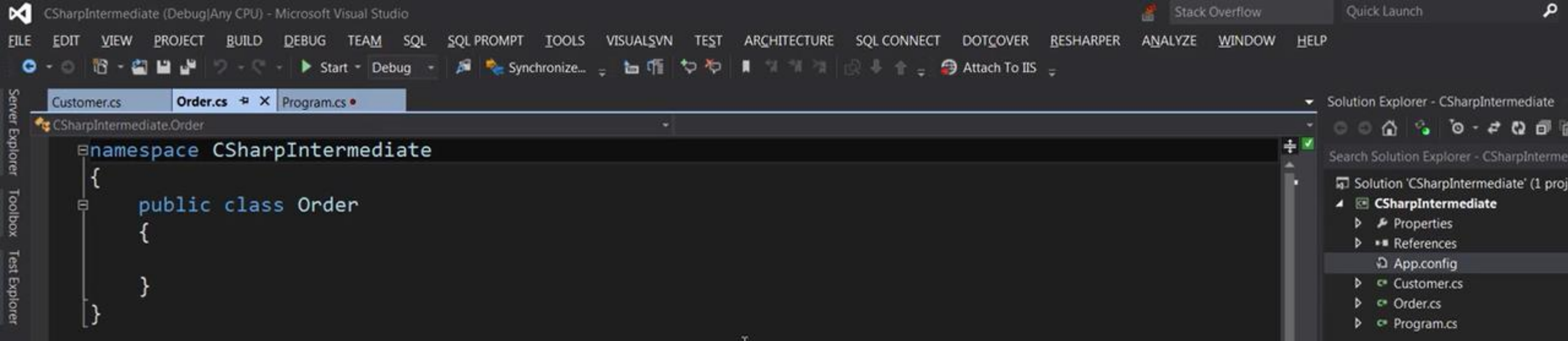
Cannot resolve constructor 'Customer()', candidates are:

Customer(int) (in class Customer)

Customer(int, string) (in class Customer)







```
namespace CSharpIntermediate
{
    class Program
    {
        static void Main(string[] args)
        {
            var customer = new Customer();
            customer.Id = 1;
            customer.Name = "John";

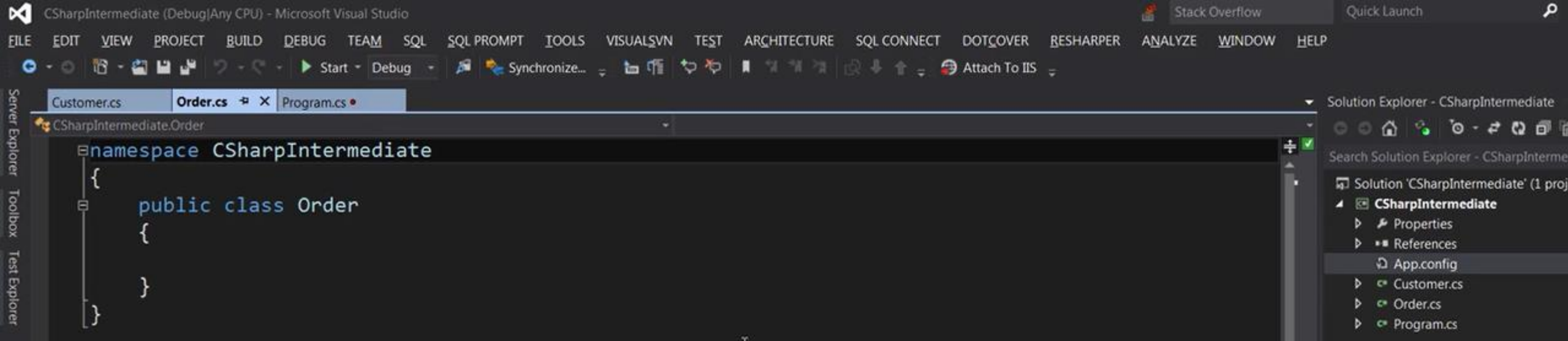
            var order = new Order();
            customer.Orders.Add(order);

            Console.WriteLine(customer.Id);
            Console.WriteLine(customer.Name);
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
Unhandled Exception: System.NullReferenceException: Object reference not set to an instance of an object.
   at CSharpIntermediate.Program.Main(String[] args) in c:\Projects\CSharpIntermediate\CSharpIntermediate\Program.cs:line 15
```





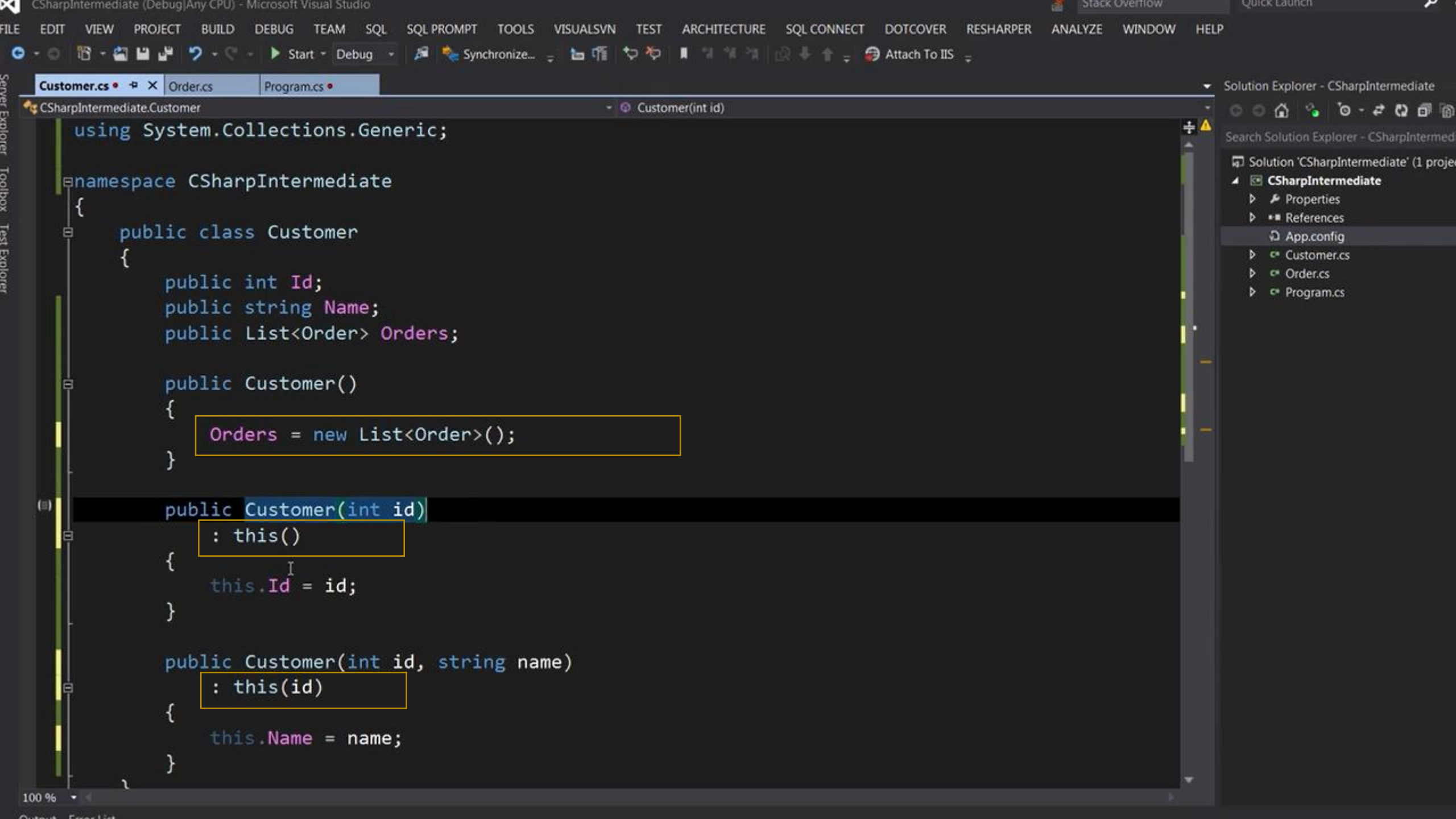
```
namespace CSharpIntermediate
{
    class Program
    {
        static void Main(string[] args)
        {
            var customer = new Customer();
            customer.Id = 1;
            customer.Name = "John";

            var order = new Order();
            customer.Orders.Add(order);

            Console.WriteLine(customer.Id);
            Console.WriteLine(customer.Name);
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
Unhandled Exception: System.NullReferenceException: Object reference not set to an instance of an object.
   at CSharpIntermediate.Program.Main(String[] args) in c:\Projects\CSharpIntermediate\CSharpIntermediate\Program.cs:line 15
```



# Object Initializers



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# What?

A syntax for quickly initialising an object without the need to call one of its constructors.





# Why?

To avoid creating multiple constructors.



# How?

```
public class Person
{
    public int Id;

    public string FirstName;

    public string LastName;

    public DateTime Birthdate;
}
```



# How?

```
public class Person
{
    public Person(int id) {}

    public Person(int id, string firstName) {}

    public Person(int id, string firstName, string lastName) {}

    public Person(int id, DateTime birthdate) {}
}
```



# How?

```
var person = new Person
{
    FirstName = "Mosh",
    LastName = "Hamedani"
};
```



# Methods



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# Agenda

- Signature of Methods
- Method Overloading
- Params modifier
- Ref modifier
- Out modifier



# Signature of a Method

- Name
- Number and Type of parameters

```
public class Point
{
    public void Move(int x, int y) {}
}
```



# Overloading Methods

- Having a method with the same name but different signatures

```
public class Point
{
    public void Move(int x, int y) {}

    public void Move(Point newLocation) {}

    public void Move(Point newLocation, int speed) {}
}
```





# A method with varying number of parameters

```
public class Calculator
{
    public int Add(int n1, int n2){}
    public int Add(int n1, int n2, int n3){}
    public int Add(int n1, int n2, int n3, int n4){}
    ...
}
```



# A method with varying number of parameters

```
public class Calculator
{
    public int Add(int[] numbers){}
}
```

```
var result = calculator.Add(new int[]{ 1, 2, 3, 4 });
```



# The Params Modifier

```
public class Calculator
{
    public int Add(params int[] numbers){}
}
```

```
var result = calculator.Add(new int[] { 1, 2, 3, 4 });
var result = calculator.Add(1, 2, 3, 4);
```



# The Ref Modifier

```
public class MyClass
{
    public void MyMethod(int a)
    {
        a += 2;
    }
}

var a = 1;
myClass.MyMethod(a);
```



# The Ref Modifier

```
public class Weirdo
{
    public void DoAWeirdThing(ref int a)
    {
        a += 2;
    }
}

var a = 1;
weirdo.DoAWeirdThing(ref a);
```



# The Out Modifier

```
public class MyClass
{
    public void MyMethod(out int result)
    {
        result = 1;
    }
}

int a;
myClass.MyMethod(out a);
```



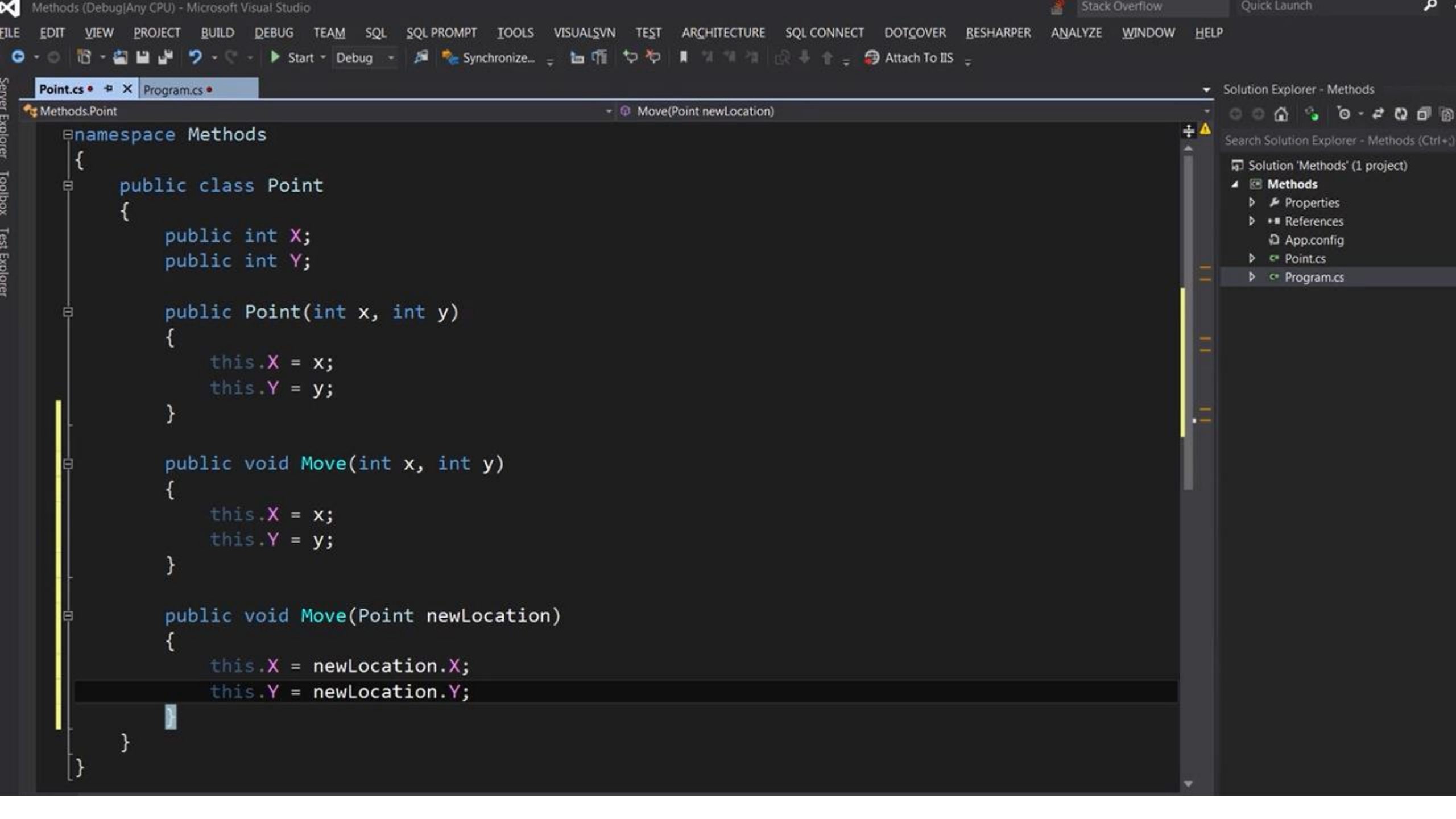
# Demo Methods



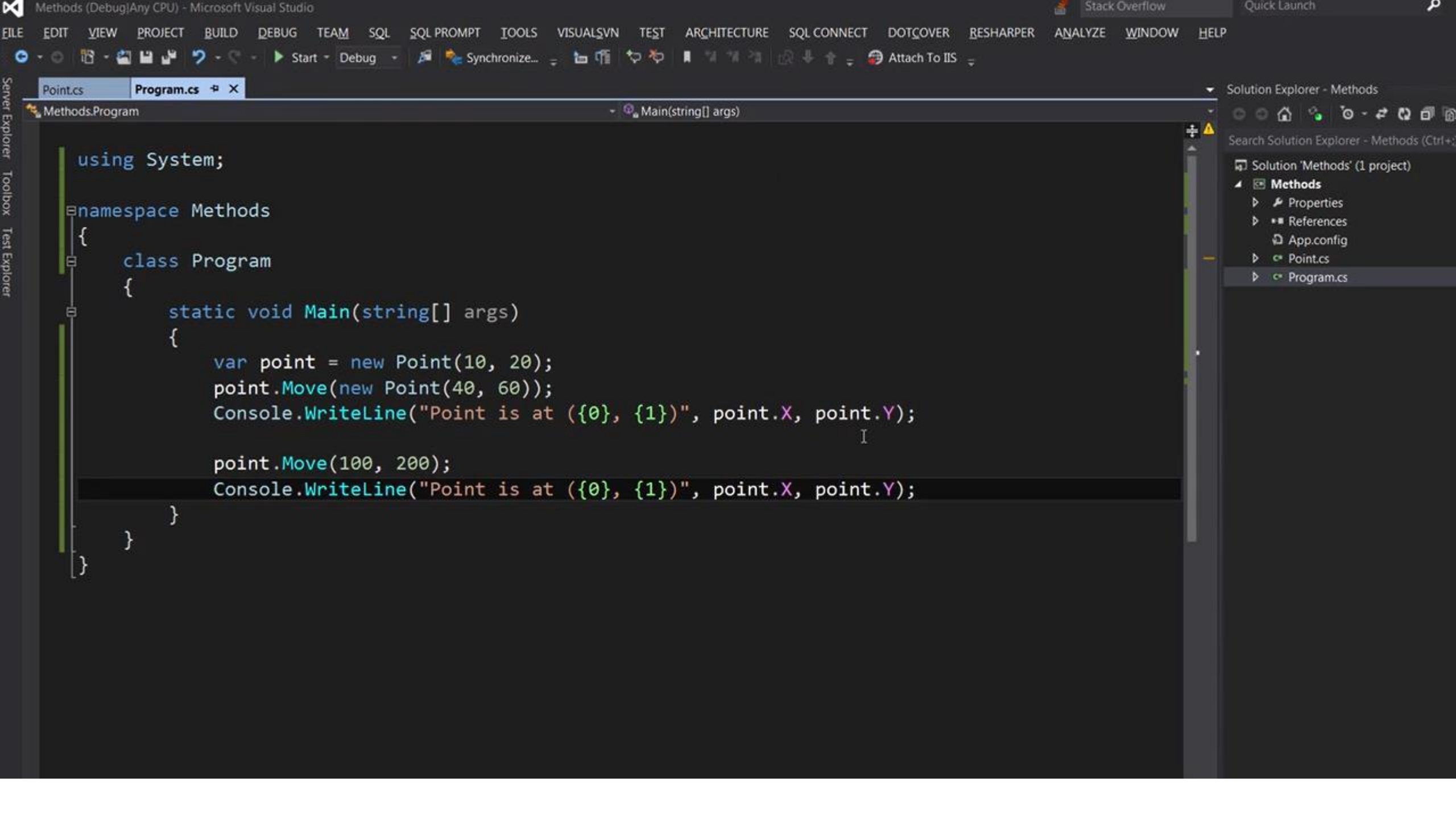
**Zouhair Rimale, Ph.D.**

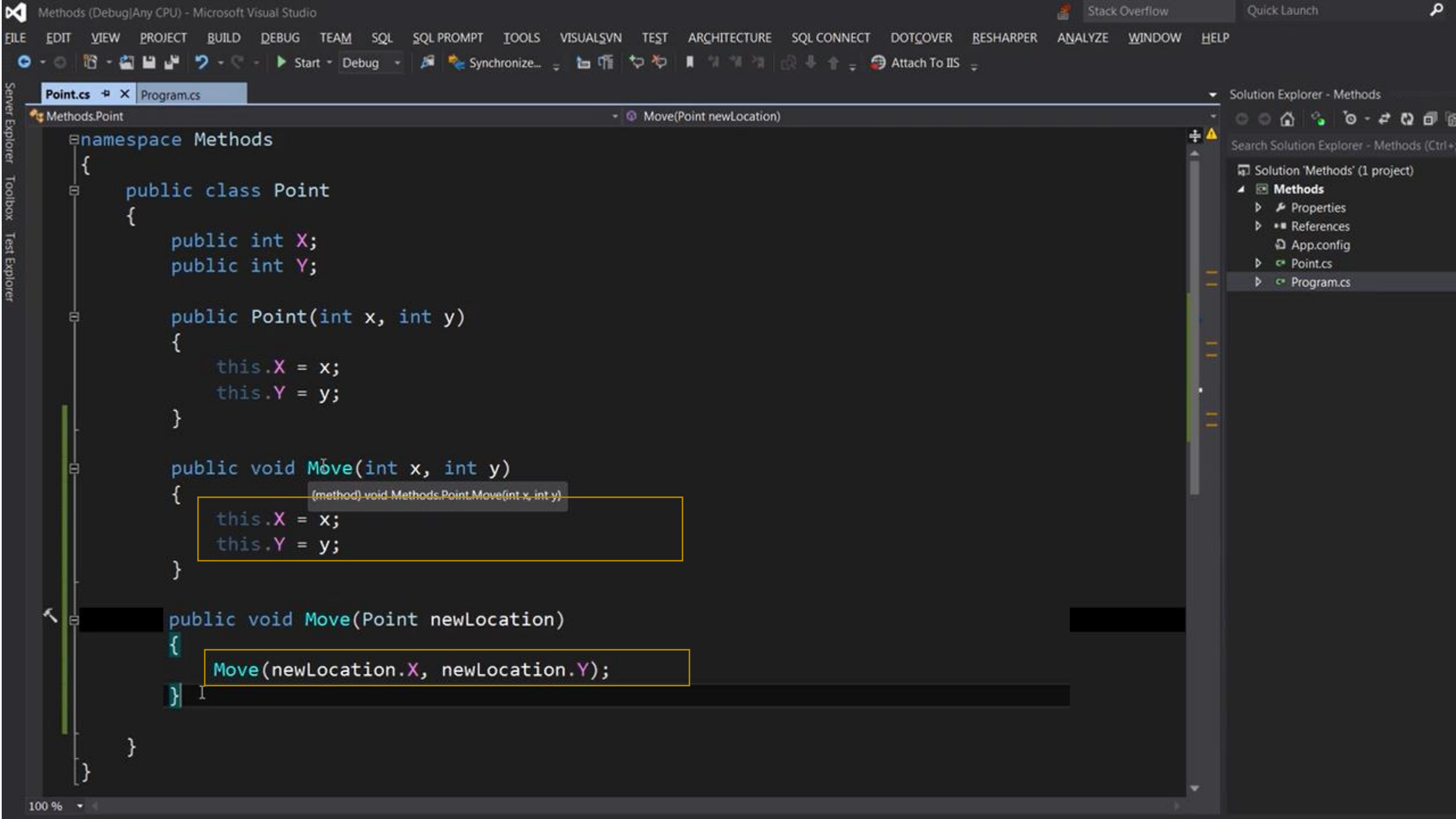
Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

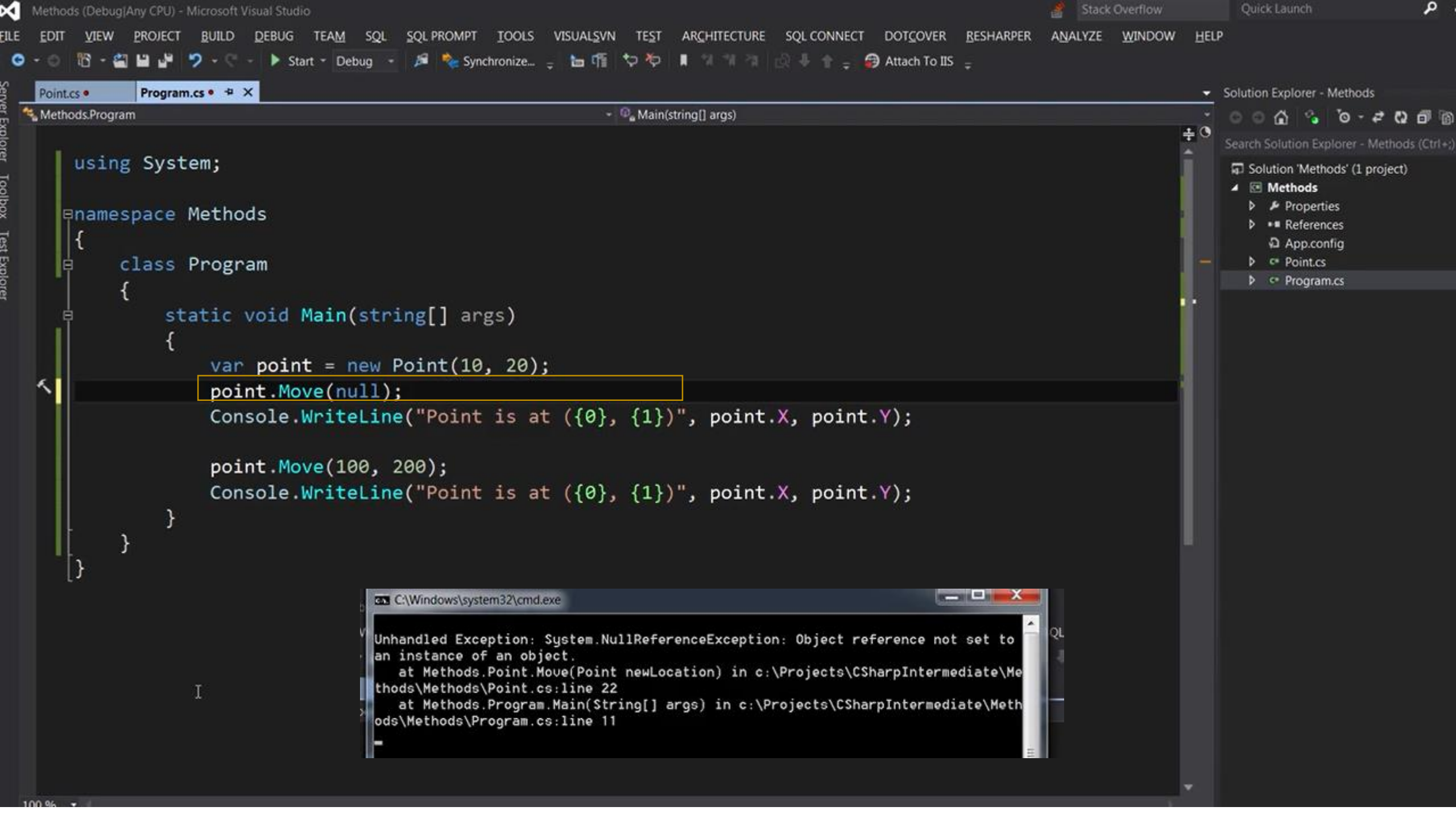


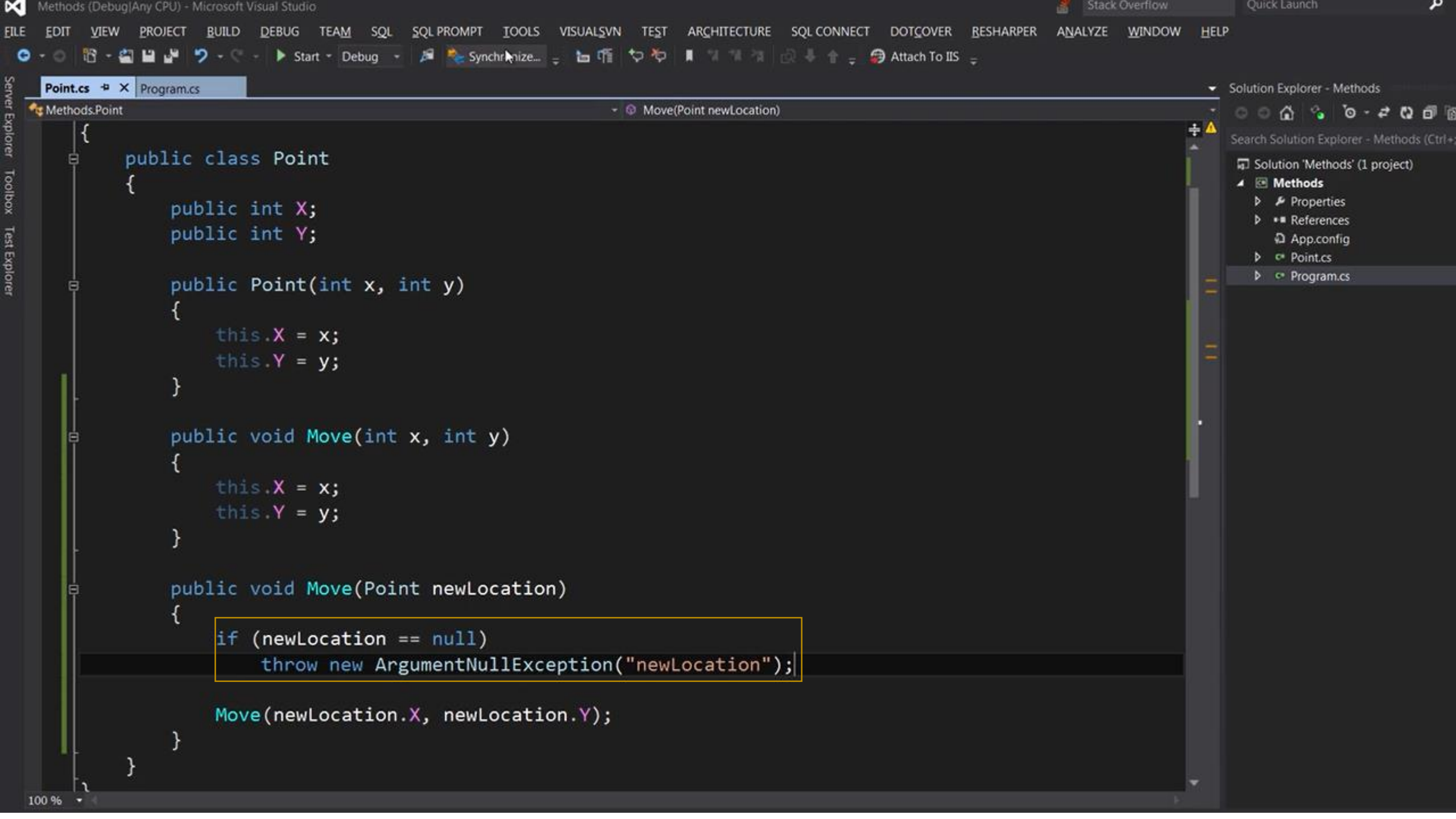




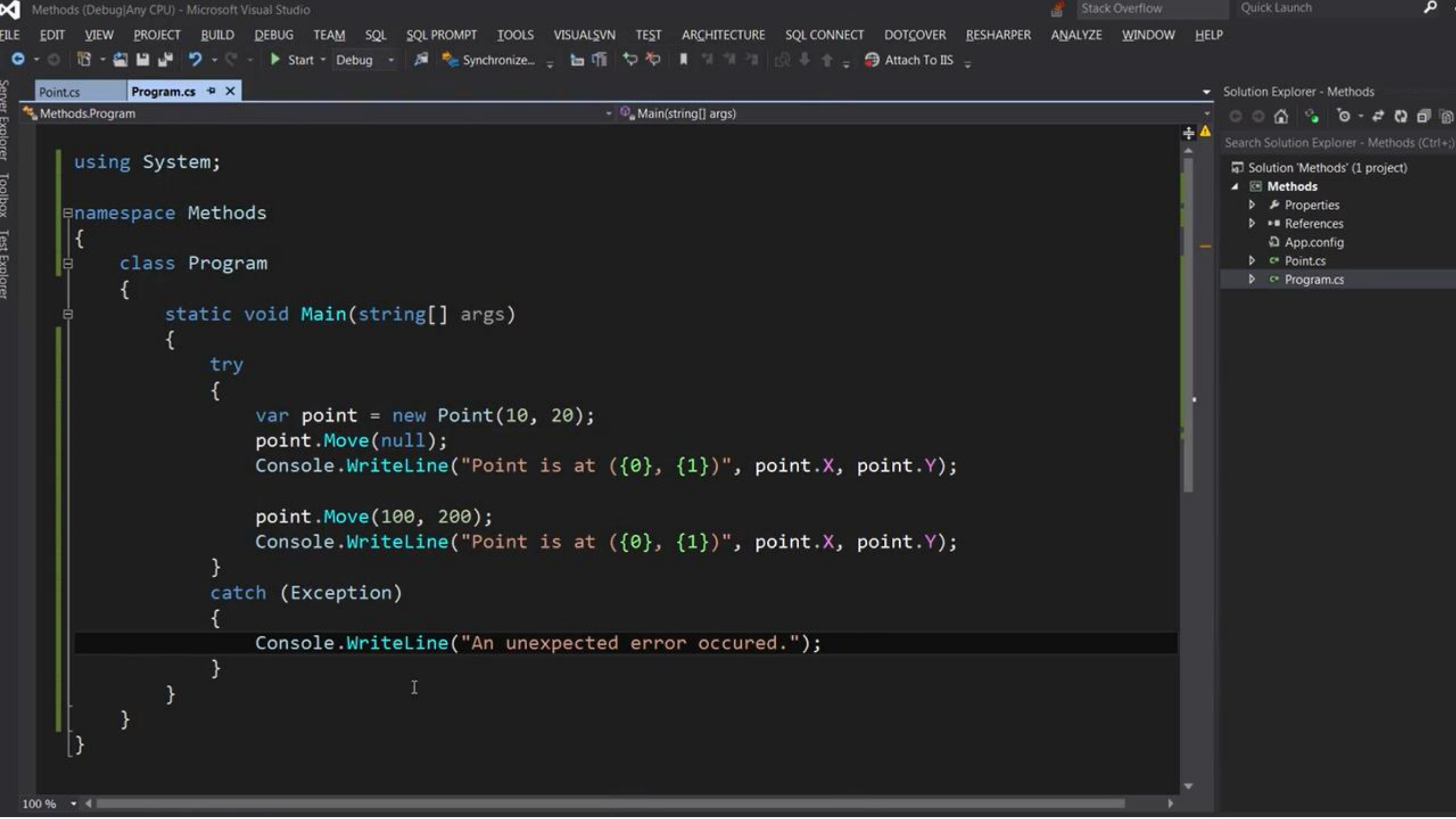












```

public class Calculator
{
    public int Add(params int[] numbers)
    {
        var sum = 0;
        foreach (var number in numbers)
        {
            sum += number;
        }

        return sum;
    }
}

```

- Methods
  - Properties
  - References
  - App.config
  - Calculator.cs
  - Point.cs
  - Program.cs

Methods (Debug)Any CPU - Microsoft Visual Studio

Stack Overflow

Quick Launch

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL SQL PROMPT TOOLS VISUALSVN TEST ARCHITECTURE SQL CONNECT DOTCOVER RESHARPER ANALYZE WINDOW HELP

Start Debug Synchronize Attach To IIS

Calculator.cs Point.cs Program.cs

Methods.Program

Main(string[] args)

```

using System;

namespace Methods
{
    class Program
    {
        static void Main(string[] args)
        {
            var calculator = new Calculator();
            Console.WriteLine(calculator.Add(1, 2));
            Console.WriteLine(calculator.Add(1, 2, 3));
            Console.WriteLine(calculator.Add(1, 2, 3, 4));
            Console.WriteLine(calculator.Add(new int[] { 1, 2, 3, 4, 5 }));
        }
    }
}

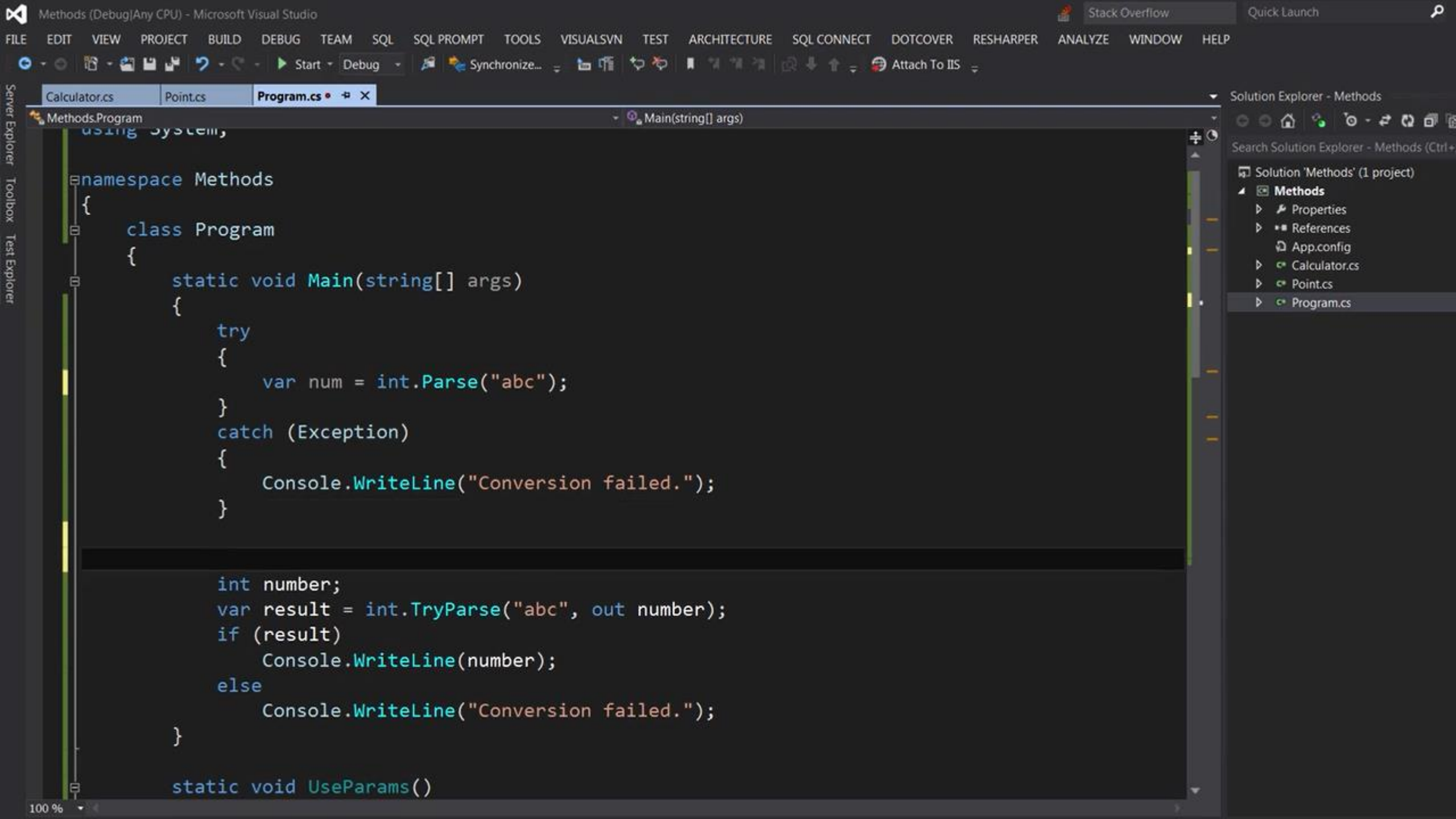
```

Solution Explorer - Methods

Search Solution Explorer - Methods (Ctrl+)

Solution 'Methods' (1 project)

- Methods
  - Properties
  - References
  - App.config
  - Calculator.cs
  - Point.cs
  - Program.cs



# Fields



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular



# Agenda

- Initialization
- Read-only fields



# Initialization

```
public class Customer
{
    List<Order> Orders;

    public Customer()
    {
        Orders = new List<Order>();
    }
}
```



# Initialization

```
public class Customer
{
    List<Order> Orders = new List<Order>();
}
```



# Read-only Fields

```
public class Customer
{
    readonly List<Order> Orders = new List<Order>();
}
```



# Demo Fields



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular



Customer.cs Program.cs

Fields.Customer

Id

```
using System.Collections.Generic;
```

```
namespace Fields
```

```
{
```

```
    public class Customer
```

```
    {
```

```
        public int Id;
```

```
        public string Name;
```

```
        public List<Order> Orders = new List<Order>();
```

```
        public Customer(int id)
```

```
        {
```

```
            this.Id = id;
```

```
        }
```

```
        public Customer(int id, string name)
```

```
            : this(id)
```

```
        {
```

```
            this.Name = name;
```

```
        }
```

```
        public void Promote()
```

```
        {
```

```
            Orders = new List<Order>();
```

```
            // ....
```

```
        }
```

Solution Explorer - Fields

Search Solution Explorer - Fields (Ctrl+Shift+F)

Solution 'Fields' (1 project)

Fields

Properties

References

App.config

Customer.cs

Program.cs

Server Explorer  
Toolbox  
Test Explorer

Fields.Order

```
namespace Fields
```

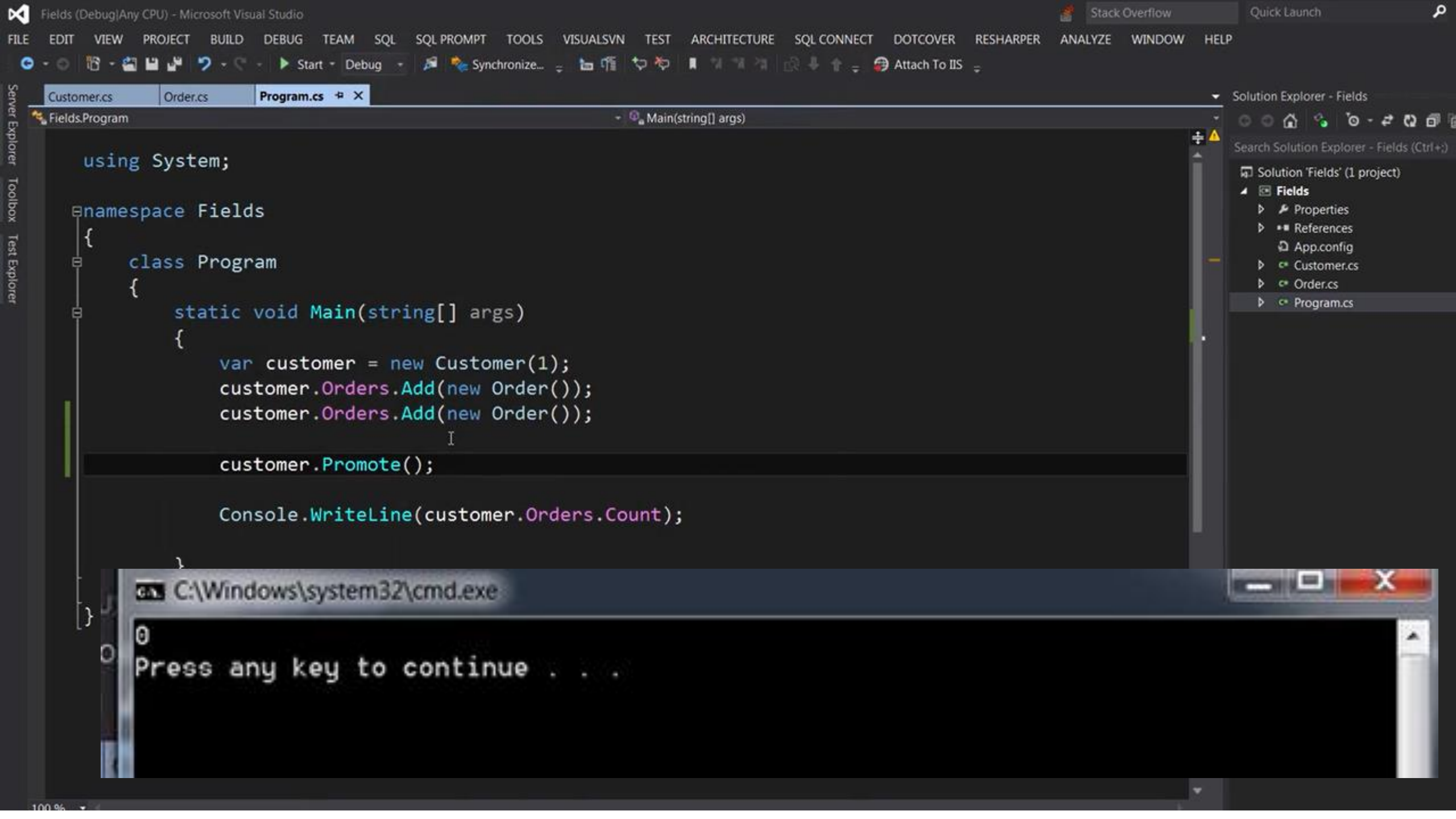
```
{
```

```
    public class Order
```

```
    {
```

```
    }
```

```
}
```



```
using System;
```

```
namespace Fields
```

```
{  
    class Program
```

```
{  
    static void Main(string[] args)
```

```
{  
        var customer = new Customer(1);  
        customer.Orders.Add(new Order());  
        customer.Orders.Add(new Order());
```

```
        customer.Promote();
```

```
        Console.WriteLine(customer.Orders.Count);
```

```
    }
```

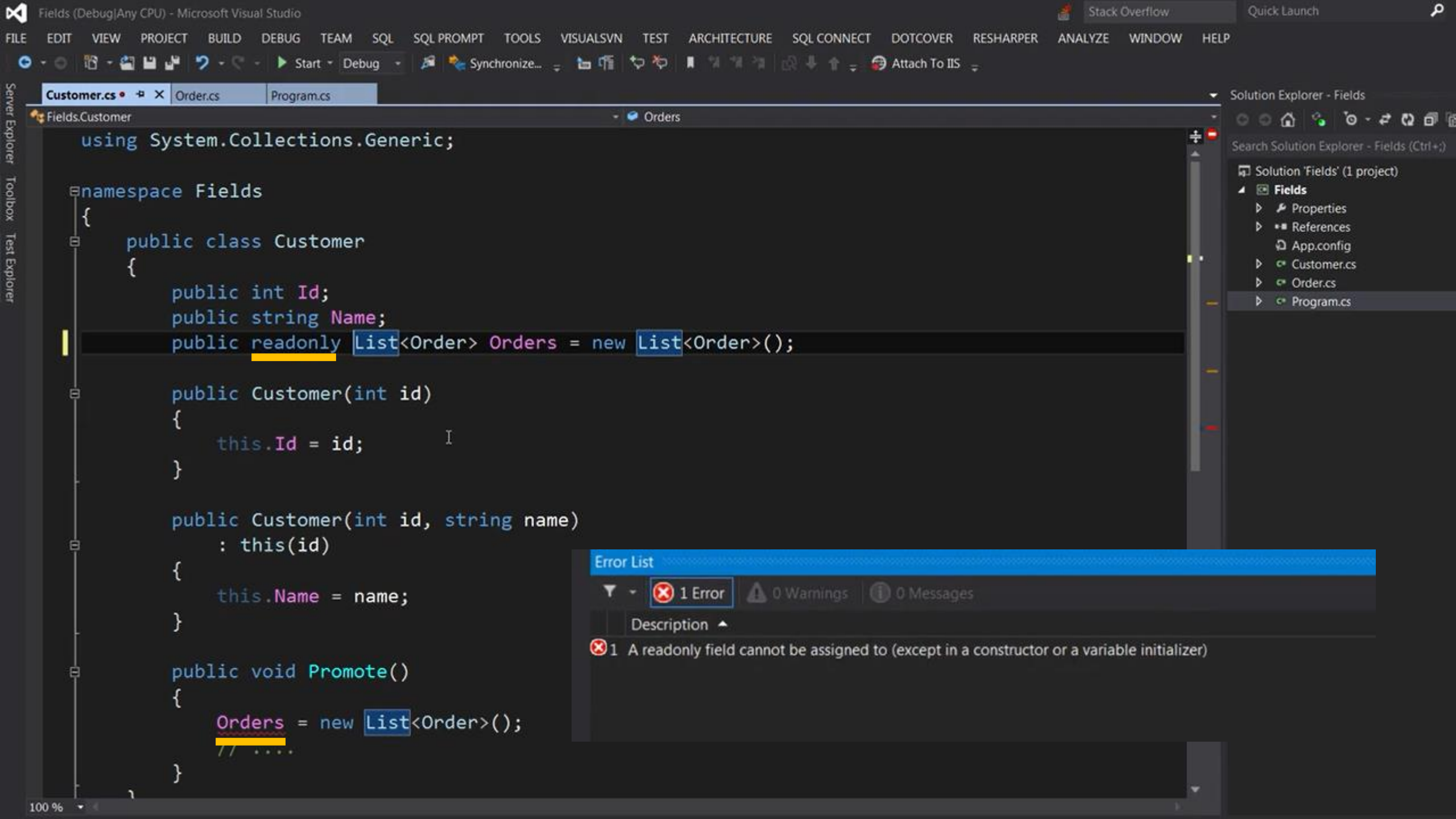
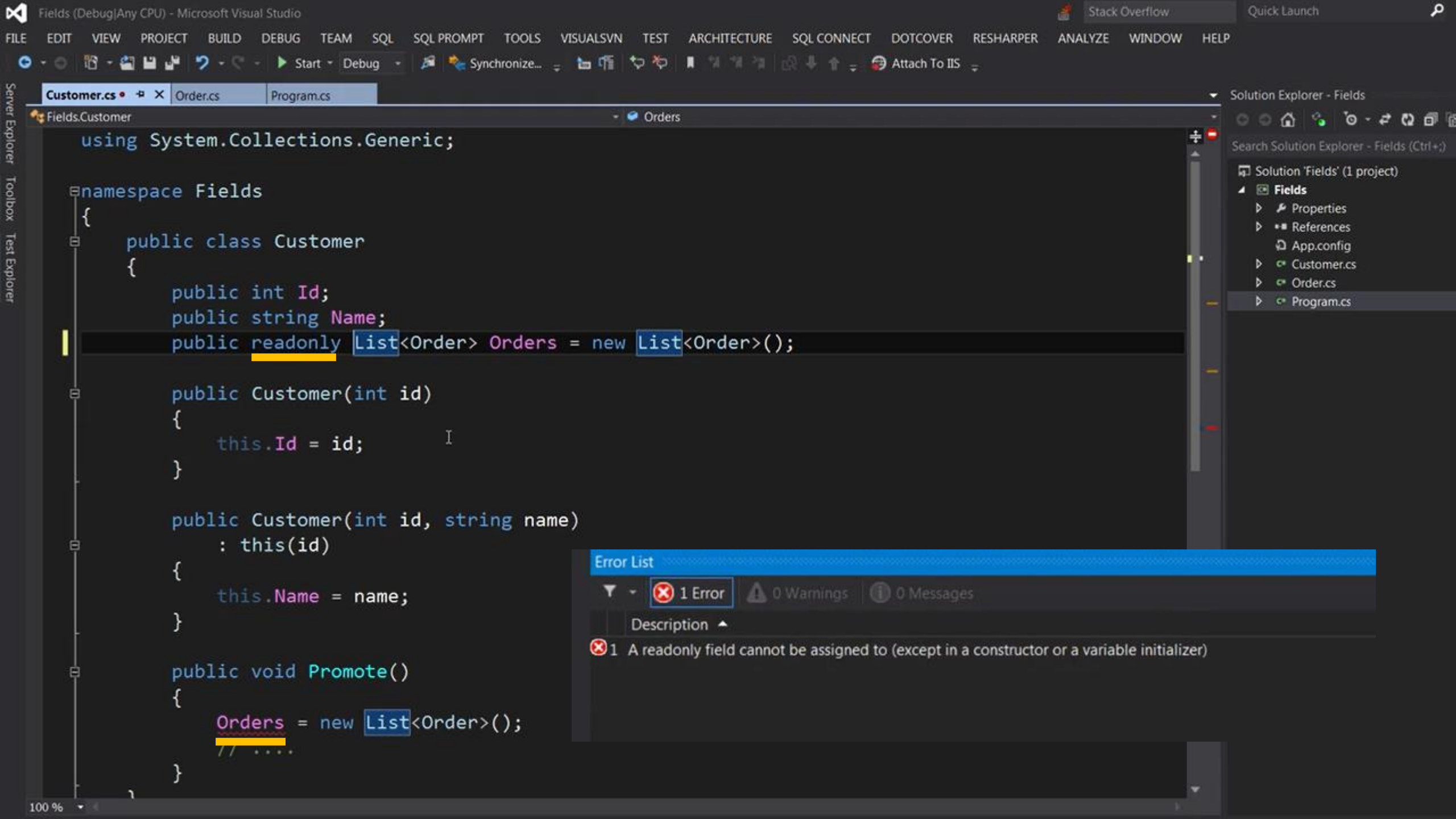
```
}
```

C:\Windows\system32\cmd.exe

0

Press any key to continue . . .







# Access Modifiers



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# Access Modifiers

- Public
- Private
- Protected
- Internal
- Protected Internal



# What?

A way to control access to a class and/or its members.



# Why?

To create safety in our programs.



# How?

```
public class Customer
{
    private string Name;
}

...
var john = new Customer();
john.Name; // won't compile
```



# Object-oriented programming

- Encapsulation / Information Hiding
- Inheritance
- Polymorphism





**C# Intermediate**



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# Encapsulation (in practice)

- Define fields as private
- Provide getter/setter methods as public





```
public class Person
{
    private string _name;

    public void SetName(string name)
    {
        if (!String.IsNullOrEmpty(name))
            this._name = name;
    }

    public string GetName()
    {
        return _name;
    }
}
```



# Demo Access Modifiers



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

Program.cs

AccessModifiers.Person

GetBirthdate()

```
using System;

namespace AccessModifiers
{
    public class Person
    {
        private DateTime _birthdate;

        public void SetBirthdate(DateTime birthdate)
        {
            _birthdate = birthdate;
        }

        public DateTime GetBirthdate()
        {
            return _birthdate;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var person = new Person();
            person.SetBirthdate(new DateTime(1982, 1, 1));
            Console.WriteLine(person.GetBirthdate());
        }
    }
}
```

Solution Explorer - AccessModifiers

Search Solution Explorer - AccessModifiers

Solution 'AccessModifiers' (1 project)

AccessModifiers

- Properties
- References
- App.config
- Program.cs

100 %

Output Error List

# Properties



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# What?

A class member that encapsulates a getter/setter for accessing a field.



# Why?

To create a getter/setter with less code.



```
public class Person
{
    private DateTime _birthdate;

    public void SetBirthdate(DateTime birthdate)
    {
        this._birthdate = birthdate;
    }

    public DateTime GetBirthdate()
    {
        return _birthdate;
    }
}
```





# How?

```
public class Person
{
    private DateTime _birthdate;

    public DateTime Birthdate
    {
        get { return _birthdate; }
        set { _birthdate = value; }
    }
}
```



# Auto-implemented Properties

```
public class Person
{
    public DateTime Birthdate { get; set; }
}
```

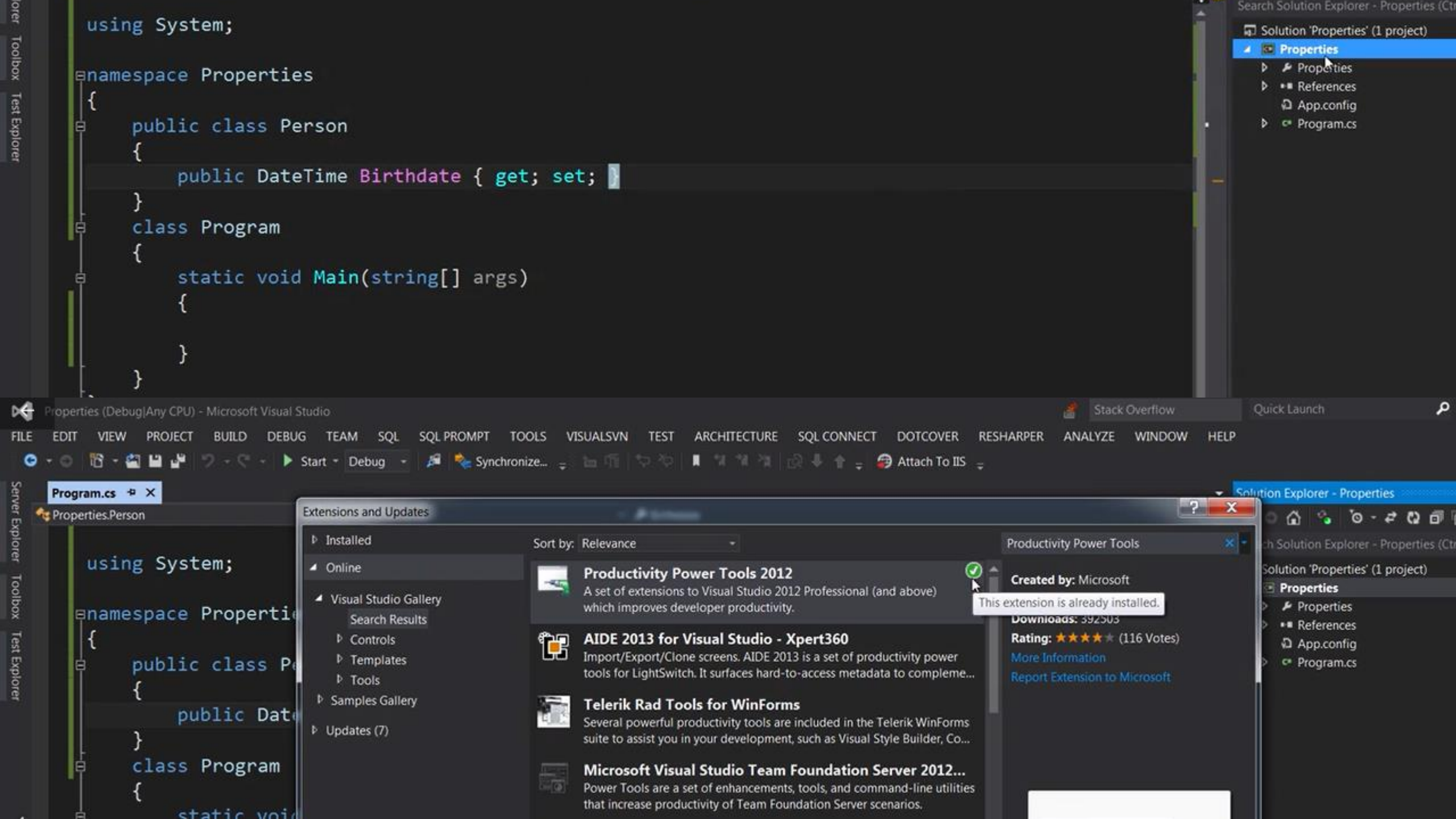


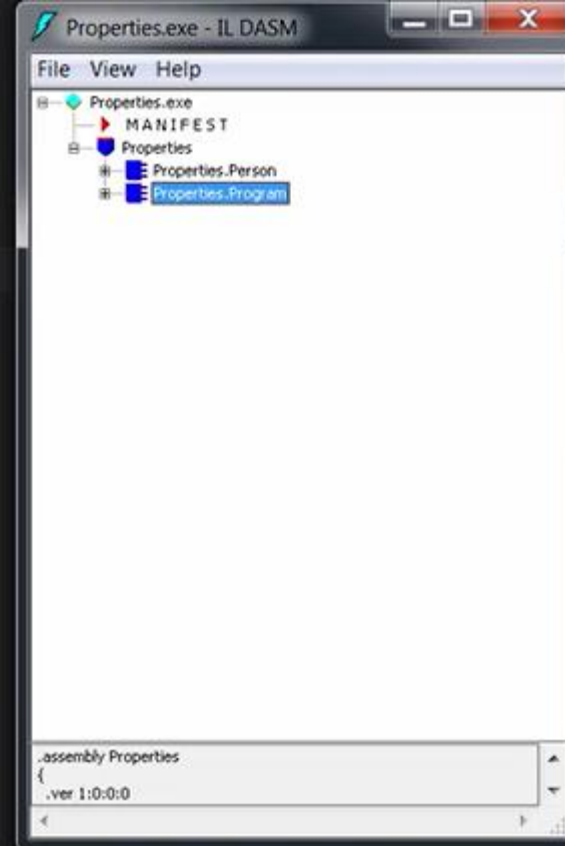
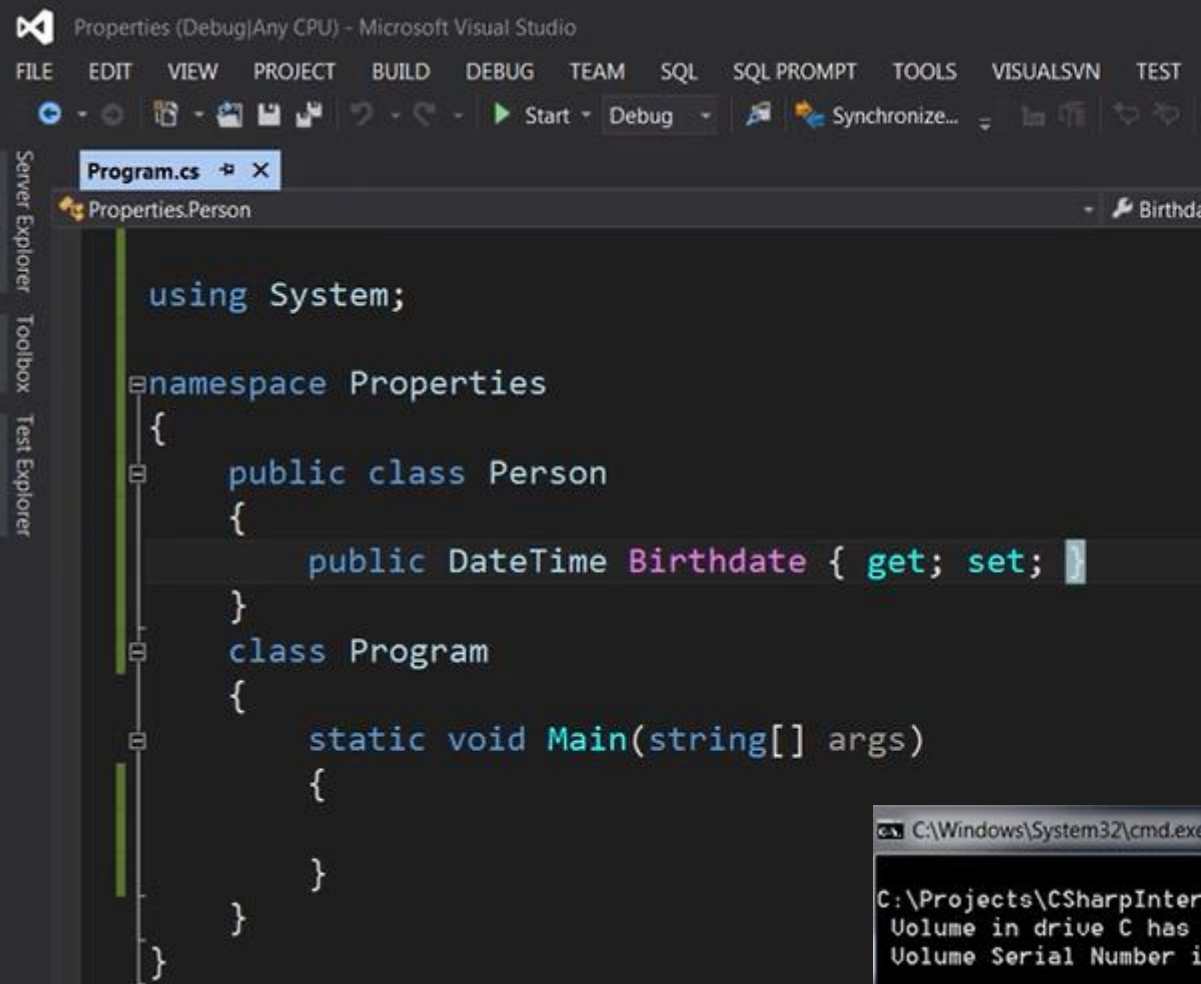
# Demo Properties



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular





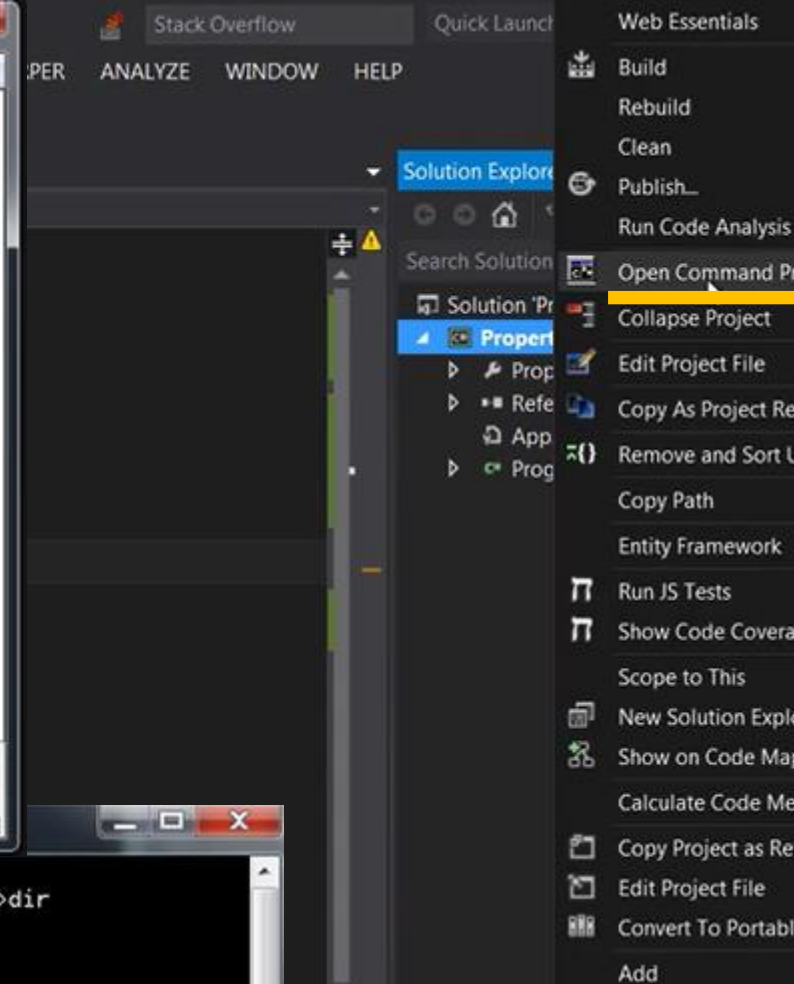
```
C:\Windows\System32\cmd.exe

C:\Projects\CSharpIntermediate\Properties\Properties\bin\Debug>dir
Volume in drive C has no label.
Volume Serial Number is 663B-866B

Directory of C:\Projects\CSharpIntermediate\Properties\Properties\bin\Debug

09/01/2015  09:05 PM    <DIR>          .
09/01/2015  09:05 PM    <DIR>          ..
09/01/2015  09:10 PM             5,120 Properties.exe
09/01/2015  09:05 PM             187 Properties.exe.config
09/01/2015  09:10 PM            13,824 Properties.pdb
09/01/2015  09:05 PM            22,984 Properties.ushost.exe
09/01/2015  09:05 PM             187 Properties.ushost.exe.config
06/06/2012  03:06 AM             490 Properties.ushost.exe.manifest
               6 File(s)          42,792 bytes
               2 Dir(s)  14,321,152,000 bytes free

C:\Projects\CSharpIntermediate\Properties\Properties\bin\Debug>
```





```
public class Person
```

```
{
```

```
    public DateTime Birthdate { get; private set; }
```

```
    public int Age
```

```
{
```

```
        get
```

```
{
```

```
            var timeSpan = DateTime.Today - Birthdate;
```

```
            var years = timeSpan.Days/365;
```

```
            return years;
```

```
        }
```

```
    }
```

```
}
```

```
}
```

- References
- App.config
- Person.cs
- Program.cs

Properties.Program

Main(string[] args)

```
using System;
```

```
namespace Properties
```

```
{
```

```
    class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
{
```

```
        var person = new Person();
```

```
        person.Birthdate = new DateTime(1982, 1, 1);
```

```
        Console.WriteLine(person.Age);
```

```
    }
```

```
}
```

```
}
```

Search Solution Explorer - Properties (Ctrl+Shift+F)

Solution 'Properties' (1 project)

Properties

- Properties
- References
- App.config
- Person.cs
- Program.cs

```
{  
    public Person(DateTime birthdate)  
    {  
        Birthdate = birthdate;  
    }  
}
```

```
    public DateTime Birthdate { get; private set; }  
  
    public int Age  
    {  
        get  
        {  
            var timeSpan = DateTime.Today - Birthdate;  
            var years = timeSpan.Days/365;  
  
            return years;  
        }  
    }  
}
```

```
namespace Properties  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            var person = new Person(new DateTime(1982, 1, 1));  
            Console.WriteLine(person.Age);  
        }  
    }  
}
```

Person.cs  
Program.cs

Properties  
Properties  
References  
App.config  
Person.cs  
Program.cs



# Indexers



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# What?

A way to access elements in a class that represents a list of values.



# Examples

```
var array = new int[5];  
array[0] = 1;
```

```
var list = new List<int>();  
list[0] = 1;
```



# Example

```
var cookie = new HttpCookie();  
cookie.Expire = DateTime.Today.AddDays(5);
```

```
cookie["name"] = "Mosh";  
cookie.SetItem("name", "Mosh");
```

```
var name = cookie["name"];  
var name = cookie.GetItem("name");
```



# How?

```
public class HttpCookie
{
    public string this[string key]
    {
        get { ... }
        set { ... }
    }
}
```



# Demo Indexers



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

```
public class HttpCookie
{
    private readonly Dictionary<string, string> _dictionary;
    public DateTime Expiry { get; set; }

    public HttpCookie()
    {
        _dictionary = new Dictionary<string, string>();
    }

    public string this[string key]
    {
        get { return _dictionary[key]; }
        set { _dictionary[key] = value; }
    }
}
```

```
namespace Indexers
{
    class Program
    {
        static void Main(string[] args)
        {
            var cookie = new HttpCookie();
            cookie["name"] = "Mosh";
            Console.WriteLine(cookie["name"]);
        }
    }
}
```



HttpCookie.cs Program.cs

Indexers.HttpCookie

this[string key]

```
using System;
using System.Collections.Generic;

namespace Indexers
{
    public class HttpCookie
    {
        private readonly Dictionary<string, string> _dictionary;
        public DateTime Expiry { get; set; }

        public HttpCookie()
        {
            _dictionary = new Dictionary<string, string>();
        }

        public void SetItem(string key, string value)
        {
        }

        public string GetItem(string key)
        {
        }
    }
}
```

Solution Explorer - Indexers

Search Solution Explorer - Indexers (Ctrl)

Solution 'Indexers' (1 project)

Indexers

- Properties
- References
- App.config
- HttpCookie.cs
- Program.cs

# Classes - Exercises



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

# Exercise 1 : Design a Stopwatch

Design a class called Stopwatch. The job of this class is to simulate a stopwatch. It should provide two methods: Start and Stop. We call the start method first, and the stop method next.

Then we ask the stopwatch about the duration between start and stop. Duration should be a value in TimeSpan. Display the duration on the console.

We should also be able to use a stopwatch multiple times. So we may start and stop it and then start and stop it again. Make sure the duration value each time is calculated properly.

We should not be able to start a stopwatch twice in a row (because that may overwrite the initial start time). So the class should throw an InvalidOperationException if its started twice.

**Educational tip:** The aim of this exercise is to make you understand that a class should be always in a valid state. We use encapsulation and information hiding to achieve that. The class should not reveal its implementation detail. It only reveals a little bit, like a blackbox. From the outside, you should not be able to misuse a class because you shouldn't be able to see the implementation detail.



## Exercise 2 : Design a StackOverflow Post

Design a class called Post. This class models a StackOverflow post. It should have properties for title, description and the date/time it was created. We should be able to up-vote or down-vote a post. We should also be able to see the current vote value. In the main method, create a post, up-vote and down-vote it a few times and then display the the current vote value.

In this exercise, you will learn that a StackOverflow post should provide methods for up-voting and down-voting. You should not give the ability to set the Vote property from the outside, because otherwise, you may accidentally change the votes of a class to 0 or to a random number. And this is how we create bugs in our programs. The class should always protect its state and hide its implementation detail.

**Educational tip:** The aim of this exercise is to help you understand that classes should encapsulate data AND behaviour around that data. Many developers (even those with years of experience) tend to create classes that are purely data containers, and other classes that are purely behaviour (methods) providers. This is not object-oriented programming. This is procedural programming. Such programs are very fragile. Making a change breaks many parts of the code.



# Solutions



**Zouhair Rimale, Ph.D.**

Expert technique .NET | SharePoint | ASP.NET MVC | WEB API | Angular

```
public class Stopwatch
{
    public DateTime StartTime { get; set; }
    public DateTime EndTime { get; set; }
```

```
    private bool running = false;
```

```
    public void Start()
```

```
    {
        if (_running)
            throw new InvalidOperationException("Stopwatch is already running!");

        _startTime = DateTime.Now;
        _running = true;
    }
```

```
    public void Stop()
```

```
    {
        if (!_running)
            throw new InvalidOperationException("Stopwatch is not running!");
```

```
        if (_running)
```

```
        {
            _endTime = DateTime.Now;
            _running = false;
        }
```

```
    public TimeSpan GetInterval()
```

```
    {
        return _endTime - _startTime;
    }
```

```
}
```

```
static void Main(string[] args)
{
    var stopwatch = new Stopwatch();

    for (var i = 0; i < 2; i++)
    {
        stopwatch.Start();

        Thread.Sleep(1000);

        //stopwatch.Start(DateTime.Now);
        stopwatch.Stop();

        Console.WriteLine("Duration: " + stopwatch.GetInterval().ToString());

        Console.WriteLine("Press Enter to run the stopwatch one more time.");
        Console.ReadLine();
    }
}
```

