C# vs .NET
C# is a programming language, while .NET is a framework. It consists of a run-time environment (CLR) and a class library that we use for building applications.

CLR
When you compile an application, C# compiler compiles your code to IL (Intermediate Language) code. IL code is platform agnostics, which makes it possible to a take a C# program on a different computer with different hardware architecture and operating system and run it. For this to happen, we need CLR. When you run a C# application, CLR compiles the IL code into the native machine code for the computer on which it is running. This process is called Just-in-time Compilation (JIT).

Architecture of .NET Applications
In terms of architecture, an application written with C# consists of building blocks called classes. A class is a container for data (attributes) and methods (functions). Attributes represent the state of the application. Methods include code. They have logic. That's where we implement our algorithms and write code.

A namespace is a container for related classes. So as your application grows in size, you may want to group the related classes into various namespaces for better maintainability.

As the number of classes and namespaces even grow further, you may want to physically separate related namespaces into separate assemblies. An assembly is a file (DLL or EXE) that contains one or more namespaces and classes. An EXE file represents a program that can be executed. A DLL is a file that includes code that can be re-used across different programs.

In the next section, you'll learn about basics of the C# language, including variables, constants, type conversion and operators.

Explicit Type Conversion

```
int i = 1;

byte b = i;          // won't compile
```

Explicit Type Conversion

```
int i = 1;

byte b = (byte)i;
```

Postfix Increment

```
int a = 1;
int b = a++;
```

a = 2, b = 1

Prefix Increment

```
int a = 1;
int b = ++a;
```

```
a = 2, b = 2
```

When to Use

To explain whys, hows, constrains, etc.
not the whats.

String Elements

```
string name = "Mosh";

char firstChar = name[0];

name[0] = 'm';
```

Types

- int
- char
- float
- bool
- classes
- structures
- arrays (System.Array)
- strings (System.String)

Types

Structures

- Primitive types
- Custom structures

Classes

- Arrays
- Strings
- Custom classes

Value Types

Structures

- Allocated on stack
- Memory allocation done automatically
- Immediately removed when out of scope

Reference Types

Classes

- You need to allocate memory
- Memory allocated on heap
- Garbage collected by CLR

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) {}
}
```

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 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);
```

Read-only Fields

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

How?

```
public class Person
{
    private DateTime _birthdate;

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

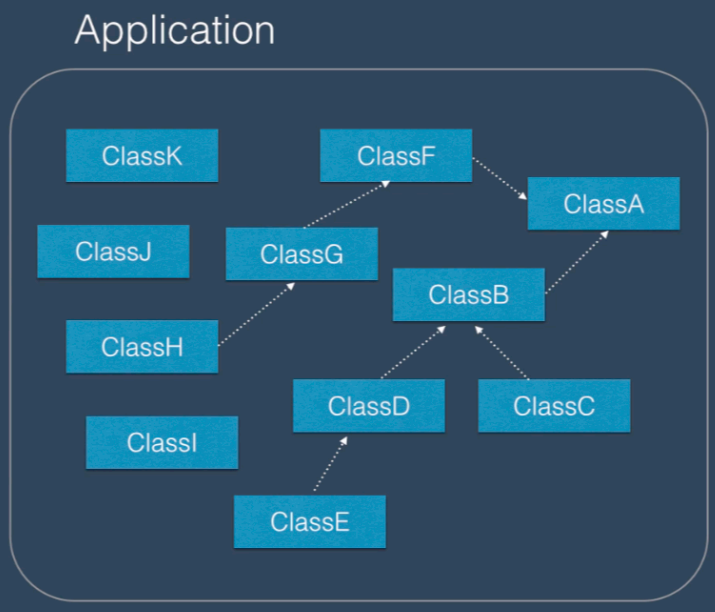
Auto-implemented Properties

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

How?

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

Loosely Coupled



But how?

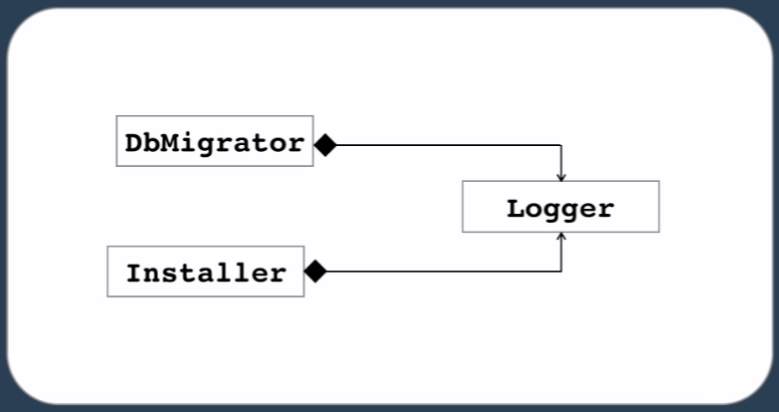
You need to understand

- Encapsulation
- The relationships between classes
- Interfaces

What is Composition?

- A kind of relationship between two classes that allows one to contain the other.
- Has-a relationship
- Example: Car has an Engine

In UML



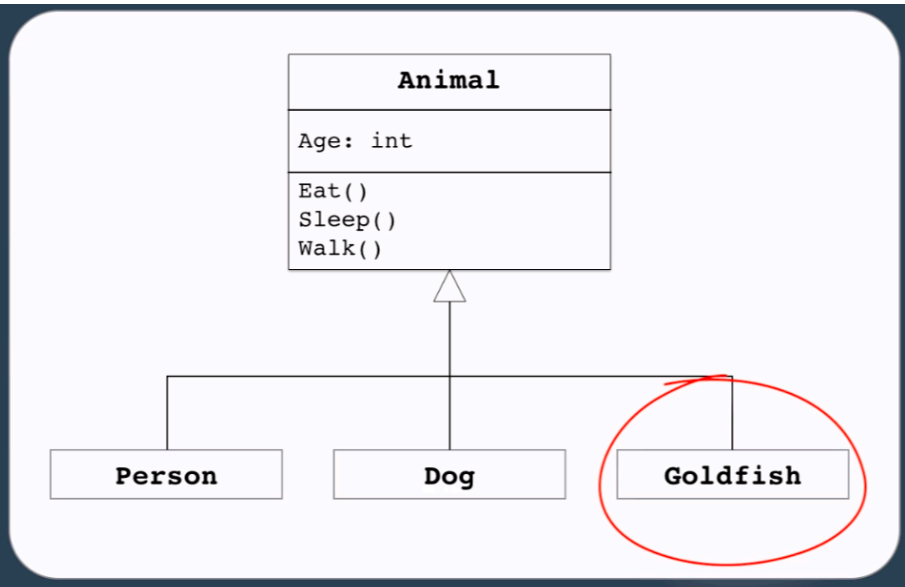
Problems with Inheritance

- Easily abused by amateur designers / developers
- Large hierarchies
- Fragility
- Tightly coupling

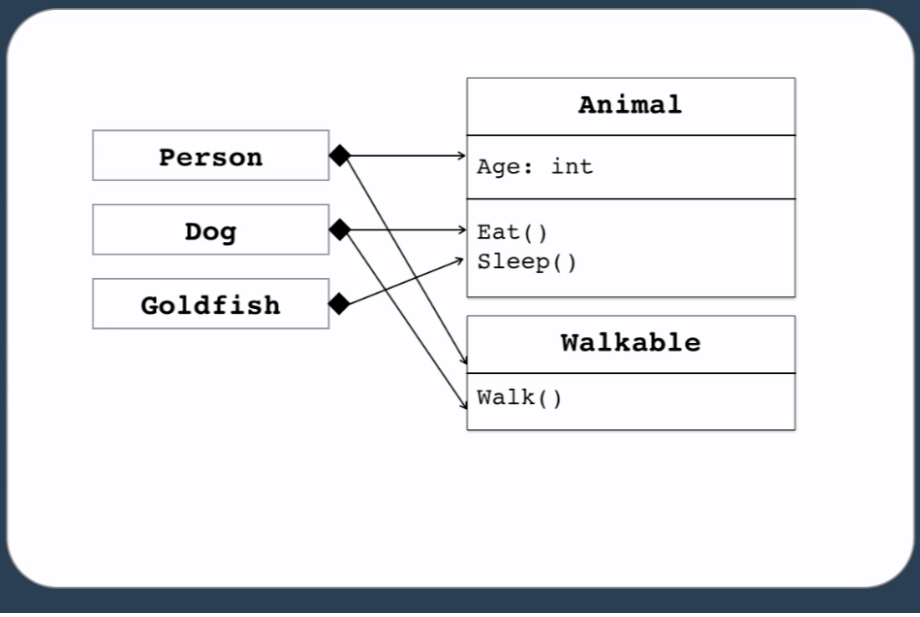
Composition

- Any Inheritance relationship can be translated to Composition.

Add Goldfish



Add GoldFish



Summary

- Inheritance
  - Pros: Code re-use, easier to understand
  - Cons: Tightly coupled, fragile, can be abused

Summary

- Composition
  - Pros: Code re-use, great flexibility, loose coupling
  - Cons: A little harder to understand

Constructor Inheritance

- Base class constructors are always executed first.
- Base class constructors are not inherited.

The base keyword

```
public class Car : Vehicle
{
    public Car(string registrationNumber)
        : base(registrationNumber)
    {
        // Initialise fields specific to the Car class
    }
}
```

The as keyword

```
Car car = (Car) obj;


Car car = obj as Car;
if (car != null)
{
    …
}
```

The is keyword

```
if (obj is Car)
{
    Car car = (Car) obj;
    …
}
```

Value Types

- Are stored on the stack.
- Examples:
  - All primitive types: byte, int, float, char, bool
  - The struct type

Earlier in this section...

- An object reference can be implicitly converted to a base class reference.

```
Circle circle = new Circle();
Shape shape = circle;
```

Also...

- The Object class is the base of all classes in .NET Framework.

```
Circle circle = new Circle();
Shape shape = circle;
object shape = circle;
```

Boxing

- The process of converting a value type instance to an object reference

```
int number = 10;
object obj = number;

// or
object obj = 10;
```

Unboxing

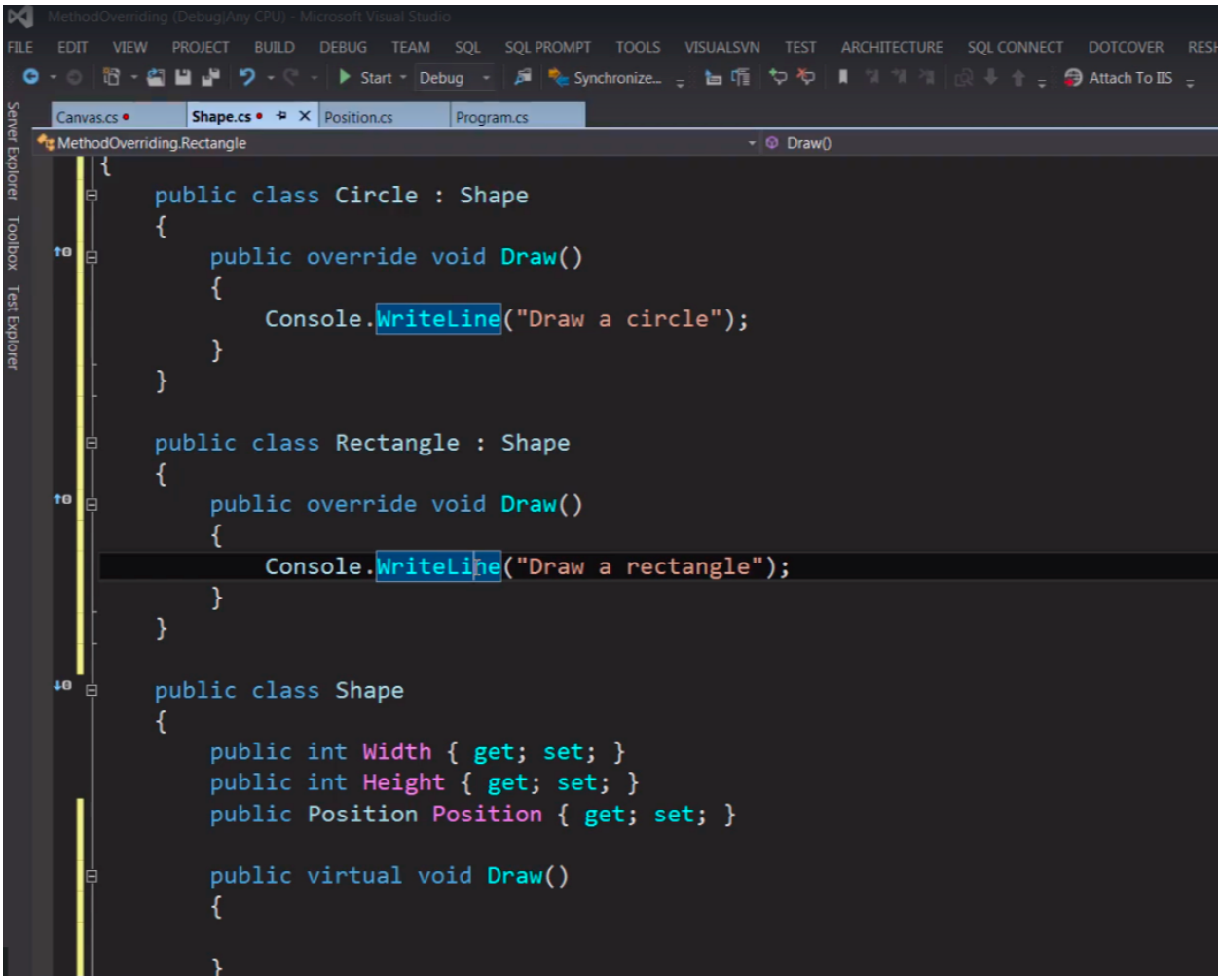
```
object obj = 10;
int number = (int)obj;
```

Method Overriding

- Modifying the implementation of an inherited method.

```
public class Shape
{
    public virtual void Draw()
    {
        // Default implementation
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        // New implementation
    }
}
```


MethodOverriding (Debug|Any CPU) - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL SQL PROMPT TOOLS VISUALSVN TEST ARCHITECTURE SQL CONNECT DOTCOVER
Start Debug Synchronize... Attach To IIS
Canvas.cs Shape.cs Position.cs Program.cs
MethodOverriding.Rectangle
Draw()
Server Explorer Toolbox Test Explorer
{
public class Circle : Shape
{
public override void Draw()
{
Console.WriteLine("Draw a circle");
}
}
public class Rectangle : Shape
{
public override void Draw()
{
Console.WriteLine("Draw a rectangle");
}
}
public class Shape
{
public int Width { get; set; }
public int Height { get; set; }
public Position Position { get; set; }
public virtual void Draw()
{
}

```
public abstract class Shape
{
    public abstract void Draw();
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Implementation for Circle
    }
}
```

Abstract Members

- Do not include implementation.

```
public abstract void Draw();
```

Derived Classes

- Must implement all abstract members in the base abstract class.

```
public class Circle : Shape
{
    public override void Draw()
    {
        // Implementation for Circle
    }
}
```

Why to use Abstract?

- When you want to provide some common behaviour, while forcing other developers to follow your design.

Sealed Modifier

- Prevents derivation of classes or overriding of methods.

Why do we need delegates?

- For designing extensible and flexible applications (eg frameworks)

Delegates

- An object that knows how to call a method (or a group of methods)
- A reference to a function

Interfaces or Delegates?

Use a delegate when

- An eventing design pattern is used.
- The caller doesn't need to access other properties or methods on the object implementing the method.

What is a Lambda Expression?

An anonymous method

- No access modifier
- No name
- No return statement

Events

- A mechanism for communication between objects
- Used in building *Loosely Coupled Applications*
- Helps extending applications

Delegates

- Agreement / Contract between Publisher and Subscriber
- Determines the signature of the event handler method in Subscriber

You can query

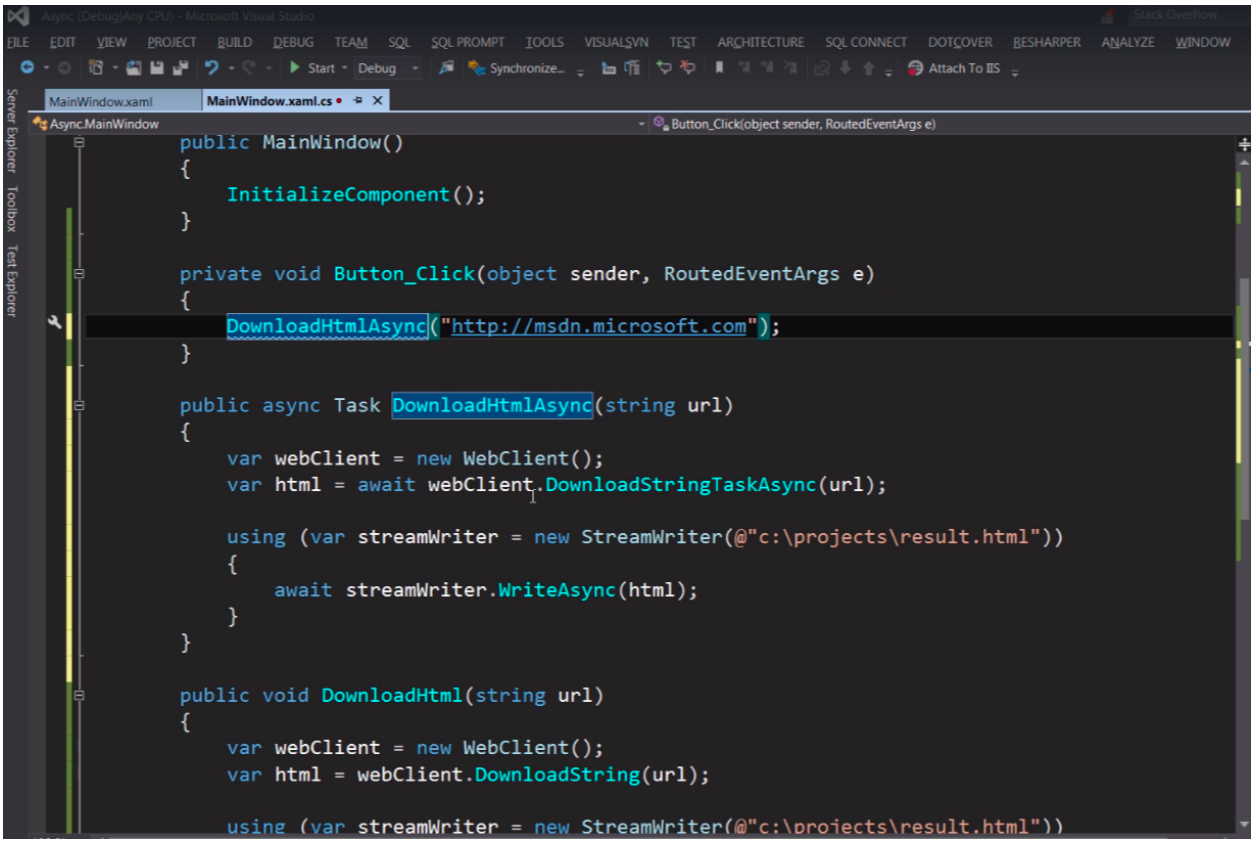
- Objects in memory, eg collections *(LINQ to Objects)*
- Databases *(LINQ to Entities)*
- XML *(LINQ to XML)*
- ADO.NET Data Sets *(LINQ to Data Sets)*

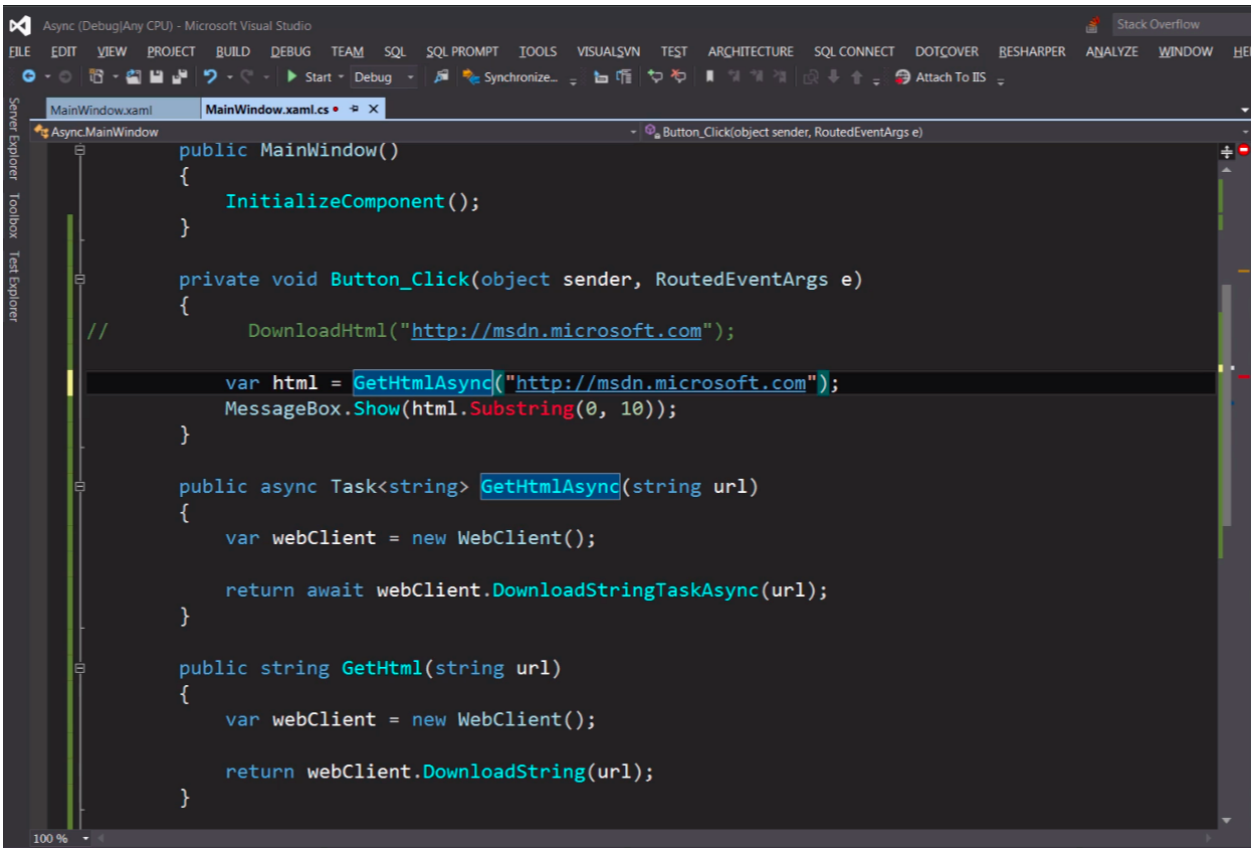
Synchronous Program Execution

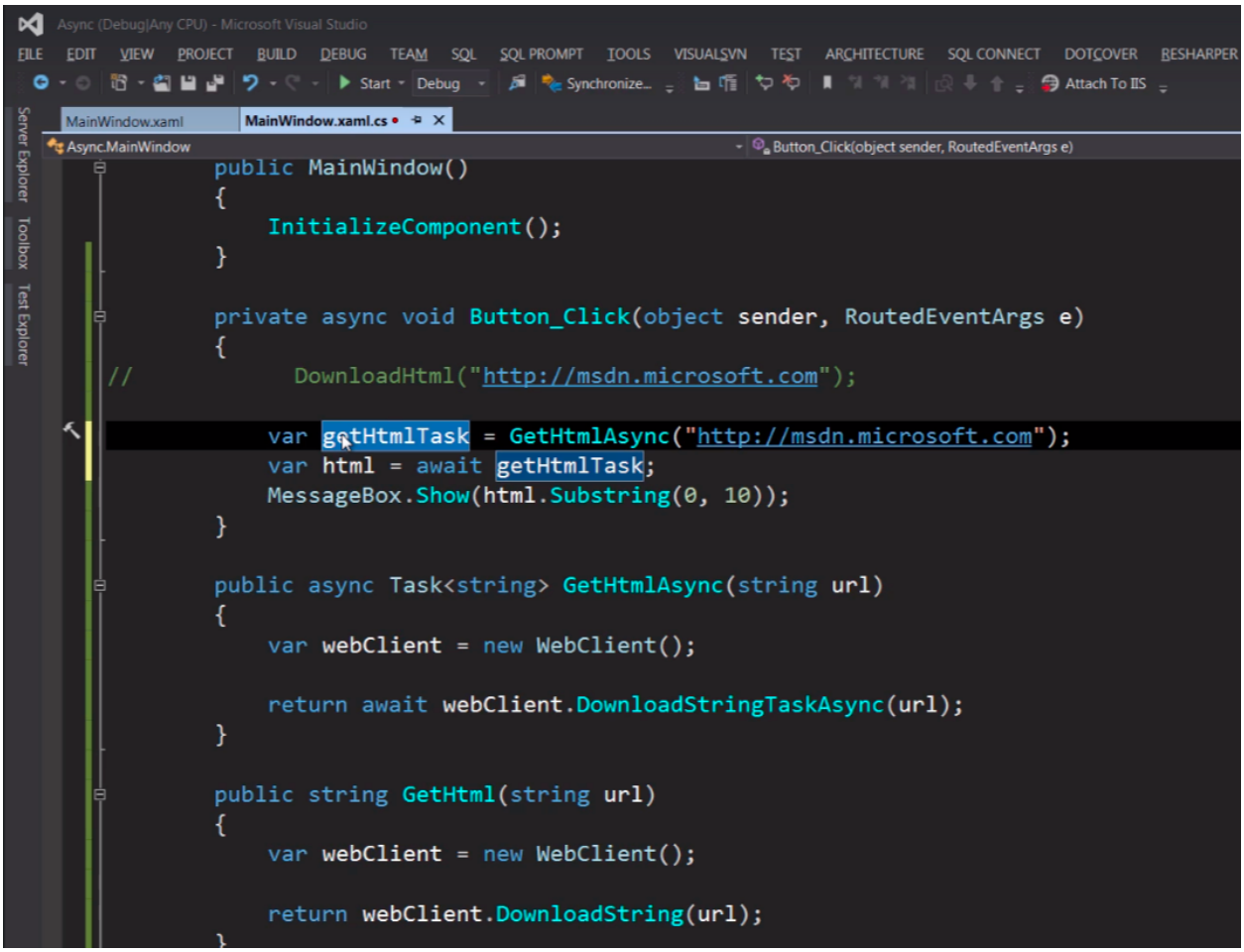
- Program is executed line by line, one at a time.
- When a function is called, program execution has to wait until the function returns.

Asynchronous Program Execution

- When a function is called, program execution continues to the next line, ***without*** waiting for the function to complete.

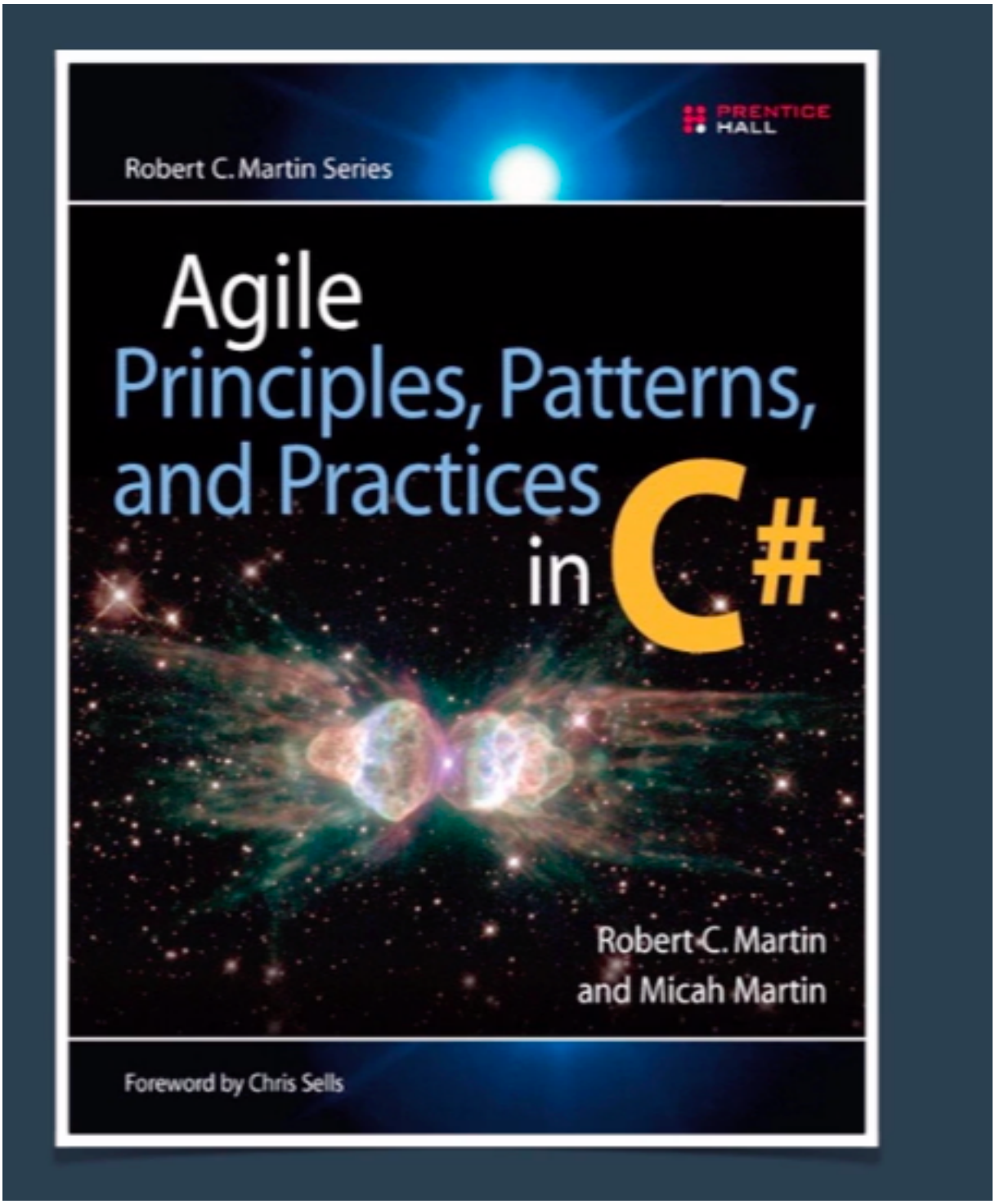

Async (Debug|Any CPU) - Microsoft Visual Studio
MainWindow.xaml
MainWindow.xaml.cs
Async.MainWindow
Button_Click(object sender, RoutedEventArgs e)
public MainWindow()
{
InitializeComponent();
}
private void Button_Click(object sender, RoutedEventArgs e)
{
DownloadHtmlAsync("http://msdn.microsoft.com");
}
public async Task DownloadHtmlAsync(string url)
{
var webClient = new WebClient();
var html = await webClient.DownloadStringTaskAsync(url);
using (var streamWriter = new StreamWriter(@"c:\projects\result.html"))
{
await streamWriter.WriteAsync(html);
}
}
public void DownloadHtml(string url)
{
var webClient = new WebClient();
var html = webClient.DownloadString(url);

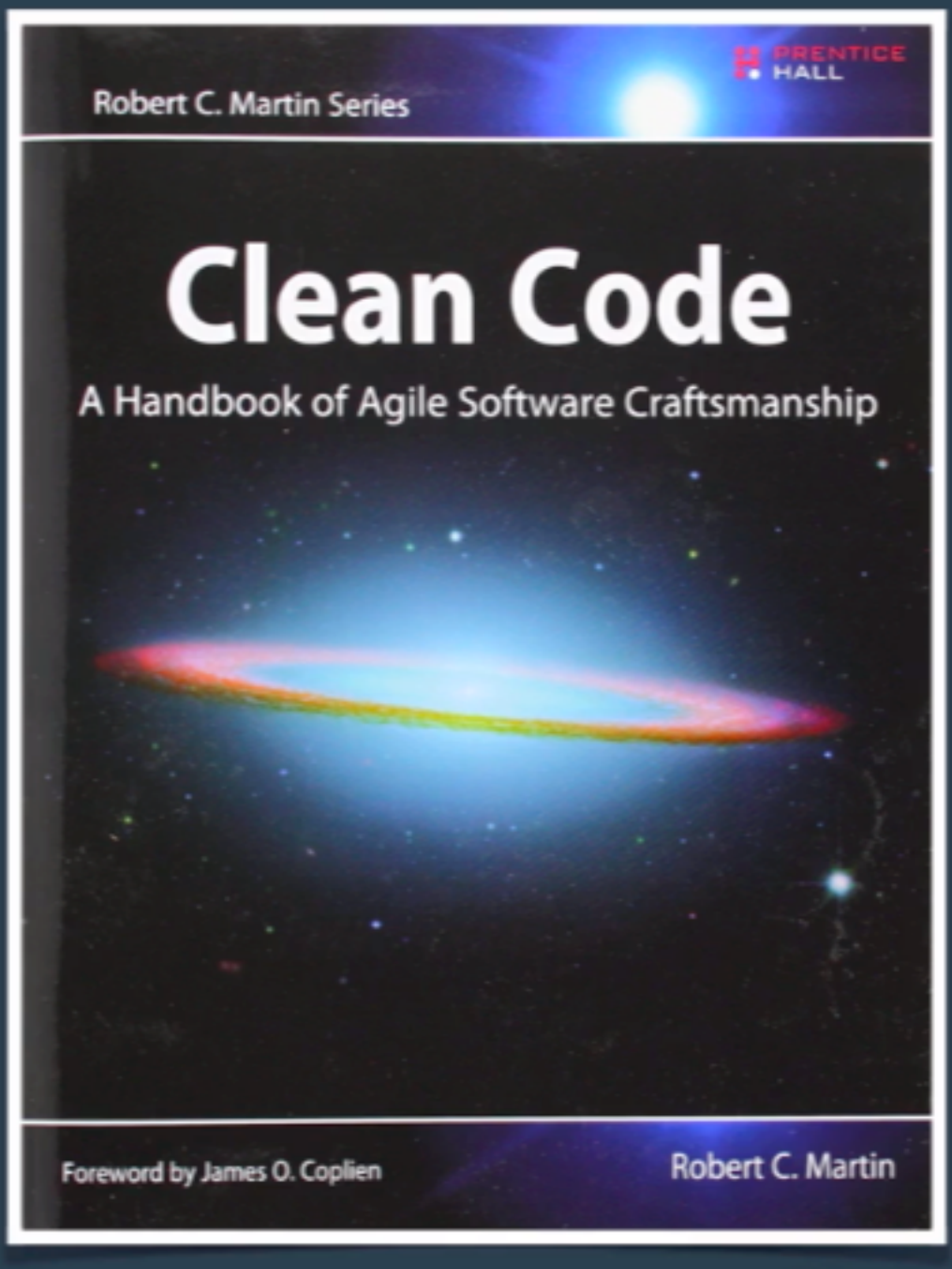

Async (Debug|Any CPU) - Microsoft Visual Studio
MainWindow.xaml
MainWindow.xaml.cs
Async.MainWindow
Button_Click(object sender, RoutedEventArgs e)
public MainWindow()
{
InitializeComponent();
}
private void Button_Click(object sender, RoutedEventArgs e)
{
// DownloadHtml("http://msdn.microsoft.com");
var html = GetHtmlAsync("http://msdn.microsoft.com");
MessageBox.Show(html.Substring(0, 10));
}
public async Task<string> GetHtmlAsync(string url)
{
var webClient = new WebClient();
return await webClient.DownloadStringTaskAsync(url);
}
public string GetHtml(string url)
{
var webClient = new WebClient();
return webClient.DownloadString(url);
}


Async (Debug|Any CPU) - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL SQL PROMPT TOOLS VISUALSVN TEST ARCHITECTURE SQL CONNECT DOTCOVER RESHARPER
Start Debug Synchronize... Attach To IIS
Server Explorer Toolbox Test Explorer
MainWindow.xaml MainWindow.xaml.cs
Async.MainWindow
Button_Click(object sender, RoutedEventArgs e)
public MainWindow()
{
InitializeComponent();
}
private async void Button_Click(object sender, RoutedEventArgs e)
{
// DownloadHtml("http://msdn.microsoft.com");
var getHtmlTask = GetHtmlAsync("http://msdn.microsoft.com");
var html = await getHtmlTask;
MessageBox.Show(html.Substring(0, 10));
}
public async Task<string> GetHtmlAsync(string url)
{
var webClient = new WebClient();
return await webClient.DownloadStringTaskAsync(url);
}
public string GetHtml(string url)
{
var webClient = new WebClient();
return webClient.DownloadString(url);
}

Fundamentals

- Algorithms
- Data Structures
- Object-oriented Programming
- Clean Coding
- Refactoring


PRENTICE HALL
Robert C. Martin Series
Agile Principles, Patterns, and Practices in C#
Robert C. Martin and Micah Martin
Foreword by Chris Sells


Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

the art of

UNIT TESTING

with Examples in .NET

manning

ROY OSHEROVE

Important Commands

- Ctrl + Tab = Move between documents.

Important Shortcuts

- ctor = Generate constructor.
- cw = Print something.
- prop = Create property.

Coding Principles - Very Important

- Don’t repeat yourself.
- Don’t add a lot of comments, your code should be good enough to indicate what is happening.
- Set good and descriptive names for the variables, methods, and classes.
- Prefer composition over inheritance.
- Try to use interfaces for extensibility and testing of your software.
- Keep your code open to extensions and close to modifications.
- Always do unit/integration testing.

Important C# Concepts

- Func() : Function with return.
- Action(): Function with no return.