Diagrams and Screen snapshots from the Course.

This document contains screen snapshots I took during the course. There is a mixture of annotated code and diagrams. At the top of each page, I indicate the module in the course a diagram applies to, and I usually write something about it.

What you're looking at is a PDF document. If you want a better browsing experience you can download it, zoom in, whatever.

Mark.

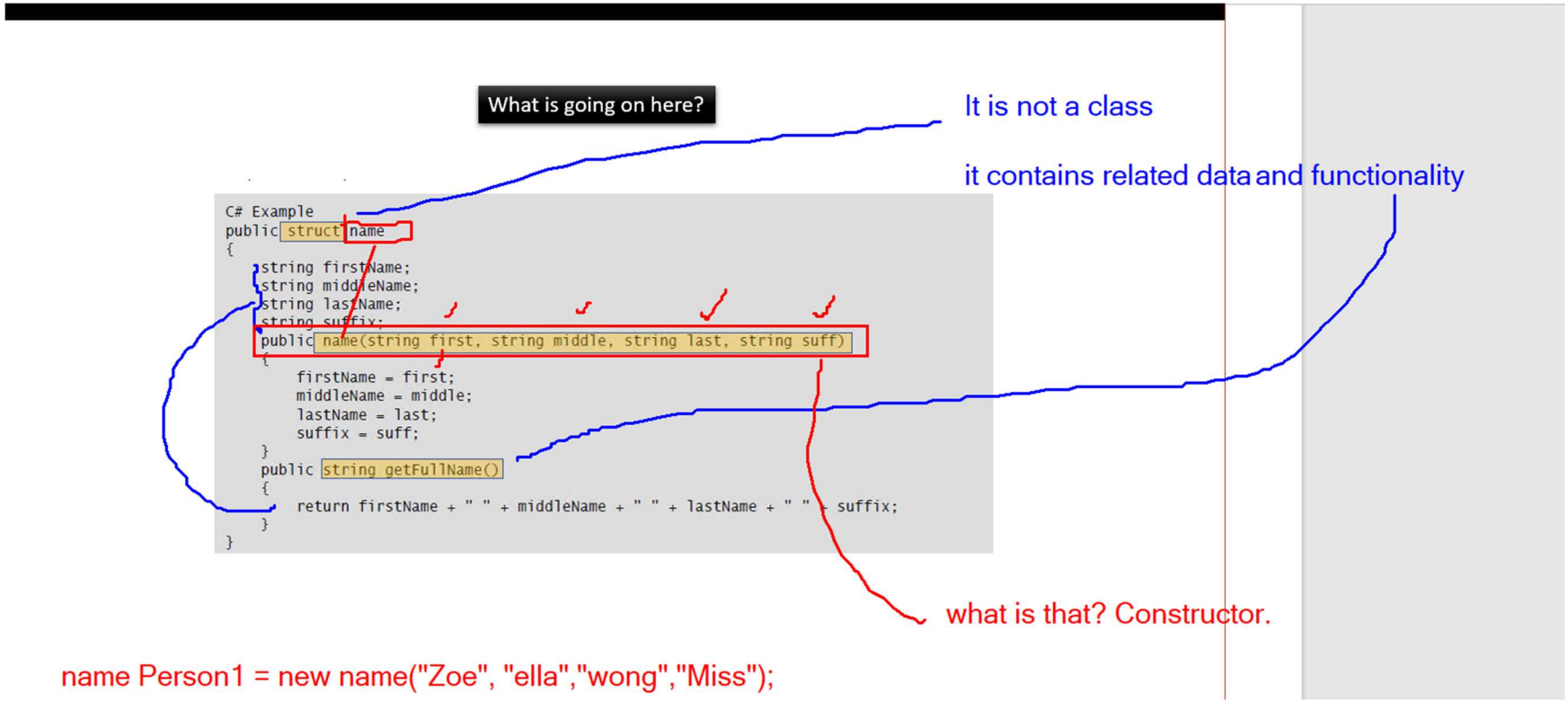
MODULE 1.

When does this loop end?

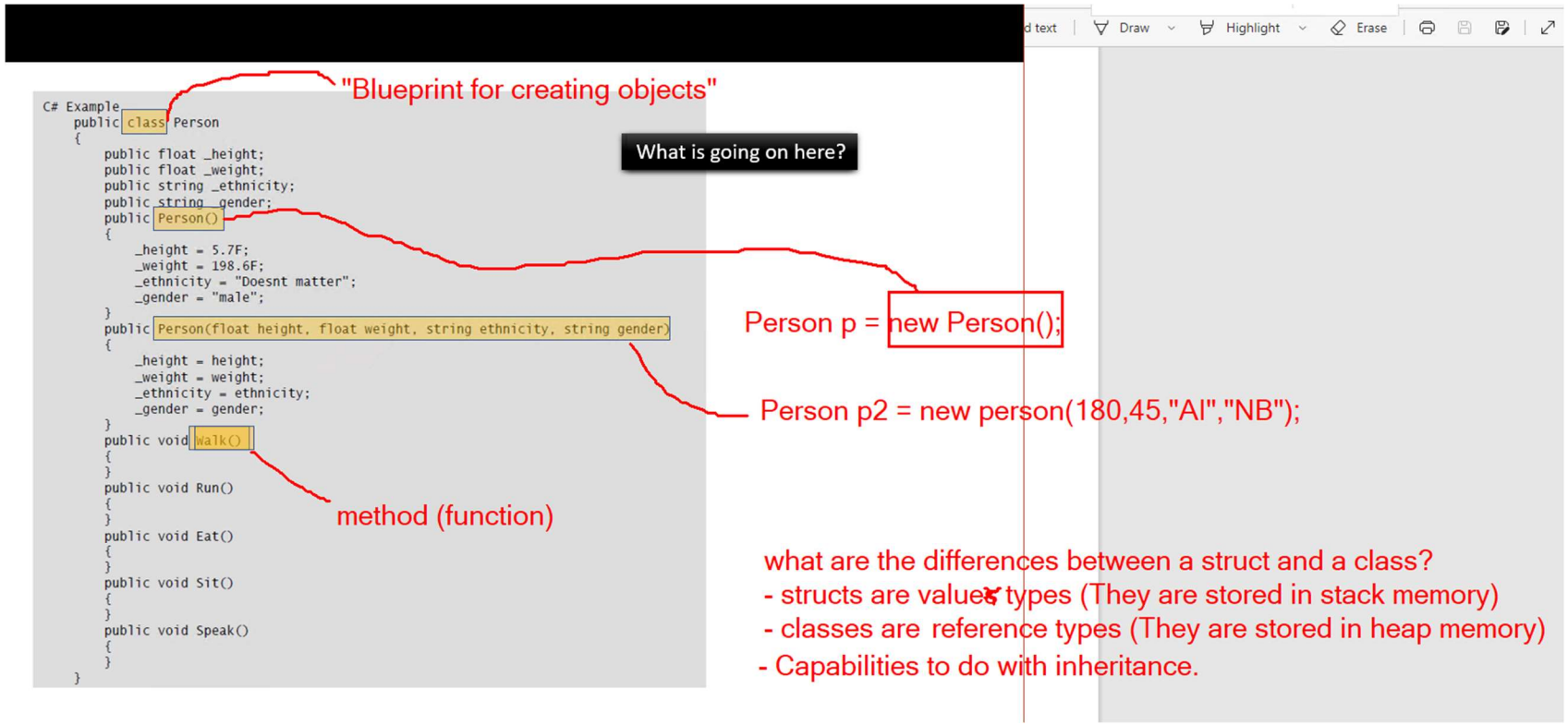
```
for(int loopCounter = 0; loopCounter < 10; loopCounter++)
{
  Console.Out.WriteLine("loopCounter is at: " + loopCounter);
  total = total + loopCounter;
}
Console.Out.WriteLine("Total of values is: " + total);
Console.Out.WriteLine("Average is: " + total / 10);
```

```
console.Out.WriteLine($"Total of values is: {total}");
```

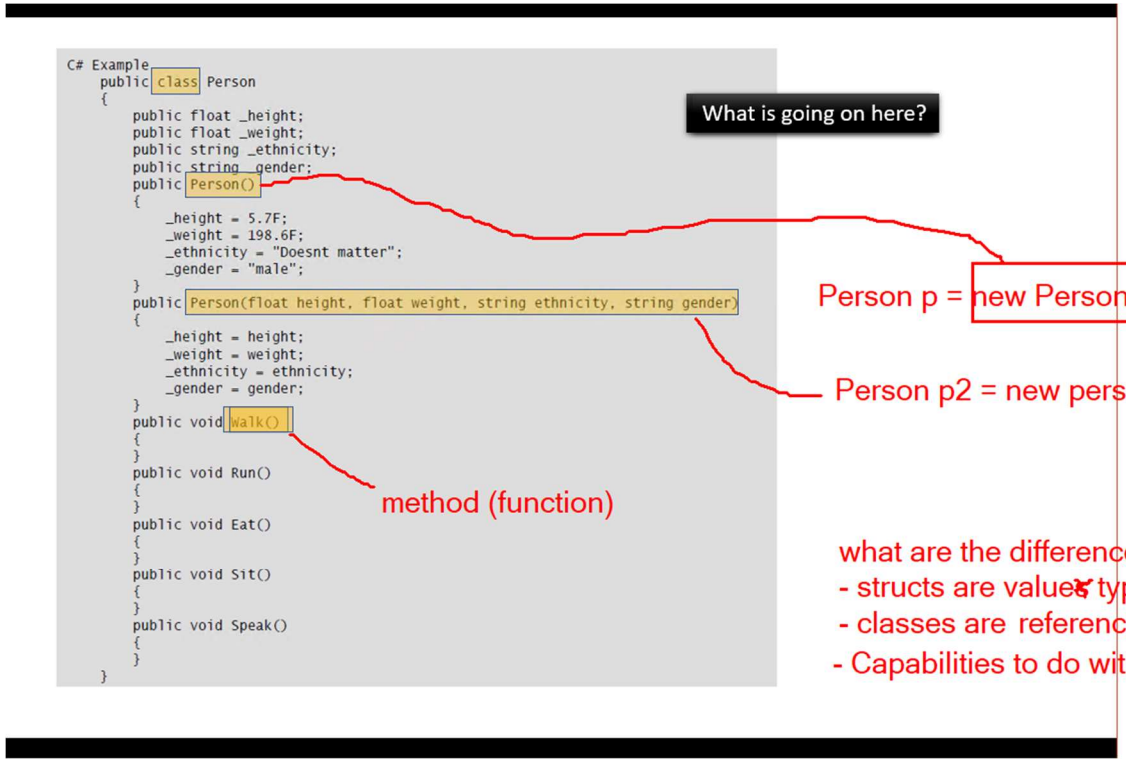
MODULE 1.


What is going on here?
It is not a class
it contains related data and functionality
C# Example
public struct name
{
string firstName;
string middleName;
string lastName;
string suffix;
public name(string first, string middle, string last, string suff)
{
firstName = first;
middleName = middle;
lastName = last;
suffix = suff;
}
public string getFullName()
{
return firstName + " " + middleName + " " + lastName + " " + suffix;
}
}
what is that? Constructor.
name Person1 = new name("Zoe", "ella","wong","Miss");

MODULE 1.


"Blueprint for creating objects"
What is going on here?
C# Example
public class Person
{
public float _height;
public float _weight;
public string _ethnicity;
public string _gender;
public Person()
{
_height = 5.7F;
_weight = 198.6F;
_ethnicity = "Doesnt matter";
_gender = "male";
}
public Person(float height, float weight, string ethnicity, string gender)
{
_height = height;
_weight = weight;
_ethnicity = ethnicity;
_gender = gender;
}
public void Walk()
{
}
public void Run()
{
}
public void Eat()
{
}
public void Sit()
{
}
public void Speak()
{
}
}
Person p = new Person();
Person p2 = new person(180,45,"AI","NB");
method (function)
what are the differences between a struct and a class?
- structs are values types (They are stored in stack memory)
- classes are reference types (They are stored in heap memory)
- Capabilities to do with inheritance.

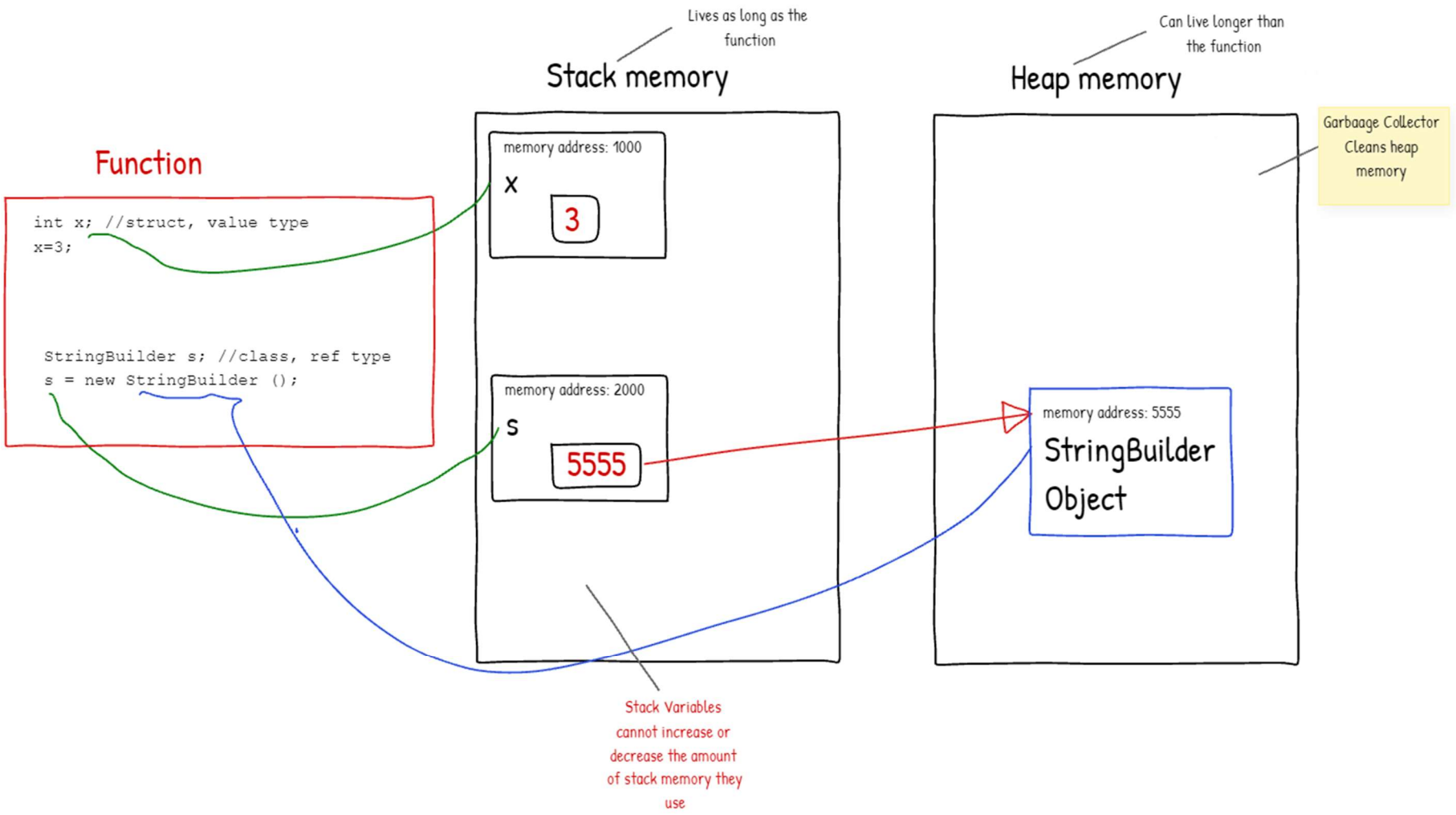
MODULE 1.



what are the differences between a struct and a class?
- structs are values types (They are stored in stack memory)
- classes are reference types (They are stored in heap memory)
- Capabilities to do with inheritance.

MODULE 1.

Value Type Vs Reference Type



MODULE 1.

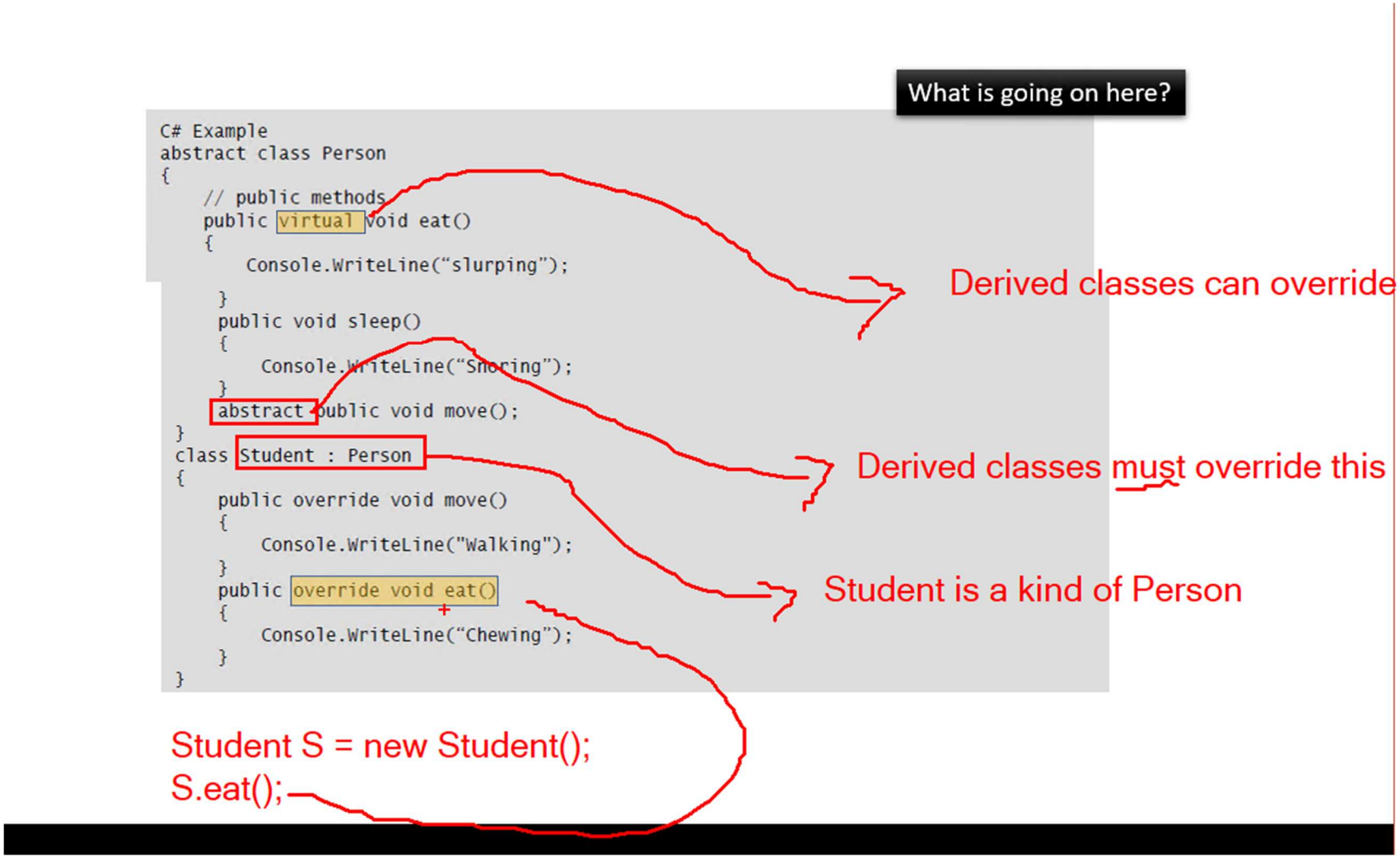
When 2 or more objects have the same interface they can be used interchangeably.|

What is *Polymorphism*?

MODULE 1.

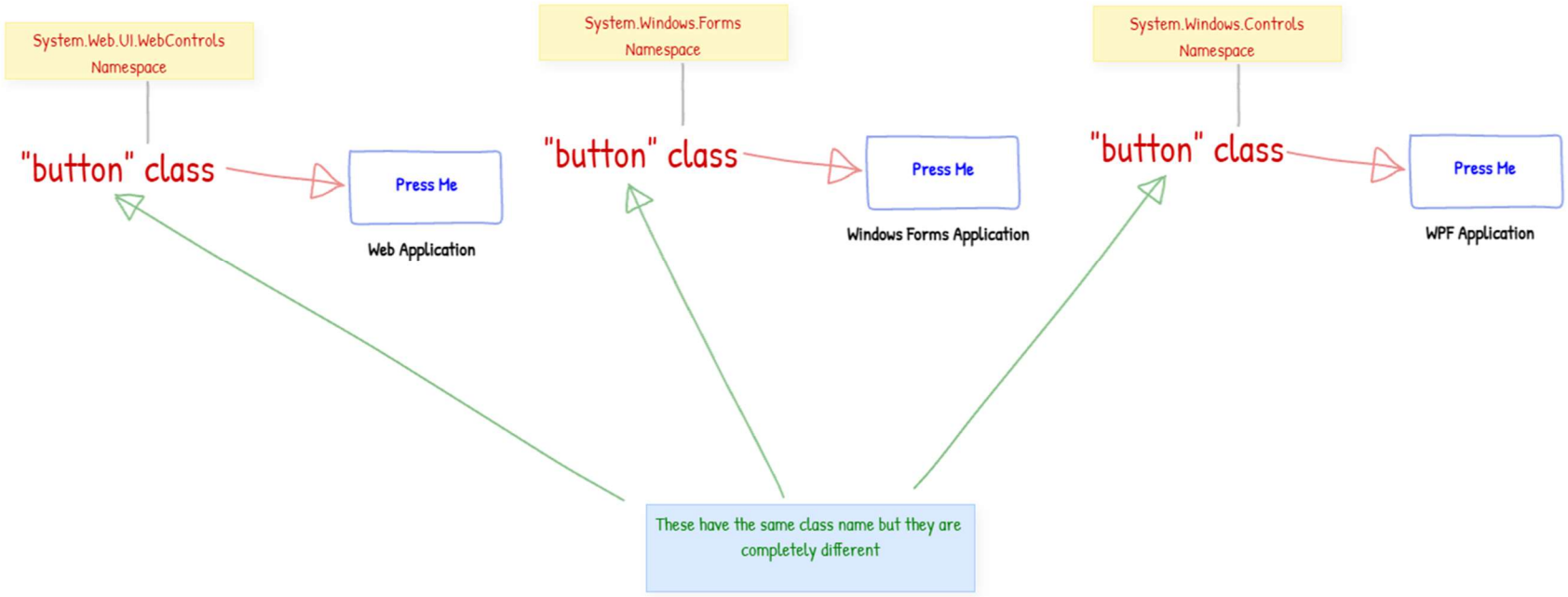
PolyMorphism in everyday life

MODULE 1.


What is going on here?
C# Example
abstract class Person
{
// public methods
public virtual void eat()
{
Console.WriteLine("slurping");
}
public void sleep()
{
Console.WriteLine("Snoring");
}
abstract public void move();
}
class Student : Person
{
public override void move()
{
Console.WriteLine("Walking");
}
public override void eat()
{
Console.WriteLine("Chewing");
}
}
Derived classes can override this
Derived classes must override this
Student is a kind of Person
Student S = new Student();
S.eat();

MODULE 1.

“Namespaces” in the .net framework organise datatypes (classes, structures, etc) so that they are easy for us to find and they have a context. In this diagram you see there are multiple classes called “button”. Namespaces distinguish between each of them.



MODULE 1.

Creating and Using Arrays

page 1-23,24

- C# supports:
  - Single-dimensional arrays
  - Multidimensional arrays
  - Jagged arrays

Tell me anything you can remember about Arrays:
- Every element in the array has the same data type.
- Arrays cannot increase or decrease the number of elements.
- Arrays are objects, they are created in Heap memory.
- An array is a collection of elements that are logically related.
- Arrays are contiguous storage, therefore they are fast|

- Creating an array

```
int[] arrayName = new int[10];
```

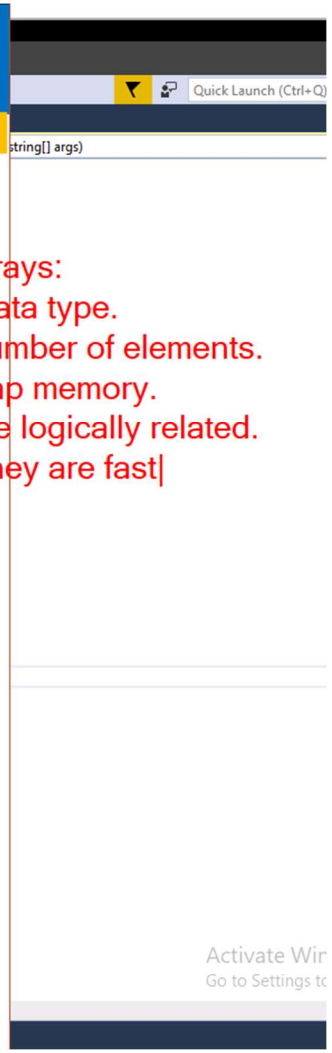
- Accessing data in an array:
  - By index

subscript (a.k.a. indexer)

```
int result = arrayName[2];
```

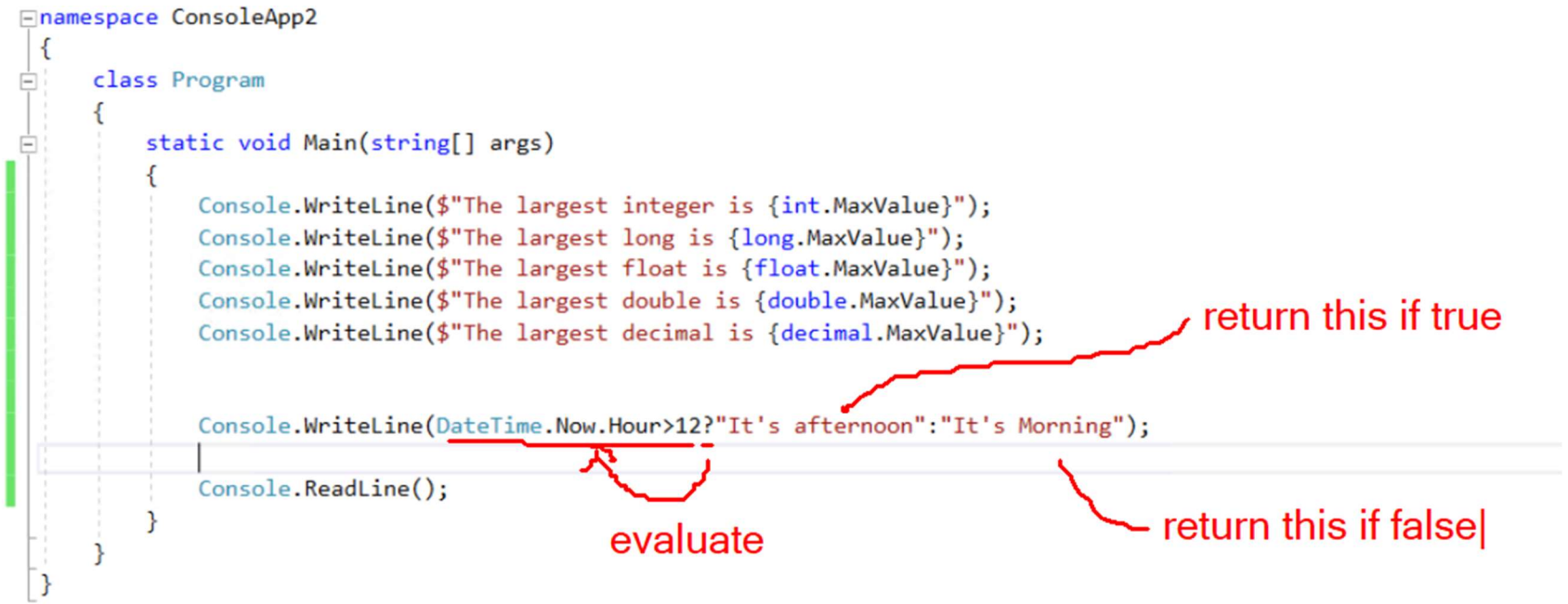
  - In a loop

```
for (int i = 0; i < arrayName.Length; i++)
{
    int result = arrayName[i];
}
```



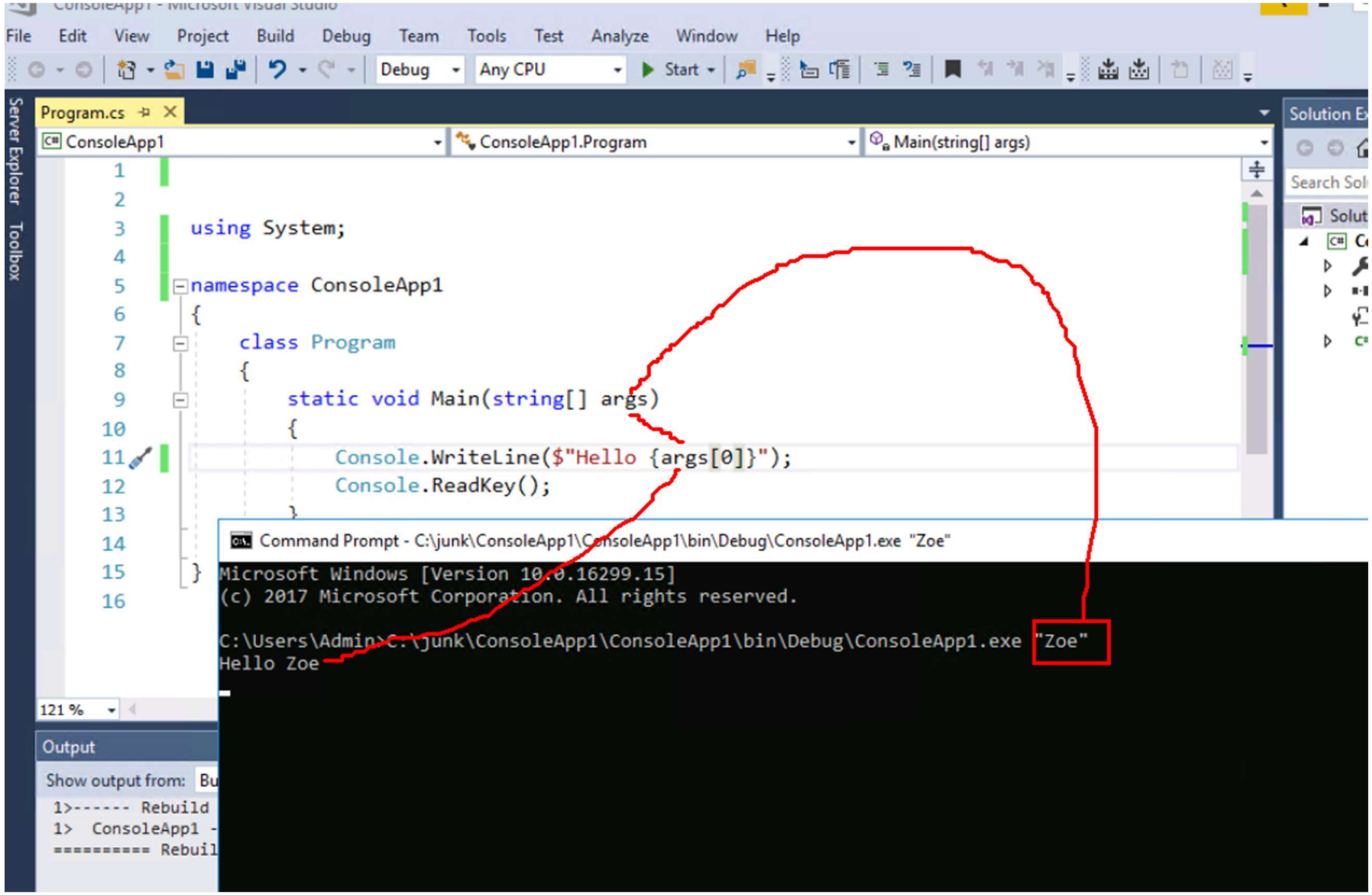
MODULE 1.

The 'Ternary' operator in C#. (? :)



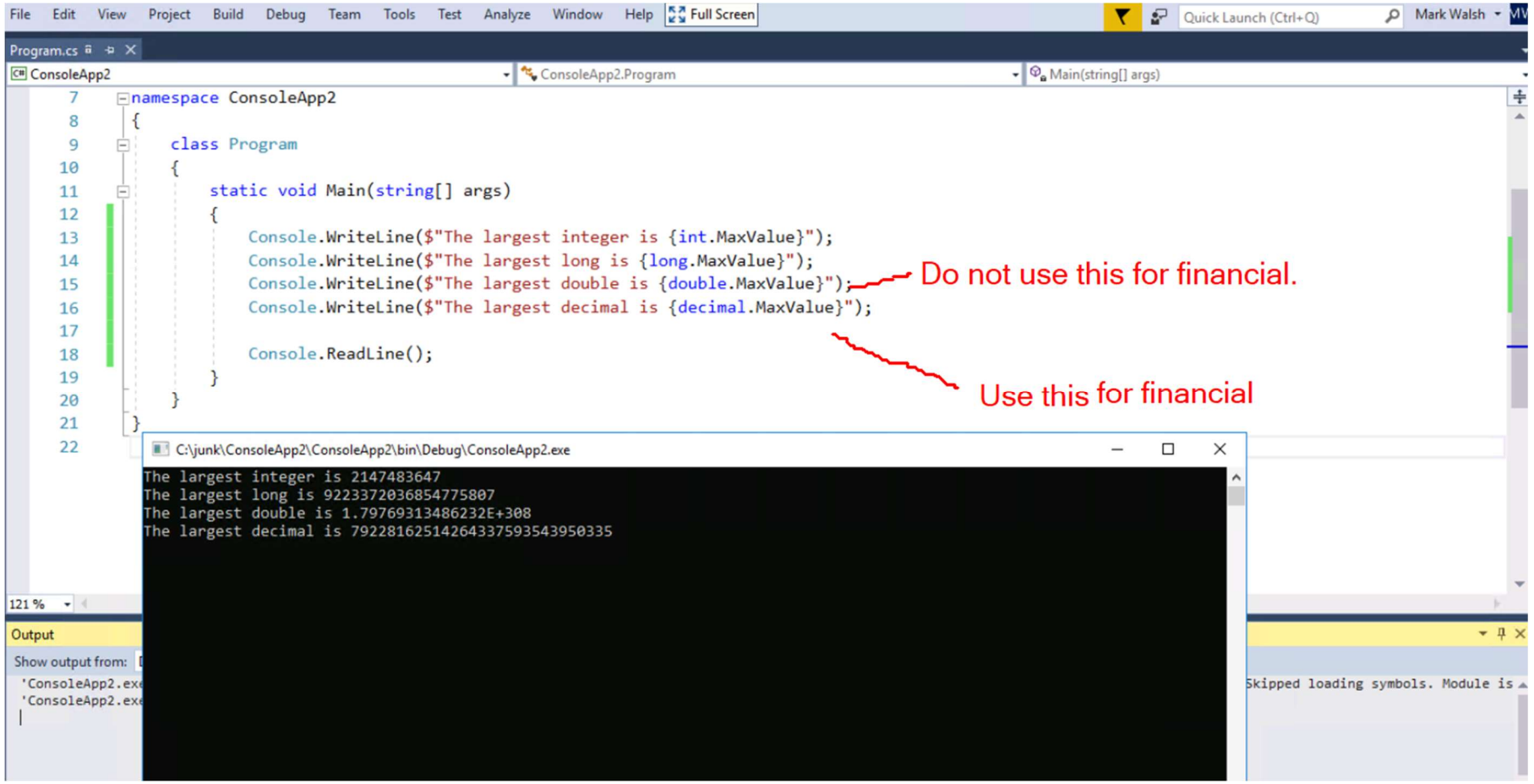
MODULE 1.

Passing arguments when starting an application



MODULE 1.

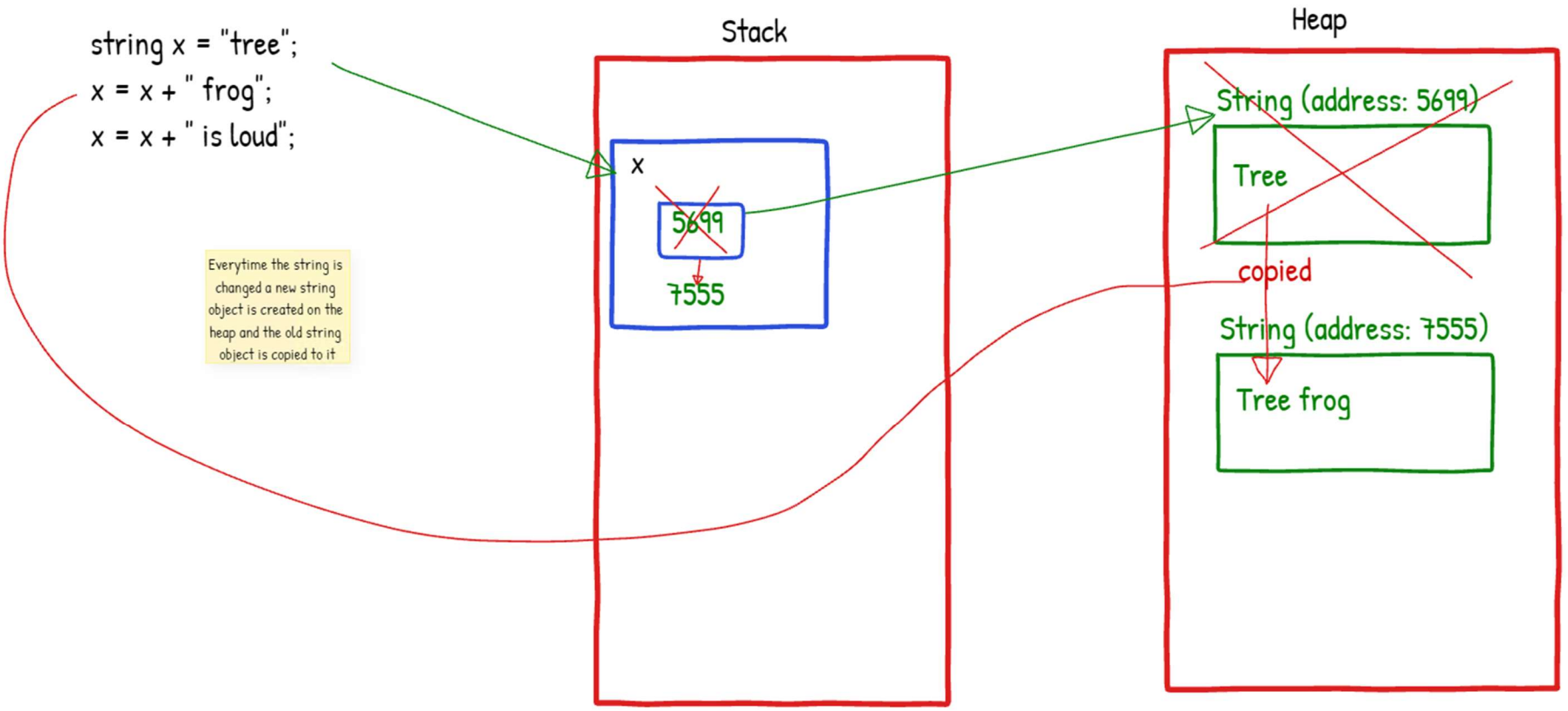
Be careful with the datatypes you choose.



`

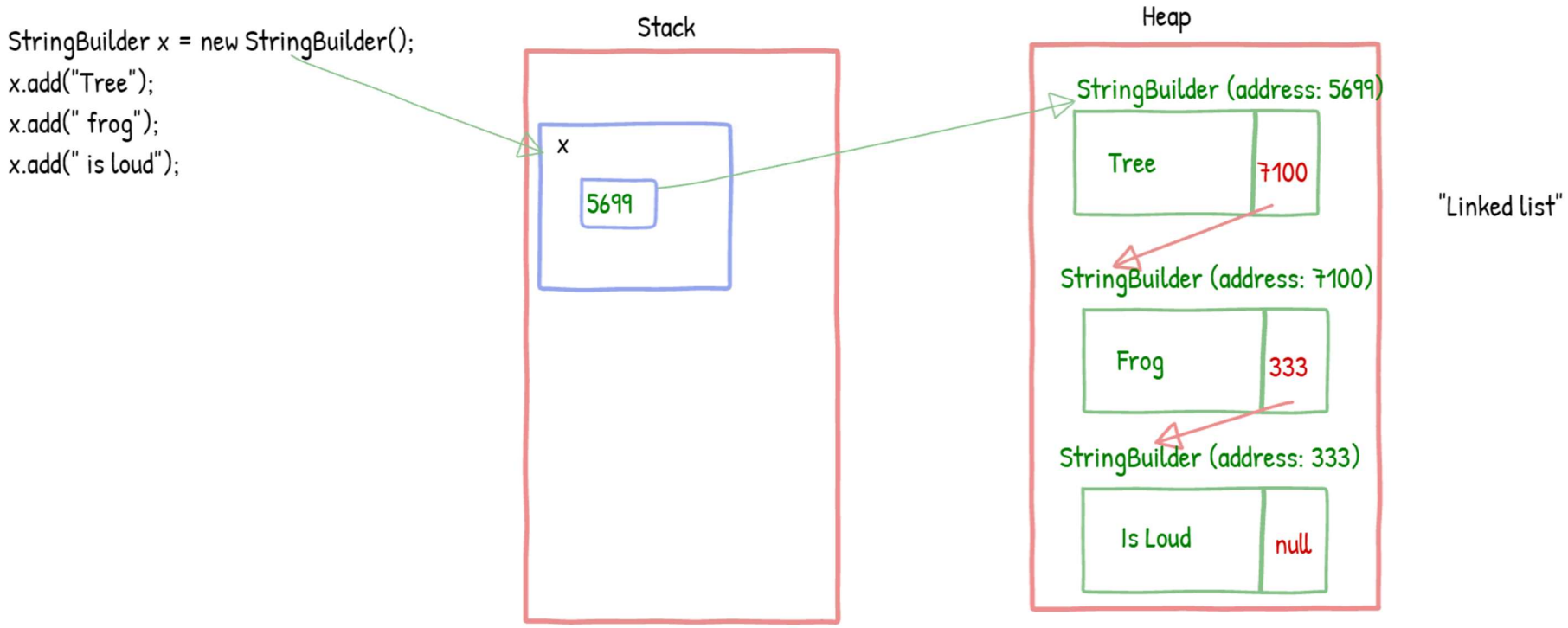
MODULE 1.

Many changes to a string can result in slowing down your application and the device it is running on. In this case whenever a string is changed in code an new string object must be created and the old string contents copied to it; why? Because strings are *arrays*, and arrays cannot change in size. All this copying palaver slows things down.



MODULE 1.

If you are making many changes to a string then use the string builder class. See how it adds a new object each time for the next part of the string and a pointer is updated to indicate where the new object is?



MODULE 2.

The terminology of methods

Creating Methods

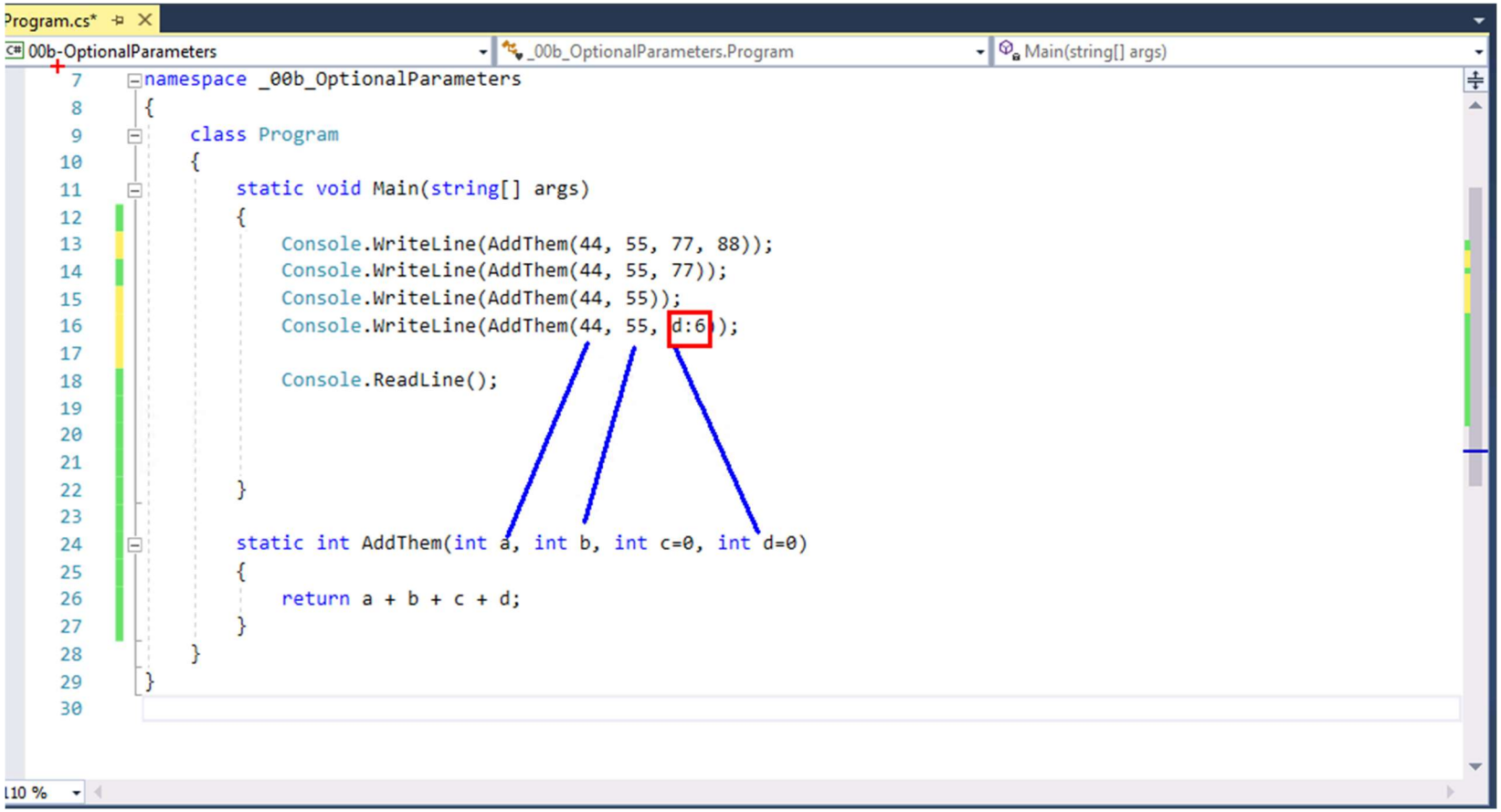
page 2-3,4

- Methods comprise two elements:
  - Method specification (return type, name, parameters)
  - Method body
- Use the **ref** keyword to pass parameter references

```
void StartService(int upTime, bool shutdownAutomatically)
{
    // Perform some processing here.
}
```

MODULE 2.

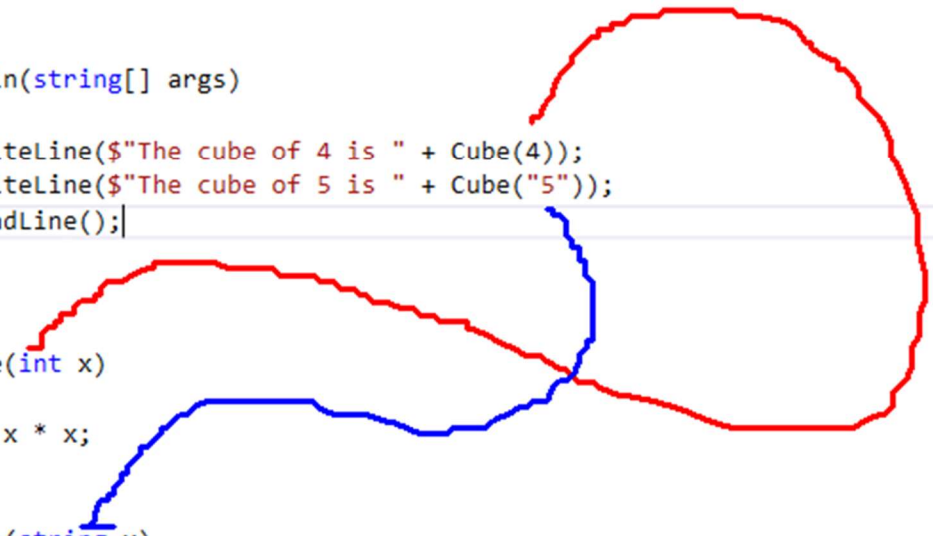
"named parameters". In this case I wanted to pass a value for ***d*** but not for ***c***



MODULE 2.

When two methods have the same name but a different number of paramaters and/or different parameter types they are said to be ***overloaded*** . Can you work out why we have this concept?

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace _00a_OverloadedMethods
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              Console.WriteLine($"The cube of 4 is " + Cube(4));
14              Console.WriteLine($"The cube of 5 is " + Cube("5"));
15              Console.ReadLine();
16
17          }
18
19          static int Cube(int x)
20          {
21              return x * x * x;
22          }
23
24          static int Cube(string x)
25          {
26              int z = Convert.ToInt32(x);
27              return z * z * z;
28          }
29      }
30  }
31
```



MODULE 2.

An example of an OUT parameter

```
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void btn_Click(object sender, RoutedEventArgs e)
    {
        int num;
        if (int.TryParse(TextBox1.Text, out num))
        {
            lblResult.Content = $"The square of {num} is {(num * num)}";
        }
        else
        {
            lblResult.Content = "Try a number!";
        }
    }
}
```

if the value in Textbox1.Text is an integer then tryparse will return true and num will contain the integer.

MODULE 2.

Another example of an OUT parameter. The GetFiles function returns the number of files and the OUT parameter called *fileNames* will contain an array with all the file names.

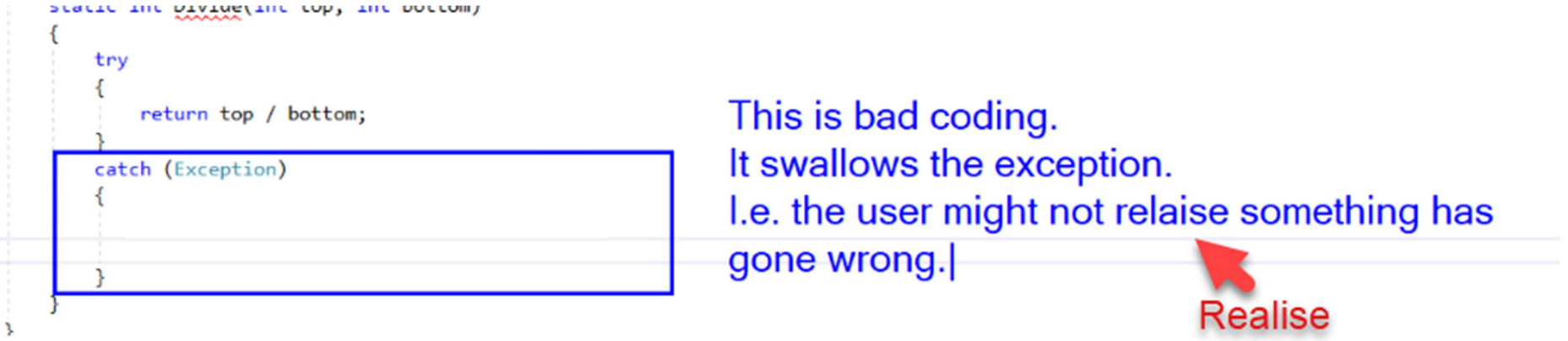
```
static void Main(string[] args)
{
    ArrayList x;

    int filecount=GetFiles(@"C:\Windows\system32", out x);

    Console.WriteLine($"The number of files in C:\\Windows\\system32 is {filecount}. Press enter to see them all.");
    Console.ReadLine();

    foreach (var item in x)
    {
        Console.WriteLine(item);
    }

    Console.WriteLine("OK. Done. Press enter to finish");

    Console.ReadLine();
}

static int GetFiles(string path, out ArrayList fileNames)
{
    int fileCount = 0;
    fileNames = new ArrayList();
    foreach (var item in Directory.GetFiles(path))
    {
        fileCount++;
        fileNames.Add(item);
    } ≤ 1ms elapsed


    return fileCount;
}
```

MODULE 2.

This example shows the basics of implementing exception handling

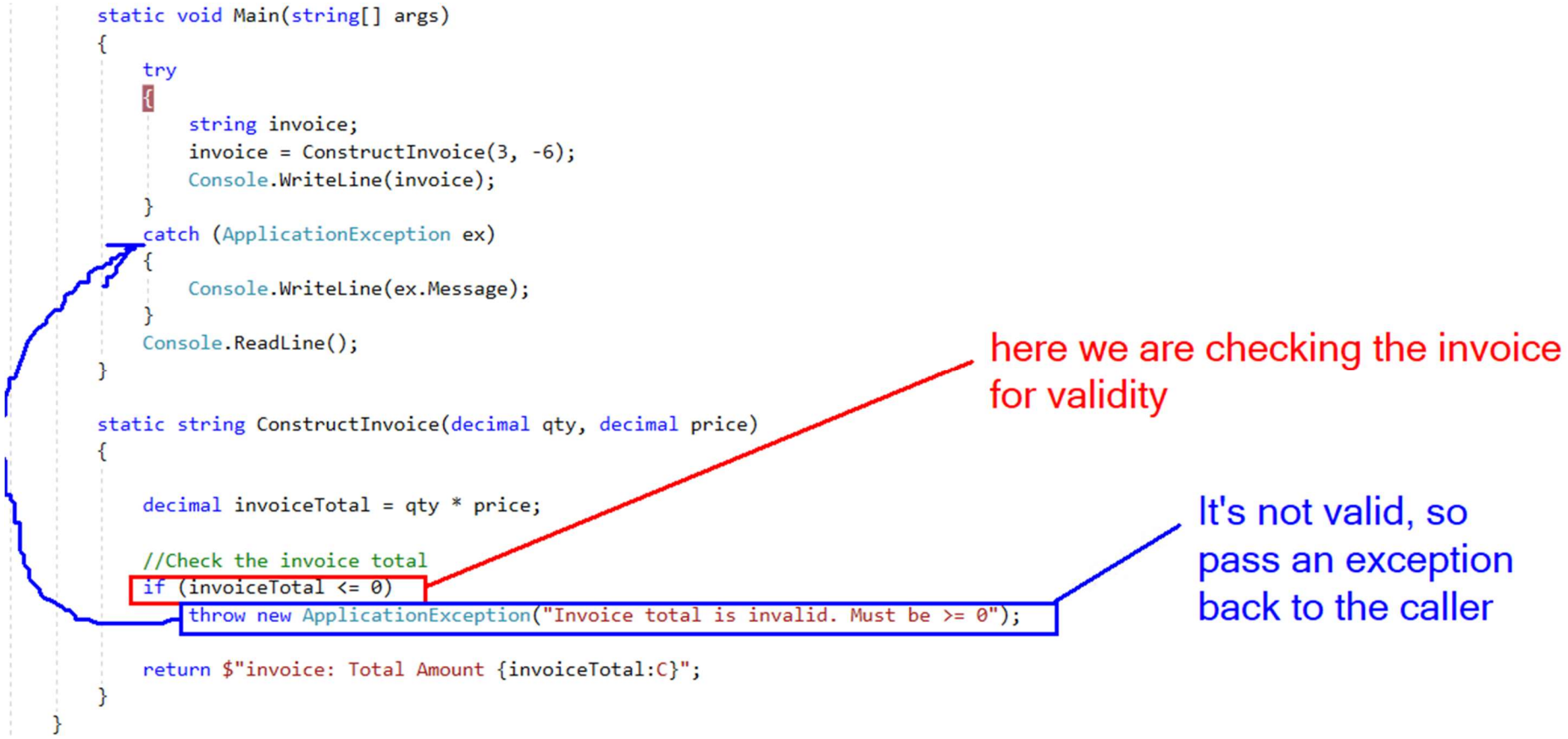
MODULE 2.

One Situation you want to avoid is when you application gets an exception and you can't get around it and you don't inform the user that something has gone wrong. The application might continue in a state where the data is corrupt for example.



MODULE 2.

Sometimes you want to throw exceptions in your code because you have a situation where an application shouldn't continue. In this case an invoice is being raised with a negative amount; that's not allowed.

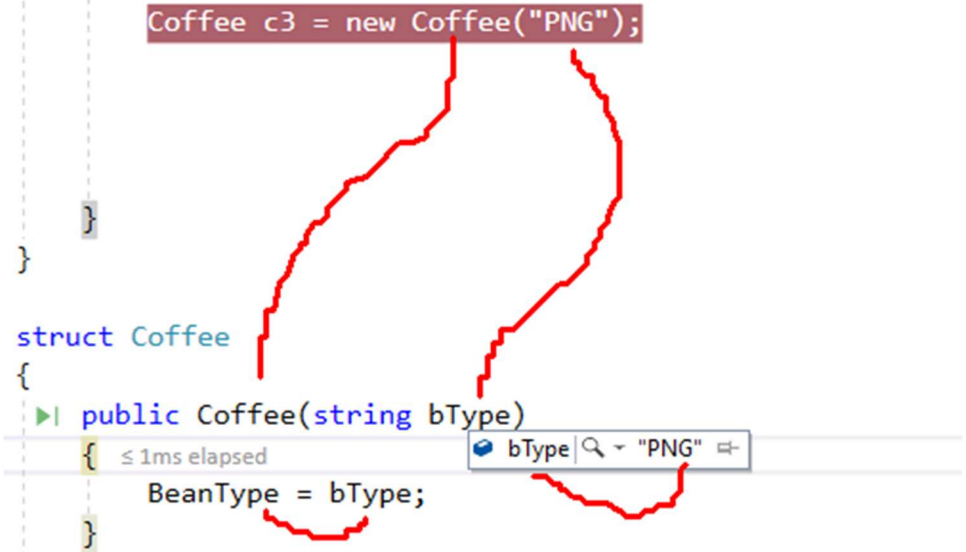



```
static void Main(string[] args)
{
    try
    {
        string invoice;
        invoice = ConstructInvoice(3, -6);
        Console.WriteLine(invoice);
    }
    catch (ApplicationException ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.ReadLine();
}

static string ConstructInvoice(decimal qty, decimal price)
{

    decimal invoiceTotal = qty * price;

    //Check the invoice total
    if (invoiceTotal <= 0)
        throw new ApplicationException("Invoice total is invalid. Must be >= 0");

    return $"invoice: Total Amount {invoiceTotal:C}";
}
}
```

MODULE 3.

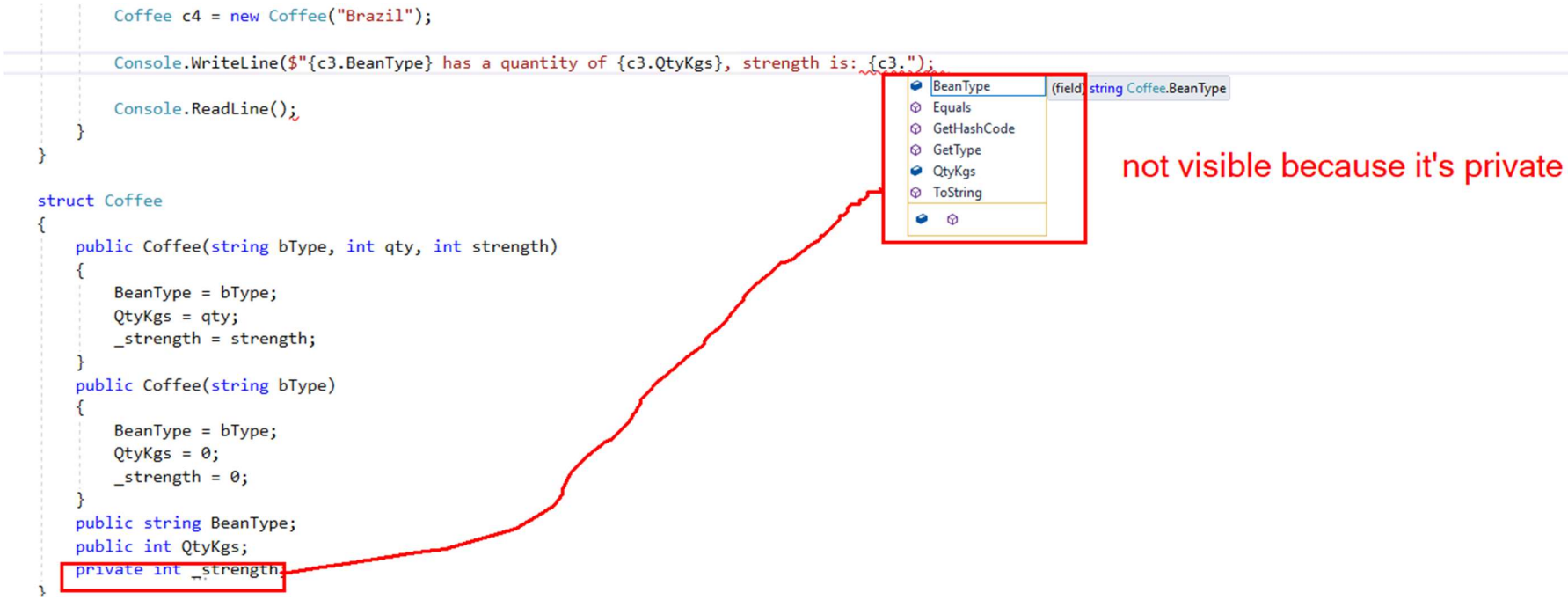
This is an example of using a constructor with a Struct.

```
class Program
{
    static void Main(string[] args)
    {
        Coffee c1 = new Coffee();
        c1.BeanType = "Aribica";

        Coffee c2 = new Coffee();
        c2.BeanType = "Kenyan";


        Coffee c3 = new Coffee("PNG");



    }
}

struct Coffee
{
    public Coffee(string bType)
    {  ≤ 1ms elapsed
        BeanType = bType;
    }
    public string BeanType;
}
```



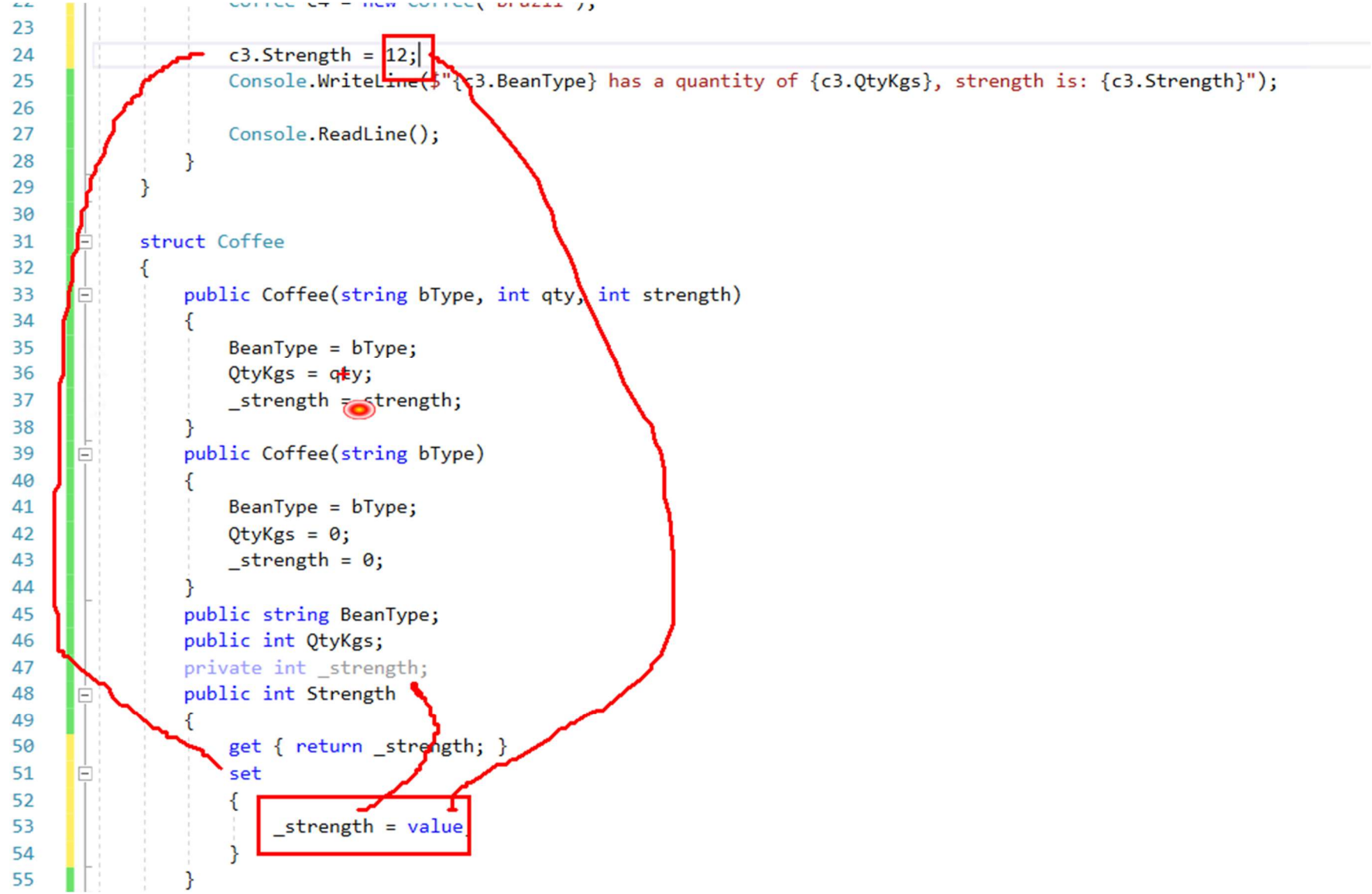
MODULE 3.

Private fields are not visible outside a struct or class



MODULE 3.

This is an example of protecting a private field by using a Property.



```
23
24              c3.Strength = 12;
25              Console.WriteLine($"{c3.BeanType} has a quantity of {c3.QtyKgs}, strength is: {c3.Strength}");
26
27              Console.ReadLine();
28          }
29      }
30
31      struct Coffee
32      {
33          public Coffee(string bType, int qty, int strength)
34          {
35              BeanType = bType;
36              QtyKgs = qty;
37              _strength = strength;
38          }
39          public Coffee(string bType)
40          {
41              BeanType = bType;
42              QtyKgs = 0;
43              _strength = 0;
44          }
45          public string BeanType;
46          public int QtyKgs;
47          private int _strength;
48          public int Strength
49          {
50              get { return _strength; }
51              set
52              {
53                  _strength = value;
54              }
55          }
```

MODULE 3.

The reason you'd define an indexer is to make it easier for other people who use your class or struct. Indexers are special methods that use a *subscript* to set or get data. The method name for an indexer is always *this*.

- Use the **this** keyword to declare an indexer
- Use **get** and **set** accessors to provide access to the collection

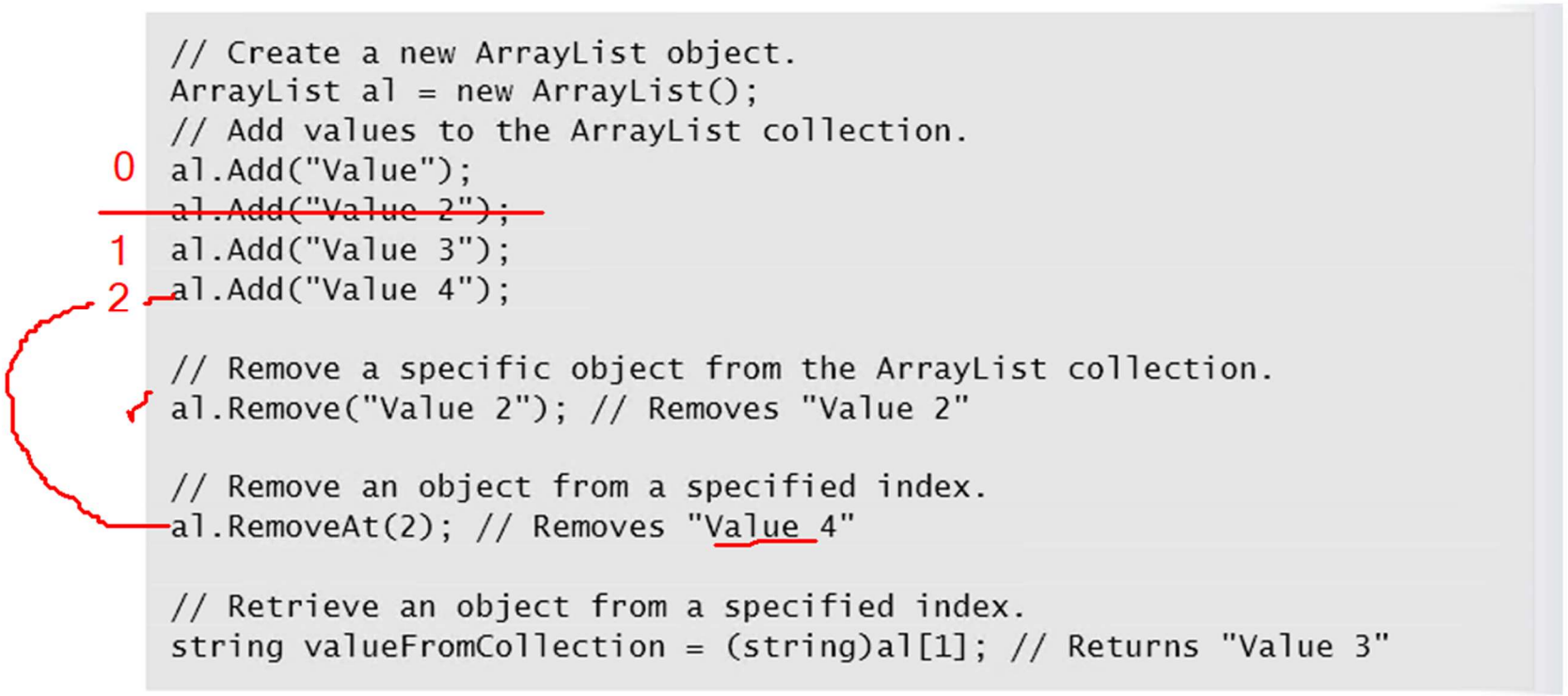
```
public int this[int index]
{
   get { return this.beverages[index]; }
   set { this.beverages[index] = value; }
}
```

- Use the instance name to interact with the indexer

```
Menu myMenu = new Menu();
string firstDrink = myMenu[0];
```

MODULE 3.

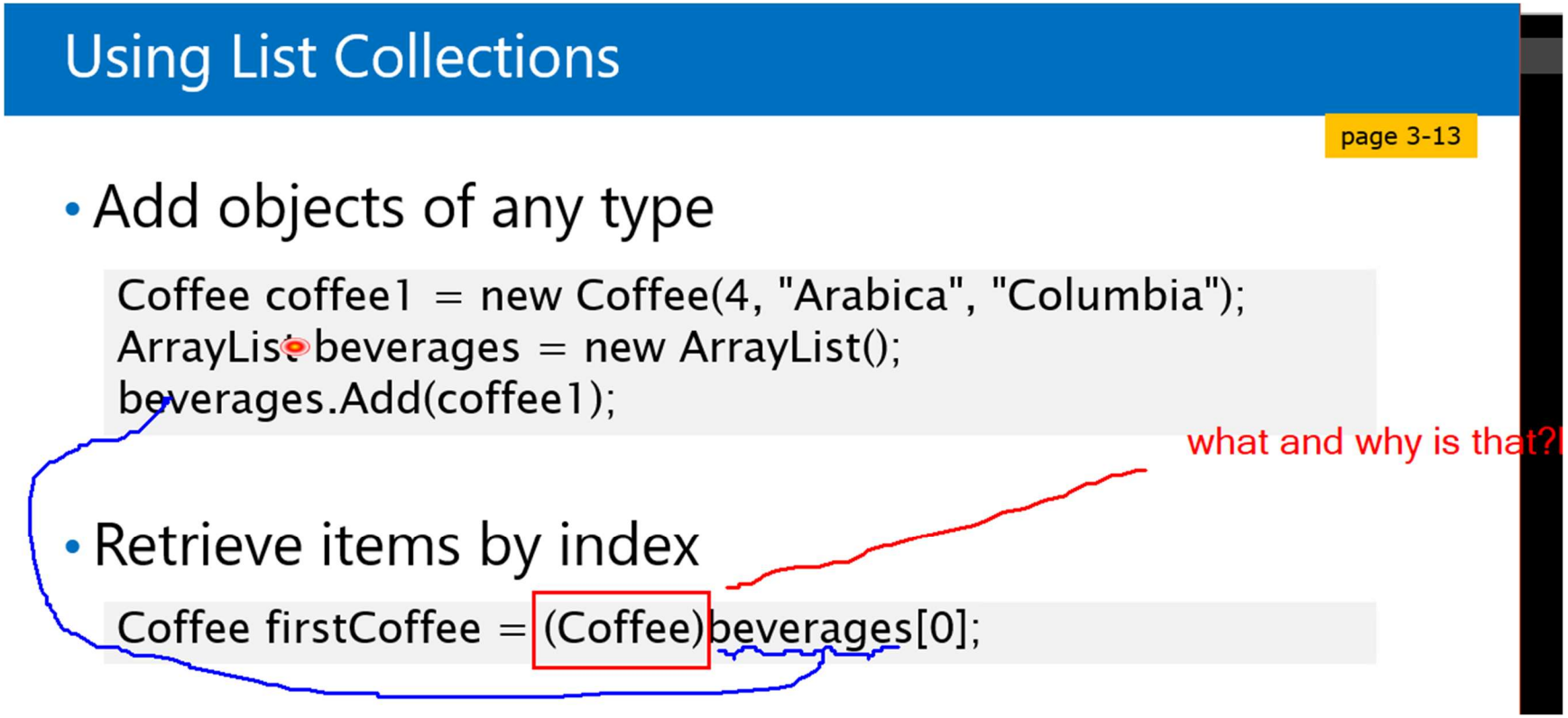
From your notes. A simple arraylist.



```
// Create a new ArrayList object.
ArrayList al = new ArrayList();
// Add values to the ArrayList collection.
0 al.Add("Value");
al.Add("Value 2");
1 al.Add("Value 3");
2 al.Add("Value 4");

// Remove a specific object from the ArrayList collection.
al.Remove("Value 2"); // Removes "Value 2"

// Remove an object from a specified index.
al.RemoveAt(2); // Removes "Value 4"

// Retrieve an object from a specified index.
string valueFromCollection = (string)al[1]; // Returns "Value 3"
```

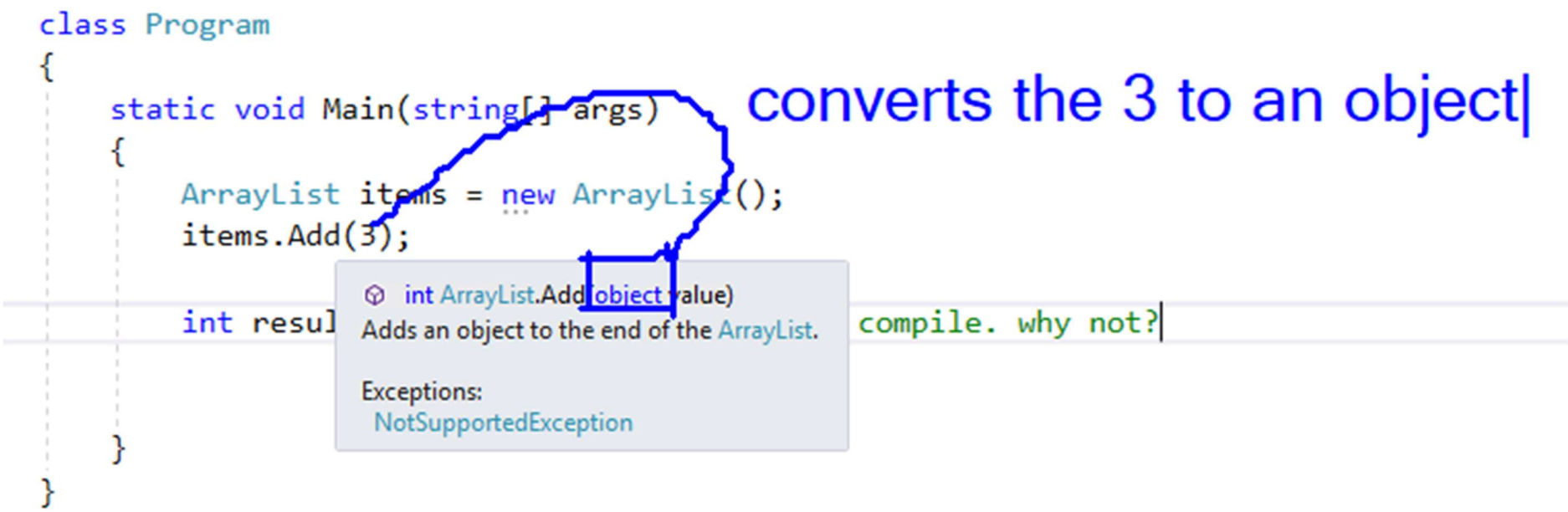
MODULE 3.

Every element in an *ArrayList* has the data type *object*. When you retrieve an item from an arraylist it is returned as an object. That means you need to *Cast (Convert)* it to the Data type of the receiving variable.



MODULE 3.

Every element in an *ArrayList* has the data type *object*. When you retrieve an item from an arraylist it is returned as an object. That means you need to *Cast (Convert)* it to the Data type of the receiving variable.



MODULE 3.

- Create a delegate for the event

```
public delegate void OutOfBeansHandler(Coffee coffee,
EventArgs args);
```

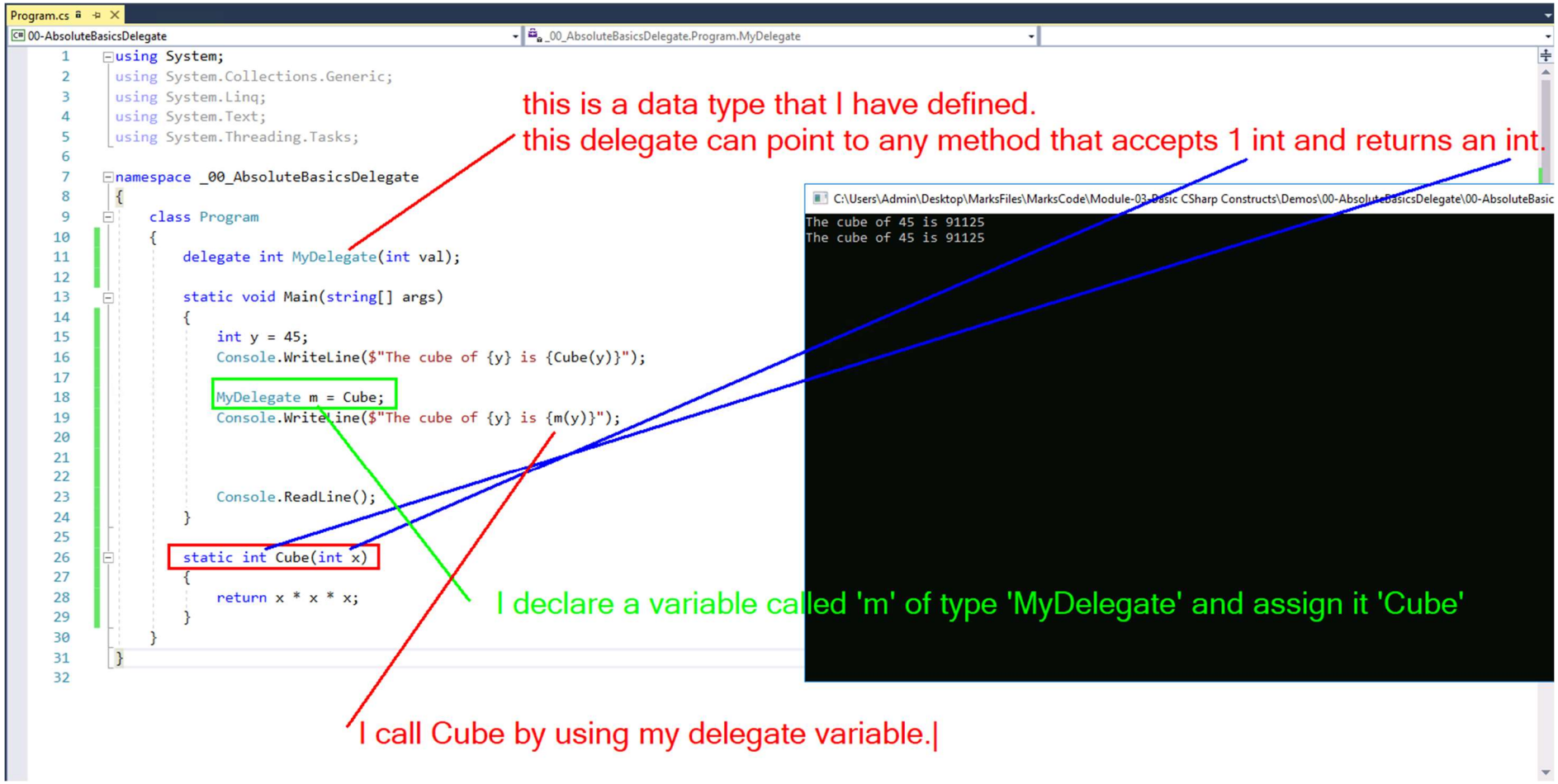
- Create the event and specify the delegate

```
public event OutOfBeansHandler OutOfBeans;
```

A delegate is a data type that can point to a function.|

MODULE 3.

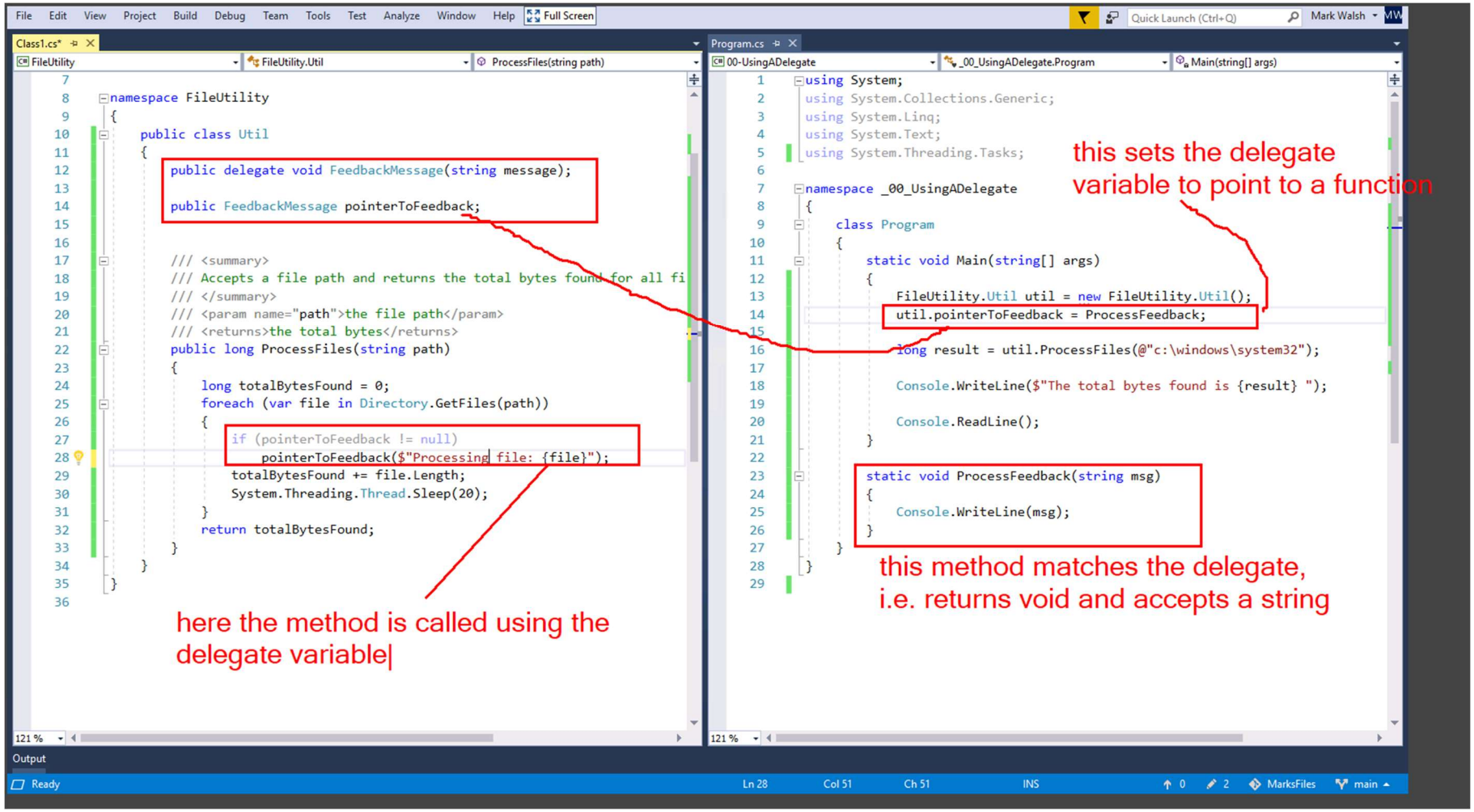
Matches my demo 00g-AbsoluteBasicsDelegate. A delegate is a data type that can point to a method.




```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace _00_AbsoluteBasicsDelegate
8  {
9      class Program
10     {
11         delegate int MyDelegate(int val);
12
13         static void Main(string[] args)
14         {
15             int y = 45;
16             Console.WriteLine($"The cube of {y} is {Cube(y)}");
17
18             MyDelegate m = Cube;
19             Console.WriteLine($"The cube of {y} is {m(y)}");
20
21
22
23             Console.ReadLine();
24         }
25
26         static int Cube(int x)
27         {
28             return x * x * x;
29         }
30     }
31 }
32
```

MODULE 3.

Matches my demo ***00j-UsingADelegate.*** The reason for implementing a delegate in this example is so the console application can get feedback while the *ProcessFiles* method is executing.



MODULE 3.

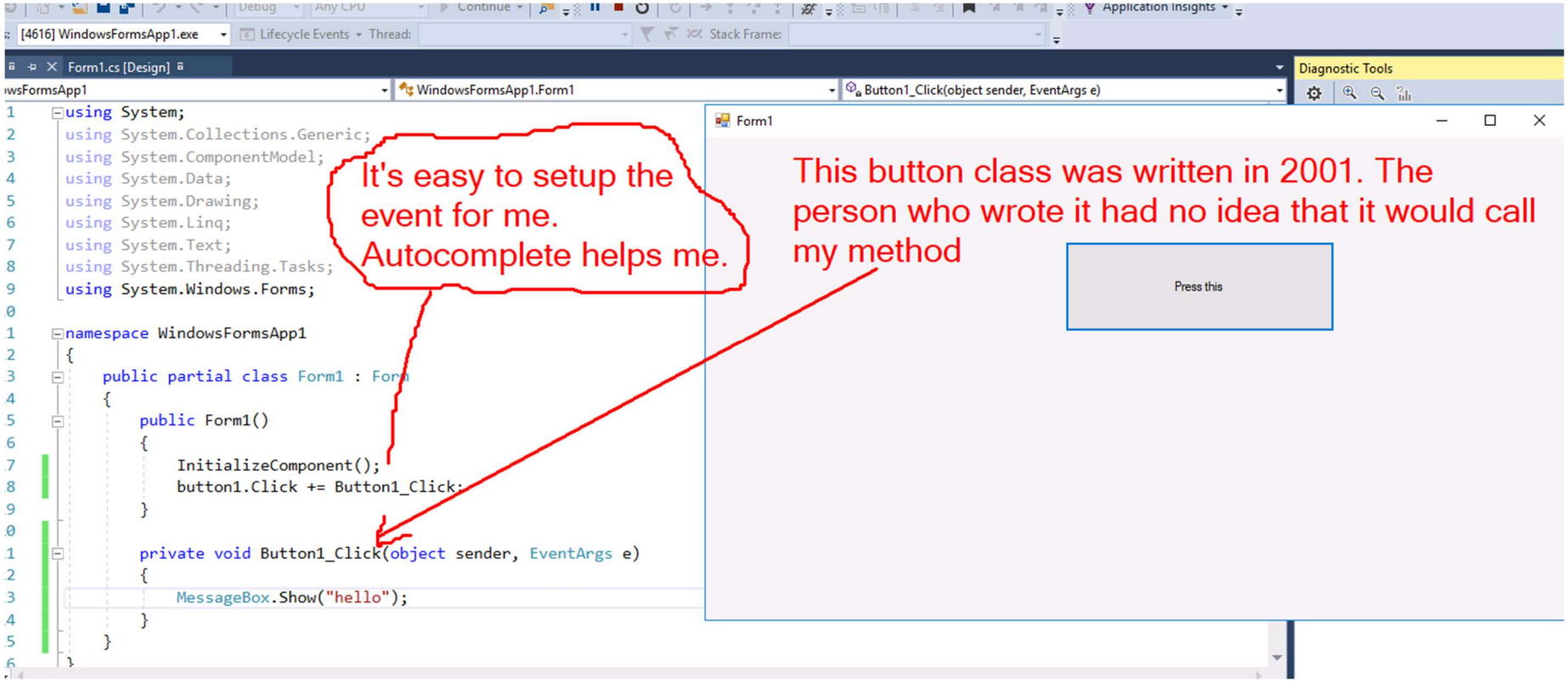
Events extend the idea of delegates and make things easier for the caller of a struct or class. Line 13 (left) is the extra key line I added to define an event. Then, line 14 (right) connects the event to a method. When line 29 (left) runs the method at line 24 (right) will execute.

```
12          public delegate void FeedbackMessage(string message);
13          public event FeedbackMessage FeedbackEvent;
14
15          public FeedbackMessage pointerToFeedback;
16
17
18          /// <summary>
19          /// Accepts a file path and returns the total bytes found for all fi
20          /// </summary>
21          /// <param name="path">the file path</param>
22          /// <returns>the total bytes</returns>
23          public long ProcessFiles(string path)
24          {
25              long totalBytesFound = 0;
26              foreach (var file in Directory.GetFiles(path))
27              {
28                  if (FeedbackEvent != null)
29                      FeedbackEvent($"Processing file: {file}");
30                  totalBytesFound += file.Length;
31                  System.Threading.Thread.Sleep(20);
32              }
33              return totalBytesFound;
34          }
35      }
36  }
37
```

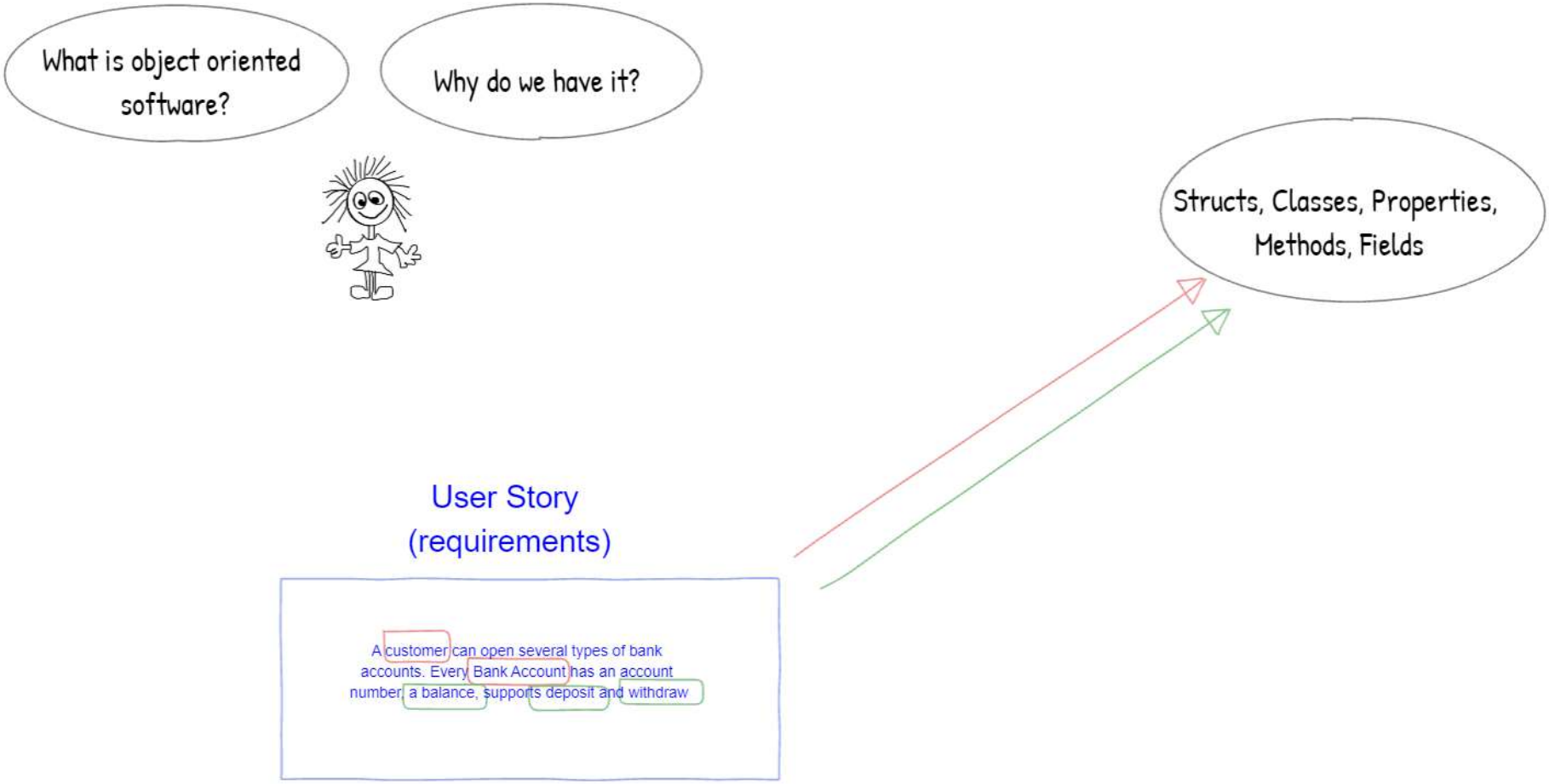
```
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace _00_FromDelegatestoEvents
8   {
9       class Program
10      {
11          static void Main(string[] args)
12          {
13              FileUtility.Util util = new FileUtility.Util();
14              util.FeedbackEvent += Util_FeedbackEvent;
15
16
17              long result = util.ProcessFiles(@"c:\windows\system32");
18
19              Console.WriteLine($"The total bytes found is {result} ");
20
21              Console.ReadLine();
22          }
23
24          private static void Util_FeedbackEvent(string message)
25          {
26              Console.WriteLine(message);
27          }
28      }
29  }
```

MODULE 3.

I included this example to show how a delegate and event was used in practice

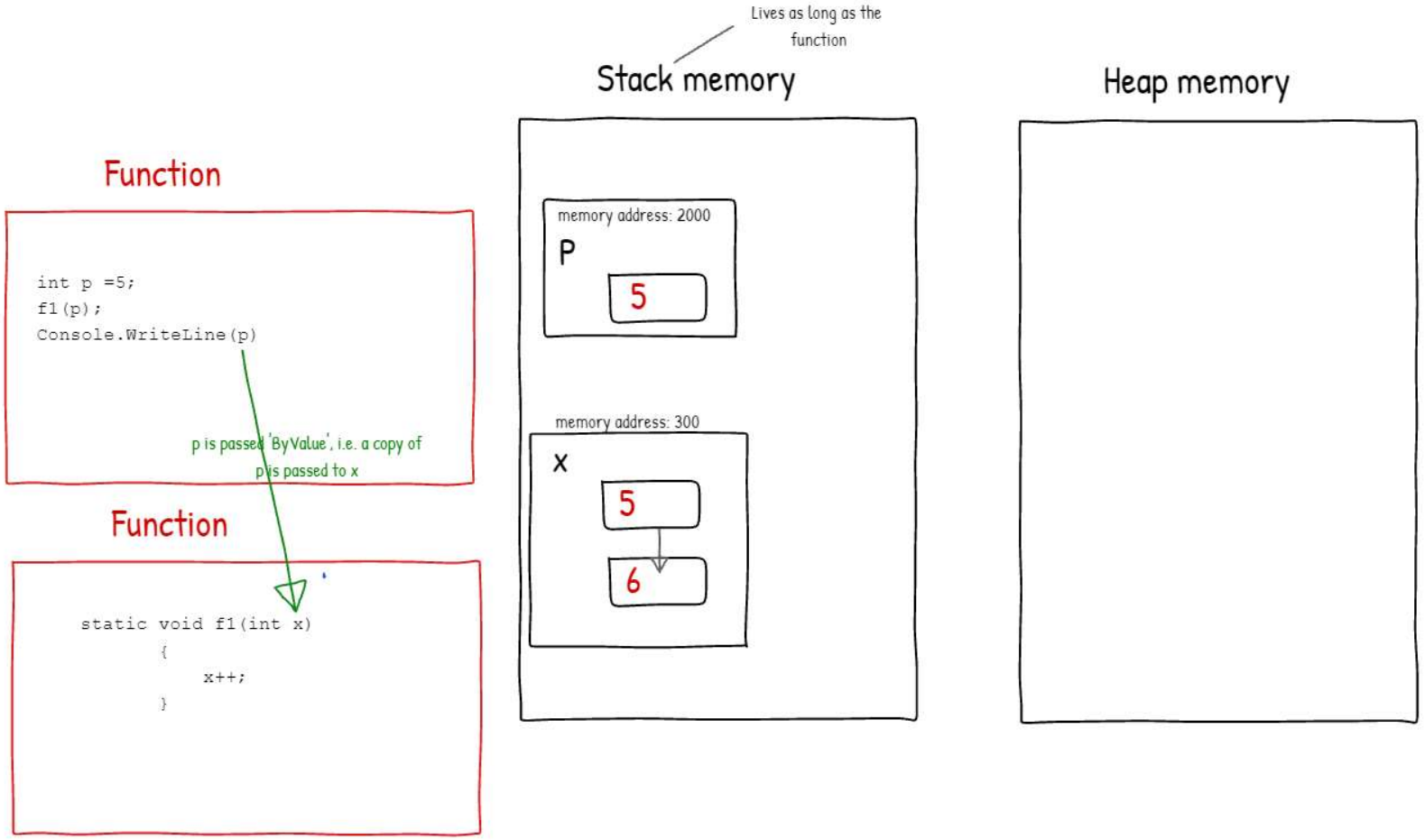


MODULE 4


What is object oriented software?
Why do we have it?
Structs, Classes, Properties, Methods, Fields
User Story (requirements)
A customer can open several types of bank accounts. Every Bank Account has an account number, a balance, supports deposit and withdraw

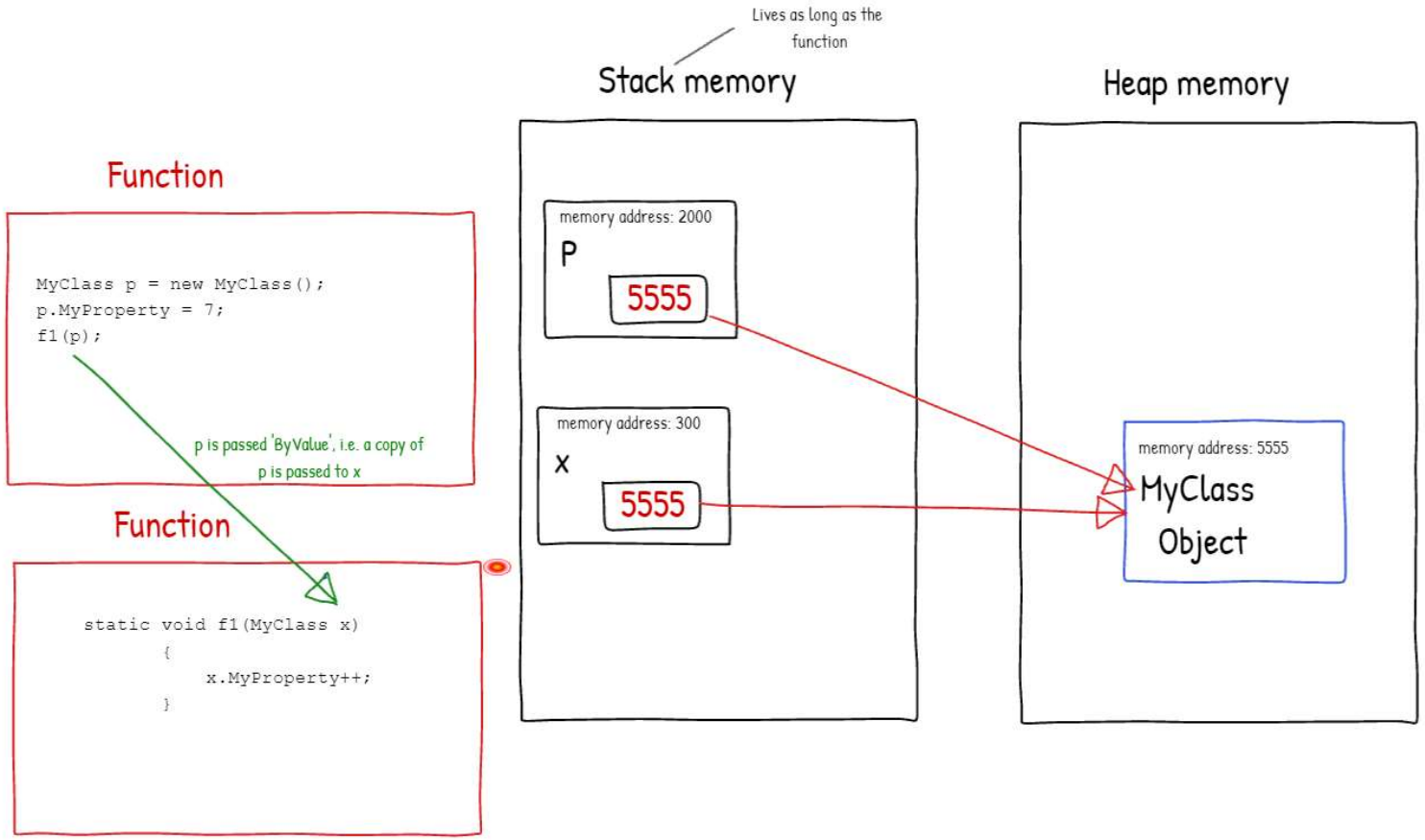
MODULE 4.

When you pass parameters to functions they are passed ***ByValue*** by default. That means a copy of the stack variable is passed to the parameter of the receiving function. If the receiving function changes the parameter it doesn't affect the stack variable that's passed. Note that in this example the datatype being used is an int which is a Struct.



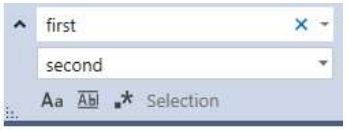
MODULE 4.

When you pass parameters to functions they are passed ***ByValue*** by default. That means a copy of the stack variable is passed to the parameter of the receiving function. If the receiving function changes the parameter it doesn't affect the stack variable that's passed. Note that in this example the datatype being used is ***MyClass*** which is a **Class**. You want to be extra careful here, the function ***f1*** does not change the parameter ***x,*** **but does change a property of the object that x points to, which is the same object that p points to.**



MODULE 4.

Interfaces are contracts. They describe a set of functionality that a class guarantees to implement. In this example there is an interface that two bank account classes implement which includes a deposit and withdraw function.




```
        BankAccount first = new BankAccount();
        first.AccountNumber = "111";
        first.Deposit(200);


        BankAccount second = new BankAccount();
        second.AccountNumber = "222";
        second.Deposit(100);

        Transfer(first, second,20);


        Console.WriteLine($"After, Bank Account {first.AccountNumber}, Balance {first.Balance:C}");
        Console.WriteLine($"After, Bank Account {first.AccountNumber}, Balance {first.Balance:C}");


        Console.ReadLine();
    }

    static void Transfer(IBank from, IBank To, decimal amount)
    {
        from.Withdraw(amount);
        To.Deposit(amount);
    }
}

interface IBank
{
    void Deposit(decimal amount);
    void Withdraw(decimal amount);
}

class BankAccount : IBank
{
    public string AccountNumber { get; set; } //property

    private decimal _balance; //field
    public decimal Balance... //property

    public void Deposit(decimal amount) //Method
```

I can pass any objects here that implement IBank|

this class implements IBank interface

It must implement deposit and withdraw

MODULE 4.

Interfaces can be defined in a different file to the class that implements them

MODULE 4.

From my slides about generics. The code in the centre of the page below doesn't compile. Why not?

Generics DEMO 6

Generics and Constraints

What's wrong here?

```
string a="zz";
string b="gg";

string c = a * b;
```

string does not support this operator

```
class Testerr<T>
{
    public T test(T value)
    {
        return value;
    }

    public T ReturnTheBiggest(T x, T y)
    {
        if (x > y)
        {
            return x;
        }
        else
        {
            return y;
        }
    }
}
```

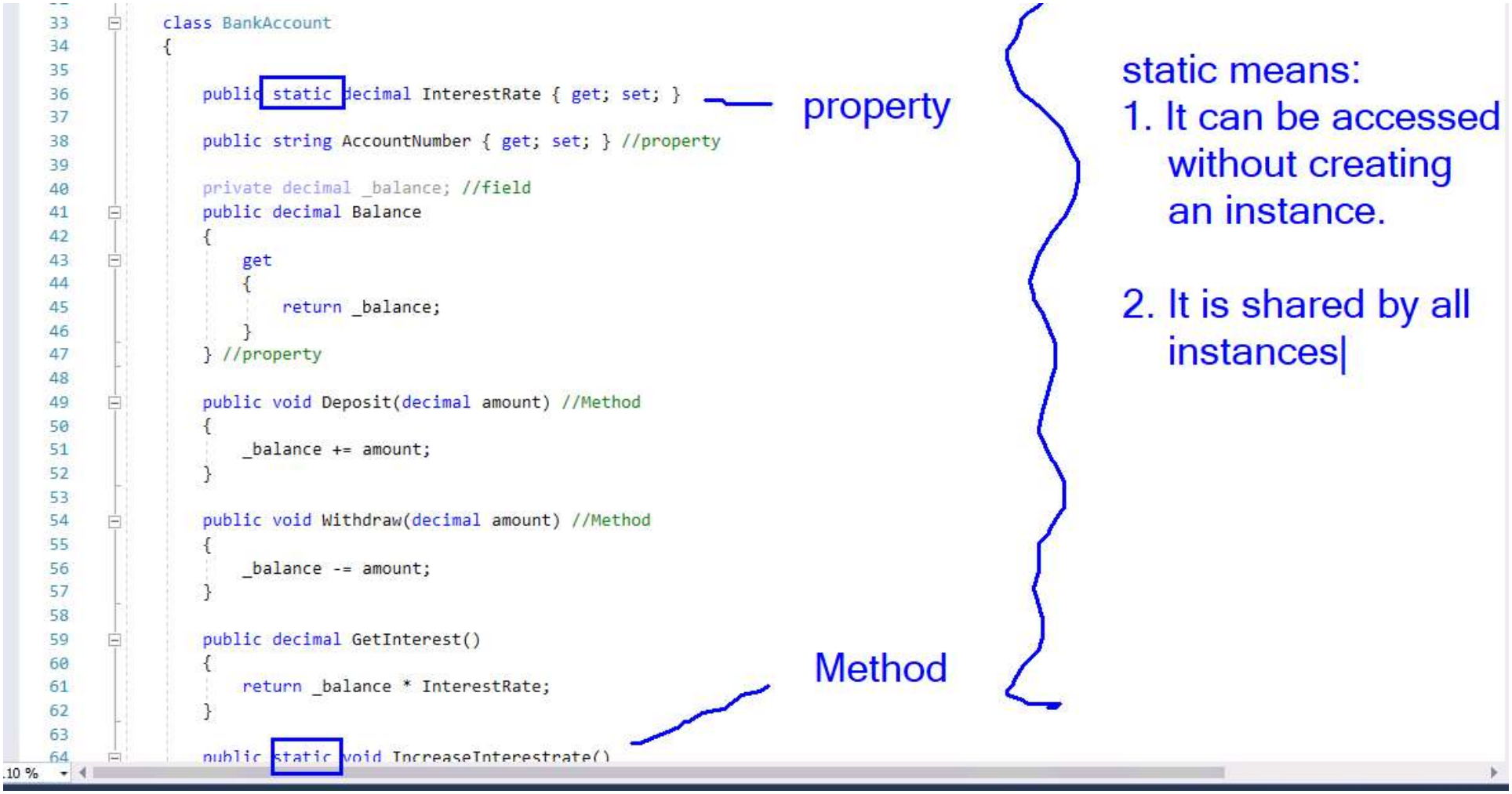
there is no guarantee that type T supports this operator.|

MODULE 4.

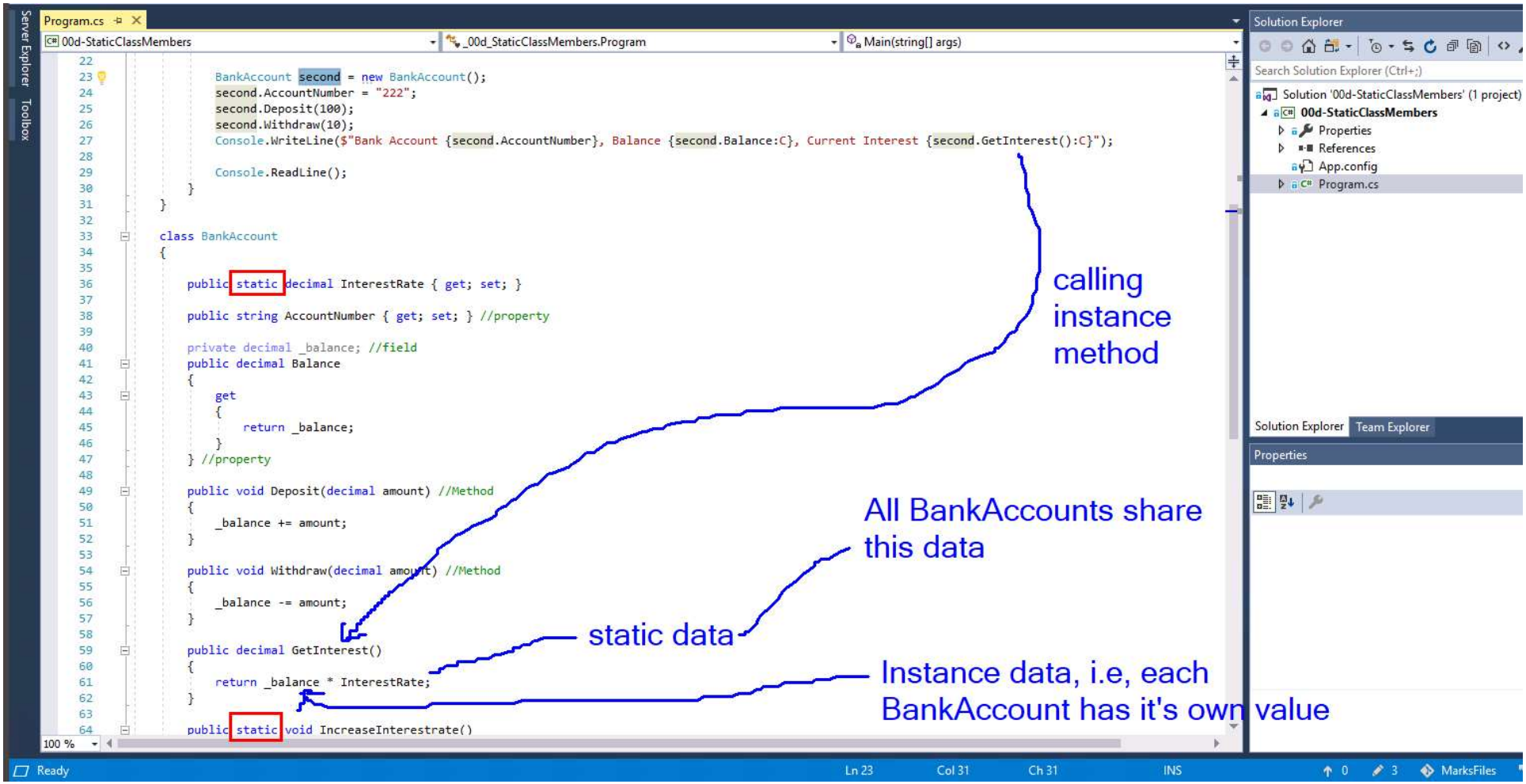
An example of using a Generic Class that is part of the .net framework

```
Dictionary<string, int> people = new Dictionary<string, int>();
people.Add("dave", 22);
people.Add("Eng", 55);
```

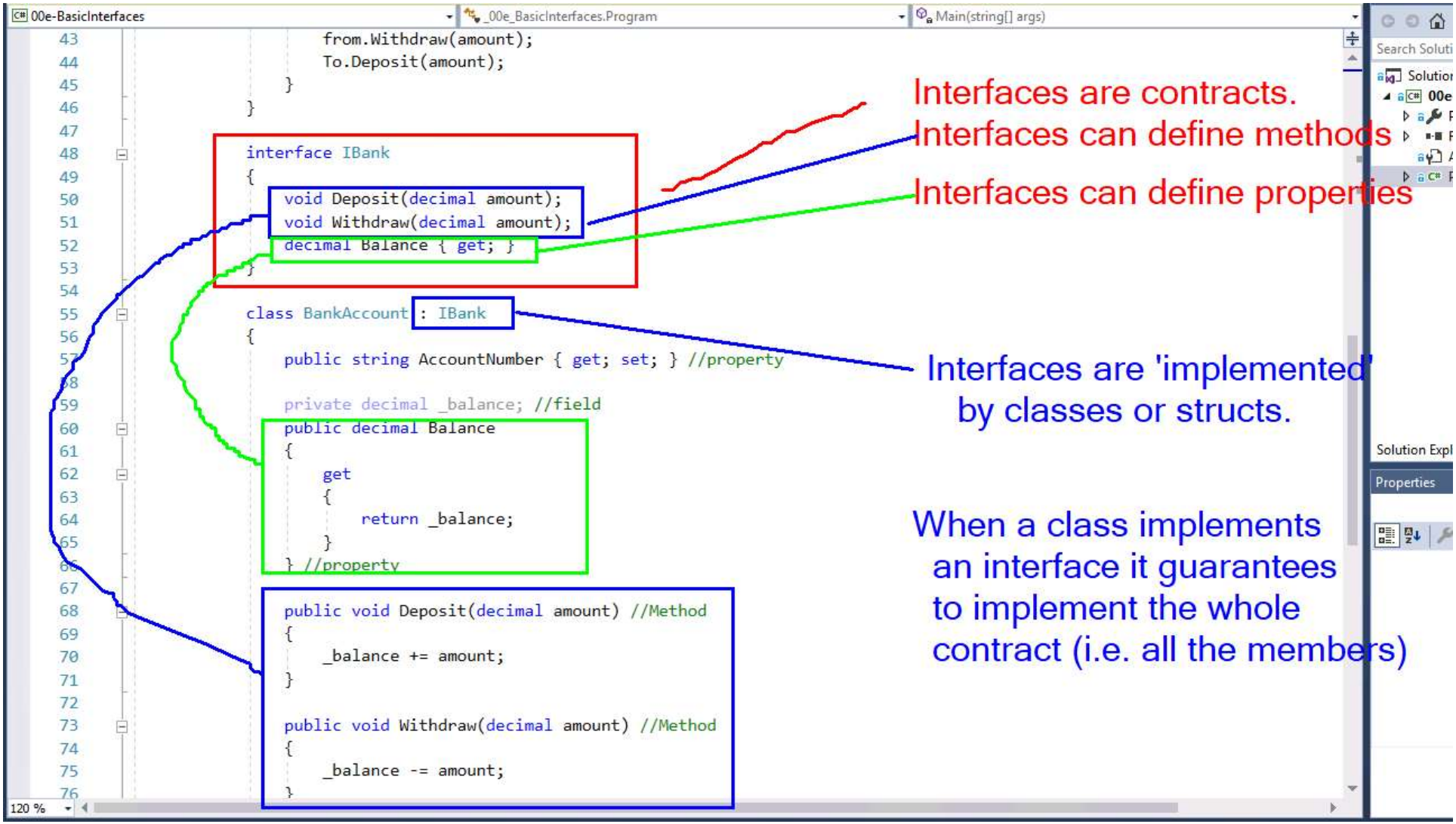
MODULE 4, review



MODULE 4, review

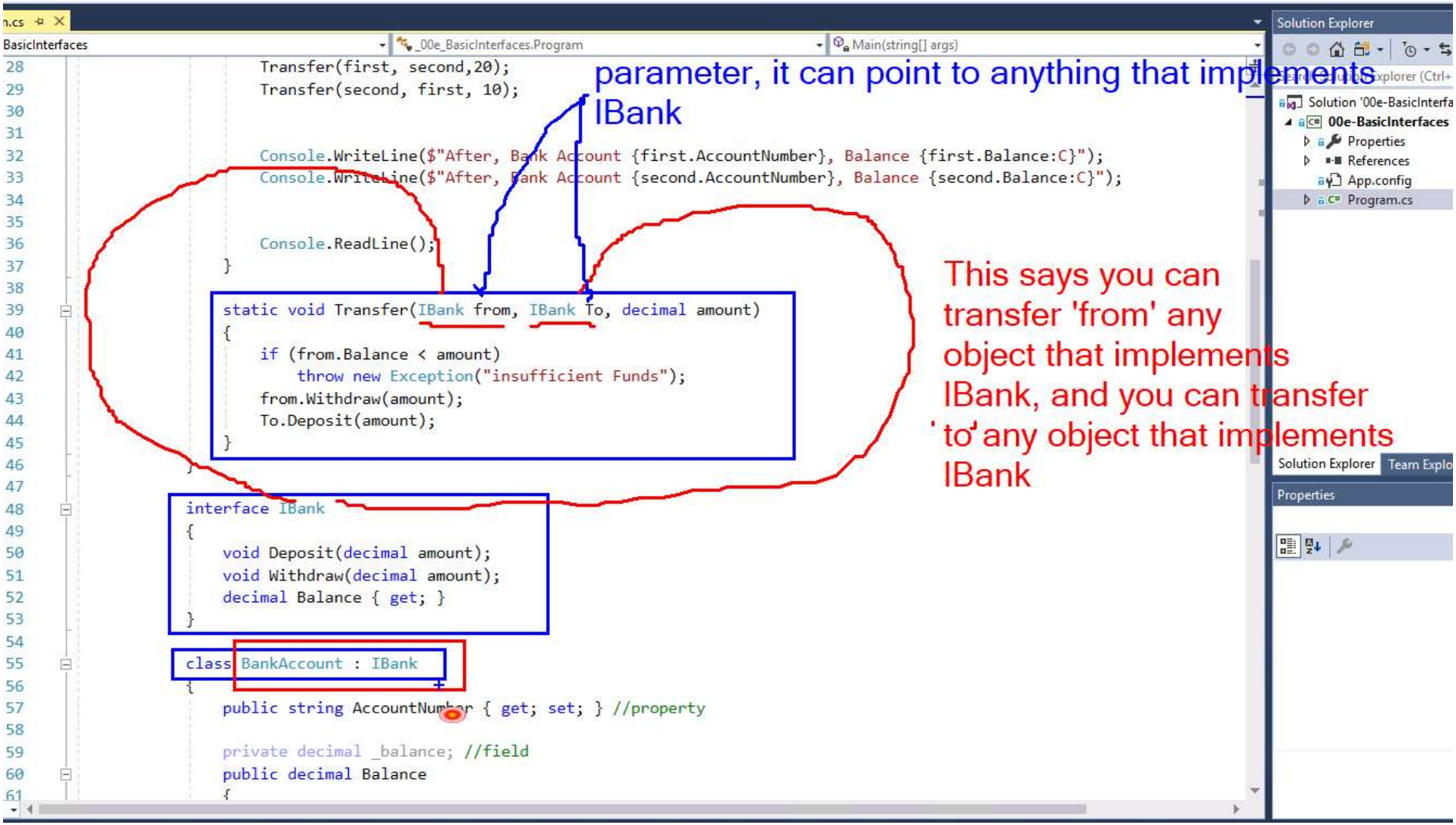


MODULE 4, review



MODULE 4, review

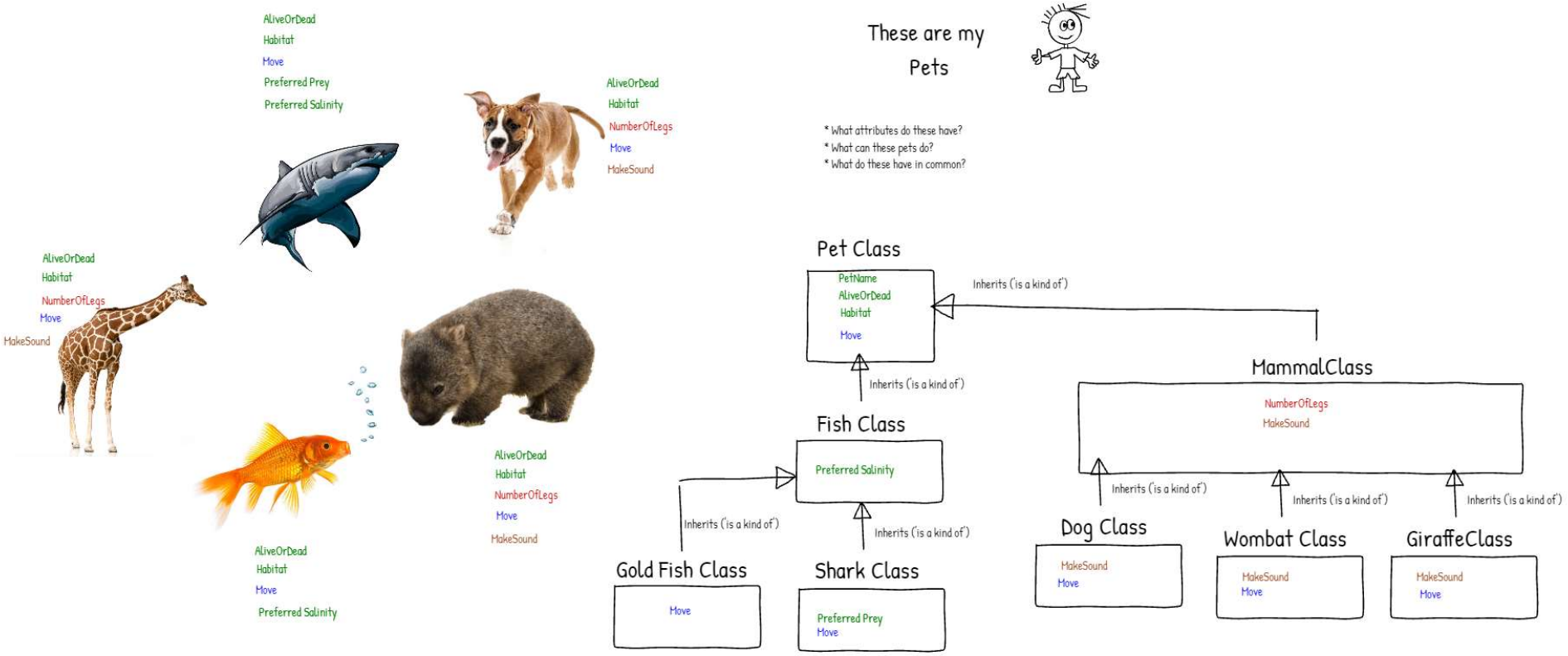
MODULE 4, review




```
BasicInterfaces | _00e_BasicInterfaces.Program | Main(string[] args)

28          Transfer(first, second,20);
29          Transfer(second, first, 10);
30
31
32          Console.WriteLine($"After, Bank Account {first.AccountNumber}, Balance {first.Balance:C}");
33          Console.WriteLine($"After, Bank Account {second.AccountNumber}, Balance {second.Balance:C}");
34
35
36          Console.ReadLine();
37      }
38
39      static void Transfer(IBank from, IBank To, decimal amount)
40      {
41          if (from.Balance < amount)
42              throw new Exception("insufficient Funds");
43          from.Withdraw(amount);
44          To.Deposit(amount);
45      }
46  }
47
48  interface IBank
49  {
50      void Deposit(decimal amount);
51      void Withdraw(decimal amount);
52      decimal Balance { get; }
53  }
54
55  class BankAccount : IBank
56  {
57      public string AccountNumber { get; set; } //property
58
59      private decimal _balance; //field
60      public decimal Balance
61      {
```

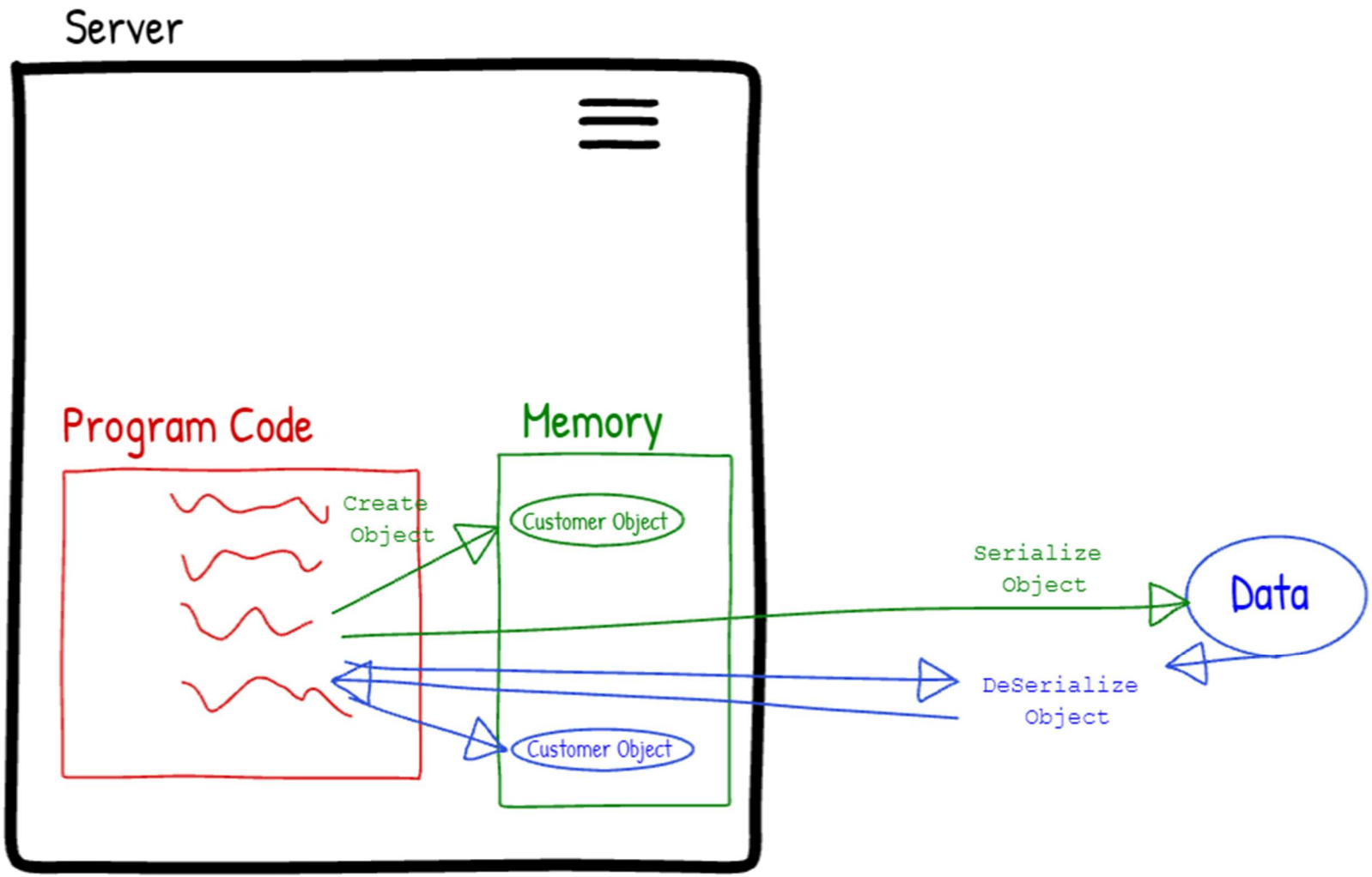
MODULE 5

This is the diagram built during class to help understand how to define class hierarchies. It matches the code that I wrote.



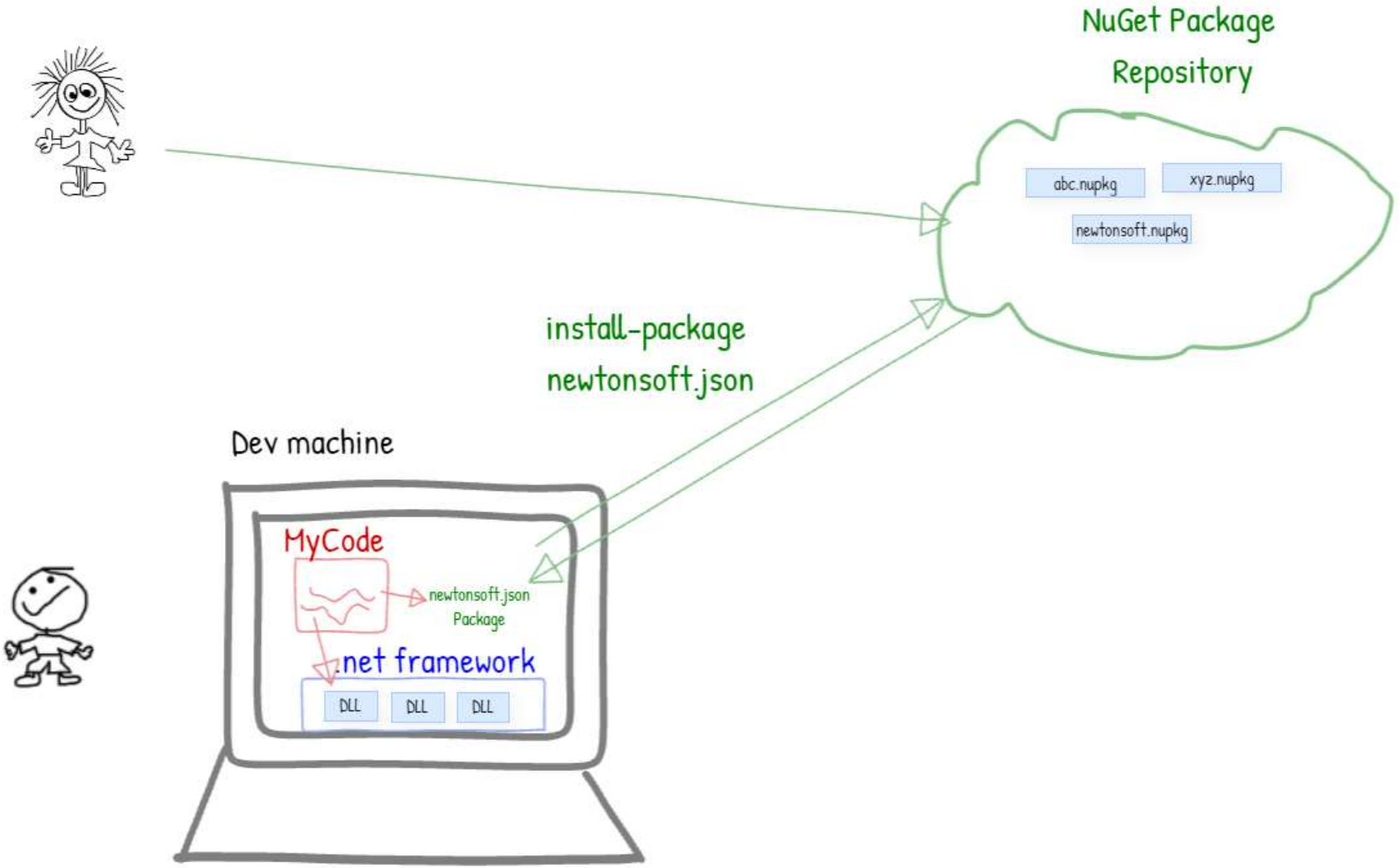
MODULE 6.

It's often a requirement to take objects in memory and then save or send them somewhere. That process is called *serialization* and the reverse process is called *deserialization*.



MODULE 6.

The lab in Module6 used the Newtonsoft.json NuGet Package

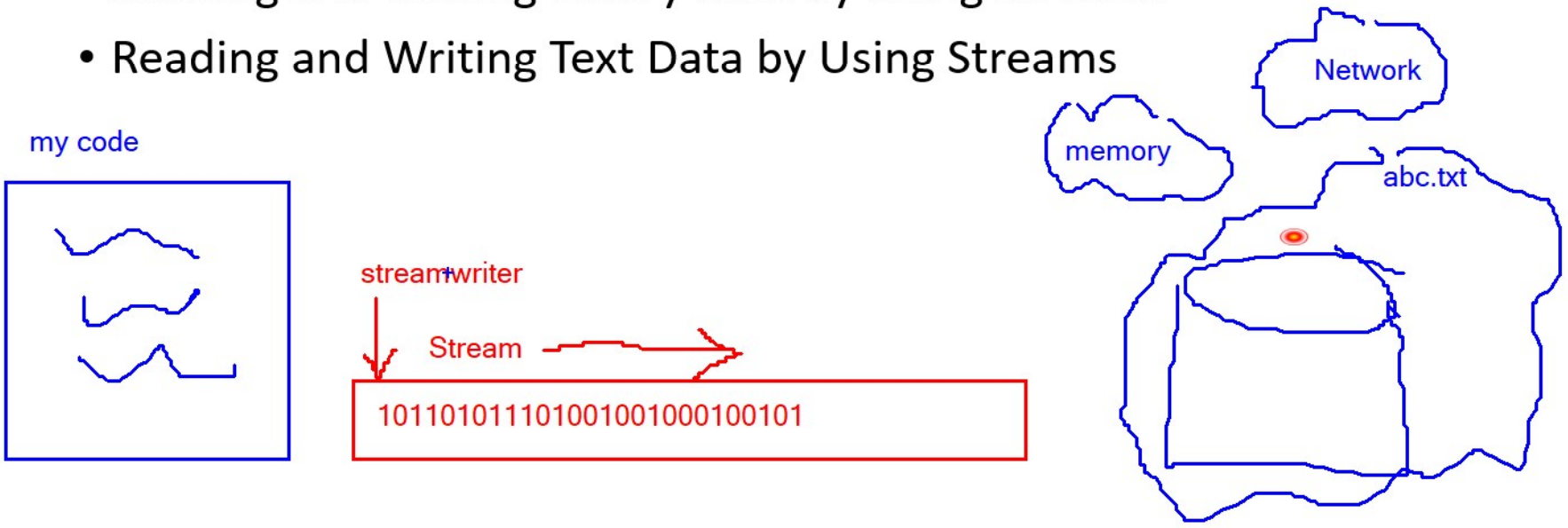


MODULE 6.

Notice *Stream* and *StreamWriter* in the diagram below

Lesson 3: Performing I/O by Using Streams

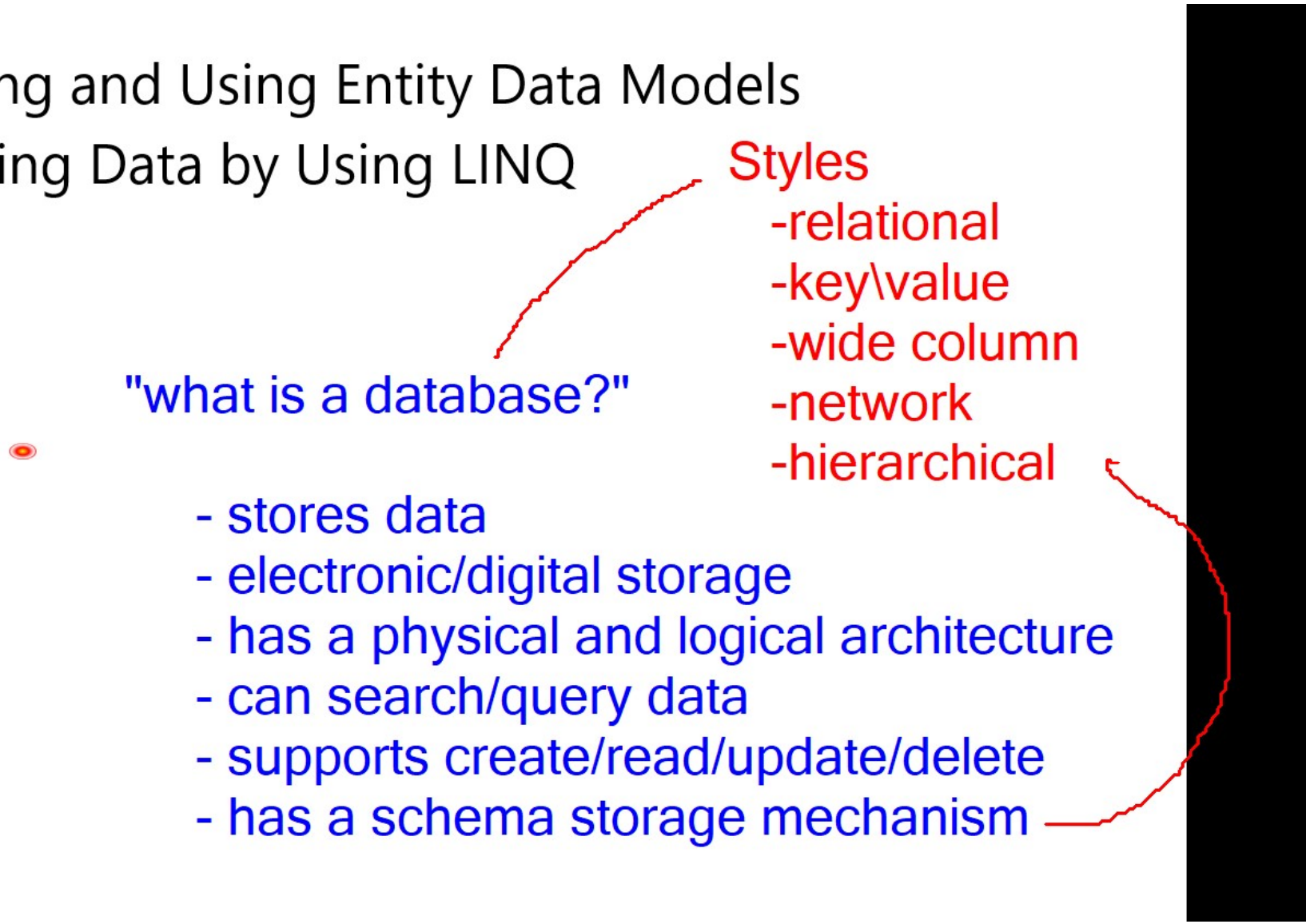
- What are Streams?
- Types of Streams in the .NET Framework
- Reading and Writing Binary Data by Using Streams
- Reading and Writing Text Data by Using Streams



MODULE 7.

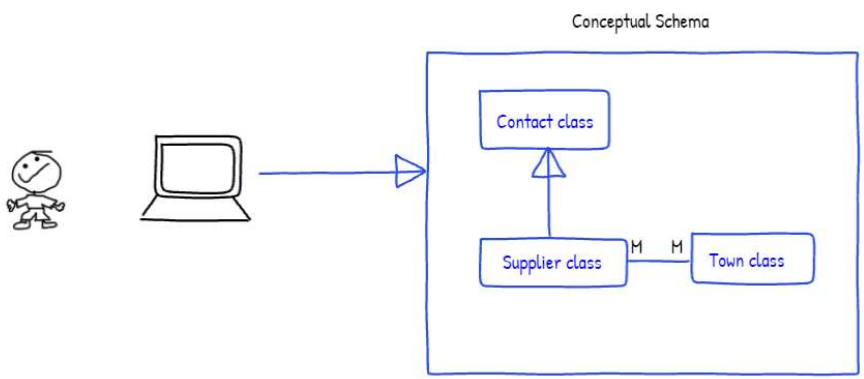
Before we started the module, it's a good idea to ask *What is a database*?

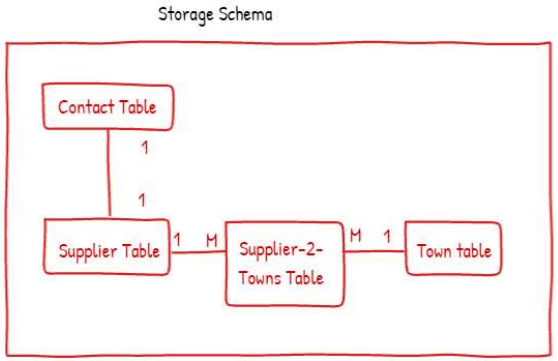
- Creating and Using Entity Data Models
- Querying Data by Using LINQ



MODULE 7.

Before we had the entity framework writing a C# application that performed CRUD operations on a database was cumbersome. Why? because we would need to write extensive amounts of code to move the data coming from a database into our C# conceptual schema (i.e. our C# classes). Relational database schema usually do not match the OO class structure we write in our code, so we'd have to write a whole heap of mapping code. The diagram below shows a database structure that does not match our C# classes. Relational databases don't support inheritance or many-to-many relationships.





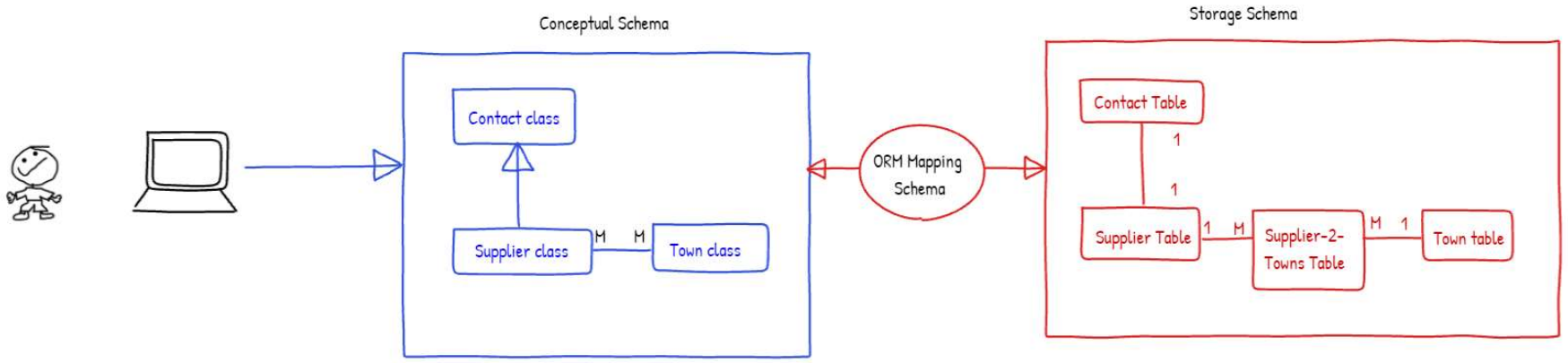
sid	s_sname
1	ACME
2	Coke

sid	tid
1	2
1	1
2	2

tid	t_name
1	Lithgow
2	Perth

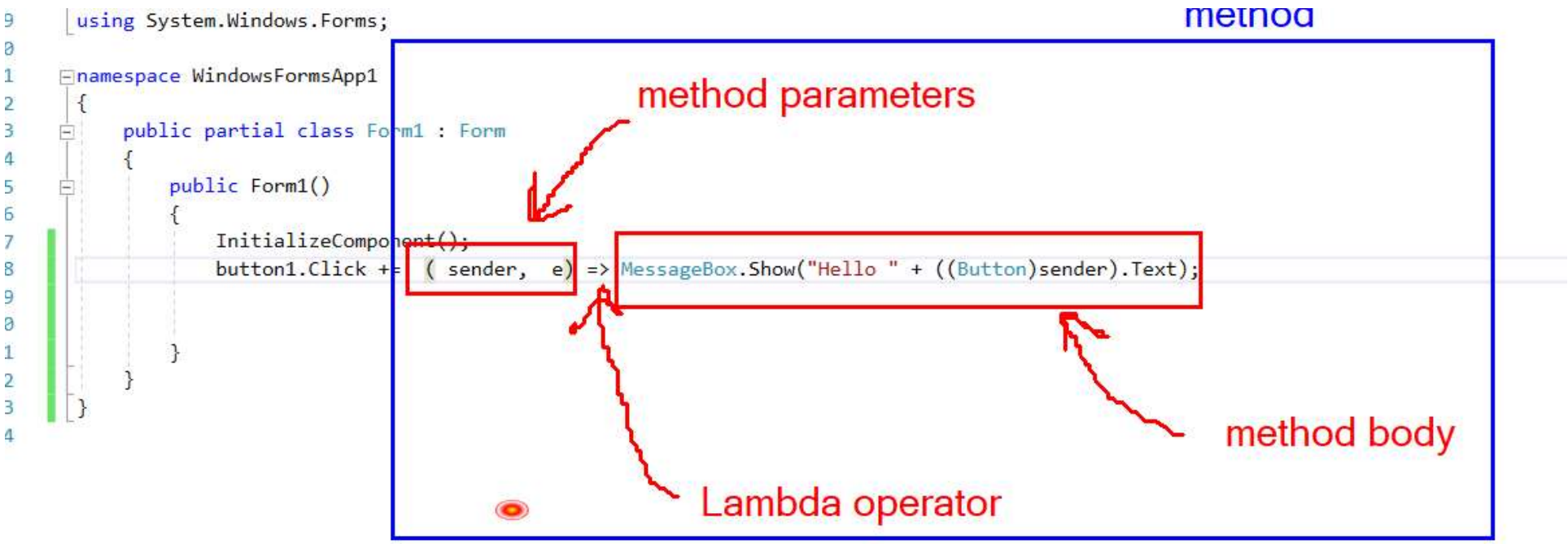
MODULE 7.

The Entity Framework provides the ORM mapping code for us so we can concentrate on writing our code against C# classes.



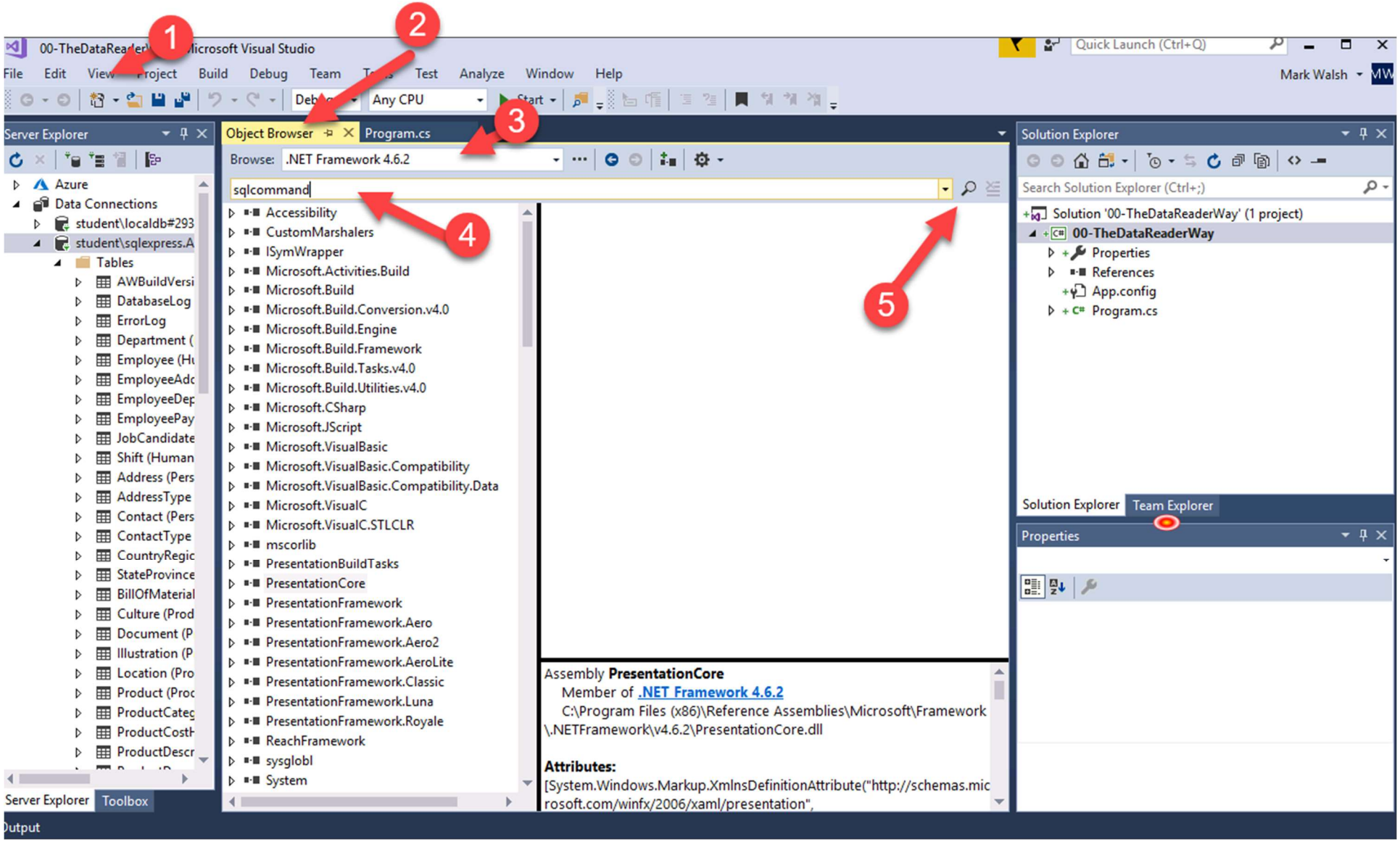
MODULE 7.

When using the Entity Framework and in many other .Net programming situations you find yourself needing to use Lambda expressions. A *Lambda* expression is simply an in-line function. Button1.Click (a property) must be assigned a delegate (a data type that points to a method). In the below code that is done with a Lambda.



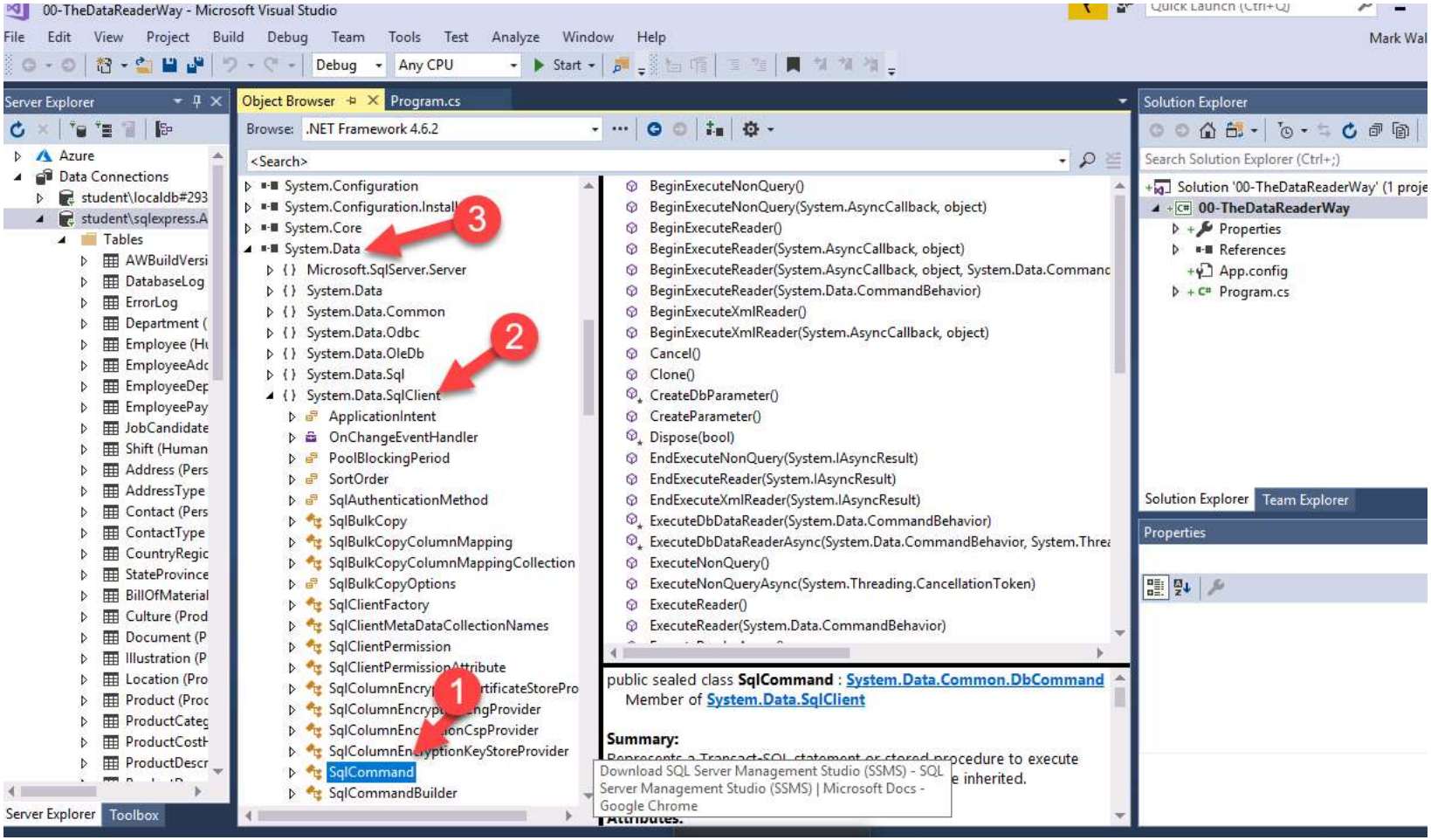
MODULE ALL

A very useful too in Visual Studio is the *Object Browser.* It's great for finding classes and structs and determining their parent namespaces and the DLL they reside in.



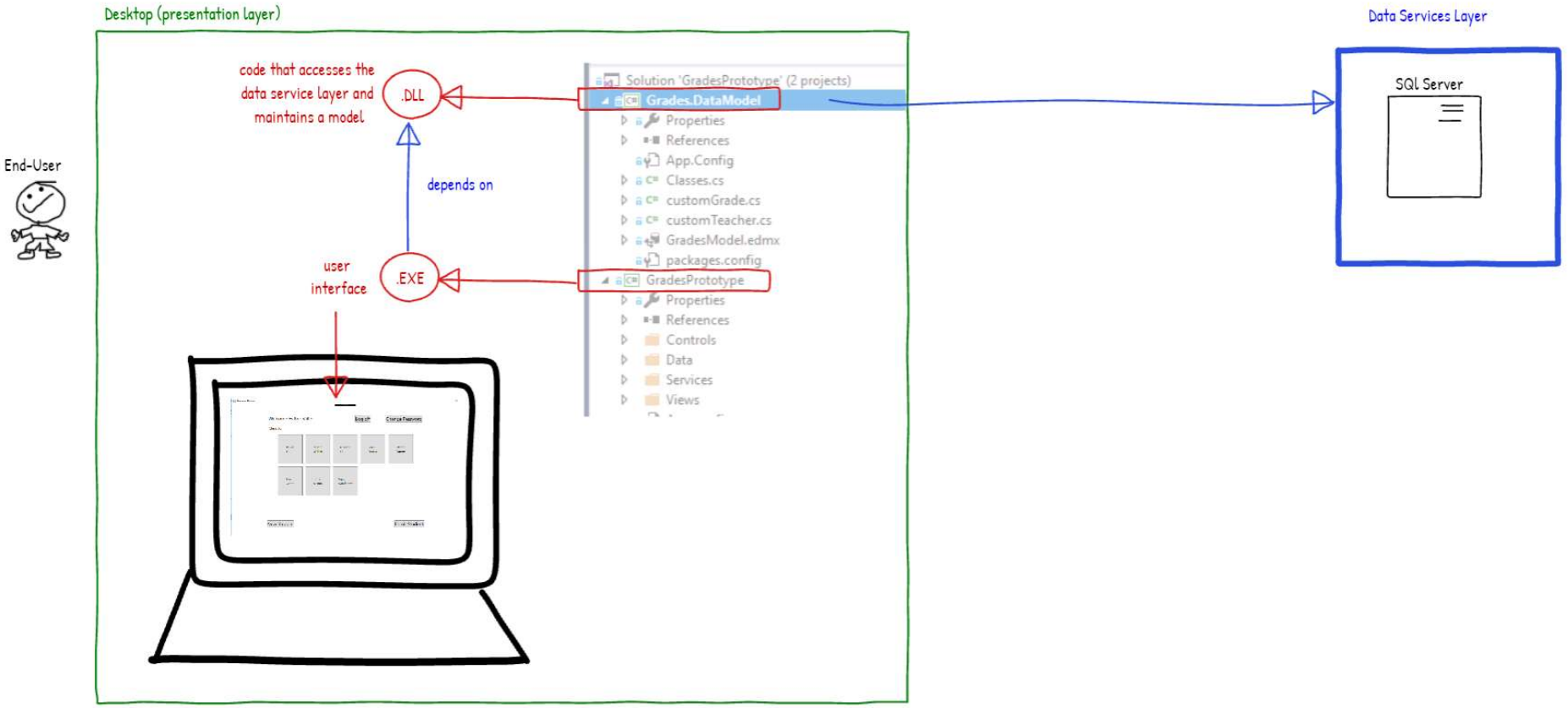
MODULE ALL

A very useful too in Visual Studio is the *Object Browser*. It's great for finding classes and structs and determining their parent namespaces and the DLL they reside in.



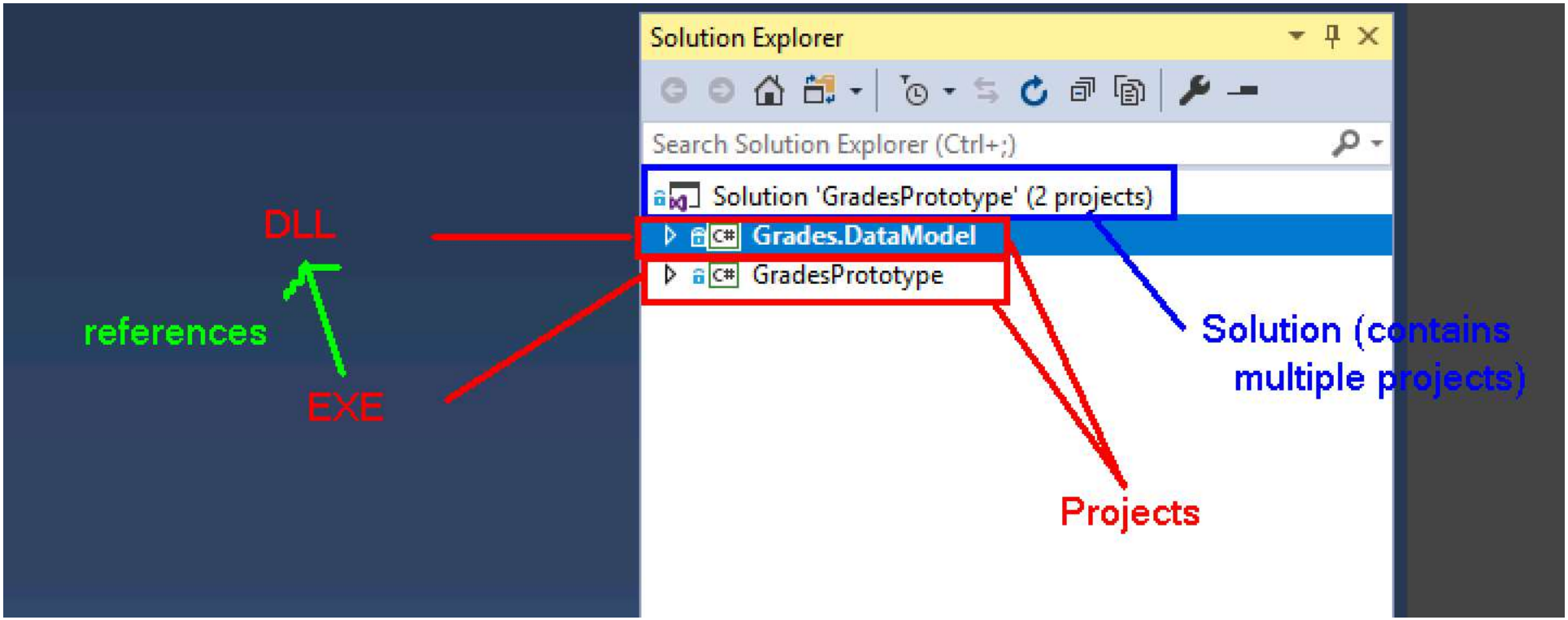
MODULE 7

The module 7 lab exercise architecture

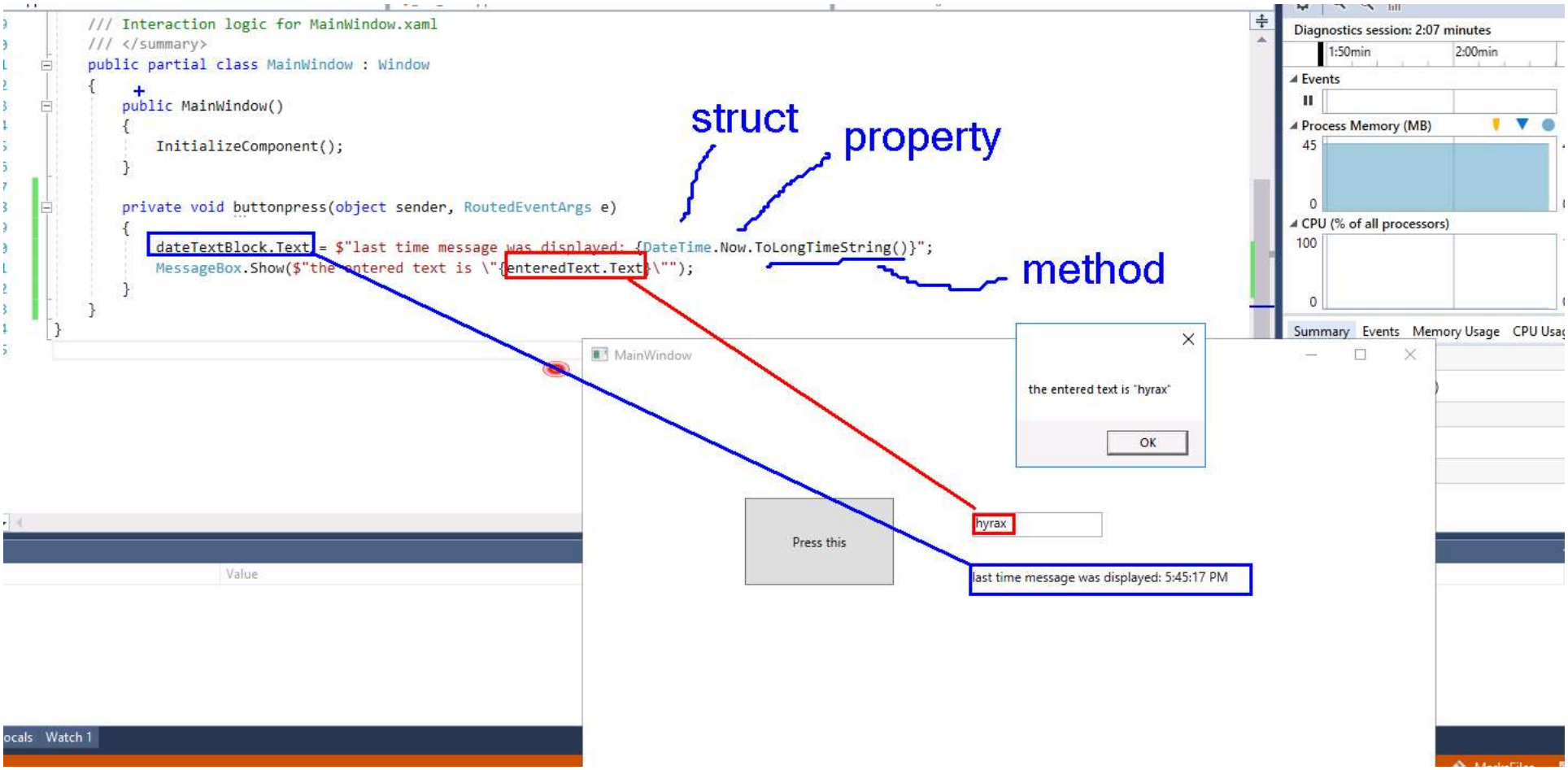


MODULE All Modules

When you work with Visual Studio, there is a concept of a *Solution* and of *Projects*. A solution is composed of one or more projects. Each project is concerned with a certain aspect of an application such as the user interface or accessing a database.

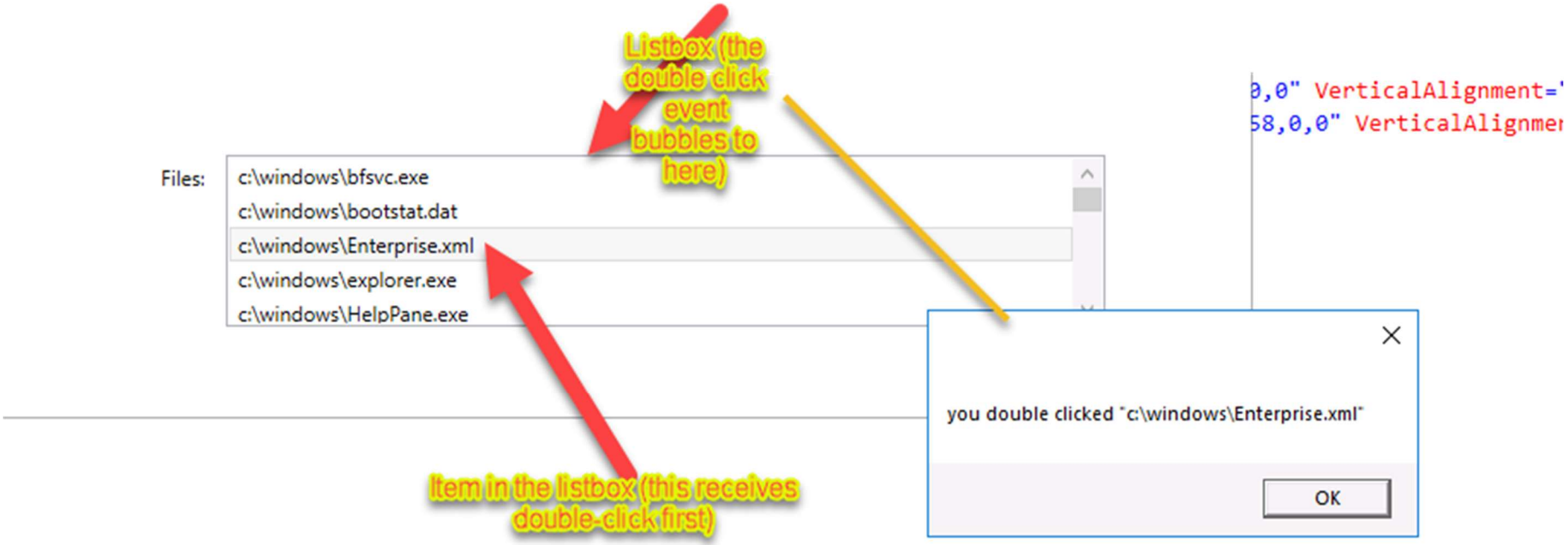


MODULE 9


/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window
{
public MainWindow()
{
InitializeComponent();
}
private void buttonpress(object sender, RoutedEventArgs e)
{
dateTextBlock.Text = $"last time message was displayed: {DateTime.Now.ToLongTimeString()}";
MessageBox.Show($"the entered text is \"{enteredText.Text}\"");
}
}
}
struct
property
method
Diagnostics session: 2:07 minutes
1:50min
2:00min
Events
Process Memory (MB)
45
0
CPU (% of all processors)
100
0
Summary
Events
Memory Usage
CPU Usag
MainWindow
the entered text is "hyrax"
OK
Press this
hyrax
last time message was displayed: 5:45:17 PM
Value
ocals
Watch 1

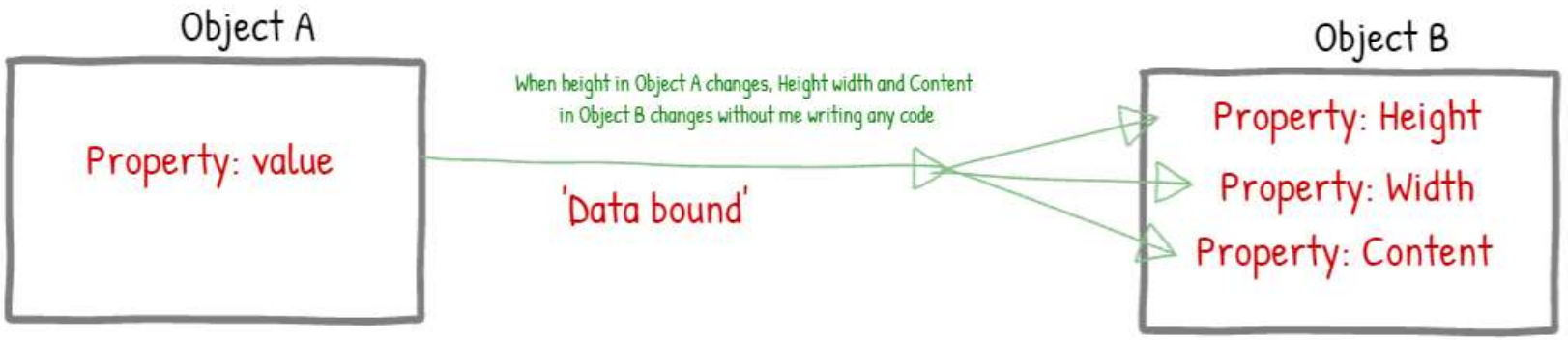
MODULE 9

Concept of *Event Bubbling* in WPF.



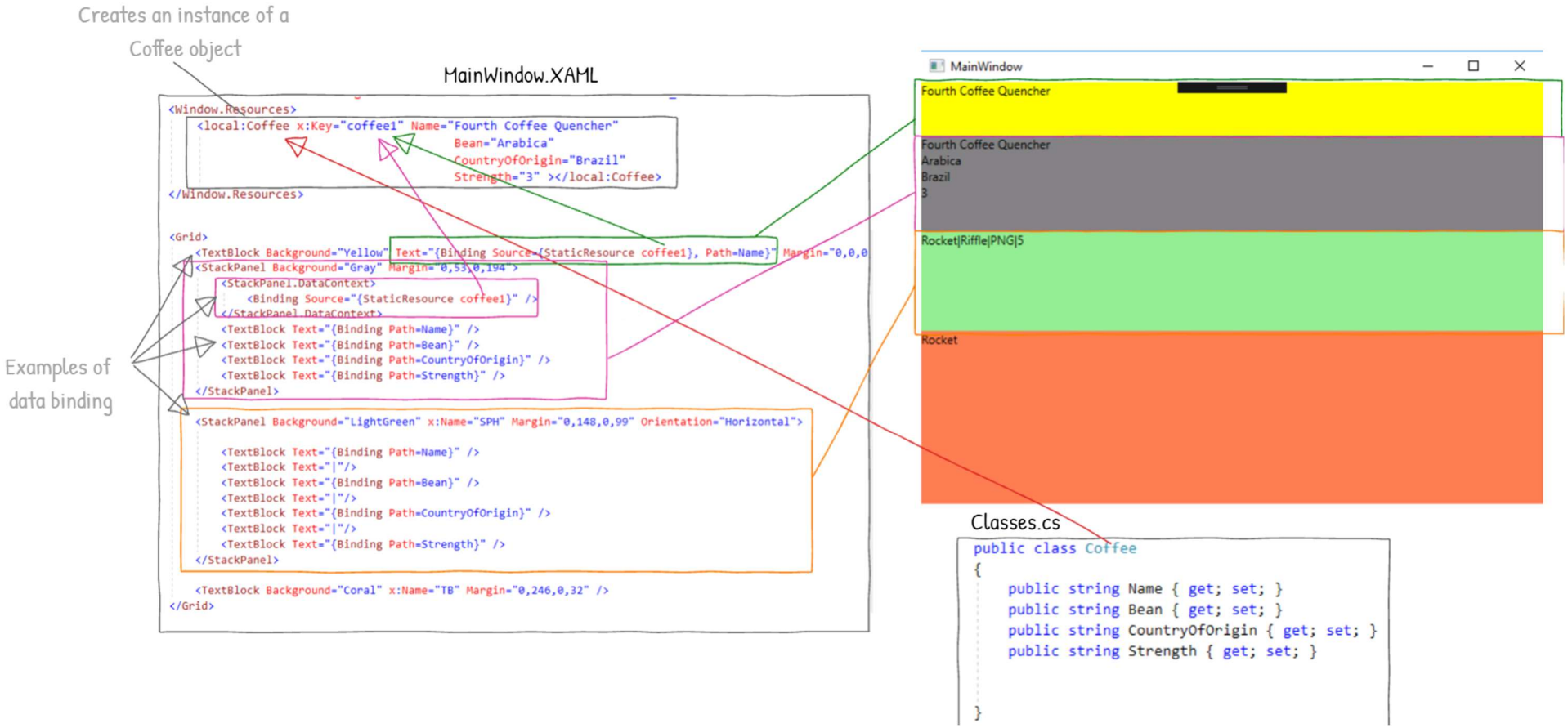
MODULE 9

The concept of *Data Binding*



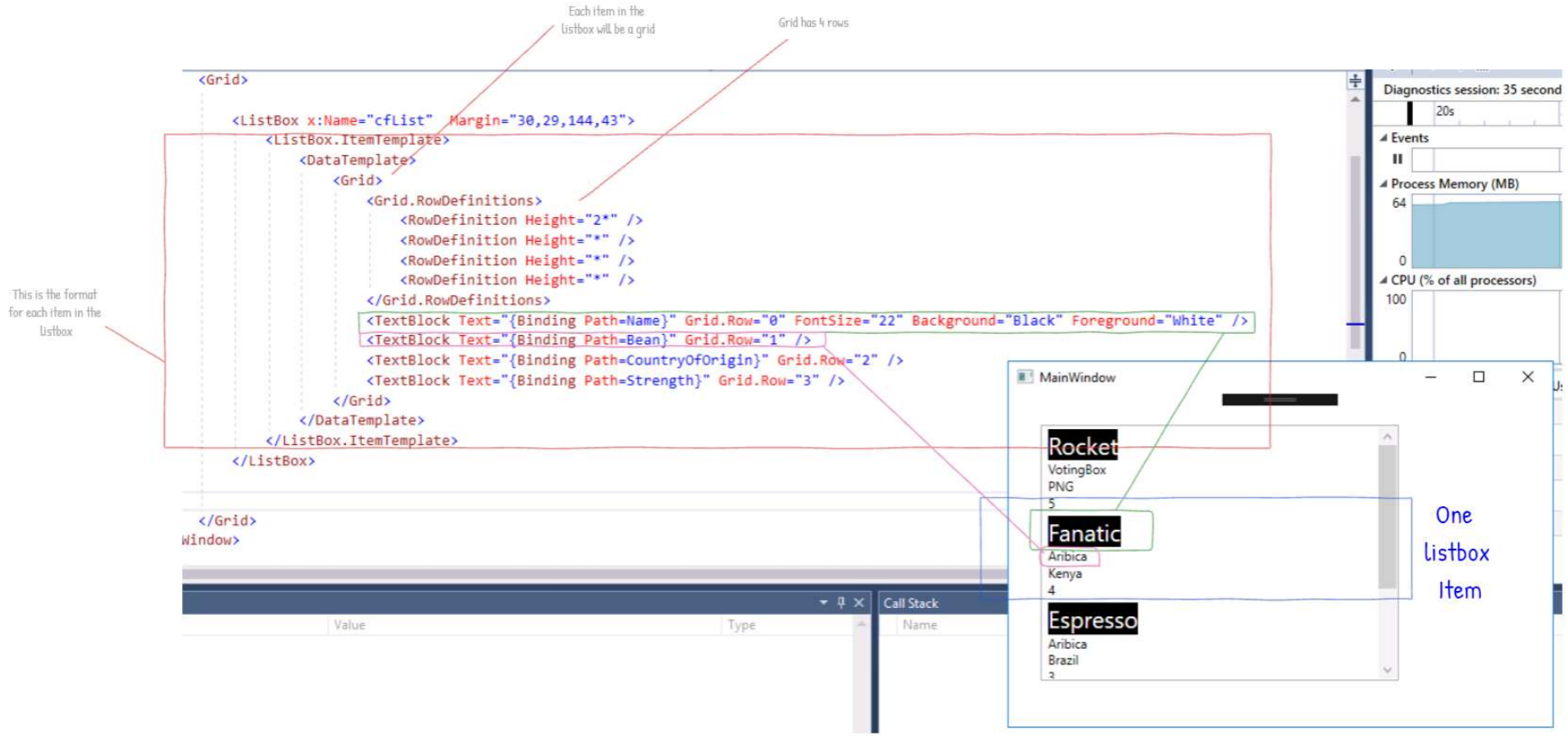
MODULE 9

A *data binding* example



MODULE 9

A data binding example using a ListBox item template.

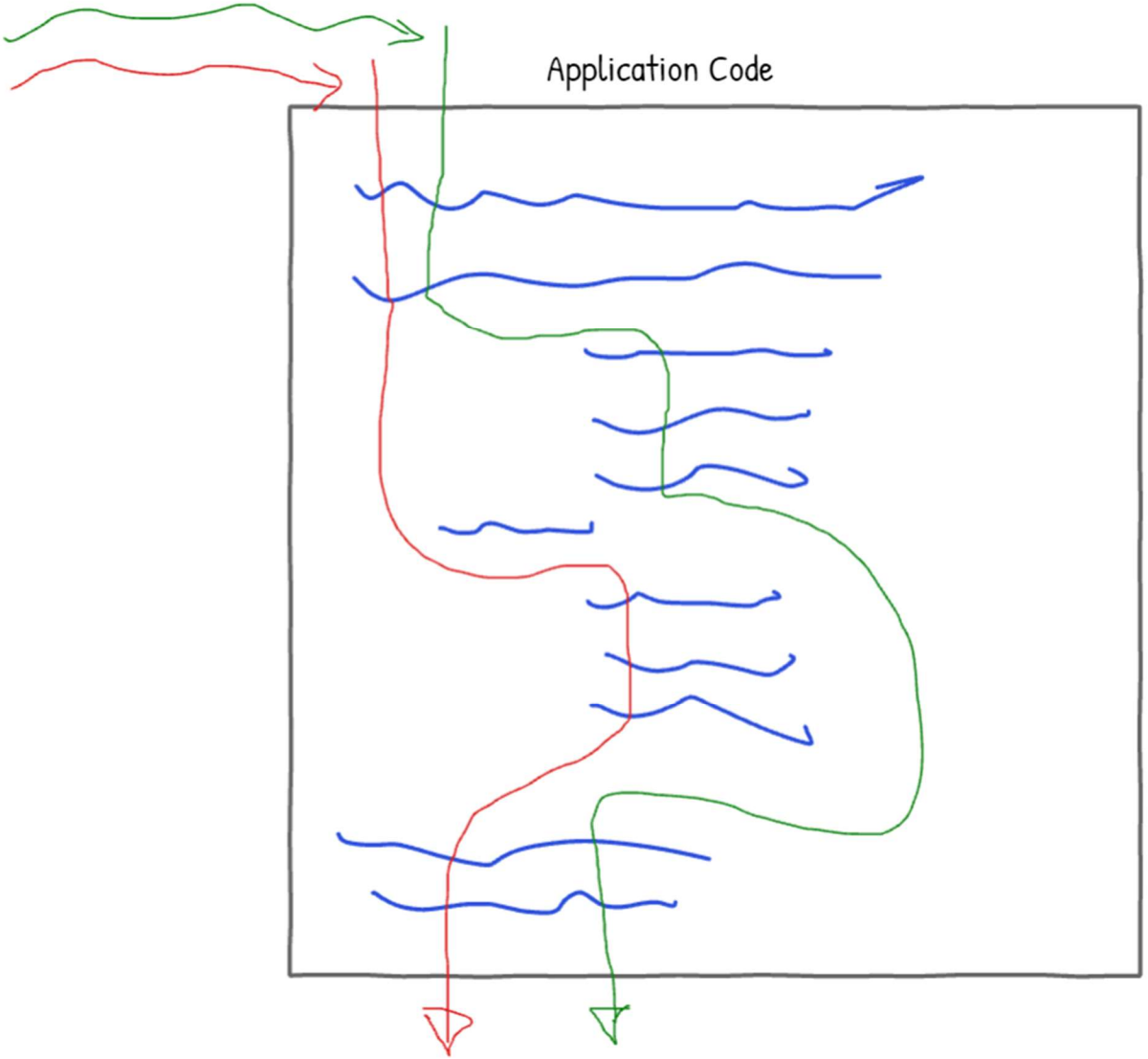


MODULE 10

What is Multi-threading?

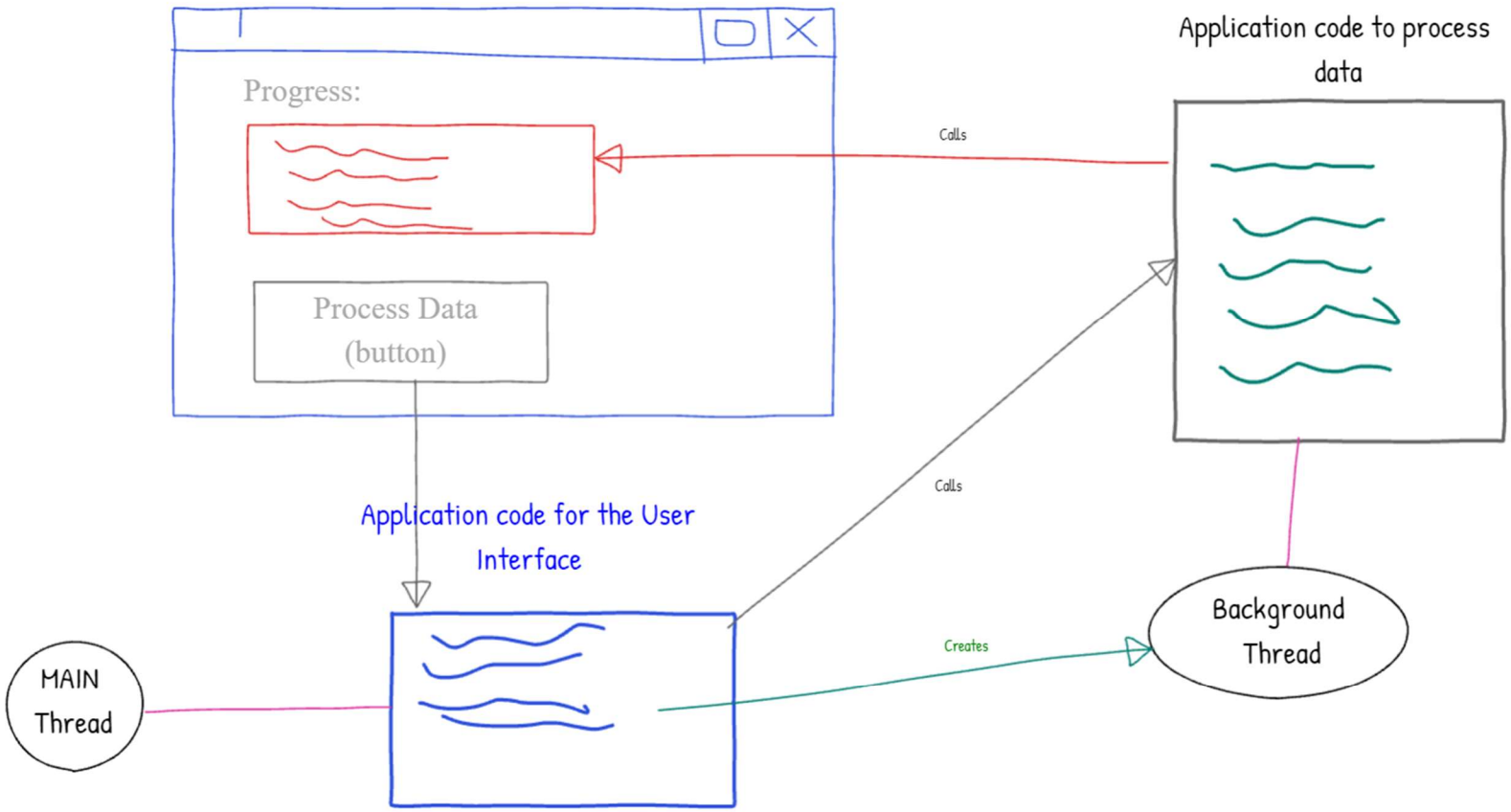
These are execution paths through the application. Can you see they look a bit like threads of cotton?

If the hardware on your device allows both paths to execute simulataneously then it is said to tbe muulti-threaded.



MODULE 10

Why do we need multi-threading? For this application, if the code to process the data was run on the main thread then the user interface would freeze. By running it on a separate thread both things can happen at the same time.



MODULE 10

```
class Program
{
    //delegate void MyDelegate(int x, int y, string z);
    static void Main(string[] args)
    {
        //MyDelegate x = AddThem;

        Action<int, int, string> x = AddThem;
        x(3, 4, "5");

        Func<int, int, string, int> y = AddThem2;
        Console.WriteLine(y(5,6,"7"));
        Console.ReadLine();
    }

    static void AddThem(int x, int y, string z)
    {
        Console.WriteLine(x + y + Convert.ToInt32(z));
    }

    static int AddThem2(int x, int y, string z)
    {
        return x + y + Convert.ToInt32(z);
    }
}
```

Action is a data type
Action is a generic
Action can point to a method that returns a void

Func is a data type
Func is a generic
Func can point to a method that returns a value

MODULE 10

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Type a number");
        string x = Console.ReadLine();


        Task T = new Task(p => Console.WriteLine(Convert.ToInt32(p) * Convert.ToInt32(p) * Convert.ToInt32(p)), x);

        T.Start();
        T.Wait();

        Console.ReadLine();

    }
}
```

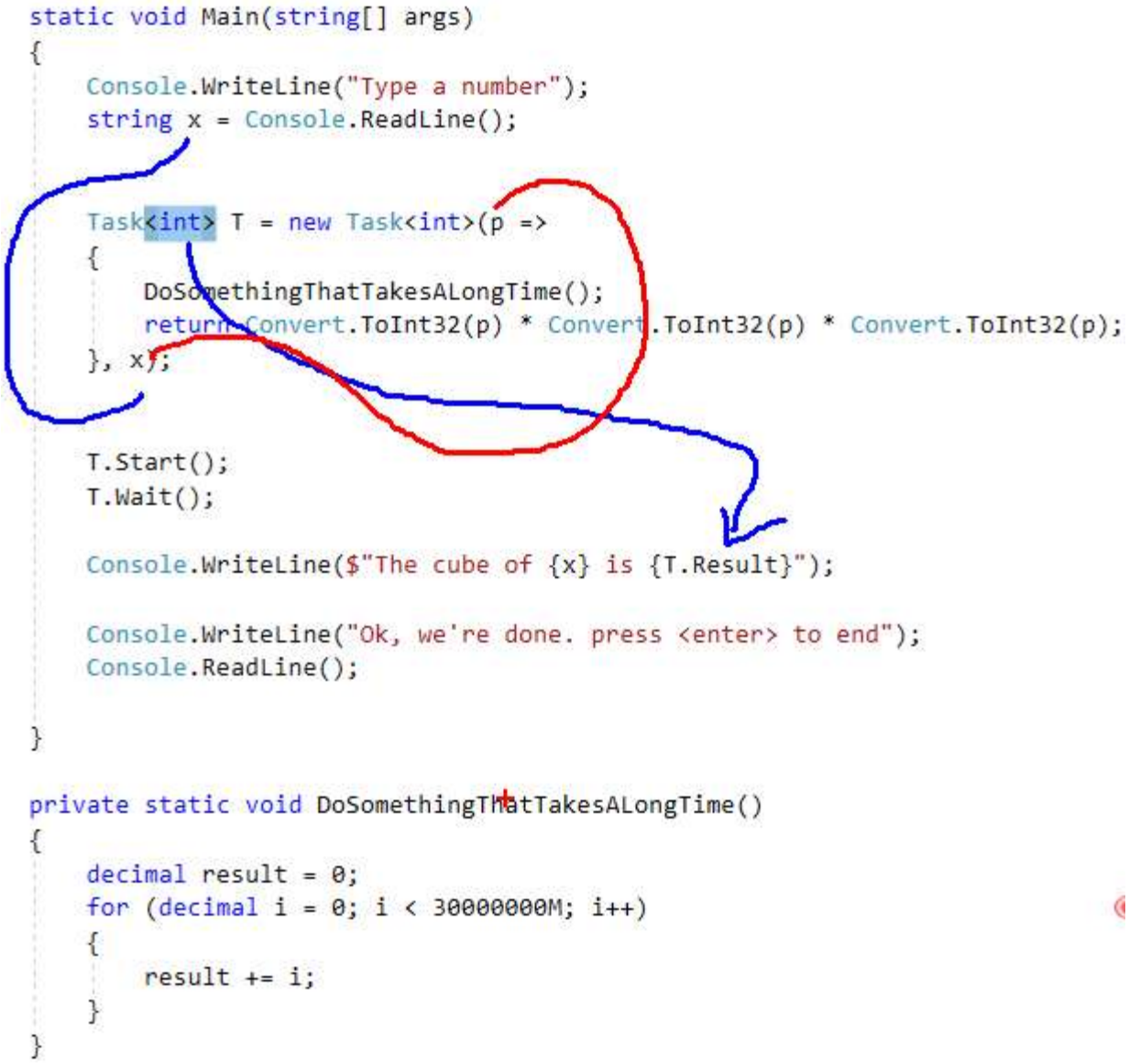
Lambda

This lambda is an inline method
it accepts one parameter called p

This is the value passed to p

MODULE 10

A simple example of running a task. Notice that was the task is to do is defined by a Lambda expression.

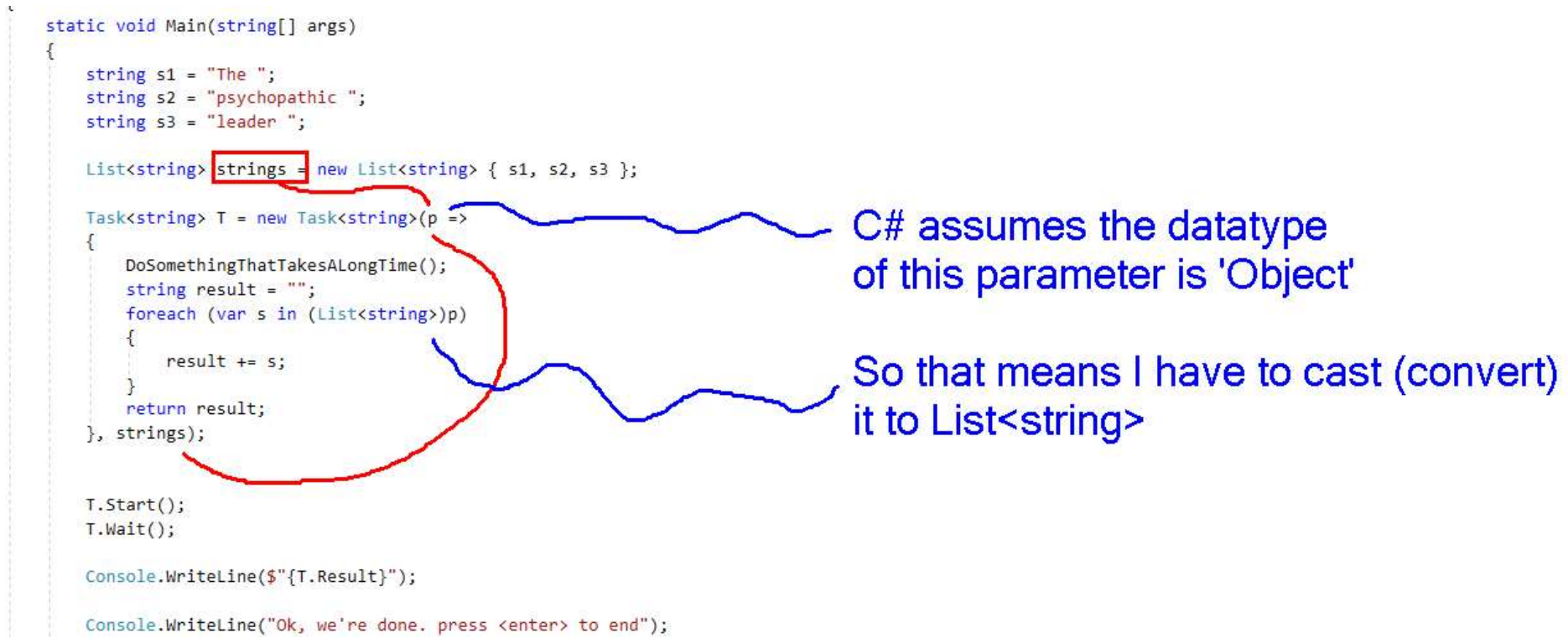


```
static void Main(string[] args)
{
    Console.WriteLine("Type a number");
    string x = Console.ReadLine();


    Task<int> T = new Task<int>(p =>
    {
        DoSomethingThatTakesALongTime();
        return Convert.ToInt32(p) * Convert.ToInt32(p) * Convert.ToInt32(p);
    }, x);


    T.Start();
    T.Wait();

    Console.WriteLine($"The cube of {x} is {T.Result}");

    Console.WriteLine("Ok, we're done. press <enter> to end");
    Console.ReadLine();

}

private static void DoSomethingThatTakesALongTime()
{
    decimal result = 0;
    for (decimal i = 0; i < 30000000M; i++)
    {
        result += i;
    }
}
```

MODULE 10

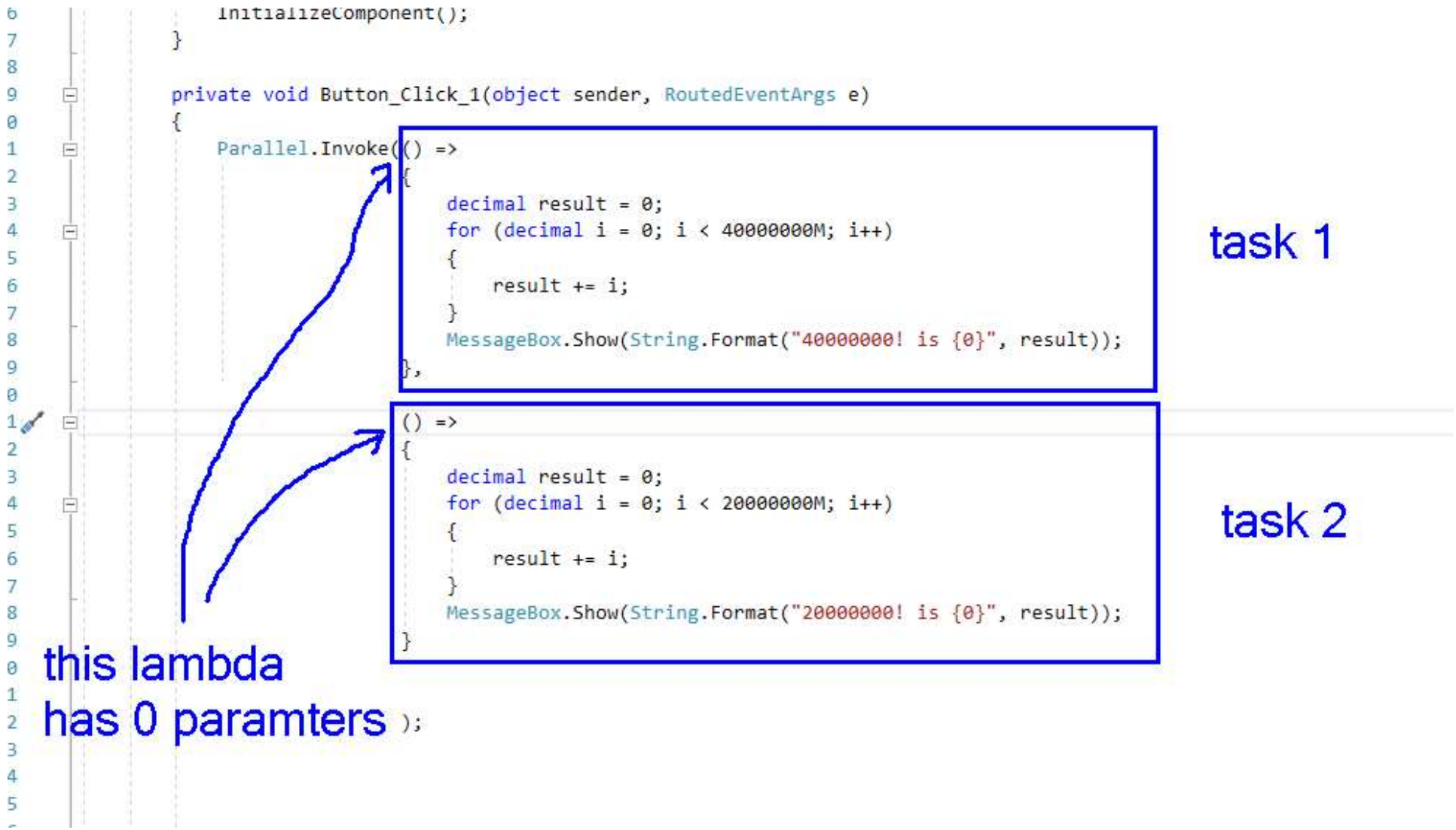
Starting a Task with a parameter. In this case the parameter is a list which can hold multiple items.

```
static void Main(string[] args)
{
    string s1 = "The ";
    string s2 = "psychopathic ";
    string s3 = "leader ";

    List<string> strings = new List<string> { s1, s2, s3 };

    Task<string> T = new Task<string>(p =>
    {
        DoSomethingThatTakesALongTime();
        string result = "";
        foreach (var s in (List<string>)p)
        {
            result += s;
        }
        return result;
    }, strings);


    T.Start();
    T.Wait();

    Console.WriteLine($"{T.Result}");

    Console.WriteLine("Ok, we're done. press <enter> to end");
```



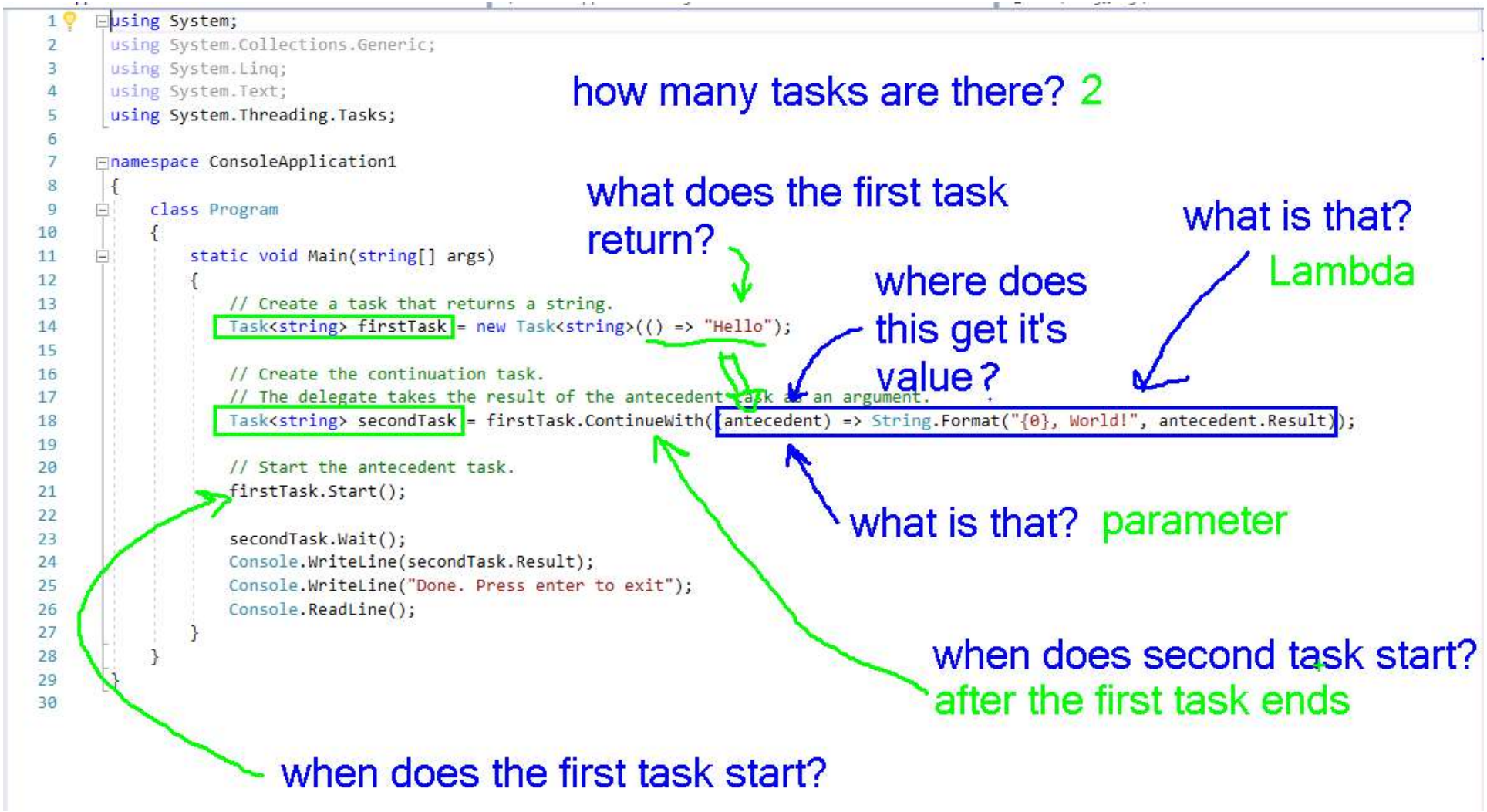
MODULE 10

Parallel Invoke



MODULE 10

Task ContinueWith




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create a task that returns a string.
            Task<string> firstTask = new Task<string>(() => "Hello");

            // Create the continuation task.
            // The delegate takes the result of the antecedent task as an argument.
            Task<string> secondTask = firstTask.ContinueWith((antecedent) => String.Format("{0}, World!", antecedent.Result));

            // Start the antecedent task.
            firstTask.Start();

            secondTask.Wait();
            Console.WriteLine(secondTask.Result);
            Console.WriteLine("Done. Press enter to exit");
            Console.ReadLine();
        }
    }
}
```

MODULE 10

If a worker thread tries to access a WPF user control that was created on the main thread, you will get a cross-threading exception. This is solved by either using Dispatcher.BeginInvoke or by using the async-await pattern.