


INTRODUCTION

- Name: Srinivasa Nihant Gangadharabhatla
- Program: M.S in Computer Science
- Class: Object Oriented Analysis and Design(CSCI 5448)
- Instructor: M.Kenneth Anderson
- Graduate School: University of Colorado, Boulder

INDEX

- INTRODUCTION
- ABOUT .NET FRAMEWORKS & C#
- SOME FACTS ON C#
- PROPERTIES AND FEATURES OF C#
- MISCELLANEOUS ABOUT C#
- GRAPHICAL USER INTERFACE IN C#
- SUMMARY
- REFERENCES

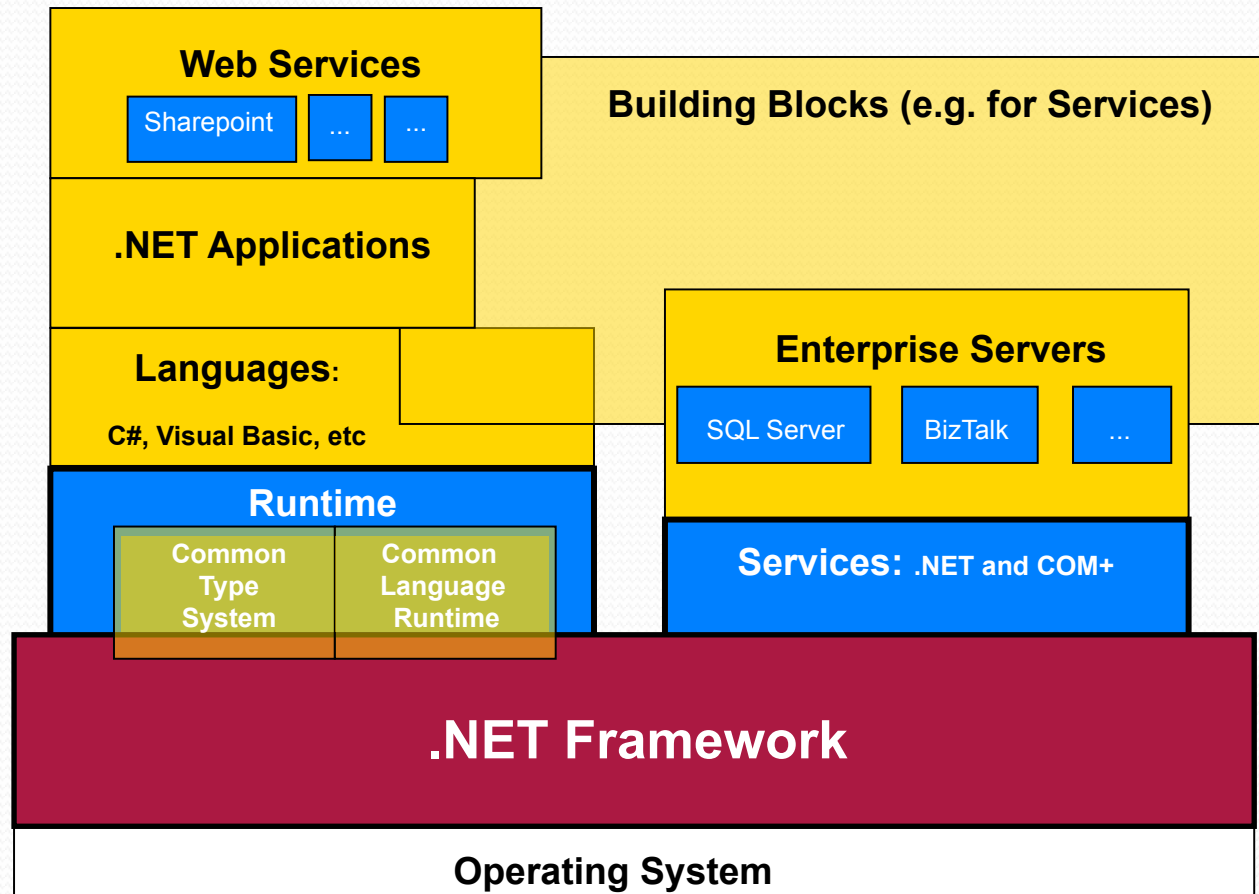
TOPIC - DETAILS

- *Topic name:* C# and .NET (# musical notation) A musical staff with a treble clef, a sharp sign on the first line, and two notes on the second line.
- *Details:* The presentation is about some features and implementation details of C# and .NET Frameworks

.NET Frameworks – Development

- The .NET Framework (pronounced dot net) is a software framework developed by Microsoft that runs primarily on Microsoft Windows.
- Although the development started in 1990s, the initial release was in the year 2000.
- The class library and the CLR(Common Language Runtime) together constitute the .NET Framework.
- The .NET Framework's Base Class Library(BCL) provides user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications.

What is .NET?

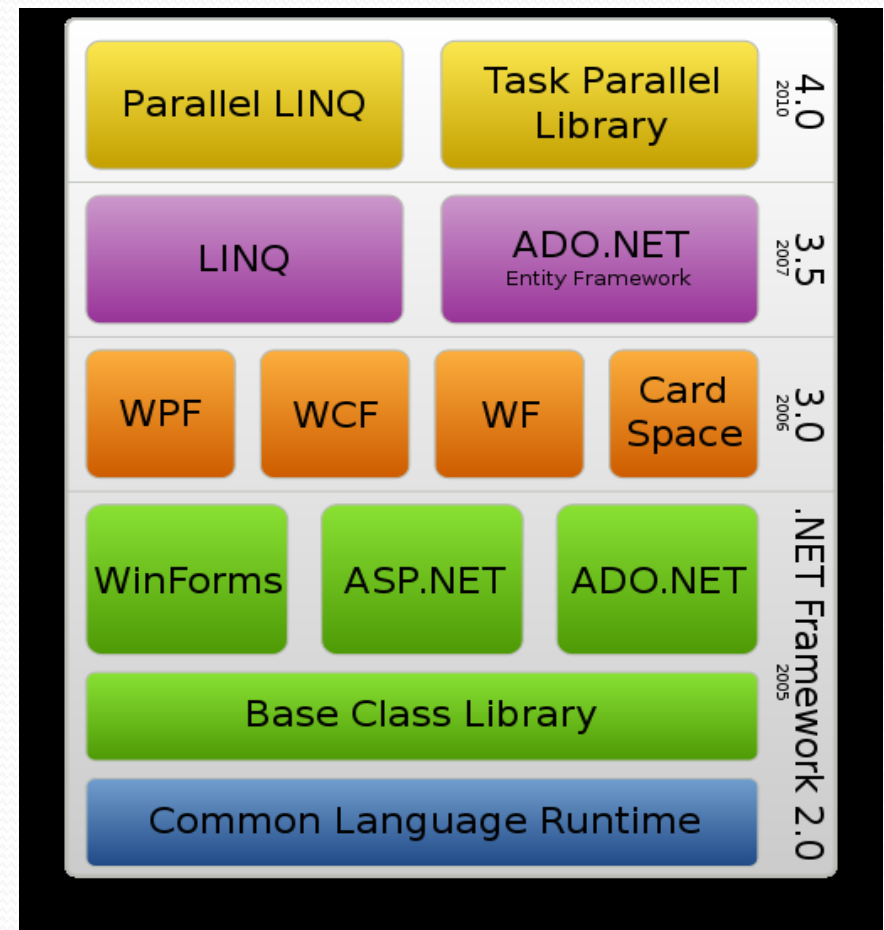


From MSDN

.NET Basic Understanding

- Given here, is the .NET Framework Stack.

- from Wikipedia



.NET Design Features

- ***Interoperability***: Generally, the computer systems commonly need interaction between newer and older applications, the .NET Framework provides means to access functionality implemented in newer and older programs that execute outside the .NET environment.
- ***The Common Language Runtime (CLR)***: CLR serves as the execution engine of the .NET Framework. All .NET programs execute under the supervision of the CLR. It guarantees certain properties and behaviors in the areas of memory management, security, and exception handling.
- ***Language Independence***: The .NET Framework introduces a Common Type System, or CTS. The CTS specification defines all possible datatypes and programming constructs supported by the CLR and how they may or may not interact with each other conforming to the Common Language Infrastructure (CLI).

.NET Design Features

- **Base Class Library(BCL):** The Base Class Library (BCL), part of the Framework Class Library (FCL), is a library of functionality available to all languages using the .NET Framework. The BCL provides classes that encapsulate a number of common functions, including file reading and writing, graphic rendering, database interaction, XML document manipulation, and many other. It consists of classes, interfaces of reusable types that integrates with CLR(Common Language Runtime).
- **Security:** The design addresses some of the vulnerabilities, such as buffer overflows. Moreover, .NET provides a common security model for all applications.
- .NET also is provided by features such as *portability*, *simplified deployment*.

.NET Basics

- Languages that comply with the CLI specification of .NET, can be implemented on .NET Framework.
- Following are common .NET Languages:
 - C#
 - Visual Basic
 - A# (CLI Implementation of ADA)
 - J# (CLI Implementation of JAVA)
 - Iron Ruby/Iron Python (An open-source CLI implementation)
 - Jscript.NET (A CLI Implementation)

.NET Languages

- Languages that comply with the CLI specification of .NET, can be implemented on .NET Framework.
- Following are common .NET Languages:
 - C# (Provided by Microsoft)
 - Visual Basic (Provided by Microsoft)
 - A# (CLI Implementation of ADA)
 - J# (CLI Implementation of JAVA, Provided by Microsoft)
 - Iron Ruby/Iron Python (An open-source CLI implementation)
 - Jscript.NET (A CLI Implementation, Provided by Microsoft)

Some IDEs for .NET development

- The below are some IDEs for development and testing of .NET Frameworks are:
 - Visual Basic
 - Visual C++
 - Visual C#
 - Visual Web Developer

C# is a multi-paradigm language

- C# is a multi-paradigm language:
 - Object Oriented
 - Strongly typed
 - Functional, declarative
 - Generic
 - Component Oriented
- It was developed by Microsoft within its .NET initiative and later approved as a standard by ECMA and ISO .
- C# is one of the programming languages designed for the Common Language Infrastructure.
- It was focussed to be a simple, modern, general purpose Object Oriented Programming language

Interesting things about C#

- C# was released in the year 2000 by Microsoft
- Anders Hejlsberg of Microsoft was the principle developer.
- Current version is 5.0, released on August 15, 2012.
- C# resembles a lot of features of C++ and JAVA.
- It supports Graphical User Interface for personal computers really easy.

Applications

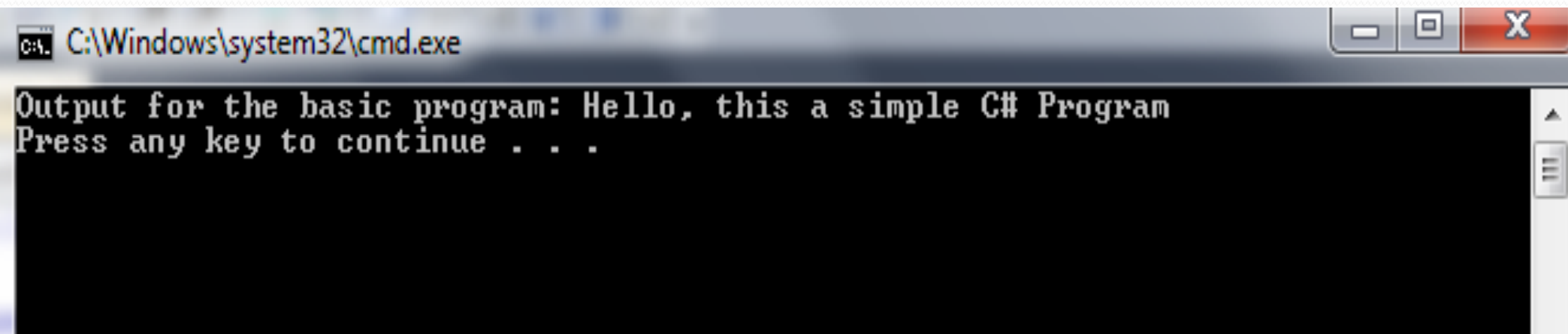
- Applications that can be run on C#
 - Windows Form Application: It is Graphical User Interface based
 - Console Application: This application is implemented on command prompt

A Simple Basic C# Program

```
1 using System;
2
3 namespace ConsoleApplication1
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Console.WriteLine("Output for the basic program: Hello, this a simple C# Program");
10        }
11    }
12 }
```

➤ Output to be continued..

A Simple Basic C# Program output



A screenshot of a Windows command prompt window. The title bar shows the path `C:\Windows\system32\cmd.exe` and standard window controls (minimize, maximize, close). The main area of the window is black with white text. The text displayed is: `Output for the basic program: Hello, this a simple C# Program` followed by `Press any key to continue . . .` on the next line. A vertical scrollbar is visible on the right side of the window.

```
C:\Windows\system32\cmd.exe
Output for the basic program: Hello, this a simple C# Program
Press any key to continue . . .
```


Keywords

➤ Some of the common keywords used are:

- abstract - extern - object - try
- break - finally - override - this
- case - for - params - throw
- catch - foreach - private - typeof
- class - if - protected - using
- const - interface - public - virtual
- delegate - internal - return - void
- enum - namespace - static - while

Different TYPES

- Different Types are:
- *Value Type*: The variable name consists of the value only
 - int, double, float, enum
- *Reference Type*: Here, the refernece or the pointer of the actual address of the memory is stored
 - Array, Delegate, Class, Interface and so on.

Comments

- Comments can be assigned, so that the code is easy to understand for single line comments and delimited comments, documentation comments

- *Syntax*: `/* Here text is ignored by compiler */`

(OR)

`// Comments is also valid`

(OR)

`/// Documentation Comments`

- Nested Comments are not allowed

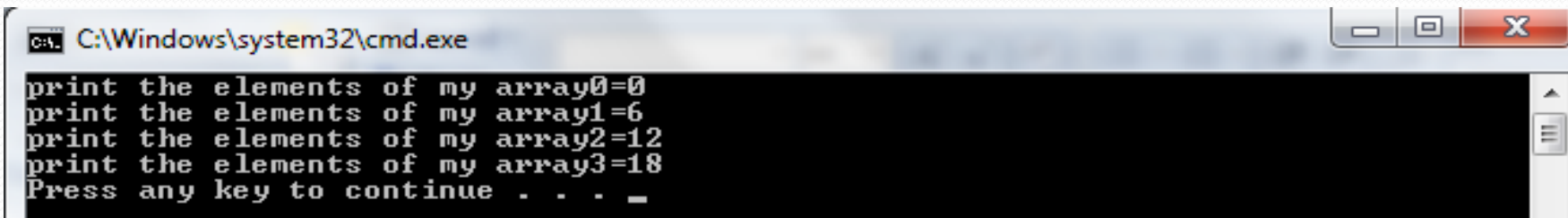
Arrays

- C# provides two types of arrays:
 - One dimensional arrays can be thought of as a single line or vectors of elements
 - Multi-dimensional arrays are composed such that each position in the primary vector is itself an array, called a sub-array

One-dimensional Arrays

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] array = new int[4];
            for(int i=0;i<4;i++)
                array[i] = i*6;
            for (int i=0;i<4;i++)
                Console.WriteLine("print the elements of my array{0}={1}", i, array[i]);
        }
    }
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output of the program is displayed in a monospaced font:

```
print the elements of my array0=0
print the elements of my array1=6
print the elements of my array2=12
print the elements of my array3=18
Press any key to continue . . . _
```

Multi-dimensional Arrays

➤ Multi-dimensional arrays are of two types:

- Rectangular Arrays

- Where all sub-arrays in a particular dimension have same length

int x = myArr[2,3,1] //one set square brackets

- Jagged arrays

- Each sub-array is an independent array

- can have sub-array of different lengths

jagArr[2][3][5] //three sets of square brackets

Explicit Initialization of Arrays

- For a one dimension-arrays we can set explicit initial values by including an initializing list

```
int[] intArr = new int[5] { 10,25,30,40};
```

- For initializing a rectangular array

```
int[,] intArr1 = new int[3,2] {  
    {10,1},{2,3},{4,5}};
```

Properties

- There are two types of methods: Properties are members of the class which provide two simplified methods setters and getters implementation of its private fields
- Implementation is coming up...

Properties

```
Program.cs* X
TimePeriod seconds
using System;

class TimePeriod
{
    private double seconds;

    public double TotalHours
    {
        get { return seconds / 3600; }
        set { seconds = value * 3600; }
    }
}

class Program
{
    static void Main()
    {
        TimePeriod t = new TimePeriod();

        // the Hours property causes the 'set' accessor to be called.
        t.TotalHours = 24;

        // the Hours property causes the 'get' accessor to be called.
        System.Console.WriteLine("Calculate Time: " + t.TotalHours);
    }
}
```

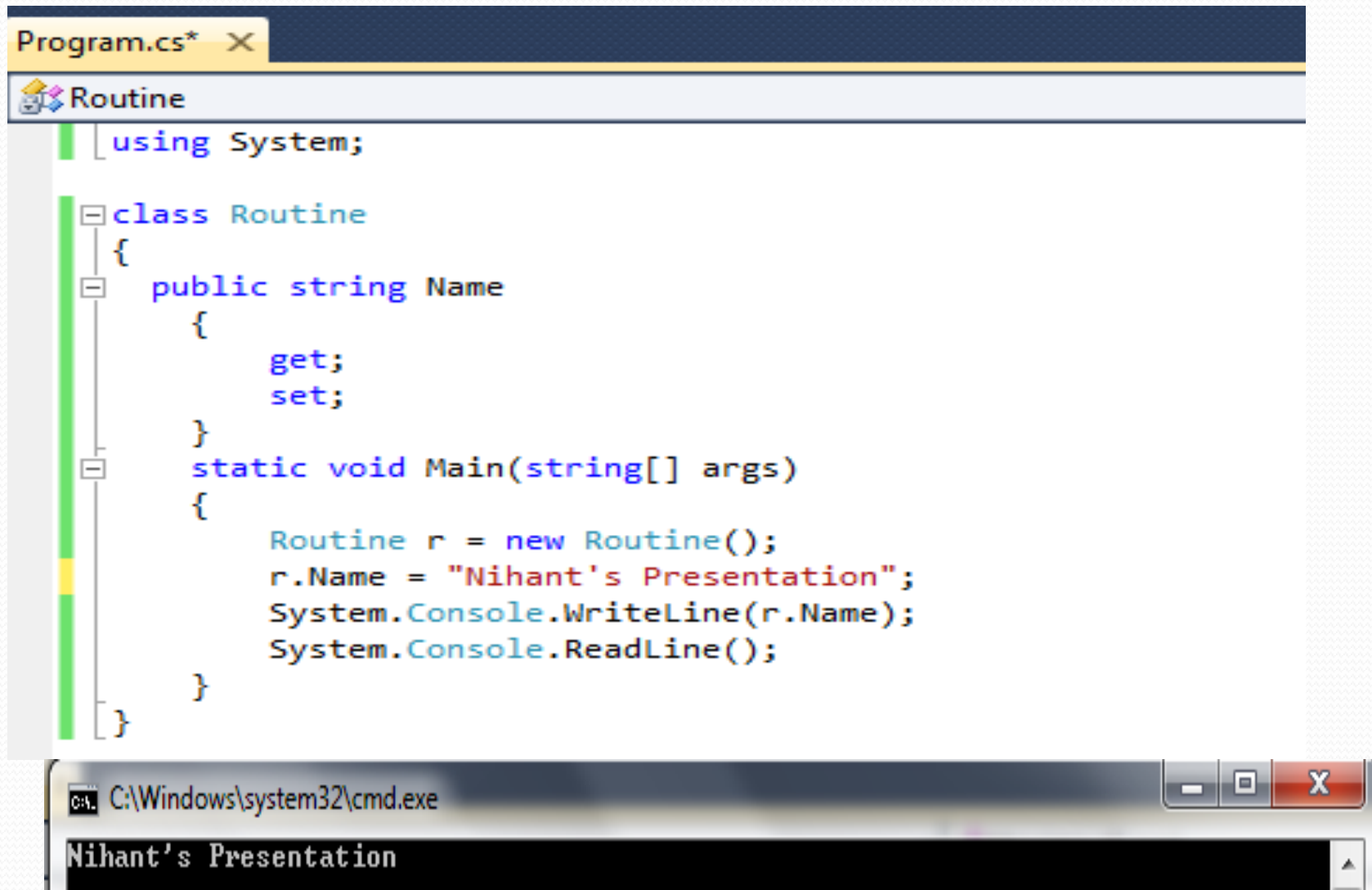
C:\Windows\system32\cmd.exe

```
Calculate Time: 24
Press any key to continue . . . .
```

The properties can be set automatic in C#

- C# allows the feature to automatically implement the getter and setter.
- Hence, we can have a direct access to the data members of the class.
- The implementation is in the following slide..

Properties can be set automatic



```
Program.cs* X
Routine
using System;

class Routine
{
    public string Name
    {
        get;
        set;
    }
    static void Main(string[] args)
    {
        Routine r = new Routine();
        r.Name = "Nihant's Presentation";
        System.Console.WriteLine(r.Name);
        System.Console.ReadLine();
    }
}
```

C:\Windows\system32\cmd.exe

Nihant's Presentation

Classes and Structs

- Classes are declared by using the class keyword

- Explained as,

```
public class Employer{
```

```
// Fields, properties, methods and others
```

```
...}
```

- Structs are defined by using the struct keyword

- Explained as,

```
public struct Employee { // Fields, properties,  
methods and so on
```

```
... }
```

Nested Classes

```
class Program
{
    public class Program1
    {
        private int b;
        // Other variables, methods| declarations and so on..
    }
    static void Main(string[] args)
    {
        // method invocation and so on go here..
    }
}
```

Nested Classes are supported by C#, defaults to private

Polymorphism

➤ Polymorphism

- Methods default to being non-virtual
- To be virtual it must be defined as virtual
 - *Example:* `public virtual int Sum(int a, int b);`
- To override a virtual method, you use the override keyword
 - *Example:* `public override int Sum(int a, int b);`
 - Methods not marked virtual are equivalent to Java final methods

Note: Methods can be marked with `new` to break the virtual chain

What is an Indexer?

- An Indexer is a set of get and set accessors, similar to those of properties.
- Representation of an indexer:

```
string this [int index]
{
    set
    {
        SetAccessorCode
    }
    get
    {
        GetAccessorCode
    }
}
```

Inheritance

- Inheritance is considered to be one of the three pillars (apart from encapsulation and polymorphism) of object oriented programming
- Inheritance allows us to reuse, modify and extend the characteristics (behavior) that is defined in other classes.
- Derived classes inherit behavior from base classes.
- To override a method defined in the base class, the keyword 'override' is used

Class Inheritance

- We can use the existing class, called the based class, as the basis for a new class called the derived class.
- Members of the derived class consist of:
 - members in its own class
 - members of the base class
- A derived class can never delete any of the members it has inherited.

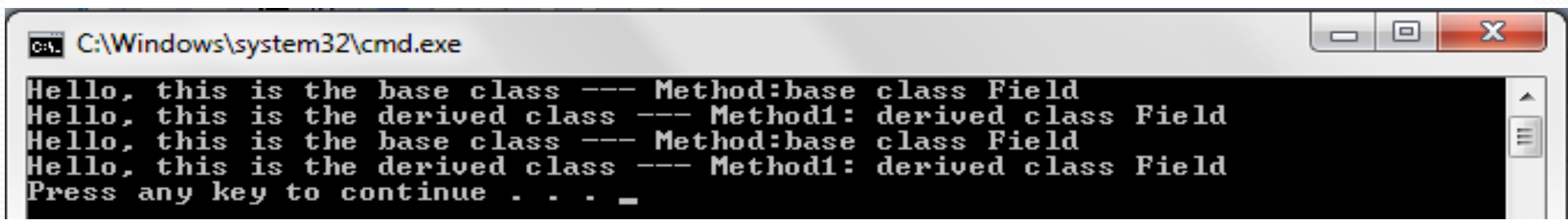
```
class OneClass : SomeClass {  
    ... // colon is to directly inherit from the base  
        class  
}
```

Accessing the inherited members

```
class SomeClass{
    public string Field = "base class Field";
    public void Method(string value){
        Console.WriteLine("Hello, this is the base class --- Method: {0}", value);
    }
}

class OneClass:SomeClass{
    public string Field1 = "derived class Field";
    public void Method1(string value){
        Console.WriteLine("Hello, this is the derived class --- Method1: {0}", value);
    }
}

class MainProgram
{
    static void main(String[] args)
    {
        OneClass oc = new OneClass();
        oc.Method(oc.Field);
        oc.Method1(oc.Field1);
        oc.Method(oc.Field1);
        oc.Method1(oc.Field);
    }
}
```



```
C:\Windows\system32\cmd.exe
Hello, this is the base class --- Method:base class Field
Hello, this is the derived class --- Method1: derived class Field
Hello, this is the base class --- Method:base class Field
Hello, this is the derived class --- Method1: derived class Field
Press any key to continue . . . _
```

Access Modifiers

- There are five categories:
 - private
 - public
 - protected
 - internal
 - protected internal

- The 'partial' keyword can be used to split up a class definition in to multiple location. It may be helpful when many developers are working on different parts of the same class.

Delegates

- A delegate can be thought of as an object that contains an ordered list of methods with the same signature and return type
 - It contains a list of methods is called the invocation list
 - When a delegate is invoked, it calls each method in its invocation list.
- Delegates are used to implement GUI and event handling in the .net framework. They are very similar to pointers in C (and C++)

Delegates

- Declaration of a variable of a delegate type:

```
MyDel delVar;
```

↑ ↑
Delegate Type Variable

- Delegate objects can be created by a *new* operator. The operand of new operator consists:
 - delegate type name
 - a set of parenthesis containing the name of a method to use as the first member in invocation list

How does Delegation work?

```
class Simpleprogram
{
    delegate int mydel(int a);

    int add(int a) { return a * a; }
    int mul(int a) { return a + a; }

    static void Main(string[] args)
    {
        Simpleprogram sp = new Simpleprogram();
        mydel d = sp.add;
        System.Console.WriteLine(d(7));
        d = sp.mul;
        System.Console.WriteLine(d(6));
        System.Console.Read();
    }
}
```

C:\Windows\system32\cmd.exe

49
12

Lambda Expression in C#

- The lambda operation is denoted by “=>”. C#, implementors (functions) that are targeted by a delegate.

```
namespace ConsoleApplication
{
    using System;
    class Program
    {
        delegate int del(int a, int b, int c);
        static void Main(string[] args) {
            del d=(a,b,c) => a*3+b-c;
            // => operation here is lambda
            System.Console.WriteLine("output for the delegation lambda operation is: {0}", d(4,5,6));
            System.Console.Read();
        }
    }
}
```

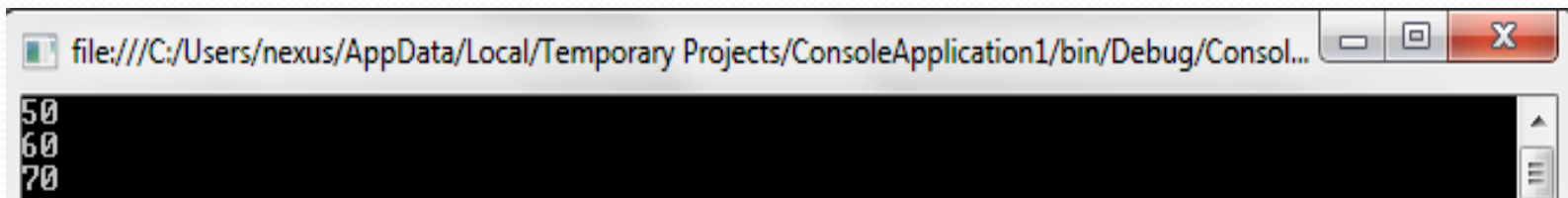
file:///C:/Users/nexus/documents/visual studio 2010/Projects/Program/Program/bin/Debug/Progr...

output for the delegation lambda operation is: 11

Anonymous Methods

- An anonymous method is declared in-line, at the point of instantiating a delegate.

```
class Program
{
    public static int Mul10(int x)
    {
        return x * 10;
    }
    delegate int MainDel(int Param);
    static void Main(string[] args)
    {
        MainDel del = Mul10;
        Console.WriteLine("{0}", del(5));
        Console.WriteLine("{0}", del(6));
        Console.WriteLine("{0}", del(7));
        Console.Read();
    }
}
```



The screenshot shows a console window with the following output:

```
50
60
70
```


Events

- In fact, an event is a simpler delegate that is specialized for a particular use.
- Following are some important terms related to events:
 - Raising an event: Term for invoking or firing an event
 - Publisher: A class or a struct that makes an event available to other classes or structs for their use.
 - Subscriber: A class or a struct that registers methods with a publisher.
 - Event handler: A method that is registered with an event. It can be declared in the same class or struct as the event, or in a different class or struct

Events

- To declare an event is simple. It requires only a delegate type and a name.
- *Syntax is:*

```
class MyTimer{  
public event EventHandler Elapsed;
```

Delegate type

- We can also declare more than one event in a declaration statement using a comma separated list

Interface

- An interface is a reference type that represents a set of function members , but does not implement them.
- Although we define the method, its name, parameters and a return type. There is no implementation. Instead the implementation is replaced by a semicolon

```
public interface ICompare
{
    int CompareTo(object o);
}
```

Let's work on a Simple Interface

```
using System;
```

```
interface Simple // Declare the interface
{
    void PrintOut(string s);
}

class MyClass : Simple // Declare the class
{
    public void PrintOut(string s) // Implementation goes here
    {
        Console.WriteLine("Call the methods which I want to implement through: {0}", s);
    }
}

class Program
{
    static void main(String[] args)
    {
        MyClass mc = new MyClass(); // Create instance
        mc.PrintOut("the object assigned"); // Call the method
    }
}
```

C:\Windows\system32\cmd.exe

Call the methods which I want to implement through: the object assigned

Generics

- Generics allows us to refactor the code we write and add an additional layer of abstraction to that, for certain kinds of code, for data types that are not hard-coded.
- It is particularly designed for cases in which there are many(multiple) sections of code performing the same instructions, but on different data types.
- A placeholder for type `<T>` is used and when these methods or classes are used, the user just simply has to plug in the appropriate type.
- The following slide has its implementation...

```

using System;
using System.Collections.Generic;
namespace ConsoleApplication1
{
    class Stack<T>
    {
        T[] StackArray;
        int StackPointer = 0;
        public void Push(T x)
        {
            if (!IsStackFull)
                StackArray[StackPointer++] = x;
        }
        public T pop(){
            return (!IsStackEmpty)
                ? StackArray[--StackPointer] : StackArray[0];
        }
        const int MaxStack = 10;
        bool IsStackFull { get { return StackPointer >= MaxStack; } }
        bool IsStackEmpty { get { return StackPointer <= 0; } }
        public Stack()
        {
            StackArray = new T[MaxStack];
        }

        public void Print()
        {
            for(int i=StackPointer-1; i>=0; i--)
                Console.WriteLine("Value is: {0}", StackArray[i]);
        }
    }
    class Program
    {
        static void Main()
        {
            Stack<int> StackInt = new Stack<int>();
            Stack<string> StackString = new Stack<string>();
            StackInt.Push(5);
            StackInt.Print();
            StackString.Push("This is a presentation");
            StackString.Print();
        }
    }
}

```

C:\Windows\system32\cmd.exe

```

Value is: 5
Value is: This is a presentation

```

Object Initializers

- Object initializers allow for initializing an object at creation time without explicit constructors. It can also be used with anonymous types
- The following slide gives the demo for Object Initializers concept

Demo: Object Initializers

```
namespace ConsoleApplication2
{
    class Demo
    {
        public string Name
        { get; set;}
        public int num
        { get; set; }

        static void Main(string[] args)
        {
            Demo d = new Demo {Name="This is Object Oriented Analysis and Design"};
            Demo d1 = new Demo { num = 2012 };
            //object initializer
            System.Console.WriteLine("Details of the presentation is: {0}",d.Name);
            System.Console.WriteLine("Present year is:{0}",d1.num);
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
Details of the presentation is: This is Object Oriented Analysis and Design
Present year is:2012
```


Enumerators

- An array can produce, upon request, an object called an enumerator.
- An enumerator knows the order of the items.
- Types of Enumerators:
 - IEnumerator/IEnumerable interfaces called the non generic interface form.
 - IEnumerator<T>/IEnumerable<T> interfaces called the generic interface form.

Iterators

- The following method declarations implements an iterator
 - The iterator returns a generic enumerator that returns three items of type string
 - The *yield* return statements declare that this is the next item in the enumeration.
- Let's look at the two examples here...

Iterators

```
IEnumerator<string> ColorsChose()           //Version 1
{
    yield return "black";                   // yield return
    yield return "gray";                   // yield return
    yield return "white";                  // yield return
}
```

```
IEnumerator<string> ColorsChose()           //Version 2
{
    string[] Colors = { "black", "grey", "white"};
    for(int i=0; i< TheColors.Length;i++)
    yield return Colors[i];                 // yield return
}
```

Namespaces

- Namespaces group a set of types together and give them a name called namespace name
- A namespace must be distinctive from other namespace names and it should be descriptive of the contents.

Declaration:

```
namespace SimpleNamespace  
{  
    TypeDeclarations  
}
```

Example: Namespace

```
namespace spaceA
{
    class ClassA
    {
        public void display()
        {
            System.Console.WriteLine("This is Class A");
        }
    }
}
namespace spaceB
{
    class Program
    {
        static void Main(string[] args)
        {
            spaceA.ClassA c = new spaceA.ClassA();
            c.display();
            Console.Read();
        }
    }
}
```

C:\Windows\system32\cmd.exe

This is Class A

Attribute

- An attribute is a special type of class, designed specifically for storing information about the program constructs. We can apply attributes to a construct of a program's source code, to declare something about the construct.
- Attributes are declared in square brackets above the class name, method name and others.
- We apply attributes to program constructs in the source code.
- The compiler takes the source code and produces metadata from attributes, and places that metadata in the assembly.

Attributes

```
using System;
using System.Diagnostics;

namespace AttrObsolete
{
    class Program
    {
        [Obsolete("Use method SuperPrintOut")] //Apply the attribute
        static void PrintOut(string str)
        {
            Console.WriteLine(str);
        }
        static void Main(string[] args)
        {
            PrintOut("Here starts the Main"); //Invoke the obsolete method
            Console.Read();
        }
    }
}
```

C:\Windows\system32\cmd.exe

Here starts the Main

Miscellaneous Features

- String Handling: The C# predefined type string represents the .NET class System.String. The most important things to know about strings are the following:
 - strings are arrays of Unicode characters
 - strings are immutable—they cannot be changed
- Some useful Members of the string type:
Length, Concat, Contains, Format, Insert, Remove, Replace, Substring, ToUpper, ToLower

Miscellaneous

- Threading: It's possible to write multi-threaded programs using the System.Threading class library.
- Nullable Types: Nullable types allow us to create a value type variable that can be marked as valid or invalid so that we can make sure a variable is valid before using it. Regular value types are called *non-nullable types*

int? MyInt = 28



name of the nullable type includes suffix

Miscellaneous

- Documentation comments: The documentation comments feature allows us to include documentation of our program in the form of XML elements
- Visual Studio assists us in inserting the elements, and will read them from the source file and copy them to a separate XML file.
- Some Documentation code XML tags are:
<code> <example> <remarks> <summary>
<param> <value> <seealso>

Miscellaneous

- Exception Handling: Exceptions can be handled by C#'s try, catch, throw and finally.
- Collection classes: With the help of data structures such as lists, hash tables and queues.
- Conversions: It is a process of taking a value of one type and using it as the equivalent value of another type. Boxing, Unboxing, user-defined conversions and reference conversions are some of them.

Miscellaneous details about C#

➤ No COM Required

- The .NET Framework frees the programmer from the COM legacy. As a C# programmer, you don't need to use COM, and therefore do not need any of the following:
 - *The IUnknown interface:* In COM, all objects must implement interface IUnknown. In contrast, all .NET objects derive from a single class called object. Interface programming is still an important part of .NET, but it is no longer the central theme.
 - *Type libraries:* In .NET, information about a program's types is kept together with the code in the program file, not in a separate type library the way it is in COM.
 - *Reference counting:* The programmer no longer has to keep track of references to objects. In .NET, the GC keeps track of references and deletes objects when appropriate.
 - *HRESULT:* The HRESULT data type used in COM to return runtime error codes is not used in .NET. Instead, all runtime errors produce *exceptions*.
 - *The registry:* This system-wide database that holds information about the operating system and application configurations is not used when deploying .NET applications. This simplifies installation and removal of programs.

GUI: In C#

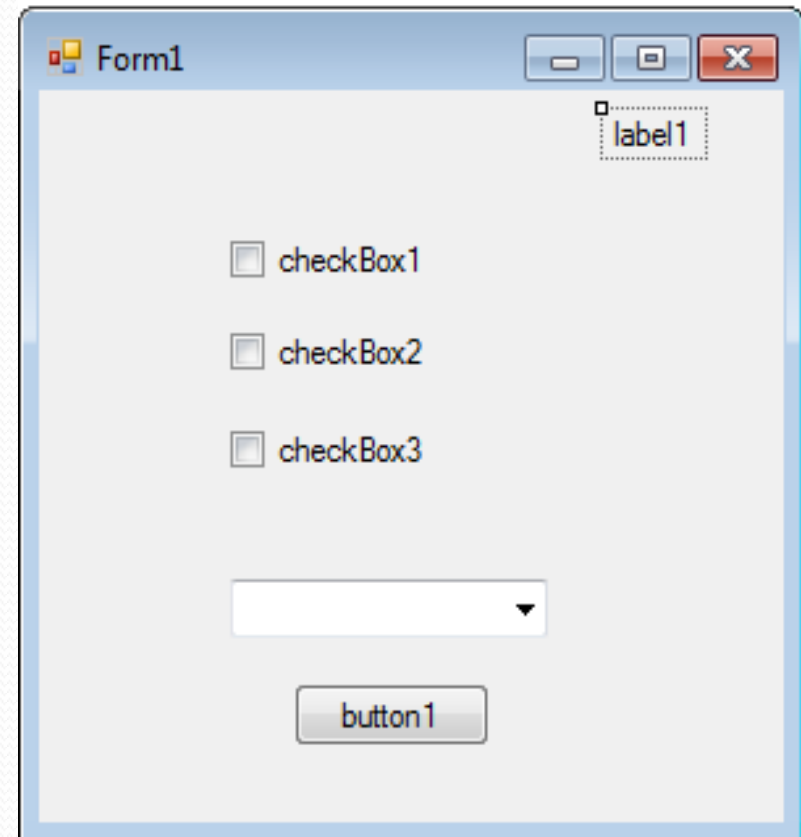
- The .NET framework provides a class library of various graphical user interface tools such as frame, text box, buttons, combo boxes and many others, that C# can use to implement a GUI very easily and fast.
- It also equips these classes with events and event handlers to perform action upon interacting with these visual items by the user.

Windows Forms Application

- Different Components(For GUI) are:
 - Form: To display a form
 - Label: To display a label
 - Radio button: To display a selectable button
 - TextBox: To write/modify text
 - Button: To click a button.
 - Progress bars, checked list box and others.

Windows Forms Applications: Components' quick view

The Windows Form Application here is provided by the components label, checkboxes, combobox and the button.



To sum up

- The .NET Framework offers programmers considerable improvements over previous Windows programming environments
- The CLR, the BCL and C# have all been designed to be thoroughly object-oriented and well integrated environment.
- The CLR has a tool called the Garbage Collector(GC), which automatically manages memory.

References and IDE used

➤ Books:

- *Illustrated C# by Daniel Solis*

➤ LINKS:

- [Microsoft\(MSDN\)](#)
- [Wikipedia-C#](#)

The IDE used for the presentation is

- Microsoft Visual C# 2010 Express



THANK YOU!