

**www.PragimTech.com**

Training + Placements = Our Success

Pragim@PragimTech.com

Marathahalli, Bangalore - 09900113931

# Part 1 - Introduction to C#

Venkat

PRAGIM Technologies

kudvenkat@gmail.com

# In this session

- Basic structure of a C# program.
- What is a Namespace.
- Purpose of Main method.

**www.PragimTech.com**

**Training + Placements = Our Success**

**Pragim@PragimTech.com**

**Marathahalli, Bangalore - 09900113931**

# Sample Program

```
// Namespace Declaration  
using System;  
  
class Pragim  
{  
    public static void Main()  
    {  
        // Write to console  
        Console.WriteLine ("Welcome to PRAGIM Technologies!");  
    }  
}
```

**www.PragimTech.com**

Training + Placements = Our Success

Pragim@PragimTech.com

Marathahalli, Bangalore - 09900113931

# Using System declaration

The namespace declaration, `using System`, indicates that you are using the `System` namespace.

A namespace is used to organize your code and is collection of classes, interfaces, structs, enums and delegates.

Main method is the entry point into your application.

**www.PragimTech.com**

Training + Placements = Our Success

Pragim@PragimTech.com

Marathahalli, Bangalore - 09900113931

**www.PragimTech.com**

Training + Placements = Our Success

Pragim@PragimTech.com

Marathahalli, Bangalore - 09900113931

## **Part 2 - Reading & Writing to Console**

Venkat

PRAGIM Technologies

kudvenkat@gmail.com

# In this session

- Reading from the console
- Writing to the console
- 2 ways to write to console
  - a) Concatenation
  - b) Place holder syntax – Most preferred

**www.PragimTech.com**

**Training + Placements = Our Success**

**Pragim@PragimTech.com**

**Marathahalli, Bangalore - 09900113931**

# Reading and writing to console

```
using System;  
  
class Program  
{  
    static void Main()  
    {  
        // Prompt the user for his name  
        Console.WriteLine("Please enter your name");  
        // Read the name from console  
        string UserName = Console.ReadLine();  
        // Concatenate name with hello word and print  
        Console.WriteLine("Hello " + UserName);  
  
        //Placeholder syntax to print name with hello word  
        //Console.WriteLine("Hello {0}", UserName);  
    }  
}
```

**Note: C# is case sensitive**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 3 – Built-in types in C#

Venkat

PRAGIM Technologies  
kudvenkat@gmail.com

# In this session we will learn

- Built-in types in C#

**Boolean type** – Only true or false

**Integral Types** - sbyte, byte, short, ushort, int, uint, long, ulong, char

**Floating Types** – float and double

**Decimal Types**

**String Type**

- Escape Sequences in C#

<http://msdn.microsoft.com/en-us/library/h21280bw.aspx>

- Verbatim Literal

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# Verbatim Literal

**Verbatim literal**, is a string with an @ symbol prefix, as in @“Hello”.

**Verbatim literals make escape sequences translate as normal printable characters to enhance readability.**

## **Practical Example:**

**Without Verbatim Literal :** “C:\\\\Pragim\\\\DotNet\\\\Training\\\\Csharp” – Less Readable

**With Verbatim Literal :** @“C:\\Pragim\\DotNet\\Training\\Csharp” – Better Readable

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 4 – Built-in string type in C#

Venkat

PRAGIM Technologies  
kudvenkat@gmail.com

# In this session we will learn

- Built-in types in C#

**Boolean type** – Only true or false

**Integral Types** - sbyte, byte, short, ushort, int, uint, long, ulong, char

**Floating Types** – float and double

**Decimal Types**

**String Type**

- Escape Sequences in C#

<http://msdn.microsoft.com/en-us/library/h21280bw.aspx>

- Verbatim Literal

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# Verbatim Literal

**Verbatim literal**, is a string with an @ symbol prefix, as in @“Hello”.

**Verbatim literals make escape sequences translate as normal printable characters to enhance readability.**

## **Practical Example:**

**Without Verbatim Literal :** “C:\\\\Pragim\\\\DotNet\\\\Training\\\\Csharp” – Less Readable

**With Verbatim Literal :** @“C:\\Pragim\\DotNet\\Training\\Csharp” – Better Readable

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 5 – Common Operators in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- Assignment Operator =
- Arithmetic Operators like +,-,\*,/,%
- Comparison Operators like ==, !=,>, >=, <, <=
- Conditional Operators like &&, ||
- Ternary Operator ?:
- Null Coalescing Operator ??

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Program without ternary Operator

```
using System;
class Program
{
    static void Main()
    {
        int Number = 10;
        bool IsNumber10;

        if (Number == 10)
        {
            IsNumber10 = true;
        }
        else
        {
            IsNumber10 = false;
        }

        Console.WriteLine("i == 10 is {0}", IsNumber10);
    }
}
```

# Same program with ternary Operator

```
using System;

class Program
{
    static void Main()
    {
        int Number = 10;
        bool IsNumber10 = Number == 10 ? true : false;

        Console.WriteLine("i == 10 is {0}", IsNumber10);
    }
}
```

## Part 6 – Nullable types in C#

- Venkat
- PRAGIM Technologies
- [kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)
- <http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- Nullable types in C#
- Null Coalescing Operator ??

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Types in C#

**In C# types are divided into 2 broad categories**

**Value Types** -int, float, double, structs, enums etc

**Reference Types** – Interface, Class, delegates, arrays etc

**By default value types are non nullable. To make them nullable use ?**

int i = 0 (i is non nullable, so I cannot be set to null, i = null will generate compiler error)

int? j = 0 (j is nullable int, so j=null is legal)

**Nullable types bridge the differences between C# types and Database types**

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Null Coalescing Operator ??

```
using System;
class Program
{
    static void Main()
    {
        int AvailableTickets;
        int? TicketsOnSale = null;

        if (TicketsOnSale == null)
        {
            AvailableTickets = 0;
        }
        else
        {
            AvailableTickets = (int)TicketsOnSale;
        }

        Console.WriteLine("Available Tickets={0}", AvailableTickets);
    }
}
```

# Null Coalescing Operator ??

```
using System;

class Program
{
    static void Main()
    {
        int AvailableTickets;
        int? TicketsOnSale = null;

        //Using null coalesce operator ??
        AvailableTickets = TicketsOnSale ?? 0;

        Console.WriteLine("Available Tickets={0}", AvailableTickets);
    }
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 7 - Datatype conversions in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- Implicit conversions
- Explicit Conversions
- Difference between Parse() and TryParse()

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Implicit & Explicit conversion

**Implicit conversion is done by the compiler:**

1. When there is no loss of information if the conversion is done
2. If there is no possibility of throwing exceptions during the conversion

**Example:** Converting an int to a float will not lose any data and no exception will be thrown, hence an implicit conversion can be done.

Where as when converting a float to an int, we loose the fractional part and also a possibility of overflow exception. Hence, in this case an explicit conversion is required. For explicit conversion we can use cast operator or the convert class in c#

# Implicit Conversion Example

```
using System;

class Program
{
    static void Main()
    {
        int i = 100;

        // float is a bigger datatype than int. So, No loss of
        // data and Exceptions. Hence implicit conversion
        float f = i;

        Console.WriteLine(f);
    }
}
```

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Explicit Conversion Example

```
using System;
class Program
{
    static void Main()
    {
        float f = 100.25F;

        // Cannot implicitly convert float to int
        // Fractional part will be lost. Float is a
        // bigger data type than int, so there is
        // also a possibility of overflow exception
        // int i = f;

        // Use explicit Conversion using cast () operator
        int i = (int)f;

        // Or use Convert class
        // int i = Convert.ToInt32(f);

        Console.WriteLine(i);
    }
}
```

# Difference between Parse and TryParse

If the number is in a string format you have 2 options - Parse() and TryParse()

Parse() method throws an exception if it cannot parse the value, whereas TryParse() returns a bool indicating whether it succeeded or failed.

Use Parse() if you are sure the value will be valid, otherwise use TryParse()

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 8 – Arrays in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- Arrays
- Advantages and dis-advantages of arrays

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Arrays

An array is a collection of similar data types.

Examples:

```
//Initialize and Assign Values in different lines
int[] EvenNumbers = new int[3];
EvenNumbers[0] = 0;
EvenNumbers[1] = 2;
EvenNumbers[2] = 4;

//Initialize and Assign Values in the same line
int[] OddNumbers = { 1, 2, 3 };
```

**Advantages:** Arrays are strongly typed.

**Disadvantages:** Arrays cannot grow in size once initialized.

Have to rely on integral indices to store or retrieve items from the array.

## **Part 9 – Comments in C#**

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- Single line comments
- Multi line comments
- Introduction to XML documentation comments

# Comments in C#

- Single line Comments - //
- Multi line Comments - /\* \*/
- XML Documentation Comments - ///

Comments are used to document what the program does and what specific blocks or lines of code do. C# compiler ignores comments.

**To Comment and Uncomment, there are 2 ways**

1. Use the designer
2. Keyboard Shortcut: Ctrl+K, Ctrl+C and Ctrl+K, Ctrl+U

**Note:** Don't try to comment every line of code. Use comments only for blocks or lines of code that are difficult to understand

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 10 - if statement in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- if statement
- If else statement
- Difference between && and &.
- Difference between || and |.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 11 – switch statement in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- switch - statement.
- break - statement.
- goto – statement **(Always Avoid)**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Switch statement

Multiple if else statements can be replaced with a switch statement

```
if (Number == 10)
{
    Console.WriteLine("Your number is 10");
}
else if (Number == 20)
{
    Console.WriteLine("Your number is 20");
}
else if (Number == 30)
{
    Console.WriteLine("Your number is 30");
}
else
{
    Console.WriteLine("Your number is not 10, 20 & 30");
}
```

```
switch (Number)
{
    case 10:
        Console.WriteLine("Your number is 10");
        break;
    case 20:
        Console.WriteLine("Your number is 20");
        break;
    case 30:
        Console.WriteLine("Your number is 30");
        break;
    default:
        Console.WriteLine("Your number is not 10, 20 & 30");
        break;
}
```

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

```
switch(Number)
{
    case 10:
    case 20:
    case 30:
        Console.WriteLine("Your number is {0}.", Number);
        break;
    default:
        Console.WriteLine("Your number is not 10, 20 & 30");
        break;
}
```

**Note:** Case statements , with no code in-between, creates a single case for multiple values. A case without any code will automatically fall through to the next case. In this example, case 10 and case 20 will fall through and execute code for case 30.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

**break statement:** If break statement is used inside a switch statement, the control will leave the switch statement.

**goto statement:** You can either jump to another case statement, or to a specific label.

**Warning:** Using goto is bad programming style. We can should avoid goto by all means.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 12 – switch statement in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- switch - statement.
- break - statement.
- goto – statement **(Always Avoid)**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Switch statement

Multiple if else statements can be replaced with a switch statement

```
if (Number == 10)
{
    Console.WriteLine("Your number is 10");
}
else if (Number == 20)
{
    Console.WriteLine("Your number is 20");
}
else if (Number == 30)
{
    Console.WriteLine("Your number is 30");
}
else
{
    Console.WriteLine("Your number is not 10, 20 & 30");
}
```

```
switch (Number)
{
    case 10:
        Console.WriteLine("Your number is 10");
        break;
    case 20:
        Console.WriteLine("Your number is 20");
        break;
    case 30:
        Console.WriteLine("Your number is 30");
        break;
    default:
        Console.WriteLine("Your number is not 10, 20 & 30");
        break;
}
```

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

```
switch(Number)
{
    case 10:
    case 20:
    case 30:
        Console.WriteLine("Your number is {0}.", Number);
        break;
    default:
        Console.WriteLine("Your number is not 10, 20 & 30");
        break;
}
```

**Note:** Case statements , with no code in-between, creates a single case for multiple values. A case without any code will automatically fall through to the next case. In this example, case 10 and case 20 will fall through and execute code for case 30.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

**break statement:** If break statement is used inside a switch statement, the control will leave the switch statement.

**goto statement:** You can either jump to another case statement, or to a specific label.

**Warning:** Using goto is bad programming style. We can should avoid goto by all means.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

## Part 13 - Loops in C# - while loop

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- while
- do, for, foreach
- Difference between while and do loop

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# The while loop

1. While loop checks the condition first.
2. If the condition is true, statements within the loop are executed.
3. This process is repeated as long as the condition evaluates to true.

***Note: Don't forget to update the variable participating in the condition, so the loop can end, at some point***

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Do While loop

1. A do loop checks its condition at the end of the loop.
2. This means that the do loop is guaranteed to execute at least one time.
3. Do loops are used to present a menu to the user

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Difference - while & do while

1. While loop checks the condition at the beginning, where as do while loop checks the condition at the end of the loop.
2. Do loop is guaranteed to execute at least once, where as while loop is not.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

## Part 14 - Loops in C# - do while loop

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- while (Covered in Part 13)
- Do (In this session)
- for, foreach (In a later session)
- Difference between while and do loop

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# The while loop

1. While loop checks the condition first.
2. If the condition is true, statements within the loop are executed.
3. This process is repeated as long as the condition evaluates to true.

***Note: Don't forget to update the variable participating in the condition, so the loop can end, at some point***

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Do While loop

1. A do loop checks its condition at the end of the loop.
2. This means that the do loop is guaranteed to execute at least one time.
3. Do loops are used to present a menu to the user

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Difference - while & do while

- 1. While loop checks the condition at the beginning, where as do while loop checks the condition at the end of the loop.**
- 2. Do loop is guaranteed to execute at least once, where as while loop is not.**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 15 - Loops in C# - for & foreach

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- while (Covered in Part 13)
- Do (Covered in Part 14)
- for, foreach (In this session)

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# The for loop

A for loop is very similar to while loop. In a while loop we do the initialization at one place, condition check at another place, and modifying the variable at another place, whereas for loop has all of these at one place.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# foreach loop

A foreach loop is used to iterate through the items in a collection. For example, foreach loop can be used with arrays or collections such as ArrayList, HashTable and Generics. We will cover collections and generics in a later session.

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 16 - Methods in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- the structure of a method.
- the difference between static and instance methods.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Why methods

**Methods** are also called as **functions**. These terms are used interchangeably.

Methods are extremely useful because they allow you to define your **logic once**, and use it, at **many places**.

Methods make the **maintenance** of your application easier.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Methods

```
[attributes]  
access-modifiers return-type method-name ( parameters )  
{  
    Method Body  
}
```

1. We will talk about attributes and access modifiers in a later session
2. Return type can be any valid data type or void.
3. Method name can be any meaningful name
4. Parameters are optional

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Static Vs Instance methods

**When a method declaration includes a static modifier, that method is said to be a static method.**

**When no static modifier is present, the method is said to be an instance method.**

**Static method is invoked using the class name, whereas an instance method is invoked using an instance of the class.**

**The difference between instance methods and static methods is that multiple instances of a class can be created (or instantiated) and each instance has its own separate method. However, when a method is static, there are no instances of that method, and you can invoke only that one definition of the static method.**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

## Part 17 – Different types of method parameters

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- The 4 types of method parameters
- Value Parameters
- Reference Parameters
- Out Parameters
- Parameter Arrays
- Method parameters Vs Method Arguments

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Method parameter types

There are 4 different types of parameters a method can have

1. **Value Parameters** : Creates a copy of the parameter passed, so modifications does not affect each other.
2. **Reference Parameters** : The `ref` method parameter keyword on a method parameter causes a method to refer to the same variable that was passed into the method. Any changes made to the parameter in the method will be reflected in that variable when control passes back to the calling method.
3. **Out Parameters** : Use when you want a method to return more than one value.
4. **Parameter Arrays** : The `params` keyword lets you specify a method parameter that takes a variable number of arguments. You can send a comma-separated list of arguments, or an array, or no arguments. `Params` keyword should be the last one in a method declaration, and only one `params` keyword is permitted in a method declaration.

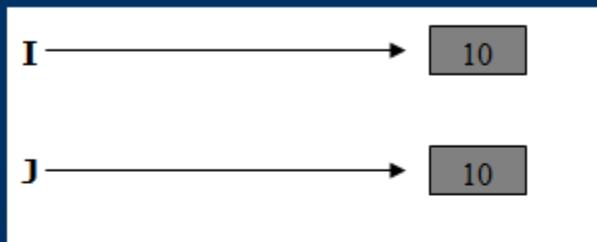
**Note : Method Parameter Vs Method Argument**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

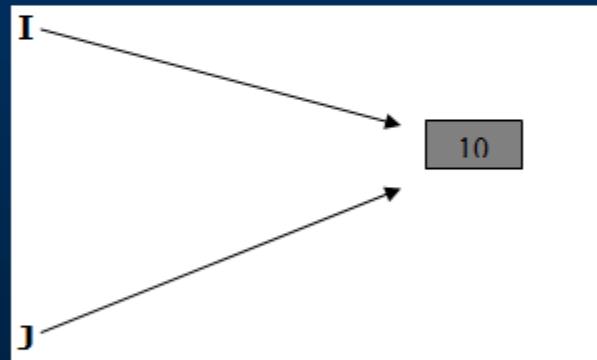
# Pass by value/reference

## Pass by value :



I and J are pointing to different memory locations. Operations one variable will not affect the value of the other variable.

## Pass by reference :



I and J are pointing to the same memory location. Operations one variable will affect the value of the other variable.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 18 – Namespaces in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Namespace basics
- Using alias directive
- Different namespace members.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Why Namespaces

Namespaces are used to organize your programs.

They also provide assistance in avoiding name clashes.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Namespaces

**Namespaces don't correspond to file, directory or assembly names. They could be written in separate files and / or separate assemblies and still belong to the same namespace.**

**Namespaces can be nested in 2 ways.**

**Namespace alias directives.** Sometimes you may encounter a long namespace and wish to have it shorter. This could improve readability and still avoid name clashes with similarly named methods.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Namespaces example

```
using System;
namespace ProjectA.TeamA
{
    class ClassA
    {
        public static void Print()
        {
            Console.WriteLine("This is Team A Print method");
        }
    }
}

namespace ProjectA.TeamB
{
    class ClassA
    {
        public static void Print()
        {
            Console.WriteLine("This is Team B Print method");
        }
    }
}
```

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Avoid ambiguity using fully qualified names

```
using System;
using ProjectA.TeamA;
using ProjectA.TeamB;
class Demo
{
    public static void Main()
    {
        //Avoid ambiguity by using the fully qualified name
        ProjectA.TeamA.ClassA.Print();
        ProjectA.TeamB.ClassA.Print();
    }
}
```

**Note : Common interview question**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

## Avoid ambiguity using alias directives

```
using System;
using PTA = ProjectA.TeamA;
using PTB = ProjectA.TeamB;
class Demo
{
    public static void Main()
    {
        //Avoid ambiguity by using alias directives
        PTA.ClassA.Print();
        PTB.ClassA.Print();
    }
}
```

**Note : Common interview question**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Different namespace members

**A namespace can contain**

- 1. Another namespace**
- 2. Class**
- 3. Interface**
- 4. struct**
- 5. enum**
- 6. delegate**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 19 - Introduction to Classes

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- What is a class
- Purpose of a class constructor
- Overloading class constructor
- Understanding this keyword
- Destructors

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# What is a class?

So far in this video tutorial we have seen simple data types like int, float, double etc. If you want to create complex custom types, then we can make use of classes.

A class consists of data and behavior. Class data is represented by its fields and behavior is represented by its methods.

An example will make this clear

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# In this session we will learn

- What is a class
- Purpose of a class constructor
- Overloading class constructor
- Understanding this keyword
- Destructors

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Purpose of a class constructor

**The purpose of a class constructor is to initialize class fields. A class constructor is automatically called when an instance of a class is created.**

**Constructors do not have return values and always have the same name as the class.**

**Constructors are not mandatory. If we do not provide a constructor ,a default parameter less constructor is automatically provided.**

**Constructors can be overloaded by the number and type of parameters.**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Destructors

**Destructors have the same name as the class with ~ symbol in front of them.**

**They don't take any parameters and do not return a value.**

**Destructors are places where you could put code to release any resources your class was holding during its lifetime.**

**We will cover this in detail in a later session:**

**They are normally called when the C# garbage collector decides to clean your object from memory.**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 20 - Static and instance class members

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

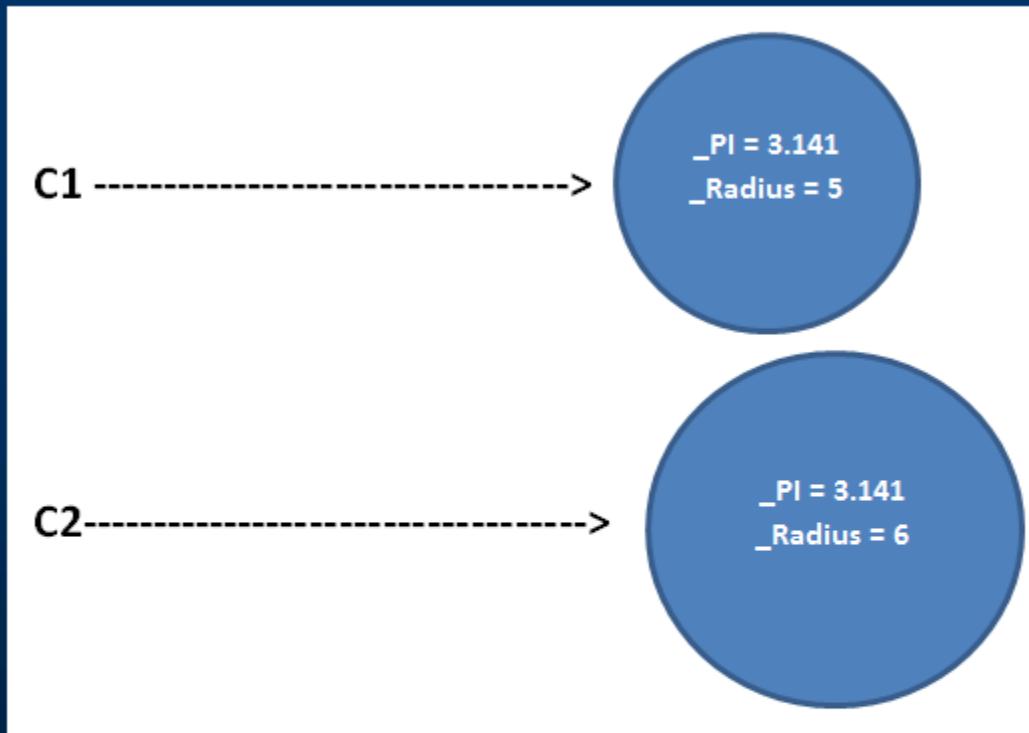
# In this session we will learn

- Static class members
- Instance class members
- Difference between static and instance members
- An example explaining when you should make certain members static

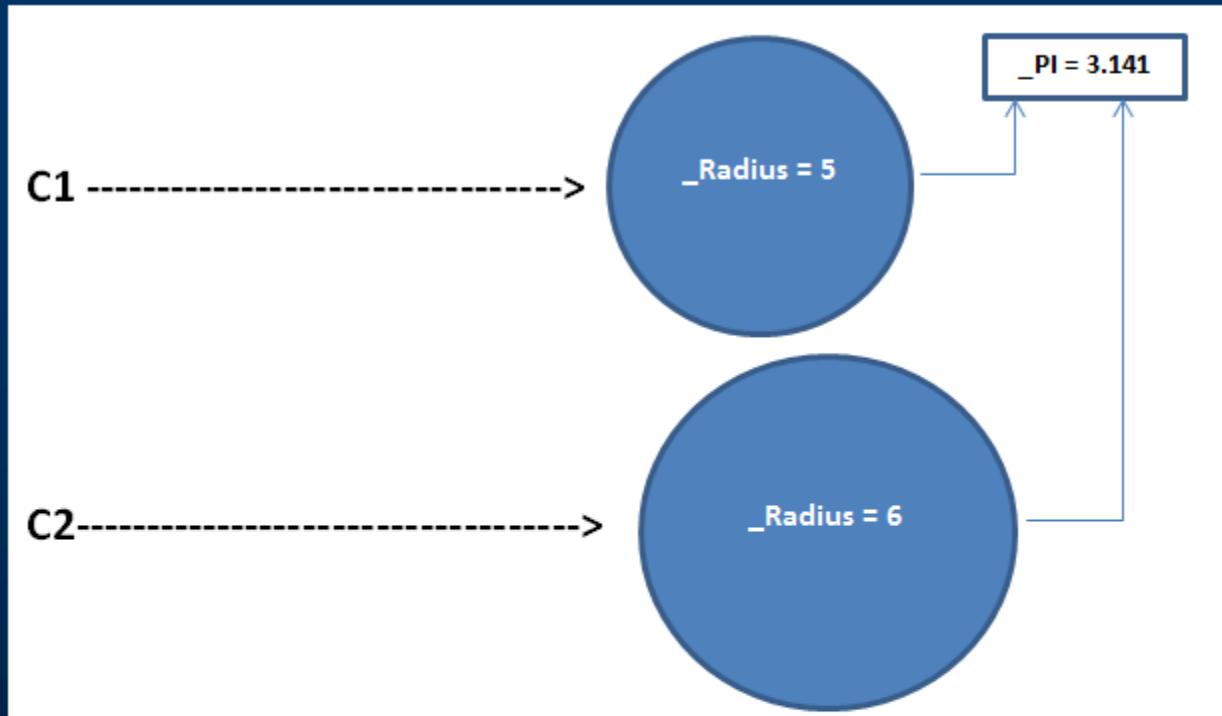
PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# If both \_PI and \_Radius are instance fields



# If \_PI is static and \_Radius is an instance field



# Static and Instance class members

When a class member includes a static modifier, the member is called as static member. When no static modifier is present the member is called as non static member or instance member.

Static members are invoked using class name, where as instance members are invoked using instances (objects) of the class.

An instance member belongs to specific instance(object) of a class. If I create 3 objects of a class, I will have 3 sets of instance members in the memory, whereas there will ever be only one copy of a static member, no matter how many instances of a class are created.

**Note:** Class members = fields, methods, properties, events, indexers, constructors.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Static constructor

**Static constructors are used to initialize *static* fields in a class.**

**You declare a *static* constructor by using the keyword *static* in front of the constructor name.**

**Static Constructor is called only once, no matter how many instances you create.**

**Static constructors are called before instance constructors**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 21 - Inheritance

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Why Inheritance
- Advantages of inheritance
- Inheritance Syntax
- Inheritance Concepts

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Why inheritance

```
public class FullTimeEmployee
{
    string FirstName;
    string LastName;
    string Email;
    float YearlySalary;

    public void PrintFullName()
    {
    }
}
```

```
public class PartTimeEmployee
{
    string FirstName;
    string LastName;
    string Email;
    float HourlyRate;

    public void PrintFullName()
    {
    }
}
```

A lot of code between these 2 classes is duplicated

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Using inheritance

```
public class Employee
{
    string FirstName;
    string LastName;
    string Email;

    public void PrintFullName()
    {
    }
}
```

**Move all the common code into base Employee class**

```
public class FullTimeEmployee
{
    float YearlySalary;
}
```

```
public class PartTimeEmployee
{
    float HourlyRate;
}
```

**FullTime and PartTime employee specific code in the respective derived classes**

# Why Inheritance

## Pillars of Object Oriented Programming

1. Inheritance
2. Encapsulation
3. Abstraction
4. Polymorphism

1. Inheritance is one of the primary pillars of object oriented programming.
2. It allows code reuse.
3. Code reuse can reduce time and errors.

***Note: You will specify all the common fields, properties, methods in the base class, which allows reusability. In the derived class you will only have fields, properties and methods, that are specific to them.***

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Inheritance Syntax

```
public class ParentClass
{
    // Parent Class Implementation
}

public class DerivedClass : ParentClass
{
    // ChildClass Implementation
}
```

- 1. In this example DerivedClass inherits from ParentClass.**
- 2. C# supports only single class inheritance.**
- 3. C# supports multiple interface inheritance.**
- 4. Child class is a specialization of base class.**
- 5. Base classes are automatically instantiated before derived classes.**
- 6. Parent Class constructor executes before Child Class constructor.**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 22 – Method hiding in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Method hiding
- Invoke hidden base class members

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Inheritance Concepts

**Use the new keyword to hide a base class member. You will get a compiler warning, if you miss the new keyword.**

**Different ways to invoke a hidden base class member from derived class**

- 1. Use base keyword**
- 2. Cast child type to parent type and invoke the hidden member**
- 3. ParentClass PC = new ChildClass()  
PC.HiddenMethod()**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 23 – Polymorphism in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Overriding virtual methods
- Polymorphism

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Polymorphism

**Polymorphism is one of the primary pillars of object-oriented programming.**

**Polymorphism allows you to invoke derived class methods through a base class reference during runtime.**

**In the base class the method is declared virtual , and in the derived class we override the same method.**

**The virtual keyword indicates, the method can be overridden in any derived class.**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 24 – Method Overriding Vs Method Hiding

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- The difference between method overriding and method hiding

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Method overriding Vs Method Hiding

```
public class BaseClass
{
    public virtual void Print()
    {
        Console.WriteLine("Base Class Print Method");
    }
}
public class DerivedClass : BaseClass
{
    public override void Print()
    {
        Console.WriteLine("Child Class Print Method");
    }
}
public class Program
{
    public static void Main()
    {
        BaseClass B = new DerivedClass();
        B.Print();
    }
}
```

```
public class BaseClass
{
    public virtual void Print()
    {
        Console.WriteLine("Base Class Print Method");
    }
}
public class DerivedClass : BaseClass
{
    public new void Print()
    {
        Console.WriteLine("Child Class Print Method");
    }
}
public class Program
{
    public static void Main()
    {
        BaseClass B = new DerivedClass();
        B.Print();
    }
}
```

In method overriding a base class reference variable pointing to a child class object, will invoke the **overridden method in the Child class**

In method hiding a base class reference variable pointing to a child class object, will invoke the **hidden method in the Base class**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 25 – Method Overloading

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Method overloading

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Method overloading

Function overloading and Method overloading terms are used interchangeably.

Method overloading allows a class to have multiple methods with the same name, but, with a different signature. So, in C# functions can be overloaded based on the number, type(int, float etc) and kind(Value, Ref or Out) of parameters.

The signature of a method consists of the name of the method and the type, kind (value, reference, or output) and the number of its formal parameters. The signature of a method does not include the return type and the params modifier. So, it is not possible to overload a function, just based on the return type or params modifier.

**Note:** If you want to know about different kinds of method parameters, please watch Part 17 - Method parameters, in this video series.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 26 – Properties

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- The need for properties

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Why Properties

Marking the class fields public and exposing to the external world is bad, as you will not have control over what gets assigned and returned.

```
public class Student
{
    public int ID;
    public string Name;
    public int PassMark;
}
public class Program
{
    public static void Main()
    {
        Student C1 = new Student();
        C1.ID = -101;
        C1.Name = null;
        C1.PassMark = -100;
        Console.WriteLine("ID = {0} & Name = {1} & PassMark = {2}",
            C1.ID, C1.Name, C1.PassMark);
    }
}
```

## Problems with Public Fields

1. ID should always be non negative number
2. Name cannot be set to NULL
3. If Student Name is missing “No Name” should be returned
4. PassMark should be read only

# Getter and Setter Methods

Programming languages that does not have properties use getter and setter methods to encapsulate and protect fields.

```
public class Student
{
    private int _id;
    public string _name;
    public int _passMark;

    public void SetId(int Id)
    {
        if (Id <= 0)
        {
            throw new Exception("Student Id should be greater than zero");
        }
        this._id = Id;
    }
    public int GetId()
    {
        return _id;
    }
}
public class Program
{
    public static void Main()
    {
        Student C1 = new Student();
        C1.SetId(101);
        Console.WriteLine("Student Id = {0}", C1.GetId());
    }
}
```

In this example we use the **SetId(int Id)** and **GetId()** methods to encapsulate **\_id** class field.

As a result, we have better control on what gets assigned and returned from the **\_id** field.

**Note:** Encapsulation is one of the primary pillars of object oriented programming.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 27 – Properties in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Read/Write Properties
- Read Only Properties
- Write Only Properties
- Auto Implemented Properties

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Properties

In C# to encapsulate and protect fields we use properties

1. We use get and set accessors to implement properties
2. A property with both get and set accessor is a **Read/Write** property
3. A property with only get accessor is a **Read only** property
4. A property with only set accessor is a **Write only** property

**Note:** The advantage of properties over traditional Get() and Set() methods is that, you can access them as if they were public fields

# Auto Implemented Properties

If there is no additional logic in the property accessors, then we can make use of auto-implemented properties introduced in C# 3.0.

Auto-implemented properties reduce the amount of code that we have to write.

When you use auto-implemented properties, the compiler creates a private, anonymous field that can only be accessed through the property's get and set accessors.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 28 - Structs in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- About structs

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Structs

Just like classes structs can have

1. Private Fields
2. Public Properties
3. Constructors
4. Methods

Object initializer syntax, introduced in C# 3.0 can be used to initialize either a struct or a class.

Note: There are several differences between classes and structs which we will be looking at in a later session.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Structs Vs Class Implementation

```
public class Customer
{
    // Private Fields
    private int _id;
    private string _name;
    // Constructor
    public Customer(int Id, string Name)
    {
        this._id = Id;
        this._name = Name;
    }
    // Public Properties
    public int ID
    {
        get { return this._id; }
        set { this._id = value; }
    }
    public string Name
    {
        get { return this._name; }
        set { this._name = value; }
    }
    // Method
    public void PrintName()
    {
        Console.WriteLine("Id = {0} && Name = {1}",
            this._id, this._name);
    }
}
```

```
public struct Customer
{
    // Private Fields
    private int _id;
    private string _name;
    // Constructor
    public Customer(int Id, string Name)
    {
        this._id = Id;
        this._name = Name;
    }
    // Public Properties
    public int ID
    {
        get { return this._id; }
        set { this._id = value; }
    }
    public string Name
    {
        get { return this._name; }
        set { this._name = value; }
    }
    // Method
    public void PrintName()
    {
        Console.WriteLine("Id = {0} && Name = {1}",
            this._id, this._name);
    }
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 29 – Difference between classes and structs

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Difference between classes and structs

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Structs

In Part 28, we understood that structs are very similar to classes. We use struct keyword to create a struct.

Just like classes structs can have fields, properties, constructors and methods.

However, there are several differences between classes and structs, which we will explore in this session.

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Classes Vs Structs

A struct is a value type where as a class is a reference type.

All the differences that are applicable to value types and reference types are also applicable to classes and structs.

Structs are stored on stack, where as classes are stored on the heap.

**Value types hold their value in memory where they are declared, but reference types hold a reference to an object in memory.**

**Value types are destroyed immediately after the scope is lost, whereas for reference types only the reference variable is destroyed after the scope is lost. The object is later destroyed by garbage collector. (We will talk about this in the garbage collection session)**

**When you copy a struct into another struct, a new copy of that struct gets created and modifications on one struct will not affect the values contained by another struct.**

**When you copy a class into another class, we only get a copy of the reference variable. Both the reference variables point to the same object on the heap. So, operations on one variable will affect the values contained by another reference variable.**

# Classes Vs Structs

Structs can't have destructors, but classes can have destructors.

Structs cannot have explicit parameter less constructor where as a class can.

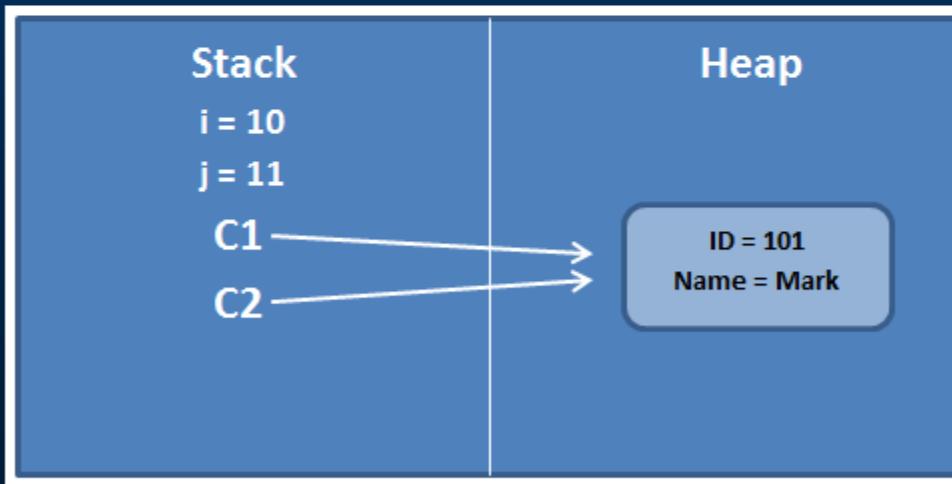
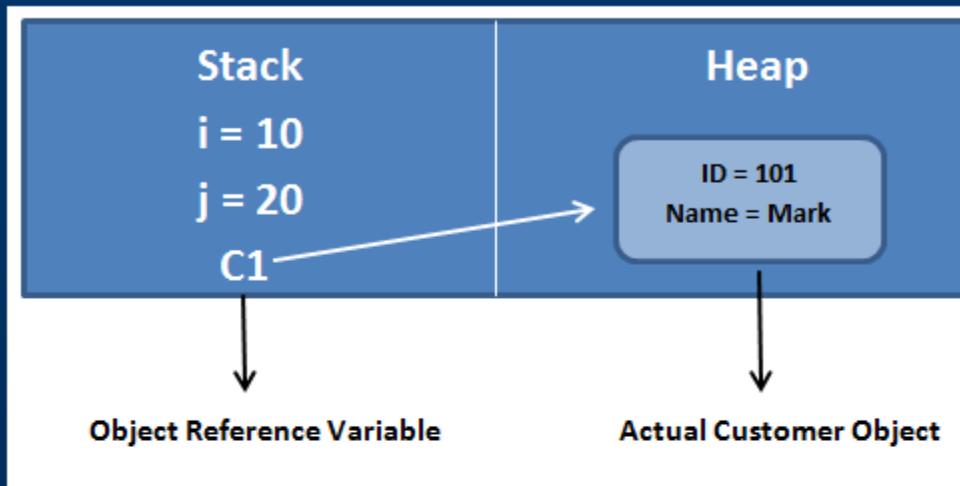
Struct can't inherit from another class where as a class can, Both structs and classes can inherit from an interface.

Examples of structs in the .NET Framework - int(System.Int32), double(System.Double) etc.

*Note 1: A class or a struct cannot inherit from another struct. Struct are sealed types.*

*Note 2: How do you prevent a class from being inherited? Or What is the significance of sealed keyword?*

# Stack and Heap



[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 30 – Introduction to interfaces

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Interface basics

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Interfaces

We create interfaces using interface keyword. Just like classes interfaces also contains properties, methods, delegates or events, but only declarations and no implementations.

**It is a compile time error to provide implementations for any interface member.**

**Interface members are public by default, and they don't allow explicit access modifiers.**

**Interfaces cannot contain fields.**

**If a class or a struct inherits from an interface, it must provide implementation for all interface members. Otherwise, we get a compiler error.**

**A class or a struct can inherit from more than one interface at the same time, but whereas, a class cannot inherit from more than once class at the same time.**

**Interfaces can inherit from other interfaces. A class that inherits this interface must provide implementation for all interface members in the entire interface inheritance chain.**

**We cannot create an instance of an interface, but an interface reference variable can point to a derived class object.**

**Interface Naming Convention: Interface names are prefixed with capital I.**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 31 - Explicit interface implementation

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Explicit interface implementation

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Explicit interface implementation

**Question:** A class inherits from 2 interfaces and both the interfaces have the same method name. How should the class implement the method for both the interfaces?

We are using Explicit Interface Implementation technique to solve this problem.

**Note:** When a class explicitly implements an interface member, the interface member can no longer be accessed thru class reference variable, but only thru the interface reference variable.

Access modifiers are not allowed on explicitly implemented interface members.

```
using System;

class Program : I1, I2
{
    static void Main()
    {
        Program P = new Program();
        ((I1)P).InterfaceMethod();
        ((I2)P).InterfaceMethod();
    }

    void I1.InterfaceMethod()
    {
        Console.WriteLine("I1 interface method implemented");
    }

    void I2.InterfaceMethod()
    {
        Console.WriteLine("I2 interface method implemented");
    }
}

interface I1
{
    void InterfaceMethod();
}

interface I2
{
    void InterfaceMethod();
}
```

# Default & Explicit Implementation

**Note: If you want to make one of the interface method, the default, then implement that method normally and the other interface method explicitly. This makes the default method to be accessible thru class instance.**



```
using System;

interface I1
{
    void InterfaceMethod();
}

interface I2
{
    void InterfaceMethod();
}

public class Program : I1, I2
{
    public static void Main()
    {
        Program P = new Program();
        P.InterfaceMethod();
        ((I2)P).InterfaceMethod();
    }

    public void InterfaceMethod()
    {
        Console.WriteLine("Interface 1 Method");
    }

    void I2.InterfaceMethod()
    {
        Console.WriteLine("Interface 2 Method");
    }
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 32 – Abstract classes

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- About abstract classes

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Abstract classes

The abstract keyword is used to create abstract classes.

An abstract class is incomplete and hence cannot be instantiated.

An abstract class can only be used as base class.

An abstract class cannot be sealed.

An abstract class may contain abstract members(methods, properties, indexers, and events), but not mandatory.

A non-abstract class derived from an abstract class must provide implementations for all inherited abstract members.

If a class inherits an abstract class, there are 2 options available for that class

**Option 1:** Provide Implementation for all the abstract members inherited from the base abstract class.

**Option 2:** If the class does not wish to provide Implementation for all the abstract members inherited from the abstract class, then the class has to be marked as abstract.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 33 – Abstract classes Vs Interfaces

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- About the differences between abstract classes and interfaces

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Abstract classes Vs Interfaces

**Abstract classes can have implementations for some of its members (Methods), but the interface can't have implementation for any of its members.**

**Interfaces cannot have fields where as an abstract class can have fields.**

**An interface can inherit from another interface only and cannot inherit from an abstract class, whereas an abstract class can inherit from another abstract class or another interface.**

**A class can inherit from multiple interfaces at the same time, whereas a class cannot inherit from multiple classes at the same time.**

**Abstract class members can have access modifiers whereas interface members cannot have access modifiers.**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 34 – Problems of multiple class inheritance

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- About the problems of multiple class inheritance

**Prerequisite:**

**Inheritance (Part 21)**

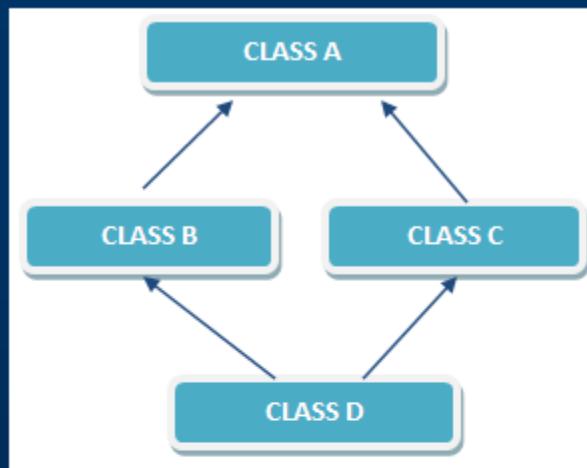
**Polymorphism (Part 23)**

**Difference between method overriding and method hiding (Part 24)**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Multiple Class Inheritance Problem



1. Class B and Class C inherit from Class A.
2. Class D inherits from both B and C.
3. If a method in D calls a method defined in A (and does not override the method), and B and C have overridden that method differently, then from which class does it inherit: B, or C?

This ambiguity is called as Diamond problem

# Multiple Class Inheritance Problem

```
using System;
class A
{
    public virtual void Print()
    {
        Console.WriteLine("Class A Implementation");
    }
}
class B : A
{
    public override void Print()
    {
        Console.WriteLine("Class B Overriding Print() Method");
    }
}
class C : A
{
    public override void Print()
    {
        Console.WriteLine("Class C Overriding Print() Method");
    }
}
class D : B, C
{
}
class Program
{
    public static void Main()
    {
        D d = new D();
        d.Print();
    }
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 35 – Multiple class inheritance using interfaces

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Implementation of multiple class inheritance using interfaces

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Multiple Class Inheritance using interfaces

```
interface IA
{
    void AMethod();
}
class A : IA
{
    public void AMethod()
    {
        Console.WriteLine("A");
    }
}

interface IB
{
    void BMethod();
}
class B : IB
{
    public void BMethod()
    {
        Console.WriteLine("B");
    }
}
```

```
class AB : IA, IB
{
    A a = new A();
    B b = new B();
    public void AMethod()
    {
        a.AMethod();
    }
    public void BMethod()
    {
        b.BMethod();
    }
}
```

```
class Program
{
    public static void Main()
    {
        AB ab = new AB();
        ab.AMethod();
        ab.BMethod();
    }
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 36 – Delegates

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- The basics of delegates

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# What is a delegate

A **delegate** is a type safe function pointer. That is, it holds a reference (Pointer) to a function.

The signature of the delegate must match the signature of the function, the delegate points to, otherwise you get a compiler error. This is the reason delegates are called as type safe function pointers.

A Delegate is similar to a class. You can create an instance of it, and when you do so, you pass in the function name as a parameter to the delegate constructor, and it is to this function the delegate will point to.

*Tip to remember delegate syntax: Delegates syntax look very much similar to a method with a delegate keyword.*

# Delegate example

```
using System;

public delegate void HelloFunctionDelegate(string Message);

class Pragim
{
    public static void Main()
    {
        HelloFunctionDelegate del = new HelloFunctionDelegate(Hello);
        del("Hello from Delegate");
    }

    public static void Hello(string strMessge)
    {
        Console.WriteLine(strMessge);
    }
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 37 – Delegates usage in c# - I

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# Demo

Demo Demo Demo

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 38 – Delegates usage in c# - II

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# Demo

Demo   Demo   Demo

# Part 39 – Multicast Delegates

Delegates basics discussed in parts 36, 37, & 38

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# Multicast Delegates

A Multicast delegate is a delegate that has references to more than one function. When you invoke a multicast delegate, all the functions the delegate is pointing to, are invoked.

There are 2 approaches to create a multicast delegate. Depending on the approach you use

+ or += to register a method with the delegate

- or -= to un-register a method with the delegate

**Note:** A multicast delegate, invokes the methods in the invocation list, in the same order in which they are added.

If the delegate has a return type other than void and if the delegate is a multicast delegate, only the value of the last invoked method will be returned. Along the same lines, if the delegate has an out parameter, the value of the output parameter, will be the value assigned by the last method.

**Common interview question - Where do you use multicast delegates?**

Multicast delegate makes implementation of observer design pattern very simple. Observer pattern is also called as publish/subscribe pattern.

*Observer design pattern - In a later session*

# Multicast Delegate Examples

```
public delegate void SampleDelegate();

public class Sample
{
    static void Main()
    {
        SampleDelegate del1 = new SampleDelegate(SampleMethodOne);
        SampleDelegate del2 = new SampleDelegate(SampleMethodTwo);
        SampleDelegate del3 = new SampleDelegate(SampleMethodThree);

        SampleDelegate del4 = del1 + del2 + del3 - del2;

        del4();
    }

    public static void SampleMethodOne()
    {
        Console.WriteLine("SampleMethodOne Invoked");
    }

    public static void SampleMethodTwo()
    {
        Console.WriteLine("SampleMethodTwo Invoked");
    }

    public static void SampleMethodThree()
    {
        Console.WriteLine("SampleMethodThree Invoked");
    }
}
```

```
public delegate void SampleDelegate();

public class Sample
{
    static void Main()
    {
        SampleDelegate del = new SampleDelegate(SampleMethodOne);
        del += SampleMethodTwo;
        del += SampleMethodThree;
        del -= SampleMethodTwo;

        del();
    }

    public static void SampleMethodOne()
    {
        Console.WriteLine("SampleMethodOne Invoked");
    }

    public static void SampleMethodTwo()
    {
        Console.WriteLine("SampleMethodTwo Invoked");
    }

    public static void SampleMethodThree()
    {
        Console.WriteLine("SampleMethodThree Invoked");
    }
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 40 – Exception Handling

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- The basics of exception handling

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Exception Handling

An exception is an unforeseen error that occurs when a program is running.

**Examples:**

Trying to read from a file that does not exist, throws **FileNotFoundException**.

Trying to read from a database table that does not exist, throws a **SqlException**.

**Showing actual unhandled exceptions to the end user is bad for two reasons**

1. Users will be annoyed as they are cryptic and does not make much sense to the end users.
2. Exceptions contain information, that can be used for hacking into your application

An exception is actually a class that derives from **System.Exception** class. The **System.Exception** class has several useful properties, that provide valuable information about the exception.

**Message:** Gets a message that describes the current exception

**Stack Trace:** Provides the call stack to the line number in the method where the exception occurred.

# Releasing System Resources

**We use try, catch and finally blocks for exception handling:**

**try** - The code that can possibly cause an exception will be in the try block.

**catch** - Handles the exception.

**finally** - Clean and free resources that the class was holding onto during the program execution. Finally block is optional.

**Note:** It is a good practice to always release resources in the finally block, because finally block is guaranteed to execute, irrespective of whether there is an exception or not.

**Specific exceptions will be caught before the base general exception, so specific exception blocks should always be on top of the base exception block. Otherwise, you will encounter a compiler error.**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 41 - Inner Exception

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- The basics of inner exceptions

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Inner Exception

**The InnerException property returns the Exception instance that caused the current exception.**

**To retain the original exception pass it as a parameter to the constructor, of the current exception**

**Always check if inner exception is not null before accessing any property of the inner exception object, else, you may get Null Reference Exception**

**To get the type of InnerException use GetType() method**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 42 – Custom Exceptions

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- When to create custom exceptions
- Creating a custom exception from the scratch
- Throwing and catching the custom exception

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Custom Exception

To understand custom exceptions, you should have good understanding of

Part 21 - Inheritance

Part 40 - Exception Handling Basics

Part 41 - Inner Exception

**When do you usually go for creating your own custom exceptions?**

If none of the already existing dotnet exceptions adequately describes the problem.

**Example:**

1. I have an asp.net web application.
2. The application should allow the user to have only one logged in session.
3. If the user is already logged in, and if he opens another browser window and tries to login again, the application should throw an error stating he is already logged in another browser window.

**With in the .NET framework we don't have any exception, that adequately describes this problem. So this scenario is one of the examples where you want to create a custom exception.**

# Custom Exception - Steps

1. Create a class that derives from System.Exception class. As a convention, end the class name with Exception suffix. All .NET exceptions end with, exception suffix.
2. Provide a public constructor, that takes in a string parameter. This constructor simply passes the string parameter, to the base exception class constructor.
3. Using InnerExceptions, you can also track back the original exception. If you want to provide this capability for your custom exception class, then overload the constructor accordingly.
4. If you want your Exception class object to work across application domains, then the object must be serializable. To make your exception class serializable mark it with Serializable attribute and provide a constructor that invokes the base Exception class constructor that takes in SerializationInfo and StreamingContext objects as parameters.

**Note:** It is also possible to provide your own custom serialization, which will discuss in a later session.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 43 – Exception handling abuse

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- About exception handling abuse

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Exception Handling abuse

Exceptions are unforeseen errors that occur when a program is running. For example, when an application is executing a query, the database connection is lost. Exception handling is generally used to handle these scenarios.

Using exception handling to implement program logical flow is bad and is termed as exception handling abuse.

## Examples

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 44 – Exception handling abuse solved

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- How to prevent exception handling abuse

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Exception Handling abuse

Exceptions are unforeseen errors that occur when a program is running. For example, when an application is executing a query, the database connection is lost. Exception handling is generally used to handle these scenarios.

Using exception handling to implement program logical flow is bad and is termed as exception handling abuse.

## Examples

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 45 – Why Enums

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Why enums are required.
- What are the problems of not using enums

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Why Enums

Enums are strongly typed constants.

If a program uses set of integral numbers, consider replacing them with enums. Otherwise the program becomes less

Readable

Maintainable

*In the next session we will replace, these integral numbers with enums, which makes the program better readable and maintainable.*

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 46 – Enums Example

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Using enum in an example
- The advantages of using enums

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Enums

**Enums are strongly typed constants.**

**If a program uses set of integral numbers, consider replacing them with enums, which makes the program more**

**Readable**

**Maintainable**

*In the next session we will discuss different concepts related to enums*

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 47 – Enums

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Enum concepts

## Prerequisite

**Part 45 – Why Enums**

**Part 46 – Enums in an example**

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Enums

If a program uses set of integral numbers, consider replacing them with enums, which makes the program more

Readable

Maintainable

1. Enums are enumerations.
2. Enums are strongly typed constants. Hence, an explicit cast is needed to convert from enum type to an integral type and vice versa. Also, an enum of one type cannot be implicitly assigned to an enum of another type even though the underlying value of their members are the same.
3. The default underlying type of an enum is int.
4. The default value for first element is ZERO and gets incremented by 1.
5. It is possible to customize the underlying type and values.
6. Enums are value types.
7. Enum keyword (all small letters) is used to create enumerations, whereas Enum class, contains static GetValues() and GetNames() methods which can be used to list Enum underlying type values and Names.

# Enums

```
customers[0] = new Customer()
{
    Name = "Mark",
    Gender = 1
};
customers[1] = new Customer()
{
    Name = "Mary",
    Gender = 2
};
customers[2] = new Customer()
{
    Name = "Sam",
    Gender = 0
};
```

```
customers[0] = new Customer
{
    Name = "Mark",
    Gender = Gender.Male
};
customers[1] = new Customer
{
    Name = "Mary",
    Gender = Gender.Female
};
customers[2] = new Customer
{
    Name = "Sam",
    Gender = Gender.Unkown
};
```

```
public static string GetGender(int gender)
{
    switch (gender)
    {
        case 0:
            return "Unknown";
        case 1:
            return "Male";
        case 2:
            return "Female";
        default:
            return "Invalid Data for Gender";
    }
}
```

```
public static string GetGender(Gender gender)
{
    switch (gender)
    {
        case Gender.Unkown:
            return "Unknown";
        case Gender.Male:
            return "Male";
        case Gender.Female:
            return "Female";
        default:
            return "Invalid data detected";
    }
}
```

# Enums

```
// Default underlying type is int and the value starts at ZERO
public enum Gender
{
    Unknown,
    Male,
    Female
}
// Gender enum underlying type is now short and the value starts at ONE
public enum Gender : short
{
    Unknown = 1,
    Male = 2,
    Female = 3
}
// Enum values need not be in sequential order Any valid underlying type value is allowed
public enum Gender : short
{
    Unknown = 10,
    Male = 22,
    Female = 35
}
// This enum will not compile, bcos the maximum value allowed for short data type is 32767.
public enum Gender : short
{
    Unknown = 10,
    Male = 32768,
    Female = 35
}
```

**Note: Use `short.MaxValue` to find out the maximum value that a `short` data type can hold**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 48 – Difference between types and type members

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will understand

- The difference between types and type members
- Organise code with regions

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Types Vs Type Members

In this example **Customer** is the **Type** and **fields, Properties and method** are **type members**.

So, in general **classes, structs, enums, interfaces, delegates** are called as **types** and **fields, properties, constructors, methods etc.,** that normally reside in a type are called as **type members**.

In C# there are 5 different access modifiers:

1. Private
2. Protected
3. Internal
4. Protected Internal
5. Public

Type members can have all the access modifiers, whereas types can have only 2 (internal, public) of the 5 access modifiers

Note: Using regions you can expand and collapse sections of your code either manually, or using visual studio **Edit-> Outlining-> Toggle All Outlining**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 49 –Access Modifiers

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- About different access modifiers that can be applied for type members
- Private, Public, Protected

## Prerequisite

### Part 48 – Difference between types and type members

## Next Session

### Internal, Protected Internal

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Private, Public & Protected

There are 5 different access modifiers in C#.

1. Private
2. Protected
3. Internal
4. Protected Internal
5. Public

Private members are available only with in the containing type, where as public members are available anywhere. There is no restriction.

Protected Members are available, with in the containing type and to the types that derive from the containing type.

Access Modifier	Accessibility
Private	Only with in the containing class
Public	Any where, No Restrictions
Protected	With in the containing types and the types derived from the containing type

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 50 -Access Modifiers

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- Internal and Protected Internal

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Internal and Protected Internal

A member with internal access modifier is available anywhere within the containing assembly. It's a compile time error to access, an internal member from outside the containing assembly.

Protected Internal members can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly. It is a combination of protected and internal. If you have understood protected and internal, this should be very easy to follow.

Access Modifier	Accessibility
Private	Only within the containing class
Public	Anywhere, No Restrictions
Protected	Within the containing types and the types derived from the containing type
Internal	Anywhere within the containing assembly
Protected Internal	Anywhere within the containing assembly, and from within a derived class in any other assembly

# Part 51 -Access Modifiers for Types

Please watch the following videos, before continuing with this part.

[Part 48 - Difference between Types and Type Members](#)

[Part 49 - Private, Public and Protected access modifiers](#)

[Part 50 - Internal and Protected Internal access modifiers](#)

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- About the Access Modifiers that can be applied for types

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Internal and Protected Internal

In c# there are 5 different access modifiers.

1. Private
2. Public
3. Protected
4. Internal
5. Protected Internal

You can use all the 5 access modifiers with type members, but types allows only internal and public access modifiers. It is a compile time error to use private, protected and protected internal access modifiers with types.

Access Modifier	Accessibility
Private	Only with in the containing class (Default for Type Members)
Public	Anywhere, No Restrictions
Protected	With in the containing types and the types derived from the containing type
Internal	Anywhere with in the containing assembly(Default for Types)
Protected Internal	Anywhere with in the containing assembly, and from within a derived class in any another assembly

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 52 -Attributes

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- The purpose of attributes
- Using an attribute
- Customizing attribute using parameters

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Attributes

Attributes allow you to add declarative information to your programs. This information can then be queried at runtime using reflection.

There are several Pre-defined Attributes provided by .NET. It is also possible to create your own Custom Attributes.

**A few pre-defined attributes with in the .NET framework:**

- Obsolete - Marks types and type members outdated
- WebMethod - To expose a method as an XML Web service method
- Serializable - Indicates that a class can be serialized

**It is possible to customize the attribute using parameters**

**An attribute is a class that inherits from System.Attribute base class.**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 53 – Reflection

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will learn

- The basics of reflection
- Understand the uses of reflection

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Reflection

**Reflection is the ability of inspecting an assemblies' metadata at runtime. It is used to find all types in an assembly and/or dynamically invoke methods in an assembly.**

**Uses of reflection:**

- 1. When you drag and drop a button on a win forms or an asp.net application. The properties window uses reflection to show all the properties of the Button class. So, reflection is extensively used by IDE or a UI designers.**
- 2. Late binding can be achieved by using reflection. You can use reflection to dynamically create an instance of a type, about which we don't have any information at compile time. So, reflection enables you to use code that is not available at compile time.**
- 3. Consider an example where we have two alternate implementations of an interface. You want to allow the user to pick one or the other using a config file. With reflection, you can simply read the name of the class whose implementation you want to use from the config file, and instantiate an instance of that class. This is another example for late binding using reflection.**

## Part 54 -Reflection Example

Pre Requisite:

Part 53 – Reflection Basics

Venkat

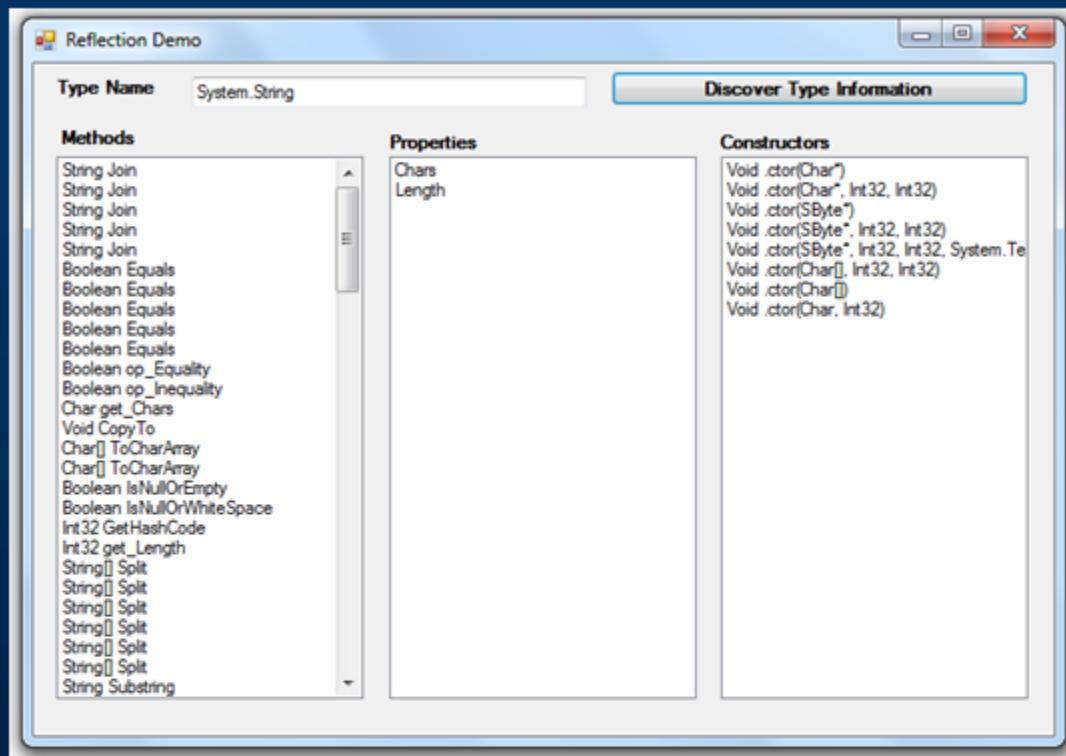
PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# Reflection Example



# Part 55 -Late binding using reflection

Pre Requisite:

Part 53 – Reflection Basics

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will understand

- Early binding and late binding
- The difference between the two approaches

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Early v/s Late Binding

```
Customer C1 = new Customer();
string fullName = C1.GetFullName("Pragim", "Tech");
Console.WriteLine("Full Name = {0}", fullName);
```

```
Assembly executingAssembly = Assembly.GetExecutingAssembly();
Type customerType = executingAssembly.GetType("Pragim.Customer");
object customerInstance = Activator.CreateInstance(customerType);
MethodInfo getFullName = customerType.GetMethod("GetFullName");
string[] methodParameters = new string[2];
methodParameters[0] = "Pragim"; //FirstName
methodParameters[1] = "Tech"; //LastName
string fullName = (string)getFullName.Invoke(customerInstance, methodParameters);
Console.WriteLine("Full Name = {0}", fullName);
```

## Difference between early and late binding:

1. Early binding can flag errors at compile time. With late binding there is a risk of run time exceptions.
2. Early binding is much better for performance and should always be preferred over late binding. Use late binding only when working with objects that are not available at compile time.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 56 – Generics

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will understand

- Generics
- Advantages of using generics

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

# Generics

**Generics are introduced in C# 2.0. Generics allow us to design classes and methods decoupled from the data types.**

**Generic classes are extensively used by collection classes available in System.Collections.Generic namespace. (Covered in the next session)**

**One way of making AreEqual() method reusable, is to use object type parameters. Since, every type in .NET directly or indirectly inherit from System.Object type, AreEqual() method works with any data type, but the problem is performance degradation due to boxing and unboxing happening.**

**Also, AreEqual() method is no longer type safe. It is now possible to pass integer for the first parameter, and a string for the second parameter. It doesn't really make sense to compare strings with integers.**

**So, the probem with using System.Object type is that**

- 1. AreEqual() method is not type safe**
- 2. Performance degradation due to boxing and unboxing.**

# Generics

To make `AreEqual()` method generic, we specify a type parameter using angular brackets as shown below.

```
public static bool AreEqual<T>(T value1, T value2)
```

At the point, When the client code wants to invoke this method, they need to specify the type, they want the method to operate on. If the user wants the `AreEqual()` method to work with integers, they can invoke the method specifying `int` as the datatype using angular brackets as shown below.

```
bool Equal = Calculator.AreEqual<int>(2, 1);
```

To operate with string data type

```
bool Equal = Calculator.AreEqual<string>("A", "B");
```

In this example, we made the method generic. Along the same lines, it is also possible to make classes, interfaces and delegates generic.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 57 – Why should we override ToString()

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will understand

- What is ToString() method
- Why should we override it.

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 58 – Why should we override Equals()

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

# In this session we will understand

- What is Equals() method
- Why should we override it.

PRAGIM Technologies | www.pragimtech.com | 9900113931

<http://csharp-video-tutorials.blogspot.com>

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 59 –Difference between Convert.ToString() and ToString()

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

**Convert.ToString() handles null, while  
ToString() doesn't, and throws a NULL  
Reference exception.**

**Depending on the type of the application,  
architecture and what you are trying to  
achieve, you choose one over the other**

# Part 60-Difference between System.String and System.Text.StringBuilder

Venkat

PRAGIM Technologies

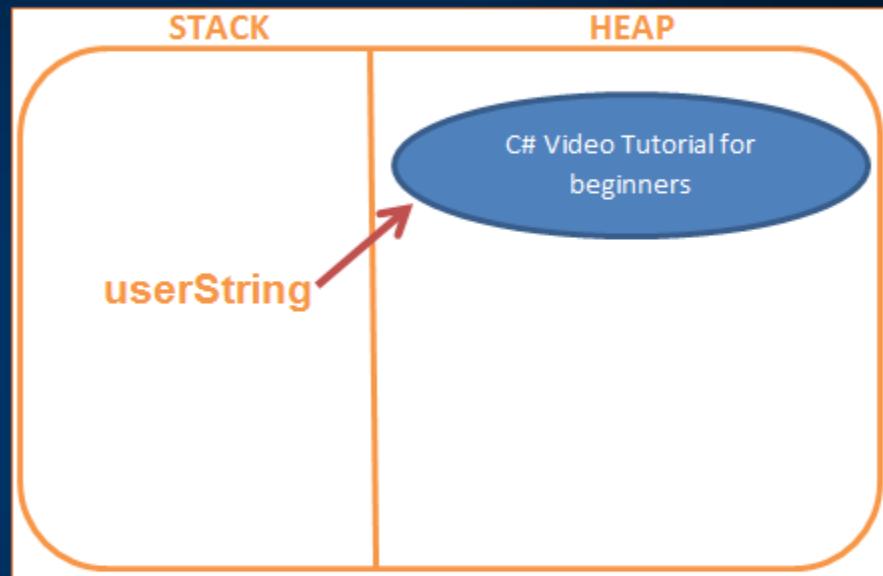
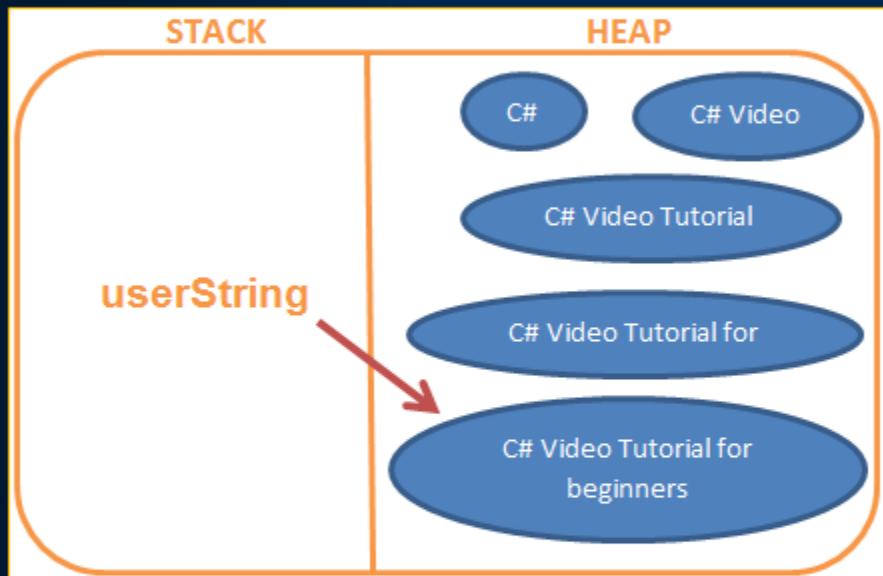
[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | [www.pragimtech.com](http://www.pragimtech.com) | 9900113931

## System.String is immutable

## StringBuilder is Mutable



As **StringBuilder** objects are mutable, they offer better performance than **string** objects of type **System.String**, when heavy string manipulation is involved.

## Part 61 - Partial Classes in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- What are partial classes?
- What are the advantages or uses of partial classes?
- Where are partial classes used?

# Partial Classes

Partial classes allow us to split a class into 2 or more files. All these parts are then combined into a single class, when the application is compiled. The partial keyword can also be used to split a struct or an interface over two or more files.

## Advantages of partial classes

1. The main advantage is that, visual studio uses partial classes to separate, automatically generated system code from the developer's code. For example, when you add a webform, two .CS files are generated

a) `WebForm1.aspx.cs` - Contains the developer code

b) `WebForm1.aspx.designer.cs` - Contains the system generated code. For example, declarations for the controls that you drag and drop on the webform.

2. When working on large projects, spreading a class over separate files allows multiple programmers to work on it simultaneously. Though, Microsoft claims this as an advantage, I haven't really seen anywhere, people using partial classes, just to work on them simultaneously.

***Next Session: Rules to keep in mind when creating partial classes***

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 62 - Creating Partial Classes

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Rules to follow when creating partial classes

## Pre-requisite:

Part 61 – Partial classes in C#

# Creating Partial Classes

1. All the parts spread across different files, must use the partial keyword.
2. All the parts spread across different files, must have the same access modifiers.
3. If any of the parts are declared abstract, then the entire type is considered abstract.
4. If any of the parts are declared sealed, then the entire type is considered sealed.
5. If any of the parts inherit a class, then the entire type inherits that class.
6. C# does not support multiple class inheritance. Different parts of the partial class, must not specify different base classes.
7. Different parts of the partial class can specify different base interfaces, and the final type implements all of the interfaces listed by all of the partial declarations.
8. Any members that are declared in a partial definition are available to all of the other parts of the partial class.

## Part 63 - Partial Methods in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- What are partial methods
- Rules to follow when creating partial methods

## Pre-requisite:

**Part 61 – Partial classes in C#**

**Part 62 – Creating Partial classes in C#**

# Partial Methods

1. A partial class or a struct can contain **partial methods**.

2. A **partial method** is created using the **partial keyword**.

3. A **partial method declaration** consists of two parts.

i) The definition (only the method signature)

ii) The implementation.

These may be in separate parts of a partial class, or in the same part.

4. The implementation for a partial method is optional. If we don't provide the implementation, the compiler removes the signature and all calls to the method.

5. Partial methods are **private by default**, and it is a **compile time error** to include any access modifiers, including **private**.

6. It is a **compile time error**, to include **declaration and implementation at the same time** for a partial method.

7. A partial method **return type must be void**. Including any other return type is a **compile time error**.

# Partial Methods

- 8. Signature of the partial method declaration, must match with the signature of the implementation.**
- 9. A partial method must be declared within a partial class or partial struct. A non partial class or struct cannot include partial methods.**
- 10. A partial method can be implemented only once. Trying to implement a partial method more than once, raises a compile time error**

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 64 - How and where are indexers used in .net

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Where are indexers used in .NET
- What are indexers in c#

## C# Tutorial

**Part 61 - Partial classes in C#**

**Part 62 - Creating Partial classes in C#**

**Part 63 - Partial methods in c#**

# Where are indexers used in .NET

To store or retrieve data from session state or application state variables, we use indexers.

```
protected void Page_Load(object sender, EventArgs e)
{
    // Using the string indexer to store session data
    Session["Session1"] = "Session 1 Data";
    // Using the string indexer to store session data
    Session["Session2"] = "Session 2 Data";

    // Using the integral indexer to retrieve data
    Response.Write("Session 1 Data = " + Session[0].ToString());
    Response.Write("<br/>");
    // Using the string indexer to retrieve data
    Response.Write("Session 2 Data = " + Session["Session2"].ToString());
}
```

# Where are indexers used in .NET

If you view the metadata of HttpSessionState class, you can see that there is an integral and string indexer defined. We use "this" keyword to create indexers in c#. We will discuss about creating indexers in our next video session.

```
...public sealed class HttpSessionState : ICollection, IEnumerable
{
    ...
    ...public int CodePage { get; set; }
    ...public HttpSessionState Contents { get; }
    ...public HttpCookieMode CookieMode { get; }
    ...public int Count { get; }
    ...public bool IsCookieless { get; }
    ...public bool IsNewSession { get; }
    ...public bool IsReadOnly { get; }
    ...public bool IsSynchronized { get; }
    ...public NameObjectCollectionBase.KeysCollection Keys { get; }
    ...public int LCID { get; set; }
    ...public SessionStateMode Mode { get; }
    ...public string SessionID { get; }
    ...public HttpStaticObjectsCollection StaticObjects { get; }
    ...public object SyncRoot { get; }
    ...public int Timeout { get; set; }

    ...
    ...public object this[int index] { get; set; }
    ...public object this[string name] { get; set; }
}
```

# Where are indexers used in .NET

Another example of indexers usage in .NET. To retrieve data from a specific column when looping thru "SqlDataReader" object, we can use either the integral indexer or string indexer.

```
string CS = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;
using (SqlConnection con = new SqlConnection(CS))
{
    SqlCommand cmd = new SqlCommand("Select * from tblEmployee", con);
    con.Open();
    SqlDataReader rdr = cmd.ExecuteReader();
    while (rdr.Read())
    {
        // Using integral indexer to retrieve Id column value
        Response.Write("Id = " + rdr[0].ToString() + " ");
        // Using string indexer to retrieve Id column value
        Response.Write("Name = " + rdr["Name"].ToString());
        Response.Write("<br/>");
    }
}
```

# Where are indexers used in .NET

Right click on **SqIDataReader** class, to view it's metadata. Notice that, there is an integral and string indexer defined.

```
... public class SqIDataReader : DbDataReader, IDataReader
{
    ... protected SqlConnection Connection { get; }
    ... public override int Depth { get; }
    ... public override int FieldCount { get; }
    ... public override bool HasRows { get; }
    ... public override bool IsClosed { get; }
    ... public override int RecordsAffected { get; }
    ... public override int VisibleFieldCount { get; }

    ... public override object this[int i] { get; }
    ... public override object this[string name] { get; }
}
```

**What are indexers in c#?**

From the above examples, it should be clear that, Indexers allow instances of a class to be indexed just like arrays.

In our next video, we will discuss about creating indexers.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 65 – Indexers in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Creating indexers

## C# Tutorial

**Part 62 - Creating Partial classes in C#**

**Part 63 - Partial methods in c#**

**Part 64 - How and where are indexers used in .net**

# Creating an Indexer

```
// Use "this" keyword to create an indexer
// This indexer takes employeeId as parameter
// and returns employee name
public string this[int employeeId]
{
    // Just like properties indexers have get and set accessors
    get
    {
        return listEmployees.
            FirstOrDefault(x => x.EmployeeId == employeeId).Name;
    }
    set
    {
        listEmployees.
            FirstOrDefault(x => x.EmployeeId == employeeId).Name = value;
    }
}
```

## Points to remember

1. Use **"this"** keyword to create an indexer
2. Just like properties indexers have **get** and **set** accessors
3. Indexers can also be **overloaded**

**Next Video:** Indexer overloading

# Using EmployeeId integral indexer

```
Company company = new Company();
Response.Write("Name of Employee with Id = 2: " + company[2]);
Response.Write("<br/>");
Response.Write("Name of Employee with Id = 5: " + company[5]);
Response.Write("<br/>");
Response.Write("Name of Employee with Id = 8: " + company[8]);

Response.Write("<br/>");
Response.Write("<br/>");

Response.Write("Changing names of employees with Id = 2,5,8");
Response.Write("<br/>");
company[2] = "Employee 2 Name Changed";
company[5] = "Employee 5 Name Changed";
company[8] = "Employee 8 Name Changed";

Response.Write("Name of Employee with Id = 2: " + company[2]);
Response.Write("<br/>");
Response.Write("Name of Employee with Id = 5: " + company[5]);
Response.Write("<br/>");
Response.Write("Name of Employee with Id = 8: " + company[8]);
```

Notice that, because of the "employeeId" indexer, I am able to use company object like an array.

## Part 66 - Overloading Indexers

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Overloading indexers

## C# Tutorial

**Part 63 - Partial methods in c#**

**Part 64 - How and where are indexers used in .net**

**Part 65 - Creating Indexers**

# Overloading Indexers

Indexers are overloaded based on the number and type of parameters.

```
public string this[string gender]
{
    get
    {
        // Returns the total count of employees whose gender matches
        // with the gender that is passed in.
        return listEmployees.Count(x => x.Gender == gender).ToString();
    }
    set
    {
        // Changes the gender of all employees whose gender matches
        // with the gender that is passed in.
        foreach (Employee employee in listEmployees)
        {
            if (employee.Gender == gender)
            {
                employee.Gender = value;
            }
        }
    }
}
```

## Part 67 - Optional Parameters

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Different ways that are available to make method parameters optional

## C# Tutorial

**Part 64 - How and where are indexers used in .net**

**Part 65 - Creating Indexers**

**Part 66 - Overloading indexers**

# Optional Parameters

There are 4 ways that can be used to make method parameters optional

1. Use parameter arrays
2. Method overloading
3. Specify parameter defaults
4. Use **OptionalAttribute** that is present in **System.Runtime.InteropServices** namespace

```
public static void AddNumbers(int firstNumber, int secondNumber, params object[] restOfTheNumbers)
{
    int result = firstNumber + secondNumber;
    foreach (int i in restOfTheNumbers)
    {
        result += i;
    }

    Console.WriteLine("Total = " + result.ToString());
}
```

Please note that, a parameter array must be the last parameter in a formal parameter list. The following function will not compile.

```
public static void AddNumbers(int firstNumber, params object[] restOfTheNumbers, int secondNumber)
{
}
```

Next Video: Method Overloading

# Part 68 - Making method parameters optional using method overloading

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Making method parameters optional using method overloading

## C# Tutorial

**Part 65 - Creating Indexers**

**Part 66 - Overloading Indexers**

**Part 67 - Optional Parameters**

# Optional Parameters

There are 4 ways that can be used to make method parameters optional

1. Use parameter arrays – Part 67

2. Method overloading – In this video

3. Specify parameter defaults

4. Use **OptionalAttribute** that is present in **System.Runtime.InteropServices** namespace

```
public static void AddNumbers(int firstNumber, int secondNumber, int[] restOfTheNumbers)
{
    int result = firstNumber + secondNumber;
    foreach (int i in restOfTheNumbers)
    {
        result += i;
    }

    Console.WriteLine("Total = " + result.ToString());
}
```

We can make the 3rd parameter optional by overloading **AddNumbers()** function

```
public static void AddNumbers(int firstNumber, int secondNumber)
{
    AddNumbers(firstNumber, secondNumber, new int[] { });
}
```

Next Video: Making method parameters optional by Specify parameter defaults

# Part 69 – Making method parameters optional by specifying parameter defaults

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Making method parameters optional by specifying parameter defaults

## C# Tutorial

**Part 66 - Overloading Indexers**

**Part 67 - Optional Parameters**

**Part 68 - Making method parameters optional using method overloading**

# Optional Parameters

There are 4 ways that can be used to make method parameters optional

1. Use parameter arrays – Part 67
2. Method overloading – Part 68
3. Specify parameter defaults – In this video
4. Use **OptionalAttribute** that is present in **System.Runtime.InteropServices** namespace

```
public static void AddNumbers(int firstNumber, int secondNumber, int[] restOfTheNumbers = null)
{
    int result = firstNumber + secondNumber;

    if (restOfTheNumbers != null)
    {
        foreach (int i in restOfTheNumbers)
        {
            result += i;
        }
    }
    Console.WriteLine("Total = " + result.ToString());
}
```

Optional parameters must appear after all required parameters.

# Named Parameters

In the following method, parameters "b" & "c" are optional.

```
public static void Test(int a, int b = 10, int c = 20)
{
    Console.WriteLine("a = " + a);
    Console.WriteLine("b = " + b);
    Console.WriteLine("c = " + c);
}
```

When we invoke this method as shown below, "1" is passed as the argument for parameter "a" and "2" is passed as the argument for parameter "b" by default.

```
Test(1, 2);
```

My intention is to pass "2" as the argument for parameter "c". To achieve this we can make use of named parameters, as shown below. Notice that, I have specified the name of the parameter for which value "2" is being passed.

```
Test(1, c: 2);
```

Next Video: Making method parameters optional by using OptionalAttribute

# Part 70 - Making method parameters optional by using OptionalAttribute

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Making method parameters optional by using OptionalAttribute

**C# Tutorial**

**Part 67 - Optional Parameters**

**Part 68 - Making method parameters optional using method overloading**

**Part 69 - Making method parameters optional by using parameter defaults**

# Optional Parameters

There are 4 ways that can be used to make method parameters optional

1. Use parameter arrays – Part 67
2. Method overloading – Part 68
3. Specify parameter defaults – Part 69
4. Use **OptionalAttribute** that is present in **System.Runtime.InteropServices** namespace

```
public static void AddNumbers(int firstNumber, int secondNumber, [Optional] int[] restOfTheNumbers)
{
    int result = firstNumber + secondNumber;

    if (restOfTheNumbers != null)
    {
        foreach (int i in restOfTheNumbers)
        {
            result += i;
        }
    }

    Console.WriteLine("Total = " + result.ToString());
}
```

**OptionalAttribute** is present in **System.Runtime.InteropServices** namespace

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 71 - Code Snippets in Visual Studio

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Code Snippets in Visual Studio

## C# Tutorial

**Part 68 - Making method parameters optional using method overloading**

**Part 69 - Making method parameters optional by using parameter defaults**

**Part 70 - Making method parameters optional by using OptionalAttribute**

# Code Snippets in Visual Studio

**Code snippets are ready-made snippets of code you can quickly insert into your code.**

- 1. Keyboard shortcut: CTRL K + X**
- 2. Right click and select "Insert Snippet...", from the context menu**
- 3. Click on Edit - Intellisense - Insert Snippet**
- 4. Use code snippets short cut. For example to use "for loop" code snippet, type "for" and press TAB key twice**

**Once a code snippet is inserted, the editable fields are highlighted in yellow, and the first editable field is selected automatically. Upon changing the first editable field, press TAB to move to the next editable field. To come to the previous editable field use SHIFT + TAB. Press ENTER or ESC keys to cancel field editing and return the Code Editor to normal.**

**Code Snippet Types:**

**Expansion:** These snippets allows the code snippet to be inserted at the cursor.

**SurroundsWith:** These snippets allows the code snippet to be placed around a selected piece of code.

**Refactoring:** These snippets are used during code refactoring.

# Surround-with Code Snippets

**Surround-with snippets surrounds the selected code, with the code snippets code.**

- 1. Select the code to surround, and use keyboard shortcut CTRL K + S**
- 2. Select the code to surround, right click and select "Surround with.." option from the context menu**
- 3. Select the code to surround, then click on Edit menu, select "IntelliSense" and then select the "Surround With" command.**

**Code snippets can be used with any type of applications that you create with visual studio. For example, you can use them with**

- 1. Console applications**
- 2. ASP.NET web applications**
- 3. ASP.NET MVC applications etc..**

**Code snippets are available for the following languages.**

- 1. C#**
- 2. Visual Basic**
- 3. XML**
- 4. HTML**
- 5. Jscript**
- 6. SQL**

# Code Snippet Manager

**Code Snippet Manager can be used to Add or remove code snippets. You can also find the following information about a code snippet.**

- 1. Description**
- 2. Shortcut**
- 3. Snippet Type**
- 4. Author**

**To access code snippet manager, click on "Tools" and then select "Code Snippet Manager". Code snippets are xml files and have .snippet extension.**

## Part 72 - What is dictionary in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Dictionary in c#

## C# Tutorial

**Part 69 - Making method parameters optional by using parameter defaults**

**Part 70 - Making method parameters optional by using OptionalAttribute**

**Part 71 - Code snippets in Visual Studio**

# Dictionary in c#

1. A dictionary is a collection of (key, value) pairs.
2. Dictionary class is present in System.Collections.Generic namespace.
3. When creating a dictionary, we need to specify the type for key and value.
4. Dictionary provides fast lookups for values using keys.
5. Keys in the dictionary must be unique.

```
Dictionary<int, Customr> dictionaryCustomers = new Dictionary<int, Customr>();  
  
Customr customr1 = new Customr() { ID = 101, Name = "Mark", Salary = 5000 };  
Customr customr2 = new Customr() { ID = 102, Name = "Pam", Salary = 7000 };  
Customr customr3 = new Customr() { ID = 104, Name = "Rob", Salary = 5500 };  
  
dictionaryCustomers.Add(customr1.ID, customr1);  
dictionaryCustomers.Add(customr2.ID, customr2);  
dictionaryCustomers.Add(customr3.ID, customr3);  
  
Customr customer101 = dictionaryCustomers[101];  
Console.WriteLine("ID = {0}, Name = {1}, Salary = {2}",  
    customer101.ID, customer101.Name, customer101.Salary);  
  
foreach (KeyValuePair<int, Customr> customerKeyValuePair in dictionaryCustomers)  
{  
    Console.WriteLine("Key = " + customerKeyValuePair.Key);  
    Customr cust = customerKeyValuePair.Value;  
    Console.WriteLine("ID = {0}, Name = {1}, Salary = {2}", cust.ID, cust.Name, cust.Salary);  
}
```

## Part 73 - What is dictionary in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Dictionary in c#

## C# Tutorial

Part 70 - Making method parameters optional by using **OptionalAttribute**

Part 71 - Code snippets in Visual Studio

Part 72 - What is dictionary in C#

# Dictionary in C#

Please watch Part 72 from the c# tutorial before proceeding with this video. This is a continuation to Part 72.

In this video, we will discuss the following methods of Dictionary class

1. TryGetValue()
2. Count()
3. Remove()
4. Clear()
5. Using LINQ extension methods with Dictionary
6. Different ways to convert an array into a dictionary

## Part 74 - List collection class in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- List collection class in C#

## **C# Tutorial**

**Part 71 - Code snippets in Visual Studio**

**Part 72 - What is dictionary in C#**

**Part 73 - What is dictionary in c# continued**

# List collection class in C#

List is one of the generic collection classes present in System.Collections.Generic namespace. There are several generic collection classes in System.Collections.Generic namespace as listed below.

1. Dictionary - Discussed in Parts 72 & 73
2. List
3. Stack
4. Queue etc

A List class can be used to create a collection of any type.

For example, we can create a list of Integers, Strings and even complex types.

The objects stored in the list can be accessed by index.

Unlike arrays, lists can grow in size automatically.

This class also provides methods to search, sort, and manipulate lists.

## Demo

## Part 75 - List collection class in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- List collection class in C#

## C# Tutorial

**Part 72 - What is dictionary in C#**

**Part 73 - What is dictionary in c# continued**

**Part 74 - List collection class in c#**

# List collection class in C#

This is continuation to Part 74. Please watch Part 74, before proceeding.

**Contains() function** - Checks if an item exists in the list. This method returns true if the item exists, else false

**Exists() function** - Checks if an item exists in the list based on a condition. This method returns true if the item exists, else false

**Find() function** - Searches for an element that matches the conditions defined by the specified lambda expression and returns the first matching item from the list

**FindLast() function** - Searches for an element that matches the conditions defined by the specified lambda expression and returns the last matching item from the list

**FindAll() function** - Returns all the items from the list that match the conditions specified by the lambda expression

# List collection class in C#

**FindIndex()** function - Returns the index of the first item, that matches the condition specified by the lambda expression. There are 2 other overloads of this method which allows us to specify the range of elements to search, within the list.

**FindLastIndex()** function - Returns the index of the last item, that matches the condition specified by the lambda expression. There are 2 other overloads of this method which allows us to specify the range of elements to search, within the list.

**Convert an array to a List** - Use **ToList()** method

**Convert a list to an array** - Use **ToArray()** method

**Convert a List to a Dictionary** - Use **ToDictionary()** method

## Part 76 - Generic list class and ranges

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

**PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)**

# In this session we will learn

- Working with generic list class and ranges

## C# Tutorial

**Part 73 - What is dictionary in c# continued**

**Part 74 - List collection class in c#**

**Part 75 - List collection class in c# continued**

# Working with Generic list class & Ranges

Please watch Part 74 & 75 before proceeding with this video

**AddRange()**- **Add()** method allows you to add one item at a time to the end of the list, whereas **AddRange()** allows you to add another list of items, to the end of the list.

**GetRange()**- Using an item index, we can retrieve only one item at a time from the list, if you want to get a list of items from the list, then use **GetRange()** function. This function expects 2 parameters, i.e the start index in the list and the number of elements to return.

**InsertRange()** - **Insert()** method allows you to insert a single item into the list at a specified index, whereas **InsertRange()** allows you, to insert another list of items to your list at the specified index.

**RemoveRange()** - **Remove()** function removes only the first matching item from the list. **RemoveAt()** function, removes the item at the specified index in the list. **RemoveAll()** function removes all the items that matches the specified condition. **RemoveRange()** method removes a range of elements from the list. This function expects 2 parameters, i.e the start index in the list and the number of elements to remove. If you want to remove all the elements from the list without specifying any condition, then use **Clear()** function.

## Part 77 - Sort a list of simple types

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Sorting a list of simple types

## C# Tutorial

**Part 74 - List collection class in c#**

**Part 75 - List collection class in c# continued**

**Part 76 - Working with generic list class and ranges in c#**

# Sorting a list of simple types

Please watch Part 76 before proceeding with this video

<http://www.youtube.com/user/kudvenkat/videos?view=1>

Sorting a list of simple types like int, string, char etc, is straight forward. Just invoke the sort() method on the list instance and the data will be automatically sorted in ascending order.

```
List<int> numbers = new List<int> { 1, 8, 7, 5, 2, 3, 4, 9, 6 };
numbers.Sort();
```

If you want the data to be retrieved in descending order, use Reverse() method on the list instance.

```
numbers.Reverse();
```

However, when you do the same thing on a complex type like Customer, we get a runtime invalid operation exception - Failed to compare 2 elements in the array. This because, .NET runtime does not know, how to sort complex types. We have to tell the way we want data to be sorted in the list by implementing IComparable interface.

How is the sort functionality working for simple types like int, string, char etc?

That is because these types (int, string, decimal, char etc) have implemented IComparable interface already.

## Part 78 - Sort a list of complex types

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Sorting a list of complex types

## C# Tutorial

**Part 75 - List collection class in c# continued**

**Part 76 - Working with generic list class and ranges in c#**

**Part 77 - Sort a list of simple types in c#**

# Sorting a list of complex types

To sort a list of complex types without using LINQ, the complex type has to implement **IComparable** interface and provide implementation for **CompareTo()** method. **CompareTo()** method returns an integer, and the meaning of the return value is shown below. **RETURN VALUE** is

**Greater than ZERO** - The current instance is greater than the object being compared with.

**Less than ZERO** - The current instance is less than the object being compared with.  
**is ZERO** - The current instance is equal to the object being compared with.

Alternatively you can also invoke **CompareTo()** method. **Salary** property of the **Customer** object is **int**. **CompareTo()** method is already implemented on **int** type, so we can invoke this method and return its value.



```
public class Customer : IComparable<Customer>
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int Salary { get; set; }

    public int CompareTo(Customer obj)
    {
        //if (this.Salary > obj.Salary)
        //    return 1;
        //else if (this.Salary < obj.Salary)
        //    return -1;
        //else
        //    return 0;

        return this.Salary.CompareTo(obj.Salary);
    }
}
```

# Sorting a list of complex types

If you prefer not to use the Sort functionality provided by the class, then you can provide your own, by implementing IComparer interface. For example, if I want the customers to sorted by name instead of salary.

## Step 1: Implement IComparer interface

```
public class SortByName : IComparer<Customer>
{
    public int Compare(Customer x, Customer y)
    {
        return x.Name.CompareTo(y.Name);
    }
}
```

## Step 2: Pass an instance of the class that implements IComparer interface, as an argument to the Sort() method.

```
SortByName sortByName = new SortByName();
listCustomers.Sort(sortByName);
```

# Part 79 - Sort a list of complex types using Comparison delegate

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Sorting a list of complex types using Comparison delegate

## C# Tutorial

**Part 76 - Working with generic list class and ranges in c#**

**Part 77 - Sort a list of simple types in c#**

**Part 78 - Sort a list of complex types in c#**

# Using Comparison delegate with List<T>

Please watch Part 78 from C# Tutorial video series, before proceeding.

<http://www.youtube.com/user/kudvenkat/videos?view=1>

One of the overloads of the Sort() method in List class expects Comparison delegate to be passed as an argument.

```
public void Sort(Comparison<T> comparison);
```

**Approach 1:**

**Step 1:** Create a function whose signature matches the signature of System.Comparison delegate. This is the method where we need to write the logic to compare 2 customer objects.

```
private static int CompareCustomers(Customer c1, Customer c2)
{
    return c1.ID.CompareTo(c2.ID);
}
```

**Step 2:** Create an instance of System.Comparison delegate, and then pass the name of the function created in Step1 as the argument. So, at this point "Comparison" delegate is pointing to our function that contains the logic to compare 2 customer objects.

```
Comparison<Customer> customerComparer = new Comparison<Customer>(CompareCustomers);
```

**Step 3:** Pass the delegate instance as an argument, to Sort() function.

```
listCutomers.Sort(customerComparer);
```

# Sorting a list of complex types

## Approach 2:

In Approach 1 this is what we have done

1. Created a private function that contains the logic to compare customers
2. Created an instance of Comparison delegate, and then passed the name of the private function to the delegate.
3. Finally passed the delegate instance to the Sort() method.

Do we really have to follow all these steps. Isn't there any other way?

The above code can be simplified using delegate keyword as shown below.

```
listCustomers.Sort(delegate(Customer c1, Customer c2)
{
    return (c1.ID.CompareTo(c2.ID));
});
```

Approach 3: The code in Approach 2, can be further simplified using lambda expression as shown below.

```
listCustomers.Sort((c1, c2) => c1.ID.CompareTo(c2.ID));
```

## Part 80 - Some useful methods of List collection class

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Some useful methods of List collection class

## C# Tutorial

**Part 77 - Sort a list of simple types in c#**

**Part 78 - Sort a list of complex types in c#**

**Part 79 - Sort a list of complex types using Comparison delegate**

# Useful methods of List collection class

Please watch Part 79 from C# Tutorial video series, before proceeding.

<http://www.youtube.com/user/kudvenkat/videos?view=1>

In this video, we will discuss the following methods

**TrueForAll()** - Returns true or false depending on whether if every element in the list matches the conditions defined by the specified predicate.

**AsReadOnly()** - Returns a read-only wrapper for the current collection. Use this method, if you don't want the client code to modify the collection i.e add or remove any elements from the collection. The ReadOnlyCollection will not have methods to add or remove items from the collection. You can only read items from this collection.

**TrimExcess()** - Sets the capacity to the actual number of elements in the List, if that number is less than a threshold value.

**According to MSDN:**

This method can be used to minimize a collection's memory overhead if no new elements will be added to the collection. The cost of reallocating and copying a large List<T> can be considerable. So the TrimExcess method does nothing if the list is at more than 90 percent of capacity. This avoids incurring a large reallocation cost for a relatively small gain. The current threshold is 90 percent, but this could change in the future.

## Part 81 - When to use a Dictionary over List in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- When to use a dictionary over list in c#

## C# Tutorial

**Part 78 - Sort a list of complex types in c#**

**Part 79 - Sort a list of complex types using Comparison delegate**

**Part 80 - Some useful methods of List collection class**

# When to use a Dictionary over List

Please watch Part 80 from C# Tutorial video series, before proceeding.

<http://www.youtube.com/user/kudvenkat/videos?view=1>

**Find() method of the List class loops thru each object in the list until a match is found. So, if you want to lookup a value using a key dictionary is better for performance over list. So, use dictionary when you know the collection will be primarily used for lookups.**

```
cmd C:\Windows\system32\cmd.exe
Please enter country code
IND
Name = INDIA Capital =New Delhi
Do you want to continue - YES or NO?
TEST GHY
Do you want to continue - YES or NO?
yes
Please enter country code
GBR
Name = UNITED KINGDOM Capital =London
Do you want to continue - YES or NO?
no
Press any key to continue . . .
```

Country Code:	<input type="text" value="TEST"/>
Name:	<input type="text"/>
Capital:	<input type="text"/>
Country code not found	

Country Code:	<input type="text" value="GBR"/>
Name:	<input type="text" value="UNITED KINGDOM"/>
Capital:	<input type="text" value="London"/>

## Part 82 - Generic Queue collection class

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Generic Queue collection class

## C# Tutorial

**Part 79 - Sort a list of complex types using Comparison delegate**

**Part 80 - Some useful methods of List collection class**

**Part 81 - When to use a dictionary over list**

# When to use a Dictionary over List

C#, ASP.NET, ADO.NET, Dotnet basics, MVC and SQL Server video tutorial playlists

<http://www.youtube.com/user/kudvenkat/videos?view=1>

Queue is a generic FIFO (First In First Out) collection class that is present in System.Collections.Generic namespace. The Queue collection class is analogous to a queue at the ATM machine to withdraw money. The order in which people queue up, will be the order in which they will be able to get out of the queue and withdraw money from the ATM. The Queue collection class operates in a similar fashion. The first item to be added (enqueued) to the queue, will be the first item to be removed (dequeued) from the Queue.

To add items to the end of the queue, use Enqueue() method.

To remove an item that is present at the beginning of the queue, use Dequeue() method.

A foreach loop iterates thru the items in the queue, but will not remove them from the queue.

To check if an item, exists in the queue, use Contains() method.

What is the difference between Dequeue() and Peek() methods?

Dequeue() method removes and returns the item at the beginning of the queue, whereas Peek() returns the item at the beginning of the queue, without removing it.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 83 - Generic Stack collection class

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Generic stack collection class

## C# Tutorial

**Part 80 - Some useful methods of List collection class**

**Part 81 - When to use a dictionary over list**

**Part 82 - Generic Queue collection class**

# Generic Stack collection class

Stack is a generic LIFO (Last In First Out) collection class that is present in System.Collections.Generic namespace. The Stack collection class is analogous to a stack of plates. If you want to add a new plate to the stack of plates, you place it on top of all the already existing plates. If you want to remove a plate from the stack, you will first remove the one that you have last added. The stack collection class also operates in a similar fashion. The last item to be added (pushed) to the stack, will be the first item to be removed (popped) from the stack.



To insert an item at the top of the stack, use Push() method.

To remove and return the item that is present at the top of the stack, use Pop() method.

A foreach loop iterates thru the items in the stack, but will not remove them from the stack. The items from the stack are retrieved in LIFO (Last In First Out), order. The last element added to the Stack is the first one to be returned.

To check if an item exists in the stack, use Contains() method.

What is the difference between Pop() and Peek() methods?

Pop() method removes and returns the item at the top of the stack, whereas Peek() returns the item at the top of the stack, without removing it.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 84 - Real time example of queue collection class in c#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Real time example where a queue collection class can be used

## C# Tutorial

**Part 81 - When to use a dictionary over list**

**Part 82 - Generic Queue collection class**

**Part 83 - Generic stack collection class**

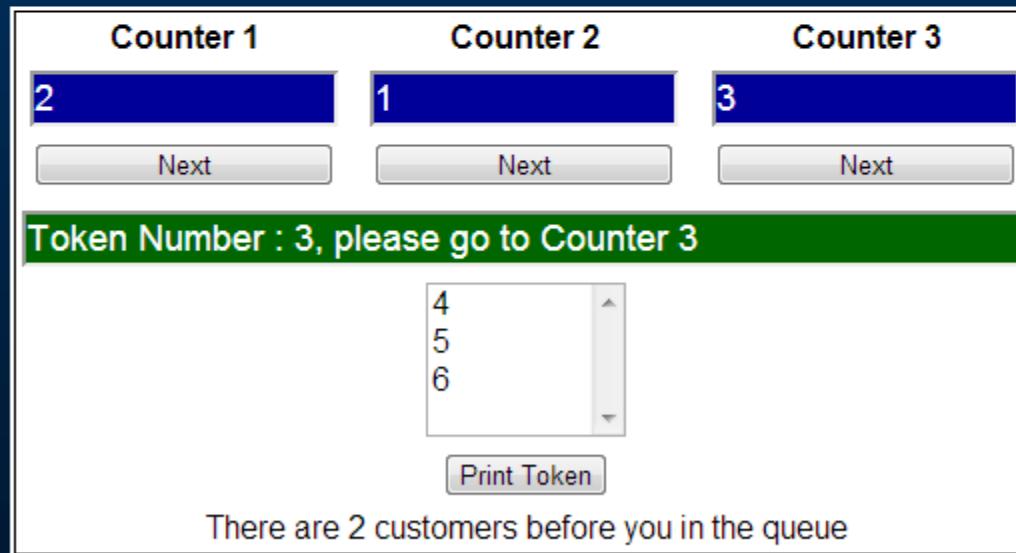
# Real time example - Queue

When you walk into a bank or a passport office, you will collect a token and wait in the queue for your token number to be called.

From the application perspective, when a token is issued, the token number will be added to the end of the Queue.

When a representative at the counter is available to server a customer, he will push the "Next" button and the token number that is present at the beginning of the queue, will be dequeued.

So, this is one example, where a Queue collection class can be effectively used.



## Part 85 - Real time example of Stack collection class in c#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Real time example where a Stack collection class can be used

## C# Tutorial

**Part 82 - Generic Queue collection class**

**Part 83 - Generic stack collection class**

**Part 84 - Real time example of queue collection class in c#**

# Real time example - Stack

Two common scenarios, where a stack can be used

1. Implementing UNDO functionality
2. Implementing browser back button

## Links

- [WebForm1](#)
- [WebForm2](#)
- [WebForm3](#)
- [WebForm4](#)

This is WebForm4

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 86 - Multithreading in C#

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- What is a process and a thread
- Simple multithreading example

## C# Tutorial

**Part 83 - Generic stack collection class**

**Part 84 - Real time example of queue collection class in c#**

**Part 85 - Real time example of stack collection class in c#**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

<http://www.youtube.com/user/kudvenkat/playlists>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](http://www.facebook.com/pragimtech)

<http://csharp-video-tutorials.blogspot.com>

# Multithreading in C#

Link for C#, ASP.NET, SQL Server, WCF & MVC tutorial

<http://www.youtube.com/user/kudvenkat/playlists>

## What is a Process:

Process is what the operating system uses to facilitate the execution of a program by providing the resources required. Each process has a unique process Id associated with it. You can view the process within which a program is being executed using windows task manager.

## What is Thread:

Thread is a light weight process. A process has at least one thread which is commonly called as main thread which actually executes the application code. A single process can have multiple threads.

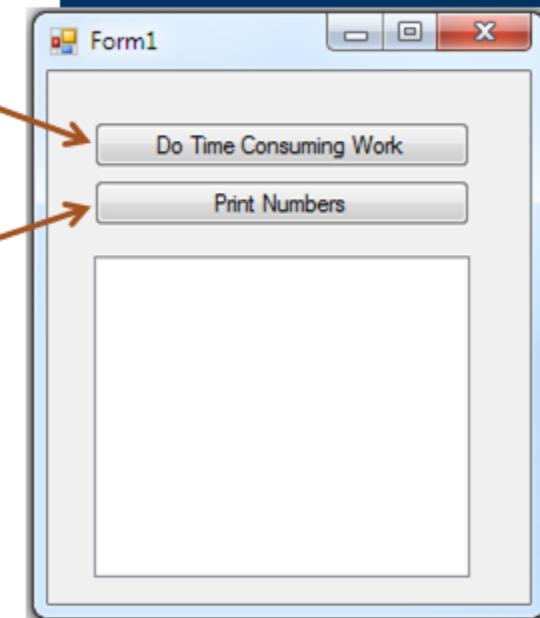
**Please Note:** All the threading related classes are present in System.Threading namespace.

# Multithreading in C#

```
private void btnTimeConsumingWork_Click(object sender, EventArgs e)
{
    btnTimeConsumingWork.Enabled = false;
    btnPrintNumbers.Enabled = false;

    DoTimeConsumingWork();

    btnTimeConsumingWork.Enabled = true;
    btnPrintNumbers.Enabled = true;
}
private void DoTimeConsumingWork()
{
    Thread.Sleep(5000);
}
private void btnPrintNumbers_Click(object sender, EventArgs e)
{
    for (int i = 1; i <= 10; i++)
    {
        listBoxNumbers.Items.Add(i);
    }
}
```



Next video: Advantages and disadvantages of multithreaded programming

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 87 - Advantages & Disadvantages of multithreading

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Advantages and disadvantages of multithreading

## C# Tutorial

**Part 84 - Real time example of queue collection class in c#**

**Part 85 - Real time example of stack collection class in c#**

**Part 86 - Multithreading in C#**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# Advantages & Disadvantages

Link for C#, ASP.NET, SQL Server, WCF & MVC tutorial

<http://www.youtube.com/user/kudvenkat/playlists>

## Advantages of multithreading:

1. To maintain a responsive user interface
2. To make efficient use of processor time while waiting for I/O operations to complete
3. To split large, CPU-bound tasks to be processed simultaneously on a machine that has multiple processors/cores

## Disadvantages of multithreading:

1. On a single processor/core machine threading can affect performance negatively as there is overhead involved with context-switching
2. Have to write more lines of code to accomplish the same task
3. Multithreaded applications are difficult to write, understand, debug and maintain

**Please Note:** Only use multithreading when the advantages of doing so outweigh the disadvantages.

## Part 88 - ThreadStart Delegate

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- The purpose of ThreadStart delegate

## C# Tutorial

**Part 85 - Real time example of stack collection class in c#**

**Part 86 - Multithreading in C#**

**Part 87 - Advantages and disadvantages of multithreading**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# ThreadStart Delegate

Link for C#, ASP.NET, SQL Server, WCF & MVC tutorial

<http://www.youtube.com/user/kudvenkat/playlists>

```
class Program
{
    public static void Main()
    {
        Thread T1 = new Thread(Number.PrintNumbers);
        T1.Start();
    }
}

class Number
{
    public static void PrintNumbers()
    {
        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

To create a THREAD, create an instance of Thread class and to it's constructor pass the name of the function that we want the thread to execute.

# ThreadStart Delegate

```
Thread T1 = new Thread(Number.PrintNumbers);  
T1.Start();
```

We can rewrite the above line using ThreadStart delegate as shown below

```
Thread T1 = new Thread(new ThreadStart(Number.PrintNumbers));  
T1.Start();
```

**Why a delegate need to be passed as a parameter to the Thread class constructor?**  
The purpose of creating a Thread is to execute a function. A delegate is a type safe function pointer, meaning it points to a function that the thread has to execute. In short, all threads require an entry point to start execution. Any thread you create will need an explicitly defined entry point i.e a pointer to the function where they should begin execution. So threads always require a delegate.

**In the code below, we are not explicitly creating the ThreadStart delegate, then how is it working here?**

```
Thread T1 = new Thread(Number.PrintNumbers);  
T1.Start();
```

**It's working in spite of not creating the ThreadStart delegate explicitly because the framework is doing it automatically for us.**

# ThreadStart Delegate

We can also rewrite the same line using delegate() keyword

```
Thread T1 = new Thread(delegate() { Number.PrintNumbers(); });
```

The same line rewritten using lambda expression

```
Thread T1 = new Thread(() => Number.PrintNumbers());
```

```
class Program
{
    public static void Main()
    {
        Number number = new Number();
        Thread T1 = new Thread(number.PrintNumbers);
        T1.Start();
    }
}
```

```
class Number
{
    public void PrintNumbers()
    {
        for (int i = 1; i <= 10; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

Thread function need not be a static function always. It can also be a non-static function. In case of non-static function we have to create an instance of the class.

## Part 89 - ParameterizedThreadStart Delegate

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- The purpose of ParameterizedThreadStart delegate

## C# Tutorial

**Part 86 - Multithreading in C#**

**Part 87 - Advantages and disadvantages of multithreading**

**Part 88 - ThreadStart Delegate**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# ParameterizedThreadStart Delegate

```
class Number
{
    public static void PrintNumbers(object target)
    {
        int number = 0;
        if (int.TryParse(target.ToString(), out number))
        {
            for (int i = 1; i <= number; i++)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

```
class Program
```

```
{
    public static void Main()
    {
        Console.WriteLine("Please enter the target number");
        object target = Console.ReadLine();

        ParameterizedThreadStart parameterizedThreadStart =
            new ParameterizedThreadStart(Number.PrintNumbers);
        Thread T1 = new Thread(parameterizedThreadStart);
        T1.Start(target);
    }
}
```

## Use

ParameterizedThreadStart  
delegate to pass data to  
the thread function

# ParameterizedThreadStart Delegate

The code in the Main() function can also be written as shown below

```
public static void Main()
{
    Console.WriteLine("Please enter the target number");
    object target = Console.ReadLine();

    Thread T1 = new Thread(Number.PrintNumbers);
    T1.Start(target);
}
```

Here we are not explicitly creating an instance of ParameterizedThreadStart delegate. Then how is it working?

It's working because, the compiler implicitly converts

```
new Thread(Number.PrintNumbers);
```

TO

```
new Thread(new ParameterizedThreadStart(Number.PrintNumbers));
```

When to use ParameterizedThreadStart over ThreadStart delegate?

Use ParameterizedThreadStart delegate if you have some data to pass to the Thread function, otherwise just use ThreadStart delegate.

# ParameterizedThreadStart Delegate

**Please note:** Using ParameterizedThreadStart delegate and Thread.Start(Object) method to pass data to the Thread function is not type safe as they operate on object datatype and any type of data can be passed.

If you try to change the data type of the target parameter of PrintNumbers() function from object to int, a compiler error will be raised as the signature of PrintNumbers() function does not match with the signature of ParameterizedThreadStart delegate.

**Next Video:** Passing data to the Thread function without loosing the type saftey

# Part 90 - Passing data to the Thread function in a type safe manner

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Passing data to the Thread function in a type safe manner

## C# Tutorial

**Part 87 - Advantages and disadvantages of multithreading**

**Part 88 - ThreadStart Delegate**

**Part 89 - ParameterizedThreadStart delegate**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

<http://www.youtube.com/user/kudvenkat/playlists>

# Passing data to the Thread function

```
class Number
{
    int _target;
    public Number(int target)
    {
        this._target = target;
    }

    public void PrintNumbers()
    {
        for (int i = 1; i <= _target; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

To pass data to the Thread function in a type safe manner, encapsulate the thread function and the data it needs in a helper class and use the ThreadStart delegate to execute the thread function.

**Next Video: Retrieving data from Thread function using callback method**

```
class Program
{
    public static void Main()
    {
        Console.WriteLine("Please enter the target number");
        int target = Convert.ToInt32(Console.ReadLine());

        Number number = new Number(target);
        Thread T1 = new Thread(new ThreadStart(number.PrintNumbers));
        T1.Start();
    }
}
```

# Part 91 - Retrieving data from Thread function using callback method

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Retrieving data from Thread function using callback method

## C# Tutorial

**Part 88 - ThreadStart Delegate**

**Part 89 - ParameterizedThreadStart delegate**

**Part 90 - Passing data to the Thread function in a type safe manner**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# Callback method to get data from thread

## Example:

Please enter the target number

5

Sum of numbers is 15

Main thread retrieves the target number from the user.

Main thread creates a child thread and pass the target number to the child thread.

The child thread computes the sum of numbers and then returns the sum to the Main thread using callback function.

The callback method prints the sum of numbers

**Step 1: Create a callback delegate. The actual callback method signature should match with the signature of this delegate.**

```
public delegate void SumOfNumbersCallback(int sumOfNumbers);
```

# Callback method to get data from thread

**Step 2: Create a helper class to compute the sum of numbers and to call the callback method.**

```
class Number
{
    int _target;
    SumOfNumbersCallback _callbackMethod;

    public Number(int target, SumOfNumbersCallback callbackMethod)
    {
        this._target = target;
        this._callbackMethod = callbackMethod;
    }

    public void ComputeSumOfNumbers()
    {
        int sum = 0;
        for (int i = 1; i <= _target; i++)
        {
            sum = sum + i;
        }
        if (_callbackMethod != null)
            _callbackMethod(sum);
    }
}
```

# Callback method to get data from thread

**Step 3: This class consumes the Number class created in Step 2**

```
class Program
{
    public static void PrintSumOfNumbers(int sum)
    {
        Console.WriteLine("Sum of numbers is " + sum);
    }

    public static void Main()
    {
        Console.WriteLine("Please enter the target number");
        int target = Convert.ToInt32(Console.ReadLine());

        SumOfNumbersCallback callbackMethod = new SumOfNumbersCallback(PrintSumOfNumbers);

        Number number = new Number(target, callbackMethod);

        Thread T1 = new Thread(new ThreadStart(number.ComputeSumOfNumbers));
        T1.Start();
    }
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 92 - Thread.Join & Thread.IsAlive

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Significance of Thread.Join and Thread.IsAlive functions

## C# Tutorial

**Part 89 - ParameterizedThreadStart delegate**

**Part 90 - Passing data to the Thread function in a type safe manner**

**Part 91 - Retrieving data from Thread function using callback method**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# Thread.Join & Thread.IsAlive functions

**Join** blocks the current thread and makes it wait until the thread on which **Join** method is invoked completes. **Join** method also has a overload where we can specify the timeout. If we don't specify the timeout the calling thread waits indefinitely, until the thread on which **Join()** is invoked completes. This overloaded **Join(int millisecondsTimeout)** method returns boolean. True if the thread has terminated otherwise false.

**Join** is particularly useful when we need to wait and collect result from a thread execution or if we need to do some clean-up after the thread has completed.

**IsAlive** returns boolean. True if the thread is still executing otherwise false.

## Part 93 - Protecting shared resources

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

**PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)**

# In this session we will learn

- What happens if shared resources are not protected from concurrent access in multithreaded program
- How to protect shared resources from concurrent access

## C# Tutorial

**Part 90 - Passing data to the Thread function in a type safe manner**

**Part 91 - Retrieving data from Thread function using callback method**

**Part 92 - Significance of Thread.Join and Thread.IsAlive functions**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# Protecting Shared Resources

What happens if shared resources are not protected from concurrent access in multithreaded program?

The output or behaviour of the program can become inconsistent if the shared resources are not protected from concurrent access in multithreaded program

```
class Program
{
    static int Total = 0;
    public static void Main()
    {
        AddOneMillion();
        AddOneMillion();
        AddOneMillion();
        Console.WriteLine("Total = " + Total);
    }

    public static void AddOneMillion()
    {
        for (int i = 1; i <= 1000000; i++)
        {
            Total++;
        }
    }
}
```

This program is a single-threaded program. In the Main() method, AddOneMillion() method is called 3 times, and it updates the Total field correctly as expected, and finally prints the correct total i.e 3M.

# Protecting Shared Resources

Everytime we run the above program, we get a different output. The inconsistent output is because the Total field which is a shared resource is not protected from concurrent access by multiple threads. The operator ++ is not thread safe.

```
class Program
{
    static int Total = 0;
    public static void Main()
    {
        Thread thread1 = new Thread(Program.AddOneMillion);
        Thread thread2 = new Thread(Program.AddOneMillion);
        Thread thread3 = new Thread(Program.AddOneMillion);

        thread1.Start(); thread2.Start(); thread3.Start();
        thread1.Join(); thread2.Join(); thread3.Join();

        Console.WriteLine("Total = " + Total);
    }
    public static void AddOneMillion()
    {
        for (int i = 1; i <= 1000000; i++)
        {
            Total++;
        }
    }
}
```

```
C:\ThreadingExample\ThreadingExample\bin\Debug>
Total = 2273015
C:\ThreadingExample\ThreadingExample\bin\Debug>
Total = 2078843
C:\ThreadingExample\ThreadingExample\bin\Debug>
Total = 2137500
C:\ThreadingExample\ThreadingExample\bin\Debug>
Total = 2157744
C:\ThreadingExample\ThreadingExample\bin\Debug>
Total = 3000000
```

# Protecting Shared Resources

**Using Interlocked.Increment() method: Increments a specified variable and stores the result, as an atomic operation.**

```
public static void AddOneMillion()
{
    for (int i = 1; i <= 1000000; i++)
    {
        Total++;
    }
}
```

```
public static void AddOneMillion()
{
    for (int i = 1; i <= 1000000; i++)
    {
        Interlocked.Increment(ref Total);
    }
}
```

## Locking:

```
static object _lock = new object();

public static void AddOneMillion()
{
    for (int i = 1; i <= 1000000; i++)
    {
        lock (_lock)
        {
            Total++;
        }
    }
}
```

# Protecting Shared Resources

**Which option is better?**

**From a performance perspective using Interlocked class is better over using locking.**  
**Locking locks out all the other threads except a single thread to read and increment the Total variable. This will ensure that the Total variable is updated safely. The downside is that since all the other threads are locked out, there is a performance hit.**

**The Interlocked class can be used with addition/subtraction (increment, decrement, add, etc.) on and int or long field. The Interlocked class has methods for incrementing, decrementing, adding, and reading variables atomically.**

# Protecting Shared Resources

The following code prints the time taken in ticks. 1 millisecond consists of 10000 ticks.

```
public static void Main()
{
    Stopwatch stopwatch = Stopwatch.StartNew();

    Thread thread1 = new Thread(Program.AddOneMillion);
    Thread thread2 = new Thread(Program.AddOneMillion);
    Thread thread3 = new Thread(Program.AddOneMillion);

    thread1.Start(); thread2.Start(); thread3.Start();

    thread1.Join(); thread2.Join(); thread3.Join();

    Console.WriteLine("Total = " + Total);

    stopwatch.Stop();

    Console.WriteLine("Time Taken in Ticks = " + stopwatch.ElapsedTicks);
}
```

Please Note: You can use the TimeSpan object to find ticks per second, ticks per millisecond etc.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 94 - Monitor v/s lock

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Difference between Monitor and lock in C#

## C# Tutorial

**Part 91 - Retrieving data from Thread function using callback method**

**Part 92 - Significance of Thread.Join and Thread.IsAlive functions**

**Part 93 - Protecting shared resources in multithreading**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

<http://www.youtube.com/user/kudvenkat/playlists>

# Monitor v/s lock

Both Monitor class and lock provides a mechanism that synchronizes access to objects. lock is the shortcut for Monitor.Enter with try and finally

```
static object _lock = new object();
public static void AddOneMillion()
{
    for (int i = 1; i <= 1000000; i++)
    {
        lock (_lock)
        {
            Total++;
        }
    }
}
```

==

```
static object _lock = new object();
public static void AddOneMillion()
{
    for (int i = 1; i <= 1000000; i++)
    {
        Monitor.Enter(_lock);
        try
        {
            Total++;
        }
        finally
        {
            Monitor.Exit(_lock);
        }
    }
}
```

# Monitor v/s lock

In C# 4, it is implemented slightly differently

```
static object _lock = new object();
public static void AddOneMillion()
{
    for (int i = 1; i <= 1000000; i++)
    {
        bool lockTaken = false;

        Monitor.Enter(_lock, ref lockTaken);
        try
        {
            Total++;
        }
        finally
        {
            if (lockTaken)
                Monitor.Exit(_lock);

        }
    }
}
```

So, in short, lock is a shortcut and it's the option for the basic usage. If you need more control to implement advanced multithreading solutions using TryEnter(), Wait(), Pulse(), & PulseAll() methods, then the Monitor class is your option.

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 95 - Deadlocks

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Why and how a deadlock can occur in multithreading

## C# Tutorial

**Part 92 - Significance of Thread.Join and Thread.IsAlive functions**

**Part 93 - Protecting shared resources in multithreading**

**Part 94 - Difference between Monitor and lock in C#**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# Deadlocks

Link for C#, ASP.NET, SQL Server, WCF & MVC tutorials

<http://www.youtube.com/user/kudvenkat/videos?view=1>

When a deadlock can occur

Let's say we have 2 threads

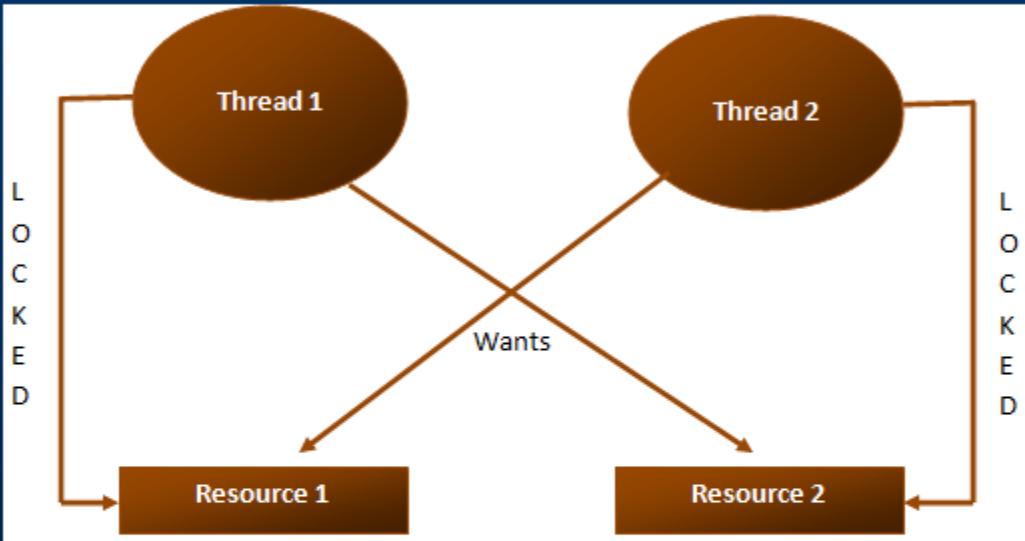
a) Thread 1

b) Thread 2

and 2 resources

a) Resource 1

b) Resource 2



Thread 1 has already acquired a lock on Resource 1 and wants to acquire a lock on Resource 2. At the same time Thread 2 has already acquired a lock on Resource 2 and wants to acquire a lock on Resource 1.

Two threads never give up their locks, hence a deadlock

# Deadlocks

```
public class Account
{
    double _balance; int _id;
    public Account(int id, double balance)
    {
        this._id = id; this._balance = balance;
    }
    public int ID
    {
        get { return _id; }
    }
    public void Withdraw(double amount)
    {
        _balance -= amount;
    }
    public void Deposit(double amount)
    {
        _balance += amount;
    }
}
```

# Deadlocks

```
public class AccountManager
{
    Account _fromAccount; Account _toAccount; double _amountToTransfer;
    public AccountManager(Account fromAccount, Account toAccount, double amountToTransfer)
    {
        this._fromAccount = fromAccount;
        this._toAccount = toAccount;
        this._amountToTransfer = amountToTransfer;
    }
    public void Transfer()
    {
        lock (_fromAccount)
        {
            Thread.Sleep(1000);
            lock (_toAccount)
            {
                _fromAccount.Withdraw(_amountToTransfer);
                _toAccount.Deposit(_amountToTransfer);
            }
        }
    }
}
```

# Deadlocks

```
public static void Main()
{
    Console.WriteLine("Main Started");
    Account accountA = new Account(101, 5000);
    Account accountB = new Account(102, 3000);

    AccountManager accountManagerA = new AccountManager(accountA, accountB, 1000);
    Thread T1 = new Thread(accountManagerA.Transfer);
    T1.Name = "T1";

    AccountManager accountManagerB = new AccountManager(accountB, accountA, 2000);
    Thread T2 = new Thread(accountManagerB.Transfer);
    T2.Name = "T2";

    T1.Start();
    T2.Start();

    T1.Join();
    T2.Join();
    Console.WriteLine("Main Completed");
}
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 96 - Resolving Deadlocks

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Resolving a deadlock in a multithreaded program

## C# Tutorial

**Part 93 - Protecting shared resources in multithreading**

**Part 94 - Difference between Monitor and lock in C#**

**Part 95 - Deadlock in a multithreaded program**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

<http://www.youtube.com/user/kudvenkat/playlists>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](http://www.facebook.com/pragimtech)

<http://csharp-video-tutorials.blogspot.com>

# Resolving Deadlocks

Link for C#, ASP.NET, SQL Server, WCF & MVC tutorials

<http://www.youtube.com/user/kudvenkat/videos?view=1>

There are several techniques to avoid and resolve deadlocks. For example

1. Acquiring locks in a specific defined order
2. Mutex class
3. Monitor.TryEnter() method

In this video, we will discuss, acquiring locks in a specific defined order to resolve a deadlock.

# Resolving Deadlocks

```
public void Transfer()
{
    lock (_fromAccount)
    {
        Thread.Sleep(1000);
        lock (_toAccount)
        {
            _fromAccount.Withdraw(_amount);
            _toAccount.Deposit(_amountToT
        }
    }
}
```

```
public void Transfer()
{
    object _lock1, _lock2;
    if (_fromAccount.ID < _toAccount.ID)
    {
        _lock1 = _fromAccount; _lock2 = _toAccount;
    }
    else
    {
        _lock1 = _toAccount; _lock2 = _fromAccount;
    }
    lock (_lock1)
    {
        Thread.Sleep(1000);
        lock (_lock2)
        {
            _fromAccount.Withdraw(_amountToTransfer);
            _toAccount.Deposit(_amountToTransfer);
        }
    }
}
```

```
AccountManager accountManagerA = new AccountManager(accountA, accountB, 1000);
Thread T1 = new Thread(accountManagerA.Transfer);
```

```
AccountManager accountManagerB = new AccountManager(accountB, accountA, 2000);
Thread T2 = new Thread(accountManagerB.Transfer);
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 97 - Performance implications of a multithreaded program

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Performance implications of a multithreaded program when run on a single core/processor machine versus multi core/processor machine

## C# Tutorial

**Part 94 - Difference between Monitor and lock in C#**

**Part 95 - Deadlock in a multithreaded program**

**Part 96 - How to resolve a deadlock in a multithreaded program**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

<http://www.youtube.com/user/kudvenkat/playlists>

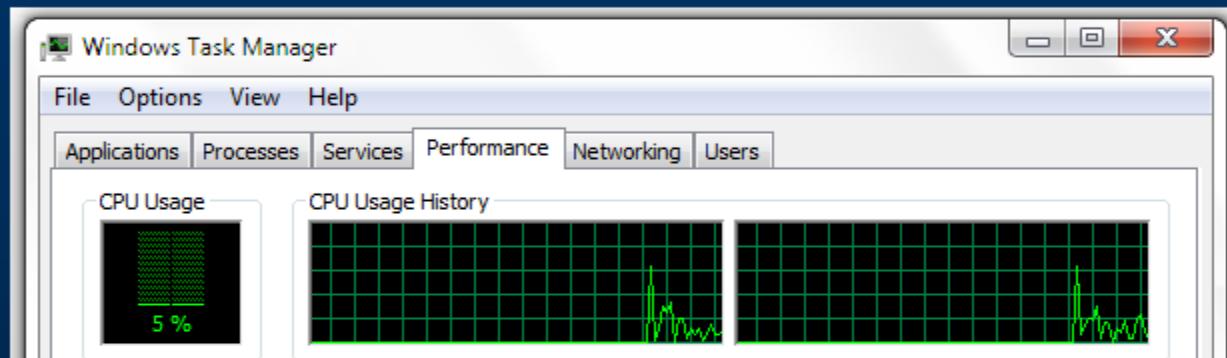
# Performance Implications

Link for C#, ASP.NET, SQL Server, WCF & MVC tutorials

<http://www.youtube.com/user/kudvenkat/videos?view=1>

How to find out how many processors you have on your machine

## 1. Using Task Manager



## 2. Use the following code in any .net application

```
Environment.ProcessorCount
```

## 3. On the windows command prompt window, type the following

```
echo %NUMBER_OF_PROCESSORS%
```

# Performance Implications

On a machine that has multiple processors, multiple threads can execute application code in parallel on different cores. For example, if there are two threads and two cores, then each thread would run on an individual core. This means, performance is better.

If two threads take 10 milli-seconds each to complete, then on a machine with 2 processors, the total time taken is 10 milli-seconds.

On a machine that has a single processor, multiple threads execute, one after the other or wait until one thread finishes. It is not possible for a single processor system to execute multiple threads in parallel. Since the operating system switches between the threads so fast, it just gives us the illusion that the threads run in parallel. On a single core/processor machine multiple threads can affect performance negatively as there is overhead involved with context-switching.

If two threads take 10 milli-seconds each to complete, then on a machine with 1 processor, the total time taken is  
20 milli-seconds + (Thread context switching time, if any)

## Part 98 - Anonymous Methods

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Anonymous methods

## C# Tutorial

**Part 95 - Deadlock in a multithreaded program**

**Part 96 - How to resolve a deadlock in a multithreaded program**

**Part 97 - Performance of a multithreaded program**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# Anonymous Methods

## What is an anonymous method?

**Anonymous method is a method without a name. Introduced in C# 2, they provide us a way of creating delegate instances without having to write a separate method.**

```
// Step 1: Create a method whose signature matches
// with the signature of Predicate<Employee> delegate
private static bool FindEmployee(Employee Emp)
{
    return Emp.ID == 102;
}

// Step 2: Create an instance of Predicate<Employee> delegate and pass the
// method name as an argument to the delegate constructor
Predicate<Employee> predicateEmployee = new Predicate<Employee>(FindEmployee);

// Step 3: Now pass the delegate instance as the argument for Find() method
Employee employee = listEmployees.Find(x => predicateEmployee(x));
Console.WriteLine("ID = {0}, Name {1}", employee.ID, employee.Name);

// Anonymous method is being passed as an argument to the Find() method
// This anonymous method replaces the need for Step 1, 2 and 3
employee = listEmployees.Find(delegate(Employee x) { return x.ID == 102; });
Console.WriteLine("ID = {0}, Name {1}", employee.ID, employee.Name);
```

# Anonymous Methods

Subscribing for an event handler is another example

```
private void Form1_Load(object sender, EventArgs e)
{
    Button Button1 = new Button();
    Button1.Text = "Click Me";
    Button1.Click += new EventHandler(Button1_Click);
    this.Controls.Add(Button1);
}

void Button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Button Clicked");
}
```

Anonymous Method

```
private void Form1_Load(object sender, EventArgs e)
{
    Button Button1 = new Button();
    Button1.Text = "Click Me";
    Button1.Click += delegate(object obj, EventArgs eventArgs)
    {
        MessageBox.Show("Button Clicked");
    };
    this.Controls.Add(Button1);
}
```

# Anonymous Methods

Link for C#, ASP.NET, SQL Server, WCF & MVC tutorials

<http://www.youtube.com/user/kudvenkat/videos?view=1>

With anonymous Methods delegate parameters are optional. This means the below code

```
Button1.Click += delegate(object obj, EventArgs eventArgs)
{
    MessageBox.Show("Button Clicked");
};
```

can be rewritten as shown below

```
Button1.Click += delegate
{
    MessageBox.Show("Button Clicked");
};
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

## Part 99 - Lambda Expressions

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

PRAGIM Technologies | 9900113931 | [www.pragimtech.com](http://www.pragimtech.com) | [www.facebook.com/pragimtech](https://www.facebook.com/pragimtech)

# In this session we will learn

- Lambda Expressions

## C# Tutorial

**Part 96 - How to resolve a deadlock in a multithreaded program**

**Part 97 - Performance of a multithreaded program**

**Part 98 - Anonymous methods in c#**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# Lambda Expressions

## What are Lambda expressions?

Anonymous methods and Lambda expressions are very similar. Anonymous methods were introduced in C# 2 and Lambda expressions in C# 3.

### To find an employee with Id = 102, using anonymous method

```
Employee employee = listEmployees.Find(delegate(Employee Emp) { return Emp.ID == 102; });
```

### To find an employee with Id = 102, using lambda expression

```
Employee employee = listEmployees.Find(Emp => Emp.ID == 102);
```

### You can also explicitly specify the Input type but not required

```
Employee employee = listEmployees.Find((Employee Emp) => Emp.ID == 102);
```

=> is called lambda operator and read as GOES TO. Notice that with a Lambda expression you don't have to use the delegate keyword explicitly and don't have to specify the input parameter type explicitly. The parameter type is inferred. Lambda expressions are more convenient to use than anonymous methods. Lambda expressions are particularly helpful for writing LINQ query expressions.

# Lambda Expressions

In most of the cases Lambda expressions supersedes anonymous methods. To my knowledge, the only time I prefer to use anonymous methods over lambdas is, when we have to omit the parameter list when it's not used within the body.

Anonymous methods allow the parameter list to be omitted entirely when it's not used within the body, whereas with lambda expressions this is not the case.

For example, with anonymous method notice that we have omitted the parameter list as we are not using them within the body

```
Button1.Click += delegate
{
    MessageBox.Show("Button Clicked");
};
```

The above code can be rewritten using lambda expression as shown below. Notice that with lambda we cannot omit the parameter list.

```
Button1.Click += (eventSender, eventAgrs) =>
{
    MessageBox.Show("Button Clicked");
};
```

[www.PragimTech.com](http://www.PragimTech.com)

Pragim@PragimTech.com

# Part 100 - Func Delegate

Venkat

PRAGIM Technologies

[kudvenkat@gmail.com](mailto:kudvenkat@gmail.com)

<http://csharp-video-tutorials.blogspot.com>

[PRAGIM Technologies | 9900113931 | www.pragimtech.com | www.facebook.com/pragimtech](#)

# In this session we will learn

- Purpose of Func<T, TResult> delegate in c#

## C# Tutorial

**Part 97 - Performance of a multithreaded program**

**Part 98 - Anonymous methods in c#**

**Part 99 - Lambda expressions**

**Link to Dot Net Basics, ASP.NET, C#, ADO.NET and SQL Server video series**

**<http://www.youtube.com/user/kudvenkat/playlists>**

# Func Delegate

What is Func<T, TResult> in C#?

In simple terms, Func<T, TResult> is just a generic delegate. Depending on the requirement, the type parameters (T and TResult) can be replaced with the corresponding type arguments.

For example, Func<Employee, string> is a delegate that represents a function expecting Employee object as an input parameter and returns a string.

```
public class Employee
{
    public int ID { get; set; }
    public string Name { get; set; }
}
```

```
List<Employee> listEmployees = new List<Employee>()
{
    new Employee{ ID = 101, Name = "Mark"},
    new Employee{ ID = 102, Name = "John"},
    new Employee{ ID = 103, Name = "Mary"},
};
```

```
Func<Employee, string> selector = employee => "Name = " + employee.Name;
IEnumerable<string> names = listEmployees.Select(selector);
```

```
foreach (string name in names)
{
    Console.WriteLine(name);
}
```

```
Name = Mark
Name = John
Name = Mary
```

```
// Lambda expression can also be used to achieve the same thing
IEnumerable<string> names = listEmployees.Select(employee => "Name = " + employee.Name);
```

# Func Delegate

**What is the difference between Func delegate and lambda expression?**

**They're the same, just two different ways to write the same thing. The lambda syntax is newer, more concise and easy to write.**

**What if I have to pass two or more input parameters?**

**As of this recording there are 17 overloaded versions of Func, which enables us to pass variable number and type of input parameters. In the example below, Func<int, int, string> represents a function that expects 2 int input parameters and returns a string.**

```
class Program
{
    public static void Main()
    {
        Func<int, int, string> funcDelegate = (firstNumber, secondNumber) =>
            "Sum = " + (firstNumber + secondNumber).ToString();

        string result = funcDelegate(10, 20);
        Console.WriteLine(result);
    }
}
```