



WEB ENGINEERING

.Net

Week 2 - Days 10 & 11



: Basics

Recap Task

Review questions will be asked to the participants for review:

- The students will Create their First ASP.Net MVC Application



Introduction

Learning Objectives

By the end of this session, the students will have developed an understanding of:

- ▶ What is C#
- ▶ C# Syntax
- ▶ Abstraction, Polymorphism, Encapsulation
- ▶ Enumerations & Overriding
- ▶ Interfaces
- ▶ Classes & Structures
- ▶ LINQ





What is C#?

C# : What is it?

- C# (pronounced "See Sharp") is a modern, object-oriented, and type-safe programming language. C# enables developers to build many types of secure and robust applications that run in .NET. C# has its roots in the C family of languages
- C# (C-Sharp) is a programming language developed by Microsoft that runs on the .NET Framework.
- C# is used to develop web apps, desktop apps, mobile apps, games and much more.

C# : What is it?

- C# is an object-oriented, component-oriented programming language.
- C# provides language constructs to directly support these concepts, making C# a natural language in which to create and use software components.
- Since its origin, C# has added features to support new workloads and emerging software design practices.
- At its core, C# is an object-oriented language.
- You define types and their behavior.

C# : What is it?

- C# programs run on .NET, a virtual execution system called the common language runtime (CLR) and a set of class libraries.
- The CLR is the implementation by Microsoft of the common language infrastructure (CLI), an international standard.
- The CLI is the basis for creating execution and development environments in which languages and libraries work together seamlessly.
- As C# is close to C, C++ and Java, it makes it easy for programmers to switch to C# or vice versa

C# is used for:

- Mobile applications
- Desktop applications
- Web applications
- Web services
- Web sites
- Games
- VR
- Database applications
- And much, much more!



C# Get Started

C# IDE

- The easiest way to get started with C#, is to use an IDE.
- An IDE (Integrated Development Environment) is used to edit and compile code.
- In our course, we will use Visual Studio Community, which is free to download from <https://visualstudio.microsoft.com/vs/community/> .
- Applications written in C# use the .NET Framework, so it makes sense to use Visual Studio, as the program, the framework, and the language, are all created by Microsoft.



C# Install & Create a New Project



C# Syntax

C# - Basic Syntax

- C# is an object-oriented programming language. In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions. The actions that an object may take are called methods. Objects of the same kind are said to have the same type or, are said to be in the same class.
- For example, let us consider a Rectangle object. It has attributes such as length and width. Depending upon the design, it may need ways for accepting the values of these attributes, calculating the area, and displaying details.

C# - Basic Syntax

Let us look at implementation of a Rectangle class and discuss C# basic syntax –

Try writing this code on your computers and then [check the output in the live demo here.](#)

```
using System;

namespace RectangleApplication {
    class Rectangle {

        // member variables
        double length;
        double width;

        public void Acceptdetails() {
            length = 4.5;
            width = 3.5;
        }
        public double GetArea() {
            return length * width;
        }
        public void Display() {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }
    }
    class ExecuteRectangle {
        static void Main(string[] args) {
            Rectangle r = new Rectangle();
            r.Acceptdetails();
            r.Display();
            Console.ReadLine();
        }
    }
}
```

The *using* Keyword

The first statement in any C# program is

```
using System;
```

The using keyword is used for including the namespaces in the program. A program can include multiple using statements.

The *class* Keyword

The class keyword is used for declaring a class.

Comments in C#

Comments are used for explaining code. Compilers ignore the comment entries. The multiline comments in C# programs start with /* and terminates with the characters */ as shown below –

```
/* This program demonstrates  
The basic syntax of C# programming  
Language */
```

Single-Line Comments

Single-line comments are indicated by the '//' symbol. For example,

```
}//end class Rectangle
```

Try this example code
and [check output here](#):

Example

```
// This is a comment  
Console.WriteLine("Hello World!");
```

Multi-Line Comments

- Multi-line comments start with `/*` and ends with `*/`.
- Any text between `/*` and `*/` will be ignored by C#.
- This example uses a multi-line comment (a comment block) to explain the code:

Try this example code
and [check output here](#):

Example

```
/* The code below will print the words Hello World  
to the screen, and it is amazing */  
Console.WriteLine("Hello World!");
```

C# Exercises

Open these links and complete the practice exercises on your computers:

1. [C# Syntax Exercise](#)
2. [C# Comments Exercise](#)



C# Variables

Types of Variables

Variables are containers for storing data values.

In C#, there are different types of variables (defined with different keywords), for example:

int - stores integers (whole numbers), without decimals, such as 123 or -123

double - stores floating point numbers, with decimals, such as 19.99 or -19.99

char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes

string - stores text, such as "Hello World". String values are surrounded by double quotes

bool - stores values with two states: true or false



C# OOP

C# - What is OOP?

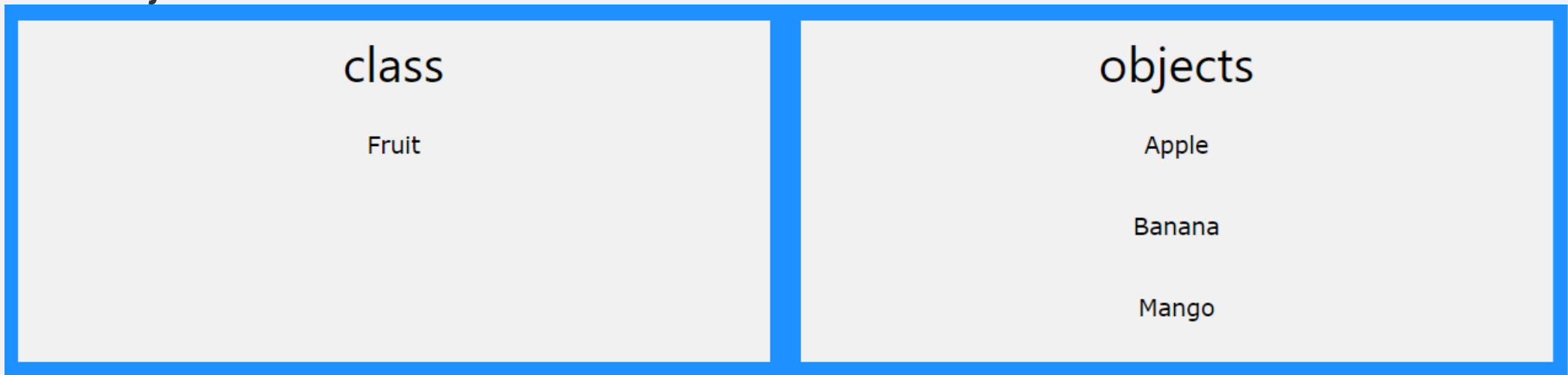
C# is an object-oriented programming language. The four basic principles of object-oriented programming are:

- **Abstraction:** Modeling the relevant attributes and interactions of entities as classes to define an abstract representation of a system.
- **Encapsulation:** Hiding the internal state and functionality of an object and only allowing access through a public set of functions.
- **Inheritance:** Ability to create new abstractions based on existing abstractions.
- **Polymorphism:** Ability to implement inherited properties or methods in different ways across multiple abstractions.

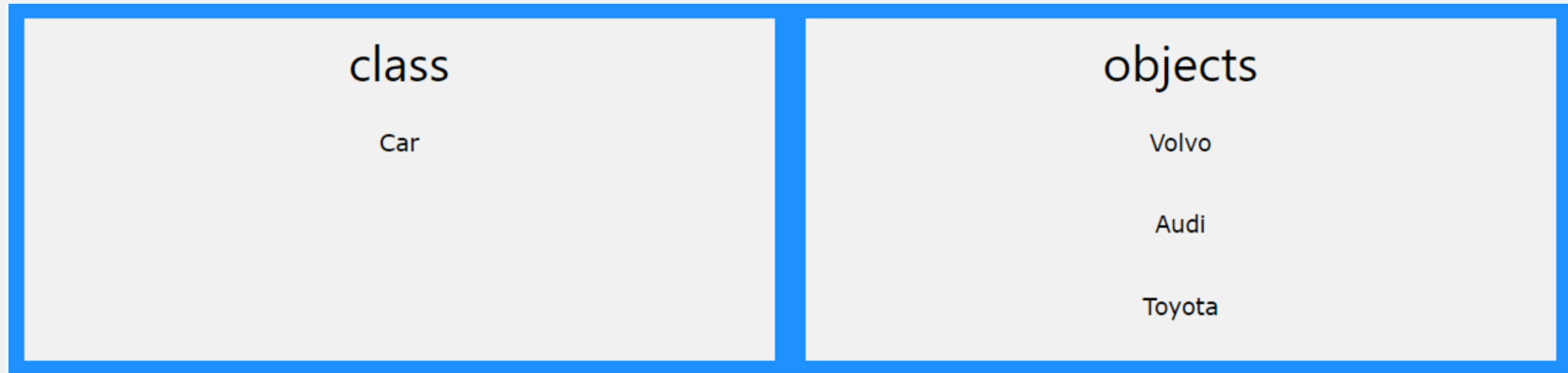
C# - What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:



Another Example



- So, a class is a template for objects, and an object is an instance of a class.
- When the individual objects are created, they inherit all the variables and methods from the class.

Abstraction

- The word *abstract* means a concept or an idea not associated with any specific instance.
- In C# programming, we apply the same meaning of abstraction by making classes not associated with any specific instance.
Abstraction is needed when we need to only inherit from a certain class, but do not need to instantiate objects of that class.
- In such a case the base class can be regarded as "Incomplete".
Such classes are known as an "Abstract Base Class".

Code example

(Write these codes and the check the output given on the next slide.)

Let's have a look at its code:

This is the Abstract Base Class, if I make both of its methods abstract then this class would become a pure Abstract Base Class.

Now, we derive a class of 'dog' from the class animal.

```
01. abstract class animal {  
02.     public abstract void eat();  
03.     public void sound() {  
04.         Console.WriteLine("dog can sound");  
05.     }  
06. }
```

```
01. abstract class animal {  
02.     public abstract void eat();  
03.     public void sound() {  
04.         Console.WriteLine("dog can sound");  
05.     }  
06. }  
07. class dog: animal {  
08.     public override void eat() {  
09.         Console.WriteLine("dog can eat");  
10.     }  
11. }
```

In the derived class, we have the same name method but this method has its body.

We are doing abstraction here so that we can access the method of the derived class without any trouble.

Let's have a look and try this code on computers:

```
01. class program
02. {
03.     abstract class animal
04.     {
05.         public abstract void eat();
06.         public void sound()
07.         {
08.             Console.WriteLine("dog can sound");
09.         }
10.     }
11.     class dog : animal
12.     {
13.         public override void eat()
14.         {
15.             Console.WriteLine("dog can eat");
16.         }
17.     }
18.     static void Main(string[] args)
19.     {
20.         dog mydog = new dog();
21.         animal thePet = mydog;
22.         thePet.eat();
23.         mydog.sound();
24.     }
25. }
```


Output

A screenshot of a macOS terminal window. The title bar at the top shows standard window controls (red, yellow, green buttons) and a home icon, followed by the text "praveen — demoproject.dll — bash -c clear; cd "/Applications/Visual Studio.app/...". The terminal content displays two lines of output: "dog can eat" and "dog can sound". Below these, the prompt "Press any key to continue..." is shown with a small black cursor block at the end. A vertical scrollbar is visible on the right side of the terminal window.

```
praveen — demoproject.dll — bash -c clear; cd "/Applications/Visual Studio.app/...  
dog can eat  
dog can sound  
Press any key to continue...
```

Practice Task: Abstraction:

Open this link on your computers, try writing the code individually on your computer and then check the output:

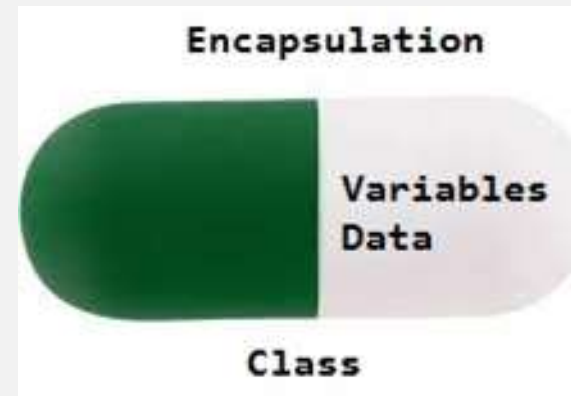
C# Abstraction

Encapsulation

- Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In a different way, encapsulation is a protective shield that prevents the data from being accessed by the code outside this shield.
- Technically in encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.
- Encapsulation can be achieved by: Declaring all the variables in the class as private and using C# Properties in the class to set and get the values of variables.

Encapsulation

- Abstraction and encapsulation are related features in object oriented programming. Abstraction allows making relevant information visible and encapsulation enables a programmer to implement the desired level of abstraction.
- Encapsulation is implemented by using access specifiers. An access specifier defines the scope and visibility of a class member. C# supports the following access specifiers –
 - Public
 - Private
 - Protected
 - Internal
 - Protected internal



Public Access Specifier

Public access specifier allows a class to expose its member variables and member functions to other functions and objects. Any public member can be accessed from outside the class.

The following example illustrates this –

Try writing the code for this output and then tally your code from the next slide:

```
Length: 4.5  
Width: 3.5  
Area: 15.75
```

View Live
Demo of the
code [here](#).

```
using System;

namespace RectangleApplication {
    class Rectangle {
        //member variables
        public double length;
        public double width;

        public double GetArea() {
            return length * width;
        }
        public void Display() {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }
    } //end class Rectangle

    class ExecuteRectangle {
        static void Main(string[] args) {
            Rectangle r = new Rectangle();
            r.length = 4.5;
            r.width = 3.5;
            r.Display();
            Console.ReadLine();
        }
    }
}
```

Private Access Specifier

Private access specifier allows a class to hide its member variables and member functions from other functions and objects. Only functions of the same class can access its private members. Even an instance of a class cannot access its private members.

The following example illustrates this –

Try writing the code for this output and then tally your code from the next slide –

```
Enter Length:
4.4
Enter Width:
3.3
Length: 4.4
Width: 3.3
Area: 14.52
```

View Live
Demo of the
code [here](#).

```
using System;

namespace RectangleApplication {
    class Rectangle {
        //member variables
        private double length;
        private double width;

        public void Acceptdetails() {
            Console.WriteLine("Enter Length: ");
            length = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("Enter Width: ");
            width = Convert.ToDouble(Console.ReadLine());
        }
        public double GetArea() {
            return length * width;
        }
        public void Display() {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }
    } //end class Rectangle

    class ExecuteRectangle {
        static void Main(string[] args) {
            Rectangle r = new Rectangle();
            r.Acceptdetails();
            r.Display();
            Console.ReadLine();
        }
    }
}
```


Protected Access Specifier

Protected access specifier allows a child class to access the member variables and member functions of its base class. This way it helps in implementing inheritance.

Internal Access Specifier

Internal access specifier allows a class to expose its member variables and member functions to other functions and objects in the current assembly. In other words, any member with internal access specifier can be accessed from any class or method defined within the application in which the member is defined.

The following example illustrates this –

Try writing the code for this output and then tally your code from the next slide:

```
Length: 4.5  
Width: 3.5  
Area: 15.75
```

View Live
Demo of the
code [here](#).

```
using System;

namespace RectangleApplication {
    class Rectangle {
        //member variables
        internal double length;
        internal double width;

        double GetArea() {
            return length * width;
        }
        public void Display() {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }
    }
}

class ExecuteRectangle {
    static void Main(string[] args) {
        Rectangle r = new Rectangle();
        r.length = 4.5;
        r.width = 3.5;
        r.Display();
        Console.ReadLine();
    }
}
}
```



C# Polymorphism and Overriding Methods

Internal Access Specifier

- Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.
- Like we specified in the previous chapter; Inheritance lets us inherit fields and methods from another class.
- Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

Internal Access Specifier

For example, think of a base class called **Animal** that has a method called **animalSound()**.

Derived classes of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

Example with Trainer's Coding Demo

While your trainer will demonstrate the code on their computer, observe it carefully to be able to write it independently. .

Check out the example with live demo [here.](#)

Practice Task:

Now practice writing a code for the following output:

```
The animal makes a sound  
The pig says: wee wee  
The dog says: bow wow
```

Tally your code from [this](#). You may practice with different animal names.



C# Interface

Interfaces:

Another way to achieve abstraction in C#, is with interfaces.

An interface is a completely "abstract class", which can only contain abstract methods and properties (with empty bodies):

Example

```
// interface
interface Animal
{
    void animalSound(); // interface method (does not have a body)
    void run(); // interface method (does not have a body)
}
```

Interfaces:

Another way to achieve abstraction in C#, is with inTo access the interface methods, the interface must be "implemented" by another class.

To implement an interface, use the : symbol (just like with inheritance). The body of the interface method is provided by the "implement" class.

Note that you do not have to use the override keyword when implementing an interface:terfaces.

An interface is a completely "abstract class", which can only contain abstract methods and properties (with empty bodies):

Interfaces:

The trainer will demonstrate writing this code on their computer.

Tally your code from [this](#). You may practice with different animal names.

```
// Interface
interface IAnimal
{
    void animalSound(); // interface method (does not have a body)
}

// Pig "implements" the IAnimal interface
class Pig : IAnimal
{
    public void animalSound()
    {
        // The body of animalSound() is provided here
        Console.WriteLine("The pig says: wee wee");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
    }
}
```

Task: Multiple Interfaces:

Now that you have observed coding of interfaces, try writing code for multiple interfaces. The output is given below.

Tip: To implement multiple interfaces, separate them with a comma.

```
Some text..  
Some other text...
```

Tally your code with the code given [here](#) once you have finished the task.



C# Classes & Structures

C# Class

- A class is like a blueprint of a specific object. In the real world, every object has some color, shape, and functionalities - for example, the luxury car Ferrari. Ferrari is an object of the luxury car type.
- The luxury car is a class that indicates some characteristics like speed, color, shape, interior, etc. So any company that makes a car that meets those requirements is an object of the luxury car type.
- For example, every single car of BMW, Lamborghini, Cadillac are an object of the class called 'Luxury Car'. Here, 'Luxury Car' is a class, and every single physical car is an object of the luxury car class.

C# Class

- Likewise, in object-oriented programming, a class defines some properties, fields, events, methods, etc. A class defines the kinds of data and the functionality their objects will have.
- A class enables you to create your custom types by grouping variables of other types, methods, and events.
- In C#, a class can be defined by using the class keyword.

Example: C# Class

```
public class MyClass
{
    public string myField = string.Empty;

    public MyClass()
    {
    }

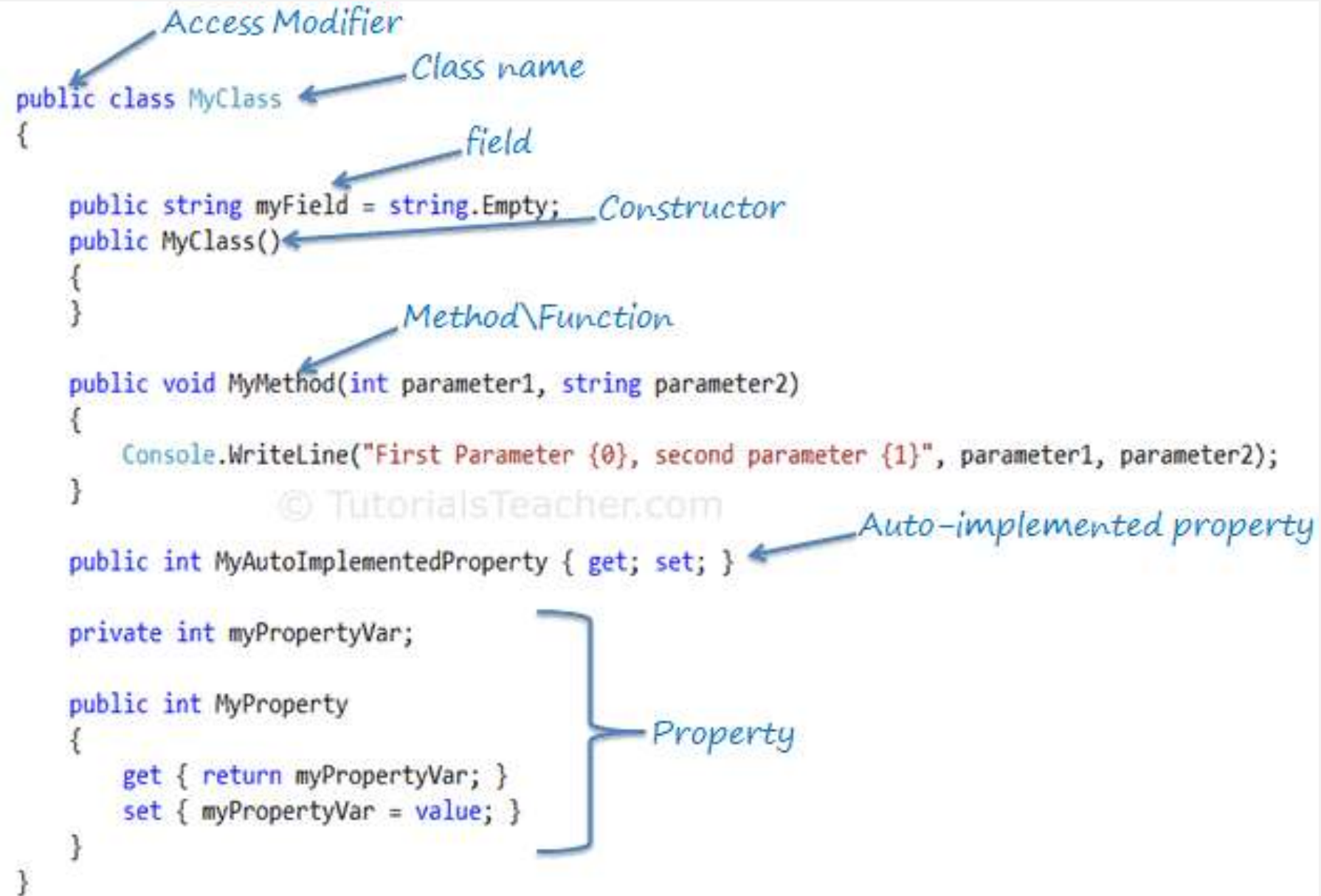
    public void MyMethod(int parameter1, string parameter2)
    {
        Console.WriteLine("First Parameter {0}, second parameter {1}",
                           parameter1, parameter2);
    }

    public int MyAutoImplementedProperty { get; set; }

    private int myPropertyVar;

    public int MyProperty
    {
        get { return myPropertyVar; }
        set { myPropertyVar = value; }
    }
}
```

The following image shows the important building blocks of C# class.



The image displays a C# class definition with handwritten annotations identifying its components. The code is as follows:

```
public class MyClass
{
    public string myField = string.Empty;
    public MyClass()
    {
    }
    public void MyMethod(int parameter1, string parameter2)
    {
        Console.WriteLine("First Parameter {0}, second parameter {1}", parameter1, parameter2);
    }
    public int MyAutoImplementedProperty { get; set; }
    private int myPropertyVar;
    public int MyProperty
    {
        get { return myPropertyVar; }
        set { myPropertyVar = value; }
    }
}
```

Annotations and their corresponding code elements:

- Access Modifier:** points to `public` in `public class MyClass`.
- Class name:** points to `MyClass` in `public class MyClass`.
- field:** points to `myField` in `public string myField = string.Empty;`.
- Constructor:** points to `MyClass()` in `public MyClass()`.
- Method\Function:** points to `MyMethod` in `public void MyMethod`.
- Auto-implemented property:** points to `MyAutoImplementedProperty` in `public int MyAutoImplementedProperty { get; set; }`.
- Property:** a bracket groups `MyAutoImplementedProperty` and `MyProperty` (which includes `myPropertyVar` and its `get/set` methods).

© TutorialsTeacher.com

Access Modifiers and Field

C# Access Modifiers

Access modifiers are applied to the declaration of the class, method, properties, fields, and other members. They define the accessibility of the class and its members. Public, private, protected, and internal are access modifiers in C#. We will learn about it in the keyword section.

C# Field

The field is a class-level variable that holds a value. Generally, field members should have a private access modifier and used with property.

C# Constructor

- A class can have parameterized or parameterless constructors. The constructor will be called when you create an instance of a class.
- Constructors can be defined by using an access modifier and class name: `<access modifiers> <class name>(){ }`

Example: Constructor in C#

```
class MyClass
{
    public MyClass()
    {
    }
}
```

Instantiating a Class

In the preceding program, the class ExecuteRectangle contains the Main() method and instantiates the Rectangle class.

Identifiers

An identifier is a name used to identify a class, variable, function, or any other user-defined item. The basic rules for naming classes in C# are as follows –

- A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore. The first character in an identifier cannot be a digit.
- It must not contain any embedded space or symbol such as ? - + ! @ # % ^ & * () [] { } . ; : " ' / and \. However, an underscore (_) can be used.
- It should not be a C# keyword.

C# Keywords

- Keywords are reserved words predefined to the C# compiler.
- These keywords cannot be used as identifiers. However, if you want to use these keywords as identifiers, you may prefix the keyword with the @ character.
- In C#, some identifiers have special meaning in context of code, such as get and set are called contextual keywords.
- The following table lists the reserved keywords and contextual keywords in C# –

Reserved Keywords

| | | | | | | |
|-----------|-----------|----------|------------|------------------------|-----------------------|---------|
| abstract | as | base | bool | break | byte | case |
| catch | char | checked | class | const | continue | decimal |
| default | delegate | do | double | else | enum | event |
| explicit | extern | false | finally | fixed | float | for |
| foreach | goto | if | implicit | in | in (generic modifier) | int |
| interface | internal | is | lock | long | namespace | new |
| null | object | operator | out | out (generic modifier) | override | params |
| private | protected | public | readonly | ref | return | sbyte |
| sealed | short | sizeof | stackalloc | static | string | struct |
| switch | this | throw | true | try | typeof | uint |
| ulong | unchecked | unsafe | ushort | using | virtual | void |
| volatile | while | | | | | |

Contextual Keywords

| | | | | | | |
|---------------------|--------|-----------|------------|---------|---------|----------------|
| add | alias | ascending | descending | dynamic | from | get |
| global | group | into | join | let | orderby | partial (type) |
| partial (method) | remove | select | set | | | |



C# - Struct

Structure

- In C#, struct is the value type data type that represents data structures. It can contain a parameterized constructor, static constructor, constants, fields, methods, properties, indexers, operators, events, and nested types.
- struct can be used to hold small data values that do not require inheritance, e.g. coordinate points, key-value pairs, and complex data structure.

Features of C# Structures

- Structures in C# are quite different from that in traditional C or C++. The C# structures have the following features –
- Structures can have methods, fields, indexers, properties, operator methods, and events.
- Structures can have defined constructors, but not destructors. However, you cannot define a default constructor for a structure. The default constructor is automatically defined and cannot be changed.
- Unlike classes, structures cannot inherit other structures or classes.

Features of C# Structures

- Structures cannot be used as a base for other structures or classes.
- A structure can implement one or more interfaces.
- Structure members cannot be specified as abstract, virtual, or protected.
- When you create a struct object using the New operator, it gets created and the appropriate constructor is called. Unlike classes, structs can be instantiated without using the New operator.
- If the New operator is not used, the fields remain unassigned and the object cannot be used until all the fields are initialized.

Structure Declaration

A structure is declared using struct keyword. The default modifier is internal for the struct and its members.

The following example declares a structure Coordinate for the graph.

Example: Structure

```
struct Coordinate
{
    public int x;
    public int y;
}
```

Example: Create Structure

The trainer will demonstrate writing this code.

Check the output [here](#):

```
struct Coordinate
{
    public int x;
    public int y;
}

Coordinate point = new Coordinate();
Console.WriteLine(point.x); //output: 0
Console.WriteLine(point.y); //output: 0
```

Task: Create Structure Without new Keyword

For the following output, individually, try writing a code to demonstrate a structure without new keyword:

Output:

1

Once all students have completed writing the code, this code will be displayed for the students to match theirs.

Code

Task:

Write a structure code for the following output:

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id :6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

[Click here](#) to match your code.

Difference Between Struct And Class In C#

| S.N | Struct | Classes |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Structs are value types, allocated either on the stack or inline in containing types. | Classes are reference types, allocated on the heap and garbage-collected. |
| 2 | Allocations and de-allocations of value types are in general cheaper than allocations and de-allocations of reference types. | Assignments of large reference types are cheaper than assignments of large value types. |
| 3 | In structs, each variable contains its own copy of the data (except in the case of the ref and out parameter variables), and an operation on one variable does not affect another variable. | In classes, two variables can contain the reference of the same object and any operation on one variable can affect another variable. |

When to use struct or classes?

Struct should be used only when you are sure that,

- It logically represents a single value, like primitive types (int, double, etc.).
- It is immutable.
- It should not be boxed and un-boxed frequently.
- In all other cases, you should define your types as classes.

Example:

The trainer will demonstrate writing this code on their computer.

You may also copy this along with the trainer.

[Check the output here.](#)

```
01. struct Location
02. {
03.     public int x, y;
04.     public Location(int x, int y)
05.     {
06.         this.x = x;
07.         this.y = y;
08.     }
09. }
10. Location a = new Location(20, 20);
11. Location b = a;
12. a.x = 100;
13. System.Console.WriteLine(b.x)
14. ;
```

Task:

Write a structure code for the following output:

```
Title : C Programming
Author : Nuha Ali
Subject : C Programming Tutorial
Book_id : 6495407
Title : Telecom Billing
Author : Zara Ali
Subject : Telecom Billing Tutorial
Book_id : 6495700
```

[Click here](#) to match your code.

Task:

Write a program in C# Sharp to declare a simple structure and use of static fields inside a structure.

Expected Output :

Structure with the use of static fields inside a structure :

The sum of x and y is 40



C# Enumerations

Enum

- An enumeration is a set of named integer constants. An enumerated type is declared using the enum keyword.
- C# enumerations are value data type. In other words, enumeration contains its own values and cannot inherit or cannot pass inheritance.

Declaring enum Variable

- The general syntax for declaring an enumeration is –

```
enum <enum_name> {  
    enumeration list  
};
```


Declaring enum Variable

- The general syntax for declaring an enumeration is –

```
enum <enum_name> {  
    enumeration list  
};
```

Declaring enum Variable

Where,

- The enum_name specifies the enumeration type name.
- The enumeration list is a comma-separated list of identifiers.

Each of the symbols in the enumeration list stands for an integer value, one greater than the symbol that precedes it. By default, the value of the first enumeration symbol is 0. For example –

```
enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
```

Example

The following example demonstrates use of enum variable –

Click [here](#) to match the output.

```
using System;

namespace EnumApplication {
    class EnumProgram {
        enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

        static void Main(string[] args) {
            int WeekdayStart = (int)Days.Mon;
            int WeekdayEnd = (int)Days.Fri;

            Console.WriteLine("Monday: {0}", WeekdayStart);
            Console.WriteLine("Friday: {0}", WeekdayEnd);
            Console.ReadKey();
        }
    }
}
```



C# LINQ

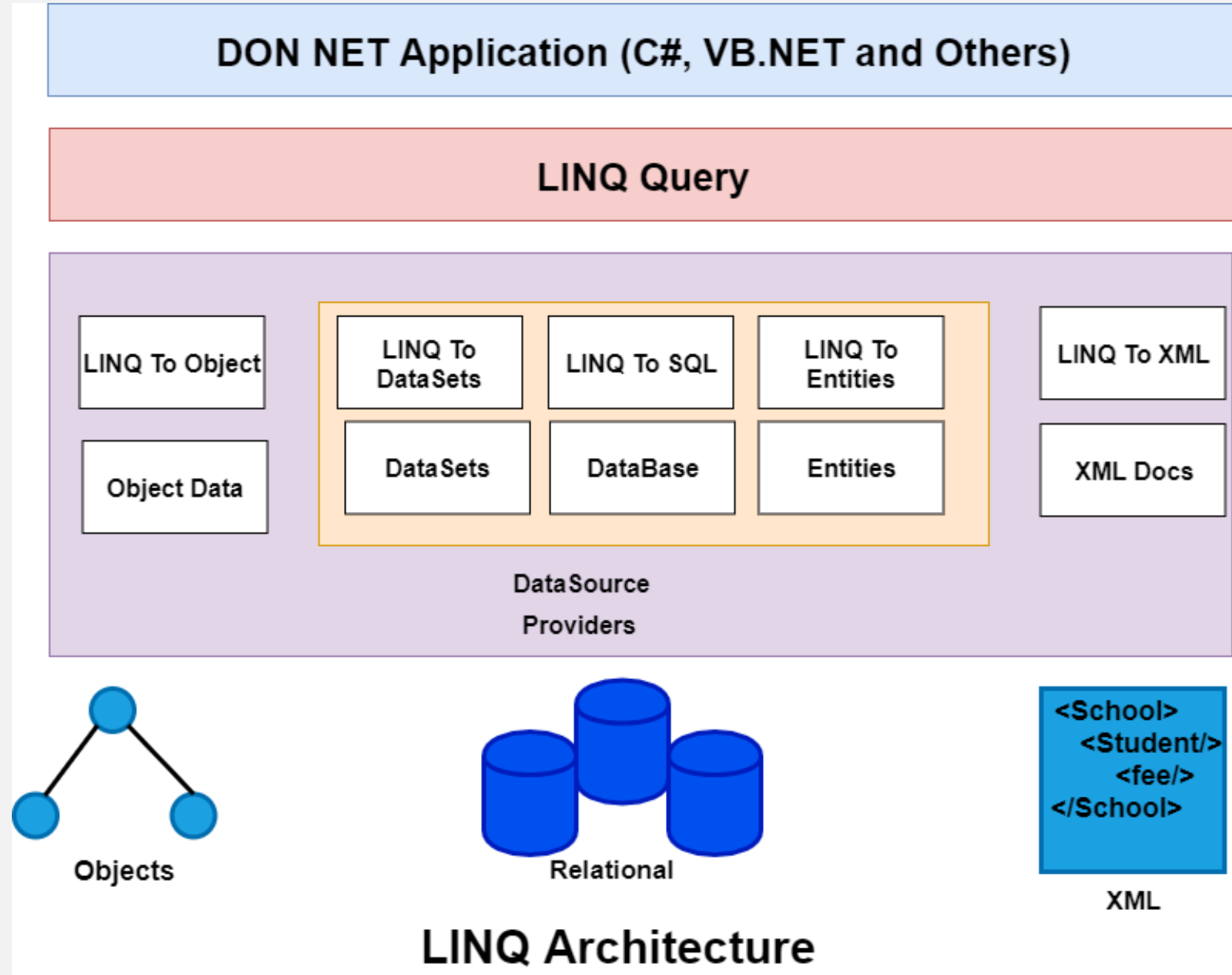
Language Integrated Query (LINQ)

- The full form of LINQ is 'Language Integrated Query,' and introduced in .NET Framework 3.5 to query the data from different sources of data such as collections, generics, XML documents, ADO.NET Datasets, SQL, Web Services, etc. in C# and VB.NET.
- LINQ provides the rich, standardized query syntax in a .NET programming language such as C# and VB.NET, which allows the developers to interact with any data sources.

Language Integrated Query (LINQ)

- In C# or VB.NET, LINQ functionality can be achieved by importing the System.Linq namespace in our application.
- Generally, the LINQ contains a set of extension methods which allows us to query the source of data object directly in our code based on the requirement.

LINQ Architecture



Advantages of LINQ

In our applications, the benefits of LINQ are:

- We do not need to learn new query language syntaxes for different sources of data because it provides the standard query syntax for the various data sources.
- In LINQ, we have to write the Less code in comparison to the traditional approach. With the use of LINQ, we can minimize the code.

Advantages of LINQ

- LINQ provides a lot of built-in methods that we can be used to perform the different operations such as filtering, ordering, grouping, etc. which makes our work easy.
- The query of LINQ can be reused.

Disadvantages of LINQ

- With the use of LINQ, it's very difficult to write a complex query like SQL.
- It was written in the code, and we cannot make use of the Cache Execution plan, which is the SQL feature as we do in the stored procedure.
- If the query is not written correctly, then the performance will be degraded.
- If we make some changes to our queries, then we need to recompile the application and need to redeploy the dll to the server.

C# LINQ query and method syntax

In LINQ, we can use either the query or the method syntax. A few methods, such as Append or Concat, do not have equivalents in the query syntax.

Example

Program.cs

```
var words = new string[] { "falcon", "eagle", "sky", "tree", "water" };

// Query syntax
var res = from word in words
          where word.Contains('a')
          select word;

foreach (var word in res)
{
    Console.WriteLine(word);
}

Console.WriteLine("-----");

// Method syntax
var res2 = words.Where(word => word.Contains('a'));

foreach (var word in res2)
{
    Console.WriteLine(word);
}
```

Tasks:

- 1- Write a program in C# to display the name of the days of a week in LINQ Query
- 2- Write a program in C# to create a list of numbers and display the numbers greater than 20 in LINQ Query

Build C# Game Programming

Develop

Develop a really simple game named 'Running T Rex'.

Learning Objectives

By the end of this session, the students have practised



What is C#



C# Syntax



Abstraction, Polymorphism, Encapsulation



Enumerations & Overriding



Interfaces, Classes & Structures



LINQ



Conclusion & Q/A

See you tomorrow!