
C# Tutorial for Beginners

By : Benjamin Walker

🕒 September 3, 2025

C# Tutorial Summary

C# (C Sharp) is one of the most popular programming languages which is widely used for building Windows applications, mobile applications, and games. This course is taught in a practical GOAL oriented way to learn C# programming. It is recommended for you to practice the code assignments given after each C Sharp tutorial to learn C Sharp fast and easily.

What is C#?

C# (C Sharp) is a general purpose, multi-paradigm programming language developed by Microsoft that runs on the .NET Framework. C# is widely used for building mobile applications, games and windows applications.

What should I know?

Basic knowledge of C programming and OOP concept will be an additional help.

C# Syllabus

Introduction

<input type="checkbox"/> Lesson 1	What is .NET Framework? — Explain Architecture & Components
<input type="checkbox"/> Lesson 2	C# and .Net Version History — A Brief Version History of C# and .Net
<input type="checkbox"/> Lesson 3	Install Visual Studio for C# — How to Download and Install Visual Studio for C#
<input type="checkbox"/> Lesson 4	C# Hello World Program — C# Hello World! First Console Application Program

Advanced Stuff

<input type="checkbox"/> Lesson 1	C# Data Types — Learn With Example
<input type="checkbox"/> Lesson 2	C# Enumeration — Learn C# Enum With Example
<input type="checkbox"/> Lesson 3	C# Variables & Operators — Learn C# Variables & Operators with Example
<input type="checkbox"/> Lesson 4	C# IF, Switch, For, While Loop Statements — Learn With Example
<input type="checkbox"/> Lesson 5	C# Array Tutorial — Create, Declare, Initialize
<input type="checkbox"/> Lesson 6	C# Class & Object Tutorial — Learn With Example
<input type="checkbox"/> Lesson 7	C# Access Modifiers — Learn With Program Example
<input type="checkbox"/> Lesson 8	C# Inheritance & Polymorphism — Learn With Example
	C# Abstract Class Tutorial — What is

<input type="checkbox"/> Lesson 9	C# Abstract Class Tutorial — What is Abstraction with Example
<input type="checkbox"/> Lesson 10	C# Interface Tutorial — What is an Interface Class?
<input type="checkbox"/> Lesson 11	C# Collections Tutorial — What is Collections in C#?
<input type="checkbox"/> Lesson 12	C# ArrayList Tutorial — What is ArrayList in C#? & Examples
<input type="checkbox"/> Lesson 13	C# Stack — Learn With Example
<input type="checkbox"/> Lesson 14	C# Queue — What is Queue in C#? with Examples
<input type="checkbox"/> Lesson 15	C# Hashtable — Learn C# Hashtable with Examples
<input type="checkbox"/> Lesson 16	C# Windows Forms Application Tutorial — Create a Windows Forms app with C#
<input type="checkbox"/> Lesson 17	C# Database Connection — How to connect SQL Server (Example)
<input type="checkbox"/> Lesson 18	C# File I/O Handling Operations — Learn With Example
<input type="checkbox"/> Lesson 19	C# Stream Tutorial — StreamReader, StreamWriter with Example
<input type="checkbox"/> Lesson 20	C# Serialization & Deserialization — Learn With Example
<input type="checkbox"/> Lesson 21	Coded UI Test Automation Framework Tutorial — Introduction to Coded UI

RELATED ARTICLES

- [C# Database Connection: How to connect SQL Server \(Example\)](#)
- [C# Variables & Operators with Example](#)
- [C# IF, Switch, For, While Loop Statements Tutorial \[Examples\]](#)
- [Stream in C# Tutorial: StreamReader & StreamWriter \[Example\]](#)

Must Know!

<input type="checkbox"/> Lesson 1	Best C# IDE — 10 Best C# IDE for Windows, Linux, Mac
<input type="checkbox"/> Lesson 2	C# Interview Questions — Top 50 C# Interview Questions and Answers
<input type="checkbox"/> Lesson 3	C# Books — 14 BEST C# Books
<input type="checkbox"/> Lesson 4	C# Tutorial PDF — Download C# Tutorial PDF for Beginners

Why to Learn C#?

Learning C# helps to understand the basic Computer theories easily, and it is a middle-level programming language; therefore, it is easy to understand and user-friendly. Moreover, C# contains fewer libraries, and it has a faster execution time; therefore, it is widely used in embedded programming.

Applications of C Sharp Programming

C Sharp programming is widely used for developing Desktop applications, Web applications, Web services, large scale Windows applications as well as developing Games.

Prerequisites for learning C#

For learning C# for beginners, it is required to have a basic computer knowledge, basic knowledge of Object-Oriented programming approach, and basic knowledge of C programming language concepts to understand C# basics easily.

What will you learn in this C# Tutorial?

First in this C Sharp tutorial, you will learn the C# basics like introduction, history of C# and

architecture. Then, you will learn the advanced stuff for C# programming like C# data types, variables, classes & objects, interface, collections, file operations, etc.



[Report a Bug](#)

[Next →](#)

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

What is .NET Framework? Explain Architecture & Components

By : Benjamin Walker

🕒 August 10, 2024

What is .Net Framework?

.Net Framework is a software development platform developed by Microsoft for building and running Windows applications. The .Net framework consists of developer tools, programming languages, and libraries to build desktop and web applications. It is also used to build websites, web services, and games.

The .Net framework was meant to create applications, which would run on the Windows Platform. The first version of the .Net framework was released in the year 2002. The version was called .Net framework 1.0. The Microsoft .Net framework has come a long way since then, and the current version is .Net Framework 4.7.2.

The Microsoft .Net framework can be used to create both – **Form-based** and **Web-based** applications. [Web services](#) can also be developed using the .Net framework.

The framework also supports various programming languages such as Visual Basic and C#. So developers can choose and select the language to develop the required application. In this chapter, you will learn some basics of the .Net framework.

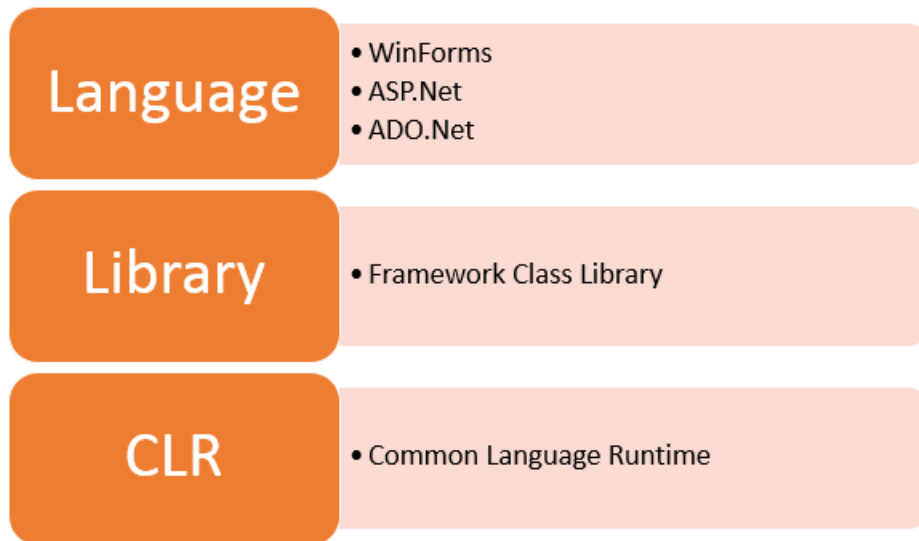
Table of Contents:



.Net Framework Architecture

.Net Framework Architecture is a programming model for the .Net platform that provides an execution environment and integration with various programming languages for simple development and deployment of various Windows and desktop applications. It consists of class libraries and reusable components.

The basic architecture of the .Net framework is as shown below.



.Net Framework Architecture Diagram

.NET Components

The architecture of .Net framework is based on the following key components;

1. Common Language Runtime

The “Common Language Infrastructure” or CLI is a platform in .Net architecture on which the .Net programs are executed.

The CLI has the following key features:

Exception Handling – Exceptions are errors which occur when the application is executed.

Examples of exceptions are:

- If an application tries to open a file on the local machine, but the file is not present.
- If the application tries to fetch some records from a [database](#), but the connection to the database is not valid.

Garbage Collection – Garbage collection is the process of removing unwanted resources when they are no longer required.

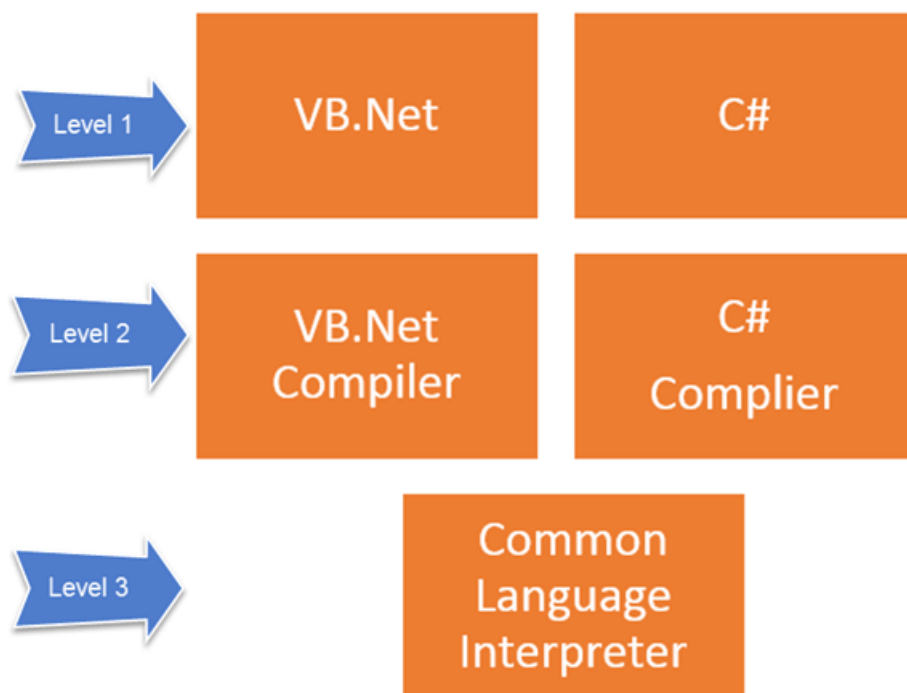
Examples of garbage collection are

- A File handle which is no longer required. If the application has finished all operations on a file, then the file handle may no longer be required.
- The database connection is no longer required. If the application has finished all operations on a database, then the database connection may no longer be required.

Working with Various programming languages –

As noted in an earlier section, a developer can develop an application in a variety of .Net programming languages.

1. **Language** – The first level is the programming language itself, the most common ones are VB.Net and C#.
2. **Compiler** – There is a compiler which will be separate for each programming language. So underlying the VB.Net language, there will be a separate VB.Net compiler. Similarly, for C#, you will have another compiler.
3. **Common Language Interpreter** – This is the final layer in .Net which would be used to run a .net program developed in any [programming language](#). So the subsequent compiler will send the program to the CLI layer to run the .Net application.



2. Class Library

The .NET Framework includes a set of standard class libraries. A class library is a collection of methods and functions that can be used for the core purpose.

For example, there is a class library with methods to handle all file-level operations. So there is a method which can be used to read the text from a file. Similarly, there is a method to write text to a file.

Most of the methods are split into either the System.* or Microsoft.* namespaces. (The asterisk * just means a reference to all of the methods that fall under the System or Microsoft namespace)

A namespace is a logical separation of methods. We will learn these namespaces more in detail in the subsequent chapters.

3. Languages

The types of applications that can be built in the .Net framework is classified broadly into the following categories.

WinForms – This is used for developing Forms-based applications, which would run on an end user machine. Notepad is an example of a client-based application.

ASP.Net – This is used for developing web-based applications, which are made to run on any browser such as Internet Explorer, Chrome or Firefox.

- The Web application would be processed on a server, which would have Internet Information Services Installed.
- Internet Information Services or IIS is a Microsoft component which is used to execute an [Asp.Net](#) application.
- The result of the execution is then sent to the client machines, and the output is shown in the browser.

ADO.Net – This technology is used to develop applications to interact with Databases such as Oracle or Microsoft [SQL](#) Server.

Microsoft always ensures that .Net frameworks are in compliance with all the supported Windows operating systems.

RELATED ARTICLES

→ [How to Download and Install Visual Studio for C# in Windows](#)

- [C# Inheritance and Polymorphism with Program Examples](#)
- [C# Abstract Class Tutorial with Example: What is Abstraction?](#)
- [C# Tutorial PDF \(Download Now\)](#)

.Net Framework Design Principle

Now in this .Net Architecture tutorial, we will learn the design principles of .Net framework. The following design principles of the .Net framework is what makes it very relevant to create .Net based applications.

- 1) Interoperability** – The .Net framework provides a lot of backward support. Suppose if you had an application built on an [older version of the .Net framework](#), say 2.0. And if you tried to run the same application on a machine which had the higher version of the .Net framework, say 3.5. The application would still work. This is because with every release, Microsoft ensures that older framework versions gel well with the latest version.
- 2) Portability** – Applications built on the .Net framework can be made to work on any Windows platform. And now in recent times, Microsoft is also envisioning to make Microsoft products work on other platforms, such as iOS and [Linux](#).
- 3) Security** – The .NET Framework has a good security mechanism. The inbuilt security mechanism helps in both validation and verification of applications. Every application can explicitly define their security mechanism. Each security mechanism is used to grant the user access to the code or to the running program.
- 4) Memory management** – The Common Language runtime does all the work or [memory management](#). The .Net framework has all the capability to see those resources, which are not used by a running program. It would then release those resources accordingly. This is done via a program called the “Garbage Collector” which runs as part of the .Net framework. The garbage collector runs at regular intervals and keeps on checking which system resources are not utilized, and frees them accordingly.
- 5) Simplified deployment** – The .Net framework also have tools, which can be used to package applications built on the .Net framework. These packages can then be distributed to client machines. The packages would then automatically install the application.

Summary

- .Net is a programming language developed by Microsoft. It was designed to build applications which could run on the Windows platform.
- The .Net programming language can be used to develop Forms based applications, Web based applications, and Web services.

- Developers can choose from a variety of programming languages available on the Microsoft .Net framework platform. The most common ones are VB.Net and C#.



[Report a Bug](#)

[← Prev](#)

[Next →](#)

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

© Copyright - Guru99 2025 Privacy Policy | Affiliate Disclaimer | ToS | Editorial Policy

C# and .Net Version History

By : Benjamin Walker

🕒 August 10, 2024

.Net Framework Version History

The first version of the .Net framework was released in the year 2002. The version was called .Net framework 1.0. The .Net framework has come a long way since then, and the current version is 4.7.1.

Below is the table of .Net framework versions, which have been released with their release dates. Every version has relevant changes to the framework.

For example, in framework 3.5 and onwards a key framework called the **Entity framework** was released. This framework is used to change the approach in which the applications are developed while working with [databases](#).

In this tutorial, you will learn-

- [.Net Framework Version History](#)
- [C# Version History](#)

Version number	CLR version	Release date
1.0	1.0	2002-02-13
1.1	1.1	2003-04-24
2.0	2.0	2005-11-07
3.0	2.0	2006-11-06
3.5	2.0	2007-11-19
4.0	4	2010-04-12
4.5	4	2012-08-15
4.5.1	4	2013-10-17
4.5.2	4	2014-05-05
4.6	4	2015-07-20
4.6.1	4	2015-11-17
4.6.2	4	2016-08-02
4.7	4	2017-04-05
4.7.1	4	2017-10-17

The biggest advantage of the [.Net framework](#) is that it supports Windows platform. Almost everyone works with Windows machines.

Microsoft always ensures that .Net frameworks are in compliance with all the supported Windows operating systems.

C# Version History

Version	.NET Framework	Visual Studio	Important Features
---------	----------------	---------------	--------------------

Version	.NET Framework	Visual Studio	Important Features
C# 1.0	.NET Framework 1.0/1.1	Visual Studio .NET 2002	First release of C#
C# 2.0	.NET Framework 2.0	Visual Studio 2005	<ul style="list-style-type: none"> • Generics • Partial types • Anonymous methods • Nullable types • Iterators • Covariance and contravariance
C# 3.0	.NET Framework 3.0/3.5	Visual Studio 2008	<ul style="list-style-type: none"> • Auto-implemented properties • Anonymous types • Query expressions • Lambda expression • Expression trees • Extension methods
C# 4.0	.NET Framework 4.0	Visual Studio 2010	<ul style="list-style-type: none"> • Dynamic binding • Named/optional arguments • Generic covariant and contravariant • Embedded interop types
C# 5.0	.NET Framework 4.5	Visual Studio 2012/2013	<ul style="list-style-type: none"> • Asynchronous members • Caller info attributes
C# 6.0	.NET Framework 4.6	Visual Studio 2013/2015	<ul style="list-style-type: none"> • Static imports • Exception filters • Property initializers • Expression bodied members • Null propagator • String interpolation • nameof operator • Dictionary initializer

Version	NET Framework	Visual Studio	Important Features
C# 7.0	:NET Core	Visual Studio 2017	<ul style="list-style-type: none"> • Improved performance and productivity • Azure Support • AI Support • Game development • Cross platform • Mobile App Development • Window App Development



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Your Email Address

Subscribe for Free



Chosen by over **350,000+** professionals





Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

Data Types in C#: Double, Integer, Float, Char

By : Benjamin Walker

🕒 August 10, 2024

What are Data Types in C#?

The C# language comes with a set of Basic data types. These data types are used to build values which are used within an application. Let's explore the basic data types available in C#. For each example, we will modify just the main function in our Program.cs file.

Table of Content:



1) Integer

An Integer data types are used to work with numbers. In this case, the numbers are whole numbers like 10, 20 or 30. In C#, the datatype is denoted by the **Int32 keyword**. Below is an example of how this datatype can be used. In our example, we will define an Int32 variable called num. We will then assign an Integer value to the variable and then display it accordingly.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Int32 num=30;
            Console.Write(num);

            Console.ReadKey();
        }
    }
}
```

Code Explanation:-

1. The Int32 data type is specified to declare an Integer variable called num. The variable is then assigned a value of 30.
2. Finally the console.write function is used to display the number to the console.

If the above code is entered properly and the program is executed successfully, following output will be displayed.

Output:

From the output, you can clearly see that the Integer variable called num was displayed in the console

2) Double

A double data type is used to work with decimals. In this case, the numbers are whole numbers like 10.11, 20.22 or 30.33. In C#, the datatype is denoted by the keyword **“Double”**. Below is an example of this datatype.

In our example, we will define a double variable called num. We will then assign a Double value to the variable and then display it accordingly.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            double num=30.33;
            Console.Write(num);

            Console.ReadKey();
        }
    }
}
```

Code Explanation:-

1. The double data type is specified to declare a double type **variable** called num. The variable is then assigned a value of 30.33.
2. Finally the console.write function is used to display the number to the console.

If the above code is entered properly and the program is executed successfully, following output will be displayed.

Output:

From the output, you can clearly see that the double variable called num was displayed in the console

3) Boolean

A boolean data type is used to work with Boolean values of **true and false**. In C#, the datatype is denoted by the Boolean keyword. Below is an example of this datatype can be

used.

In our example, we will define a Boolean variable called 'status.' We will then assign a boolean value to the variable and then display it accordingly.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Boolean status=true;
            Console.Write(status);

            Console.ReadKey();
        }
    }
}
```

Code Explanation:-

1. The boolean data type is specified to declare a Boolean variable called 'status.' The variable is then assigned a value of true/false.
2. Finally the console.write function is used to display the Boolean value to the console.

If the above code is entered properly and the program is executed successfully, the output will be displayed.

Output:

From the output, you can clearly see that the Boolean variable which equals true was displayed in the console

4) String

A String data type is used to work with String values. In C#, the datatype is denoted by the keyword 'String'. Below is an example of this datatype.

In our example, we will define a String variable called 'message.' We will then assign a

String value to the variable and then display it accordingly.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class program
    {
        static void Main(string[] args)
        {
            String message="Hello";
            Console.Write(message);

            Console.ReadKey();
        }
    }
}
```

Code Explanation:-

1. The String data type is specified to declare a string variable called message. The variable is then assigned a value of "Hello".
2. Finally, the console.write function is used to display the string value to the console.

If the above code is entered properly and the program is executed successfully, the output will be displayed.

Output:

From the output, you can clearly see that the String variable called message was displayed in the console



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Your Email Address

Subscribe for Free

C# Enum(Enumeration) with Example

By : Benjamin Walker

🕒 August 10, 2024

C# Enumeration

An enumeration is used in any programming language to define a constant set of values. For example, the days of the week can be defined as an enumeration and used anywhere in the program. In C#, the enumeration is defined with the help of the keyword 'enum'.

Let's see an example of how we can use the 'enum' keyword.

In our example, we will define an enumeration called days, which will be used to store the days of the week. For each example, we will modify just the main function in our Program.cs file.

```
namespace DemoApplication {  
    class Program  
    {  
        enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };  
        static void Main(string[] args)  
        {  
            Console.Write(Days.Sun);  
            Console.ReadKey();  
        }  
    }  
}
```

enum data type declaration

1

2

Displaying a value of the enum data type

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Program
    {
        enum Days{Sun,Mon,tue,Wed,thu,Fri,Sat};

        static void Main(string[] args)
        {
            Console.Write(Days.Sun);

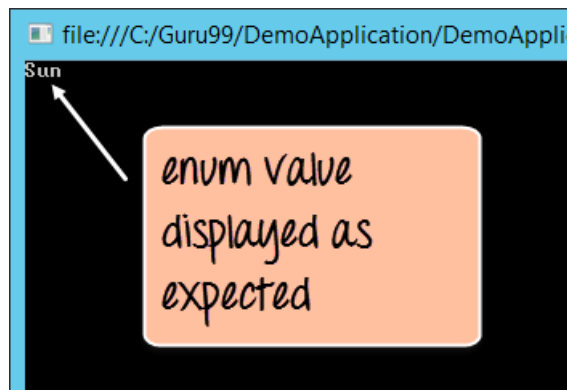
            Console.ReadKey();
        }
    }
}
```

Code Explanation:-

1. The 'enum' **data type** is specified to declare an enumeration. The name of the enumeration is Days. All the days of the week are specified as values of the enumeration.
2. Finally the console.write function is used to display one of the values of the enumeration.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



From the output, you can see that the 'Sun' value of the enumeration is displayed in the console.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Subscribe for Free



Chosen by over **350,000+** professionals





Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

C# IF, Switch, For, While Loop Statements Tutorial [Examples]

By : Benjamin Walker

🕒 August 10, 2024

Flow Control and conditional statements

Flow control and conditional statements are available in any programming language to alter the flow of a program.

For example, if someone wants to execute only a particular set of statements based on some certain logic, then Flow control, and conditional statements will be useful.

You will get a better understanding as we go through the various statements which are available in C#.

Please note that all the code below is made to the Program.cs file.

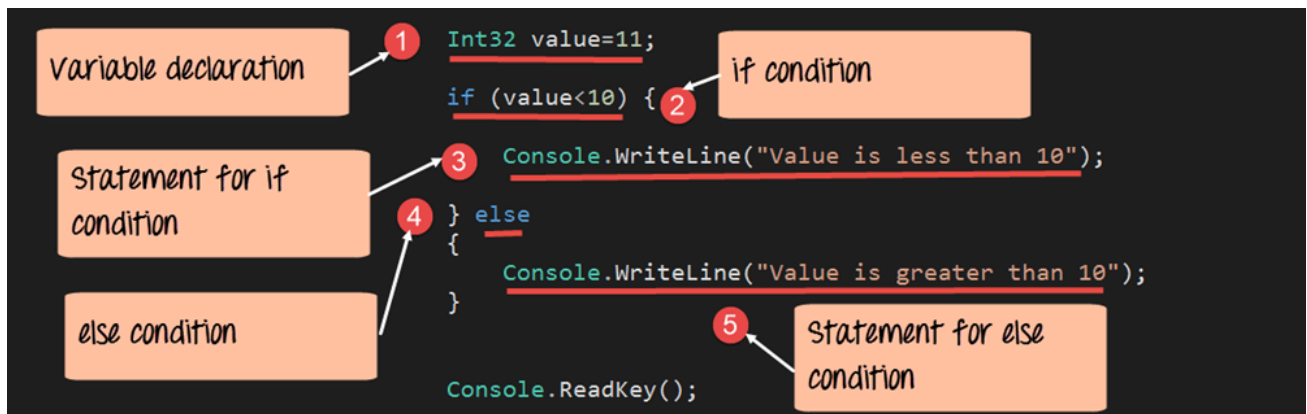
Table of Content:



1) If statement

The if statement is used to evaluate a boolean expression before executing a set of statements. If an expression evaluates to true, then it will run one set of statements else it will run another set of statements.

In our example below, a comparison is made for a variable called value. If the value of the variable is less than 10, then it will run one statement, or else it will run on another statement.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Int32 value = 11;

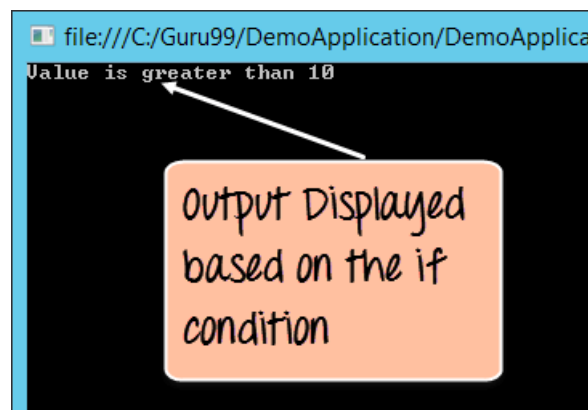
            if(value<10)
            {
                Console.WriteLine("Value is less than 10");
            }
            else
            {
                Console.WriteLine("Value is greater than 10");
            }
            Console.ReadKey();
        }
    }
}
```

Code Explanation

1. We first define a variable called value and set it to the value of 11.
2. We then use the 'if' statement to check if the value is less than 10 of the variable. The result will either be true or false.
3. If the if condition evaluates to true, we then send the message "Value is less than 10" to the console.
4. If the if condition evaluates to false, we then send the message "Value is greater than 10" to the console.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



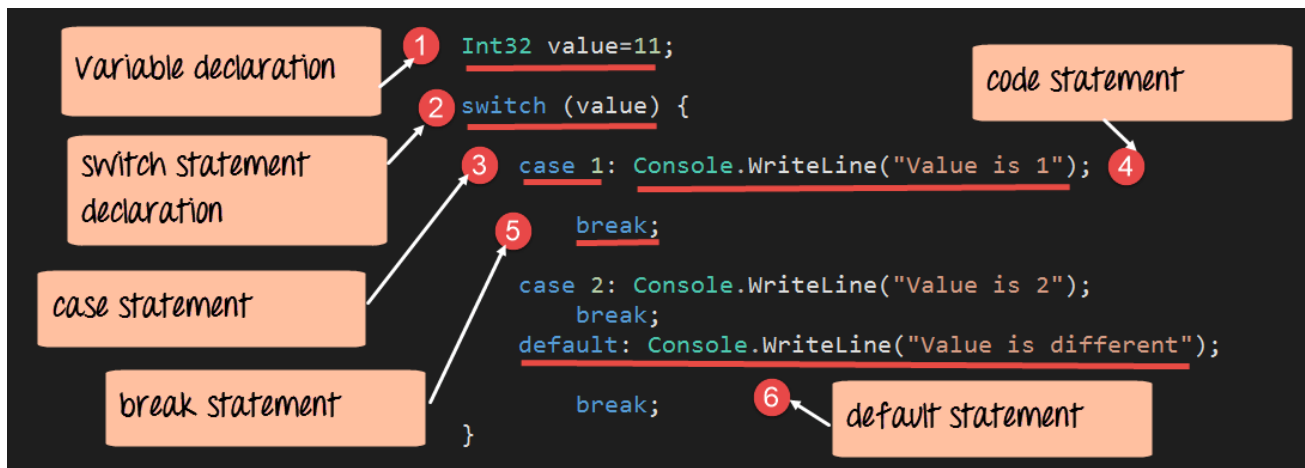
We can clearly see that the 'if' statement was evaluated to false. Hence the message "Value is greater than 10" was sent to the console.

2) Switch statement

The switch statement is an enhancement to the 'if' statement. If you have multiple expressions that need to be evaluated in one shot, then writing multiple 'if' statements becomes an issue.

The switch statement is used to evaluate an expression and run different statements based on the result of the expression. If one condition does not evaluate to true, the switch statement will then move to the next condition and so forth.

Let's see, how this works with the below example. Here, we are again comparing the value of a variable called 'value.' We then check if the value is equal to 1, or 2, or something totally different.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Int32 value=11;
            switch(value)
            {
                case 1: Console.WriteLine("Value is 1");
                        break;
                case 2: Console.WriteLine("Value is 2");
                        break;
                default: Console.WriteLine("value is different");
                        break;
            }
        }
    }
}
```

Code Explanation:-

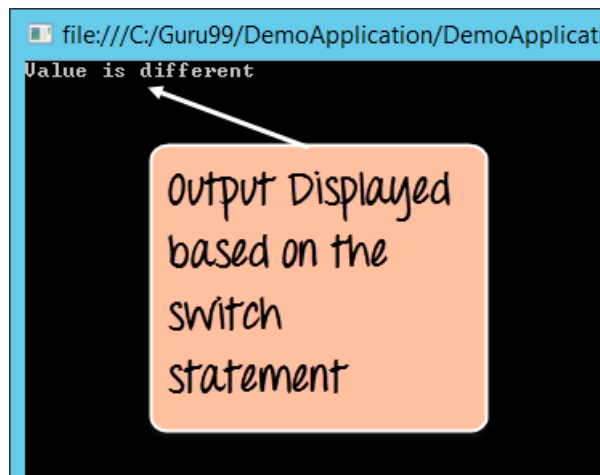
1. We first define a variable called 'value' and set it to the value of 11.
2. We then use the 'switch' statement to check the value of the variable 'value.'
3. Case statements are used to set different conditions. Based on the conditions, a set of statements can be executed. A switch statement can have multiple case conditions. The first case statement checks to see if the value of the variable is equal to 1.
4. If the first case statement is true, then the message "Value is 1" is written to the

console.

5. The break statement is used to break from the entire switch statement, once a condition is true.
6. The default condition is a special condition. This just means that if no case expression evaluates to true, then run the set of statements for the default condition.

If the above code is entered properly and the program is executed successfully, the following output will be displayed. The output prints the default value "Value is different", since no condition is satisfied.

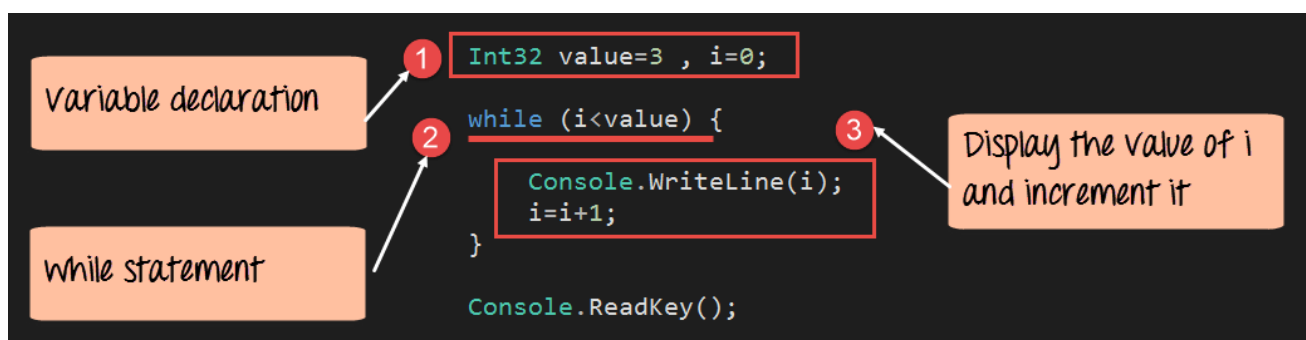
Output:



3) While loop

The while loop is used for iterative purposes. Suppose if you want to repeat a certain set of statements for a particular number of times, then while loop is used.

In our example below, we use the while statement to display the value of a variable 'i'. The while statement is used to display the value 3 times.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Int32 value=3,i=0;

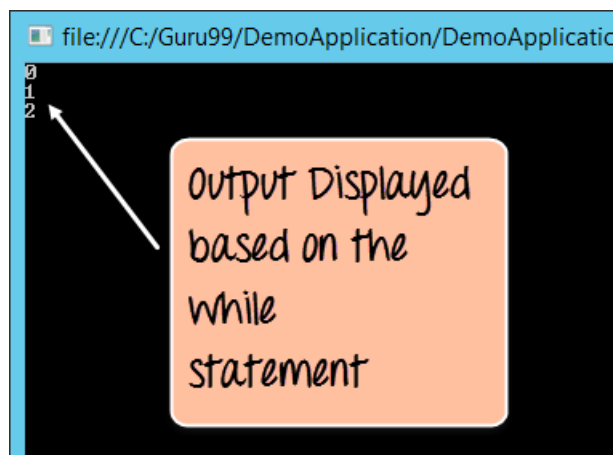
            while(i<value)
            {
                Console.WriteLine(i);
                i=i+1;
            }
            Console.ReadKey();
        }
    }
}
```

Code Explanation:-

1. Two Integer **variables** are defined, one being value and the other being 'i'. The value variable is used as the upper limit to which we should iterate our while statement. And 'i' is the variable which will be processed during the iteration.
2. In the while statement, the value of 'i' is constantly checked against the upper limit.
3. Here we display the value of 'i' to the console. We also increment the value of 'i'.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:

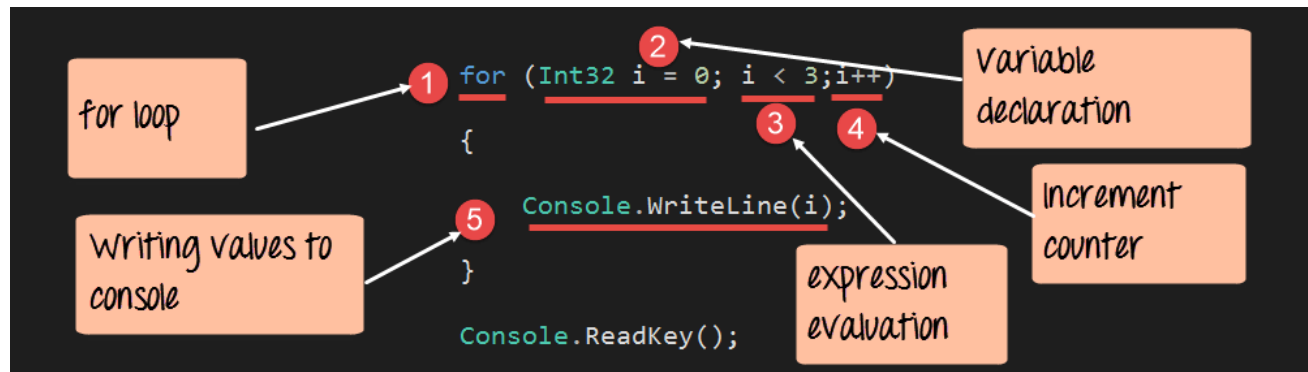


Here you can see that the while statement is executed 3 times and incremented at the same time. And each time, it displayed the current value of the variable 'i'.

4) For loop

The 'for' loop is also used for iterative purposes. Suppose if you want to repeat a certain set of statements for a particular number of times, then forloop is used.

In our example below, we use the 'for' statement to display the value of a variable 'i'. The 'for' statement is used to display the value 3 times.



```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
namespace DemoApplication  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            for(Int32 i=0;i<3;i++)  
            {  
                Console.WriteLine(i);  
            }  
            Console.ReadKey();  
        }  
    }  
}
```

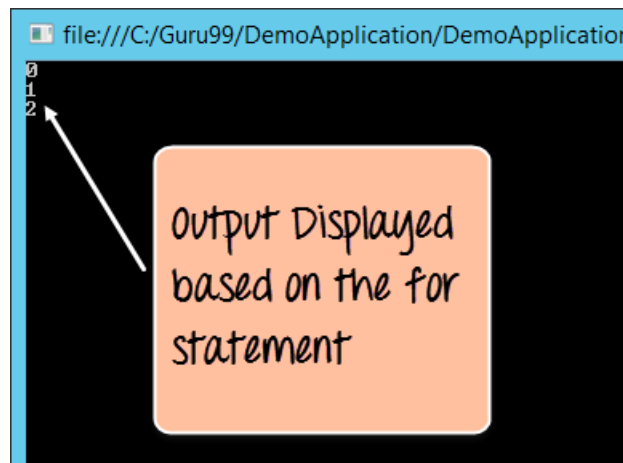
Code Explanation:-

1. The 'for' keyword is used to start off the 'for loop' statement.
2. In the 'for loop', we define 3 things. The first is to initialize the value of a variable, which will be used in the 'for loop'.

3. The second is to compare the value of the 'i' against an upper limit. In our case, the upper limit is the value of 3 ($i < 3$).
4. Finally, we increment the value of 'i' accordingly.
5. Here we display the value of 'i' to the console.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



Here you can see that the 'for' statement is executed 3 times. And each time, it displayed the current value of the variable 'i'.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

C# Array Tutorial: Create, Declare, Initialize

By : Benjamin Walker

🕒 August 10, 2024

What is an Arrays in C#?

An array is used to store a collection or series of elements. These elements will be of the same type.

So for example, if you had an array of Integer values, the array could be a collection of values such as [1, 2, 3, 4]. Here the number of elements in the array is 4.

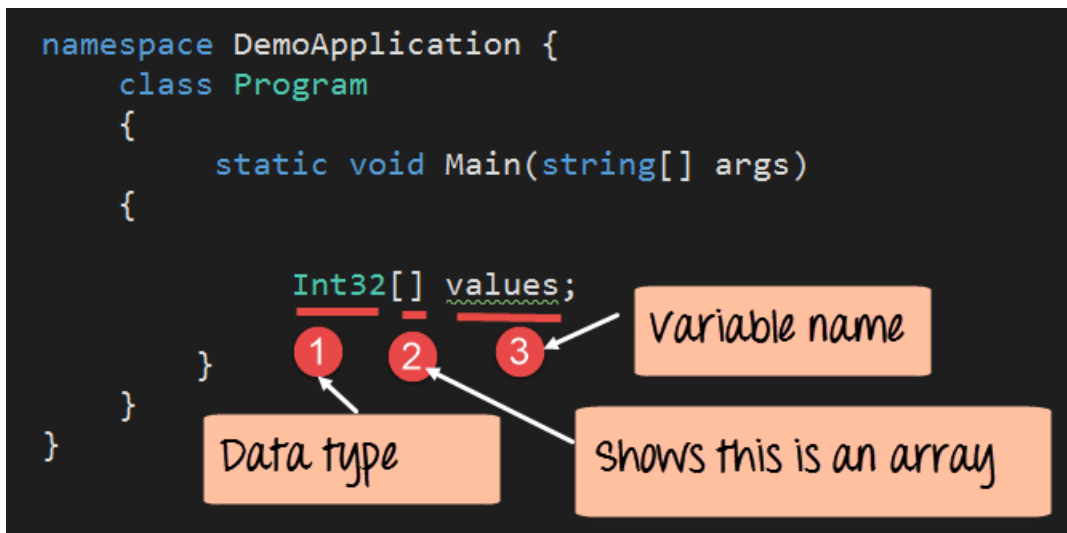
Arrays are useful when you want to store a collection of values of the same type. So instead of declaring a **variable** for every element, you can just declare one variable.

This variable will point to an array or list of elements, which will be responsible for storing the elements of the array.

Let's look at how we can work with arrays in C#. In our example, we will declare an array of Integers and work with them accordingly.

Note that all of the below code is being made to the Program.cs file.

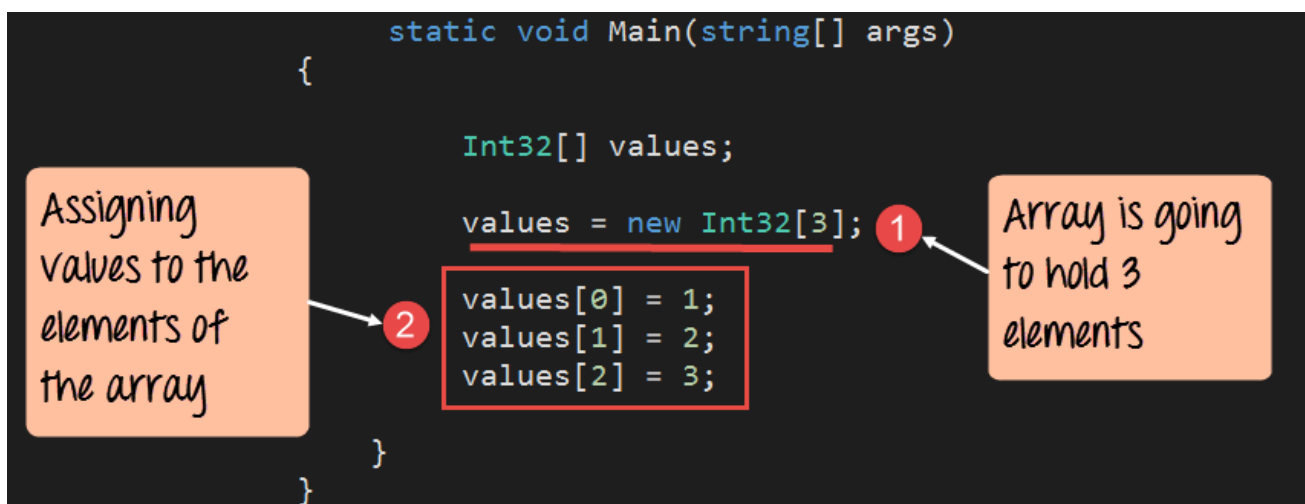
Step 1) Declaring an array – The first step is to declare an array. Let's see how we can achieve this by the below code example.



Code Explanation:-

1. The first part is the **datatype**. It specifies the type of elements used in the array. So in our case, we are creating an array of Integers.
2. The second part `[]`, which specifies the rank of the array. (The rank is a placeholder which specifies the number of elements the array will contain)
3. Next is the Name of the array which in our case is 'values'. Note you see a green squiggly underline, don't worry about that. That is just .Net saying that you have declared an array, but not using it anywhere.

Step 2) The next step is to initialize the array. Here we are going to specify the number of values the array will hold. We are also going to assign values to each element of the array.



Code Explanation:-

1. First, we are setting the number of elements the array will hold to 3. So in the square brackets, we are saying that the array will hold 3 elements.
2. Then we are assigning values to each element of the array. We can do this by specifying the variable name + the index position in the array. So `values[0]` means

that we are storing a value in the first position of the array. Similarly to access the second position, we use the notation of `values[1]` and so on and so forth.

Note: – In Arrays, the index position starts from 0.

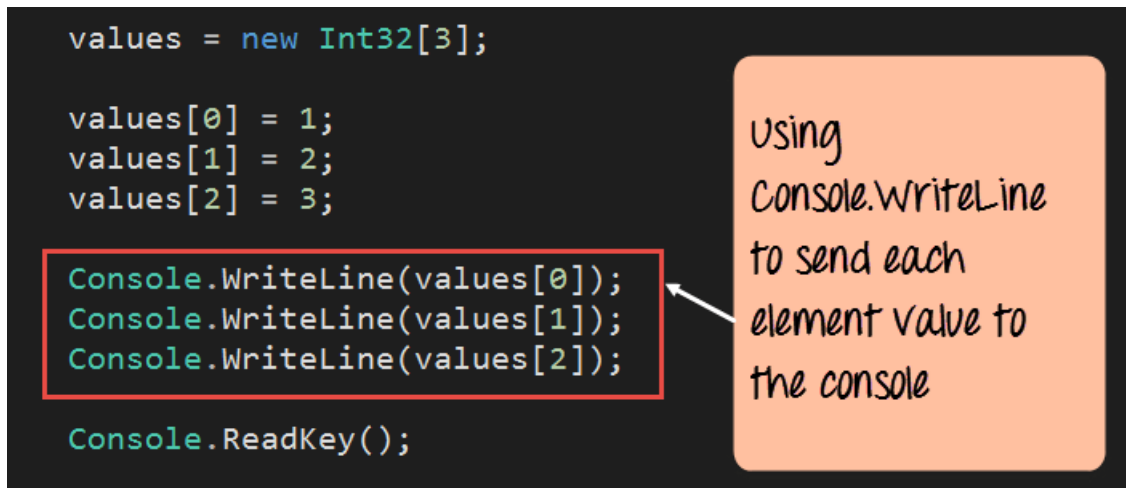
Step 3) Let's now display the individual elements of the array in the Console. Let's add the below code to achieve this.

```
values = new Int32[3];

values[0] = 1;
values[1] = 2;
values[2] = 3;

Console.WriteLine(values[0]);
Console.WriteLine(values[1]);
Console.WriteLine(values[2]);

Console.ReadKey();
```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Int32[] value;
            value=new Int32[3];

            value[0]=1;
            value[1]=2;
            value[2]=3;

            Console.WriteLine(value[0]);
            Console.WriteLine(value[1]);
            Console.WriteLine(value[2]);

            Console.ReadKey();
        }
    }
}
```

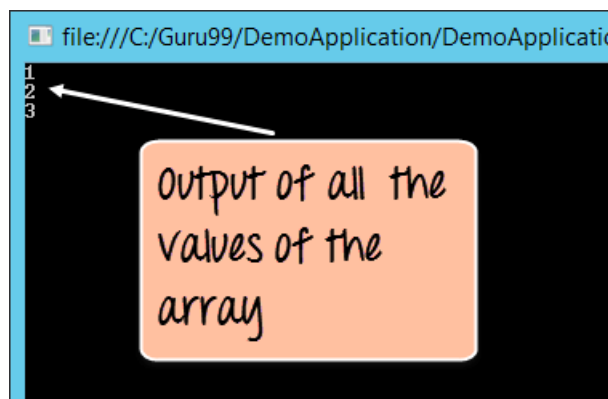

Code Explanation:-

This is the simple part wherein we just use the `Console.WriteLine` method to send each value of the element to the console.

Note that again, we are accessing each element with the help of the array variable name along with the index position.

If the above code is entered properly and the program is executed, the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/DemoApplicatio
1
2
3
```

output of all the values of the array

From the output, you can see all the values of the array being displayed in the Console.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

C# Class & Object Tutorial with Examples

By : Benjamin Walker

🕒 August 10, 2024

C# is based on the C++ programming language. Hence, the C# programming language has in-built support for classes and objects. A class is nothing but an encapsulation of properties and methods that are used to represent a real-time entity.

For example, if you want to work with employee's data in a particular application.

The properties of the employee would be the ID and name of the employee. The methods would include the entry and modification of employee data.

All of these operations can be represented as a class in C#. In this chapter, we will look at how we can work with classes and objects in C# in more detail.

Table of Content:



What is Class and Object?

Let's first begin with classes.

As we discussed earlier classes are an encapsulation of **data properties** and **data methods**.

- The properties are used to describe the data the class will be holding.
- The methods tell what are the operations that can be performed on the data.

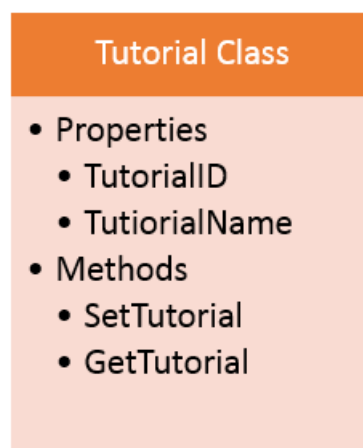
To get a better understanding of class and objects, let's look at an example below of how a class would look like.

The name of the class is “Tutorial”. The class has the following properties

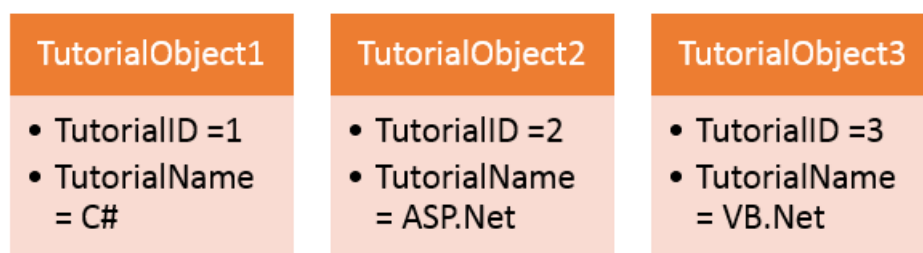
1. **Tutorial ID** – This will be used to store a unique number which would represent the Tutorial.
2. **Tutorial Name** – This will be used to store the name of the tutorial as a string.

A class also comprises of methods. Our class has the following methods,

1. **SetTutorial** – This method would be used to set the ID and name of the Tutorial. So for example, if we wanted to create a tutorial for .Net, we might create an object for this. The object would have an ID of let’s say 1. Secondly, we would assign a name of “.Net” as the name of the Tutorial. The ID value of 1 and the name of “.Net” would be stored as a property of the object.
2. **GetTutorial** – This method would be used to get the details of a specific tutorial. So if we wanted to get the name of the Tutorial, this method would return the string “.Net”.



Below is a snapshot of how an object might look like for our Tutorial class. We have 3 objects, each with their own respective TutorialID and TutorialName.



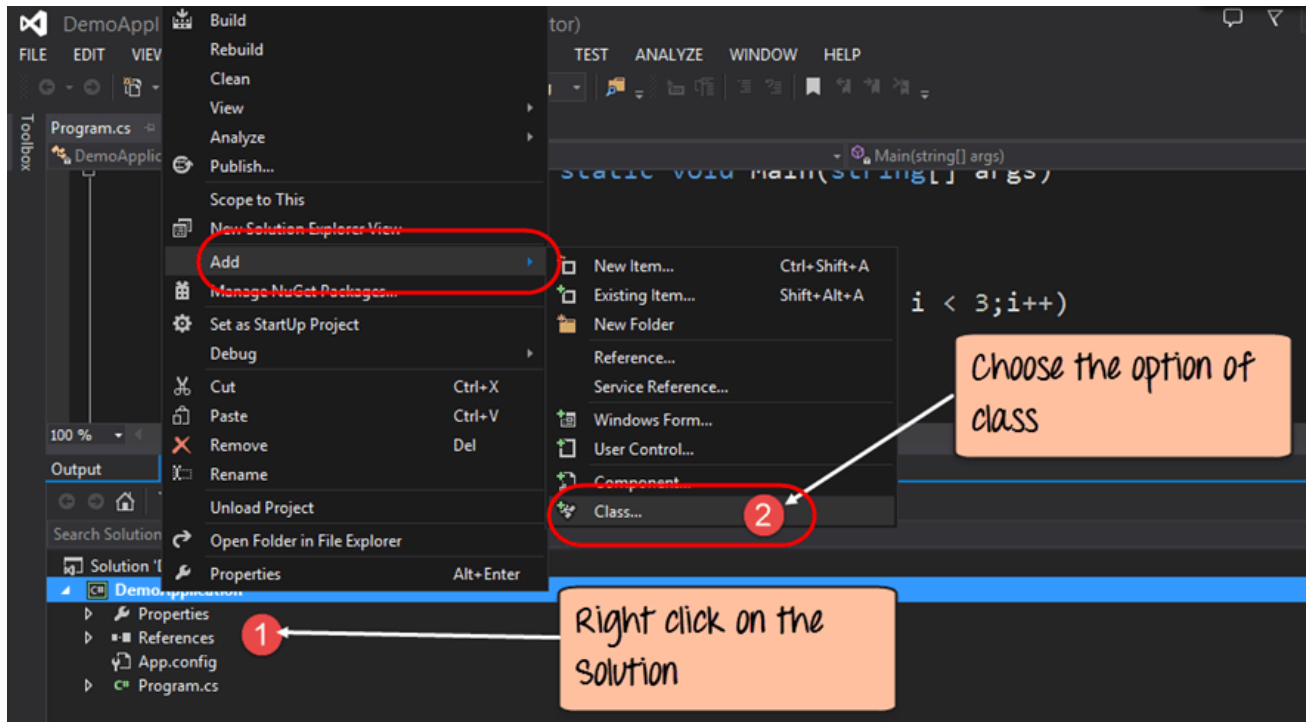
How to Create a Class and Object

Let’s now dive into [Visual Studio](#) to create our class. We are going to build upon our existing

console application which was created in our earlier chapter. We will create a class in Visual Studio for our current application.

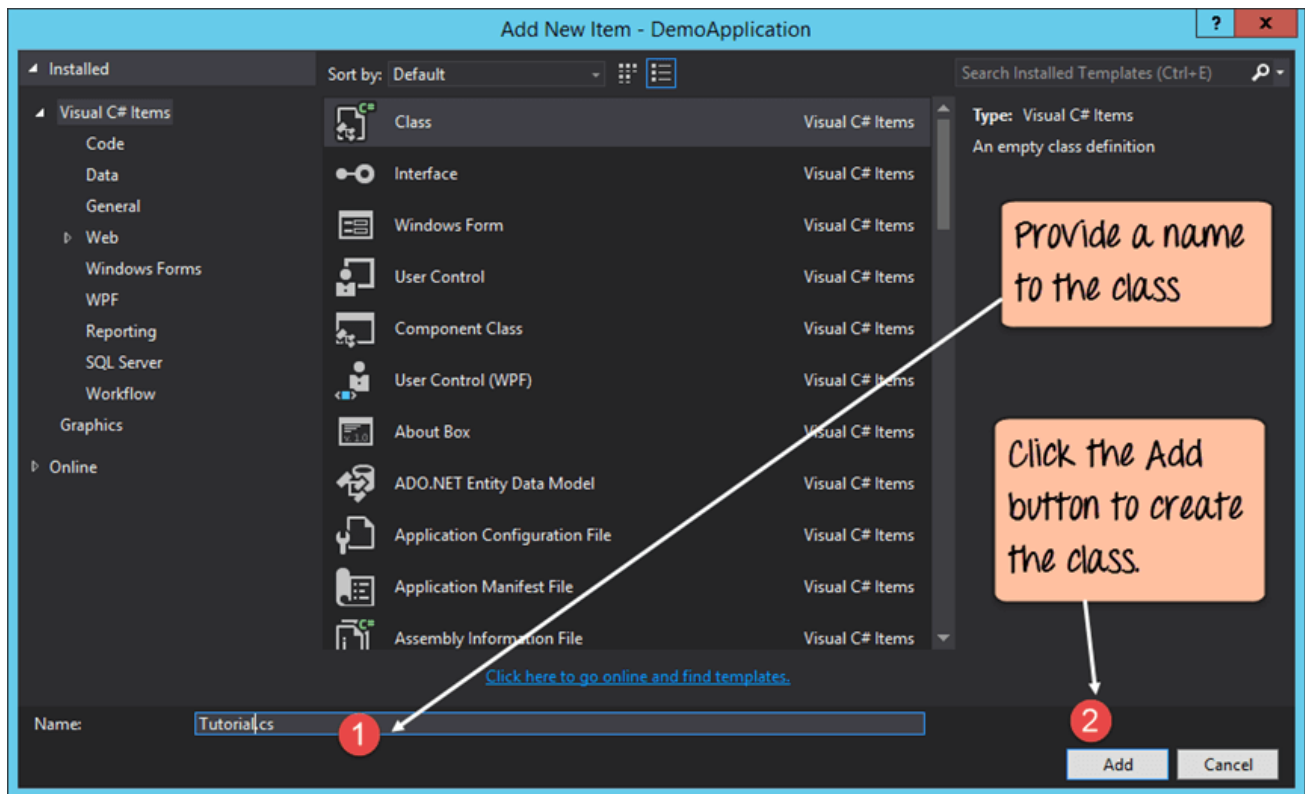
Let's follow the below-mentioned steps to get this example in place.

Step 1) The first step involves the creation of a new class within our existing application. This is done with the help of Visual Studio.



1. The first step is to the right click on the solution, which in our case is 'DemoApplication'. This will bring up a context menu with a list of options.
2. From the context menu choose the option Add->Class. This will provide the option to add a class to the existing project.

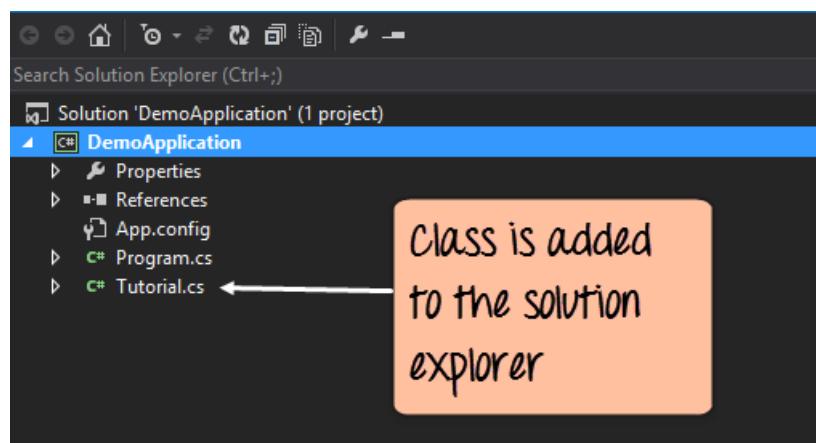
Step 2) The next step is to provide a name for the class and add it to our solution.



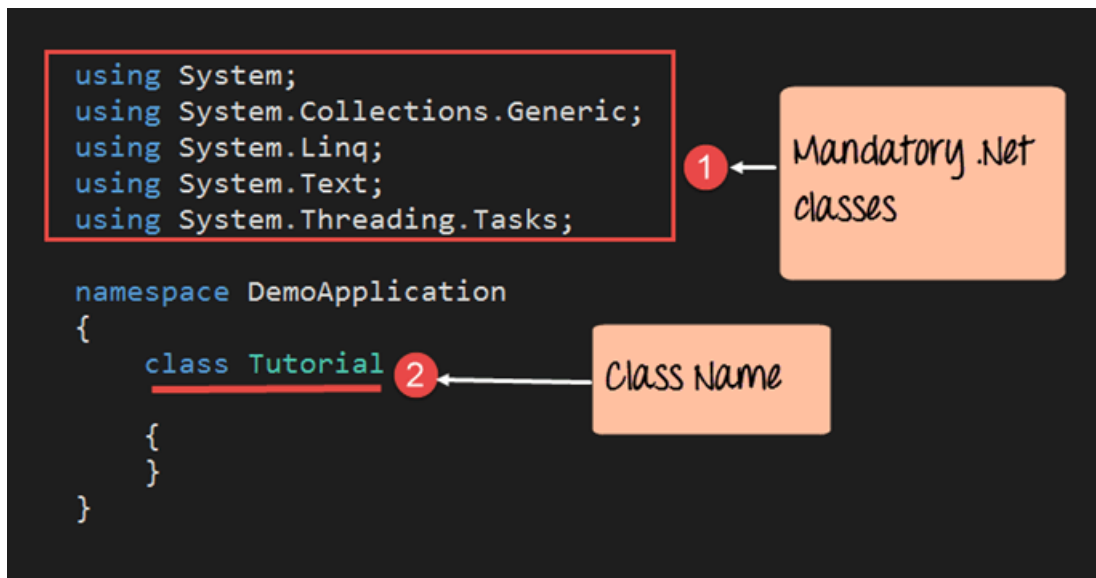
1. In the project dialog box, we first need to provide a name for our class. Let's provide a name of Tutorial.cs for our class. Note that the file name should end with .cs to ensure it is treated as a proper class file.
2. When we click the Add button, the class will be added to our solution.

If the above steps are followed, you will get the below output in Visual Studio.

Output:-



A class named Tutorial.cs will be added to the solution. If you open the file, you will find the below code added to the class file.



Code Explanation:-

1. The first part contains the mandatory modules which Visual Studio adds to any .Net file. These modules are always required to ensure any .Net program runs in a Windows environment.
2. The second part is the class which is added to the file. The class name is 'Tutorial' in our case. This is the name which was specified with the class was added to the solution.

For the moment, our class file does not do anything. In the following topics, we will look into more details on how to work with the class.

Fields and methods

We have already seen how fields and methods are defined in classes in the earlier topic.

For our Tutorial class, we can have the following properties.

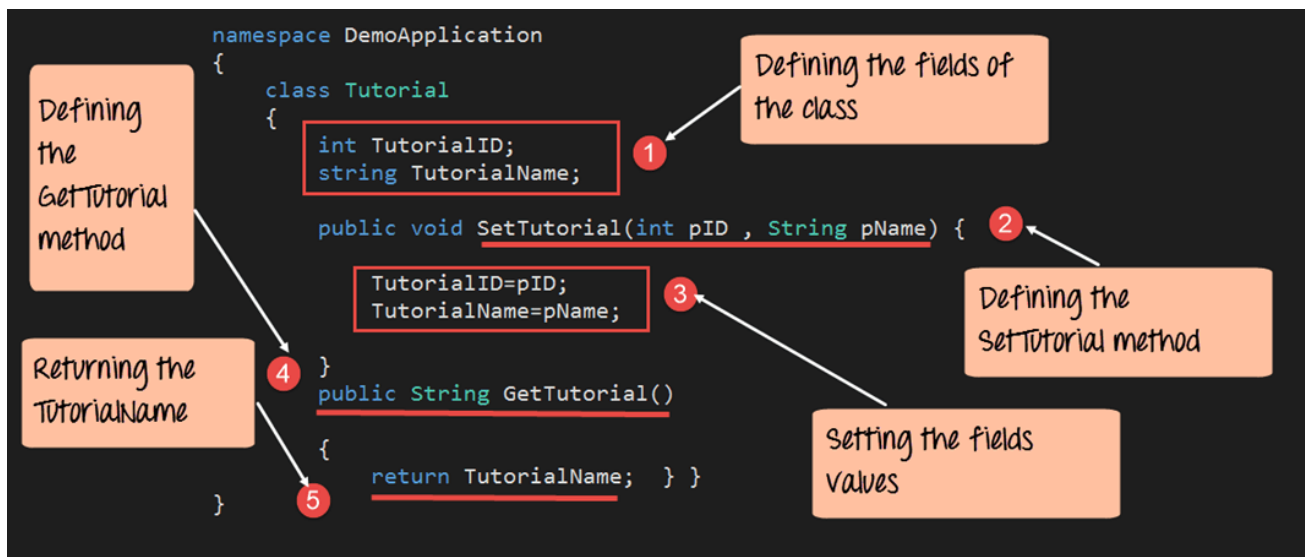
1. Tutorial ID – This will be used to store a unique number which would represent the Tutorial.
2. Tutorial Name – This will be used to store the name of the tutorial as a **string**.

Our Tutorial class can also have the below-mentioned methods.

1. SetTutorial – This method would be used to set the ID and name of the Tutorial.
2. GetTutorial – This method would be used to get the details of a specific tutorial.

Let's now see how we can incorporate fields and methods in our code.

Step 1) The first step is to ensure the Tutorial class has the right fields and methods defined. In this step, we add the below code to the Tutorial.cs file.

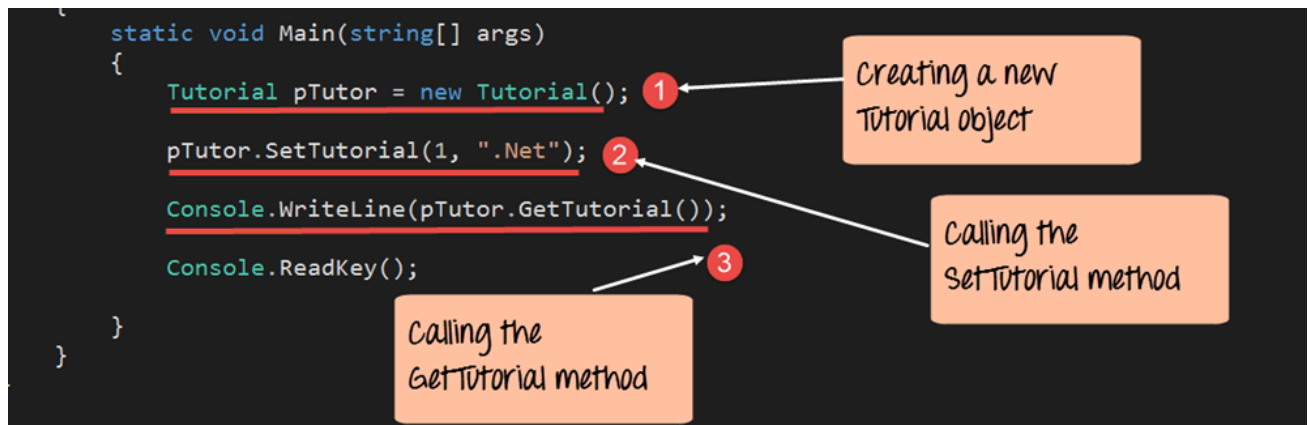


Code Explanation:-

1. The first step is to add the fields of `TutorialID` and `TutorialName` to the class file. Since the `TutorialID` field will be a number, we define it as an Integer, while `TutorialName` will be defined as a string.
2. Next, we define the `SetTutorial` method. This method accepts 2 parameters. So if `Program.cs` calls the `SetTutorial` method, it would need to provide the values to these parameters. These values will be used to set the fields of the `Tutorial` object.
Note: let's take an example and assume our `Program.cs` file calls the `SetTutorial` with the parameters "1" and ".Net". The below steps would be executed as a result of this,
 1. The value of `pID` would become 1
 2. The value of `pName` would be .Net.
 3. In the `SetTutorial` method, these values would then be passed to `TutorialID` and `TutorialName`.
 4. So now `TutorialID` would have a value of 1 and `TutorialName` would have a value of ".Net".
3. Here we set the fields of the `Tutorial` class to the parameters accordingly. So we set `TutorialID` to `pID` and `TutorialName` to `Pname`.
4. We then define the `GetTutorial` method to return value of the type "String". This method will be used to return the `TutorialName` to the calling program. Likewise, you can also get the tutorial id with method `Int GetTutorial`
5. Here we return the value of the `TutorialName` field to the calling program.

Step 2) Now let's add code to our `Program.cs`, which is our Console application. The Console application will be used to create an object of the "Tutorial class" and call the `SetTutorial` and `GetTutorial` methods accordingly.

(**Note:-** An object is an instance of a class at any given time. The difference between a class and an object is that the object contains values for the properties.)



RELATED ARTICLES

- [C# Windows Forms Application Tutorial with Example](#)
- [Data Types in C#: Double, Integer, Float, Char](#)
- [C# Enum\(Enumeration\) with Example](#)
- [C# Queue with Examples: What is C# Queue and How to Use?](#)

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Tutorial
    {
        int TutorialID;
        string TutorialName;

        public void SetTutorial(int pID,string pName)
        {
            TutorialID=pID;
            TutorialName=pName;
        }
        public String GetTutorial()
        {
            return TutorialName;
        }

        static void Main(string[] args)
        {
            Tutorial pTutor=new Tutorial();

            pTutor.SetTutorial(1,".Net");

            Console.WriteLine(pTutor.GetTutorial());

            Console.ReadKey();
        }
    }
}

```

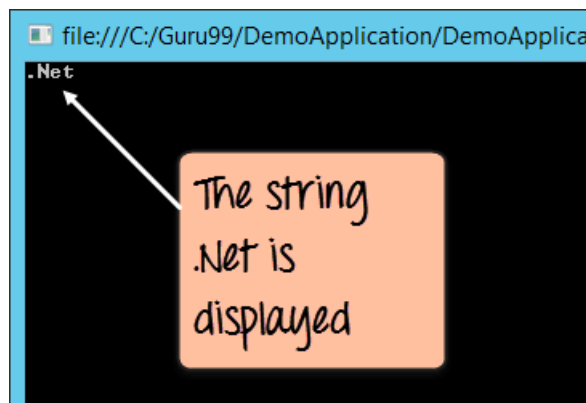
Code Explanation:-

1. The first step is to create an object for the Tutorial class. Mark here that this is done by using the keyword 'new'. The 'new' keyword is used to create an object from a class in C#. The object is then assigned to the pTutor **variable**.
2. The method SetTutorial is then called. The parameters of 1 and ".Net" are passed to the SetTutorial method. These will then be used to set the "TutorialID" and "TutorialName" fields of the class accordingly.
3. We then use the GetTutorial method of the Tutorial class to get the TutorialName. This is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is run the following output will be

displayed.

Output:



From the output, you can clearly see that the string “.Net” was returned by the GetTutorial method.

Summary

- The class is an encapsulation of data properties and methods. The properties are used to define the type of data in the class. The methods define the operations which can be performed on the data.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Your Email Address

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

C# Class & Object Tutorial with Examples

By : Benjamin Walker

🕒 August 10, 2024

C# is based on the C++ programming language. Hence, the C# programming language has in-built support for classes and objects. A class is nothing but an encapsulation of properties and methods that are used to represent a real-time entity.

For example, if you want to work with employee's data in a particular application.

The properties of the employee would be the ID and name of the employee. The methods would include the entry and modification of employee data.

All of these operations can be represented as a class in C#. In this chapter, we will look at how we can work with classes and objects in C# in more detail.

Table of Content:



What is Class and Object?

Let's first begin with classes.

As we discussed earlier classes are an encapsulation of **data properties** and **data methods**.

- The properties are used to describe the data the class will be holding.
- The methods tell what are the operations that can be performed on the data.

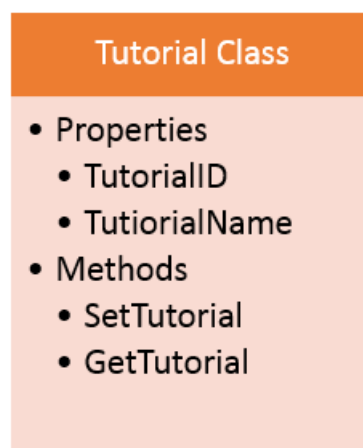
To get a better understanding of class and objects, let's look at an example below of how a class would look like.

The name of the class is “Tutorial”. The class has the following properties

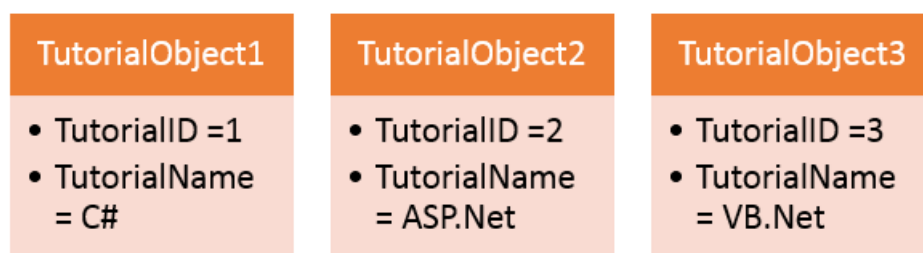
1. **Tutorial ID** – This will be used to store a unique number which would represent the Tutorial.
2. **Tutorial Name** – This will be used to store the name of the tutorial as a string.

A class also comprises of methods. Our class has the following methods,

1. **SetTutorial** – This method would be used to set the ID and name of the Tutorial. So for example, if we wanted to create a tutorial for .Net, we might create an object for this. The object would have an ID of let’s say 1. Secondly, we would assign a name of “.Net” as the name of the Tutorial. The ID value of 1 and the name of “.Net” would be stored as a property of the object.
2. **GetTutorial** – This method would be used to get the details of a specific tutorial. So if we wanted to get the name of the Tutorial, this method would return the string “.Net”.



Below is a snapshot of how an object might look like for our Tutorial class. We have 3 objects, each with their own respective TutorialID and TutorialName.



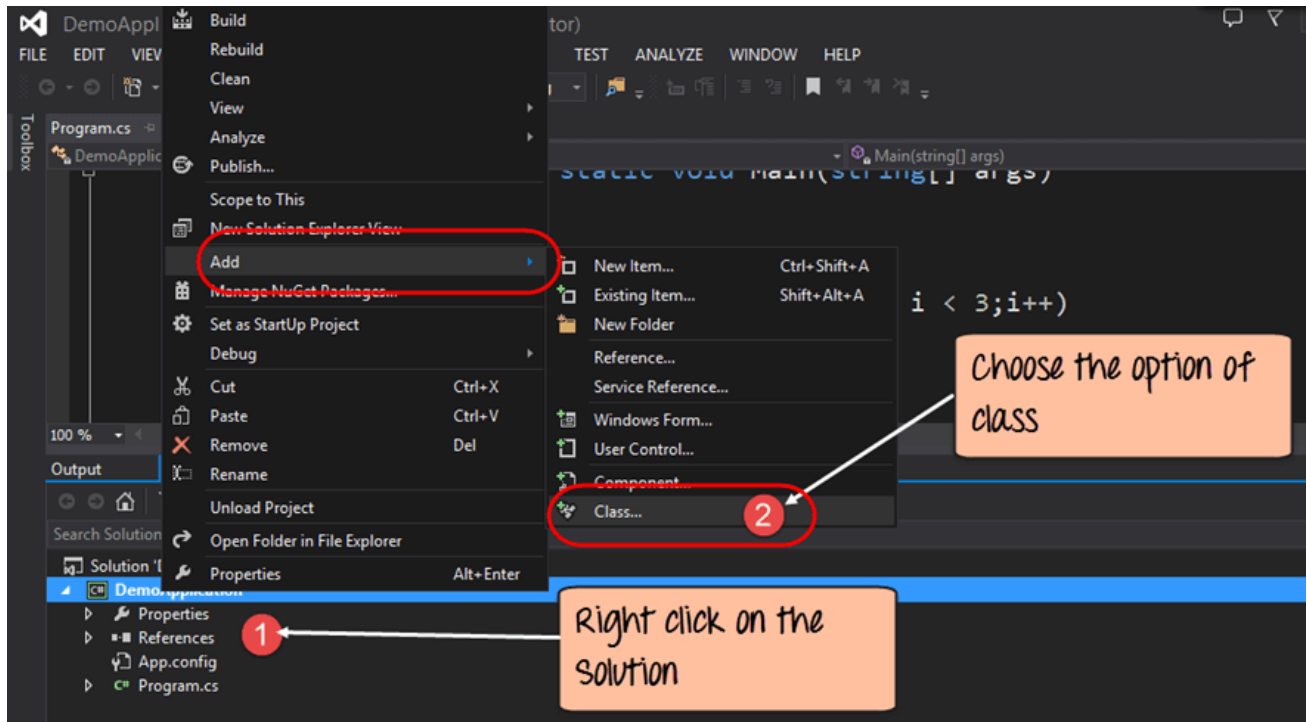
How to Create a Class and Object

Let’s now dive into [Visual Studio](#) to create our class. We are going to build upon our existing

console application which was created in our earlier chapter. We will create a class in Visual Studio for our current application.

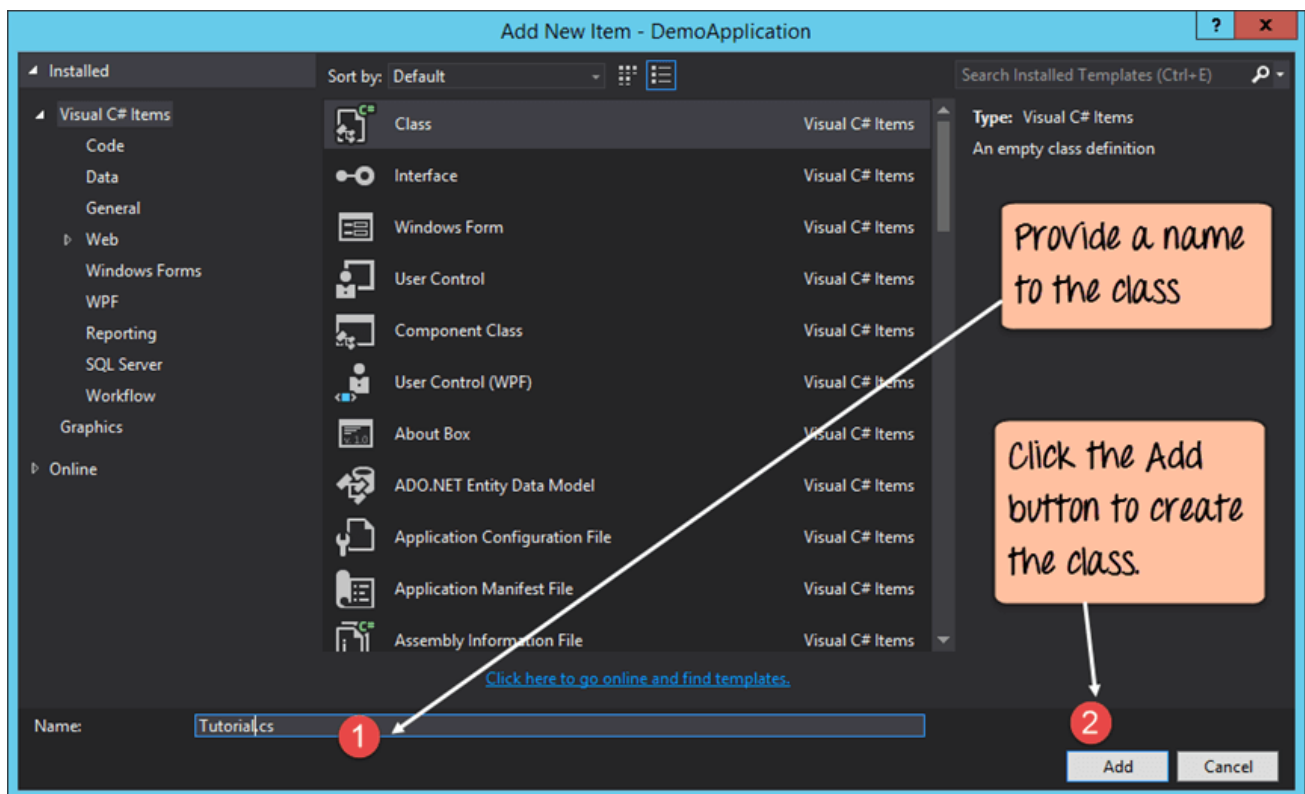
Let's follow the below-mentioned steps to get this example in place.

Step 1) The first step involves the creation of a new class within our existing application. This is done with the help of Visual Studio.



1. The first step is to the right click on the solution, which in our case is 'DemoApplication'. This will bring up a context menu with a list of options.
2. From the context menu choose the option Add->Class. This will provide the option to add a class to the existing project.

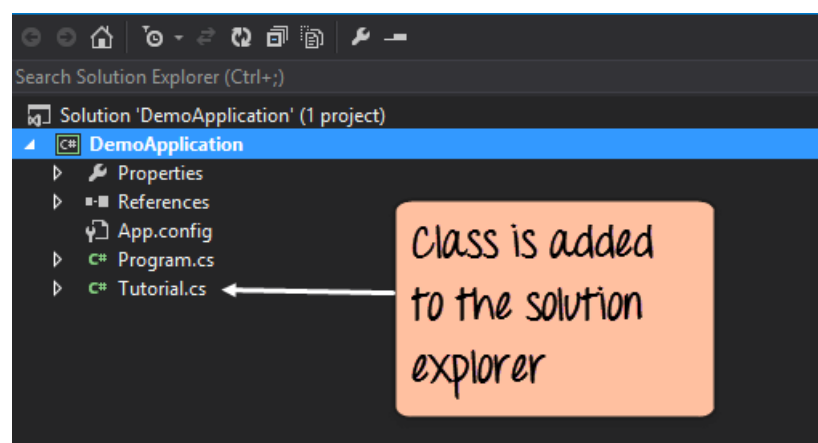
Step 2) The next step is to provide a name for the class and add it to our solution.



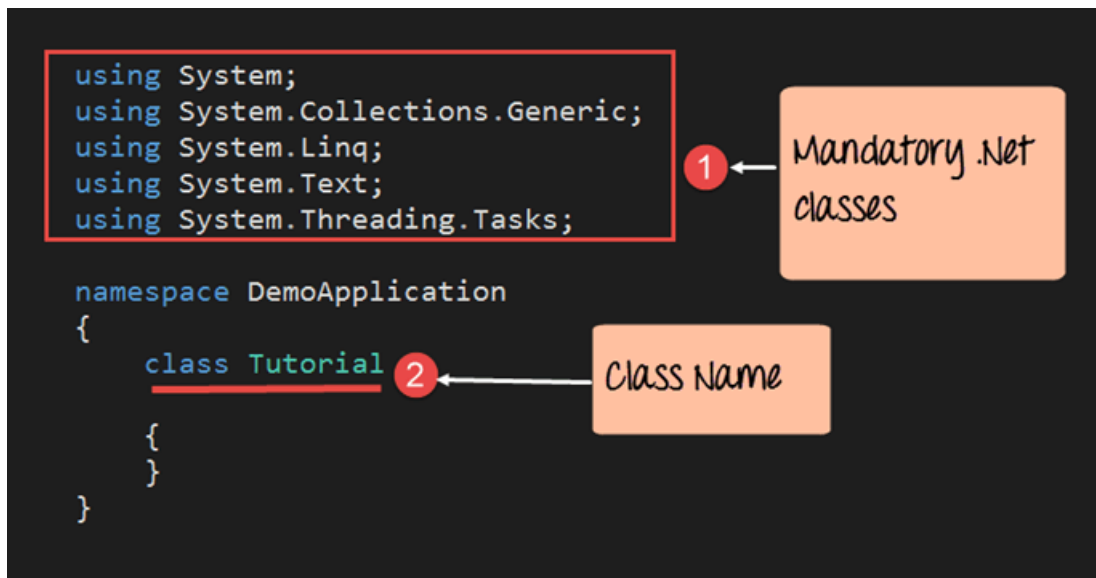
1. In the project dialog box, we first need to provide a name for our class. Let's provide a name of Tutorial.cs for our class. Note that the file name should end with .cs to ensure it is treated as a proper class file.
2. When we click the Add button, the class will be added to our solution.

If the above steps are followed, you will get the below output in Visual Studio.

Output:-



A class named Tutorial.cs will be added to the solution. If you open the file, you will find the below code added to the class file.



Code Explanation:-

1. The first part contains the mandatory modules which Visual Studio adds to any .Net file. These modules are always required to ensure any .Net program runs in a Windows environment.
2. The second part is the class which is added to the file. The class name is 'Tutorial' in our case. This is the name which was specified with the class was added to the solution.

For the moment, our class file does not do anything. In the following topics, we will look into more details on how to work with the class.

Fields and methods

We have already seen how fields and methods are defined in classes in the earlier topic.

For our Tutorial class, we can have the following properties.

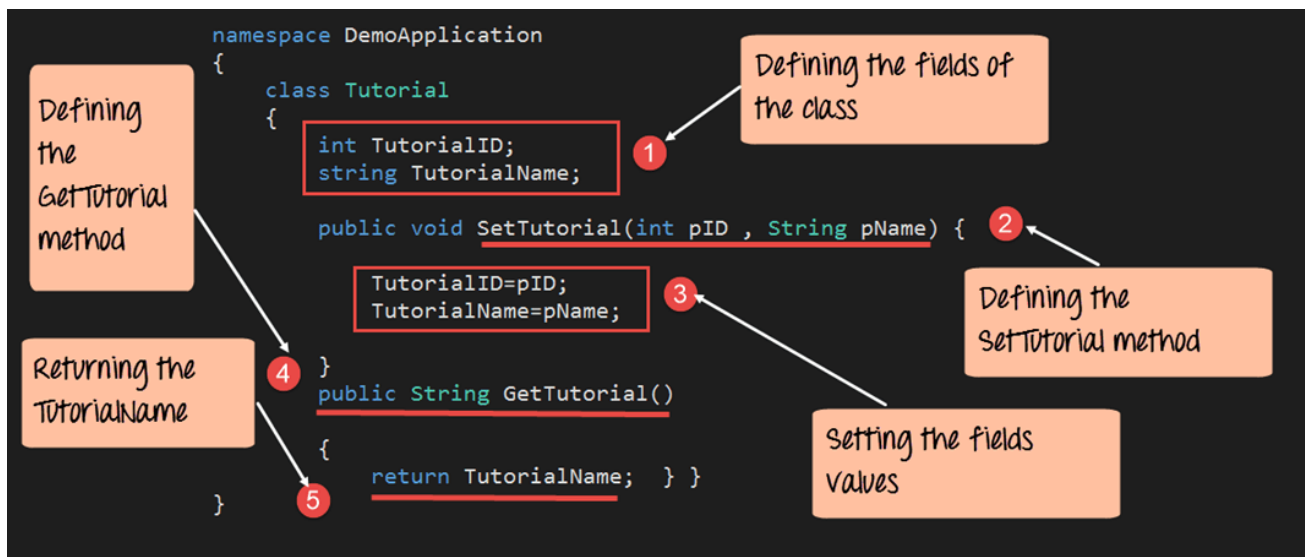
1. Tutorial ID – This will be used to store a unique number which would represent the Tutorial.
2. Tutorial Name – This will be used to store the name of the tutorial as a **string**.

Our Tutorial class can also have the below-mentioned methods.

1. SetTutorial – This method would be used to set the ID and name of the Tutorial.
2. GetTutorial – This method would be used to get the details of a specific tutorial.

Let's now see how we can incorporate fields and methods in our code.

Step 1) The first step is to ensure the Tutorial class has the right fields and methods defined. In this step, we add the below code to the Tutorial.cs file.

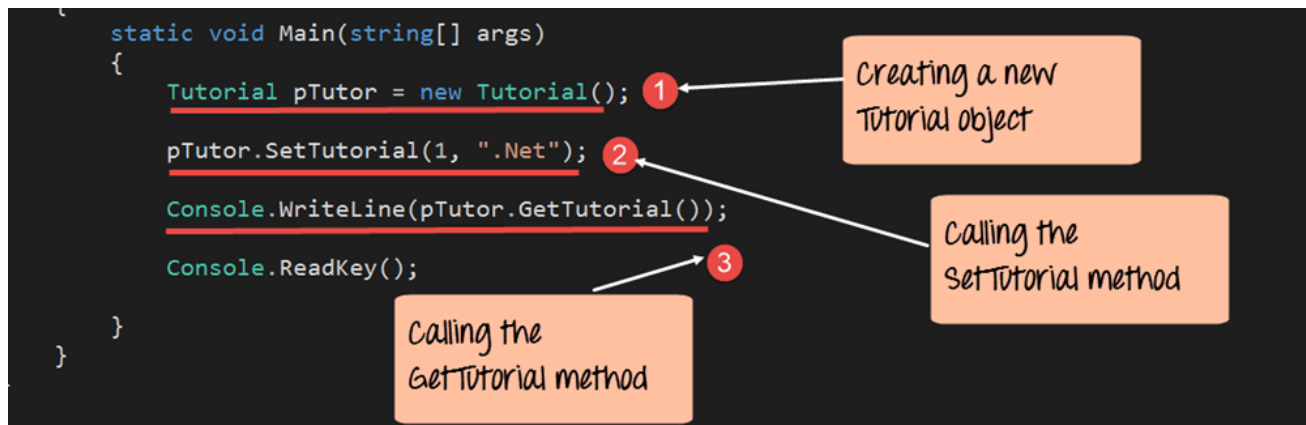


Code Explanation:-

1. The first step is to add the fields of `TutorialID` and `TutorialName` to the class file. Since the `TutorialID` field will be a number, we define it as an Integer, while `TutorialName` will be defined as a string.
2. Next, we define the `SetTutorial` method. This method accepts 2 parameters. So if `Program.cs` calls the `SetTutorial` method, it would need to provide the values to these parameters. These values will be used to set the fields of the `Tutorial` object.
Note: let's take an example and assume our `Program.cs` file calls the `SetTutorial` with the parameters "1" and ".Net". The below steps would be executed as a result of this,
 1. The value of `pID` would become 1
 2. The value of `pName` would be .Net.
 3. In the `SetTutorial` method, these values would then be passed to `TutorialID` and `TutorialName`.
 4. So now `TutorialID` would have a value of 1 and `TutorialName` would have a value of ".Net".
3. Here we set the fields of the `Tutorial` class to the parameters accordingly. So we set `TutorialID` to `pID` and `TutorialName` to `Pname`.
4. We then define the `GetTutorial` method to return value of the type "String". This method will be used to return the `TutorialName` to the calling program. Likewise, you can also get the tutorial id with method `Int GetTutorial`
5. Here we return the value of the `TutorialName` field to the calling program.

Step 2) Now let's add code to our `Program.cs`, which is our Console application. The Console application will be used to create an object of the "Tutorial class" and call the `SetTutorial` and `GetTutorial` methods accordingly.

(**Note:-** An object is an instance of a class at any given time. The difference between a class and an object is that the object contains values for the properties.)



RELATED ARTICLES

- [C# Windows Forms Application Tutorial with Example](#)
- [Data Types in C#: Double, Integer, Float, Char](#)
- [C# Enum\(Enumeration\) with Example](#)
- [C# Queue with Examples: What is C# Queue and How to Use?](#)

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Tutorial
    {
        int TutorialID;
        string TutorialName;

        public void SetTutorial(int pID,string pName)
        {
            TutorialID=pID;
            TutorialName=pName;
        }
        public String GetTutorial()
        {
            return TutorialName;
        }

        static void Main(string[] args)
        {
            Tutorial pTutor=new Tutorial();

            pTutor.SetTutorial(1,".Net");

            Console.WriteLine(pTutor.GetTutorial());

            Console.ReadKey();
        }
    }
}

```

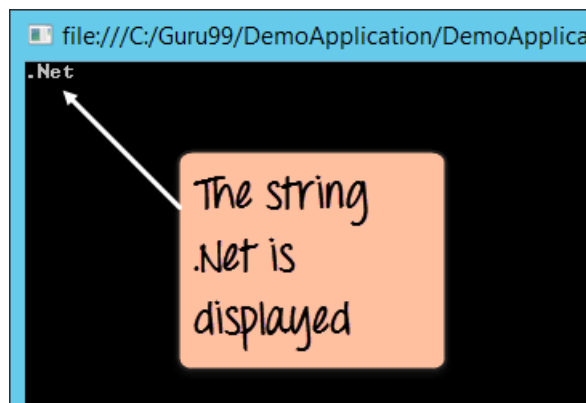
Code Explanation:-

1. The first step is to create an object for the Tutorial class. Mark here that this is done by using the keyword 'new'. The 'new' keyword is used to create an object from a class in C#. The object is then assigned to the pTutor **variable**.
2. The method SetTutorial is then called. The parameters of 1 and ".Net" are passed to the SetTutorial method. These will then be used to set the "TutorialID" and "TutorialName" fields of the class accordingly.
3. We then use the GetTutorial method of the Tutorial class to get the TutorialName. This is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is run the following output will be

displayed.

Output:



From the output, you can clearly see that the string “.Net” was returned by the GetTutorial method.

Summary

- The class is an encapsulation of data properties and methods. The properties are used to define the type of data in the class. The methods define the operations which can be performed on the data.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Your Email Address

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

Access Modifiers (Specifiers) in C# with Program Examples

By : Benjamin Walker

🕒 August 10, 2024

What is Access Modifier (Specifier) in C#?

Access Modifiers or Access Specifiers in C# are the keywords used to define the visibility of a class property or method. It is used when you don't want other programs to see the properties or methods of a class. Access modifiers restrict access so that other programs cannot see the properties or methods of a class.

There are 6 types of access modifiers in C#:

- Private
- Public
- Protected
- Internal
- Protected Internal
- Private Protected

Table of Content:



We will learn about the main access modifiers in C# with program examples as explained below.

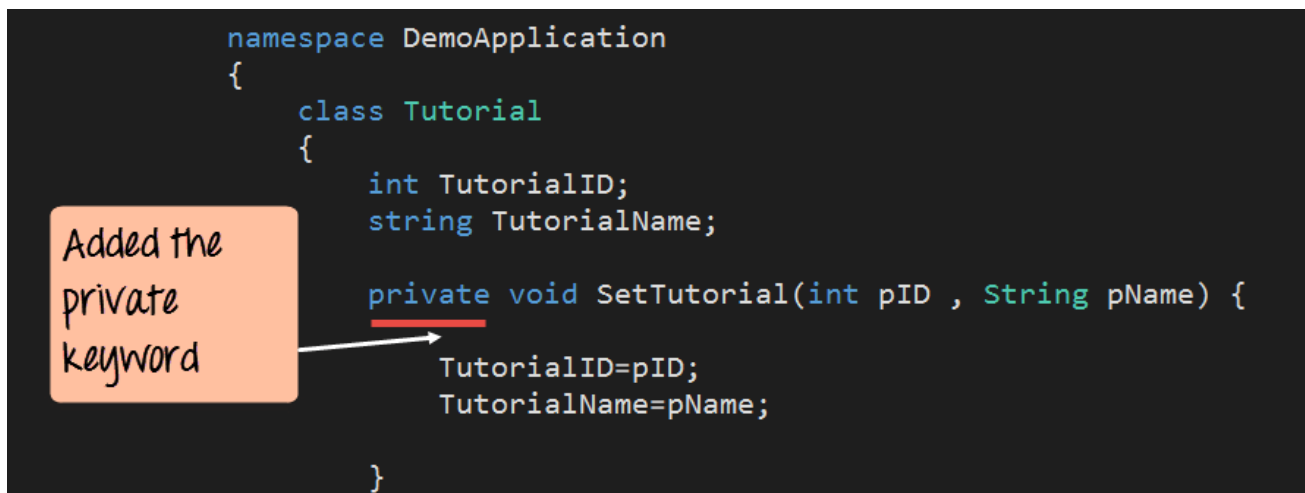
Private Access Modifiers in C#

When Private access modifier is attached to either a property or a method, it means that those members cannot be accessed from any external program.

Example of Private Access Modifier

Let's take an example and see what happens when we use the private access modifier.

Let's modify the current code in our Tutorial.cs file. In the SetTutorial method, let's change the public keyword to private.



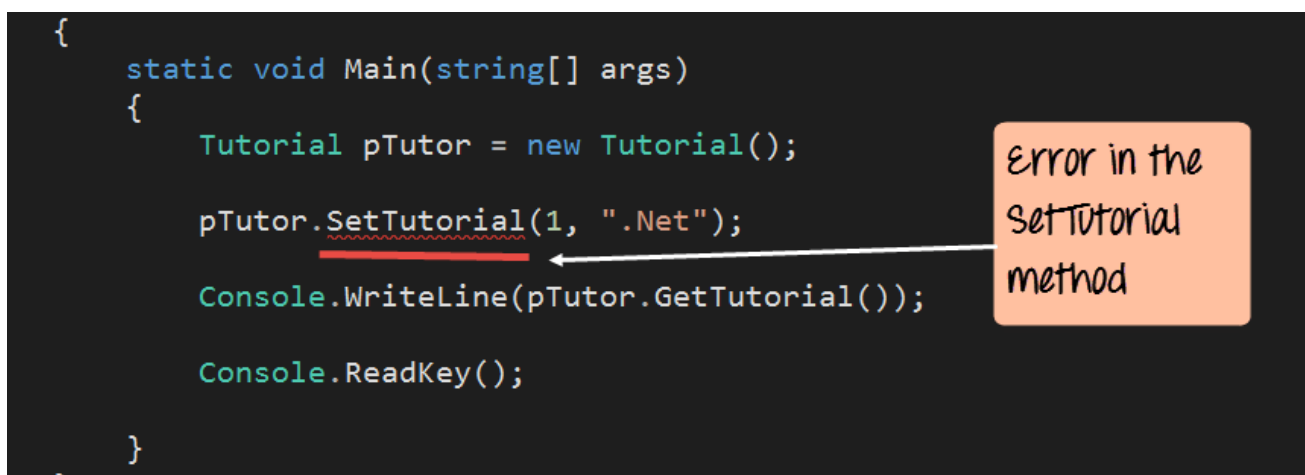
```
namespace DemoApplication
{
    class Tutorial
    {
        int TutorialID;
        string TutorialName;

        private void SetTutorial(int pID , String pName) {
            TutorialID=pID;
            TutorialName=pName;
        }
    }
}
```

Added the private keyword

Now let's switchover to our Program.cs file. You will notice that there is a red squiggly line under the SetTutorial method.

Since we have now declared the SetTutorial method as private in our Tutorial class, Visual Studio has detected this. It has told the user by highlighting it that now this method will not work from the Program.cs file.



```
{
    static void Main(string[] args)
    {
        Tutorial pTutor = new Tutorial();
        pTutor.SetTutorial(1, ".Net");
        Console.WriteLine(pTutor.GetTutorial());
        Console.ReadKey();
    }
}
```


Error in the SetTutorial method

C# Public Access Modifiers

When Public access modifier is attached to either a property or a method, it means that

those members can be accessed from any external program. We have already seen this in our earlier examples.

Example of Public Access Modifier



```
int TutorialID;  
string TutorialName;  
  
public void SetTutorial(int pID , String pName) {  
    TutorialID=pID;  
    TutorialName=pName;  
}  
public String GetTutorial()  
{
```

Since we have defined our methods as public in the Tutorial class, they can be accessed from the Program.cs file.

Protected Access Modifiers in C#

When Protected access modifier is attached to either a property or a method, it means that those members can be accessed only by **classes inherited** from the current **class**. This will be explained in more detail in the Inheritance class.

C# Internal Access Modifiers

When an Internal access modifier is attached to either a property or a method, those members can be accessed only by an internal program but not by an external program.

C# Constructor

C# Constructors are used to initializing the values of class fields when their corresponding objects are created. A constructor is a method which has the same name as that of the class. If a constructor is defined in a class, then it will provide the first method which is called when an object is created. Suppose if we had a class called Employee. The constructor method would also be named as Employee().

The following key things need to be noted about constructor methods

1. The C# default access modifier for the constructor needs to be made as public.
2. There should be no return type for the constructor method.

Example of C# Constructor

Let's now see how we can incorporate the use of constructors in our code. We will use constructors to initialize the TutorialID and TutorialName fields to some default values when the object is created.

Step 1) The first step is to create the constructor for our Tutorial class. In this step, we add the below code to the Tutorial.cs file.

```
namespace DemoApplication
{
    public class Tutorial
    {
        int TutorialID;
        string TutorialName;
        public Tutorial() {
            TutorialID = 0;
            TutorialName = "Default";
        }
        public void SetTutorial(int pID , String pName) {

            TutorialID=pID;
            TutorialName=pName;

        }
    }
}
```

The diagram illustrates the constructor code in the `Tutorial` class. A box labeled "Constructor" points to the `public Tutorial()` method signature, which is marked with a red circle containing the number 1. Another box labeled "Initializing the class fields" points to the initialization code `TutorialID = 0; TutorialName = "Default";`, which is marked with a red circle containing the number 2.

Code Explanation:-

1. We first add a new method which has the same name as that of the class. Because it is the same name as the class, C# treats this as a constructor method. So now whenever the calling method creates an object of this class, this method will be called by default.
2. In the Tutorial constructor, we are setting the value of TutorialID to 0 and TutorialName to "Default". So whenever an object is created, these fields will always have these default values.

Now let's switchover to our Program.cs file and just remove the line, which calls the SetTutorial method. This is because we want just to see how the constructor works.

```
static void Main(string[] args)
{
    Tutorial pTutor = new Tutorial();
    Console.WriteLine(pTutor.GetTutorial());
    Console.ReadKey();
}
```

The diagram illustrates the `Main` method in the `Program.cs` file. A box labeled "Create Tutorial object" points to the line `Tutorial pTutor = new Tutorial();`, which is marked with a red circle containing the number 1. Another box labeled "Invoke GetTutorial method" points to the line `Console.WriteLine(pTutor.GetTutorial());`, which is marked with a red circle containing the number 2.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Tutorial
    {
        public int TutorialID;
        public string TutorialName;

        public Tutorial()
        {
            TutorialID=0;
            TutorialName="Default";
        }
        public void SetTutorial(int pID,string pName)
        {
            TutorialID=pID;
            TutorialName=pName;
        }
        public String GetTutorial()
        {
            return TutorialName;
        }

        static void Main(string[] args)
        {
            Tutorial pTutor=new Tutorial();

            Console.WriteLine(pTutor.GetTutorial());

            Console.ReadKey();
        }
    }
}

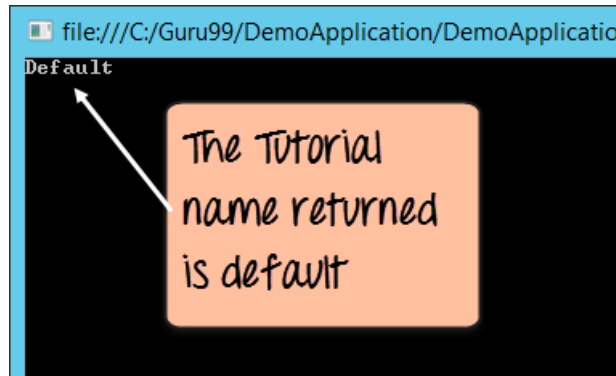
```

Code Explanation:-

1. The first step is to create an object for the Tutorial class. This is done via the 'new' keyword.
2. We use the GetTutorial method of the Tutorial class to get the TutorialName. This is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is executed, the following output will be displayed.

Output:



From the output, we can see that the constructor was indeed called and that the value of the TutorialName was set to “Default”.

Note: Here the value “default” is fetched from the constructor.

Summary

- C# Access Modifiers or Access Specifiers are used to define the visibility of a class property or method.
- When Private access modifier is attached to either a property or a method, it means that those members cannot be accessed from any external program.
- When Public access modifier is attached to either a property or a method, it means that those members can be accessed from any external program.
- When Protected access modifier is attached to either a property or a method, it means that those members can be accessed only by classes inherited from the current class.
- When an Internal access modifier is attached to either a property or a method, those members can be accessed only by an internal program but not by an external program.
- C# Constructors are used to initializing the values of class fields when their corresponding objects are created.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Your Email Address

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

C# Inheritance and Polymorphism with Program Examples

By : Benjamin Walker

August 10, 2024 ⌚

?#What is Inheritance in C

Inheritance is an important concept of C#. Inheritance is a concept in which you define parent classes and child classes. The child classes inherit methods and properties of the parent class, but at the same time, they can also modify the behavior of the methods if required. The child class can also define methods of its own if required.

[:Table of Content](#)



:Let's get a better understanding of C# Inheritance by a Program Example

C# Inheritance Example

.Let's now see how we can incorporate the concept of inheritance in our code

Step 1) The first step is to change the code for our Tutorial class. In this step, we add the below code to the Tutorial.cs file

```
namespace DemoApplication
{
    public class Tutorial
    {
        protected int TutorialID;
        protected string TutorialName;

        public void SetTutorial(int pID , String pName) {

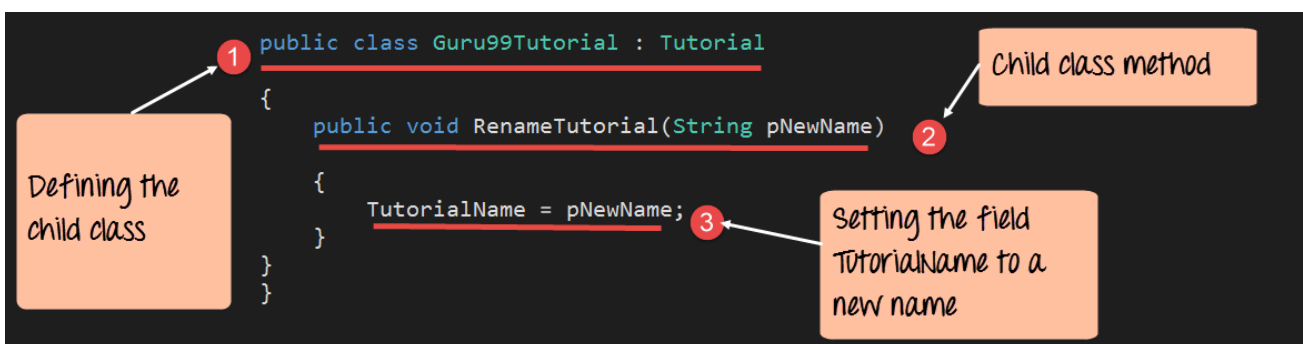
            TutorialID=pID;
            TutorialName=pName;
        }
    }
}
```

Mark the fields as protected.

Note that we need to now add the access modifier of 'protected' to both the TutorialID and TutorialName field

Remember we had mentioned this access modifier in the Access Modifier tutorial. Well here you can see the purpose of having this. Only when you have this access modifier .(protected), the child class be able to use the fields of the parent class

Step 2) The second step is to add our new child class. The name of this class will be “Guru99Tutorial”. In this step, we add the below code to the Tutorial.cs file. The code should .be placed after the Tutorial class definition



-:Code Explanation

The first step is to create the Guru99Tutorial child class. We also need to mention .1 that this class is going to be a child class of the Tutorial class. This is done by the ‘:’ keyword

Next, we are defining a method called RenameTutorial. It will be used to rename the .2 TutorialName field. This method accepts a string variable which contains the new .name of the Tutorial

.We then assigned the parameter pNewName to the TutorialName field .3

Note: – Even though we have not defined the TutorialName field in the “Guru99Tutorial” class, we are still able to access this field. This is because of the fact that “Guru99Tutorial” is a child class of Tutorial class. And because we made .the fields of the Tutorial class as protected, they can be accessed by this class

Step 3) The last step is to modify our main Program.cs file. In our console application, we are going to make an object of the Guru99Tutorial class. With this object, we are going to call the RenameTutorial method. We are then going to display the TutorialName field with the .help of the GetTutorial method

```
class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        Guru99Tutorial pTutor = new Guru99Tutorial();
```

```
        pTutor.RenameTutorial(".Net by Guru99");
```

```
        Console.WriteLine(pTutor.GetTutorial(););
```

```
        Console.ReadKey();
```

1

Creating object of
child class

2

Child class method

3

Child class method


```

        ;using System
;using System.Collections.Generic
        ;using System.Linq
        ;using System.Text
;using System.Threading.Tasks
        namespace DemoApplication
        {
            public class Tutorial
            {
                ;protected int TutorialID
                ;protected string TutorialName

                (public void SetTutorial(int pID,string pName
                {
                    ;TutorialID=pID
                    ;TutorialName=pName
                }

                ()public String GetTutorial
                {
                    ;return TutorialName
                }

                public class Guru99Tutorial:Tutorial
                {
                    (public void RenameTutorial(String pNewName
                    {
                        ;TutorialName=pNewName
                    }

                    (static void Main(string[] args
                    {
                        ;()Guru99Tutorial pTutor=new Guru99Tutorial

                        ;("pTutor.RenameTutorial("".Net by Guru99

                        ;(()Console.WriteLine(pTutor.GetTutorial

                        ;()Console.ReadKey
                    {
                    {
                    {

```

-:Code Explanation

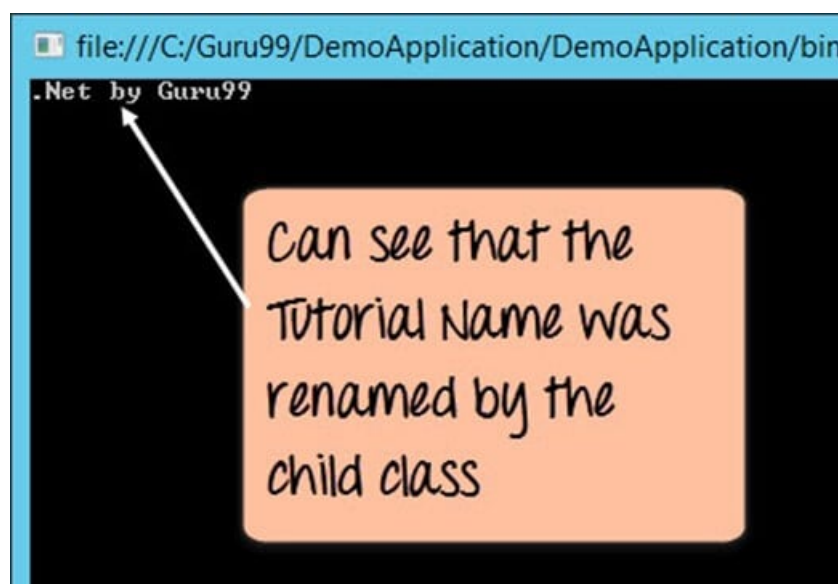
The first step is to create an object for the Guru99Tutorial class. This is done via the `.1` 'new' keyword. Note that this time we are not creating an object of the Tutorial `.class`

We use the RenameTutorial method of the Guru99Tutorial class to change the TutorialName field. We pass the string “.Net by Guru99” to the RenameTutorial method.

We then call the GetTutorial method. Note that even though this method is not defined in the Guru99Tutorial class, we are still able to access this method. The output of the GetTutorial method is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is executed successfully, the following output will be displayed

:Output



From the output, we can clearly see that the TutorialName field was renamed to “.Net by Guru99”. This was made possible of the RenameTutorial method called by the child class

?#What is Polymorphism in C

Polymorphism in C# is an OOPs concept where one name can have many forms. For example, you have a smartphone for communication. The communication mode you choose could be anything. It can be a call, a text message, a picture message, mail, etc. So, the goal is common, that is, communication, but their approach is different. This is called Polymorphism.

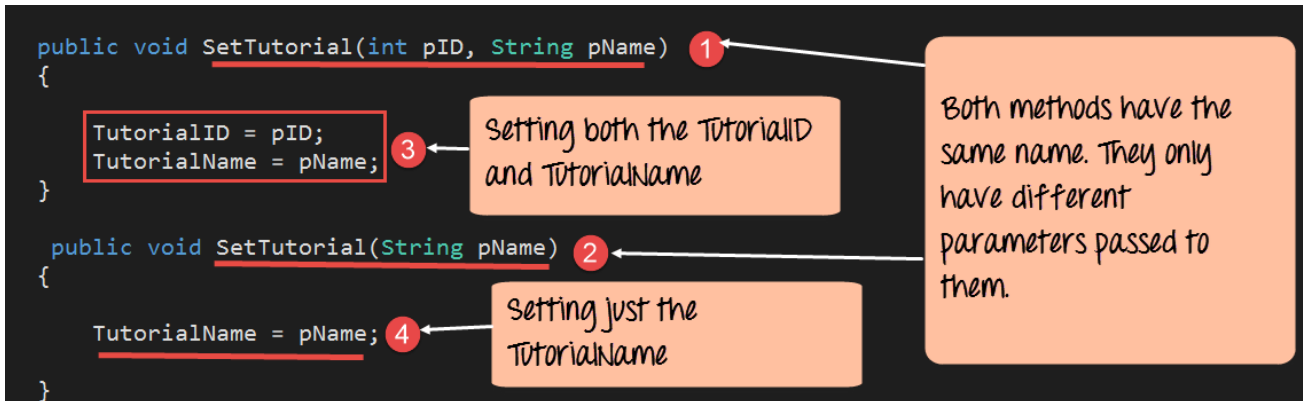
:You will get a better understanding of C# Polymorphism by the below Program Example

C# Polymorphism Example

.Let’s now see, how we can incorporate the concept of Polymorphism in our code

Step 1) The first step is to change the code for our Tutorial class. In this step, we add the

.below code to the Tutorial.cs file



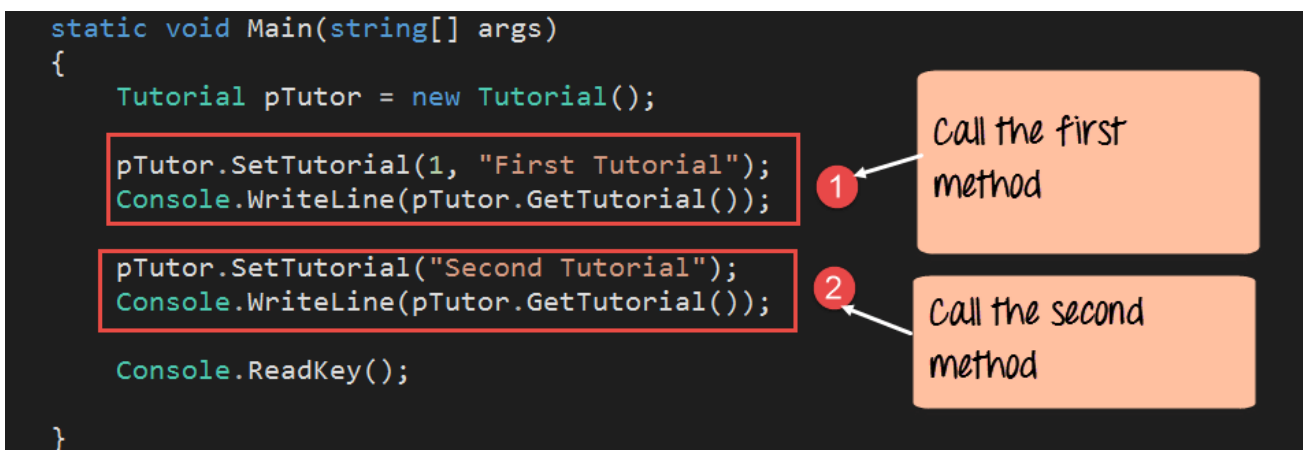
-:Code Explanation

The first step is the same as in our earlier examples. We are keeping the definition of (2 & 1) the `SetTutorial` method as it is

This method sets the `TutorialID` and the `TutorialName` based on the parameters `pID` and (3) `pName`

This is where we make a change to our class wherein we add a new method with the (4) same name of `SetTutorial`. Only this time we are only passing one parameter which is the `pName`. In this method, we are just setting the field of `TutorialName` to `pName`

Step 2) The last step is to modify our main `Program.cs` file. In our console application, we are going to make an object of the `Guru99Tutorial` class



```

        ;using System
;using System.Collections.Generic
        ;using System.Linq
        ;using System.Text
;using System.Threading.Tasks
        namespace DemoApplication
        {
            class Tutorial
            {
                ;public int TutorialID
                ;public string TutorialName

                (public void SetTutorial(int pID,string pName
                    }
                    ;TutorialID=pID
                    ;TutorialName=pName
                    {
                (public void SetTutorial(string pName
                    }
                    ;TutorialName=pName
                    {
                ()public String GetTutorial
                    }
                    ;return TutorialName
                    {

                (static void Main(string[] args
                    }

                ;()Tutorial pTutor=new Tutorial

                ;("pTutor.SetTutorial(1,"First Tutorial
                ;()Console.WriteLine(pTutor.GetTutorial

                ;("pTutor.SetTutorial("Second Tutorial
                ;()Console.WriteLine(pTutor.GetTutorial

                ;()Console.ReadKey
                {
                {
                {

```

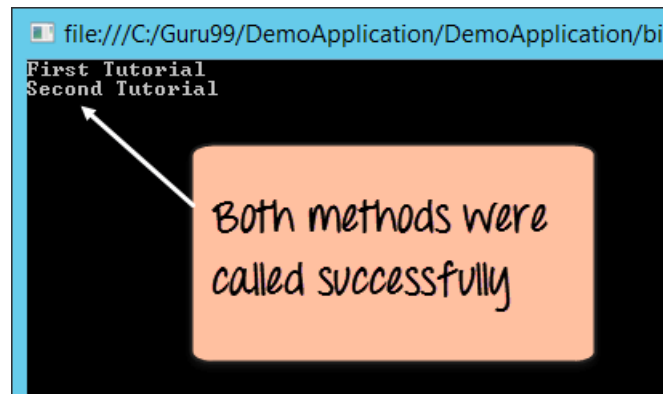
-:Code Explanation

1. In the first step, we are using the SetTutorial method with 2 parameters. Where we .are passing both the TutorialID and TutorialName to this method
2. In the second step, we are now calling the SetTutorial method with just one .parameter. We are just passing the TutorialName to this method

If the above code is entered properly and the program is run the following output will be displayed. If in case you wanted to also fetch the Tutorial ID along with the Tutorial Name , you should follow the below step

1. Create a separate method called `public int GetTutorialID`
2. In that method write the code line `"return TutorialID."` This can be used to return the `TutorialID` to the calling program

:Output



```
file:///C:/Guru99/DemoApplication/DemoApplication/bin
First Tutorial
Second Tutorial
```

Both methods were called successfully

From the output, we can clearly see that both methods were called successfully. Because of this, the strings "First Tutorial" and "Second Tutorial" were sent to the console

Summary

- Inheritance is where a child class inherits the fields and methods of the parent class.
- The child class can then also define its own methods
- Polymorphism in `C#` is a OOPs concept where one name can have many forms



[Report a Bug](#)

Prev ←

→ Next

Stay Updated on AI

.Get Weekly AI Skills, Trends, Actionable Advice

Sign up for the newsletter

Your Email Address

Subscribe for Free

Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

Copyright - Guru99 2025 Privacy Policy | Affiliate Disclaimer | ToS | Editorial Policy ©

C# Abstract Class Tutorial with Example: What is Abstraction?

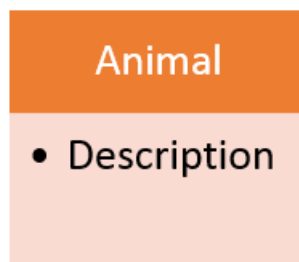
By : Benjamin Walker

🕒 November 23, 2024

What is Abstract Class in C#?

Abstract Class can never be instantiated and is marked by the keyword `abstract`. An abstract class contains zero or more abstract methods in it. Abstract class acts as a base class and is designed to be inherited by subclasses that either implement or either override its method.

Let's learn abstract class in C# with example given below. Below is the definition of a class called 'Animal.' When the 'Animal' class is defined, there is nothing known about the animal, whether it is a dog or a cat. The method called description is just a generic method defined for the class.



Now when it is known what exactly the `Animal` is going to be, we create another class which inherits the base class. If we know that the animal is in fact a `Dog`, we create `Dog` class which inherits the main base class. The key difference here is that the `Dog` class cannot change the definition of the `Description` method of the `Animal` class. It has to define its own C# abstract method called `Dog-Description`. This is the basic concept of C# abstract classes.

Dog : Animal

- Dog-Description

Create an Abstract Class in C#

Let's see abstract class in C# with real time examples on how we can change our code to include a C# abstract class. Note that we will not be running the code, because there is nothing that can be run using an C# abstraction class.

Step 1) As a first step, let's create an abstract class. The class will be called Tutorial and will just have one method. All the code needs to be written in the Program.cs file.

```
using System.Threading.Tasks;

namespace DemoApplication
{
    abstract class Tutorial 1
    {
        public virtual void Set() 2
        {
        }
    }
}
```

Define an abstract class Tutorial

Define an abstract class Tutorial

Code Explanation:-

1. We first define the abstract class. Note the use of the abstract keyword. This is used to denote that the class is an abstract class.
2. Next, we are defining our method which does nothing. The method must have the keyword called virtual. This means that the method cannot be changed by the child class. This is a basic requirement for any abstract class.

Step 2) Now let's add our child class. This code is added to the Program.cs file.

```
public class Guru99Tutorial : Tutorial
{
    protected int TutorialID;
    protected string TutorialName;

    public void SetTutorial(int pID, String pName)
    {
        TutorialID = pID;
        TutorialName = pName;
    }

    public String GetTutorial()
    {
        return TutorialName;
    }
}
```

Define a class that
inherits the base class

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    abstract class Tutorial
    {
        public virtual void Set()
        {

        }
    }
    class Guru99Tutorial:Tutorial
    {
        protected int TutorialID;
        protected string TutorialName;

        public void SetTutorial(int pID,string pName)
        {
            TutorialID=pID;
            TutorialName=pName;
        }

        public String GetTutorial()
        {
            return TutorialName;
        }

        static void Main(string[] args)
        {
            Guru99Tutorial pTutor=new Guru99Tutorial();

            pTutor.SetTutorial(1,".Net");

            Console.WriteLine(pTutor.GetTutorial());

            Console.ReadKey();
        }
    }
}

```

There is nothing exceptional about this code. We just define a class called ‘Guru99Tutorial’ which inherits the abstract Tutorial class. We then define the same methods as we have been using from before.

Note:

Here we cannot change the definition of the Set method which was defined in the Tutorial class. In the Tutorial class, we had defined a method called 'Set' (public virtual void Set()). Since the method was part of the abstract class C#, we are not allowed to define the Set method again in the Guru99Tutorial class.

Summary

An abstract class in **C sharp** is a blueprint for other classes, marked with the `abstract` keyword. It cannot be instantiated directly and may include abstract methods that derived classes must implement. Abstract classes enforce a consistent structure across subclasses while allowing specific functionality. This concept is commonly used to define base behaviors and ensure code flexibility through inheritance.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Subscribe for Free



Chosen by over **350,000+** professionals





Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

C# ArrayList Tutorial with Examples

By : Benjamin Walker

🕒 August 10, 2024

What is ArrayList in C#?

The ArrayList collection is similar to the Arrays data type in C#. The biggest difference is the dynamic nature of the array list collection.

For arrays, you need to define the number of elements that the array can hold at the time of array declaration. But in the case of the Array List collection, this does not need to be done beforehand. Elements can be added or removed from the Array List collection at any point in time. Let's look at the operations available for the array list collection in more detail.

Declaration of an Array List

The declaration of an ArrayList is provided below. An array list is created with the help of the ArrayList Datatype. The "new" keyword is used to create an object of an ArrayList. The object is then assigned to the variable a1. So now the variable a1 will be used to access the different elements of the array list.

```
ArrayList a1 = new ArrayList()
```

Adding elements to an array

The add method is used to add an element to the ArrayList. The add method can be used to add any sort of data type element to the array list. So you can add an Integer, or a string, or even a Boolean value to the array list. The general syntax of the addition method is given below

```
ArrayList.add(element)
```

Below are some examples of how the "add" method can be used. The add method can be used to add various data types to the Array List collection.

Below you can see examples of how we can add Integer's Strings and even Boolean values

to the Array List collection.

- `a1.add(1)` – This will add an Integer value to the collection
- `a1.add("Example")` – This will add a String value to the collection
- `a1.add(true)` – This will add a Boolean value to the collection

Now let's see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our `Program.cs` file.

In the program below, we will write the code to create a new array list. We will also show to add elements and to display the elements of the Array list.

```
static void Main(string[] args)
{
    ArrayList a1 = new ArrayList();
    a1.Add(1);
    a1.Add("Example");
    a1.Add(true);

    Console.WriteLine(a1[0]);
    Console.WriteLine(a1[1]);
    Console.WriteLine(a1[2]);

    Console.ReadKey();
}
```

1 Defining an array list

2 Adding elements to the array list

3 Displaying the elements of the array list

```

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

namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            ArrayList a1 = new ArrayList();
            a1.Add(1);
            a1.Add("Example");
            a1.Add(true);

            Console.WriteLine(a1[0]);
            Console.WriteLine(a1[1]);
            Console.WriteLine(a1[2]);
            Console.ReadKey();
        }
    }
}

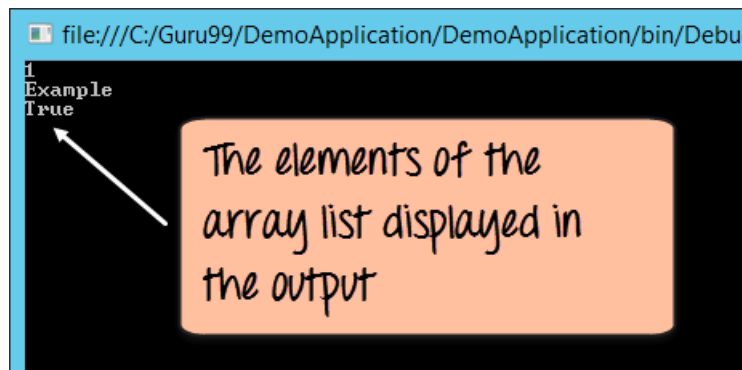
```

Code Explanation:-

1. The first step is used to declare our Array List. Here we are declaring a1 as a variable to hold the elements of our array list.
2. We then use the add keyword to add the number 1 , the String "Example" and the Boolean value 'true' to the array list.
3. We then use the Console.WriteLine method to display the value of each array lists element to the console. You will notice that just like arrays, we can access the elements via their index positions. So to access the first position of the Array List, we use the [0] index position. And so on and so forth.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/DemoApplication/bin/Debu
1
Example
True
```

The elements of the array list displayed in the output

From the output, you can see that all of the elements from the array list are sent to the console.

Let's look at some more methods which are available as part of the ArrayList.

Count

This method is used to get the number of items in the ArrayList collection. Below is the general syntax of this statement.

`ArrayList.Count()` – This method will return the number of elements that the array list contains.

Contains

This method is used to see if an element is present in the ArrayList collection. Below is the general syntax of this statement

`ArrayList.Contains(element)` – This method will return true if the element is present in the list, else it will return false.

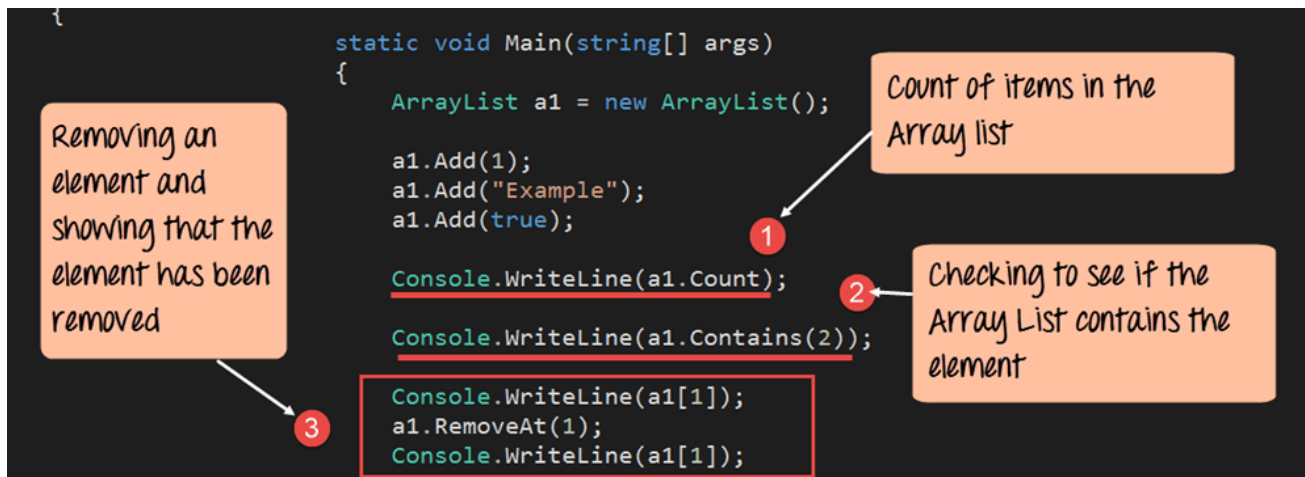
RemoveAt

This method is used to remove an element at a specific position in the ArrayList collection. Below is the general syntax of this statement

`ArrayList.RemoveAt(index)` – This method will remove an element from a specific position of the Array List.

Now let's see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our Program.cs file.

In the below program, we will write the code to see how we can use the above-mentioned methods.



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

```
namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            ArrayList a1 = new ArrayList();
            a1.Add(1);
            a1.Add("Example");
            a1.Add(true);

            Console.WriteLine(a1.Count);
            Console.WriteLine(a1.Contains(2));
            Console.WriteLine(a1[1]);
            a1.RemoveAt(1);
            Console.WriteLine(a1[1]);
            Console.ReadKey();
        }
    }
}
```

Code Explanation:-

1. So the first property we are seeing is the Count property. We are getting the Count property of the array list a1 and then writing it to the Console.
2. In the second part, we are using the Contains method to see if the ArrayList a1 contains the element 2. We then write the result to the Console via the Writeline

command.

3. Finally, to showcase the Remove element method, we are performing the below steps,
 1. First, we write the value of the element at Index position 1 of the array list to the console.
 2. Then we remove the element at Index position 1 of the array list.
 3. Finally, we again write the value of the element at Index position 1 of the array list to the console. This set of steps will give a fair idea whether the remove method will work as it should be.

If the above code is entered properly and the program is run the following output will be displayed.

Output:

Why is the last value true?

If you see the sequence of events, the element Example is removed from the array because this is at position 1. Position 1 of the array then gets replaced by what was in position 2 earlier which the value 'true'

Summary

The Array List collection is used to store a group of elements. The advantage of the Array list collection is that it is dynamic. You can add and remove elements on the fly to the array list collection.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Your Email Address

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

C# Stack with Push & Pop Examples

By : Benjamin Walker

🕒 August 10, 2024

What is Stack in C#?

The stack is a special case collection which represents a last in first out (LIFO) concept. To first understand LIFO, let's take an example. Imagine a stack of books with each book kept on top of each other.

The concept of last in first out in the case of books means that only the top most book can be removed from the stack of books. It is not possible to remove a book from between, because then that would disturb the setting of the stack.

Hence in **C#**, the stack also works in the same way. Elements are added to the stack, one on the top of each other. The process of adding an element to the stack is called a push operation. To remove an element from a stack, you can also remove the top most element of the stack. This operation is known as pop.

Table of Content:



Let's look at the operations available for the Stack collection in more detail.

Declaration of the stack

A stack is created with the help of the Stack Data type. The keyword "new" is used to create an object of a Stack. The object is then assigned to the variable st.

```
Stack st = new Stack()
```

Adding elements to the stack

The push method is used to add an element onto the stack. The general syntax of the statement is given below.

```
Stack.push(element)
```

Removing elements from the stack

The pop method is used to remove an element from the stack. The pop operation will return the topmost element of the stack. The general syntax of the statement is given below

C# Hashtable with Examples

By : Benjamin Walker

🕒 August 10, 2024

What is Hashtable in C#?

A hash table is a special collection that is used to store key-value items. So instead of storing just one value like the stack, array list and queue, the hash table stores 2 values. These 2 values form an element of the hash table.

Below are some example of how values of a hash table might look like.

```
{ "001" , ".Net" }  
{ "002" , "C#" }  
{ "003" , "ASP.Net" }
```

Above we have 3 key value pairs. The keys of each element are 001, 002 and 003 respectively. The values of each key value pair are “.Net”, “C#” and “ASP.Net” respectively.

Table of Content:



Let's look at the operations available for the Hashtable collection in more detail.

Declaration of the Hashtable

The declaration of a Hashtable is shown below. A Hashtable is created with the help of the Hashtable Datatype. The “new” keyword is used to create an object of a Hashtable. The object is then assigned to the variable ht.

```
Hashtable ht = new Hashtable()
```

Adding elements to the Hashtable

The Add method is used to add an element on to the [queue](#). The general syntax of the statement is given below

```
HashTable.add("key","value")
```

Example 1:

Remember that each element of the hash table comprises of 2 values, one is the key, and the other is the value.

Now, let's see this working at a code level. All of the below-mentioned code will be written to our Console application.

The code will be written to our Program.cs file. In the below program, we will write the code to see how we can use the above-mentioned methods.

For now in our example, we will just look at how we can create a hashtable , add elements to the hashtable and display them accordingly.

```
{
    static void Main(string[] args)
    {
        Hashtable ht = new Hashtable();
        ht.Add("001", ".Net");
        ht.Add("002", "C#");
        ht.Add("003", "ASP.Net");
        ICollection keys = ht.Keys;
        foreach (String k in keys){
            Console.WriteLine(ht[k]);
        }
        Console.ReadKey();
    }
}
```

The code is annotated with four numbered steps in orange boxes:

- 1** Creating a hashtable variable: Points to the line `Hashtable ht = new Hashtable();`
- 2** Adding elements to the hashtable: Points to the three `ht.Add()` lines.
- 3** Getting the keys collection: Points to the line `ICollection keys = ht.Keys;`
- 4** Displaying the value for each key: Points to the `Console.WriteLine(ht[k]);` line inside the `foreach` loop.


```

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

namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Hashtable ht = new Hashtable();
            ht.Add("001", ".Net");
            ht.Add("002", "C#");
            ht.Add("003", "ASP.Net");

            ICollection keys = ht.Keys;

            foreach (String k in keys)
            {
                Console.WriteLine(ht[k]);
            }
            Console.ReadKey();
        }
    }
}

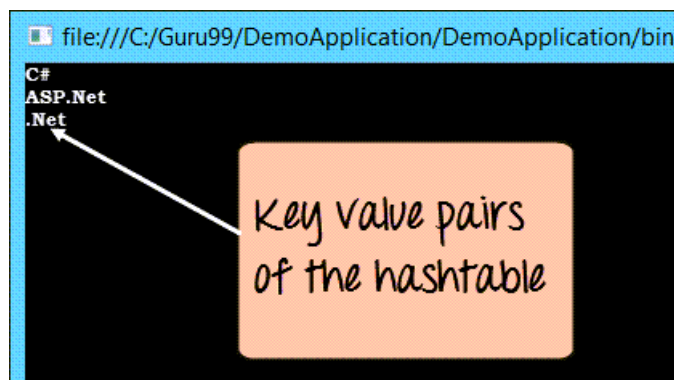
```

Code Explanation:-

1. First, we declare the hashtable variable using the Hashtable data type by using keyword "New." The name of the variable defines is 'ht'.
2. We then add elements to the hash table using the Add method. Remember that we need to add both a key and value element when adding something to the hashtable.
3. There is no direct way to display the elements of a hash table.
 - In order to display the hashtable , we first need to get the list of keys (001, 002 and 003) from the hash table.
 - This is done via the ICollection interface. This is a special data type which can be used to store the keys of a hashtable collections. We then assign the keys of the hashtable collection to the variable 'keys'.
4. Next for each key value, we get the associated value in the hashtable by using the statement ht[k].

If the above code is entered properly and the program is run the following output will be displayed.

Output:



Let's look at some more methods available for hash tables.

ContainsKey

This method is used to see if a key is present in the Hashtable. Below is the general syntax of this statement. The statement will return true if the key exists, else it will return the value false.

```
Hashtable.ContainsKey(key)
```

ContainsValue

This method is used to see if a Value is present in the Hashtable. Below is the general syntax of this statement. The statement will return true if the Value exists, else it will return the value false.

```
Hashtable.ContainsValue(value)
```

Example 2:

Let's change the code in our [Console application](#) to showcase how we can use the "Containskey" and "ContainsValue" method.

```

static void Main(string[] args)
{
    Hashtable ht = new Hashtable();

    ht.Add("001", ".Net");
    ht.Add("002", "C#");
    ht.Add("003", "ASP.Net");

    Console.WriteLine(ht.ContainsKey("001"));
    Console.WriteLine(ht.ContainsValue("C#"));
    Console.ReadKey();
}

```

Searching for a key

1

Searching for a value

2

```

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

namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Hashtable ht = new Hashtable();
            ht.Add("001", ".Net");
            ht.Add("002", "C#");
            ht.Add("003", "ASP.Net");

            Console.WriteLine(ht.ContainsKey("001"));
            Console.WriteLine(ht.ContainsValue("C#"));
            Console.ReadKey();
        }
    }
}

```

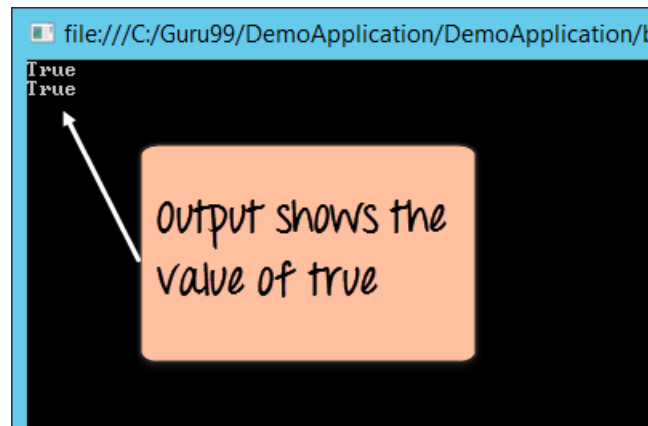
Code Explanation:-

1. First, we use the ContainsKey method to see if the key is present in the hashtable. This method will return true if the key is present in the hashtable. This method should return true since the key does exist in the hashtable.
2. We then use the ContainsValue method to see if the value is present in the

hashtable. This method will return 'true' since the Value does exist in the hashtable.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



From the output, you can clearly see that both the key and value being searched are present in the hash table.

Summary

A Hashtable is used to store elements which comprises of key values pairs. To access the value of an element , you need to know the key of the element.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

Stream in C# Tutorial: StreamReader & StreamWriter [Example]

By : Benjamin Walker

🕒 August 10, 2024

What is C# Stream?

In C# file operations, normally streams are used to read and write to files. A stream is an additional layer created between an application and a file. The stream is used to ensure smooth read and write operations to the file.

Streams are normally used when reading data from large files. By using streams, the data from large files is broken down into small chunks and sent to the stream. These chunks of data can then be read from the application.

The reason for breaking it down into small chunks is because of the performance impact of reading a big file in one shot. If you were to read the data from say, a 100 MB file at one shot, your application could just hang and become unstable. The best approach is then to use streams to break the file down into manageable chunks.

So when a write operation is carried out on file, the data to be written, is first written to the stream. From the stream, the data is then written to the file. The same goes for the read operation. In the read operation, data is first transferred from the file to the stream. The data is then read from the application via the stream. Let's look at an example of how we can read and write using streams.

Table of Content:



Stream Reader

The stream reader is used to read data from a file using streams. The data from the file is first read into the stream. Thereafter the application reads the data from the stream.

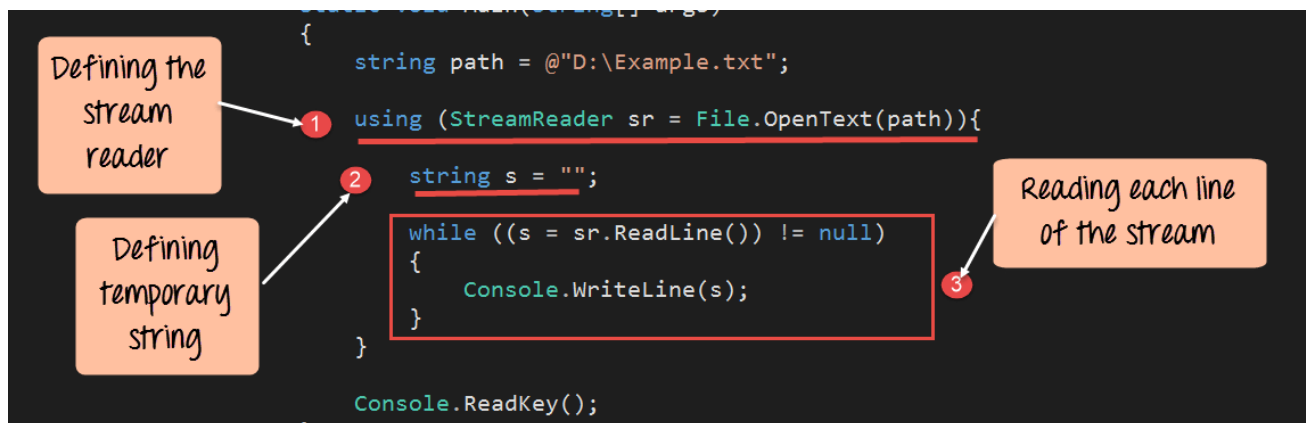
For our example, we will assume that we have a file in the D drive called Example.txt. The file will be a simple text file and have 2 lines as shown below

- Guru99 – .Net

- Guru99 -C#

For our example, we will create a simple Console application and work with File streams

Let's look at an example of how we can use streams for reading data from a file. Enter the below code in the program.cs file.



```

using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Tutorial
    {
        static void Main(string[] args)
        {
            String path = @"D:\Example.txt";

            using (StreamReader sr = File.OpenText(path))
            {
                String s = "";

                while ((s = sr.ReadLine()) != null)
                {
                    Console.WriteLine(s);
                }
            }
            Console.ReadKey();
        }
    }
}

```

Code Explanation:-

1. First, we are declaring a stream reader object. The stream reader object is used in C# to define a stream from the file to the application. The data will be pushed from the file to the stream whenever data is read from the file. The File.OpenText is used to open the file "Example.txt" in read-only mode. The handler to the file is then sent to the stream reader object.
2. Next, we are defining a temporary variable 's' which will be used to read all the data from the file.
3. We then use the stream reader method ReadLine to read each line from the stream buffer. When we perform this [operation](#), each line will be first transferred from the file to the buffer. Then the string line will be transferred from the buffer to the variable 's'. We then write the contents of the string 's' to the console.

When the above code is set, and the project is run using [Visual Studio](#), you will get the below output.

Output:-


```
file:///C:/Guru99/DemoApplicationnew/DemoApplicationnew/bin/De
Guru99 - .Net
Guru99 - C#
```

Output shows the 2 lines of the file

From the output, you can see that the Stream Reader read both the lines from the file. Finally, the lines of the string read from the stream were sent to the Console.

Stream Writer

The stream writer is used to write data to a file using streams. The data from the application is first written into the stream. After that the stream writes the data to the file. Let's look at an example of how we can use streams for writing data from a file. Enter the below code in the program.cs file.

```
static void Main(string[] args)
{
    string path = @"D:\Example.txt";
    using (StreamWriter sr = File.AppendText(path))
    {
        sr.WriteLine("Guru99 - ASP.Net");
        sr.Close();
        Console.WriteLine(File.ReadAllText(path));
    }
    Console.ReadKey();
}
```

open StreamWriter object

1

Write to the stream

2

Close the stream

3

Read the contents of the file

4

```

using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    class Tutorial
    {
        static void Main(string[] args)
        {
            String path = @"D:\Example.txt";

            using (StreamWriter sr = File.AppendText(path))
            {
                sr.WriteLine("Guru99 - ASP.Net");
                sr.Close();

                Console.WriteLine(File.ReadAllText(path));
            }
            Console.ReadKey();
        }
    }
}

```

Code Explanation:-

1. First, we are declaring a stream writer object. The stream writer object is used in C# to define a stream. The stream is then used to write data from the application to the file. The data will be pushed from the application to the stream whenever data needs to be written. The File.AppendText command is used to open the file "Example.txt" in an append mode. The handler to the file is then sent to the stream writer object.
2. We are using the stream write method Writeline to write the line "Guru99 – ASP.Net" to the stream. From the stream, the line will then be written to the file.
3. We then close the stream writer after writing to the file. It's normally a good practice to close file handlers when the file is no longer required for writing purposes.
4. Finally, we are reading the contents of the file again and writing it to the console log. This is to check as to whether the line was written to the file.

When the above code is set, and the project is run using Visual Studio, you will get the below output.

Output:-

```
file:///C:/Guru99/DemoApplicationnew/DemoApplicationnew/
Guru99 - .Net
Guru99 - C#
Guru99 - ASP.Net
```

Can see that the new line was added

From the output, you can see that the line “Guru99 – ASP.Net” was added to the file successfully. All the 3 lines of text can be seen in the console.

Summary

- Streams are used as an intermediate level between the application and the file.
- A StreamReader is used whenever data is required to be read from a file.
- A StreamWriter is used whenever data needs to be written to a file.



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)

Top 50 C# Interview Questions and Answers (2025)

By : Benjamin Walker

🕒 August 10, 2024

C# Interview Questions and Answers for Freshers and 2/3/5/10 Years Experience

Here are C# interview questions and answers for fresher as well 5 or 10 years experienced candidates to get their dream job.

1) What is C#?

C# is an object-oriented, type-safe, and managed language that is compiled by .Net framework to generate Microsoft Intermediate Language.

[📄 Free PDF Download: C# Interview Interview Questions & Answers](#)

2) Explain types of comment in C# with examples

Single line

Example:

```
//This is a single line comment
```

ii. Multiple line (`/* */`)

Example:

```
/*This is a multiple line comment  
We are in line 2  
Last line of comment*/
```

iii. XML Comments (`///`).

Example:

```
/// summary;  
/// Set error message for multilingual language.  
/// summary
```

3) Can multiple catch blocks be executed?

No, Multiple catch blocks of similar type can't be executed. Once the proper catch code executed, the control is transferred to the finally block, and then the code that follows the finally block gets executed.

4) What is the difference between public, static, and void?

Public declared variables or methods are accessible anywhere in the application. Static declared variables or methods are globally accessible without creating an instance of the class. Static member are by default not globally accessible it depends upon the type of access modifier used. The compiler stores the address of the method as the entry point and uses this information to begin execution before any objects are created. And Void is a type modifier that states that the method or variable does not return any value.



5) What is an object?

An object is an instance of a class through which we access the methods of that class. "New" keyword is used to create an object. A class that creates an object in memory will contain the information about the methods, variables, and behavior of that class.

6) Define Constructors

A constructor is a member function in a class that has the same name as its class. The constructor is automatically invoked whenever an object class is created. It constructs the values of data members while initializing the class.

RELATED ARTICLES

- [C# Windows Forms Application Tutorial with Example](#)
 - [C# Enum\(Enumeration\) with Example](#)
 - [C# Queue with Examples: What is C# Queue and How to Use?](#)
 - [BEST C# IDE for Windows, Linux & Mac \(2025 Update\)](#)
-

7) What is Jagged Arrays?

The Array which has elements of type array is called jagged Array. The elements can be of different dimensions and sizes. We can also call jagged Array as an Array of arrays.

8) What is the difference between ref & out parameters?

An argument passed as ref must be initialized before passing to the method whereas out parameter needs not to be initialized before passing to a method.

9) What is the use of 'using' statement in C#?

The 'using' block is used to obtain a resource and process it and then automatically dispose of when the execution of the block completed.

10) What is serialization?

When we want to transport an object through a network, then we have to convert the object into a stream of bytes. The process of converting an object into a stream of bytes is called Serialization. For an object to be serializable, it should implement ISerialize Interface. De-serialization is the reverse process of creating an object from a stream of bytes.

11) Can we use "this" command within a static method?

We can't use 'This' in a static method because we can only use static variables/methods in a static method.

12) What is the difference between constants and read-only?

Constant variables are declared and initialized at compile time. The value can't be changed afterward. Read-only is used only when we want to assign the value at run time.

13) What is an interface class? Give one example of it

An Interface is an abstract class which has only public abstract methods, and the methods only have the declaration and not the definition. These abstract methods must be implemented in the inherited classes.


```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace DemoApplication
{
    interface Guru99Interface
    {
        void SetTutorial(int pID, string pName);
        String GetTutorial();
    }

    class Guru99Tutorial : Guru99Interface
    {
        protected int TutorialID;
        protected string TutorialName;

        public void SetTutorial(int pID, string pName)
        {
            TutorialID = pID;
            TutorialName = pName;
        }

        public String GetTutorial()
        {
            return TutorialName;
        }

        static void Main(string[] args)
        {
            Guru99Tutorial pTutor = new Guru99Tutorial();

            pTutor.SetTutorial(1, ".Net by Guru99");

            Console.WriteLine(pTutor.GetTutorial());

            Console.ReadKey();
        }
    }
}

```

14) What are value types and reference types?

A value type holds a data value within its own memory space. Example

```
int a = 30;
```

Reference type stores the address of the Object where the value is being stored. It is a pointer to another memory location.

```
string b = "Hello Guru99!!";
```

15) What are Custom Control and User Control?

Custom Controls are controls generated as compiled code (DLLs), those are easier to use and can be added to toolbox. Developers can drag and drop controls to their web forms. Attributes can, at design time. We can easily add custom controls to Multiple Applications (If Shared DLLs). So, If they are private, then we can copy to dll to bin directory of web application and then add reference and can use them.

User Controls are very much similar to ASP include files, and are easy to create. User controls can't be placed in the toolbox and dragged – dropped from it. They have their design and code-behind. The file extension for user controls is ascx.

16) What are sealed classes in C#?

We create sealed classes when we want to restrict the class to be inherited. Sealed modifier used to prevent derivation from a class. If we forcefully specify a sealed class as base class, then a compile-time error occurs.

17) What is method overloading?

Method overloading is creating multiple methods with the same name with unique signatures in the same class. When we compile, the compiler uses overload resolution to determine the specific method to be invoke.

18) What is the difference between Array and ArrayList?

In an array, we can have items of the same type only. The size of the array is fixed when compared. To an arraylist is similar to an array, but it doesn't have a fixed size.

19) Can a private virtual method can be overridden?

No, because they are not accessible outside the class.

20) Describe the accessibility modifier “protected internal”.

Protected Internal variables/methods are accessible within the same assembly and also from the classes that are derived from this parent class.

21) What are the differences between System.String and System.Text.StringBuilder classes?

System.String is immutable. When we modify the value of a string variable, then a new memory is allocated to the new value and the previous memory allocation released.

System.StringBuilder was designed to have a concept of a mutable string where a variety of operations can be performed without allocation separate memory location for the modified string.

22) What's the difference between the System.Array.CopyTo() and System.Array.Clone() ?

Using `Clone()` method, we creates a new array object containing all the elements in the original Array and using `CopyTo()` method. All the elements of existing array copies into another existing array. Both methods perform a shallow copy.

23) How can we sort the elements of the Array in descending order?

Using `Sort()` methods followed by `Reverse()` method.

24) Write down the C# syntax to catch an exception

To catch an exception, we use try-catch blocks. Catch block can have a parameter of system.Exception type.

Eg:

```
try {  
    GetAllData();  
}  
catch (Exception ex) {  
}
```

In the above example, we can omit the parameter from catch statement.

25) What's the difference between an interface and abstract class?

Interfaces have all the methods having only declaration but no definition. In an abstract class, we can have some concrete methods. In an interface class, all the methods are public. An abstract class may have private methods.

26) What is the difference between Finalize() and Dispose() methods?

`Dispose()` is called when we want for an object to release any unmanaged resources with them. On the other hand, `Finalize()` is used for the same purpose, but it doesn't assure the garbage collection of an object.

27) What are circular references?

Circular reference is situation in which two or more resources are interdependent on each other causes the lock condition and make the resources unusable.

28) What are generics in C#.NET?

Generics are used to make reusable code classes to decrease the code redundancy, increase type safety, and performance. Using generics, we can create collection classes. To create generic collection, System.Collections.Generic namespace should be used instead of classes such as ArrayList in the System.Collections namespace. Generics promotes the usage of parameterized types.

29) What is an object pool in .NET?

An object pool is a container having objects ready to be used. It tracks the object that is

currently in use, total number of objects in the pool. This reduces the overhead of creating and re-creating objects.

30) List down the commonly used types of exceptions in .net

ArgumentException, ArgumentNullException, ArgumentOutOfRangeException, ArithmeticException, DivideByZeroException, OverflowException, IndexOutOfRangeException, InvalidCastException, InvalidOperationException, IOEndOfStreamException, NullReferenceException, OutOfMemoryException, StackOverflowException etc.

31) What are Custom Exceptions?

Sometimes there are some errors that need to be handled as per user requirements. Custom exceptions are used for them and are used defined exceptions.

32) What are delegates?

Delegates are same as function pointers in C++, but the only difference is that they are type safe, unlike function pointers. Delegates are required because they can be used to write much more generic type-safe functions.

33) How do you inherit a class into other class in C#?

Colon is used as inheritance operator in C#. Just place a colon and then the class name.

```
public class DerivedClass : BaseClass
```

34) What is the base class in .net from which all the classes are derived from?

```
System.Object
```

35) What is the difference between method overriding and method overloading?

In method overriding, we change the method definition in the derived class that changes the method behavior. Method overloading is creating a method with the same name within the same class having different signatures.

36) What are the different ways a method can be overloaded?

Methods can be overloaded using different data types for a parameter, different order of parameters, and different number of parameters.

37) Why can't you specify the accessibility modifier for methods inside the interface?

In an interface, we have virtual methods that do not have method definition. All the methods are there to be overridden in the derived class. That's why they all are public.

38) How can we set the class to be inherited, but prevent the method from being over-ridden?

Declare the class as public and make the method sealed to prevent it from being overridden.

39) What happens if the inherited interfaces have conflicting method names?

Implement is up to you as the method is inside your own class. There might be a problem when the methods from different interfaces expect different data, but as far as compiler cares you're okay.

40) What is the difference between a Struct and a Class?

Structs are value-type variables, and classes are reference types. Structs stored on the Stack causes additional overhead but faster retrieval. Structs cannot be inherited.

41) How to use nullable types in .Net?

Value types can take either their normal values or a null value. Such types are called nullable types.

```
int? someID = null;
if(someID.HasValue)
{
}
```

42) How we can create an array with non-default values?

We can create an array with non-default values using `Enumerable.Repeat`.

43) What is difference between "is" and "as" operators in c#?

“is” operator is used to check the compatibility of an object with a given type, and it returns the result as Boolean.

“as” operator is used for casting of an object to a type or a class.

44) What's a multicast delegate?

A delegate having multiple handlers assigned to it is called multicast delegate. Each handler is assigned to a method.

45) What are indexers in C# .NET?

Indexers are known as smart [arrays in C#](#). It allows the instances of a class to be indexed in the same way as an array.

Eg:

```
public int this[int index] // Indexer declaration
```

46) What is difference between the “throw” and “throw ex” in .NET?

“Throw” statement preserves original error stack whereas “throw ex” have the stack trace from their throw point. It is always advised to use “throw” because it provides more accurate error information.

47) What are C# attributes and its significance?

C# provides developers a way to define declarative tags on certain entities, eg. Class, method, etc. are called attributes. The attribute's information can be retrieved at runtime using Reflection.

48) How to implement a singleton design pattern in C#?

In a singleton pattern, a class can only have one instance and provides an access point to it globally.

Eg:

```
Public sealed class Singleton
{
    Private static readonly Singleton _instance = new Singleton();
}
```

49) What is the difference between directcast and ctype?

DirectCast is used to convert the type of object that requires the run-time type to be the same as the specified type in DirectCast.

Ctype is used for conversion where the conversion is defined between the expression and the type.

50) Is C# code is managed or unmanaged code?

C# is managed code because Common language runtime can compile C# code to Intermediate language.

51) What is Console application?

A console application is an application that can be run in the command prompt in Windows. For any [beginner on .Net](#), building a console application is ideally the first step, to begin with.

52) Give an example of removing an element from the queue

The dequeue method is used to remove an element from the queue.


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

namespace DemoApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Queue qt = new Queue();
            qt.Enqueue(1);
            qt.Enqueue(2);
            qt.Enqueue(3);

            foreach (Object obj in qt)
            {
                Console.WriteLine(obj);
            }
            Console.WriteLine(); Console.WriteLine();
            Console.WriteLine("The number of elements in the Queue " + qt.Count);
            Console.WriteLine("Does the Queue contain " + qt.Contains(3));
            Console.ReadKey();
        }
    }
}
```

These interview questions will also help in your viva(orals)



[Report a Bug](#)

← Prev

Next →

Stay Updated on AI

Get Weekly AI Skills, Trends, Actionable Advice.

Sign up for the newsletter

Your Email Address

Subscribe for Free



Chosen by over **350,000+** professionals



Contact Us

[About Us](#)

[Contact US](#)

[Advertise with Us](#)

Recommended Tools

[NinjaOne](#)

[Textline](#)

[Activtrak](#)

[Deel](#)

[Xero](#)