Chapter 2:From C++ to C#

C#:

- It is an object-oriented programming language created by Microsoft that runs on the .NET Framework.
- C# has roots from the C family, and the language is close to other popular languages like C++ and Java.
- The first version was released in year 2002. The latest version, **C# 8**, was released in September 2019.

C# is used for:

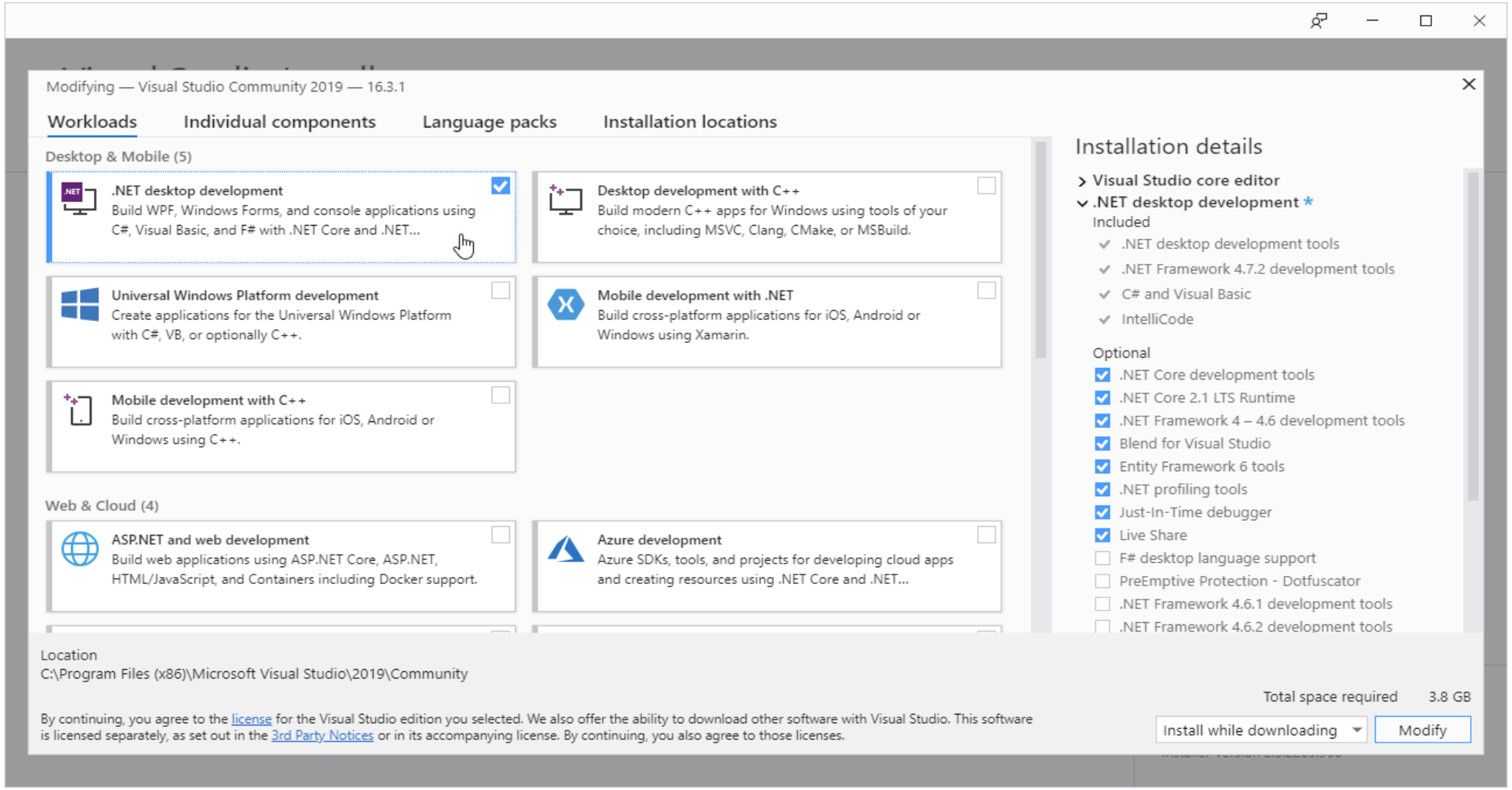
- Mobile applications
- Desktop applications
- Web applications
- Web services
- Web sites
- Games
- VR
- Database applications

Why Use C#?

- It is one of the most popular programming language in the world
- It is easy to learn and simple to use
- It has a huge community support
- C# is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- As C# is close to C, C++ and Java, it makes it easy for programmers to switch to C# or vice versa

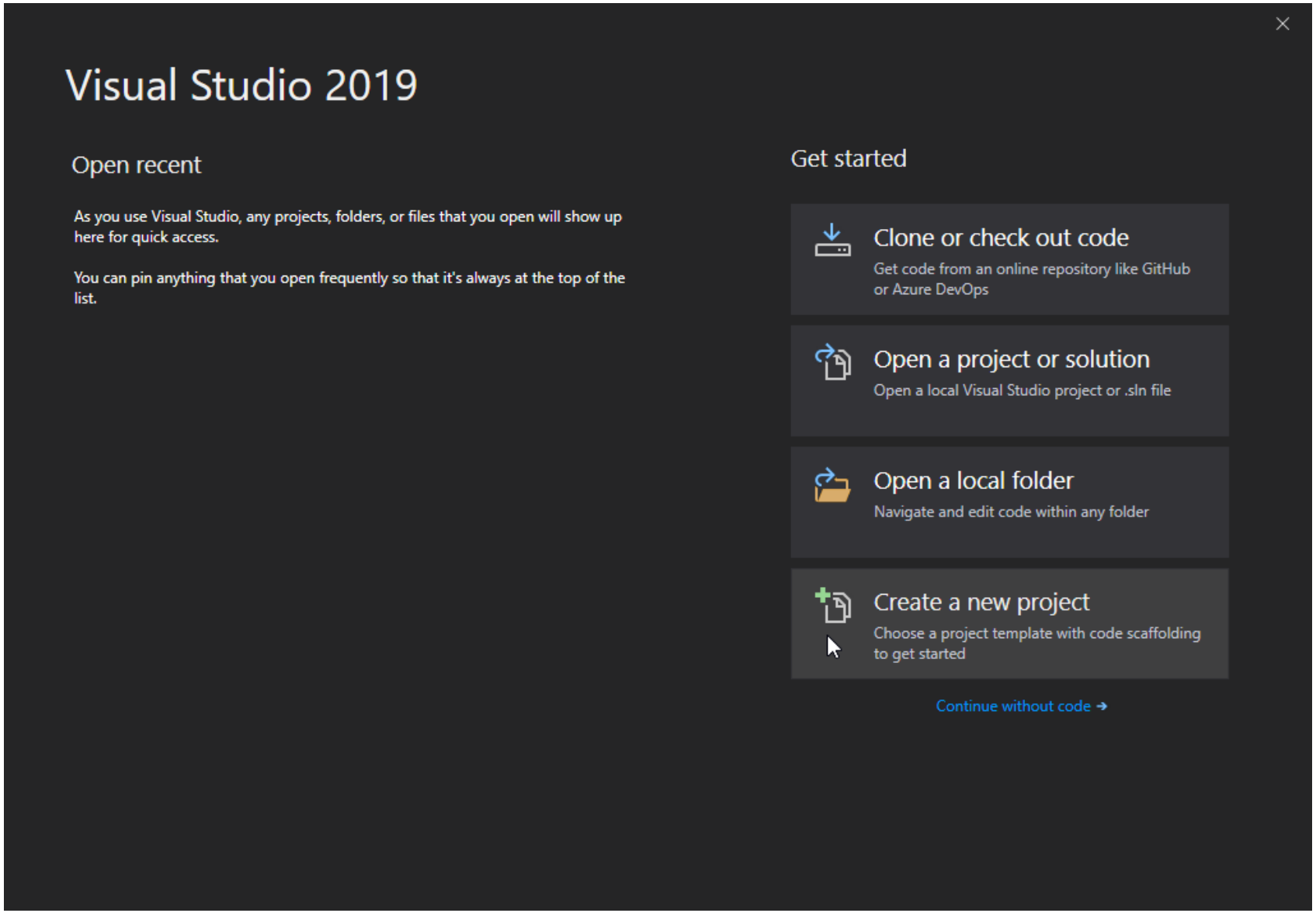
C# Install

Once the Visual Studio Installer is downloaded and installed, choose the .NET workload and click on the **Modify/Install** button:

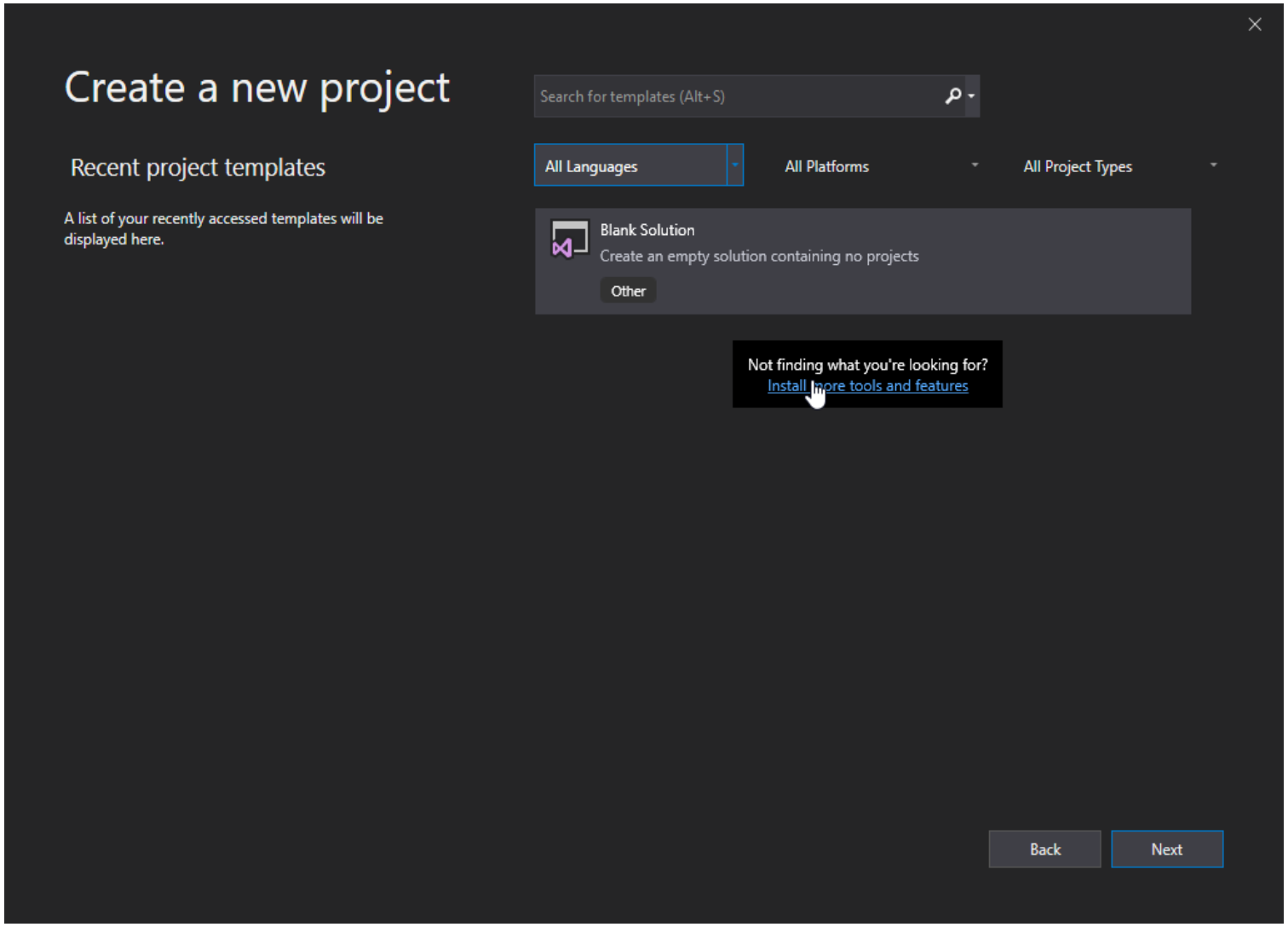


After the installation is complete, click on the **Launch** button to get started with Visual Studio.

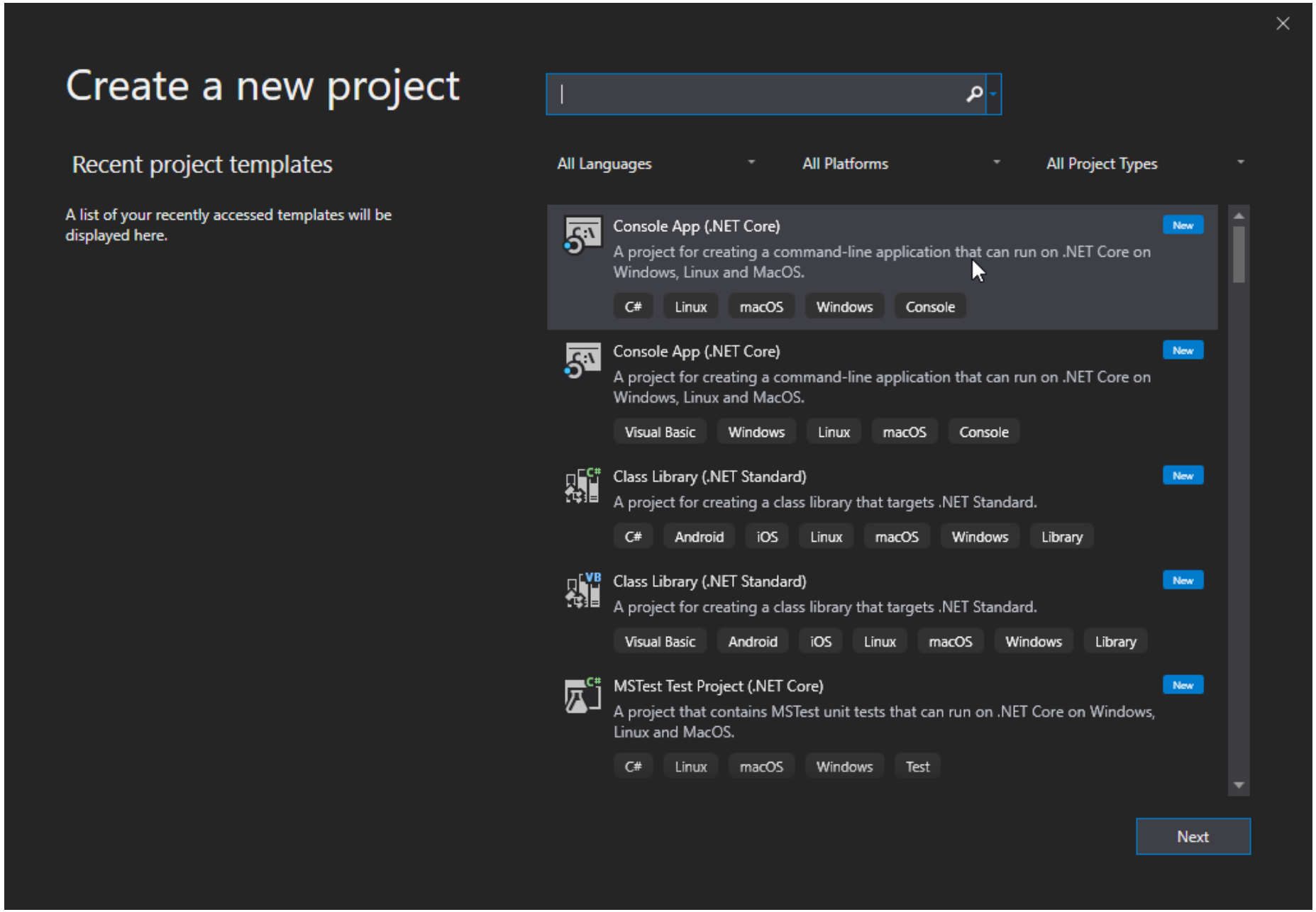
On the start window, choose **Create a new project**:



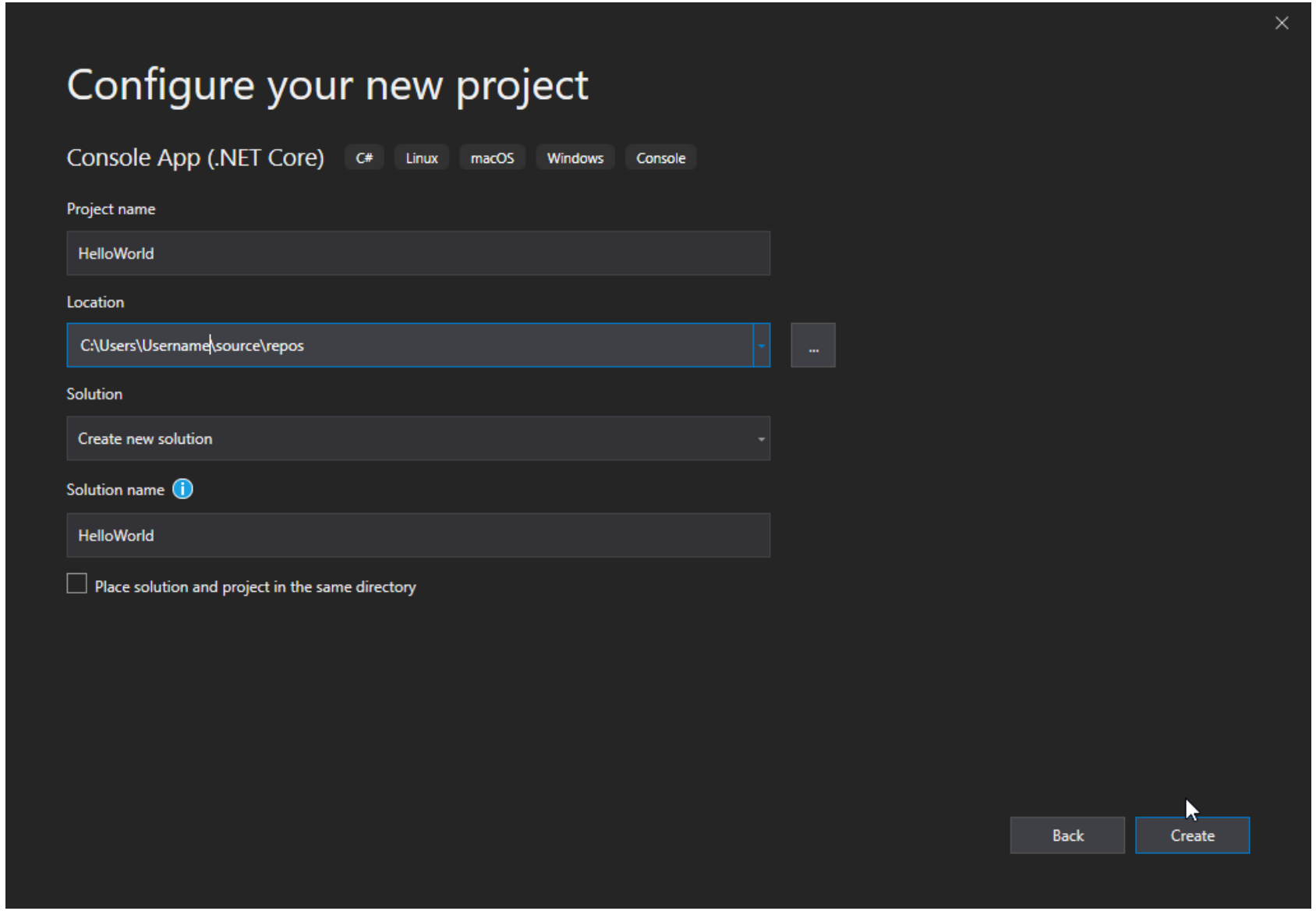
Then click on the "Install more tools and features" button:


Create a new project
Search for templates (Alt+S)
Recent project templates
All Languages
All Platforms
All Project Types
A list of your recently accessed templates will be displayed here.
Blank Solution
Create an empty solution containing no projects
Other
Not finding what you're looking for?
Install more tools and features
Back
Next

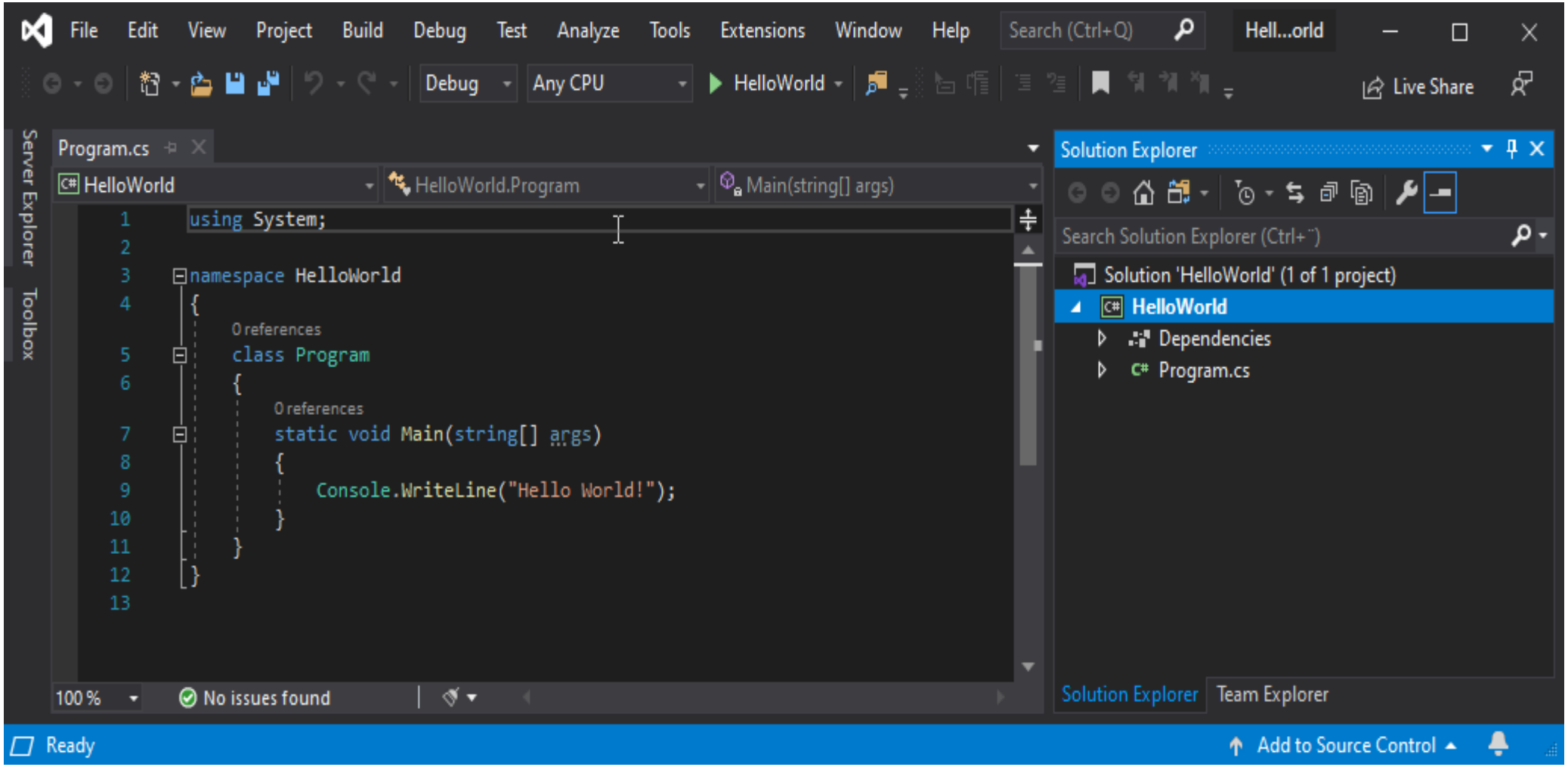
Choose "Console App (.NET Core)" from the list and click on the Next button:



Enter a name for your project, and click on the Create button:



Visual Studio will automatically generate some code for your project:



Program.cs

```
using System;

namespace HelloWorld
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

Run the program by pressing the **F5** button or click on "**Debug**" -> "**Start Debugging**"

C# Comments

Single-line Comments:

Single-line comments start with two forward slashes (//).

C# Multi-line Comments:

Multi-line comments start with /* and ends with */.

Declaring (Creating) Variables

```
type variableName = value;

string name = "John";
Console.WriteLine(name);
```

```
int myNum;
myNum = 15;
Console.WriteLine(myNum);
```

```
int myNum = 15;
myNum = 20; // myNum is now 20
Console.WriteLine(myNum);
```

C# User Input

Console.WriteLine() is used to output (print) values. Now we will use Console.ReadLine() to get user input.

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      // Type your username and press enter
      Console.WriteLine("Enter username:");

      // Create a string variable and get user input from the keyboard and
store it in the variable
      string userName = Console.ReadLine();

      // Print the value of the variable (userName), which will display the
input value
      Console.WriteLine("Username is: " + userName);
    }
  }
}
```

User Input and Numbers

The Console.ReadLine() method returns a string. Therefore, you cannot get information from another data type, such as int. The following program will cause an error:

```
Console.WriteLine("Enter your age:");
int age = Console.ReadLine();
Console.WriteLine("Your age is: " + age);
```

Error: Cannot implicitly convert type 'string' to 'int'

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

Program.cs

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Enter your age:");
      int age = Convert.ToInt32(Console.ReadLine());
      Console.WriteLine("Your age is: " + age);
    }
  }
}
```

```
Enter your age:
gfdh
System.FormatException: 'Input string was not in a correct format.'
```

Note: If you enter wrong input (e.g. text in a numerical input), you will get an exception/error message (like System.FormatException: 'Input string was not in a correct format.').

C# Type Casting

In C#, there are two types of casting:

1) **Implicit Casting** (automatically) - converting a smaller type to a larger type size
char -> int -> long -> float -> double

2) **Explicit Casting** (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char

Implicit Casting

```
int myInt = 9;
double myDouble = myInt;       // Automatic casting: int to
double

Console.WriteLine(myInt);      // Outputs 9
Console.WriteLine(myDouble);   // Outputs 9
```

Explicit Casting

```
double myDouble = 9.78;
int myInt = (int) myDouble;    // Manual casting: double to int

Console.WriteLine(myDouble);   // Outputs 9.78
Console.WriteLine(myInt);      // Outputs 9
```

Type Conversion Methods

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;

Console.WriteLine(Convert.ToString(myInt));    // convert int to
string
Console.WriteLine(Convert.ToDouble(myInt));    // convert int to
double
Console.WriteLine(Convert.ToInt32(myDouble));  // convert
double to int
Console.WriteLine(Convert.ToString(myBool));   // convert bool
to string
```

Types

Basic data types in C# are distributed into the following types:

- Integer types – **`sbyte`**, **`byte`**, **`short`**, **`ushort`**, **`int`**, **`uint`**, **`long`**, **`ulong`**;
- Real floating-point types – **`float`**, **`double`**;
- Real type with decimal precision – **`decimal`**;
- Boolean type – **`bool`**;
- Character type – **`char`**;
- String – **`string`**;
- Object type – **`object`**.

These data types are called **primitive (built-in types)**, because they are embedded in C# language at the lowest level. The table below represents the above mentioned data types, their range and their default values:

Data Types	Default Value	Minimum Value	Maximum Value
sbyte	0	-128	127
byte	0	0	255
short	0	-32768	32767
ushort	0	0	65535
int	0	-2147483648	2147483647
uint	0u	0	4294967295
long	0L	-9223372036854775808	9223372036854775807
ulong	0u	0	18446744073709551615
float	0.0f	$\pm 1.5 \times 10^{-45}$	$\pm 3.4 \times 10^{38}$
double	0.0d	$\pm 5.0 \times 10^{-324}$	$\pm 1.7 \times 10^{308}$
decimal	0.0m	$\pm 1.0 \times 10^{-28}$	$\pm 7.9 \times 10^{28}$
bool	false	Two possible values: **true** and **false**	
char	'\u0000'	'\u0000'	'\uffff'
object	null	-	-
string	null	-	-

Correspondence between C# and .NET Types

- Primitive data types in C# have a direct correspondence with the types of the common type system (CTS) in .NET Framework.
- For instance, **`int`** type in C# corresponds to **`System.Int32`** type in CTS and to **`Integer type in`** VB.NET language,
- while **`long`** type in C# corresponds to **`System.Int64 type in`** CTS and to **`Long type`** in VB.NET language.
- Due to the common types system (CTS) in .NET Framework there is compatibility between different prog-ramming languages (like for instance, C#, Managed C++, VB.NET and F#).
- For the same reason **`int`**, **`Int32`** and **`System.Int32`** types in C# are actually different aliases for one and the same data type – signed 32-bit integer.

C# - Decision Making

C# - if Statement

```
if(boolean_expression) {
  /* statement(s) will execute if the boolean expression is true */}
```

```
using System;

namespace DecisionMaking {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 10;

      /* check the boolean condition using if statement */
      if (a < 20) {
        /* if condition is true then print the following */
        Console.WriteLine("a is less than 20");
      }
      Console.WriteLine("value of a is : {0}", a);
      Console.ReadLine();
    }
  }
}
```

```
a is less than 20;
value of a is : 10
```

C# - if...else Statement

```
if(boolean_expression) {
  /* statement(s) will execute if the boolean expression is true */}
else
{ /* statement(s) will execute if the boolean expression is false */}
```

```
using System;

namespace DecisionMaking {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 100;

      /* check the boolean condition */
      if (a < 20) {
        /* if condition is true then print the following */
        Console.WriteLine("a is less than 20");
      } else {
        /* if condition is false then print the following */
        Console.WriteLine("a is not less than 20");
      }
      Console.WriteLine("value of a is : {0}", a);
      Console.ReadLine();
    } }}
```

```
a is not less than 20;
value of a is : 100
```

The if...else if...else Statement

```
if(boolean_expression 1) {
   /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2) {
   /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3) {
   /* Executes when the boolean expression 3 is true */
} else {
   /* executes when the none of the above condition is true */
}
```

The if...else if...else Statement

```
using System;
namespace DecisionMaking {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 100;
            /* check the boolean condition */
      if (a == 10) {
        /* if condition is true then print the following */
        Console.WriteLine("Value of a is 10");
      }
      else if (a == 20) {
        /* if else if condition is true */
        Console.WriteLine("Value of a is 20");
      }
      else if (a == 30) {
        /* if else if condition is true  */
        Console.WriteLine("Value of a is 30");
      } else {
        /* if none of the conditions is true */
        Console.WriteLine("None of the values is matching");
      }
      Console.WriteLine("Exact value of a is: {0}", a);
      Console.ReadLine();} }}
```

```
None of the values is matching
Exact value of a is: 100
```

Short Hand If...Else (Ternary Operator)

There is also a short-hand if else, which is known as the **ternary operator** because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

Syntax

```
variable = (condition) ? expressionTrue :  expressionFalse;
```

Example

Instead of this

```
int time = 20;
if (time < 18)
{
  Console.WriteLine("Good day.");
}
else
{
  Console.WriteLine("Good evening.");
}
```

You can simply write:

```
int time = 20;
string result = (time < 18) ? "Good day." : "Good evening.";
Console.WriteLine(result);
```

C# - Switch Statement

```
switch(expression) {
  case constant-expression1  :
    statement(s);
    break;
  case constant-expression2  :
  case constant-expression3  :
    statement(s);
    break;

  /* you can have any number of case statements */
  default : /* Optional */
  statement(s);
}
```

C# - Switch Statement

```
using System;
namespace DecisionMaking {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      char grade = 'B';

      switch (grade) {
        case 'A':
          Console.WriteLine("Excellent!");
          break;
        case 'B':
        case 'C':
          Console.WriteLine("Well done");
          break;
        case 'D':
          Console.WriteLine("You passed");
          break;
        case 'F':
          Console.WriteLine("Better try again");
          break;
        default:
          Console.WriteLine("Invalid grade");
          break;
      }
      Console.WriteLine("Your grade is  {0}", grade);
      Console.ReadLine();
    }
  }
}
```

```
Well done
Your grade is B
```

C# - nested switch Statements

```
switch(ch1) {
  case 'A':
  Console.WriteLine("This A is part of outer switch" );

  switch(ch2) {
    case 'A':
      Console.WriteLine("This A is part of inner switch" );
      break;
    case 'B': /* inner B case code */
  }
  break;
  case 'B': /* outer B case code */
}
```

```
using System;

namespace DecisionMaking {
  class Program {
    static void Main(string[] args) {
      int a = 100;
      int b = 200;

      switch (a) {
        case 100:
        Console.WriteLine("This is part of outer switch ");

        switch (b) {
          case 200:
          Console.WriteLine("This is part of inner switch ");
          break;
        }
        break;
      }
      Console.WriteLine("Exact value of a is : {0}", a);
      Console.WriteLine("Exact value of b is : {0}", b);
      Console.ReadLine();
    } }}
```

```
This is part of outer switch
This is part of inner switch
Exact value of a is : 100
Exact value of b is : 200
```

C# - Loops

A **while** loop statement in C# repeatedly executes a target statement as long as a given condition is true.

```
while(condition) {
  statement(s);}
```

```
using System;

namespace Loops {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 10;

      /* while loop execution */
      while (a < 20) {
        Console.WriteLine("value of a: {0}", a);
        a++;
      }
      Console.ReadLine();
    }
  }
}
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

C# - For Loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

```
for ( init; condition; increment ) {
   statement(s);
}
```

```
using System;

namespace Loops {
   class Program {
      static void Main(string[] args) {

         /* for loop execution */
         for (int a = 10; a < 20; a = a + 1) {
            Console.WriteLine("value of a: {0}", a);
         }
         Console.ReadLine();
      }
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

C# - Do...While Loop

Unlike **for** and **while** loops, which test the loop condition at the start of the loop, the **do...while** loop checks its condition at the end of the loop.

```
do {
   statement(s);
} while( condition );
```

```
using System;
namespace Loops {
   class Program {
      static void Main(string[] args) {
         /* local variable definition */
         int a = 10;

         /* do loop execution */
         do {
            Console.WriteLine("value of a: {0}", a);
            a = a + 1;
         }
         while (a < 20);
         Console.ReadLine();
      }
   }
}
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

C# - Nested Loops

The syntax for a **nested for loop** statement in C# is as follows −

```
for ( init; condition; increment ) {
   for ( init; condition; increment ) {
      statement(s);
   }
   statement(s);
}
```

The syntax for a nested while loop statement in C# is as follows −

```
while(condition) {
   while(condition) {
      statement(s);
   }
   statement(s);
}
```

The syntax for a nested do...while loop statement in C# is as follows −

```
do {
   statement(s);
   do {
      statement(s);
   }
   while( condition );
}
while( condition );
```

```
using System;

namespace Loops {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int i, j;

      for (i = 2; i < 100; i++) {
        for (j = 2; j <= (i / j); j++)
        if ((i % j) == 0) break; // if factor found, not prime
        if (j > (i / j)) Console.WriteLine("{0} is prime", i);
      }
      Console.ReadLine();
    }
  }
}
```

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
C# provides the following control statements.

The **break** statement in C# has following two usage −

•When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

•It can be used to terminate a case in the **switch** statement.

Loop Control Statements

```
using System;

namespace Loops {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 10;

      /* while loop execution */
      while (a < 20) {
        Console.WriteLine("value of a: {0}", a);
        a++;

        if (a > 15) {
          /* terminate the loop using break statement */
          break;
        }
      }
      Console.ReadLine();
    }
  }
}
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

The continue statement in C# works somewhat like the **break** statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.

For the **for** loop, **continue** statement causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, **continue** statement causes the program control passes to the conditional tests.

```
using System;

namespace Loops {
  class Program {
    static void Main(string[] args) {
      /* local variable definition */
      int a = 10;

      /* do loop execution */
      do {
        if (a == 15) {
          /* skip the iteration */
          a = a + 1;
          continue;
        }
        Console.WriteLine("value of a: {0}", a);
        a++;
      }
      while (a < 20);
      Console.ReadLine();
    }
  }}
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

C# Methods

A **method** is a block of code which only runs when it is called.
You can pass data, known as parameters, into a method.
Methods are used to perform certain actions, and they are also known as **functions**.

Create a Method:

```
class Program
{
  static void MyMethod()
  {
    // code to be executed
  }
}
```

MyMethod() is the name of the method

static means that the method belongs to the Program class and not an object of the Program class.

void means that this method does not have a return value.

Call a Method;

To call (execute) a method, write the method's name followed by two parentheses **()** and a semicolon**;**

```
using System;

namespace MyApplication
{
  class Program
  {
    static void MyMethod()
    {
      Console.WriteLine("I just got executed!");
    }

    static void Main(string[] args)
    {
      MyMethod();
    }
  }
}
```

I**OUTPUT:** I just got executed!

C# Method Parameters

```
using System;

namespace MyApplication
{
  class Program
  {
    static void MyMethod(string fname)
    {
      Console.WriteLine(fname + " Refsnes");
    }

    static void Main(string[] args)
    {
      MyMethod("Liam");
      MyMethod("Jenny");
      MyMethod("Anja");
    }
  }
}
```

```
OUTPUT:        Liam Refsnes
               Jenny Refsnes
               Anja Refsnes
```

When a parameter is passed to the method, it is called an argument. So, from the example above: fname is a parameter, while Liam, Jenny and Anja are arguments.

Default Parameter Value

You can also use a default parameter value, by using the equals sign (=). If we call the method without an argument, it uses the default value ("Norway"):

```
static void MyMethod(string country = "Norway")
{
  Console.WriteLine(country);
}

static void Main(string[] args)
{
  MyMethod("Sweden");
  MyMethod("India");
  MyMethod();
  MyMethod("USA");
}
```

OUTPUT:

```
Sweden
India
Norway
USA
```

Method Overloading:

With **method overloading**, multiple methods can have the same name with different parameters:

```
int MyMethod(int x)
float MyMethod(float x)
double MyMethod(double x, double y)
```

Example

```
using System;
namespace MyApplication
{
  class Program
  {
    static int PlusMethodInt(int x, int y)
    {
      return x + y;
    }
    static double PlusMethodDouble(double x, double y)
    {
      return x + y;
    }
    static void Main(string[] args)
    {
      int myNum1 = PlusMethodInt(8, 5);
      double myNum2 = PlusMethodDouble(4.3, 6.26);
      Console.WriteLine("Int: " + myNum1);
      Console.WriteLine("Double: " + myNum2);
    }
  }
}
```

Instead of defining two methods that should do the same thing, it is better to overload one.

Overload the PlusMethod method to work for both int and double

```
static int PlusMethod(int x, int y)
{
  return x + y;
}

static double PlusMethod(double x, double y)
{
  return x + y;
}

static void Main(string[] args)
{
  int myNum1 = PlusMethod(8, 5);
  double myNum2 = PlusMethod(4.3, 6.26);
  Console.WriteLine("Int: " + myNum1);
  Console.WriteLine("Double: " + myNum2);
}
```

```
Int: 13
Double: 10.559999999999999
```

Classes and Objects

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

```
class Car
{
  string color = "red";
}
```

Create an Object

```
class Car
{
  string color = "red";

  static void Main(string[] args)
  {
    Car myObj = new Car();
    Console.WriteLine(myObj.color);
  }
}
```

Create two objects of Car:

```
class Car
{
  string color = "red";
  static void Main(string[] args)
  {
    Car myObj1 = new Car();
    Car myObj2 = new Car();
    Console.WriteLine(myObj1.color);
    Console.WriteLine(myObj2.color);
  }
}
```

Using Multiple Classes

prog.cs

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      Car myObj = new Car();
      Console.WriteLine(myObj.color);
    }
  }
}
```

prog2.cs

```
using System;

namespace MyApplication
{
  class Car
  {
    public string color = "red";
  }
}
```

Access modifier, which specifies that the color variable/field of Car is accessible for other classes as well, such as Program.

C# Class Members

Car class with three class members: two fields and one method.

```
// The class
class MyClass
{
  // Class members
  string color = "red";        // field
  int maxSpeed = 200;          // field
  public void fullThrottle()   // method
  {
    Console.WriteLine("The car is going as fast as it can!");
  }
}
```

Fields

- The variables inside a class are called fields, and that you can access them by creating an object of the class, and by using the dot syntax (.).
- The following example will create an object of the Car class, with the name myObj. Then we print the value of the fields color and maxSpeed

```
class Car
{
  string color = "red";
  int maxSpeed = 200;

  static void Main(string[] args)
  {
    Car myObj = new Car();
    Console.WriteLine(myObj.color);

Console.WriteLine(myObj.maxSpeed);
  }
}
```

```
class Car
{
  string color;
  int maxSpeed;

  static void Main(string[] args)
  {
    Car myObj = new Car();
    myObj.color = "red";
    myObj.maxSpeed = 200;
    Console.WriteLine(myObj.color);
    Console.WriteLine(myObj.maxSpeed);
  }
}
```

Note: This is especially useful when creating multiple objects of one class:

Multiple objects of one class

```
class Car
{
  string model;
  string color;
  int year;

  static void Main(string[] args)
  {
    Car Ford = new Car();
    Ford.model = "Mustang";
    Ford.color = "red";
    Ford.year = 1969;

    Car Opel = new Car();
    Opel.model = "Astra";
    Opel.color = "white";
    Opel.year = 2005;

    Console.WriteLine(Ford.model);
    Console.WriteLine(Opel.model);
  }
}
```

Object Methods

```
class Car
{
  string color;                // field
  int maxSpeed;                // field
  public void fullThrottle()    // method
  {
    Console.WriteLine("The car is going as fast as it can!");
  }

  static void Main(string[] args)
  {
    Car myObj = new Car();
    myObj.fullThrottle();  // Call the method
  }
}
```

Why did we declare the method as public, and not static

a static method can be accessed without creating an object of the class, while public methods can only be accessed by objects.

Use Multiple Classes

prog2.cs

```
using System;

namespace MyApplication
{
  class Car
  {
    public string model;
    public string color;
    public int year;
    public void fullThrottle()
    {
      Console.WriteLine("The car is going as fast as it
can!");
    }
  }
}
```

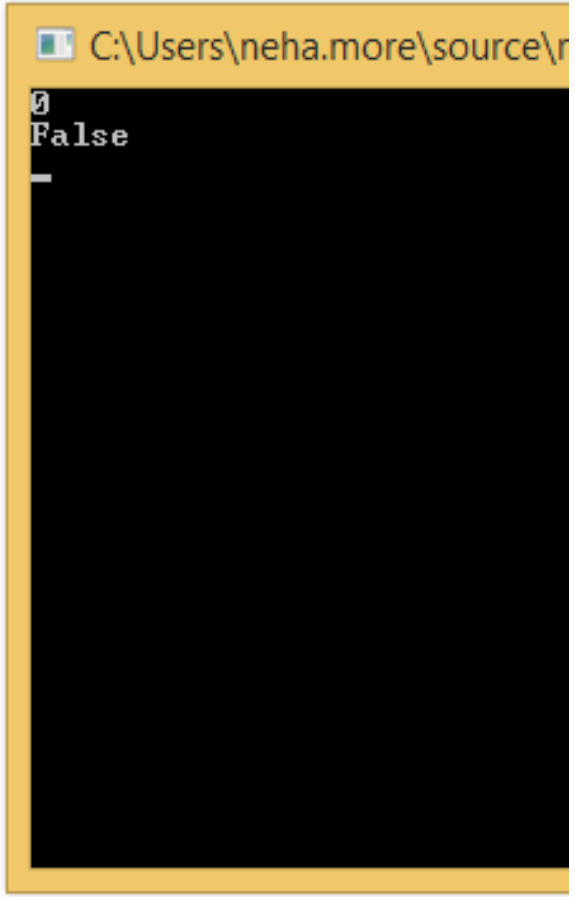
prog2.cs

```
using System;

namespace MyApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      Car Ford = new Car();
      Ford.model = "Mustang";
      Ford.color = "red";
      Ford.year = 1969;

      Car Opel = new Car();
      Opel.model = "Astra";
      Opel.color = "white";
      Opel.year = 2005;

      Console.WriteLine(Ford.model);
      Console.WriteLine(Opel.model);
    }
  }
}
```

Constructors

A constructor is a **special method** that is used to initialize objects. The advantage of a constructor, is that it is called when an object of a class is created.

```
using System;

namespace DemoProject
{
    2 references
    class Program
    {
        int i;
        bool b;
        0 references
        static void Main(string[] args)
        {
            Program p = new DemoProject.Program();
            Console.WriteLine(p.i);
            Console.WriteLine(p.b);
            Console.ReadLine();
        }
    }
}
```



```
using System;
using System.Collections.Generic;
using System.Text;

namespace DemoProject
{
    3 references
    class Class1
    {
        1 reference
        public Class1()
        {
            Console.WriteLine("Constructor is called");
        }
        0 references
        static void Main()
        {
            Class1 c1 = new Class1();
            Console.ReadLine();
        }
    }
}
```

C:\Users\neha.more\source

```
Constructor is called
```

```
class Car
{
  public string model;  // Create a field

  // Create a class constructor for the Car class
  public Car()
  {
    model = "Mustang"; // Set the initial value for model
  }

  static void Main(string[] args)
  {
    Car Ford = new Car();  // Create an object of the Car Class
(this will call the constructor)
    Console.WriteLine(Ford.model);  // Print the value of model
  }
}
```

- Note that the constructor name must match the class name, and it cannot have a return type (like void or int).
- Also note that the constructor is called when the object is created.
- All classes have constructors by default: if you do not create a class constructor yourself, C# creates one for you. However, then you are not able to set initial values for fields.

Constructor Parameters

Example

```
class Car
{
  public string model;

  // Create a class constructor with a parameter
  public Car(string modelName)
  {
    model = modelName;
  }

  static void Main(string[] args)
  {
    Car Ford = new Car("Mustang");
    Console.WriteLine(Ford.model);
  }
}
```

Example

```
class Car
{
  public string model;
  public string color;
  public int year;

  // Create a class constructor with multiple parameters
  public Car(string modelName, string modelColor, int modelYear)
  {
    model = modelName;
    color = modelColor;
    year = modelYear;
  }

  static void Main(string[] args)
  {
    Car Ford = new Car("Mustang", "Red", 1969);
    Console.WriteLine(Ford.color + " " + Ford.year + " " + Ford.model);
  }
}
```

Without constructor:

Prog.cs

```
class Program
{
  static void Main(string[] args)
  {
    Car Ford = new Car();
    Ford.model = "Mustang";
    Ford.color = "red";
    Ford.year = 1969;

    Car Opel = new Car();
    Opel.model = "Astra";
    Opel.color = "white";
    Opel.year = 2005;

    Console.WriteLine(Ford.model);
    Console.WriteLine(Opel.model);
  }
}
```

Prog2.cs

```
using System;

namespace MyApplication
{
  class Car
  {
    public string model;
    public string color;
    public int year;
    public void fullThrottle()
    {
      Console.WriteLine("The car is going as fast as it can!");
    }
  }
}
```

With constructor:

Prog.cs

```
class Program
{
  static void Main(string[] args)
  {
    Car Ford = new Car("Mustang", "Red", 1969);
    Car Opel = new Car("Astra", "White", 2005);

    Console.WriteLine(Ford.model);
    Console.WriteLine(Opel.model);
  }
}
```

Prog2.cs

```
using System;

namespace MyApplication
{
  class Car
  {
    public string model;
    public string color;
    public int year;

    public Car(string modelName, string modelColor, int modelYear)
    {
      model = modelName;
      color = modelColor;
      year = modelYear;
    }
  }
}
```

Inheritance:

```
class derived-class : base-class
{
  // methods and fields
  .
  .
}
```

Class A
{
-Members
}
Class B
{
-consuming the members of A from here
}

Syntax:
[<modifiers>] class <child class>:<parent class>

Note: In inheritance child class can consume members of its parent class as if it is the owner of those members(expect private members of parent)

Class2.cs | class1.cs

inheritance | inheritance.class1 | Test2()

```
1   using System;
2
3   namespace inheritance
4   {
        1 reference
5       class class1
6       {
            1 reference
7           public void Test1()
8           {
9               Console.WriteLine("Method 1");
10          }
            1 reference
11          public void Test2()
12          {
13              Console.WriteLine("Method 2");
14          }
15      }
16  }
17
```

```
using System;


namespace inheritance
{
    2 references
    class Class2 : class1
    {
        0 references
        static void main()
        {
            Class2 c = new Class2();
            c.
            Co
        }
    }
}
```

Equals
GetHashCode
GetType
MemberwiseClone
Test1
Test2
ToString

bool object.Equa
Determines whe
Note: Tab twice

Class2.cs | class1.cs

inheritance | inheritance.Class2

```
1   using System;
2
3
4   namespace inheritance
5   {
        2 references
6       class Class2 : class1
7       {
            0 references
8           static void Main()
9       {
10          Class2 c = new Class2();
11              c.Test1();
12              c.Test2();
13          Console.ReadLine();
14      }
15      }
16  }
17
```

```
ass1

ain()

w Class2();
;
;
ine();
```

C:\Users\neha.more\source\re

```
Method 1
Method 2
```

Every class has a implicit constructor…by default implicit constructor

Class2.cs* class1.cs

inheritance inheritance.Class2

```
1   using System;
2
3
4   namespace inheritance
5   {
        2 references
6       class Class2 : class1
7       {
            1 reference
8           public void Test3()
9           {
10              Console.WriteLine("Method 3");
11          }
            0 references
12          static void Main()
13      {
14          Class2 c = new Class2();
15              c.Test1();
16              c.Test2();
17              c.Test3();
18              Console.ReadLine();
19      }
20      }
21  }
22
```

Parent classes constructor must be accessible to child class, otherwise inheritance will not be possible

```
using System;


namespace inheritance
{
    2 references
    class Class2 : class1
    {
        1 reference
        public void Test3()
        {
            Console.WriteLine("Method 3");
        }
        0 references
        static void Main()
    {
        Class2 c = new Class2();
            c.Test1();
            c.Test2();
            c.Test3();
            Console.ReadLine();
    }
    }
}
```

C:\Users\neha.more\source\repo

```
class 1 constructor called
Method 1
Method 2
Method 3
```

Whenever child class instantiate --- child class implicitly called its parents class constructor..

- **In inheritance child class can consume members of its parent class as if it is the owner of those members(expect private members of parent)**
- **<u>All the members of the class are by default private</u>.**

```
1   using System;
2
3   namespace inheritance
4   {
        2 references
5       class class1
6       {
            0 references
7           class1()
8           {
9               Console.WriteLine("class 1 constructor called");
10          }
            1 reference
11          public void Test1()
12          {
13              Console.WriteLine("Method 1");
14          }
            1 reference
15          public void Test2()
16          {
17              Console.WriteLine("Method 2");
18          }
19      }
20  }
```

	Code	Description	Project	File	Line	Suppression State
⊗	CS0122	'class1.class1()' is inaccessible due to its protection level	inheritance	Class2.cs	6	Active
ⓘ	IDE1006	Naming rule violation: These words must begin with upper case characters: class1	inheritance	class1.cs	5	Active

In inheritance child class can access parent classes members but parent classes can never access any member that is purely defined under the child class.

```
0 references
public void Test3()
{
    Console.WriteLine("Method 3");
}
0 references
static void Main()
{
    class1 c = new class1();
    c.
    Co                          ne();
}
}
```

Equals
GetHashCode
GetType
Test1
Test2
ToString

void class1.Test2()

Test3 is not accessible through class 1 instance.

```
9       {
            0 references
10          public void Test3()
11          {
12              Console.WriteLine("Method 3");
13          }
            0 references
14          static void Main()
15          {
16              class1 c = new class1();
17              c.Test1();
18              c.Test2();
19              c.Test3();
20              Console.ReadLine();
21          }
22
23      }
24  }
25
```

1 0 Ln: 19 Ch: 23

List

re Solution | 1 Error | 0 Warnings | 2 Messages | Build + IntelliSense | Search Error List

	Code	Description	Project	File	Line	Suppression S
⊗	CS1061	'class1' does not contain a definition for 'Test3' and no accessible extension method 'Test3' accepting a first argument of type 'class1' could be found (are you missing a using directive or an assembly reference?)	inheritance	class2.cs	19	Active
ⓘ	IDE1006	Naming rule violation: These words must begin with upper case characters: class1	inheritance	class1.cs	5	Active
ⓘ	IDE1006	Naming rule violation: These words must begin with upper case characters: class2	inheritance	class2.cs	7	Active

We can initialize a parent classes variable by using the child class inheritance to make it as a reference so that the reference will be consuming the memory of child class instance, but in this case also we cant call pure child class member by using reference.

```
class class2:class1

    {
        public void Test3()
        {
            Console.WriteLine("Method 3");
        }
        static void Main()
        {
            //class1 c = new class1();
            //c.Test1();
            //c.Test2();
            //c.Test3();
            class1 p;//p is a variable of class1
            p.Test1();
            p.Test2();
            Console.ReadLine();
        }
    }
```

```
0 references
public void Test3()
{
    Console.WriteLine("Method 3");
}
0 references
static void Main()
{
    //class1 c = new class1();
    //c.Test1();
    //c.Test2();
    //c.Test3();
    class1 p;//p is a variable of class1
    p.Test1();
    p.Test2();
    Console.ReadLine();
}

}
```

1 0

List

e Solution | 1 Error | 0 Warnings | 2 Messages | Build + IntelliSense

Code	Description	Project	File
CS0165	Use of unassigned local variable 'p'	inheritance	class2.cs

P is not instance it is variable. you will not able to access methods.

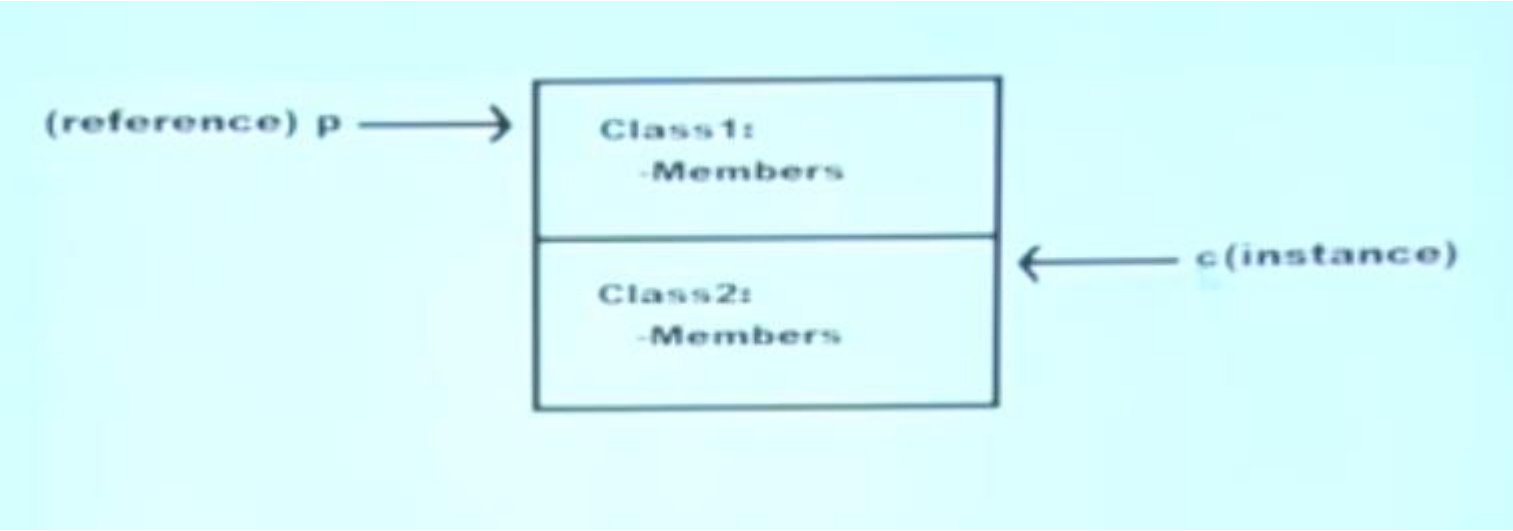
```
0 references
class class2:class1

{
    0 references
    public void Test3()
    {
        Console.WriteLine("Method 3");
    }
    0 references
    static void Main()
    {
                class1 p;//p is a variable of class1
        class2 c = new class2();//c is instance of class 2
        //p= new class1();memory allocation done
        p = c;//initialize the parent class variable using instance of child class
        p.Test1();
        p.Test2();
        Console.ReadLine();
    }
```

D:\AWT\inheritance\inheritance\bin\Debug\netcoreapp3.1\inheritance.exe

```
Method 1
Method 2
```

p is a reference it is consuming a memory of instance c.

P is a reference of parent class created by using child class instance.



Memory allocation is done only for the instance not for the references.
p and c are using same memory, still p can not access the child class members.

Access specifier in C#

Access Modifiers	Same Assembly(Accessmodifers)			Different Assembly(Demoproject1)	
	Same class Program.cs	Derived Class Class1.cs	Non Derived Class2.cs	Derived Class Class3.cs	Non Derived Class Class4.cs
Private Test1()	+	×	×	×	×
Protected Test2()	+	+	×	+	×
Internal Test3()	+	+	+	×	×
Protected internal Test4()	+	+	+	+	×
Public Test5()	+	+	+	+	+

Sample Program: **Consuming members of class from the same class**

Class Program

```
using System;

namespace accessdemo1
{
    class Program
    {
        private void Test1()
        {
            Console.WriteLine("private method");
        }
        internal void Test2()
        {
            Console.WriteLine("Internal method");
        }
        protected void Test3()
        {
            Console.WriteLine("protected method");
        }
```

Make class as a public

```
protected internal void Test4()
        {
            Console.WriteLine("protected internal method");
        }
        public void Test5()
        {
            Console.WriteLine("public method");
        }
        static void Main(string[] args)
        {
            Program p = new Program();
            p.Test1();
            p.Test2();
            p.Test3();
            p.Test4();
            p.Test5();
            Console.ReadLine();
        }
    }
}
```

```
        {
            Console.WriteLine("protected internal method")
        }
        0 references
        public void Test5()
        {
            Console.WriteLine("public method");
        }
        0 references
        static void Main(string[] args)
        {
            Program p = new Program();
            p.Test1();
            p.Test2();
            p.
        }
    }
```

GetHashCode
GetType
MemberwiseClone
Test1
Test2
Test3
Test4
Test5
ToString

void Program.Test2()

0

:kage Manager

ccessdemo1\acce csproj (in 4.34 sec).
30:18.3469339
ed ==========

D:\AWT\accessdemo1\accessdemo1\bin\Debug\netcoreapp3.1\accessdemo1....

```
private method
Internal method
protected method
protected internal method
public method
```

Class Two :-To demonstrate Private access specifier

Consuming members of class from child class

```
Class Two

using System;
namespace accessdemo1
{
    class Two:Program
    {
        static void Main()
        {
            Two t1 = new Two();
            t1.Test2();
            t1.Test3();
            t1.Test4();
            t1.Test5();
            Console.ReadLine();
        }
            }
}
```

```
namespace accessdemo1
{
    2 references
    class Two:Program
    {
        0 references
        static void main()
        {
            Two t1 = new Two();
            t1.
        }
        }
}
```

Equals
GetHashCode
GetType
MemberwiseClone
Test2
Test3
Test4
Test5
ToString

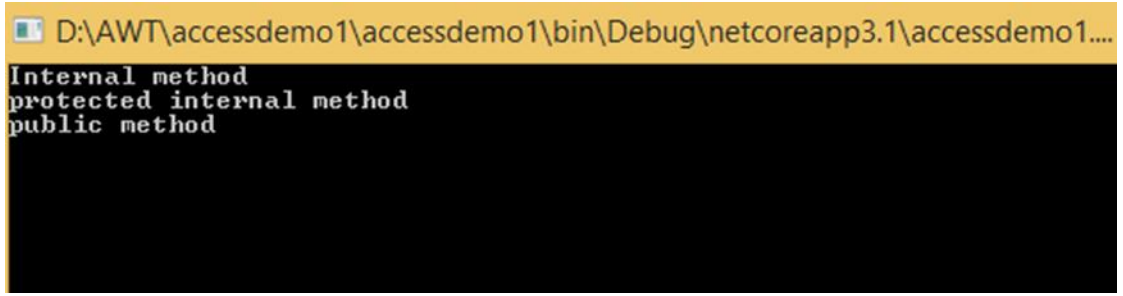
bool object.Equals(object? obj)
Determines whether the specified object is equal to the current object.
Note: Tab twice to insert the 'Equals' snippet.

D:\AWT\accessdemo1\accessdemo1\bin\Deb

```
Internal method
protected method
protected internal method
public method
```

Consuming a member of a class by creating an instance of a class(without Inheritance) same project

```
Class Three

using System;

namespace accessdemo1
{
    class Three
    {
        static void main()
        {
            Program p = new Program();
            p.Test2();
            p.Test4();
            p.Test5();
            Console.ReadLine();
        }

    }
}
```



Private:

Protected: Protected is also not detected because it can access only with the child class/Inheritance

```
namespace accessdemo1
{
    0 references
    class Three
    {
        0 references
        static void main()
        {
            Program p = new Program();
            p.
        }

    }
}
```

Equals
GetHashCode
GetType
Test2
Test4
Test5
ToString

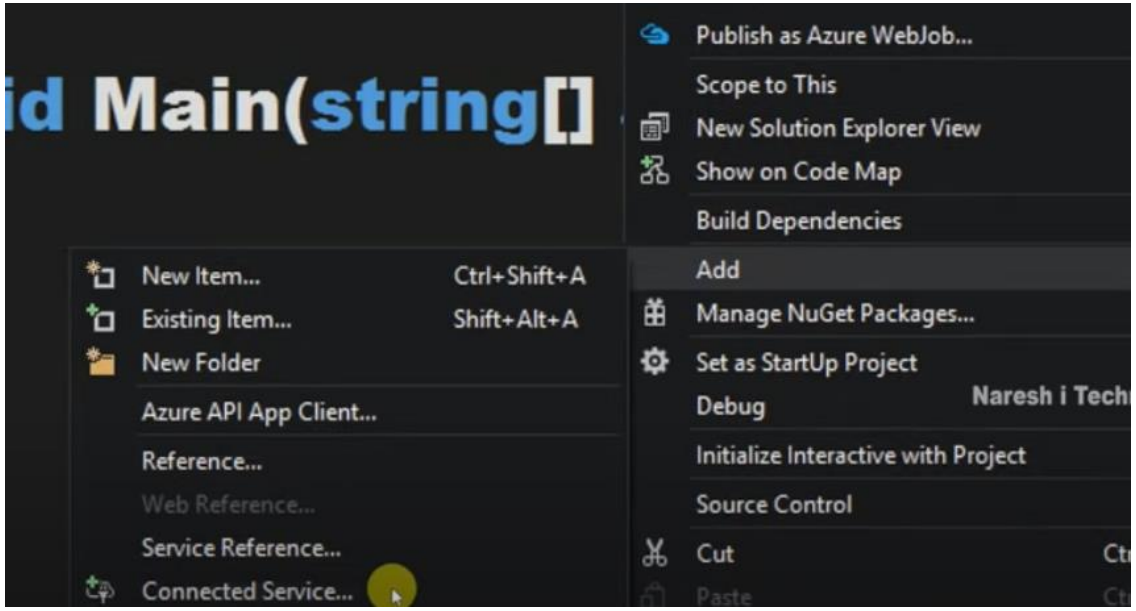
bool object.Equals(object? obj)
Determines whether the specified object is equal to the current object.
Note: Tab twice to insert the 'Equals' snippet.

New Project accessdemo2

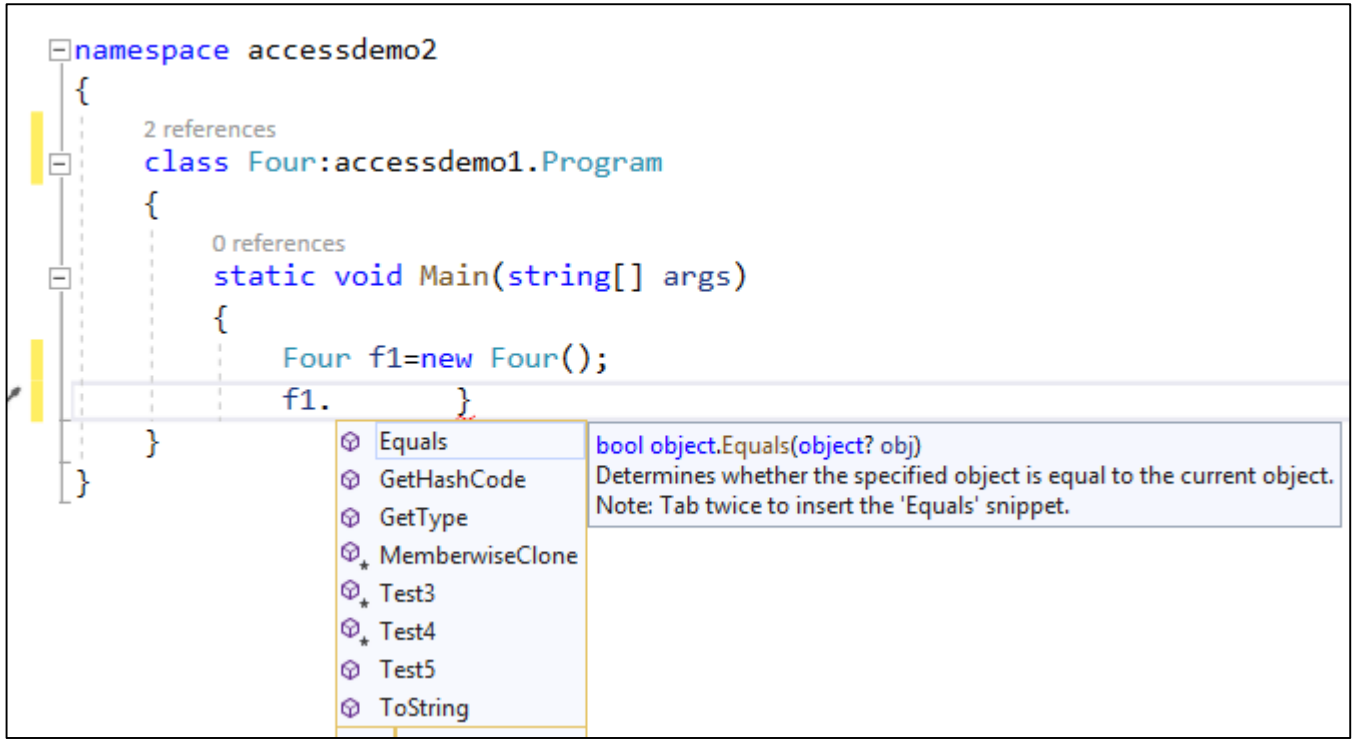
Consuming members of a class from different project

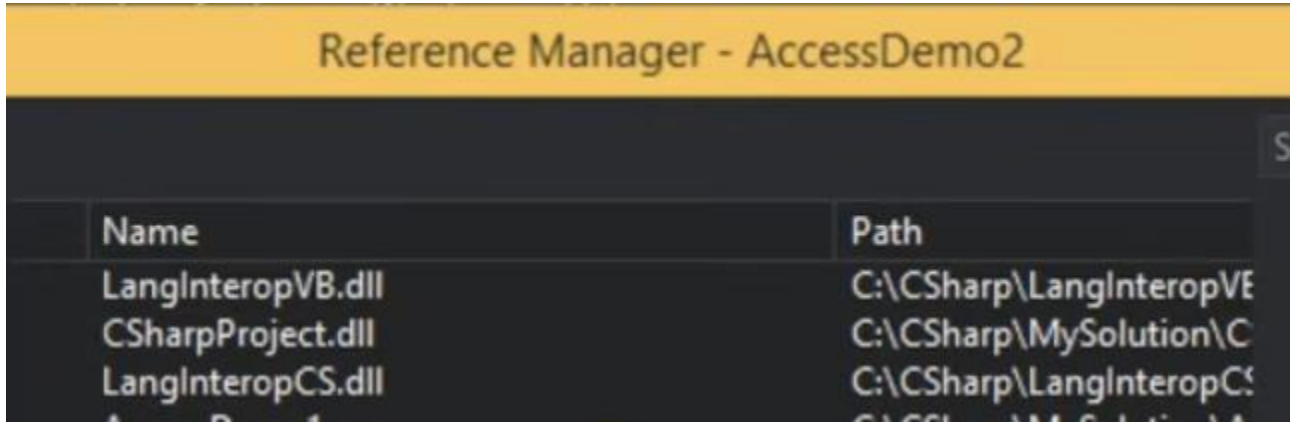
```
Four.cs

using System;

namespace accessdemo2
{
    class Four:accessdemo1.Program
    {
        static void Main(string[] args)
        {
            Four f1=new Four();
            f1.Test3();
            f1.Test4();
            f1.Test5();
            Console.ReadLine();

        }
    }
}
```

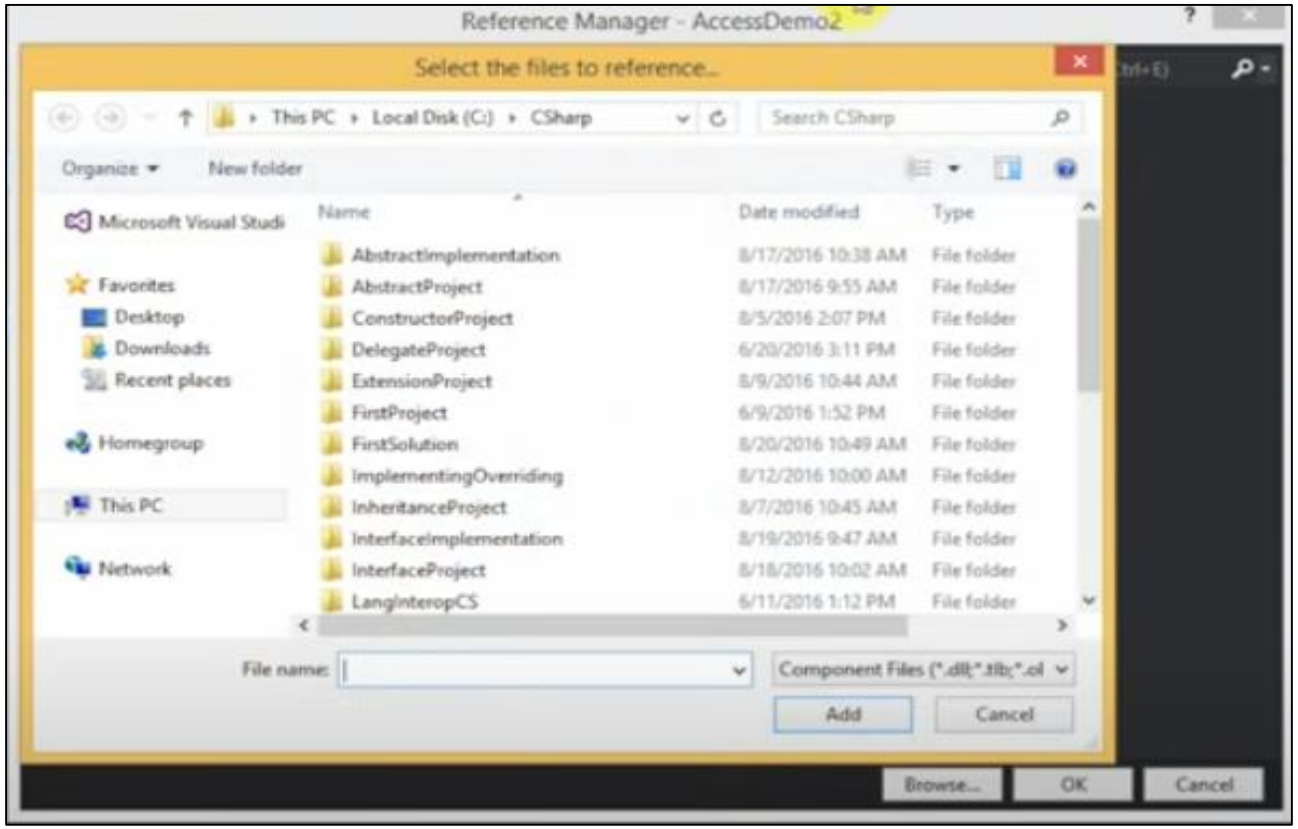
```
protected method
protected internal method
public method
```



Adding reference




Reference Manager - AccessDemo2
Name
Path
LangInteropVB.dll
C:\CSharp\LangInteropVE
CSharpProject.dll
C:\CSharp\MySolution\C
LangInteropCS.dll
C:\CSharp\LangInteropC


Reference Manager - AccessDemo2
Select the files to reference...
This PC
Local Disk (C:)
CSharp
Search CSharp
Organize
New folder
Microsoft Visual Studi
Favorites
Desktop
Downloads
Recent places
Homegroup
This PC
Network
Name
Date modified
Type
AbstractImplementation
AbstractProject
ConstructorProject
DelegateProject
ExtensionProject
FirstProject
FirstSolution
ImplementingOverriding
InheritanceProject
InterfaceImplementation
InterfaceProject
LangInteropCS
File folder
File name:
Add
Cancel
Browse...
OK
Cancel

Consuming a member of a class by creating an instance of a class(without Inheritance) different project

```
using System;
namespace accessdemo2
{
    class Five
    {
        static void Main()
        {
            accessdemo1.Program p = new accessdemo1.Program();
            p.Test5();
                Console.ReadLine();
        }}}
```

```
0 references
class Five
{
    0 references
    static void Main()
    {
        accessdemo1.Program p = new accessdemo1.Program();
        p.
            Equals
            GetHashCode
            GetType
            Test5
            ToString
            void accessdemo1.Program.Test5()
        }
    }
}
```

```
D:\AWT\accessdemo1\accessdemo2\bin\Debu
public method
```

Abstract classes and Abstract methods:

A method without any method body is known as an abstract method ,what the method contains is only declaration of the method

Ex:

```
Public void add(int x,int y)
{
}
```

Non abstract methods

Public void add(int x,int y); ---------- abstract method

Public abstract void add(int x,int y); ---------- abstract method

Abstract class

```
Abstract class Math
{
Public abstract void add(int x,int y);
}
```

If a method is declared as abstract under any class then the child class of that class is responsible for implementing the method.

It is same as method overriding:

Method overriding

```
class Class1
{
 public virtual void Show()
 {
 }
}
class Class2 : Class1
{
 public override void Show()     //Optional
 {
  -Re-Implementation
 }
}
```

Abstract class

```
abstract class Class1
{
 public abstract void Show();
}
class Class2 : Class1
{
 public override void Show()     //Mandatory
 {
  -Implementation
 }
}
```

Abstract Class:

- -Abstract Methods
- -Non-abstract Methods

Child Class of Abstract Class:

-Implement each and every abstract method of parent class.

-Now only we can consume non-abstract methods of parent class.

Example:

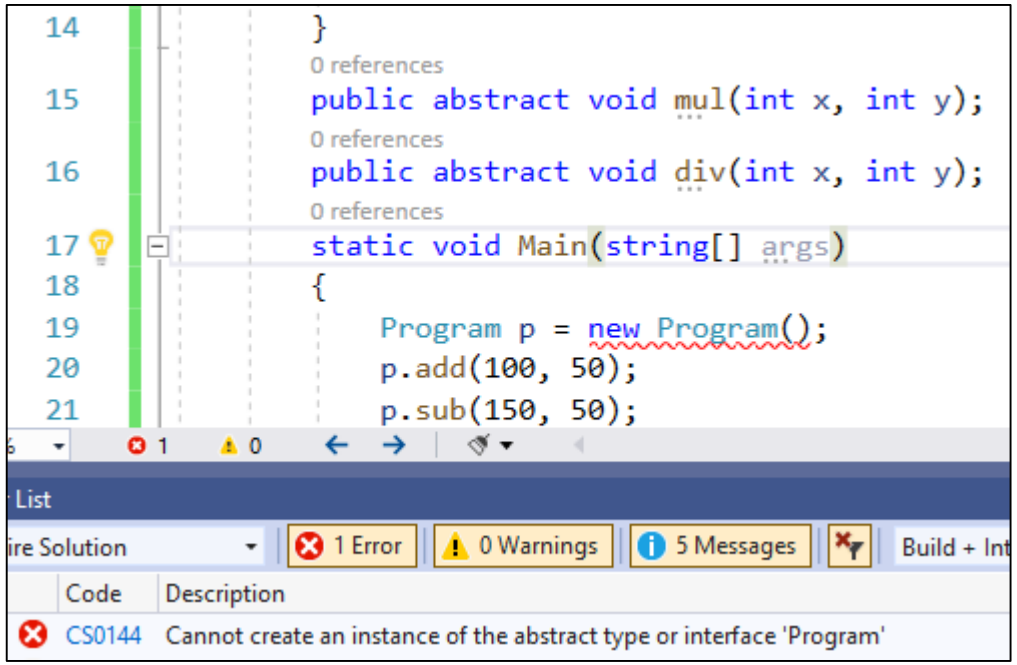
Asset and Liability
Bike and percentage

Abstract---target
Non Abstract---bike

Creating an instance of an abstract class

Parent.cs

```
using System;
namespace abstract_demo
{
   abstract class parent
   {
      public void add(int x, int y)
      {
         Console.WriteLine(x + y);
      }
      public void sub(int x, int y)
      {
         Console.WriteLine(x - y);
      }
      public abstract void mul(int x, int y);
      public abstract void div(int x, int y);
      static void Main(string[] args)
      {
         parent p = new parent();
         p.add(100, 50);
         p.sub(150, 50);
         Console.ReadLine();
      }}}
```



Child.cs

```
using System;
namespace abstract_demo
{
    class child:parent
    {
        static void Main()
        {
            child c = new child();
            c.add(100, 50);
            c.sub(150, 50);
        }
    }
}
```

```
Build started...
1>------ Build started: Project: abstract demo, Configuration: Debug Any CPU ------
1>D:\AWT\abstract demo\abstract demo\child.cs(6,11,6,16): error CS0534: 'child' does not implement inherited abstract member 'parent.mul(int, int)'
1>D:\AWT\abstract demo\abstract demo\child.cs(6,11,6,16): error CS0534: 'child' does not implement inherited abstract member 'parent.div(int, int)'
1>Done building project "abstract demo.csproj" -- FAILED.
```

Calling non abstract method and implementing abstract method.(calling abstract method is not compulsory)

Parent.cs

```
using System;
namespace abstract_demo
{
    public abstract class parent
    {
        public void add(int x, int y)
        {
            Console.WriteLine(x + y);
        }
        public void sub(int x, int y)
        {
            Console.WriteLine(x - y);
        }
        public abstract void mul(int x, int y);
        public abstract void div(int x, int y);

    }
}
```

Child.cs

```
using System;
namespace abstract_demo
{
    class child:parent
    {
        public override void mul(int x, int y)
        {
            Console.WriteLine(x*y);
        }
        public override void div(int x, int y)
        {
            Console.WriteLine(x / y);
        }
        static void Main()
        {
            child c = new child();
            c.add(100, 50);
            c.sub(150, 50);
        }
    }
}
```

OutPut:

```
150
100

D:\AWT\abstract demo\abstract demo\bin\Debug\netcoreapp3.1\abstract demo.exe
ocess 1864) exited with code 0.
Press any key to close this window . . .
```

Calling abstract and non abstract method:---- implementing abstract method.(calling abstract method is not compulsory)

Parent.cs

```
using System;

namespace abstract_demo
{
    public abstract class parent
    {
        public void add(int x, int y)
        {
            Console.WriteLine(x + y);
        }
        public void sub(int x, int y)
        {
            Console.WriteLine(x - y);
        }
        public abstract void mul(int x, int y);
        public abstract void div(int x, int y);

    }
}
```

Child.cs

```
using System;
namespace abstract_demo
{
    class child:parent
    {
        public override void mul(int x, int y)
        {
            Console.WriteLine(x*y);
        }
        public override void div(int x, int y)
        {
            Console.WriteLine(x / y);
        }
        static void Main()
        {
            child c = new child();
            c.add(100, 50);
            c.sub(150, 50);
            c.mul(15,2);
            c.div(14,2);
        }}}
```

```
150
100
30
7

D:\AWT\abstract demo\abstract demo\bin\Debug\netcoreapp3.1\abstract dem
ocess 11192) exited with code 0.
Press any key to close this window
```

In Abstract class we can not create instance but we can create reference of abstract class.

(you can use parent class reference using child class instance)

```
2 references
public override void mul(int x, int y)
{
    Console.WriteLine(x*y);
}
2 references
public override void div(int x, int y)
{
    Console.WriteLine(x / y);
}
0 references
static void Main()
{
    child c = new child();
    parent p = c;
    p.add(100, 50);
    p.sub(150, 50);
    p.mul(15,2);
    p.div(14,2);
}
}
```

Microsoft Visual Studio Debug Console

```
150
100
30
7

D:\AWT\abstract demo\abstract demo\bin\Debug\netcoreapp3.1\abstract demo.exe (pr
ocess 8820) exited with code 0.
Press any key to close this window . . .
```

Example 2: **Animal.cs**

```
using System;
namespace abstract_demo
{
    // Abstract class
    abstract class Animal
    {
        // Abstract method (does not have a body)
        public abstract void animalSound();
        // Regular method
        public void sleep()
        {
            Console.WriteLine("Zzz");
        }
    }

    // Derived class (inherit from Animal)
    class Pig : Animal
    {
        public override void animalSound()
        {
            // The body of animalSound() is provided here
            Console.WriteLine("The pig says: wee wee");
        }
        static void Main(string[] args)
        {
            Pig myPig = new Pig();  // Create a Pig object
            myPig.animalSound();
            myPig.sleep();
        } }}
```

```
The pig says: wee wee
Zzz

D:\AWT\abstract demo\abstract demo\bin\Debug
ocess 1208) exited with code 0.
Press any key to close this window . . .
```

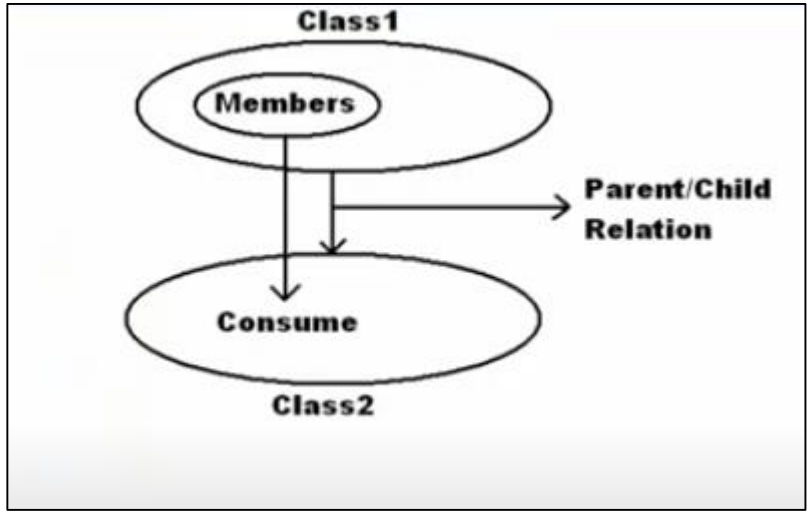
Interface

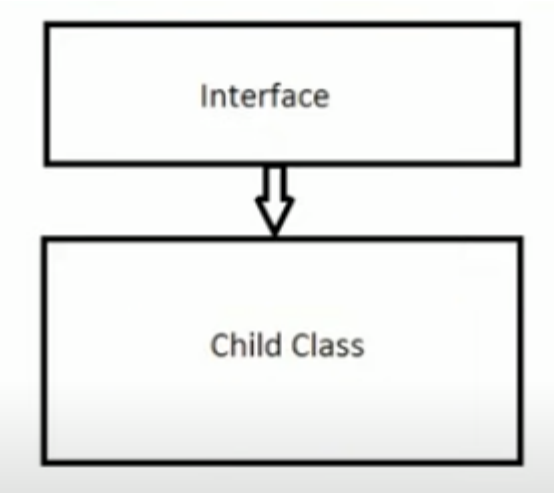
Class:
- It's a user defined data types
- Non abstract methods(methods with method body)

Interface:
- This is also an user-defined data type
- Contains only abstract Methods

Abstract Class:
- Non abstract method (Methods with method body) and also abstract Methods(Methods without method body)

Every abstract method of interface should be implemented by the child class of the interface-Mandatory





Generally class consumes the member of a class to consume the members of its parents where as class is inheriting from an interface it is to implement the members of its parent.

Note: A class can inherit from a class and interface at a time.

Syntax of Interface:

```
[<modifiers>] class <Name>
{
 -Define Members here
}

[<modifiers>] interface <Name>
{
 -Abstract Member Declarations here
}
```

Rule: The default scope the members of an interface is public whereas its private in case of a class.

To define a method in interface

Void add(int a,int b); ----------By default the methods/member are abstract and public don't require to use abstract modifier on it again just like we do in case of abstract class

```
namespace interfacedemo
{
    0 references
    interface ItestInterface1
    {
        int x;
        0 referen
        void a
    }
}
```

(field) int ItestInterface1.x

CS0525: Interfaces cannot contain instance fields

We can not declare any field/variable under an interface

If required an interface can inherit from another interface

```
using System;
namespace interfacedemo
{
    2 references
    interface ItestInterface1
    {
        0 references
        void Add(int a, int b);
    }
    0 references
    interface ItestInterface2:ItestInterface1
    {
        0 references
        void Sub(int a, int b);
    }
    0 references
    class Implementationclass:ItestInterface1
}
```

interface interfacedemo.ItestInterface1

CS0535: 'Implementationclass' does not implement interface member 'ItestInterface1.Add(int, int)'

Show potential fixes (Alt+Enter or Ctrl+.)

```
using System;
namespace interfacedemo
{
    1 reference
    interface ItestInterface1
    {
        0 references
        void Add(int a, int b);
    }
    1 reference
    interface ItestInterface2:ItestInterface1
    {
        0 references
        void Sub(int a, int b);
    }
    0 references
    class Implementationclass:ItestInterface2
}
```

interface interfacedemo.ItestInterface2

CS0535: 'Implementationclass' does not implement interface member 'ItestInterface2.Sub(int, int)'

CS0535: 'Implementationclass' does not implement interface member 'ItestInterface1.Add(int, int)'

Show potential fixes (Alt+Enter or Ctrl+.)

By default class members are private so we need to write public

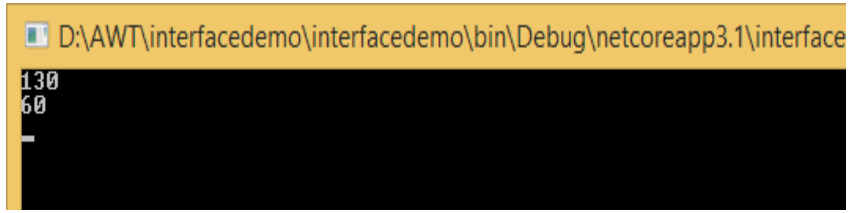
```
class Implementationclass:ItestInterface2
    {
        public void Add(int a,int b)
        {
        }
        public void Sub(int a, int b)
        {

        }
```

You can implement like this also

```
class Implementationclass:ItestInterface2
    {
        void ItestInterface1.Add(int a,int b)
        {
        }
        public void Sub(int a, int b)
        {

        }
```

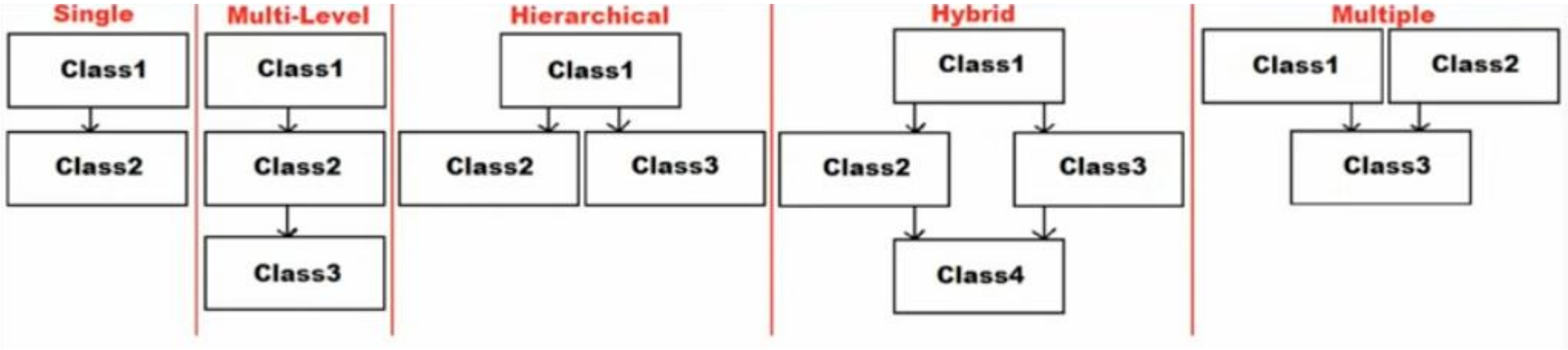
```
using System;
namespace interfacedemo
{
    interface ItestInterface1
    {
        void Add(int a, int b);
    }
    interface ItestInterface2:ItestInterface1
    {
        void Sub(int a, int b);
    }
    class Implementationclass:ItestInterface2
    {
        public void Add(int a,int b)
        {
            Console.WriteLine(a + b);
        }
        public void Sub(int a, int b)
        {
            Console.WriteLine(a - b);
        }
```

```
static void Main()
        {
            Implementationclass obj =
new Implementationclass();
            obj.Add(100,30);
            obj.Sub(100,40);
            Console.ReadLine();
        }
    }
}
```



Multiple Inheritance with Interface:

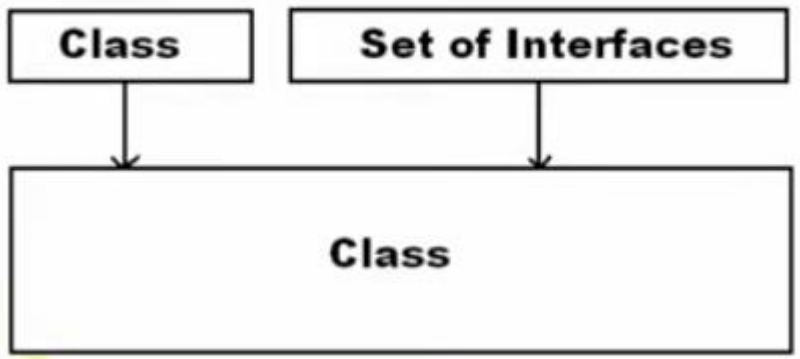
Types Of Inheritance:



In c# --It supports only single, Multilevel, hierarchical Inheritance but hybrid and Multiple Inheritances are not supported.

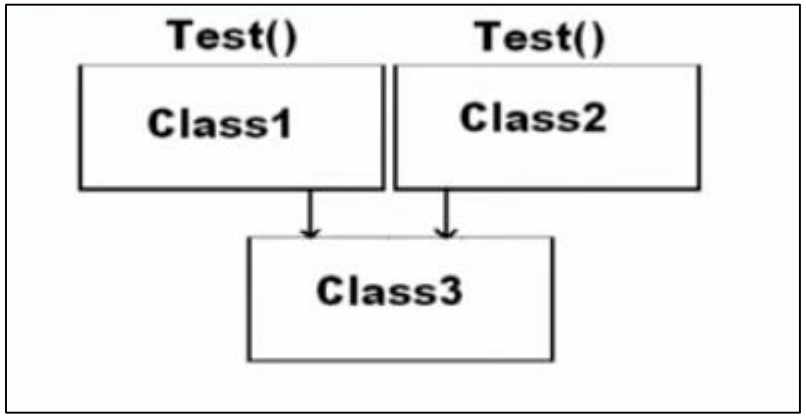
Even if Multiple inheritance is not supported through classes in c#, It is still supported through Interfaces.

A class can have one and only one immediate parent class. whereas the same class can have any number of interfaces as its parents. i.e. Multiple inheritance is supported in C# through interfaces

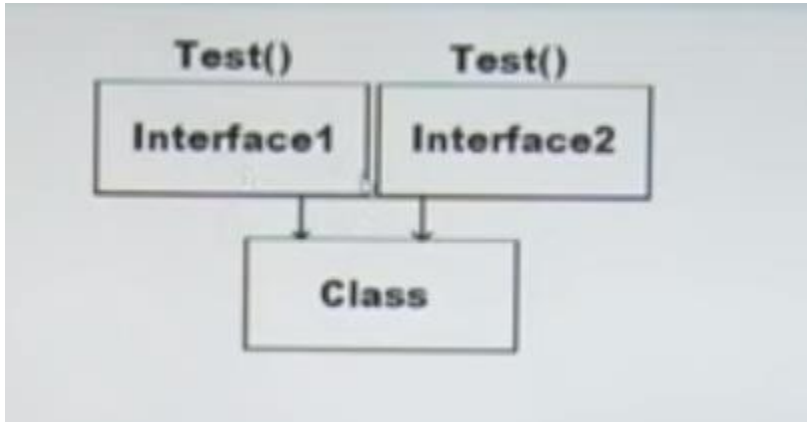


Why Multiple inheritance is not supported and how is it supported through interfaces?

Ambiguity Problem:



Consuming a methods-Directly calling



Request to a class not to consuming the method but to implement the method

Consumption will cause an ambiguity not an implementation.

Example

```
using System;
namespace interfaceinheritance
{
    interface Interface1
    {
        void Test();
    }
    interface Interface2
    {
        void Test();
    }
    class Implementation:Interface1,Interface2
    {
        static void Main()
        {
            Console.ReadLine();
        }
    }
}
```

Showing an error because methods are not implemented.

Error List

Entire Solution | 2 Errors | 0 Warnings | 3 Messages | Build + IntelliSense

Code	Description
CS0535	'Implementation' does not implement interface member 'Interface1.Test()'
CS0535	'Implementation' does not implement interface member 'Interface2.Test()'
IDE1006	Naming rule violation: These words must begin with upper case characters: nterface1
IDE1006	Naming rule violation: These words must begin with upper case characters: nterface2

Implement a method in class:-No output but blank screen

```
class Implementation:Interface1,Interface2
{
    2 references
    public void Test()
    {
        Console.WriteLine("Interface Method is implemented in child class");
    }
    0 references
    static void Main()
    {
        Console.ReadLine();
    }
}
```

D:\AWT\interfaceinheritance\interfaceinherita

Interface is asking to implement a method not to consuming a method..

Implementation is for both the interfaces.

```
2 references
class Implementation:Interface1,Interface2
{
    3 references
    public void Test()
    {
        Console.WriteLine("Interface Method is implemented in child class");
    }
    0 references
    static void Main()
    {
        Implementation i1 = new Implementation();
        i1.Test();
        Console.ReadLine();
    }
}
```

D:\AWT\interfaceinheritance\interfaceinheritance\bin\Deb

```
Interface Method is implemented in child class
```

Full Program

```
using System;
namespace interfaceinheritance
{
    interface Interface1
    {
        void Test();
    }
    interface Interface2
    {
        void Test();
    }
    class Implementation:Interface1,Interface2
    {
        public void Test()
        {
            Console.WriteLine("Interface Method is implemented in child class");
        }
        static void Main()
        {
            Implementation i1 = new Implementation();
            i1.Test();
            Console.ReadLine();
        }}}
```

Another Method:

```
using System;
namespace interfaceinheritance
{
    interface Interface1
    {
        void Test();
        void Show();
    }
    interface Interface2
    {
        void Test();
        void Show();
    }
```

```
class Implementation : Interface1, Interface2
    {
        public void Test()
        {
            Console.WriteLine("Interface Method is
implemented in child class");

        }
        void Interface1.Show()
        {
            Console.WriteLine("Declared in Interface 1");
        }
        void Interface2.Show()
       {
            Console.WriteLine("Declared in Interface 2");
       }
        static void Main()
        {
            Implementation i1 = new Implementation();
               i1.Test();
               Interface1 a = i1;a.Show();
               Interface2 b = i1;b.Show();
               Console.ReadLine();
        }}}
```