

.NET Technology (PS02CDCA34)

Unit – 2 : C# .NET-I

- C#.NET – Introduction and features
- General structure of C#.NET program
- User interface development using Windows Forms
- C#.NET – basic data types, variable, constant
- C#.NET – statements (conditional and looping)
- Type conversion - Boxing and Unboxing

C#.NET – Introduction and features

- C# is pronounced "C-Sharp".
- 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
 - Games
 - Database applications
 - And much, much more!

C#.NET – Introduction and features

C# is object oriented programming language. It provides a lot of **features** that are given below.

- Simple
- Modern programming language
- Object oriented
- Type safe
- Interoperability
- Scalable and Updateable
- Component oriented
- Structured programming language
- Rich Library
- Fast speed

C#.NET – Introduction and features

C# is object oriented programming language. It provides a lot of **features** that are given below.

- Simple

- it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

- Modern programming language

- it is very powerful and simple for building scalable, interoperable and robust applications.

- Object oriented

- OOPs makes development and maintenance easier
- In Procedure-oriented programming language it is not easy to manage if code grows as project size grow.

- Type safe

- C# type safe code can only access the memory location that it has permission to execute. Therefore it improves a security of the program.

- Interoperability

- Interoperability process enables the C# programs to do almost anything that a native C++ application can do.

C#.NET – Introduction and features

C# is object oriented programming language. It provides a lot of **features** that are given below.

- Scalable and Updateable
 - C# is automatic scalable and updateable programming language. For updating our application we delete the old files and update them with new ones.
- Component oriented
 - C# is component oriented programming language. It is the predominant software development methodology used to develop more robust and highly scalable applications.
- Structured programming language
 - We can break the program into parts using functions. So, it is easy to understand and modify.
- Rich Library
 - C# provides a lot of inbuilt functions that makes the development fast.
- Fast speed
 - The compilation and execution time of C# language is fast.

General Structure of a C# Program

- C# programs can consist of one or more files.
- Each file can contain zero or more namespaces.
- A namespace can contain types such as classes, structs, interfaces, enumerations, and delegates, in addition to other namespaces.
- The following is the skeleton of a C# program that contains all of these elements.

General Structure of a C# Program

// A skeleton of a C# program

using System;

namespace YourNamespace

{ class YourClass { }

struct YourStruct { }

interface IYourInterface { }

delegate int YourDelegate();

enum YourEnum { }

namespace YourNestedNamespace

{ struct YourStruct { } }

class YourMainClass

{ static void Main(string[] args) {

//Your program starts here... }

}

}

Namespace

- A **Namespace** in Microsoft .Net is like containers of objects. They may contain *unions, classes, structures, interfaces, enumerators* and *delegates*.
- every namespace can contain any number of classes, functions, variables and also namespaces etc., whose names are unique only inside the namespace.
- To declare namespace C# .Net has a reserved keyword `namespace`. If a new project is created in Visual Studio .NET it automatically adds some global namespaces.
- These namespaces can be different in different projects. But each of them should be placed under the **base namespace System**.
- The names space must be added and used through the using operator. **E.g. using System;**

FCL namespaces

- FCL's outermost namespace is "System"
- FCL technologies nested within System...

Namespace	Purpose	Assembly
System	Core classes, types	mscorlib.dll
System.Collections	Data structures	mscorlib.dll
System.Data	Database access	System.Data.dll
System.Windows.Forms	GUI	System.Windows.Forms.dll
System.XML	XML processing	System.Xml.dll

General Structure of a C# Program

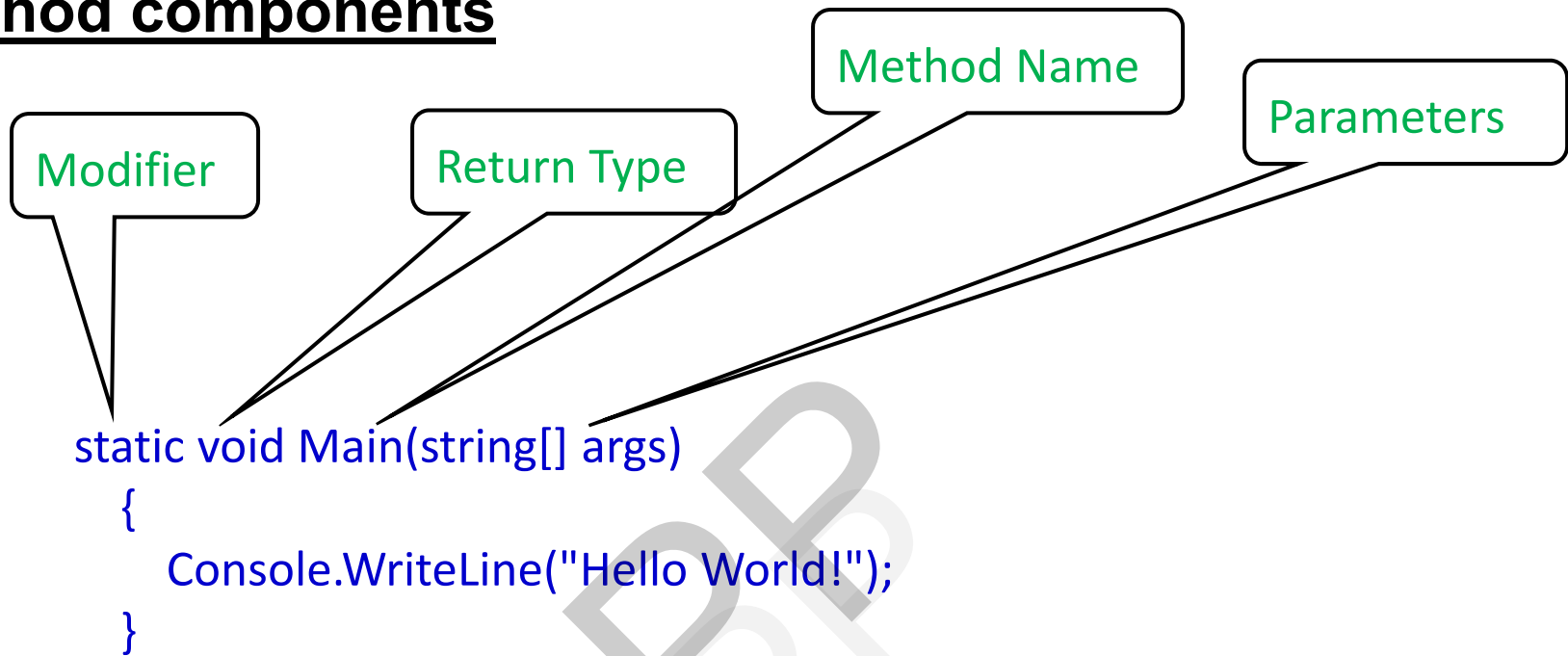
// A skeleton of a C# program

Firstprogram.cs

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

        }
    }
}
```

Method components



General Structure of a C# Program

- Using System means that we can use classes from the System namespace.
- A blank line. C# ignores white space. However, multiple lines makes the code more readable.
- Namespace is a used to organize your code, and it is a container for classes and other namespaces.
- The curly braces {} marks the beginning and the end of a block of code.
- Class is a container for data and methods, which brings functionality to your program. Every line of code that runs in C# must be inside a class. In our example, we named the class Program.

General Structure of a C# Program

- Another thing that always appear in a C# program, is the Main method. Any code inside its curly brackets {} will be executed.
- Console is a class of the System namespace, which has a **WriteLine()** **method** that is used to output/print text. In our example it will output "Hello World!".
- **Console.ReadKey()** **method** to read any key from the console. By entering this line of code, the program will wait and not exit immediately.
- **Every C# statement ends with a semicolon ;.**
- **C# is case-sensitive:** "MyClass" and "myclass" has different meaning.
- Saving the file using a proper name and add **".cs" to the end of the filename.**

WriteLine or Write

- The most common method to output something in C# is WriteLine(), but you can also use Write().
- The difference is that WriteLine() prints the output on a new line each time, while Write() prints on the same line

- **Example**

```
Console.WriteLine("Hello World!");
```

```
Console.WriteLine("I will print on a new line.");
```

```
Console.Write("Hello World! ");
```

```
Console.Write("I will print on the same line.");
```

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

- **Result:**

Hello World!

I will print on a new line.

Hello World! I will print on the same line.

Console.ReadKey() method

- One of the most common uses of the ReadKey() method is to halt program execution until the user presses a key
- To show the output window to the user and close the application when user press any key from the keyboard.

E.g.

```
Console.WriteLine("Hello World!");
```

```
Console.WriteLine("I will print on a new line.");
```

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

C# Comments

Comments can be used to explain code & to make it more readable. It can also be used to prevent execution when testing alternative code.

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

Any text between // and the end of the line is ignored by C# (will not be executed).

- Example**

```
// This is a comment
```

```
Console.WriteLine("Hello World!");
```

- C# Multi-line Comments**

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

Any text between /* and */ will be ignored by C#.

- Example**

```
/* The code below will print the words Hello World  
to the screen, and it is amazing */
```

```
Console.WriteLine("Hello World!");
```


C# Data Types

- A data type specifies the size and type of variable values.
- It is important to use the correct data type for the corresponding variable; to avoid errors, to save time and memory, but it will also make your code more maintainable and readable.
- The most common data types are:

Data Type	Size	Description
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
bool	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter, surrounded by single quotes
string	2 bytes per character	Stores a sequence of characters, surrounded by double quotes

C# Variables

- Variables are containers for storing data values.
- In C#, there are different types of variables (defined with different keywords), for example:
 - int - stores integers (whole numbers), without decimals, such as 123 or -123
 - double - stores floating point numbers, with decimals, such as 19.99 or -19.99
 - char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
 - string - stores text, such as "Hello World". It is surrounded by double quotes
 - bool - stores values with two states: true or false

C# Identifiers

- All C# **variables** must be **identified** with **unique names**.
- These unique names are called **identifiers**.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).
- **The general rules for constructing names for variables (unique identifiers) are:**
 - Names can contain letters, digits and the underscore character (_)
 - Names must begin with a letter
 - Names should start with a lowercase letter and it cannot contain whitespace
 - Names are case sensitive ("myVar" and "myvar" are different variables)
 - Reserved words (like C# keywords, such as int or double) cannot be used as names

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax

type variableName = value;

- Where type is a data type (such as int or string), and
- variableName is the name of the variable.
- The equal sign is used to assign values to the variable.

Example

- Create a variable of type string and assign it the value "John":

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

Example

- You can also declare a variable without assigning the value, and assign the value later

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

Example

- An example of using types in C#
 - declare before you use (compiler enforced)
 - initialize before you use (compiler enforced)

```
public class App
{
    public static void Main()
    {
        declarations → int width, height;
                       width  = 2;
                       height = 4;

        decl + initializer → int area = width * height;

        int x;
        error, x not set → int y = x * 2;
                           ...
    }
}
```

Type conversion

- Some automatic type conversions available
 - from smaller to larger types
- Otherwise you need a *cast* or an explicit *conversion*...
 - typecast syntax is type name inside parentheses
 - conversion based on System.Convert class

implicit conversion →

typecast required →

conversion required →

```
int    i = 5;  
double d = 3.2;  
string s = "496";
```

```
d = i;
```

```
i = (int) d;
```

```
i = System.Convert.ToInt32(s);
```

C# Constants

- if you don't want to overwrite existing values.
- It means unchangeable and read-only.

Example

```
const int myNum = 15;  
myNum = 20; // error
```

The const keyword is useful when you want a variable to always store the same value

An example that is often referred to as a constant, is PI (3.14159...).

Example

```
const float PI 3.14;
```

Get User Input

Console.ReadLine() to get user input.

ReadLine() method returns a string.

Example

// Type your username and press enter

Console.WriteLine("Enter username:");

// Create a string variable & get input from the keyboard & store it in variable.

string userName = **Console.ReadLine();**

// Print the value of the variable (userName), which will display the input value

Console.WriteLine("Username is: " + userName);

Convert Class

- This class is defined **under System namespace**.
- Convert class provides different methods to convert a base data type to another base data type.
- The base types supported by the Convert class are Boolean, Char, SByte, Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime, and String.
- It provides methods that are used to convert integer values to the non-decimal string representation, also convert the string represent the non-decimal numbers to integer values.
- `ToInt16()` Converts a specified value to a 16-bit signed integer.
- `ToInt32()` Converts a specified value to a 32-bit signed integer.
- `ToInt64()` Converts a specified value to a 64-bit signed integer.
- `ToUInt16()` Converts a specified value to a 16-bit unsigned integer.
- `ToDouble()` Converts a specified value to a double-precision floating-point number.
- `ToChar()` Converts a specified value to a Unicode character.
- `DateTime()` Converts a specified value to a DateTime value

Convert Class

To convert any type explicitly use **Convert.To** methods:

Example : Convert string into int.

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

What is the difference between int and System.Int32

- System.Int32 is 32 bit, while int is 64 bit;
- int is 32 bit integer on 32 bit platform, while it is 64 bit on 64 bit platform;
- int is value type, while System.Int32 is reference type;
- int is allocated in the stack, while System.Int32 is allocated in the heap;

sizeof Operator

The **sizeof operator** returns the number of bytes occupied by a variable of a given type.

Expression	Constant value
<code>sizeof(sbyte)</code>	1
<code>sizeof(byte)</code>	1
<code>sizeof(short)</code>	2
<code>sizeof(ushort)</code>	2
<code>sizeof(int)</code>	4
<code>sizeof(uint)</code>	4

sizeof Operator

The **sizeof operator** returns the number of bytes occupied by a variable of a given type.

Expression	Constant value
<code>sizeof(long)</code>	8
<code>sizeof(ulong)</code>	8
<code>sizeof(char)</code>	2
<code>sizeof(float)</code>	4
<code>sizeof(double)</code>	8
<code>sizeof(decimal)</code>	16

Example of sizeof operator

```
Console.WriteLine("Size of int (in Bytes)" + sizeof(int));  
Console.WriteLine("Size of Int (in Bytes)" + sizeof(Int16));  
Console.WriteLine("Size of Int (in Bytes)" + sizeof(Int32));  
Console.WriteLine("Size of Int (in Bytes)" + sizeof(Int64));
```

C# Operators

Operators are used to perform operations on variables and values.

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations:

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	$x \% y$
++	Increment	Increases the value of a variable by 1	$x++$
--	Decrement	Decreases the value of a variable by 1	$x--$

C# Operators

A list of all assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

C# Operators

C# Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

C# Operators

C# Logical Operators

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	<code>x < 5 && x < 10</code>
	Logical or	Returns true if one of the statements is true	<code>x < 5 x < 4</code>
!	Logical not	Reverse the result, returns false if the result is true	<code>!(x < 5 && x < 10)</code>

C#.NET Statements (Conditional)

- You can use these conditions to perform different actions for different decisions.
- **C# has the following conditional statements:**
 - **Use if** to specify a block of code to be executed, if a specified condition is true
 - **Use else** to specify a block of code to be executed, if the same condition is false
 - **Use else if** to specify a new condition to test, if the first condition is false
 - **Use switch** to specify many alternative blocks of code to be executed

C#.NET Statements (Conditional)

The if Statement

Use the if statement to specify a block of C# code to be executed if a condition is True.

Syntax

```
if (condition)
{
    // block of code to be executed if the condition is True
}
```

Example

```
int x = 20;    int y = 18;
if (x > y)
{ Console.WriteLine("x is greater than y"); }
```

C#.NET Statements (Conditional)

The else Statement

Use the else statement to specify a block of code to be executed if the condition is False.

Syntax

```
if (condition)
{ // block of code to be executed if the condition is True }
else
{ // block of code to be executed if the condition is False }
```

Example

```
int x = 20; int y = 18;
if (x > y)
{ Console.WriteLine("x is greater than y"); }
else
{ Console.WriteLine("y is greater than x"); }
```

C#.NET Statements (Conditional)

The else if Statement

Use the else if statement when more than two conditions.

Syntax

if (condition1)

```
{ // block of code to be executed if condition1 is True }
```

else if (condition2)

```
{ // block of code to be executed if the condition1 is false and  
condition2 is True }
```

else

```
{ // block of code to be executed if the condition1 is false and  
condition2 is False }
```

C#.NET Statements (Conditional)

The else if Statement

Example

```
int x = 20;    int y = 18;
```

```
if (x > y)
```

```
{ Console.WriteLine("x is greater than y"); }
```

```
else if ( x < y )
```

```
{ Console.WriteLine("y is greater than x"); }
```

```
else
```

```
{ Console.WriteLine("x and y both have same value."); }
```

C#.NET Statements (Conditional)

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;

```
int x = 20;    int y = 18;
```

```
int result = (x < y) ? x : y;
```

```
Console.WriteLine(result)
```


C#.NET Statements (Conditional)

Switch Statements

Use the switch statement to select one of many code blocks to be executed.

Syntax

```
switch(expression)
{
    case 1:  // code block    break;
    case 2:  // code block    break;
    .....
    default: // code block    break;
}
```

- The switch expression is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- Break statement will stop the execution of case block

C#.NET Statements (Conditional)

Switch Statement

E.g. According to weekday no. display weekday name:

```
int day = 4;
```

```
switch (day)
```

```
{ case 1: Console.WriteLine("Monday"); break;  
  case 2: Console.WriteLine("Tuesday"); break;  
  case 3: Console.WriteLine("Wednesday"); break;  
  case 4: Console.WriteLine("Thursday"); break;  
  case 5: Console.WriteLine("Friday"); break;  
  case 6: Console.WriteLine("Saturday"); break;  
  case 7: Console.WriteLine("Sunday"); break;  
}
```

// Outputs "Thursday" (day 4)

C#.NET Statements (Looping)

Loops

- Loops can execute a block of code as long as a specified condition is reached.
- Loops are handy because they save time, reduce errors, and they make code more readable.

While Loop

The while loop loops through a block of code as long as a specified condition is True:

Syntax

```
while (condition)
{
    // code block to be executed
}
```

C#.NET Statements (Looping)

E.g. the code in the loop will run, over and over again, as long as a variable (i) is less than 5:

```
int i = 0;
while (i < 5)
{
    Console.WriteLine(i);
    i++;
}
```

C#.NET Statements (Looping)

The Do/While Loop

- The do/while loop is a variant of the while loop.
- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do
{
    // code block to be executed
}
while (condition);
```

The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

C#.NET Statements (Looping)

Example

```
int i = 0;  
do  
{  
    Console.WriteLine(i);  
    i++;  
}  
while (i < 5);
```

C#.NET Statements (Looping)

For Loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

Syntax

```
for (statement 1; statement 2; statement 3)
{
    // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

C#.NET Statements (Looping)

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

```
{ int num1, num2, ans;
```

```
    Console.WriteLine("Enter the value of Num1::");
```

```
    String s = Console.ReadLine();
```

```
    num1 = Convert.ToInt32(s);
```

```
    Console.WriteLine("Enter the value of Num2::");
```

```
    s = Console.ReadLine();
```

```
    num2 = Convert.ToInt32(s);
```

```
    Console.WriteLine("Num1 Value is ::" + num1);
```

```
    Console.WriteLine("Num2 Value is ::" + num2);
```

```
    ans = num1 + num2;
```

```
    Console.WriteLine("Addition is ::" + ans);
```

```
}
```


User interface development using Windows Forms

//C# Program to Perform Celsius to Fahrenheit Conversion

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace program {
class Program {
    static void Main(string[] args) {
        int celsius, faren;
        Console.WriteLine("Enter the Temperature in Celsius(°C) : ");
        celsius = int.Parse(Console.ReadLine());
        faren = (celsius * 9) / 5 + 32;
        Console.WriteLine("Temperature in Fahrenheit is(°F) : " + faren);
        Console.ReadLine();    } } }
```

C#.NET Statements (Looping)

For Loop

E.g. To print the numbers 0 to 4:

```
for (int i = 0; i < 5; i++)  
{  
    Console.WriteLine(i);  
}
```

Break Statement

It is used to "jump out" of a switch statement or of a loop.

E.g. jumps out of the loop when i is equal to 4:

```
for (int i = 0; i < 10; i++)  
{  
    if (i == 4)  
    {  
        break;  
    }  
    Console.WriteLine(i);  
}
```

Continue Statement

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

E.g. skips the value of 4 but print the values of i as 1 to 3 & 5 to 10.

```
for (int i = 0; i < 10; i++)  
{  
    if (i == 4)  
    { continue; }  
  
    Console.WriteLine(i);  
}
```

Value vs. Reference types

- **C# separates data types into two categories**
- **Value types:**
 - variable represents a value ("bits")



- **Reference types:**
 - variable represents a reference to a heap-based object
 - actual data resides in the object



Type conversion - Boxing and Unboxing

Boxing

- The process of Converting a Value Type (char, int etc.) to a Reference Type(object) is called Boxing.
- It is implicit conversion process.

Example :

```
int num = 23;           // 23 will assigned to num  
Object Obj = num;       // Boxing
```

Type conversion - Boxing and Unboxing

Unboxing

- The process of converting reference type into the value type is known as Unboxing.
- It is explicit conversion process.

Example :

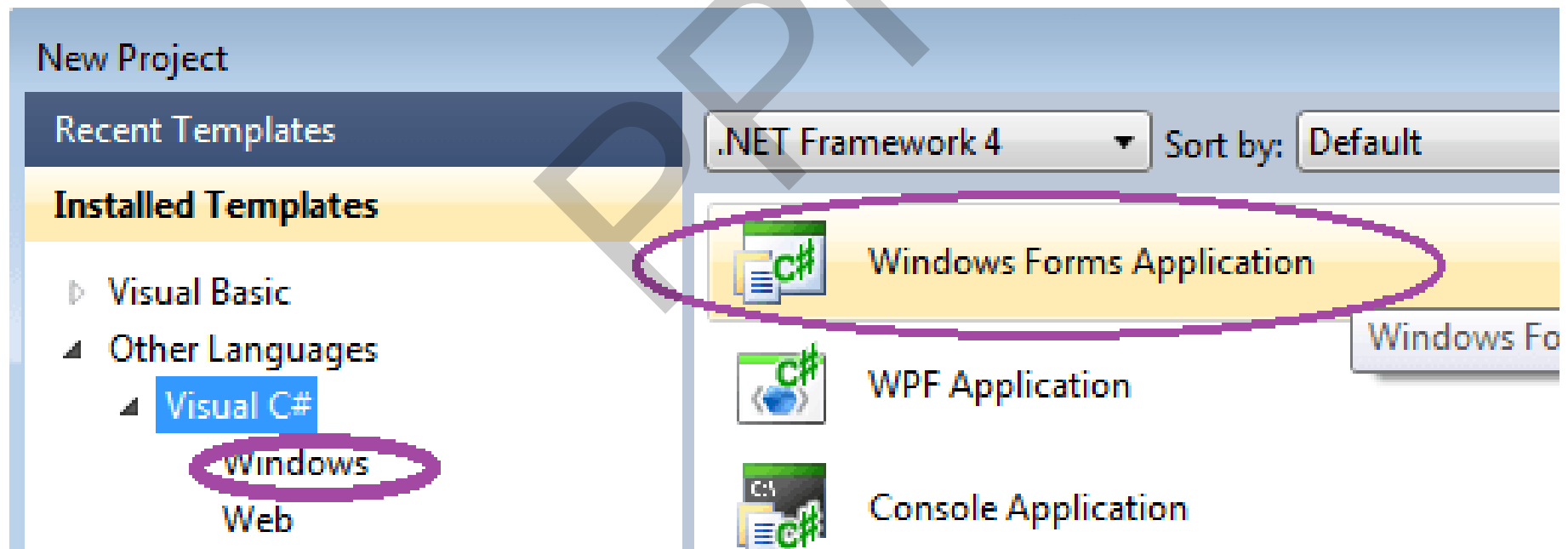
```
int num = 23;           // value type is int and assigned value 23
Object Obj = num;       // Boxing
int i = (int)Obj;       // Unboxing
```

User interface development using Windows Forms

- **C#** has all the features of any powerful, modern language. In C#, the most rapid and convenient way to create your user interface is to do so visually, using the **Windows Forms Designer** and **Toolbox**.
- Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client side Windows based applications.
- A **control** is a component on a form used to display information or accept user input.
- The **Control class** provides the base functionality for all controls that are displayed on a Form.

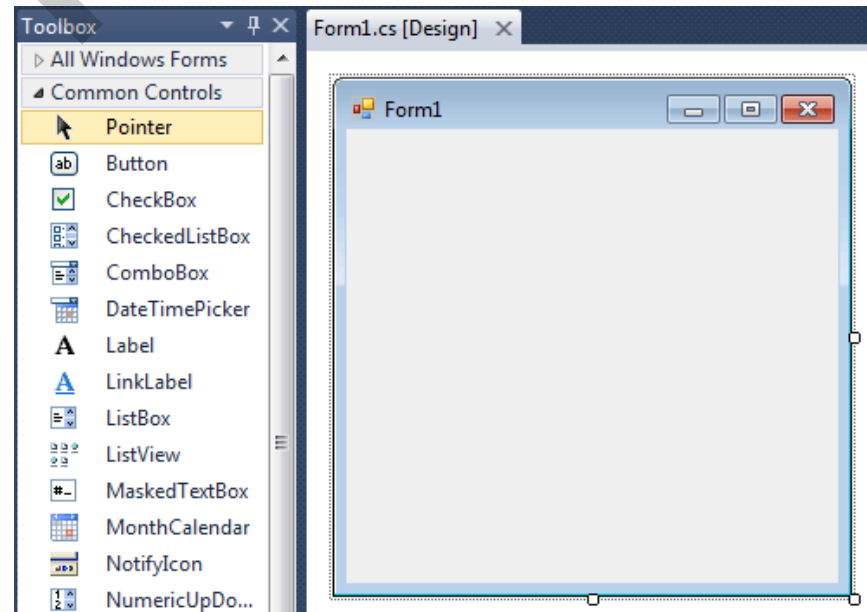
User interface development using Windows Forms

- The first step is to start a new project and build a form.
- Open your Visual Studio and select File->New Project and from the new project dialog box select Other Languages->Visual C# and select Windows Forms Application.
- Enter a project name at the bottom of the dialogue box and click OK button.



User interface development using Windows Forms

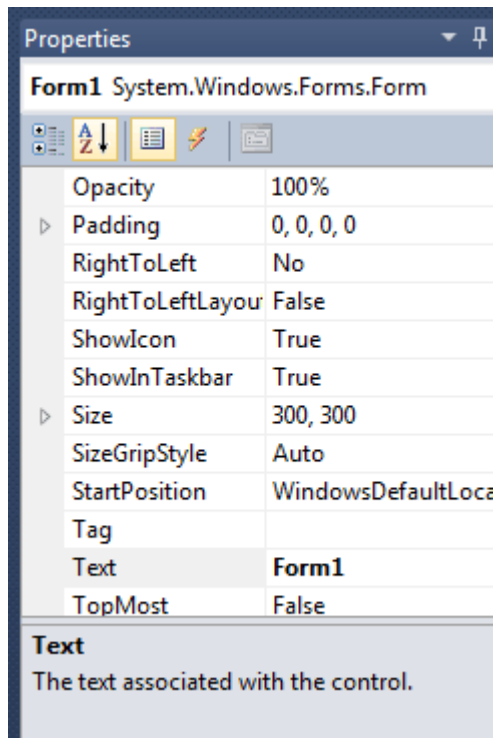
- After selecting Windows Forms Application , you can see a default Form (Form1) in your new C# project.
- The Windows Form you see in Designer view is a visual representation of the window that will open when your application is opened.
- You can switch between this view and Code view at any time by right-clicking the design surface or code window and then clicking View Code or View Designer.
- The image shows the default Form (Form1) looks like.



User interface development using Windows Forms

- At the top of the form there is a title bar which displays the forms title. Form1 is the default name, and you can change the name to your convenience .
- The title bar also includes the control box, which holds the minimize, maximize, and close buttons.
- If you want to set any properties of the Form, you can use Visual Studio Property window to change it.
- If you do not see the Properties window, on the **View menu**, **click Properties window**.
- This window lists the properties of the currently selected Windows Form or control, and its here that you can change the existing values.

User interface development using Windows Forms



For example , to change the forms title from Form1 to MyForm, click on Form1 and move to the right side down Properties window, set **Text property** to MyForm. Then you can see the Title of the form is changed.

Likewise you can set any properties of Form through Properties window.

The Toolbox

- You can view Toolbox from the View menu, or by **pressing Ctrl-Alt-X**.
- The toolbox contains a selection of all the controls available to you as a .NET developer.



The Toolbox

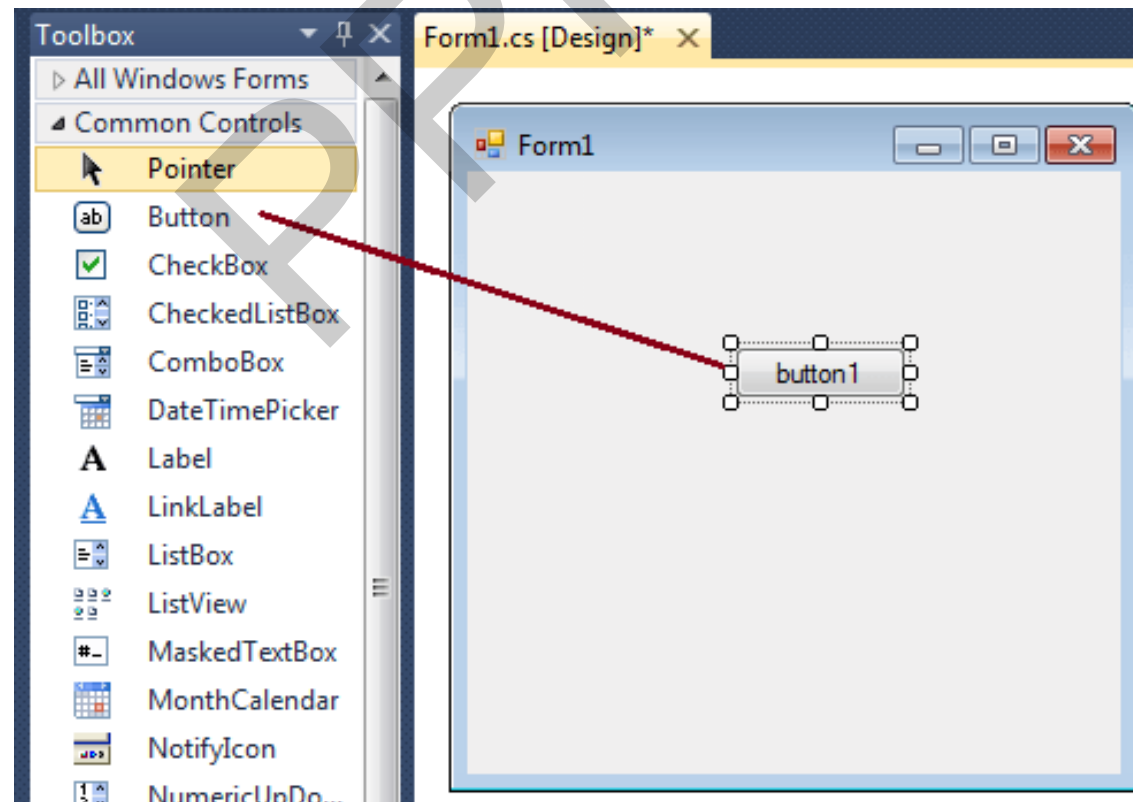
- You can view Toolbox from the View menu, or by pressing Ctrl-Alt-X.
- The toolbox contains a selection of all the controls available to you as a .NET developer.
- Most controls in .NET derive from the **System.Windows.Forms.Control** class.
- This class defines the basic functionality of the controls, which is why many properties and events in the controls we'll see are identical.
- **Properties**
- All controls have a number of properties that are used to manipulate the behavior of the control.
- The **base class of most controls, Control**, has a number of properties that other controls either inherit directly or override **to provide some kind of custom behavior**.

Windows Applications

C# Windows Forms

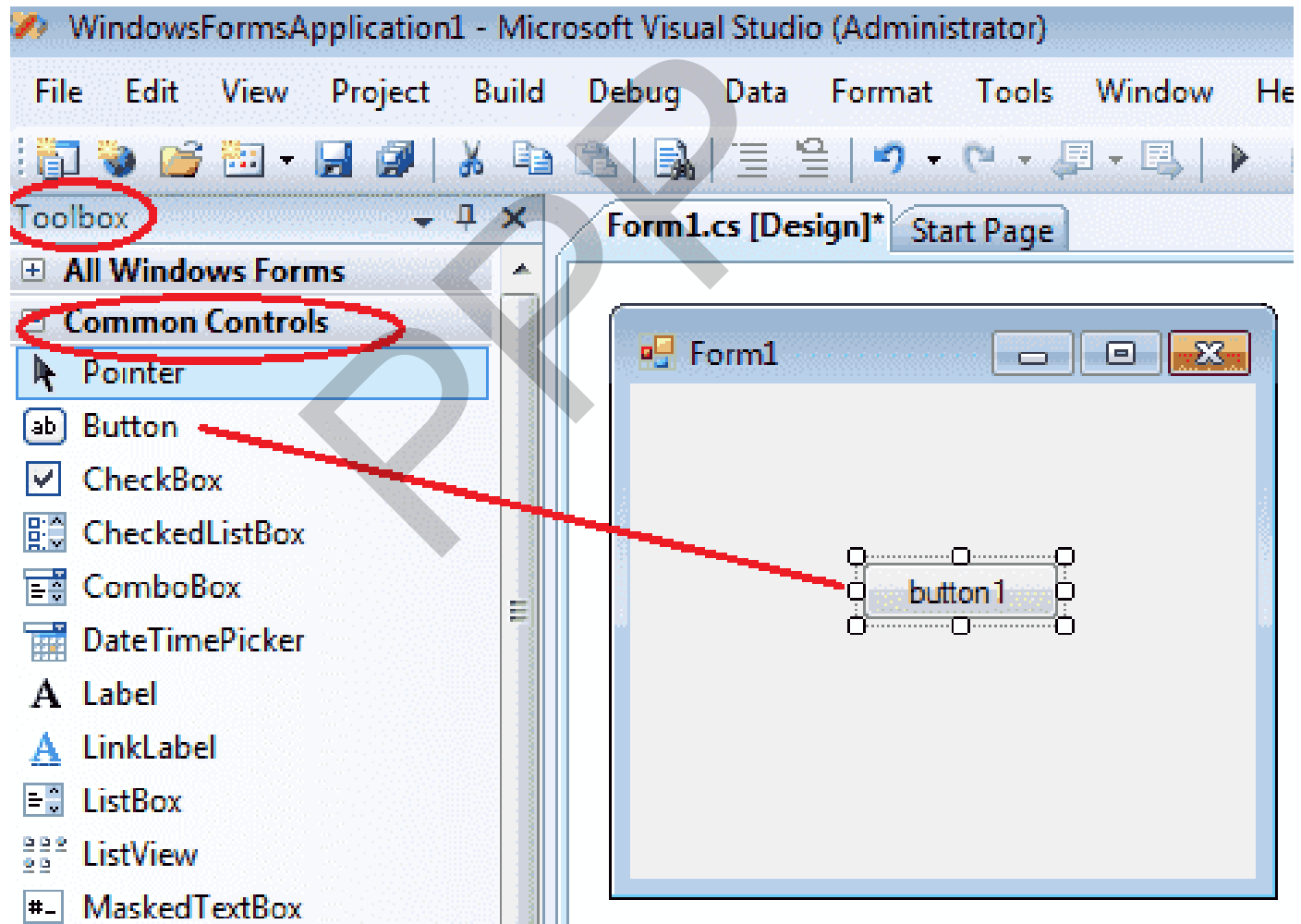
C# programmers have made extensive use of forms to build user interfaces.

Each time you create a Windows application, Visual Studio will display a default blank form, onto which you can drag the controls onto your applications main form and adjust their size and position.

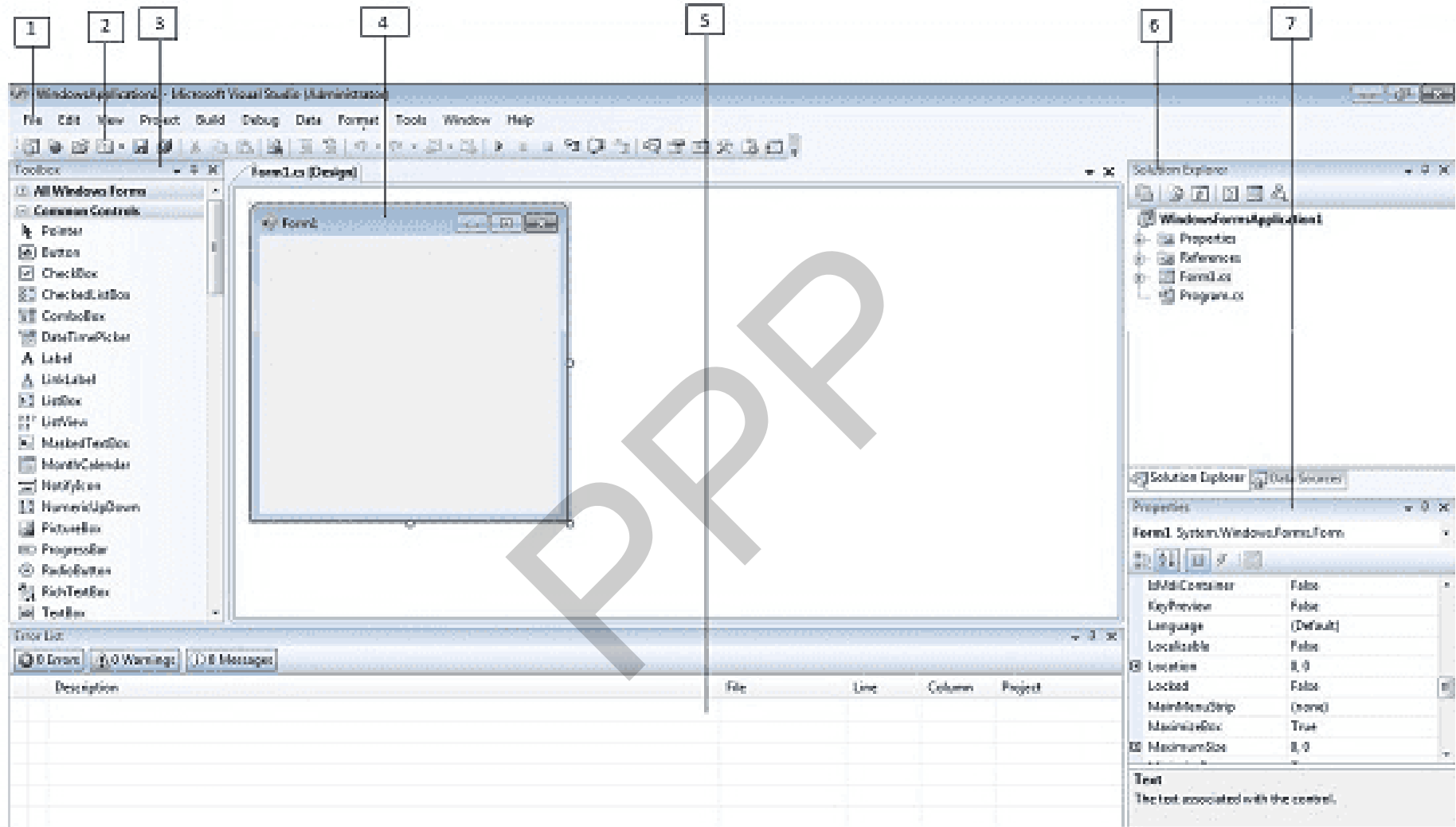


Windows Applications

- You can select basic controls from **Common Controls** group.
- You can place the control in your Form by drag and drop the control from your toolbox to Form control.



Most of the tools in visual studio



1. Menu Bar

2. Standard Toolbar

3. ToolBox

4. Forms Designer

5. Output Window

6. Solution Explorer

7. Properties Window

Common Properties of Controls

The table below shows some of the most **common properties of the Control class**. These properties will be present in most of the controls

Name	Availability	Description
BackColor	Read/Write	The background color of a control.
Enabled	Read/Write	Setting Enabled to true usually means that the control can receive input from the user. Setting Enabled to false usually means that it cannot.
ForeColor	Read/Write	The foreground color of the control.
Name	Read/Write	The name of the control. This name can be used to reference the control in code.

Common Properties of Controls

The table below shows some of the most **common properties of the Control class**. These properties will be present in most of the controls

Name	Availability	Description
TabIndex	Read/Write	The number the control has in the tab order of its container.
TabStop	Read/Write	Specifies whether the control can be accessed by the Tab key.
Visible	Read/Write	Specifies whether or not the control is visible at runtime.
Width	Read/Write	The width of the control.

Common Controls

Label Control

- Labels are one of the most frequently used C# control.
- We can use the Label control to display text in a set location on the page.
- Label controls can also be used to add descriptive text to a Form to provide the user with helpful information.
- **The Label class is defined in the System.Windows.Forms namespace.**
- To change the display text of the Label, you have to set a new text to the **Text property of Label**. E.g. This is my first Lable.



Common Controls

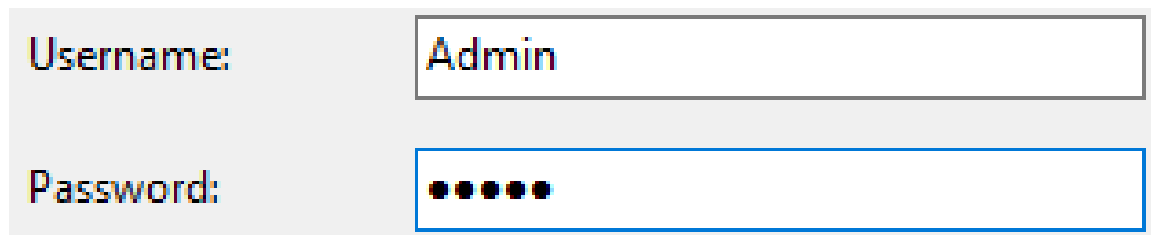
Label Control

- To specify desired border set **BorderStyle** property.
- For alignment of text use **TextAlign** Property.
- To change the display text of the Label
label1.Text = "Hello World";

Common Controls

TextBox Control

- A TextBox control is used to display, or accept as input, a single line of text.
- This control has additional functionality of multiline editing and password character masking.
- **PasswordChar property** to mask characters entered in a single line version of the control.
- **Multiline and ScrollBars properties** to enable multiple lines of text to be displayed or entered.
- A text box object is used to display text on a form or to get user input while a C# program is running.
- In a text box, a user can type data or paste it into the control from the clipboard.



Username:	<input type="text" value="Admin"/>
Password:	<input type="password" value="•••••"/>

Common Controls

TextBox Control

- Common properties for textbox are
 - Width
 - Height
 - BackColor
 - BorderStyle
 - MaxLength
 - ForeColor etc.
- To change the value of textbox. E.g.
`textbox1.Text = "New Value";`
- To retrieve value of textbox E.g.
`s = textbox1.Text;`
- To retrieve integer values from textbox E.g.
`int i; i = int.Parse (textBox1.Text);`

Common Controls

Button Control

- Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client side Windows applications.
- A button is a control, which is an interactive component that enables users to communicate with an application.
- **The Button class inherits from the ButtonBase class.**
- A Button can be clicked by using the mouse, ENTER key, or SPACEBAR if the button has focus.



Common Controls

Button Control

- The Click event is raised when the Button control is clicked. Raising an event invokes the event handler through a delegate.
- Click event Example

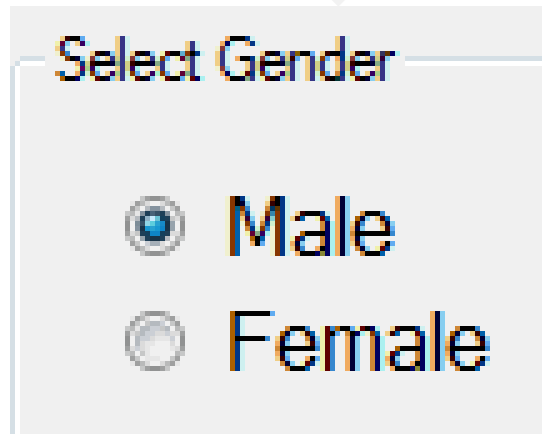
```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Button is clicked");
}
```

Common Controls

RadioButton Control

- A radio button or option button enables the user to select a single option from a group of choices when paired with other RadioButton controls.
- When a user clicks on a radio button, it becomes checked, and all other radio buttons with same group become unchecked.
- Use a radio button when you want the user **to choose only one option**.
- **Checked property** indicates the radio button is selected or not.

radioButton1.**Checked** = true;



Select Gender

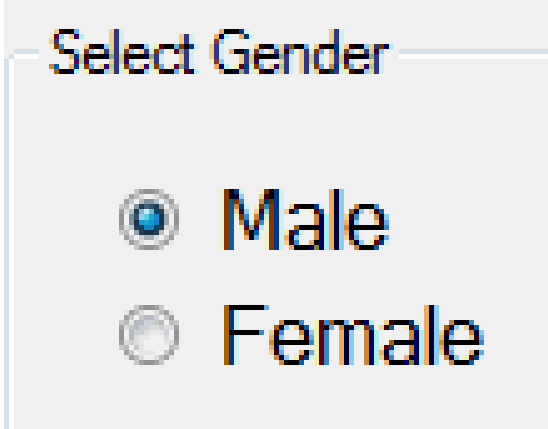
☒ Male

☐ Female

Common Controls

RadioButton Control

```
if (radioButton1.Checked == true)
{
    MessageBox.Show ("You have select Male");
    return;
}
else
{
    MessageBox.Show(" You have select Female ");
    return;
}
```



Select Gender

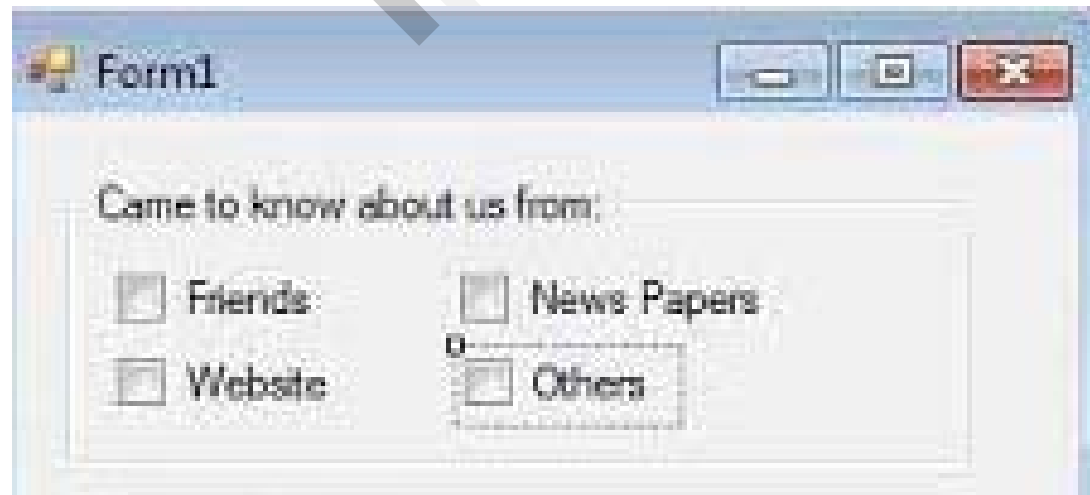
☒ Male

☐ Female

Common Controls

CheckBox Control

- CheckBoxes allow the user to make multiple selections from a number of options.
- CheckBox to give the user an option, such as true/false or yes/no.
- You can click a check box to select it and click it again to deselect it.
- When you want the user **to choose all appropriate options, use a check box.**



Common Controls

CheckBox Control : Example

```
string msg = "";  
if (checkBox1.Checked == true)  
    msg = msg + "Friends";  
if (checkBox2.Checked == true)  
    msg = msg + "News Papers";  
if (checkBox3.Checked == true)  
    msg = msg + "Website";  
if (checkBox4.Checked == true)  
    msg = msg + "Others";
```

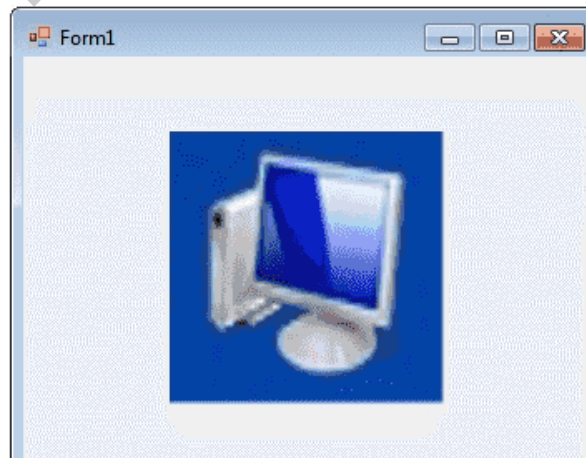
```
MessageBox.Show(msg);
```



Common Controls

PictureBox Control

- The Windows Forms PictureBox control is used to display images in bitmap, GIF , icon , or JPEG formats.
- You can set the **Image property** to the Image you want to display, either at design time or at run time.
- There are three different **PictureBoxSizeMode** is available to PictureBox control.
 - AutoSize - Sizes the picture box to the image.
 - CenterImage - Centers the image in the picture box.
 - Normal - Places the upper-left corner of the image at upper left in the picture box



Common Controls

ComboBox Control

- A ComboBox displays a text box combined with a ListBox, which enables the user to select items from the list or enter a new value.
- To set a default value for a Combo Box **SelectedIndex** property
- By default, **DropDownStyle** property of a Combobox is DropDown. In this case user can enter values to combobox.
- To make a ComboBox **readonly**, that means a user cannot write in a combo box but he can select the given items, in **two ways**.
 1. Change the **DropDownStyle** property to **DropDownList**,
 2. set **comboBox1.Enabled = false**.



Common Controls

ComboBox Control

- **To set a default value for a Combo Box**
`comboBox1.SelectedIndex = 6;`
- **To add more items to combobox**
`comboBox1.Items.Add("Sunday");`
`comboBox1.Items.Add("Monday");`
- **To retrieve value from ComboBox**
`string var;`
`var = comboBox1.Text;`
- **To remove an item from ComboBox at a the specified index or giving a specified item by name.**
`comboBox1.Items.RemoveAt(1);`
`comboBox1.Items.Remove("Friday");`

Common Controls

ListBox Control

- The **ListBox control** enables you to display a list of items to the user that the user can select by clicking.
- To add the values in List select the **Items property** and add multiple values for list.
- **To retrieve a single selected item** to a variable , use the following code.

```
string item= listBox1.Text;
```

OR

```
string item = listBox1.SelectedItem.ToString();
```

OR

```
string item = listBox1.GetItemText(listBox1.SelectedItem);
```

- **To add more items in existing list**

```
listBox1.Items.Add("Sunday");
```

- A ListBox control can provide **single or multiple selections** using the **SelectionMode property** .



Difference between Combobox and Listbox Control

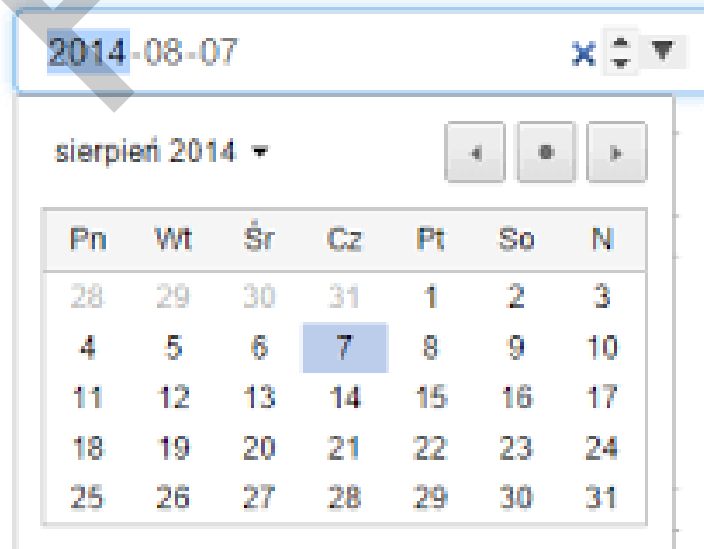
- A combo box is appropriate when there is a list of suggested choices
- A list box is appropriate when you want to limit input to what is on the list.
- In combo box multiple select is not possible
- In list box we can select multiple items.
- We can't use checkboxes within combo boxes
- We can use checkboxes with in the list box.
- Combo boxes save space on a form. Because the full list is not displayed until the user clicks the down arrow, a combo box can easily fit in a small space
- List box would not fit.

Common Controls

DateTimePicker Control

- The DateTimePicker control allows you to display and collect date and time from the user with a specified format.
- The DateTimePicker control has two parts, a label that displays the selected date and a popup calendar that allows users to select a new date.
- The most important property of the DateTimePicker is the **Value** property, which holds the selected date and time.
- **dateTimePicker1.Value = DateTime.Today;**

Date of Birth



The screenshot shows a Windows-style DateTimePicker control. The main text box displays the date "2014-08-07" in a blue selection box. To the right of the text box are three small icons: a cross (clear), a double-headed arrow (toggle), and a downward arrow (calendar). Below the text box, a calendar popup is visible. The calendar header shows "sierpień 2014" (August 2014) and three navigation buttons (back, current, forward). The calendar grid shows days of the week (Pn, Wt, Śr, Cz, Pt, So, N) and dates. The date "7" is highlighted in blue, indicating it is the selected date.

Pn	Wt	Śr	Cz	Pt	So	N
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Common Controls

DateTimePicker Control

- To get the date

DateTime iDate;

iDate = dateTimePicker1.Value;

The values can be displayed in **four formats**, which are set by the Format property: **Long, Short, Time, or Custom.**

dateTimePicker1.Format = DateTimePickerFormat.Short;

- **Converts** the specified **string** representation of a date and time **to** an equivalent **date and time value**

string iDate = "05/05/2005";

DateTime oDate = Convert.ToDateTime(iDate);

**MessageBox.Show(oDate.Day + " " + oDate.Month + " " +
oDate.Year);**