

Module 1 - Programming Paradigms

Paradigm vs. Programming Paradigm

Paradigm can also be termed as method to solve some problem or do some tasks. Programming paradigm is an approach to solve problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach.

A programming paradigm is a style, or “way,” of programming.

Some Common Programming Paradigms:

1. Imperative

Programming with an explicit sequence of commands that update state. Imperative programming (from Latin imperare = command) is the oldest programming paradigm. A program based on this paradigm is made up of a clearly-defined sequence of instructions to a computer. Therefore, the source code for imperative languages is a series of commands, which specify what the computer has to do – and when – in order to achieve a desired result. Values used in variables are changed at program runtime. To control the commands, control structures such as loops or branches are integrated into the code. Imperative programming languages are very specific, and operation is system-oriented. On the one hand, the code is easy to understand; on the other hand, many lines of source text are required to describe what can be achieved with a fraction of the commands using declarative programming languages.

These are the best-known imperative programming languages:

- Fortran
- Java
- Pascal

- ALGOL
- C
- C#
- C++
- Assembler
- BASIC
- COBOL
- Python
- Ruby

The different imperative programming languages can, in turn, be assigned to three further subordinate programming styles – structured, procedural, and modular.

The **structured programming** style extends the basic imperative principle with specific control structures: sequences, selection, and iteration. This approach is based on a desire to limit or completely avoid jump statements that make imperatively designed code unnecessarily complicated.

The **procedural approach** divides the task a program is supposed to perform into smaller sub-tasks, which are individually described in the code. This results in programming modules which can also be used in other programs. The modular programming model goes one step further by designing,

developing, and testing the individual program components independently of one another. The individual modules are then combined to create the actual software.

Imperative programming example

Imperative programming languages are characterized by their instructive nature and, thus, require significantly more lines of code to express what can be described with just a few instructions in the declarative style

Advantages & Disadvantages

Advantages

- Easy to read
- Relatively easy to learn
- Conceptual model (solution path) is very easy to beginners to understand
- Characteristics of specific application can be taken into account

Disadvantages

- Code quickly becomes very extensive and thus confusing
- Higher risk of errors when editing
- System-oriented programming means that maintenance block application development
- Optimization and extension is more difficult

2. Declarative

Programming by specifying the result you want, not how to get it.

There is no one specific definition of the paradigm, but all definitions agree on one thing: A characteristic feature of declarative programming languages is that they always describe the desired end result rather than outlining all the intermediate work steps. In declarative programming, the solution path to reach the goal is determined automatically. This works well, provided the specifications of the final state are clearly defined and an appropriate implementation procedure exists. If both of these conditions are met, declarative programming is very efficient.

Since **declarative programming** does not specifically describe the “how” but works at a very high level of abstraction, the programming paradigm also leaves room for optimization. If a better implementation procedure is developed, the integrated algorithm can identify and use it. This makes the paradigm futureproof. The procedure for how the result is to be achieved does not have to be set in stone when writing the code.

The best-known declarative programming languages are:

- **Prolog**
- **Lisp**
- **Haskell**
- **Miranda**
- **Erlang**
- **SQL (in the broadest sense)**

The different declarative programming languages can, in turn, be divided into two paradigms: **functional programming** languages and **logic programming** languages.

However, in practice, the boundaries are frequently blurred and elements of both imperative programming – with its sub-type's procedural, modular, and structured programming – and declarative programming are used to solve problems.

Advantages

- Short, efficient code
- Can be implemented using methods not yet known at the time of programming
- Easy optimization as implementation is controlled by an algorithm
- Maintenance possible independent of application development

Disadvantages

- Sometimes hard to understand for external people
- Based on an unfamiliar conceptual model for people (solution state)
-
- Hard to take characteristics of individual applications into account during programming.

Imperative Vs Declarative Programming

Imperative programming languages differ from declarative languages on one fundamental point: imperative programming focuses on the “how”, declarative programming on the “what”.

But what does that mean? **Imperative programming languages** are composed of step-by-step instructions (how) for the computer. They describe explicitly which steps are to be performed in what order to obtain the desired solution at the end. By contrast, in declarative programming, the desired result (what)

is described directly. This becomes clearer when using a cooking analogy for illustration: imperative languages provide recipes; declarative languages contribute photos of the finished meal.

In **declarative languages**, the source code remains very abstract in terms of the specific procedure. To get to the solution, an algorithm is used which automatically identifies and applies appropriate methods. This approach has numerous advantages: Programs can be written much more quickly, and applications are also very easy to optimize. If a new method is developed in the future, the abstract instructions in the source code mean that the algorithm can easily utilize the newer method.

3. Structured (modular programming)

Programming with clean, go to-free, nested control structures. Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Facilitates the creation of programs with readable code and reusable components. Where modules or elements of code can be reused from a library, it may also be possible to build structured code using modules written in different languages, as long as they can obey a common module interface or application program interface (API) specification.

Structured programming encourages dividing an application program into a hierarchy of modules or autonomous elements, which may, in turn, contain other such elements. Within each element, code may be further structured using blocks of related logic designed to improve readability and maintainability. These may include case, which tests a variable against a set of values; Repeat, while and for, which construct loops that continue until a condition is met. In all structured programming languages,

an unconditional transfer of control, or go to statement, is deprecated and sometimes not even available

4. Procedural

Imperative programming with procedure calls.

Procedural Programming may be the first programming paradigm that a new developer will learn. Fundamentally, the procedural code is the one that directly instructs a device on how to finish a task in logical steps. This paradigm uses a linear top-down approach and treats data and procedures as two different entities. Based on the concept of a procedure call, Procedural Programming divides the program into procedures, which are also known as routines or functions, simply containing a series of steps to be carried out.

Simply put, Procedural Programming involves writing down a list of instructions to tell the computer what it should do step-by-step to finish the task at hand.

Advantages

- Procedural Programming is excellent for general-purpose programming
- The coded simplicity along with ease of implementation of compilers and interpreters • A large variety of books and online course material available on tested algorithms, making it easier to learn along the way
- The source code is portable, therefore, it can be used to target a different CPU as well • The code can be reused in different parts of the program, without the need to copy it
- Through Procedural Programming technique, the memory requirement also slashes • The program flow can be tracked easily

Disadvantages

- The program code is harder to write when Procedural Programming is employed
- The Procedural code is often not reusable, which may pose the need to recreate the code if it is needed to use in another application
- Difficult to relate with real-world objects
- The importance is given to the operation rather than the data, which might pose issues in some data-sensitive cases
- The data is exposed to the whole program, making it not so much security friendly.

5. Functional

Programming with function calls that avoid any global state.

Functional Programming is the use of pure high-order functions to develop applications. This involves a focus on creating code that avoids changing state and mutating data.

6. Object-Oriented

Programming by defining objects that send messages to each other. Objects have their own internal (encapsulated) state and public interfaces.

Object orientation can be:

- a. **Class-based:** Objects get state and behavior based on membership in a class
- b. **Prototype-based:** Objects get behavior from a prototype object.

7. Event-Driven Programming

Programming with emitters and listeners of asynchronous actions.

In computer programming, event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs (embedded), or message passing from other programs or threads (e.g., using Application Programming Interface or API).

Event-driven programs can be written in any programming language, although some languages (Visual Basic for example) are specifically designed to facilitate event-driven programming, and provide an integrated development environment (IDE) that partially automates the production of code, and provides a comprehensive selection of built-in objects and controls, each of which can respond to a range of events. Virtually all object-oriented and visual languages support event-driven programming. Visual Basic, Visual C++ and Java are examples of such languages.

A visual programming IDE such as VB.Net provides much of the code for detecting events automatically when a new application is created. The programmer can therefore concentrate on issues such as interface design, which involves adding controls such as command buttons, text boxes, and labels to standard forms (a form represents an application's workspace or window). Once the user interface is substantially complete, the programmer can add event-handling code to each control as required.

The change in emphasis from procedural to event-driven programming has been accelerated by the introduction of the Graphical User Interface (GUI) which has been widely adopted for use in operating systems and end-user applications. It really began, however, with the introduction of

object-oriented (OO) programming languages and development methodologies in the late 1970s. By the 1990's, object-oriented technologies had largely supplanted the procedural programming languages and structured development methods that were

popular during the 70s and 80s.

The link between object-oriented programming and event-driven programming is fairly obvious. For example, objects on a Visual Basic form (usually referred to as controls) can be categorised into classes (e.g. "Button", "TextBox" etc.), and many instances of each can appear on a single form. Each class will have attributes (usually referred to as properties) that will be common to all objects of that type (e.g. "BackgroundColour", "Width" etc.), and each class will define a list of events to which an object of that type will respond. The methods (event-handlers) to handle specific events are usually provided as templates to which the programmer simply has to add the code that carries out the required action.

Top Programming Language Worldwide in 2021 according to PYPL.

PYPL Index: The PYPL PopularitY of Programming Language Index is created by analyzing how often language tutorials are searched on Google. The index is updated once a month.

MODULE 2 – BASIC C# PROGRAMMING AND SYNTAX

What is .NET Framework

.NET is a framework to develop software applications. It is designed and developed by Microsoft and the first beta version released in 2000. It is used to develop applications for web,

Windows, phone. Moreover, it provides a broad range of functionalities and support.

This framework contains a large number of class libraries known as Framework Class Library (FCL). The software programs written in .NET are executed in the execution environment, which is called CLR (Common Language Runtime). These are the core and essential parts of the .NET framework. This framework provides various services like memory management, networking, security, memory management, and type-safety.

The .Net Framework supports more than 60 programming languages such as C#, F#, VB.NET, J#, VC++, JScript.NET, APL, COBOL, Perl, Oberon, ML, Pascal, Eiffel, Smalltalk, Python, Cobra, ADA, etc.

What is C#

C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework

By the help of C# programming language, we can develop different types of secured and robust applications:

- Window applications
- Web applications
- Distributed applications
- Web service applications
- Database applications etc



Anders Hejlsberg is known as the founder of C# language. It is based on C++ and Java, but it has many additional extensions used to perform component-oriented programming approach.

C# has evolved much since their first release in the year 2002. It was introduced with .NET Framework 1.0 and the current version of C# is 5.0.

Description

- class: is a keyword which is used to define class.
- Program: is the class name. A class is a blueprint or template from which objects are created. It can have data members and methods. Here, it has only Main method.
- static: is a keyword which means object is not required to access static members. So it saves memory.
- void: is the return type of the method. It doesn't return any value. In such case, return statement is not required.
- Main: is the method name. It is the entry point for any C# program. Whenever we run the C# program, Main() method is invoked first before any other method. It represents start up of the program.

- `string[] args`: is used for command line arguments in C#. While running the C# program, we can pass values. These values are known as arguments which we can use in the program.
- `System.Console.WriteLine("Hello World!")`: Here, `System` is the namespace. `Console` is the class defined in `System` namespace. The `WriteLine()` is the static method of `Console` class which is used to write the text on the console.

C# Example: Using System

If we write `using System` before the class, it means we don't need to specify `System` namespace for accessing any class of this namespace. Here, we are using `Console` class without specifying `System.Console`.

C# Variable

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times. It is a way to represent memory location through symbol so that it can be easily identified.

Rules for defining variables

- A variable can have alphabets, digits and underscore.
- A variable name can start with alphabet and underscore only. It can't start with digit.
- No white space is allowed within variable name.
- A variable name must not be any reserved word or keyword belong to C# e.g. `char`, `float`, `continue`, `checked`, `public`, `return`, etc;.
- C# is case sensitive, it means that `var1 != Var1`. (!= means different).

C# Data Types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.

There are 3 types of data types in C# language.

Value Data Type – short, int, char, float, double, etc

Reference Data Type – String, Class, Object and Interface

Pointer Data Type – Pointers

Unlike humans, a computer does not know the difference between "1234" and "abcd." A data type is a classification that dictates what a variable or object can hold in computer programming. Data types are an important factor in virtually all computer programming languages, including C#, C++, JavaScript, and Visual Basic. When programmers create computer applications, both desktop and web-based, data types must be referenced and used correctly to ensure the proper result and an error-free program.

Common examples of data types:

- Boolean (e.g., `True` or `False`)
- Character (e.g., `a`)
- Date (e.g., `03/01/2016`)
- Double (e.g., `1.79769313486232E308`)
- Floating-point number (e.g., `1.234`)
- Integer (e.g., `1234`)
- Long (e.g., `123456789`)

- Short (e.g., 0)
- String (e.g., abcd)
- Void (e.g., no data)

Depending on the programming language, there may also be many more data types that serve a specific function and store data in a particular way. Understanding the different data types allows programmers to design computer applications more efficiently and accurately.

C# Operators

An operator is simply a symbol that is used to perform operations.

These are the common types of operators to perform different types of operations in C# language.

- Arithmetic Operators
- Relational Operators or Comparison
- Logical Operators
- Assignment Operators
- Unary Operators
- Ternary Operators

C# Keywords

A keyword is a reserved word. You cannot use it as a variable name, constant name etc.

Some identifiers which have special meaning in context of code are called as Contextual Keywords.

String methods

In C#, string is an object of System.String class that represent sequence of characters. We can perform many operations on strings such as concatenation, comparison, getting substring, search, trim, replacement etc.

String functions are used in computer programming languages to manipulate a string or query information about a string (some do both).

Below we have a few methods that we can use with the String class. Some of these methods, we are going to write the code for them using the structure conditional 'IF'.

- **Compare:** This method compares two strings if they are equals or not and return 0 if it is true, else 1 if it is false.
- **Concat:** This method concatenates two strings and return as results both strings together.
- **Contains:** This method verifies if there is a(some) word(s) in the current string I passed.
- **Replace:** This method substitutes a string to another.
- **ToUpper:** This method gets the string I am calling this method from and return the same string but now in uppercase.
- **ToLower:** This method gets the string I am calling this method from and return the same string but now in lowercase

- **Trim:** Removes the space of a string.

MODULE 3 – CONTROL STATEMENTS & LOOPING

Control Flow

In programming, control flow is the order in which statements and instructions are executed. Programmers are able to change a program's control flow using control structures such as conditionals.

Within an imperative programming language, a control flow statement is a statement that results in a choice being made as to which of two or more paths to follow.

In computer programming, control flow or flow of control is the order function calls, instructions, and statements are executed or evaluated when a program is running. Many programming languages have what are called control flow statements, which determine what section of code is run in a program at any time.

Control Flows Examples

C# Conditions and If Statements

C# supports the usual logical conditions from mathematics:

- Less than: $a < b$
- Less than or equal to: $a \leq b$
- Greater than: $a > b$ • Greater than or equal to: $a \geq b$
- Equal to $a == b$
- Not Equal to: $a != b$

The if Statement

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

An if statement is a programming conditional statement that, if proved true, performs a function or displays information.

The IF statement is a decision-making statement that guides a program to make decisions based on specified criteria. The IF statement executes one set of code if a specified condition is met (TRUE) or another set of code evaluates to FALSE.

In C#, an if statement executes a block of code based on whether or not the boolean expression provided in the parentheses is true or false. If the expression is true then the block of code inside the braces, {}, is executed. Otherwise, the block is skipped over.

The if/else Statement

If the first condition is not met, use the else statement to specify a block of code to be executed.

The if/else statement extends the if statement by specifying an action if the if (true/false expression) is false.

An else followed by braces, {}, containing a code block, is called an else clause. else clauses must always be preceded by an if statement.

The block inside the braces will only run if the expression in the accompanying if condition is false. It is useful for writing code that runs only if the code inside the if statement is not executed.

The if/else if Statement

The if/else if statement allows you to create a chain of if statements. The if statements are evaluated in order until one of the if expressions is true or the end of the if/else if chain is reached. If the end of the if/else if chain is reached without a true expression, no code blocks are executed.

Use the else if statement to specify a new or multiple condition

. A common pattern when writing multiple if and else statements is to have an else block that contains another nested if statement, which can contain another else, etc. A better way to express this pattern in C# is with else if statements. The first condition that evaluates to true will run its associated code block. If none are true, then the optional else block will run if it exists.

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

In C#, the ternary operator is a special syntax of the form: condition ? expression1 : expression2.

It takes one Boolean condition and two expressions as inputs. Unlike an if statement, the ternary operator is an

expression itself. It evaluates to either its first input expression or its second input expression depending on whether the condition is true or false, respectively.

C# Switch Statements

Use the switch statement to select one of many code blocks to be executed. A switch statement is a control flow structure that evaluates one expression and decides which code block to run by trying to match the result of the

expression to each case. In general, a code block is executed when the value given for a case equals the evaluated expression, i.e, when == between the two values returns true. Switch statements are often used to replace if else structures when all conditions test for equality on one value.

This is how it works:

- 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

The break Keyword

When C# reaches a break keyword, it breaks out of the switch block. This will stop the execution of more code and case testing inside the block. When a match is found, and the job is done, it's time for a break. There is no need for more testing.

One of the uses of the break keyword in C# is to exit out of switch/case blocks and resume program execution after the switch code block. In C#, each case code block inside a switch

statement needs to be exited with the break keyword (or some other jump statement), otherwise the program will not compile. It should be called once all of the instructions specific to that particular case have been executed.

The default Keyword

The default keyword is optional and specifies some code to run if there is no case match

The goto Statement

The goto statement transfers the program control directly to a labeled statement. A common use of goto is to transfer control to a specific switch-case label or the default label in a switch statement.

Loops

In computer programming, a loop is a sequence of instructions that is continually repeated until a certain condition is reached.

C# While Loop

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

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.

C# 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.

The foreach Loop

There is also a foreach loop, which is used exclusively to loop through elements in an array.

C# Break

You have already seen the break statement used in an earlier. It was used to "jump out" of a switch statement.

The break statement can also be used to jump out of a loop.

C# Continue

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

MODULE 4 – ARRAY

Array

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

To declare an array in C#, you can use the following syntax – where;;

- datatype is used to specify the type of elements in the array.
- [] specifies the rank of the array. The rank specifies the size of the array.
- arrayName specifies the name of the array.

Access the Elements of an Array

You access an array element by referring to the index number.

Change an Array Element

To change the value of a specific element, refer to the index number.

Array Length

To find out how many elements an array has, use the Length property

Loop Through an Array

You can loop through the array elements with the for loop, and use the Length property to specify how many times the loop should run

Sort Arrays

There are many array methods available, for example Sort(), which sorts an array alphabetically or in an ascending order.

System.Linq Namespace

Other useful array methods, such as Min, Max, and Sum, can be found in the System.Linq namespace.

Other Ways to Create an Array

If you are familiar with C#, you might have seen arrays created with the new keyword, and perhaps you have seen arrays with a specified size as well. In C#, there are different ways to create an array.

Multidimensional Arrays

Arrays can have more than one dimension. C# allows multidimensional arrays. Multidimensional arrays are also called rectangular array.

You can declare a 2-dimensional array of strings as –

String [,] names;

or, a 3-dimensional array of int variables as –

int [, ,] m;

The following declaration creates a two-dimensional array of four rows and two columns

```
int[,] array = new int [4,2];
```

The following declaration creates an array of three dimensions, 4, 2, and 3.

```
Int[,] array1 = new int[4,2,3];
```

Two-Dimensional Arrays

The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is a list of one-dimensional arrays.

A 2-dimensional array can be thought of as a table, which has x number of rows and y number of columns. Following is a 2-dimensional array, which contains 3 rows and 4 columns

Thus, every element in the array a is identified by an element name of the form a[i , j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in array a.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. The Following array is with 3 rows and each row has 4 columns.

Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts. That is, row index and column index of the array.

Method

A method makes a large section of code into smaller. It is re-usable parts that make the program easier to understand.

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. Why use methods? To reuse code: define the code once, and use it many times.

To use a method, you need to –

- Define the method
- Call the method

Defining Methods in C#

When you define a method, you basically declare the elements of its structure. The syntax for defining a method in C# is as follows –

Following are the various elements of a method –

- **Access Specifier** - This determines the visibility of a variable or a method from another class.
- **Type (return type)** - A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- **Function/method Name** - Method name is a unique identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.
- **Parameter/s** - Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

• **Method Body** - This contains the set of instructions needed to complete the required activity.

- Creating a Method
- Call a Method

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

Parameters and Arguments

Information can be passed to methods as parameter. Parameters act as variables inside the method.

They are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

Default Parameter Value

You can also use a default parameter value, by using the equals sign (=).

A parameter with a default value, is often known as an "optional parameter". From the example above, country is an optional parameter and "Norway" is **the default value**.

Multiple Parameters

You can have as many parameters as you like.

Note that when you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

Return Values

The void keyword, used in the examples above, indicates that the method should not return a value. If you want the method to return

a value, you can use a primitive data type (such as int or double) instead of void, and use the return keyword inside the method.

C# - What is OOP

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the C# code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

Encapsulation

Encapsulation binds together code and the data it manipulates and keeps them both safe from outside interference and misuse. Encapsulation is a protective container that prevents

code and data from being accessed by other code defined outside the container.

C# - What are Classes and Objects?

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

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

Create a Class

To create a class, use the class keyword

When a variable is declared directly in a class, it is often referred to as a field (or attribute).

It is not required, but it is a good practice to start with an uppercase first letter when naming classes. Also, it is common that the name of the C# file and the class matches, as it makes our code organized. However, it is not required (like in Java).

Create an Object

An object is created from a class.

Class Members

Fields and methods inside classes are often referred to as "Class Members".

Object Methods

Methods normally belongs to a class, and they define how an object of a class behaves.

Just like with fields, you can access methods with the dot syntax. However, note that the method must be public. And remember that we use the name of the method followed by two parantheses () and a semicolon ; to call (execute) the method

C# 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. It can be used to set initial values for fields:

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.

Access Modifiers

The public keyword is an access modifier, which is used to set the access level/visibility for classes, fields, methods and properties.

Inheritance

Inheritance (Derived and Base Class)

In C#, it is possible to inherit fields and methods from one class to another. We group the "inheritance concept" into two categories:

- Derived Class (child) - the class that inherits from another class
- Base Class (parent) - the class being inherited from

To inherit from a class, use the : symbol

Why and When to Use "Inheritance"? –

It is useful for code reusability: reuse fields and methods of an existing class when you create a new class

Polymorphism

Polymorphism is a feature that allows one interface to be used for a general class of action. This concept is often expressed as "one interface, multiple actions". The specific action is determined by the exact nature of circumstances.

Polymorphism is the ability to treat the various objects in the same manner. It is one of the significant benefits of inheritance. We can decide the correct

call at runtime based on the derived type of the base reference. This is called late binding.

Reusability

Once a class has been written, created and debugged, it can be distributed to other programmers for use in their own program. This is called reusability, or in .NET terminology this concept is called a component or a DLL. In OOP, however, inheritance provides an important extension to the idea of reusability. A programmer can use an existing class and without modifying it, add additional features to it.

MODULE 6 – WINDOWS FORMS APPLICATION

Introduction to Visual Studio

Visual Studio is an Integrated Development Environment (IDE) developed by Microsoft to develop GUI (Graphical User Interface), console, Web applications, web apps, mobile apps,

cloud, and web services etc. With the help of this IDE, you can create managed code as well as native code.

It uses the various platforms of Microsoft software development software like Windows store, Microsoft Silverlight, and Windows API etc. It is not a language specific IDE as you can use this to write code in C#, C++, VB (Visual Basic), Python, JavaScript, and many more languages. It provides support for 36 different programming languages. It is available for Windows as well as for macOS.

Evolution of Visual Studio: The first version of VS (Visual Studio) was released in 1997, named as Visual Studio 97 having version number 5.0. The latest version of Visual Studio is 16.11.5 which was released on October 12, 2021. It is also termed as Visual Studio 2019. The supported .Net Framework Versions in latest Visual Studio is 4 to 4.8. Java was supported in old versions of Visual Studio but in latest version doesn't provide any support for Java language.

There are 3 editions of Microsoft Visual Studio as follows:

1. Community (free)
2. Professional (commercial edition)
3. Enterprise (highly scalable)

Windows Form Application

So far, we have seen how to work with C# to create console-based applications. But in a real-life scenario team normally use Visual Studio and C# to create either Windows Forms or Webbased applications.

A windows form application is an application, which is designed to run on a computer. It will not run on web browser because then it becomes a web application.

A Windows forms application is one that runs on the desktop computer. A Windows forms application will normally have a collection of controls such as labels, textboxes, list boxes, etc.

Just like many programs, the menu options are at the top. The ones you'll use most often are:

1. **File** – To open your solution, and start working on it.
2. **Save** – Save the current file you're working on.
3. **Save All** – Save all changes in the current solution you're working on.
4. **Build** – This “compiles” your solution – converts it from C# code to code that the computer can understand. This will also tell you if there are any problems in your solution.
5. **Start** – This builds (compiles) your program and runs it – so you can actually use it.

Solution Explorer

This section shows all the projects in your solution and all the files in the projects.

This is where you will add files to your projects, rename or move existing files, and (sometimes) delete files from your project.

Properties

When you're working with the parts of the project that appear on the screen – from the main form to the individual buttons and boxes on it – the Properties section will show you things you change about your currently selected object.

For instance, you can set the height and width of a form. You can set the words you want displayed on a button. You can set whether or not something is visible.

Output

When you build, or run, your program, this is where you'll receive status messages.

If everything is OK, you see that everything succeeded. If there were any problems, you'll see where they are, so you can go fix them.

Toolbox

Just like the “Properties” section, the Toolbox area is filled when you're currently working on the parts of your program that are displayed on the screen.

This section shows you all the things you can add to the forms in your program – buttons, labels (text), checkboxes, radio buttons, etc. You can select what you want to add to the form and drag it to the place where you want it located on your form. You might here this called “drag-anddrop” programming.

Workspace

This is the space where you actually work on a part of your program.

You select what you want to work on, from the Solution Explorer, and work on it here. You can have several files open on

your workspace at one time (see the tabs at the top of the workspace), but you'll only have one "on top", that you're actually working on at the moment.

Code Editor

There are two main ways you will manipulate an object of your application, visually or using code. In future sections, we will explore details of visually designing a control. As you should have found out from learning C#, code of an application is ASCII text-based, written in plain English and readable to human eyes. For an application, you can use any text editor to write your code but one of Microsoft Visual Studio's main strengths is the Code Editor, which is very intuitive.

The Code Editor is a window specially designed for code writing.

To display the code editor, in the Solution Explorer, you can click the View Code button View Code or right click on the form and choose view code.

Although all languages of the Microsoft Visual Studio programming environment share the Code Editor, once you have started a type of application, the Code Editor is adapted to the language you are using. Its parser (a program used internally to analyze your code) behaves according to the language of your choice. The features and behaviors of the Code Editor are also different, depending on your language.

Regions

When code of a file is long, it can be tiresome to scroll up and down. The Microsoft Visual Studio's Code Editor allows you to create sections that allow the code to behave like the tree arrangement of the left pane of Windows Explorer. This means that you can expand or collapse section of code. The Code Editor supports this feature automatically by adding + buttons

at the beginning of the lines of sections that can be expanded or collapsed. This is the case for namespaces, classes, methods, interfaces, properties, etc. The end of an expandable section displays a - button to the beginning of the line.

Besides the default sections that the Code Editor is intuitively aware of, you can create your own region. A region must have a beginning and an end. To specify the start of a section, type #region. You can optionally add a label to the right of #region to name the region. After creating a region with #region, the Code Editor adds a + button to its left. To expand a region, you can click its + button. This changes it into a - button.

To collapse the region, click the - button. If you don't specify the end of the region, the code from #region to the end of the file would be considered as belonging to the to the region. Therefore, you should specify the end of the region you created. To mark the end of the region, in the desired line, type #endregion.

Some popular productivity features

Some popular features in Visual Studio that improve your productivity when developing software include:

- Squiggles and Quick Actions

- Squiggles are wavy underlines that alert you to errors or potential problems in your code as you type. These visual clues help you fix problems immediately, without waiting to discover errors during build or runtime. If you hover over a squiggle, you see more information about the error. A lightbulb might also appear in the left margin showing Quick Actions you can take to fix the error.

- Refactoring

- Refactoring includes operations such as intelligent renaming of variables, extracting one or more lines of code into a new method, and changing the order of method parameters.

- IntelliSense

- IntelliSense is a set of features that display information about your code directly in the editor and, in some cases, write small bits of code for you. It's like having basic documentation inline in the editor, so you don't have to look up type information elsewhere.

Container Controls

The Windows Forms development platform supports a broad set of app development features, including controls, graphics, data binding, and user input. Windows Forms features a drag-and-drop visual designer in Visual Studio to easily create Windows Forms apps.

Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client-side, Windows-based applications. Not only does Windows Forms provide many ready-to-use controls, it also provides the infrastructure for developing your own controls. You can combine existing controls, extend existing controls, or author your own custom controls.

Adding Control

Controls are added through the Visual Studio Designer. With the Designer, you can place, size, align, and move controls. Alternatively, controls can be added through code.

Layout options

The position a control appears on a parent is determined by the value of the Location property relative to the top-left of the parent surface. The top-left position coordinate in the parent is

(x0,y0). The size of the control is determined by the Size property and represents the width and height of the control.

Besides manual positioning and sizing, various container controls are provided that help with automatic placement of controls

Fixed position and size

When a control is added to a parent that enforces automatic placement, the position and size of the control is changed. In this case, the position and size of the control may not be manually adjusted, depending on the type of parent.

The MaximumSize and MinimumSize properties help set the minimum and maximum space a control can use.

Margin and Padding

There are two control properties that help with precise placement of controls: Margin and Padding.

The Margin property defines the space around the control that keeps other controls a specified distance from the control's borders.

The Padding property defines the space in the interior of a control that keeps the control's content (for example, the value of its Text property) a specified distance from the control's borders.

There are two control properties that help with precise placement of controls: Margin and Padding. The Margin property defines the space around the control that keeps other controls a specified distance from the control's borders. The Padding property defines the space in the interior of a control that keeps the control's content (for example, the value of its Text property) a specified

distance from the control's borders. The following figure shows the Margin and Padding properties on a control.^z

Dock

The Dock property sets which border of the control is aligned to the corresponding side of the parent, and how the control is resized within the parent.

Container: Form

The Form is the main object of Windows Forms. A Windows Forms application will usually have a form displayed at all times. Forms contain controls and respect the Location and Size properties of the control for manual placement. Forms also respond to the Dock property for automatic placement.

Most of the time a form will have grips on the edges that allow the user to resize the form. The Anchor property of a control will let the control grow and shrink as the form is resized.

Container: Panel

The Panel control is similar to a form in that it simply groups controls together. It supports the same manual and automatic placement styles that a form does.

A panel blends in seamlessly with the parent, and it does cut off any area of a control that falls out of bounds of the panel. If a control falls outside the bounds of the panel and AutoScroll is set to true, scroll bars appear and the user can scroll the panel.

Unlike the group box control, a panel doesn't have a caption and border.

Container: Group box

The GroupBox control provides an identifiable grouping for other controls. Typically, you use a group box to subdivide a form by function. For example, you may have a form representing personal information and the fields related to an address would be grouped together. At design time, it's easy to move the group box around along with its contained controls.

The group box supports the same manual and automatic placement styles that a form does. A group box also cuts off any portion of a control that falls out of bounds of the panel.

Unlike the panel control, a group box doesn't have the capability to scroll content and display scroll bars.

Container: Tab control

The TabControl displays multiple tabs, like dividers in a notebook or labels in a set of folders in a filing cabinet. The tabs can contain pictures and other controls. Use the tab control to produce the kind of multiple-page dialog box that appears many places in the Windows operating system, such as the Control Panel and Display Properties. Additionally, the TabControl can be used to create property pages, which are used to set a group of related properties.

The most important property of the TabControl is TabPages, which contains the individual tabs. Each individual tab is a TabPage object.

Working with the Label Control

Windows Forms Label controls are used to display text that cannot be edited by the user. They're used to identify objects on a form and to provide a description of what a certain control represents or does. For example, you can use labels to add descriptive captions to text boxes, list boxes, combo boxes, and

so on. You can also write code that changes the text displayed by a label in response to events at run time.

Textbox Control

A TextBox control accepts user input on a Form. We can create a TextBox control using a Forms designer at design-time or using the TextBox class in code at run-time (also known as dynamically).

To create a TextBox control at design-time, you simply drag and drop a TextBox control from Toolbox to a Form in Visual Studio. After you drag and drop a TextBox on a Form, the TextBox looks like Figure 1. Once a TextBox is on the Form, you can move it around and resize it using the mouse and set its properties and events.

Control Properties

A property is a piece of information that characterizes or describes a control. It could be related to its location or size. It could be its color, its identification, or any visual aspect that gives it meaning. The properties of an object can be changed either at design time or at run time. You can also manipulate these characteristics both at design and at run times. This means that you can set some properties at design time and some others at run time.

To manipulate the properties of a control at design time, first select it on the form. While a control is selected, use the Properties window to manipulate the properties of the control at design time.

The Properties window starts on top with a title bar, which displays the string Properties. If the window is docked somewhere, it displays the Window Position Window Position, the Auto Hide Auto-Hide, and the Close Close buttons on its right side. If the window is floating, it would display only the Close button

Under the title bar, the Properties window displays a combo box. The content of the combo box is the name of the form plus the names of the controls currently on the form. Besides the technique we reviewed earlier to select a control, you can click the arrow of the combo box and select a control from the list:

Accessing the Properties of One or More Controls

When a control is selected, the Properties window displays only its characteristics:

You can also change some characteristics of various controls at the same time. To do this, first select the controls on the form and access the Properties window:

When various controls have been selected:

- The Properties window displays only the characteristics that are common to the selected controls
- The combo box on top of the Properties window is empty
- Some fields of the Properties window appear empty because the various controls that are selected have different values for those properties

Properties Categories

Each field in the Properties window has two sections: the property's name and the property's value:

The name of a property is represented on the left column. This is the official name of the property. The names of properties are in one word. You can use this same name to access the property in code.

The box on the right side of each property name represents the value of the property that you can set for an object. There are various kinds of fields you will use to set the properties. To know what particular kind a field is, you can click its name. To set or change a property, you use the box on the right side of the property's name: the property's value, also referred to as the field's value.

Creating Controls at Run-Time

Creating a TextBox control at run-time is merely a work of creating an instance of TextBox class, setting its properties and adding TextBox class to the Form controls.

The first step to create a dynamic TextBox is to create an instance of TextBox class. The following code snippet creates a TextBox control object

In the next step, you may set properties of a TextBox control. The following code snippet sets background color, foreground color, Text, Name, and Font properties of a TextBox.

Once a TextBox control is ready with its properties, next step is to add the TextBox control to the Form. To do so, we use Form.Controls.Add method. The following code snippet adds a TextBox control to the current Form.

WinForms UI Controls & Components

What is Windows Control Library?

The Windows Control Library project template is used to create custom controls to use on Windows Forms

WinForms UI controls

WinForms UI controls are reusable design elements that help developers implement a wide variety of features in their desktop applications in less time.

Examples of UI Framework:

1. ModernUI Framework in Windows Forms (MetroFramework)
2. Bunifu Framework
3. Xander UI Framewor

Using ModernUI Framework (MetroFramework)

- Metro Framework is an open source DLL.
- Metro Framework provides a rich user interface. We can make our application design look good by using Metro Framework.
- We can make our desktop Windows application more attractive
- Metro Framework is basically similar to WinForm but the design is different.
- Metro Framework works with .NET Framework 4.0

Step 1 - Adding the MetroFramework UI to your VS

. Option 1: Can get from NuGet Package Manager (NuGet Extension should be installed to do this)

Option 2: Download the archive from <https://github.com/peters/winforms-modernui> and place in your local drive. Then, reference these three libraries. Please note the MetroFramework UI is open source

After referencing the libraries, the next step is to add the controls in the toolbox. This is essential because henceforth only these controls shall be used in the forms

Create your first ModernUI form

To create the form, right click on the Project in the Solution Explorer and select Add --> Windows Forms.

MODULE 7 – WORKING WITH FILES

C# - Stream

C# includes following standard IO (Input/Output) classes to read/write from different sources like files, memory, network, isolated storage, etc.

Stream: System.IO.Stream is an abstract class that provides standard methods to transfer bytes (read, write, etc.) to the source. It is like a wrapper class to transfer bytes. Classes that need to read/write bytes from a particular source must implement the Stream class.

The following classes inherit Stream class to provide the functionality to Read/Write bytes from a particular source:

- **FileStream** reads or writes bytes from/to a physical file, whether it is a .txt, .exe, .jpg, or any other file. FileStream is derived from the Stream class.
- **MemoryStream:** MemoryStream reads or writes bytes that are stored in memory.
- **BufferedStream:** BufferedStream reads or writes bytes from other Streams to improve certain I/O operations' performance.
- **NetworkStream:** NetworkStream reads or writes bytes from a network socket.

- **PipeStream:** PipeStream reads or writes bytes from different processes.
- **CryptoStream:** CryptoStream is for linking data streams to cryptographic transformations

Stream Readers and Writers

- **StreamReader:** StreamReader is a helper class for reading characters from a Stream by converting bytes into characters using an encoded value. It can be used to read strings (characters) from different Streams like FileStream, MemoryStream, etc.
- **StreamWriter:** StreamWriter is a helper class for writing a string to a Stream by converting characters into bytes. It can be used to write strings to different Streams such as FileStream, MemoryStream, etc.
- **BinaryReader:** BinaryReader is a helper class for reading primitive datatype from bytes.
- **BinaryWriter:** BinaryWriter writes primitive types in binary.



The above image shows that **FileStream** reads bytes from a physical file, and then **StreamReader** reads strings by converting those bytes to strings. In the same way, the **StreamWriter** takes a string and converts it into bytes and writes to the **FileStream**, and then the **FileStream** writes the bytes to a physical file. So, the **FileStream** deals with bytes, where as **StreamReader** and **StreamWriter** deals with strings.

Points to Remember:

1. Stream is an abstract class for transferring bytes from different sources. It is base class for all other class that reads\writes bytes to different sources.
2. FileStream class provides reading and writing functionality of bytes to physical file.
3. Reader & writer classes provides functionality to read bytes from Stream classes (FileStream, MemoryStream etc) and converts bytes into appropriate encoding.
4. StreamReader provides a helper method to read string from FileStream by converting bytes into strings. StreamWriter provides a helper method to write string to FileStream by converting strings into bytes.

Working with Files & Directories in C#

C# provides the following classes to work with the File system. They can be used to access directories, access files, open files for reading or writing, create a new file or move existing files from one location to another, etc.

CLASS NAME AND ITS USAGE

FILE

File is a static class the provides different functionalities like copy, create, move, delete, open for reading or /writing, encrypt or decrypt, check if a file exists, append lines or text to a files content, get last access time, etc.

FILEINFO

The FileInfo class provides the same functionality as a static File class. You have more control on how you do read/write operations on a file by writing code manually for reading or writing bytes form a file.

DIRECTORY

Directory is a static class that provides functionality for creating, moving, deleting and accessing subdirectories

DIRECTORYINFO

DirectoryInfo provides instance methods for creating, moving, deleting and accessing subdirectories.

PATH

Path is a static class that provides functionality such as retrieving the extension of a file, changing the extension of a file, retrieving the absolute the physical path, and other path related functionalities.

File

C# includes static File class to perform I/O operation on physical file system. The static File class includes various utility method to interact with physical file of any type e.g. binary, text etc.

Use this static File class to perform some quick operation on physical file. It is not recommended to use File class for multiple operations on multiple files at the same time due to performance reasons. Use FileInfo class in that scenario.

Some Important Methods of Static File Class

Method and Its usage

COPY

Copies an existing file to a new file. Overwriting a file of the same name is not allowed.

CREATE

Creates or overwriting a file in the specified path.

DELETE

Deletes the specified file.

EXISTS

Determines whether the specified file exists

MOVE

Moves a specified file to a new location, providing the option to specify a new file name.

OPEN

Opens a FileStream on the specified path with read/write access..

```
//Copy DummyFile.txt as new file DummyFileNew.txt
File.Copy(@"C:\DummyFile.txt", @"D:\DummyFileNew.txt");
```

```
//Check whether file is exists or not at particular location bool
isFileExists = File.Exists(@"C:\ DummyFile.txt"); // returns false
```

```
// Move file to new location File.Move(@"C:\DummyFile.txt",
@"D:\DummyFile.txt");
```

```
//Open file and returns FileStream for reading bytes from the file
FileStream fs = File.Open(@"D:\DummyFile.txt",
FileMode.OpenOrCreate);
```

```
//Delete file File.Delete(@"C:\DummyFile.txt");
```

Thus, it is easy to work with physical file using static File class. However, if you want more flexibility then use FileInfo class. The same way, use static Directory class to work with physical directories.

Points to Remember:

1. File is a static class to read\write from physical file with less coding.
2. Static File class provides functionalities such as create, read\write, copy, move, delete and others for physical files.
3. Static Directory class provides functionalities such as create, copy, move, delete etc for physical directories with less coding.
4. FileInfo and DirectoryInfo class provides same functionality as static File and Directory class.

C# Directory

A directory, also called a folder, is a location for storing files on your computer. In addition to files, a directory also stores other directories or shortcuts.

In C# we can use Directory or DirectoryInfo to work with directories. Directory is a static class that provides static methods for working with directories. An instance of a DirectoryInfo provides information about a specific directory.

The classes are available in the System.IO namespace.

C# create directory

A directory is created with the Directory.CreateDirectory method. var path = @"c:\C-Sharf Upload\"; Directory.CreateDirectory(path);

Directory.Exists method

```
if (Directory.Exists(path))
{
    Console.WriteLine("Directory exists");
}
Else
{
    Console.WriteLine("Directory does not exist");
}
```

C# delete directory

A directory is deleted with the Directory.Delete method.

```
Directory.Delete(path);
```

MODULE 8 – NAMESPACES

C# - Namespaces

A namespace is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Defining a Namespace

A namespace definition begins with the keyword namespace followed by the namespace name as follows –

```
Namespace namespace _name {
    // code declarations
}
```

To call the namespace-enabled version of either function or variable, prepend the namespace name as follows –

```
Namespace_name.item_name;
```

The using Keyword

The using keyword states that the program is using the names in the given namespace. For example, we are using the System namespace in our programs. The class Console is defined there. We just write –

```
Console.WriteLine("Hello there");
```

We could have written the fully qualified name as –

```
System.Console.WriteLine("Hellp there");
```

Nested Namespaces

You can define one namespace inside another namespace as follows-

```
Namespace namespace_name1 {
    // code declarations
    Name namespace_name2{
        // code declarations
    }
}
```

}

Dynamic Link library (DLL)

A Dynamic Link library (DLL) is a library that contains functions and codes that can be used by more than one program at a time.

Once we have created a DLL file, we can use it in many applications. The only thing we need to do is to add the reference/import the DLL File. Both DLL and .exe files are executable program modules but the difference is that we cannot execute DLL files directly.

Creating DLL File

Step 1 - Open Visual Studio then select "File" -> "New" -> "Project..." then select "Visual C#" -> "Class library"

Step 2 - Change the class name ("class1.cs").

Step 3 - In the calculate class, write methods for the addition and subtraction of two integers (for example purposes)

Step 4 - Build the solution (F6). If the build is successful then you will see a "calculation.dll" file in the "bin/debug" directory of your project

Using DLL File

Step 1 - Open Visual Studio then select "File" -> "New" -> "Project..." then select "Visual C#" -> "Windows Forms application"

Step 2 - Design the form;

Step 3 - Add a reference for the dll file, "calculation.dll", that we created earlier. Right-click on the project and then click on "Add reference"- Add a reference for the dll file, "calculation.dll", that we created earlier. Right-click on the project and then click on "Add reference"

Step 4 - Select the DLL file and add it to the project.

After adding the file, you will see that the calculation namespace has been added (in references) as in the following:

Step 5 - Add the namespace ("using calculation;")

MODULE 9 – EVENT DRIVEN PROGRAMMING USING C#

Event-Driven Programming

What is Event-Driven Programming?

Event-driven programming is a programming paradigm in which the flow of program execution is determined by events - for example a user action such as a mouse click, key press, or a message from the operating system or another program. An event-driven application is designed to detect events as they occur, and then deal with them using an appropriate event-handling procedure.

Events in C#

The Event is something special that is going to happen.

Event Loops

The Event loop keeps testing the user interface to see if anything has happened e.g. a button click or a key pressed. When an event is detected, it is passed to the list of identified trigger functions, which then launches a response to the event. The event handlers are the actual program code

modules that are executed when a particular trigger has occurred. Other (non user) activities can also cause an event to trigger – typically used in networking and when reading from or writing to files.

Trigger functions

Event driven programs respond to events triggered by the user, the programmer chooses which event(s) to respond to by selecting the appropriate trigger function. Every object has a range of trigger functions – one for each possible event. Different controls have different events (events for a text box not the same for a button), a Button has a Click event, a GotFocus event, a MouseHover event etc.

Event Handlers

Event handlers contain the code that runs when an event occurs. These event handlers are self-contained sections of code – also called Procedures or Subroutine.

This allows the programmer to work on one event of one control at a time and makes testing easier.

Forms and Controls

A GUI (graphical user interface) is what the user sees when the program runs and usually consists of one or several forms. Event driven programming is particularly suitable for GUIs - uses WIMP system Windows, Icons, Menus, Pointers. Programmer can assist user with intuitive interface, restrict input to a limited range, provide visual feedback and context-sensitive help. Event driven programs respond to events triggered by the user via the mouse or keyboard.

Common methods for user interaction include clicking the mouse or pressing a key. Each form contains various controls which allow the user to interact with the program in different ways e.g. Buttons, Textboxes, Labels, Menus, Pictureboxes, Timers.

Common Events of C#

Controls Events are what happen in and around your program. For example, when a user clicks a button, many events occur: The mouse button is pressed, the CommandButton in your program is clicked, and then the mouse button is released. These three things correspond to the MouseDown event, the Click event, and the MouseUp event. During this process, the GotFocus event for the CommandButton and the LostFocus event for whichever object previously held the focus also occur.

Again, not all controls have the same events, but some events are shared by many controls. These events occur as a result of some specific user action, such as moving the mouse, pressing a key on the keyboard, or clicking a text box. These types of events are user-initiated events and are what you will write code for most often.

Event and it Occurs When ...

Change

The user modifies text in a combo box or text box.

Click

The user clicks the primary mouse button on an object.

DbClick

The user double-clicks the primary mouse button on an object. **DragDrop**

The user drags an object to another location.

DragOver

The user drags an object over another control.

GotFocus

An object receives focus.

LostFocus

An object loses focus.

MouseDown

The user presses any mouse button while the mouse pointer is over an object.

MouseMove

The user moves the mouse pointer over an object.

MouseUp

The user releases any mouse button while the mouse pointer is over an object.

Advantages

- Flexibility: Wide choice of controls and trigger functions - programmer can decide what will happen when
- Suitability for GUI: WIMP system (Windows, Icons, Menus, Pointers)
- Simplicity and Ease of development:
 - o Programmers can add and code one object at a time, using simple constructs
 - o Pre-written code and drop-down lists showing choice of possible commands

- o Errors are highlighted and step through debugging

Disadvantages

- Errors can be more difficult to spot than with simpler, procedural programs
- Programs with complex GUIs may be slower to load and run than simpler programs – particularly if RAM is insufficient
- Programs with too many forms can be very confusing and/or frustrating for the user.

Every modern programming language contains all necessary functionalities for handling KeyBoard related events.

C# also provides us with three events KeyPress, KeyUp and KeyDown which you can use to handle Keyboard events:

KeyDown

This event is raised when a user presses a physical key. Handler receives:

- o A KeyEventArgs parameter, which provides the KeyCode property (which specifies a physical keyboard button).
- o The Modifiers property (SHIFT, CTRL, or ALT).
- o The KeyData property (which combines the key code and modifier).
- o The KeyEventArgs parameter also provides:
 - The SuppressKeyPress property, which can be used to suppress the KeyPress and KeyUp events for that keystroke.

KeyPress

This event is raised when the key or keys pressed result in a character. For example, a user presses SHIFT and the lowercase "a" keys, which result in a capital letter "A" character.

KeyPress is raised after KeyDown

The handler for KeyPress receives:

- A KeyPressEventArgs parameter, which contains the character code of the key that was pressed. This character code is unique for every combination of a character key and a modifier key.

- o The character code 65, if it is pressed with the SHIFT key

- o Or the CAPS LOCK key, 97 if it is pressed by itself,

- o And 1, if it is pressed with the CTRL key.

KeyPressEventArgs Parameter Also provides:

- o The Handled property, which can be set to prevent the underlying control from receiving the key.

KeyUp

This event is raised when a user releases a physical key. **Handler** receives:

- A KeyEventArgs parameter:
 - o Which provides the KeyCode property (which specifies a physical keyboard button).
 - o The Modifiers property (SHIFT, CTRL, or ALT).

- o The KeyData property (which combines the key code and modifier).

Keys Enum

- Specifies key codes and modifiers.

This enumeration has a **FlagsAttribute** attribute that allows a bitwise combination of its member values.

KeyEventArgs.SuppressKeyPress Property

The following code example prevents numeric keystrokes from reaching the **TextBox** control named textBox1.

You can assign true to this property in an event handler such as **KeyDown** in order to prevent user input. Setting **SuppressKeyPress** to true also sets **Handled** to true. Setting the **Handled** property in the KeyDown event handler does not prevent the **KeyPress** and **KeyUp** events from being raised for the current keystroke. Use the **SuppressKeyPress** property for this purpose.

```
private void textBox1_KeyDown(object sender, EventArgs e)
{
    if (e.KeyCode >= Keys.D0 && e.KeyCode <= Keys.D9 &&
        e.Modifiers != Keys.Shift)
    {
        e.SuppressKeyPress = true;
    }
}
```

Handled

The following example handles the KeyPress event to consume the A and a character keys. Those keys can't be typed into the text box:

```
private void textBox1_KeyPress(object sender,
KeyPressEventArgs e)
{ if (e.KeyChar == 'a' || e.KeyChar == 'A'){ e.Handled = true; }
}
```

MODULE 11.1 : WORKING WITH DATE AND TIME

Working with Date and Time in C#

C# includes **DateTime** struct to work with dates and times. To work with date and time in C#, create an object of the **DateTime** struct using the new keyword.

The following creates a **DateTime** object with the default value.

Example: Create DateTime Object

```
DateTime dt = new DateTime(); // assigns default value
01/01/0001 00:00:00
```

The default and the lowest value of a **DateTime** object is January 1, 0001 00:00:00 (midnight). The maximum value can be December 31, 9999 11:59:59 P.M.

Use different constructors of the **DateTime** struct to assign an initial value to a **DateTime** object.

Example: Set Date & Time

```
//assigns default value 01/01/0001 00:00:00
DateTime dt1 = new DateTime();
//assigns year, month, day
DateTime dt2 = new DateTime(2015, 12, 31);
//assigns year, month, day, hour, min, seconds
DateTime dt3 = new DateTime(2015, 12, 31, 5, 10, 20);
//assigns year, month, day, hour, min, seconds, UTC
timezone
DateTime dt4 = new DateTime(2015, 12, 31, 5, 10, 20,
DateTimeKind.Utc);
```

In the above example, we specified a year, a month, and a day in the constructor. The year can be from 0001 to 9999, and the Month can be from 1 to 12, and the day can be from 1 to 31. Setting any other value out of these ranges will result in a run-time exception.

Example: Invalid Date

```
DateTime dt = new DateTime(2015, 12, 32); //throws
exception: day out of range
```

DateTime Static Fields

The `DateTime` struct includes static fields, properties, and methods. The following example demonstrates important static fields and properties.

Example: Static Fields

```
DateTime currentDateTime = DateTime.Now; //returns  
current date and time
```

```
DateTime todaysDate = DateTime.Today; // returns  
today's date
```

```
DateTime currentDateTimeUTC = DateTime.UtcNow; //  
returns current UTC date and time
```

```
DateTime maxDateTimeValue = DateTime.MaxValue; //  
returns max value of DateTime
```

```
DateTime minDateTimeValue = DateTime.MinValue; //  
returns min value of DateTime
```

Operators

The `DateTime` struct overloads `+`, `-`, `==`, `!=`, `>`, `<`, `<=`, `>=` operators to ease out addition, subtraction, and comparison of dates. These make it easy to work with dates.

Example: Operators

```
DateTime dt1 = new DateTime(2015, 12, 20);  
DateTime dt2 = new DateTime(2016, 12, 31, 5, 10, 20);  
TimeSpan time = new TimeSpan(10, 5, 25, 50);
```

```
Console.WriteLine(dt2 + time); // 1/10/2017 10:36:10  
AM
```

```
Console.WriteLine(dt2 - dt1); //377.05:10:20
```

```
Console.WriteLine(dt1 == dt2); //False
```

```
Console.WriteLine(dt1 != dt2); //True
```

```
Console.WriteLine(dt1 > dt2); //False
```

```
Console.WriteLine(dt1 < dt2); //True
```

```
Console.WriteLine(dt1 >= dt2); //False
```

```
Console.WriteLine(dt1 <= dt2); //True
```

Convert String to DateTime

A valid date and time string can be converted to a `DateTime` object using `Parse()`, `ParseExact()`, `TryParse()` and `TryParseExact()` methods.

The `Parse()` and `ParseExact()` methods will throw an exception if the specified string is not a valid representation of a date and time. So, it's recommended to use `TryParse()` or `TryParseExact()` method because they return false if a string is not valid.

Example:

```
var str = "5/12/2020";  
DateTime dt;  
var isValidDate = DateTime.TryParse(str, out dt); //  
convert string into date
```



```
if(isValidDate)
    Console.WriteLine(dt);
else
    Console.WriteLine($"{str} is not a valid date string");
```

MODULE 11.2: MDI FORM

C# MDI Form

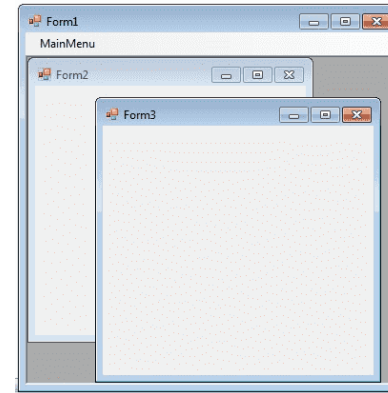
What is a Multiple Document Interface?

A multiple document interface (MDI) is a graphical user interface in which multiple windows reside under a single parent window. Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies. This contrasts with the single document interfaces (SDI) where all windows are independent of each other.

A Multiple Document Interface (MDI) programs can display multiple child windows inside them. This is in contrast to single document interface (SDI) applications, which can manipulate only one document at a time.

Visual Studio Environment is an example of Multiple Document Interface (MDI) and notepad is an

example of an SDI application. MDI applications often have a Window menu item with submenus for switching between windows or documents.




Any Forms can become an MDI parent, if you set the *IsMdiContainer* property to True.

For Runtime:

IsMdiContainer = true;

For Design Time (Properties):

| | |
|------------------|--|
| [-] Window Style | |
| ControlBox | True |
| HelpButton | False |
| [-] Icon |  (Icon) |
| IsMdiContainer | True |
| MainMenuStrip | (none) |
| MaximizeBox | True |
| MinimizeBox | True |
| Opacity | 100% |

The following C# program shows a MDI form with two child forms. Create a new C# project, then you will get a default form Form1 . Then add two more forms in the project (Form2 , Form 3) . Create a Menu on your form and call these two forms on menu click event.

This code is for displaying child form to your parent form. Remember that “this” keyword is your current or active Form, or you main parent form.

```
Form2 childForm = new Form2();
childForm.MdiParent = this;
childForm.Show();
```

NOTE: *If you want the MDI parent to auto-size the child form you can code like this.*

```
Form2 childForm = new Form2();
childForm.MdiParent = this;
childForm.Dock=DockStyle.Fill;
```

```
childForm.Show();
```

Advantages of Multiple Document Interface:

- With MDI, a single menu bar and/or toolbar is shared between all child windows, reducing clutter and increasing efficient use of the screen space.
- An application's child windows can be hidden/shown/minimized/maximized as a whole.

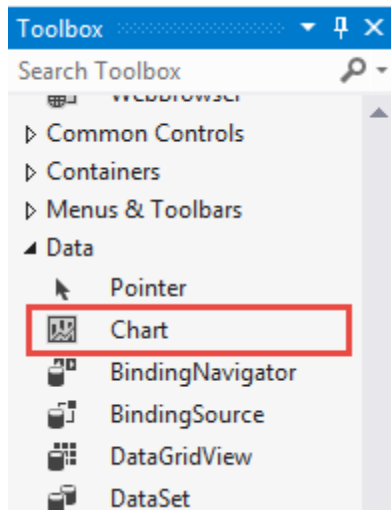
C# Chart: Windows Form

1. C# Chart: Windows Forms

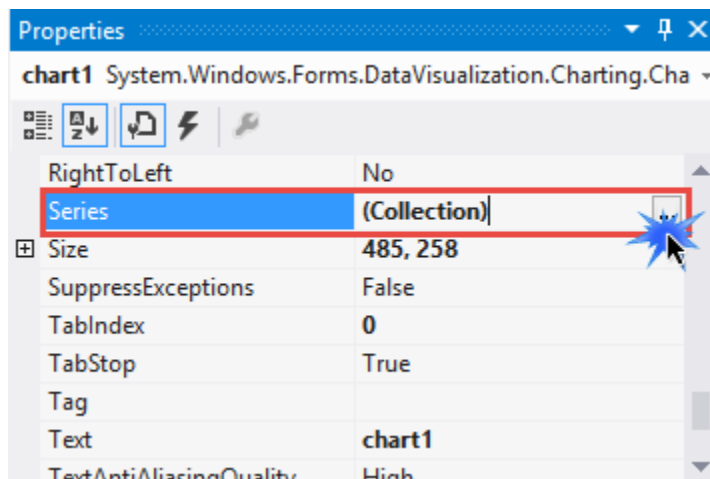
a.Chart. The Chart control visualizes your data.

It displays data in your Windows Forms program as a bar graph or chart. With Chart you can quickly display your data in a colorful graphic controlled by C# code.

Start. First, you must be using a newer version of the .NET Framework. Older versions will not have the Chart control available. Please open the Toolbox and drag the Chart item to the Form.

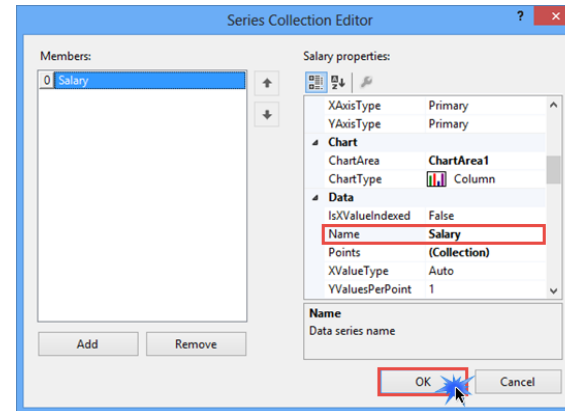


Next: Select Properties after right-clicking on the Chart. You can add Series and Titles this way.



And: For our example, remove the default Series1 that was added by Visual Studio.

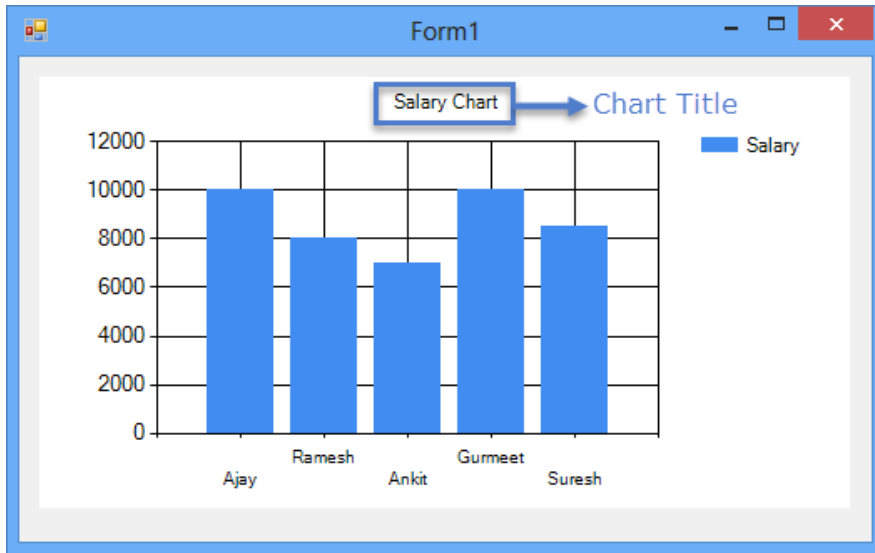
Change the name of Series. Here, I set the name to Salary. We can also change the ChartType as well as appearance from the Series Collection Editor.



Example. Now we can use the Chart. In the next step, we will use the Form1_Load event handler to initialize the Chart we just added. To add Form1_Load, double-click on the Form window in Visual Studio.

```
01. using System;
02. using System.Windows.Forms;
03.
04. namespace DemoChart
05. {
06.     public partial class Form1 : Form
07.     {
08.         public Form1()
09.         {
10.             InitializeComponent();
11.         }
12.
13.         private void Form1_Load(object sender, EventArgs e)
14.         {
15.             fillChart();
16.         }
17.         //fillChart method
18.         private void fillChart()
19.         {
20.             //AddXY value in chart1 in series named as Salary
21.             chart1.Series["Salary"].Points.AddXY("Ajay", "10000");
22.             chart1.Series["Salary"].Points.AddXY("Ramesh", "8000");
23.             chart1.Series["Salary"].Points.AddXY("Ankit", "7000");
24.             chart1.Series["Salary"].Points.AddXY("Gurmeet", "10000");
25.             chart1.Series["Salary"].Points.AddXY("Suresh", "8500");
26.             //chart title
27.             chart1.Titles.Add("Salary chart");
28.         }
29.     }
30. }
```

Preview



Charts and Graphs make data easier to understand and interpret. A chart is used to present the data in a visual form.

2.

3. Chart Class

a. Definition

Namespace:

`System.Windows.Forms.DataVisualization.Charting`

Assembly:

`System.Windows.Forms.DataVisualization.dll`

Serves as the root class of the [Chart](#) control.

This class exposes all of the properties, methods and events of the Chart Windows control.

Two important properties of the [Chart](#) class are the [Series](#) and [ChartAreas](#) properties, both of which are collection properties. The [Series](#) collection property stores [Series](#) objects, which are used to store data that is to be displayed, along with attributes of that data. The [ChartAreas](#) collection property stores [ChartArea](#) objects, which are primarily used to draw one or more charts using one set of axes.

4. Chart.Series Property

This collection property allows you to access the [SeriesCollection](#) class, which stores [Series](#) objects and also provides methods and properties used to manipulate this collection.

Acts as a wrapper around data that is to be displayed, and associate styles with the data.



Get name of first Series:

```
chart1.Series[0].Name
```

Remove Series:

```
chart1.Series.Remove(chart.Series["Male"]);
```

Add Series:

```
chart.Series.Add("Female");
```

Chart Type

Chart types are specified on each Chart Series through the type property. Essential Chart includes a comprehensive set of more than 35 chart types for all your business needs. Each one is highly and easily configurable with builtin support for creating stunning visual effects.

All the chart types are required to have at least one X and one Y value. Certain chart types need more than one Y value.

| Chart Type | Minimum Number of Series | Maximum Number of Series | Number of Y Values Required |
|------------------------|--------------------------|--------------------------|-----------------------------|
| Area Charts | 1 | Unlimited | 1 |
| Bar Charts | 1 | Unlimited | 1 |
| Box And Whisker Charts | 1 | Unlimited | 5 |
| Bubble Charts | 1 | Unlimited | 2 |
| Candle Charts | 1 | Unlimited | 4 |
| Column Charts | 1 | Unlimited | 1 |

Change Cart Type (default chart type is column):

```
chart.Series["Male"].ChartType = SeriesChartType.Area;
```

```
chart.Series["Female"].ChartType = SeriesChartType.Area;
```

Points

```
chart.Series["Male"].Points.AddXY("BSIT", 12);
chart.Series["Female"].Points.AddXY("BSIT", 6);
chart.Series["Male"].Points.AddXY("BSED", 13);
chart.Series["Male"].Points.AddXY("AB", 2);
chart.Series["Female"].Points.AddXY("AB", 16);
```

MODULE 13 – MySQL Database

MySQL

MySQL is an open-source relational database management system. Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

Database

A database is simply a collection of structured data. Think of taking a selfie: you push a button and capture an image of yourself. Your photo is data, and your phone's gallery is the database. A database is a place in which data is stored and organized. The word "relational" means that the data stored in the dataset is organized as tables. Every table relates in some ways. If the software doesn't support the relational data model, just call it DBMS.

SQL

MySQL and SQL are not the same. Be aware that MySQL is one of the most popular RDBMS software's brand names,

which implements a client-server model. So, how do the client and server communicate in an RDBMS environment? They use a domain-specific language – Structured Query Language (SQL). If you ever encounter other names that have SQL in them, like PostgreSQL and Microsoft SQL server, they are most likely brands which also use Structured Query Language syntax. RDBMS software is often written in other programming languages, but always use SQL as their primary language to interact with the database. MySQL itself is written in C and C++..

MySQL Installation

Visual Studio / Windows Application Requirements:

1. **MySQL Server** – MySQL server can install using MySQL Community installer. MySQL Community Edition is a freely downloadable version of the world's most popular open-source database that is supported by an active community of open-source developers and enthusiasts.

- MySQL Server can be also found in Wamp Server. WAMP Stands for "Windows, Apache, MySQL, and PHP." It is a variation of LAMP for Windows systems and is often installed as a software bundle (Apache, MySQL, and PHP).

2. **MySQL Workbench** – MySQL Workbench can also install using MySQL Community installer or MySQL Workbench standalone installer.

- **MySQL Workbench** provides DBAs and developers an integrated tools environment for:
 - o Database Design & Modeling
 - o SQL Development
 - o Database Administration
 - o Database Migration

- **Wamp Server** also includes DBA's tool called **PhpMyAdmin**.

- You can also use **Navicat Premium** as DBAs Tool

3. MySQL Connector .Net

- Connector/NET is a fully-managed .NET Application driver for MySQL.

- MySQL Connector/NET 8.0 is compatible with all MySQL versions starting with MySQL 5.6. Additionally, MySQL Connector/NET 8.0 supports the new X DevAPI for development with MySQL Server 8.0.

- This driver is use to connect your windows/.Net application to your database stored in MySql Server

- . - MySQL Connector/NET enables you to develop .NET applications that require secure, high-performance data connectivity with MySQL.

Using MySQL Library

MySQL References:

- MySql.Data (MySql.Data.dll)
- Right click on your solution explorer project folder and add references. Browse the MySql.Data.dll from MySQL Connector .Net installation folder.

MySql.Data provides different namespaces including **MySql.Data.MySqlClient** that will be used for .net application development.

MySql.Data.MySqlClient is a namespace use to organize the different classes.

MySql.Data.MySqlClient selected classes:

- MySqlConnection
- MySqlCommand
- MySqlDataAdapter
- MySqlDataReader

Note: Every MySqlConnection's classes contains different constructors, methods, properties use for database integration in your project

MySqlConnection Class

The MySqlConnection class represents an open connection to a MySql database in C#. We can pass the connection string to the constructor of the MySqlConnection class to initialize a new instance of the MySqlConnection class that can connect to our database. The MySqlConnection.Open() function opens the connection for performing any operation on the MySql database. The MySqlConnection.Close() function closes the previously open connection to the MySql database. Any open connections must be closed with the MySqlConnection.Close() function after the operations are performed. The following code example shows us how to connect to a MySql database with the MySql.Data package in C#.

Assembly:

MySql.Data.dll

Reference:

MySql.Data

Namespace:

using MySql.Data.MySqlClient;

Classes:

MySqlConnection - open connection to MySql Server

o Open()

o Close()

Module 14 - Working with Data (Form Control + Database)

Form Controls

1. DataGridView
2. ListView

1. DataGridView Control

DataGridView control is designed to be a complete solution for displaying tabular data with Windows Forms. This control makes it easy to define the basic appearance of cells and the display formatting of cell values.

The **Cell** is the fundamental unit of interaction for the DataGridView. All cells derive from the DataGridViewCell base class. Each cell within the DataGridView control can have its own style, such as text format, background color, foreground color, and font.

The DataGridView control is highly configurable and extensible, and it provides many properties, methods, and events to customize its appearance and behavior.

DataGridView Properties:

- Columns property
- Rows property

WORKING WITH MYSQL DATABASE

MySql.Data.MySqlClient selected classes:

- MySqlConnection
- MySqlCommand
- MySqlDataAdapter
- MySqlDataReader

MySqlCommand Class

Represents a SQL statement to execute against a MySQL database. This class cannot be inherited.

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll) Version: 6.10.9

MySqlCommand Methods:

- ExecuteReader() - Sends the CommandText to the Connection and builds a MySqlDataReader.
- ExecuteNonQuery - Executes a SQL statement against the connection and returns the number of rows affected. (Overrides DbCommand.ExecuteNonQuery()).

MySqlDataAdapter Class

Represents a set of data commands and a database connection that are used to fill a dataset and update a MySQL database. This class cannot be inherited

. Namespace: MySql.Data.MySqlClient A

assembly: MySql.Data (in MySql.Data.dll) Version: 6.10.9

MySqlDataAdapter Constructor:

- `MySqlDataAdapter(String, MySqlConnection)` - Initializes a new instance of the `MySqlDataAdapter` class with a `SelectCommand` and a `MySqlConnection` object

MySqlDataReader Class

Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited

Assembly:

MySql.Data.dll

Reference:

MySql.Data

Namespace:

using `MySql.Data.MySqlClient`;

Classes:

`MySqlConnection` - open connection to MySql Server

- o `Open()`

- o `Close()`

`MySqlCommand` - SQL Statement to execute againsts a MySql Database

- o `ExecuteReader()`

- o `ExecuteNonQuery()`

`MySqlDataAdapter` - set of data commands and db connection use to fill a dataset and update MySql db

- o `Fill()`

`DataTable` - one table of in-memory data

`MySqlDataReader` - reading of stream of row from MySql Database While the `MySqlDataReader` is in use, the associated `MySqlConnection` is busy serving the `MySqlDataReader`. While in this state, no other operations can be performed on the `MySqlConnection` other than closing it.

This is the case until the `Close()` method of the `MySqlDataReader` is called. `DataSource` - Gets or sets the data source that the `DataGridView` is displaying data fo