**TRIBHUVAN UNIVERSITY**

Faculty of Humanities and

Social Science



Ambikeshwari Campus

Lab Report of Dot Net Technology

Submitted By : Submitted To:

Apar Singh Chaudhary Er. Pawan Adhikari

**Index Sheet**

|  |  |
| --- | --- |
| **Pgm. No** | **Topic Name** |
| 1 | Write an C# program to compare any two strings |
| 2 | Write an C# program to add, subtract, multiply and divide two numbers. |
| 3 | Write an C# program to find the smallest among three numbers using if-  else-if ladder technology. |
| 4 | Write an C# program to find sum of numbers from 5 to 100. |
| 5 | Write an C# program to print the given pattern:  \*\*\*\*\*  \*\*\*\*  \*\*\*  \*\*  \* |
| 6 | Write an C# program to find the sum of 10 numbers in an array by using foreach loop. |
| 7 | Write an C# program to find maximum and minimum numbers from an array. |
| 8 | Write an C# program to sort the names  of 10 persons in alphabetical order. |
| 9 | Write an C# program to add any two matrices. |
| 10 | Write an C# program to overload unary operators. |
| 11 | Write an C# program to perform  method overloading. |
| 12 | Write an C# program to perform method overriding. |
| 13 | Write an C# program to demonstrate abstraction using abstract class. |
| 14 | Write an C# program to demonstrate  interface in C#. |
| 15 | Write a C# program to demonstrate Delegates in C#. |
| 16 | Write an C# program to demonstrate the use of ADO.NET to work with a SQL  server Database. |

**Experiment No. 1**

**Objective:**

Write a C# program to compare any two strings.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**string.Equals():** In C#, *Equals(String, String)* is a String method. It is used to determine whether two String objects have the same value or not. Basically, it checks for equality. If both strings have the same value, it returns true otherwise returns false.

**Syntax :** bool string.Equals(string str1, string str2)

Explanation: This method will take the two parameters in the form of a string object and check them for equality. After checking, this method will return Boolean values. The return value type of this method is System.Boolean This method will return true if the value of str1 is the same as the value of str2 otherwise, false. If both str1 and str2 are null, then the method will return true.

**Program:**

using System;

class Program{

static void Main(string[] args){

string str1 = "cat";

string str2 = "dog";

string str3 = "cat";

if (String.Equals(str1, str2)){

Console.WriteLine($"{str1} and {str2} have same value.");

} else{

Console.WriteLine($"{str1} and {str3} are different.");

}}}

**Input / Output:**

**cat and dog are different.**

**cat and cat have same value.**

**Conclusion:**

So after doing the above study we have understood the concept of string comparison in C#.

**Experiment No. 2**

**Objective:**

Write a C# program to add, subtract, multiply and divide two numbers.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**Arithmetic Operators:**

Basic Arithmetic Operators in C#, include the following –

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **Operator** |  |  | **Description** |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| + | |  | Adds two operands | | |
|  | |  |  | | |
| - | |  | Subtracts the second operand from the first | | |
|  | |  |  | | |
| \* | |  | Multiplies both operands | | |
|  | |  |  | | |
| / | |  | Divides the numerator by de-numerator | | |
|  | |  |  | | |
| % | |  | Modulus Operator and remainder of after an integer division | | |
|  | |  |  | | |
| ++ | |  | Increment operator increases integer value by one | | |
|  | |  |  | | |
| -- | |  | Decrement operator decreases integer value by one | | |
|  |  |  |  |  |  |

To add, use the Addition Operator −

num1 + num2;

In the same way, it works for Subtraction, Multiplication, Division, and other operators.

**Program:**

using System;

class Program{

static void Main(string[] args){

float Add(float a, float b){

return a + b;

}

float Subtract(float a, float b){

return a - b;

}

float Multiply(float a, float b){

return a \* b;

}

float Divide(float a, float b){

return a / b;

}

float m = 10;

float n = 20;

Console.WriteLine($"{m} + {n} = " + Add(m, n));

Console.WriteLine($"{m} - {n} = " + Subtract(m, n));

Console.WriteLine($"{m} \* {n} = " + Multiply(m, n));

Console.WriteLine($"{m} / {n} = " + Divide(m, n));

}}

Input / Output:

10+20=30

10 -20=-10

10 \*20=200

10 /20=0.5

**Conclusion:**

So after doing the above study we have understood the concept of basic arithmetic operations in C#.

**Experiment No. 3**

**Objective:**

Write a C# program to find the smallest among three numbers using the if-else-if ladder statement.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**if-else-if ladder:**

The if-else-if ladder statement executes one condition from multiple statements. The execution starts from top and checks for each if condition. The statement of if block will be executed which evaluates to be true. If none of the if conditions evaluates to be true then the last else block is evaluated.

**Syntax:**

if(condition1) {

// code to be executed if condition1 is true

}

else if(condition2){

// code to be executed if condition2 is true

}

else if(condition3){

// code to be executed if condition3 is true

}

else{

// code to be executed if all the conditions are false

}

Program:

using System;

class Program{

static void Main(string[] args){

int a = 5, b = 7, c = 10;

if (a <= b && a <= c)

Console.WriteLine(a + " is the smallest"); else if (b <= a && b <= c)

Console.WriteLine(b + " is the smallest"); else

Console.WriteLine(c + " is the smallest");

}}

Input / Output:

7 is the smallest

**Conclusion:**

So after doing the above study we have understood the concept of if-else-if ladder statement in C#.

**Experiment No. 4**

**Objective:**

Write a C# program to find the sum of numbers from 5 to 100.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**for loop:**

Looping in a programming language is a way to execute a statement or a set of statements multiple times depending on the result of the condition to be evaluated to execute statements.

for loop has similar functionality as while loop but with different syntax. for loops are preferred when the number of times loop statements are to be executed is known beforehand. The loop variable initialization, condition to be tested, and increment/decrement of the loop variable is done in one line in for loop thereby providing a shorter, easy to debug structure of looping.

**Syntax:**

for (variable initialization ; testing condition; increment / decrement)

{

// statements to be executed

}

**Program:**

using System;

class Program{

static void Main(string[] args){

int sum = 0;

for(int i = 5; i <= 100; i++){

sum = sum + i;

}

Console.WriteLine("Sum of numbers from 5 to 100 is " + sum);

}}

Input / Output:

Sum of numbers from 5 to 100 is 5040

Conclusion:

So after doing the above study we have understood the concept of for loop statement in C#.

**Experiment No. 5**

**Objective:**

Write a C# program to print the following pattern.

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**for loop and nested loops:**

Looping in a programming language is a way to execute a statement or a set of statements multiple times depending on the result of the condition to be evaluated to execute statements.

for loop has similar functionality as while loop but with different syntax. for loops are preferred when the number of times loop statements are to be executed is known beforehand. The loop variable initialization, condition to be tested, and increment/decrement of the loop variable is done in one line in for loop thereby providing a shorter, easy to debug structure of looping.

**Syntax:**

for (variable initialization ; testing condition; increment / decrement){

// statements to be executed

}

Nested Loops:

When loops are present inside the other loops, it is known as nested loops.

**Program:**

using System;

class Program{

static void Main(string[] args){

for(int i = 5; i >= 1; i--){

for (int j = 1; j <= i; j++){

Console.Write("\*");

}

Console.WriteLine("");

}}}

Input / Output:

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

Conclusion:

So after doing the above study we have understood the concept of nested for loop statement in C#.

**Experiment No. 6**

**Objective:**

Write a C# program to find the sum of 10 numbers in an array by using foreach loop.

**Technology:**

Windows OS, Visual Studio

**Theory and concept: foreach loop:**Looping in a programming language is a way to execute a statement or a set of statements multiple numbers of times depending on the result of a condition to be evaluated. The resulting condition should be true to execute statements within loops.The foreach loop is used to iterate over the elements of the collection. The collection may be an array or a list. It executes for each element present in the array.

* Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
* In the loop body, you can use the loop variable you created rather than using an indexed array element.

**Syntax:**

foreach(data\_type var\_name in collection\_variable){

// statements to be executed

}

**Program:**

using System;

class Program{

static void Main(string[] args){

int[] numbers = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 }; int sum = 0;

foreach(int num in numbers){

sum = sum + num;

}

Console.WriteLine("Sum of numbers in an array is " + sum);

}}

Input / Output:

Sum of numbers in an array is 550

Conclusion: So after doing the above study we have understood the concept of foreach loop statement in C#.

**Experiment No .7**

**Objective:** Write a C# program to find the maximum and minimum number from an array.

**Technology:** Windows OS, Visual Studio

**Theory and concept:**

**Arrays:** An array is a group of like-typed variables that are referred to by a common name. And each data item is called an element of the array. The data types of the elements may be any valid data type like char, int, float, etc. and the elements are stored in a contiguous location. Length of the array specifies the number of elements present in the array. In [C#](https://www.geeksforgeeks.org/introduction-to-c-sharp/) the allocation of memory for the arrays is done dynamically. And arrays are kind of objects, therefore it is easy to find their size using the predefined functions. The variables in the array are ordered and each has an index beginning from 0.

**Program:**

using System;

class Program{

static void Main(string[] args){

int max, min, n;

Console.WriteLine("Input the number of elements to be stored in the array :");

n = Convert.ToInt32(Console.ReadLine());

int[] numbers = new int[n];

Console.WriteLine("Input {0} elements in the array :", n);

for (int i = 0; i < n; i++){

Console.Write("element - {0} : ", i);

numbers[i] = Convert.ToInt32(Console.ReadLine());

}

max = numbers[0];

min = numbers[0];

for (int i = 1; i < n; i++){

if (numbers[i] > max){

max = numbers[i];

}if (numbers[i] < min){

min = numbers[i];

}}

Console.WriteLine("Maximum element is : {0}", max);

Console.WriteLine("Minimum element is : {0}", min);}}

Input / Output:

Input the number of elements to be stored in the array :

5

Input 5 elements in the array :

element - 0 : 2

element - 1 : 10

element - 2 : 5

element - 3 : 15

element - 4 : 50

Maximum element is : 50

Minimum element is : 2

Conclusion:

So after doing the above study we have understood the concept of arrays in C#.

**Experiment No. 8**

**Objective:**

Write a C# program to sort the names of 10 persons in alphabetical order.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**Array.Sort() method:**

An array is a group of like-typed variables that are referred to by a common name. And each data item is called an element of the array. Arranging the array’s elements from largest to smallest is termed as sorting the array in descending order.

Array.Sort() method sorts an array in ascending order.

**Program:**

using System;

class Program{

static void Main(string[] args){

string[] names = new string[]{

"Ram",

"Hari",

"Sita",

"Abhi",

"Binita",

"Sapana",

"Krishna",

"Bibek",

"Khima",

"Ramesh"};

Array.Sort(names);

Console.WriteLine("Sorted names are:"); foreach (string name in names) {

Console.WriteLine(name);

}}}

Input / Output:

Sorted names are:

Abhi

Bibek

Binita

Hari

Khima

Krishna

Ram

Ramesh

Sapana

Sita

Conclusion:

So after doing the above study we have understood the concept of Array.sort() method in C#.

**Experiment No. 9**

**Objective:**

Write a C# program to add any two matrices.

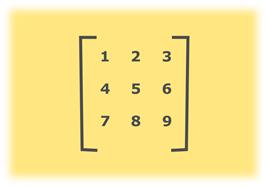
**Technology:**

Windows OS, Visual Studio

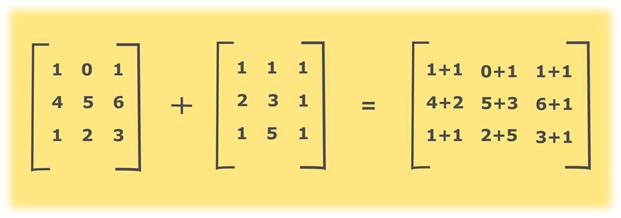
**Theory and concept:**

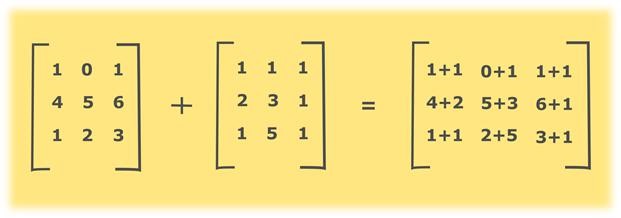
Matrix:

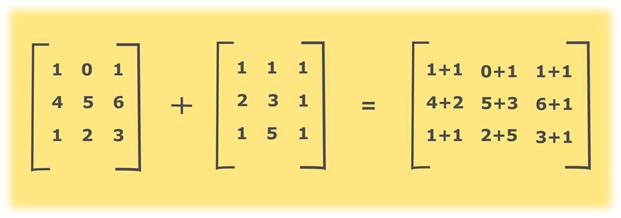
Matrix is a rectangular two-dimensional array of numbers arranged in rows and columns. A matrix with m rows and n columns can be called as m × n matrix. Individual entries in the matrix are called element and can be represented by aij which suggests that the element a is present in the ith row and jth column.



Two matrices A and B can be added if and only if they have same dimensions that are, the same number of rows and columns. It is not possible to add a 2 × 3 matrix with a 3 × 2 matrix. Addition of two matrices can be performed by adding their corresponding elements as

(A + B)ij= Aij + Bij





Program:

using System;

class Program{

static void Main(string[] args){

int i, j, n;

int[, ] arr1 = new int[20, 20];

int[, ] arr2 = new int[20, 20];

int[, ] arr3 = new int[20, 20];

// setting matrix row and columns size n = 3;

Console.Write("Enter elements in the first matrix:\n"); for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

arr1[i, j] = Convert.ToInt32(Console.ReadLine());

}}

Console.Write("Enter elements in the second matrix:\n"); for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

arr2[i, j] = Convert.ToInt32(Console.ReadLine());

}}

Console.Write("\nFirst matrix is:\n"); for (i = 0; i < n; i++) {

Console.Write("\n");

for (j = 0; j < n; j++) { Console.Write("{0}\t", arr1[i, j]);

}}

Console.Write("\nSecond matrix is:\n"); for (i = 0; i < n; i++) {

Console.Write("\n");

for (j = 0; j < n; j++) { Console.Write("{0}\t", arr2[i, j]);

}}

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

arr3[i, j] = arr1[i, j] + arr2[i, j];

}}

Console.Write("\nAdding two matrices: \n"); for (i = 0; i < n; i++) {

Console.Write("\n");

for (j = 0; j < n; j++) { Console.Write("{0}\t", arr3[i, j]);

}}

Console.Write("\n\n");

}}

Input / Output:

Enter elements in the first matrix:

2

2

2

2

2

2

2

2

2

Enter elements in the second matrix:

2

2

2

2

2

2

2

2

2

First matrix is:

2 2 2

2 2 2

2 2 2

Second matrix is:

2 2 2

2 2 2

2 2 2

Adding two matrices:

4 4 4

4 4 4

4 4 4

Conclusion:

So after doing the above study we have understood the concept of matrix addition using two dimensional array in C#.

**Experiment No. 10**

**Objective:**

Write a C# program to overload unary operators.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**Operator overloading:**

The concept of overloading a function can also be applied to operators. Operator overloading gives the ability to use the same operator to do various operations. It provides additional capabilities to C# operators when they are applied to user-defined data types. It enables to make user-defined implementations of various operations where one or both of the operands are of a user-defined class. Only the predefined set of [C#](https://www.geeksforgeeks.org/introduction-to-c-sharp/) operators can be overloaded. To make operations on a user-defined data type is not as simple as the operations on a built-in data type. To use operators with user-defined data types, they need to be overloaded according to a programmer’s requirement. An operator can be overloaded by defining a function to it. The function of the operator is declared by using the operator keyword.

**Syntax :**

access specifier className operator Operator\_symbol (parameters) {

// Code

}

Note : Operator overloading is basically the mechanism of providing a special meaning to an ideal C# operator w.r.t. a user-defined data type such as structures or classes.

**Program:**

using System;

class Calculator{

public int number1, number2;

public Calculator(int num1, int num2){

number1 = num1;

number2 = num2;

}

// Function to perform operation

// By changing sign of integers

public static Calculator operator -(Calculator c1){

c1.number1 = -c1.number1;

c1.number2 = -c1.number2;

return c1;

}

// Function to print the numbers public void Print(){

Console.WriteLine("Number1 = " + number1); Console.WriteLine("Number2 = " + number2);

}}

class Program{

static void Main(string[] args){

// using overloaded - operator

// with the class object

Calculator calc = new Calculator(15, -25);

calc = -calc;

// To display the result calc.Print();

}

**Input / Output:**

Number1 = -15

Number2 = 25

Conclusion:

So after doing the above study we have understood the concept of operator overloading in C#.

**Experiment No. 11**

**Objective:**

Write a C# program to perform method overloading.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**Method overloading:**

*Method Overloading* is the common way of implementing polymorphism. It is the ability to redefine a function in more than one form. A user can implement function overloading by defining two or more functions in a class sharing the same name. C# can distinguish the methods with different method signatures. i.e. the methods can have the same name but with different parameters list (i.e. the number of the parameters, order of the parameters, and data types of the parameters) within the same class.

* Overloaded methods are differentiated based on the number and type of the parameters passed as arguments to the methods.
* You can not define more than one method with the same name, Order and the type of the arguments. It would be a compiler error.
* The compiler does not consider the return type while differentiating the overloaded method. But you cannot declare two methods with the same signature and different return type. It will throw a compile-time error. If both methods have the same parameter types, but different return type, then it is not possible.

**Program:**

using System;

class Program{

// adding two integer values. public int Add(int a, int b){

int sum = a + b; return sum;

}

// adding three integer values.

public int Add(int a, int b, int c){

int sum = a + b + c;

return sum;

}

static void Main(string[] args){

// Creating Object

Program ob = new Program();

int sum1 = ob.Add(1, 2);

Console.WriteLine("sum of the two integer value : " + sum1);

int sum2 = ob.Add(1, 2, 3);

Console.WriteLine("sum of the three integer value : " + sum2);

}}

**Input / Output:**

sum of the two integer value : 3

sum of the three integer value : 6

**Conclusion:**

So after doing the above study we have understood the concept **of method overloading** in C#.

**Experiment No. 12**

**Objective:**

Write a C# program to perform method overriding.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**Method Overriding:**

Method Overriding is a technique that allows the invoking of functions from another class (base class) in the derived class. Creating a method in the derived class with the same signature as a method in the base class is called as method overriding.

**Program:**

using System;

class BaseClass{

// show() is 'virtual' here public virtual void show(){

Console.WriteLine("Base class");

}}

class Derived : BaseClass{

//'show()' is 'override' here

public override void show(){

Console.WriteLine("Derived class");

}}

class Program{

static void Main(string[] args){

BaseClass obj;

// 'obj' is the object of class 'baseClass' obj = new BaseClass();

// it invokes 'show()' of class 'baseClass' obj.show();

// the same object 'obj' is now the object of class 'derived' obj = new Derived();

// it invokes 'show()' of class 'derived'

obj.show();

}}

Input / Output:

Base class

Derived class

Conclusion: So after doing the above study we have understood the concept of method overriding in C#.

**Experiment No. 13**

**Experiment No. 13**

**Objective:**

Write a C# program to demonstrate abstraction using abstract class.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**Abstraction:**

Abstraction in C# is the process to hide the internal details and showing only the functionality. The abstract modifier indicates the incomplete implementation. The keyword abstract is used before the class or method to declare the class or method as abstract.

Program:

using System;

// declare class 'AreaClass' as abstract abstract class AreaClass{

// declare method 'Area' as abstract

abstract public int Area();

}

// class 'AreaClass' inherit in child class 'Square' class Square : AreaClass{

int side = 0;

// constructor public Square(int n){

side = n;

}

// the abstract method 'Area' is overridden here public override int Area(){

return side \* side;

}}

class Program{

static void Main(string[] args){

Square s = new Square(6); Console.WriteLine("Area = " + s.Area());

}}

Input / Output:

Area = 36

Conclusion:

So after doing the above study we have understood the concept of abstraction, abstract method and abstract class in C#.

**Experiment No. 14**

**Objective:**

Write a C# program to demonstrate interface in C#.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**Interface:**

Like a class, ***Interface*** can have methods, properties, events, and indexers as its members. But interfaces will contain only the declaration of the members. The implementation of the interface’s members will be given by class who implements the interface implicitly or explicitly.

* Interfaces specify what a class must do and not how.
* Interfaces can’t have private members.
* By default all the members of Interface are public and abstract.
* The interface will always defined with the help of keyword ‘***interface***‘.
* Interface cannot contain fields because they represent a particular implementation of data.
* *Multiple inheritance* is possible with the help of Interfaces but not with classes.

Program:

using System;

public interface Drawable{

void draw();

}

public class Rectangle : Drawable{

public void draw(){

Console.WriteLine("drawing rectangle...");

}}

public class Circle : Drawable{

public void draw(){

Console.WriteLine("drawing circle...");

}}

class Program{

static void Main(string[] args){

Drawable d;

d = new Rectangle();

d.draw();

d = new Circle();

d.draw();

}}

Input / Output:

drawing rectangle...

drawing circle...

Conclusion:

So after doing the above study we have understood the concept of interfaces in C#.

**Experiment No. 15**

**Objective:**

Write a C# program to demonstrate Delegates in C#.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**Delegate:**

A delegate is an object which refers to a method or you can say it is a reference type variable that can hold a reference to the methods. Delegates in C# are similar to the function pointer in C/C++. It provides a way which tells which method is to be called when an event is triggered.

**Important Points About Delegates:**

* Provides a good way to encapsulate the methods.
* Delegates are the library class in the System namespace.
* These are the type-safe pointer of any method.
* Delegates are mainly used in implementing the call-back methods and events.
* Delegates can be chained together as two or more methods can be called on a single event.
* It doesn’t care about the class of the object that it references.
* Delegates can also be used in “anonymous methods” invocation.
* Anonymous Methods(C# 2.0) and Lambda expressions(C# 3.0) are compiled to delegate types in certain contexts. Sometimes, these features together are known as anonymous functions.

**Declaration of Delegates**

Delegate type can be declared using the **delegate** keyword. Once a delegate is declared, the delegate instance will refer to and call those methods whose return type and parameter list match the delegate declaration.

**Syntax:**

[modifier] delegate [return\_type] [delegate\_name]([parameter\_list]);

Program:

using System;

public class Program{

// Declaring the delegates

// "addnum" and "subnum" are two delegate names public delegate void addnum(int a, int b); public delegate void subnum(int a, int b);

// method "sum"

public void sum(int a, int b){

Console.WriteLine("(100 + 40) = {0}", a + b);

}

// method "subtract"

public void subtract(int a, int b){

Console.WriteLine("(100 - 60) = {0}", a - b);

}

// Main Method

public static void Main(String[] args)

{

// creating object "obj" of class "Geeks" Program obj = new Program();

// instantiating the delegates

addnum del\_obj1 = new addnum(obj.sum); subnum del\_obj2 = new subnum(obj.subtract);

// pass the values to the methods by delegate object del\_obj1(100, 40);

del\_obj2(100, 60);

}}

**Input / Output:**

**(100 + 40) = 140**

**(100 - 60) = 40**

**Conclusion:** So after doing the above study we have understood the concept **of delegate** in C#.

**Experiment No. 16**

**Objective:**

Write a C# program to demonstrate the use of ADO.NET to work with a SQL server Database.

**Technology:**

Windows OS, Visual Studio

**Theory and concept:**

**ADO.NET** is a set of classes (a framework) to interact with data sources such as databases and XML files. ADO is the acronym for ActiveX Data Objects. It allows us to connect to underlying data or databases. It has classes and methods to retrieve and manipulate data.

The following are a few of the .NET applications that use ADO.NET to connect to a database, execute commands, and retrieve data from the database.

* ASP.NET Web Applications
* Console Applications
* Windows Applications.

**ADO.NET architecture:**

There are the following two types of connection architectures:

1. **Connected architecture:** the application remains connected with the database throughout the processing.
2. **Disconnected architecture:** the application automatically connects/disconnects during the processing. The application uses temporary data on the application side called a DataSet.

**Classes in ADO.NET:**

* 1. Connection Class
  2. Command Class
  3. DataReader Class
  4. DataSet.Class
  5. DataAdaptor Class

1. **Connection Class**

In ADO.NET, we use these connection classes to connect to the database. These connection classes also manage transactions and connection pooling.

**2. Command Class**

The Command class provides methods for storing and executing SQL statements and Stored Procedures. The following are the various commands that are executed by the Command Class.

* + **ExecuteReader:** Returns data to the client as rows. This would typically be an SQL select statement or a Stored Procedure that contains one or more select statements. This method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports and so forth.
  + **ExecuteNonQuery:** Executes a command that changes the data in the database, such as an update, delete, or insert statement or a Stored Procedure that contains one or more of these statements. This method returns an integer that is the number of rows affected by the query.
  + **ExecuteScalar:** This method only returns a single value. This kind of query returns a count of rows or a calculated value.
  + **ExecuteXMLReader:** (SqlClient classes only) Obtains data from an SQL Server 2000 database using an XML stream. Returns an XML Reader object.

1. **DataReader Class** The DataReader is used to retrieve data. It is used in conjunction with the Command class to execute an SQL Select statement and then access the returned rows.

**4. DataSet Class** The DataSet is the heart of ADO.NET. The DataSet is essentially a collection of **DataTable** objects. In turn, each object contains a collection of **DataColumn** and DataRow objects. The **DataSet** also contains a **Relations collection** that can be used to define relations among Data Table Objects.

**5. DataAdapter Class** The DataAdapter is used to connect DataSets to databases. The DataAdapter is most useful when using data-bound controls in Windows Forms, but it can also be used to provide an easy way to manage the connection between your application and the underlying database tables, views, and Stored Procedures.

**Program:**

using System.Data.SqlClient;

namespace AdoNetExample{

class Program{

static void Main(string[] args){

new Program().FetchData();

}

public void FetchData(){

SqlConnection con = null;

try{

// Creating Connection

con = new SqlConnection("Data

Source=localhost\\SQLEXPRESS;Initial Catalog=ado\_db;Integrated

Security=True");

// writing sql query

SqlCommand cm = new SqlCommand("Select \* from students",con);

// Opening Connection con.Open();

// Executing the SQL query SqlDataReader sdr = cm.ExecuteReader();

// Iterating Data

while (sdr.Read()){

Console.WriteLine(sdr["id"] + " " + sdr["name"] + "

" + sdr["email"]); // Displaying Record

}}

catch (Exception e){

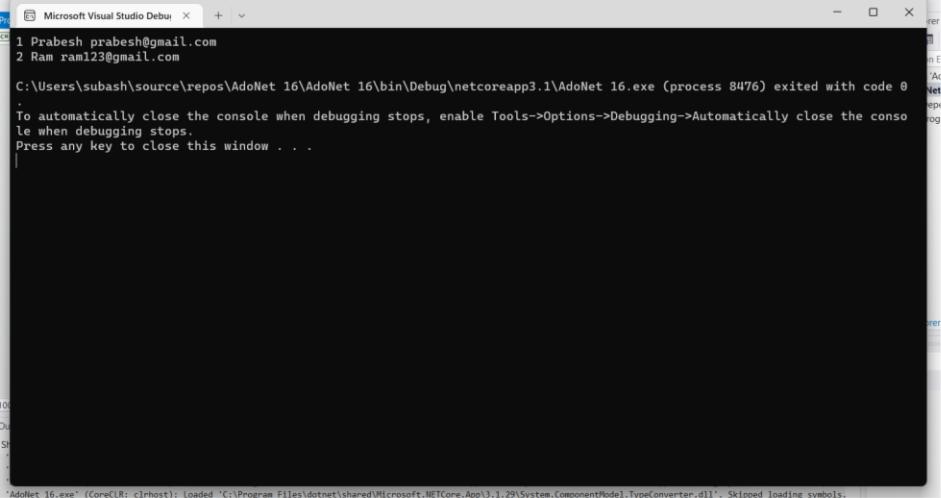
Console.WriteLine("OOPs, something went wrong.\n" + e);

}

// Closing the connection finally{

con.Close();

}}}

**Input / Output:**

**Conclusion:**

So after doing the above study we have understood the concept **of ADO.NET** in C#.