

C# best practices

What is .NET?

.NET refers to the technology featuring strong framework, product and services that was incorporated by Microsoft Corporation for the first time in year 2002.

Features of .NET

- ✓ It is a free, cross-platform and open source developer platform for building, deploying and running applications.
- ✓ In .NET, we can use multiple languages, editors and libraries to develop applications for web, mobile, desktop, gaming and IoT.
- ✓ .NET comprises of implementations like .NET Core, .NET Framework and Xamarin

Languages

- **C# is a simple, modern, object-oriented and type-safe programming language.**
- F# is a cross-platform, open-source, functional and imperative programming language.
- Visual Basic is an approachable language with a simple syntax for building type-safe, object-oriented apps.

Cross Platform

- **.NET Core** is a cross-platform .NET implementation for development of websites, services, and console apps that runs on Windows, Linux, and MacOS.
- **.NET Framework** supports websites, services, desktop apps, and other apps which runs only on Windows.
- **Xamarin / Mono** is a .NET implementation for running apps on all the major mobile operating systems.

.NET Framework

.NET Framework is an implementation which provides the excellent runtime environment in the form of CLR that supports development and execution of the various next generation **.NET** applications and services.

.NET Framework History

Releases

Version	Release date	CLR Version
4.5	2012-10-09	4
4	2010-04-12	4
3.5	2007-11-19	2.0
3.0	2006-11-21	2.0
2.0	2006-02-17	2.0

Version	Release date	CLR Version
4.8.1	2022-08-09	4
4.8	2019-04-18	4
4.7.2	2018-04-30	4
4.7.1	2017-10-17	4
4.7	2017-04-11	4
4.6.2	2016-08-02	4
4.6.1	2015-11-30	4
4.6	2015-07-29	4
4.5.2	2014-05-05	4
4.5.1	2014-01-15	4

.NET Core Releases

Version	Original release date
.NET 8	November 14, 2023
.NET 7	November 8, 2022
.NET 6	November 8, 2021
.NET 5	November 10, 2020
.NET Core 3.1	December 3, 2019
.NET Core 3.0	September 23, 2019
.NET Core 2.2	December 4, 2018
.NET Core 2.1	May 30, 2018
.NET Core 2.0	August 14, 2017
.NET Core 1.1	November 16, 2016
.NET Core 1.0	June 27, 2016

Components of .NET Framework

The major components of .NET framework are –

- 1) The **common language runtime (CLR)**, which is the main execution engine with an in-built services that support running of the applications
- 2) The **.NET framework class libraries** which provides a library of tested and re-usable codes that can be imported in any applications.
- 3) The **other integral components** are console application, windows forms, asp.net web forms and web services, MVC, Web API, ADO.NET and XML, LINQ, windows communication foundation, windows presentation foundation, Task-Parallelism, Entity Framework etc.

.NET Product

It mainly comprises of an integrated suite in the form of Microsoft Visual Studio Development IDE including tools that supports the coding, designing, testing, debugging and deployment of the .NET applications.

We can either use any one of the below specified product for coding and development

A) Visual Studio Code

B) Visual Studio 2019 or higher

Console Application using VS Code

To create new solution → open visual studio code → open terminal and type the command as –

- PS C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples> `dotnet new sln -o Con_App_Project1`
The template "Solution File" was created successfully.

```
▼ C#_EXAMPLES  
  > Con_App_Project1
```

To create a new console application inside the solution

- PS C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples> `cd Con_App_Project1`
- PS C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1> `dotnet new console -o Project1`
The template "Console App" was created successfully.

Processing post-creation actions...

Restoring C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1\Project1\Project1.csproj:

Determining projects to restore...

Restored C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1\Project1\Project1.csproj (in 135 ms).

Restore succeeded.

```
▼ C#_EXAMPLES  
  ▼ Con_App_Project1  
    > Project1  
    ≡ Con_App_Project1.sln
```

Hello World

Edit “Hello World” code in Program.cs

As

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

or

```
using System;
```

```
namespace Project1
```

```
{
```

```
    class Program
```

```
    {
```

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

```
        {
```

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

```
            Console.Read();
```

```
        }
```

```
    }
```

```
}
```

To add console application into project and build the solution

```
PS C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1> dotnet sln add Project1\Project1.csproj
Project `Project1\Project1.csproj` added to the solution.
PS C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1> dotnet build
MSBuild version 17.7.3+8ec440e68 for .NET
Determining projects to restore...
All projects are up-to-date for restore.
Project1 -> C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1\Project1\bin\Debug\net7.0\Project1.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:02.86
```

To run console application project

```
PS C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1> dotnet restore
Determining projects to restore...
All projects are up-to-date for restore.
PS C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1> cd Project1
PS C:\Users\Dell\Desktop\Master_Class_UI\C#_Examples\Con_App_Project1\Project1> dotnet run
Hello World
```

Data types in C#

Value types

byte, short, int, long, char, float, double, decimal, enum, struct

The variables of the value types allocate its position in memory stack and keep hold of the value.

Example --- `int num = 10;`

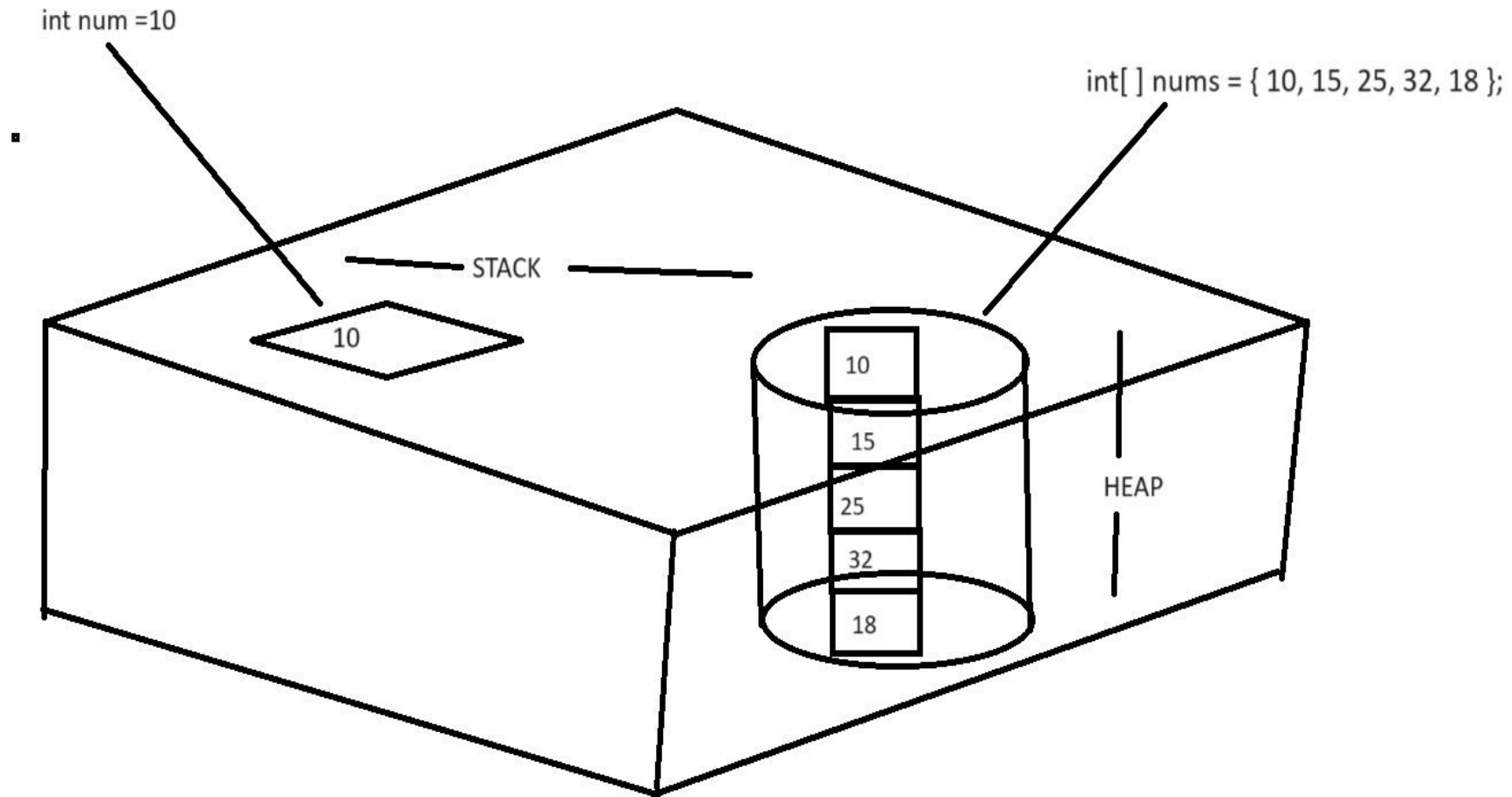
Reference types

Array, class, interface, event, delegate

The variables of the reference types take its position in memory stack which can keep reference of the address of object or resources containing data that is stored within memory heap.

Example --- `int[] nums = new int[10];`

Value Type and Reference Type



Operators in C#

Arithmetic Operator : +, -, *, /, %

Comparison or Relational Operator : >, <, >=, <=, ==, !=

Logical Operator: &&, ||, !

Assignment Operator : =, [Arithmetic-Assignment] +=, -=, *=, /=, %=

Unary Operator : -, +, ++, --

Ternary Operator ?:

(boolean_expr) ? True_Expr: False_expr

```
int x=10, y=5, z=0;
```

```
z = ((3*x) > (5*y))?(5*y):(3*x);
```

Exercise

Write a program using C# for .NET Core Application that accept and display the details like person's name, age and income tax paid in last financial year

Input Person Name:

Somenath Mukherjee

Input Age and Income Tax Paid in last FY:

43

5528

Name: Somenath Mukherjee, **Age:** 43, **Tax:** 5528

Exercises

Write a program in C# that accept the data for Length and breadth and there by calculate and display the area, perimeter and diagonal of a rectangle.

Input Length and Breadth :

12

5

Length : 12, Breadth : 5

Area : 60, Perimeter : 34, Diagonal : 13.000

Write a program using C# for .NET Core Application that **prints today's date and the date after 10 years from Today.**

```
C:\> D:\Somenath\Education\Dotnet Core\Exercises\WebAPIAppSoln\C
Current Date = 17 January 2023
Date after 10 years= 17 January 2033
```

Exercise

Write a program using C# for .NET Core Application that **accepts two numbers as byte and perform arithmetic operations.**

Input First Number

10

Input Second Number

5

Add: 15, Difference: 5, Product: 75, Remainder: 0, Division: 2

Structure in C#

Structure is a value type and maintains a collection of variables of different data types under a single unit.

Defining Structure:

Using struct keyword one can define the structure consisting of different data types in it. A structure can also contain constructors, constants, fields, methods, properties, indexers etc.

Example

C# syntax defining structure with members, that maintains the values related to sector area

```
struct Sector
{
    public int radius;
    public int angleOfSector;
    public double sectorArea;
}
```

Exercise

Write a program in C#, that define the structure type Player with members Name, Country and Skill. The program must use the variable of Player to accept the details in members and display them.

Enter Player Name, Country and Skill

MS Dhoni

India

Wicketkeeper, Batsman

Name: MS Dhoni, Country: India, Skill: Wicketkeeper, Batsman

Write a program in C#, that accept the name of a person and gender and display output as

Input Name

[use ternary operator]

Rumela

Input Gender

Female

Hello Mrs Rumela

Enumeration in C#

Enumeration (or enum) is a value type in C#. It is mainly used to assign the names or string values to integral constants, that make a program easy to read and maintain.

```
enum Gender  
{  
    Male, Female  
}
```

Exercise

**Write a program in C# that define an enumeration type "WeekDay" --> Monday, Sunday
It must return the day of the week as given below.**

Input day number:

3

Wednesday

DateTime and TimeSpan properties

```
DateTime currentDate = DateTime.Now; // Current Date and Time
```

```
Console.WriteLine("Date {0:dd-MM-yyyy}", currentDate); // 13-06-2024
```

```
Console.WriteLine("Time {0:HH:mm:ss}", currentDate); // 10:50:55
```

```
Console.WriteLine("5 days later : {0}", currentDate.AddDays(5)); // 18-06-2024
```

```
Console.WriteLine("3 days ago : {0}", currentDate.AddDays(-3)); // 10-06-2024
```

```
Console.WriteLine("Date after 3 months : {0}", currentDate.AddMonths(3)); // 13-09-2024
```

```
Console.WriteLine("Day Name : {0}", currentDate.DayOfWeek); // Thursday
```

```
DateTime dateOfBirth = DateTime.ParseExact("10-22-1979", "MM-dd-yyyy", null);
```

```
TimeSpan ts = currentDate.Subtract(dateOfBirth); // Difference between 2 dates
```

```
long days = (long)ts.TotalDays;
```

```
◦ int years = (int)days / 365;
```

```
int months = (int)(days % 365) / 30;
```

```
Console.WriteLine("You are {0} years {1} months old", years, months);
```

Typecasting

Typecasting is the process of casting the type of variable into other type. It can be classified into two kinds –

a) Implicit Typecasting b) Explicit Typecasting

Implicit Typecasting

```
int num = 15;  
long l_num = num;
```

Explicit Typecasting

```
int first_num = 15, second_num = 4;  
double div_value = (double)first_num / second_num;
```

```
long l_num = 8;  
int num = (int)l_num;
```


Boxing and Unboxing

Boxing --> It is the process of wrapping the data of value type into an Object.

```
struct Employee
```

```
{
```

```
    public int empNo;
```

```
    public string empName;
```

```
}
```

```
Employee emp;
```

```
emp.empNo = 1101;
```

```
emp.empName = "Rajiv";
```

```
Object objEmp = emp;
```

```
// Boxing
```

Boxing and Unboxing

Unboxing --> It is the reverse process that explicitly convert object into data of original value type. During unboxing the data of the original type is extracted from the Object by using explicit type casting.

```
Employee emp;  
emp.empNo = 1101;  
emp.empName = "Rajiv";  
Object objEmp = emp;    // Boxing  
  
if(objEmp is Employee)  
{  
    Employee e1 = (Employee) objEmp;        // Unboxing  
    Console.WriteLine("Emp No : {0}, Name : {1}",e1.empNo, e1.empName)  
}
```

Control Statements

if statement

```
if (boolean_exp) {  
    program_statment;  
}
```

If - else statement

```
if (boolean-exp) {  
    program_statement_if_true;  
}  
else {  
    program_statement_if_false;  
}
```

multiple if else statement

```
if(boolean_expr1) {  
    program_statement1;  
}  
else if(boolean_expr2) {  
    program_statement2;  
}  
...  
else {  
    program_else_if_all_condition_fails;  
}
```

Exercise

**Write a program in C#, that check if the number is divisible by 5
[use if statement]**

Input Number

15

15 is divisible by 5

Input Number

24

24 is not divisible by 5

Exercise

**Write a program in C#, that check if the number is a perfect square
[use if – else statement]**

Input Number

36

36 is a perfect square

Input Number

10

10 is not a perfect square

Exercise

Write a program in C# that determine the profit or loss in business and also display calculated profit [Use multiple if – else statement]

Input Buying Price of the Product

100

Input Market Price of the Product

120

Input Discount %

10

Actual Selling Price

108

Profit in business, Rs 8

Input Buying Price of the Product

100

Input Market Price of the Product

110

Input Discount %

20

Actual Selling Price

88

Loss in business, Rs 12

Nested if – else (Exercise)

Write a program in C# that accept any number and check if it is positive, the program must also determine if the number is odd or even

Input any number

25

25 is positive

25 is odd

Input any number

32

32 is positive

32 is even

Input any number

-25

-25 is negative

Control Statement (switch – case)

```
variable = value2;  
switch(variable)  
{  
    case value1:  
        program_statement1;  
        break;  
    case value2:  
        program_statement2;  
        break;  
    case value3:  
        program_statement3;  
        break;  
    default:  
        program_statement_default;  
        break;  
}
```


Exercise

Write a program in C# that allows you to select the available fruit and know its price

[Hint: `enum Fruit { Apple=150, Guava=80, Orange=10 }]`

Enter fruit you wish to buy --- 1> Apple 2> Guava 3> Orange

Apple

Cost of Apple is Rs 150 per Kg

Enter fruit you wish to buy --- 1> Apple 2> Guava 3> Orange

3

Cost of Orange is Rs 10 per Piece

Enter fruit you wish to buy --- 1> Apple 2> Guava 3> Orange

Mango

Requested fruit is not available

Statement of loop / Iteration

1. for statement
2. while statement
3. do - while statement
4. foreach statement
(reading element value from collections or array)

for statement

```
for(initialization; boolean_expr; re-initialization +/-)
{
    program_statement;
}
```

```

2      5      8      11     14     17     20
for(int i = 2, d = 3; i <= 20; i += d){
    Console.WriteLine("{0,10}",i);
}
```

while statement

initialization;

```
while (boolean_Expr) {  
    program_Statementt;  
    re-initialization;  
}
```

2 5 8 11 14 17 20

```
int i = 2, d = 3;
```

```
while(i <= 20){
```

```
    Console.Write("{0,10}",i);
```

```
    ○ i +=d;
```

```
○ }
```

do - while statement

initialization;

- do{

 program_statement;

 re-initialization;

 } while(boolean_expr);

2 5 8 11 14 17 20

int i = 2, d = 3;

do{


 Console.Write("{0,10}",i);

- i +=d;

- } while(i <= 20);

Exercise

Write a program in C#, that accept the name of the student and the class standard in upper roman format. The program display all of the class standards available, the student's name and the class in which he/she is studying.

 Microsoft Visual Studio Debug Console

```
Enter Student's Name :
```

```
Somenath
```

```
Input Student's Class Standard
```

```
XI
```

```
Class standards available:
```

```
    I   II  III   IV    V   VI  VII VIII   IX    X   XI  XII
```

```
Somenath is studying in class standard XI or class 11
```

Exercises

Write a Program in C#, that find and display the sum of First N Natural Nos.

Sum = 1 + 2 + 3 + N [Use while loop]

Enter any number

7

1 + 2 + 3 + 4 + 5 + 6 + 7 = 28

Write a Program in C#, that displays all letters of English Alphabets in Capital Format [use do -while loop]

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Exercise

Write a program in C# that accept any two numbers and display the set of prime numbers between these two numbers.

Input First Number:

10

Input Last Number:

25

Set of prime numbers between 10 and 25 are

11 13 17 19 23

foreach statement

- `foreach (var item_var in array_name)`
 {
 `Console.WriteLine(item_var);`
 }

Example-

```
string[] places = { "Kolkata", "Bangalore", "Ahmedabad", "Hyderabad", "Bhopal" };  
    foreach(string place in places)  
    {  
        Console.WriteLine("{0}",place);  
    }
```

static

We can declare any variable or method inside a given class as static, if we want to access it as class level variable or class level method

static variable and method

```
static class WeightOnMoon
{
    public static double moonWeightConstant = 0.17;
    public static double GetWeightOnMoon(double weightOnEarth)
    {
        return weightOnEarth * moonWeightConstant;
    }
}
```

Exercise

Develop a program in C# that accept the value of radius and display the calculated area of the circle (restricted up to 3 places of decimal). The program must define a class “Circle” consisting of static method whose method signature is given as

```
public static void CalculateArea(int radius, ref double area)
```

 Microsoft Visual Studio Debug Console

```
Input the value for radius
```

```
15
```

```
Area of the Circle of radius 15 is 706.858
```

Exercise

Write a program in C#, that define the static class CubeRootFinder having static variable and static method as given below.

```
public static int Value = 0;  
public static string CubeRoot() {  
  
}
```

The program must use the static variable to accept an integer and static method to calculate and return the result of cube root of the number as string statement.

 Microsoft Visual Studio Debug Console

```
Input any value:
```

```
125
```

```
The cube root of 125 is 5
```

Method Parameters

We can use 4 kinds of parameters.

- value type parameter

- ref parameter

- out parameter

- params type parameter

value type parameter

```
public class MethodUtility
{
    // method with value type parameter
    public int Adder(int num1, int num2)
    {
        return num1 + num2;
    }
}
```

Program.cs

```
-----
Console.WriteLine("Input values for num1 and num2");
    int n1 = Int32.Parse(Console.ReadLine());
    int n2 = Int32.Parse(Console.ReadLine());

    MethodUtility obj = new MethodUtility();
    int sum = obj.Adder(n1, n2);
    Console.WriteLine("Sum of {0} and {1} is {2}",n1,n2,sum);
```

ref parameter

```
public class MethodUtility
{
    // method with ref parameter
    public void GetCircleArea(int radius, ref double area)
    {
        area = Math.PI * radius * radius;
    }
}
```

Program.cs

```
Console.WriteLine("Input Radius");
    int rad = Int32.Parse(Console.ReadLine());

    double area = 0;

    MethodUtility obj = new MethodUtility();
    obj.GetCircleArea(rad, ref area);
    Console.WriteLine("Area of {0} with radius {1} is
{2:0.000}", "Circle", rad, area);
```

out parameter

```
public class MethodUtility
{
    public void GetCircumference(int radius, out double
    circumference) // out type parameter
    {
        circumference = 2 * Math.PI * radius;
    }
}
```

Program.cs

```
Console.WriteLine("Input Radius");
    int rad = Int32.Parse(Console.ReadLine());

    double area = 0;
    double circumference;

    MethodUtility obj = new MethodUtility();
    obj.GetCircumference(rad, out circumference);
    Console.WriteLine("Perimeter of {0} with radius {1}
    is {2:0.000}", "Circle", rad, circumference);
```


params parameter

```
public class MethodUtility
```

```
{
```

```
public float GetAvgofParameters(params int[] nums)
```

```
{
```

```
    float avg = 0;
```

```
    int sum = 0;
```

```
    Array.ForEach(nums, n => sum += n);
```

```
    return avg = (float)sum / nums.Length;
```

```
}
```

```
}
```

Program.cs

```
Console.WriteLine("Method using params array .....");
```

```
MethodUtility obj = new MethodUtility();
```

```
float average = obj.GetAvgofParameters(8, 3, 11, 16, 5);
```

```
Console.WriteLine("Average of parameters 8,3,11,16,5  
is {0:0.00}", average);
```

Arrays in C#

Array can be defined as a collection of elements of similar datatype.

- `type[] array_var = new type[n_size];`

Example - `int[] nums = new int[5];`

`nums[0] = 15; nums[1] = 20; nums[2] = 25; nums[3] = 5; nums[4] = 10;`

By default, the first subscripted element of the array is having an index 0

Each element of an array identified by `array_var[index];`

nums	15	20	25	5	10
	0	1	2	3	4

Exercise

Write a program in C#, that allocate the results of factorials of each number between 1 and N in an array and display them accordingly.

Input any number

6

Factorials of all 6 numbers are

1	2	6	24	120	720
---	---	---	----	-----	-----

Array utility class extension method

Develop a program in C# that read the values of temperatures recorded at 5:30 am during a week as a single line input. The program displays the maximum and minimum value of the temperature recorded.

 Microsoft Visual Studio Debug Console

```
Enter temperature recorded during last week  
21.3,20,19.8,21.2,22.5,21.6,21.1  
Maximum Temperature:22.5  
Minimum Temperature:19.8
```

Array utility class extension method (guided)

```
Console.WriteLine("Enter temperature recorded during last week");
string[] tempvals = Console.ReadLine().Split(',');

float[] temps = new float[tempvals.Length];
temps = Array.ConvertAll(tempvals, t => float.Parse(t));
float max = temps[0], min = temps[0];
Array.ForEach(temps, t =>
{
    if(t>max)
        max = t;
    if(t<min)
        min = t;
});
Console.WriteLine($"Maximum Temperature:{max}");
Console.WriteLine($"Minimum Temperature:{min}");
```

Two dimensional array

Syntax for declaration of two-dimensional-array in C#

Datatype [,] identifier = new Datatype[rows, cols];

Example –

```
int[ , ] nums_2d = {  
    {5, 8, 3},  
    {15,4, 2},  
    {0, 5, 1}  
};
```

Exercise

Develop a program in C# that accept the name of the player and his scores in three test matches. The program must display all the scores of 3 test matches, total runs and average score per test match.

```
C:\> Microsoft Visual Studio Debug Console
Input Player Name
Sachin Tendulkar
Enter the scores of First Test
186
38
Enter the scores of Second Test
0
57
Enter the scores of Third Test
215
43
Performances of Sachin Tendulkar in 3 Tests
      Test      1st Inn      2nd Inn
First Test      186        38
Second Test       0        57
Third Test      215        43
Total runs scored in all three test matches: 539
Average runs scored in each test matches: 179.67
```

Jagged array

It refers to a special type of two dimensional array where the size of the row is known, but the size of columns of each row is determined during runtime.

```
int[ ][ ] jagged = new int[3][ ];
```

```
12  4  8  16
```

```
4   7  9  10  12
```

```
22  8
```


Write a program in C#, that create and fill the elements of jagged array and display the elements.

Input no of rows

3

Input no of values in first row

5

Fill up the elements

12

8

7

6

2

Input no of values in second row

3

Fill up the elements

23

11

18

Input no of values in third row

4

Fill up the elements

2

5

9

1

Elements in jagged array are

12 8 7 6 2

23 11 18

2 5 9 1

Object Oriented Programming

Object Oriented Programming refers to a kind of programming where the programmers can use the approach to write the program for the real life entity by defining the features for similar type of objects.

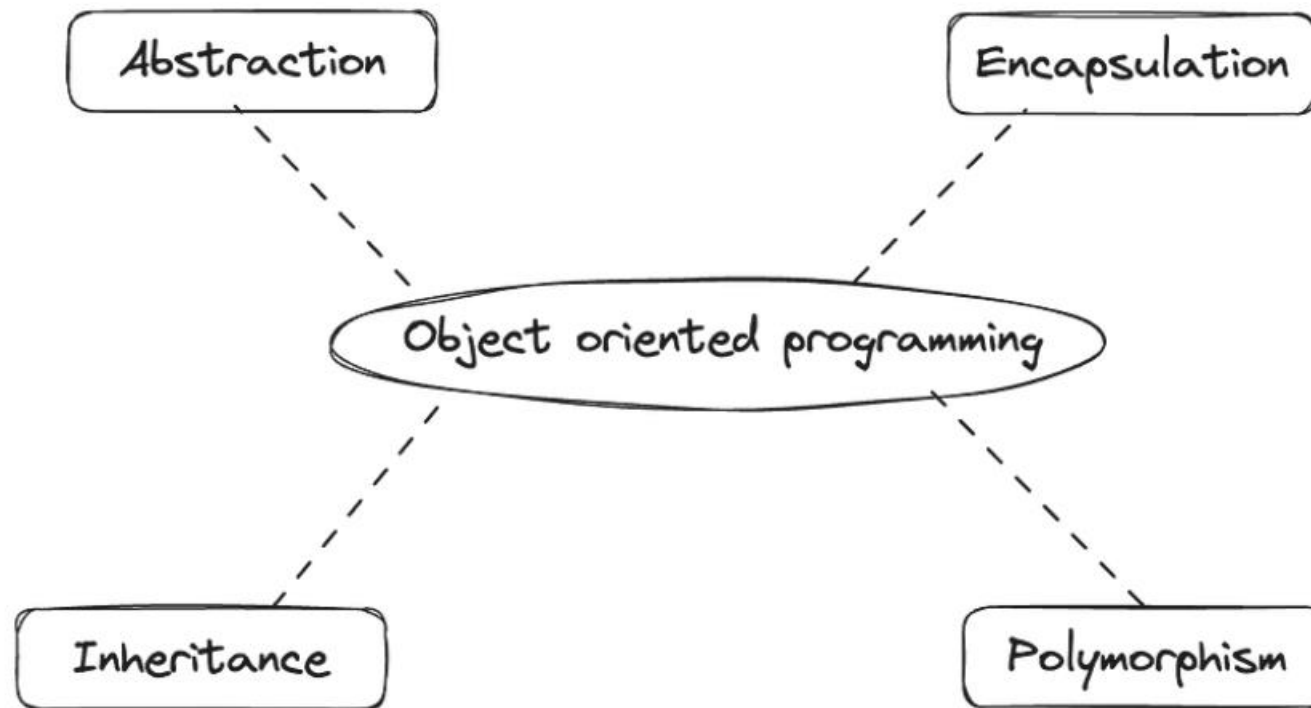
An object oriented program explain the concept about real life entity in the form of class definition, there by defining the attributes, properties and operational behaviors for each instances or the objects.

The object oriented programs are well implemented using the core principles like abstraction, encapsulation, inheritance and polymorphism.

Object Oriented Programming Principles

Abstraction: Show only the necessary things.

Encapsulation: Hides the Complexity.



Inheritance: Focus on Parent and Child Class relationships.

Polymorphism: An object can behave differently in more than one form at different levels.

Class and object

Class is a reference type as declared for an entity. It provide the blue print or the concept about the entity defining the features for similar kind of objects. A class generally consists of constructor, set of attributes or properties and operational behaviors (methods).

Object is an instance of the class that also has the physical state of existence and there by maintains the attributes or properties and perform operations using well defined methods.

Access Modifiers

There are five levels of access modifiers namely public, private, protected, internal, protected internal and private protected

	public	protected	internal	protected internal	private	private protected
Entire program	Yes	No	No	No	No	No
Containing class	Yes	Yes	Yes	Yes	Yes	Yes
Current assembly	Yes	No	Yes	Yes	No	No
Derived types	Yes	Yes	No	Yes	No	No
Derived types within current assembly	Yes	Yes	Yes	Yes	No	Yes

```

using System;
class Employee
{
    private int empId;                // member variables [encapsulated]
    private String empName;

    public Employee( ){ }             // default constructor
    public Employee(int empId, string empName) // parameterized constructor
    {
        this.empId = empId;           this.empName = empName;
    }

    public void Display() {           // member method [accessible]
        Console.WriteLine("Employee Id : {0}, Employee Name : {1}",empId, empName);
    }
}

class Program{
    public static void Main(string[] args)
    {
        Employee emp = new Employee(11001, "Raj Sundhar"); // instance creation and initialization
        emp.Display();
    }
}

```

Exercise

Write a program in C#, that define a class Player with three public members -- name, skill and country. The program must create an instance of the Player class and assign values in public member variables and display player's information.

 Microsoft Visual Studio Debug Console

```
Input player's name, country and skill
```

```
Virat Kohli
```

```
India
```

```
Batsman
```

```
Player's Information
```

```
Player's Name: Virat Kohli, Country: India, Skill: Batsman
```

Exercise

Write a program in C#, that define a class Player with three private member variables -- name, skill and country. The class should define a public method "Display()" to show player's information.

The program must create an instance of the Player class and there by assign values using constructor and display player's information by calling Display() method using instance.

 Microsoft Visual Studio Debug Console

```
Input player's name, country and skill
```

```
Virat Kohli
```

```
India
```

```
Batsman
```

```
Player's Information
```

```
Player's Name: Virat Kohli, Country: India, Skill: Batsman
```


Exercise

Write a program in C#, that define a class Player with three private members -- name, skill and country. The class should define the Properties as public to set and return data from the private members.

The program must create an instance of the Player class and there by use the properties to set and display player's information.

 Microsoft Visual Studio Debug Console

```
Input player's name, country and skill
```

```
Virat Kohli
```

```
India
```

```
Batsman
```

```
Player's Information
```

```
Player's Name: Virat Kohli, Country: India, Skill: Batsman
```

Exercise

Develop a program in C#, that define a class Player with three member variables – name, skill and country, all of them must be declared as private.

The class must define the properties with getter and setter in order to access the private members.

The program must create an instance of Player type and initialize the data by invoking 3 argument constructor and finally display the players details by calling Display().

 Microsoft Visual Studio Debug Console

```
Input player's name, country and skill
```

```
Virat Kohli
```

```
India
```

```
Batsman
```

```
Player's Information
```

```
Player's Name: Virat Kohli, Country: India, Skill: Batsman
```

Exercise

Develop a program in C# that define a class Player with three auto implemented properties.

The class must define 3 argument constructor to initialize data for an instance with properties using setters.

The class must override ToString() method to display the information of PlayerInfo type instance by calling getters of the properties.

In Program's Main method create and initialize the instance of Player and display it's details.

 Microsoft Visual Studio Debug Console

```
Input player's name, country and skill
```

```
Virat Kohli
```

```
India
```

```
Batsman
```

```
Player's Information
```

```
Player's Name: Virat Kohli, Country: India, Skill: Batsman
```

Exercise

Develop a program in C# that define a class Player with three auto implemented properties.

The class must define 3 argument constructor to initialize data for an instance with properties using setters.

The class must override ToString() method to display the information of Player type instance by calling getters of the properties.

In Program's Main method create and initialize multiple instances of Player and display their details.

Microsoft Visual Studio Debug Console

Number of Players

3

Input player's name, country and skill

Virat Kohli

India

Batsman

Input player's name, country and skill

Joe Root

England

Batsman

Input player's name, country and skill

Dale Steyn

South Africa

Bowler

All Player's Information:

Player's Name: Virat Kohli, Country: India, Skill: Batsman

Player's Name: Joe Root, Country: England, Skill: Batsman

Player's Name: Dale Steyn, Country: South Africa, Skill: Bowler

Array of objects

Array of objects in C# is just an array of object data as its value. By using array of objects, we can access the members (field and methods) of the class with each object.

The following syntax is used to declare an array of objects,
`class_name array_name[] = new class_name[SIZE];`

//creating array of objects

```
Student[] S = new Student[2];
```

//Initialising objects by defaults/inbuilt constructors

```
S[0] = new Student();
```

```
S[1] = new Student();
```

//Setting the values and printing first object

```
S[0].SetInfo("Harry", 101, 25);
```

```
S[0].printInfo();
```

//Setting the values and printing second object

```
S[1].SetInfo("Potter", 102, 27);
```

```
S[1].printInfo();
```

```
Student Record:
      Name      : Herry
      RollNo     : 101
      Age        : 25
Student Record:
      Name      : Potter
      RollNo     : 102
      Age        : 27
```

Polymorphism

Polymorphism is the term derived from greek words 'poly' (many) and 'morphos' (forms of behavior).

As per concept, any instance can expose the behaviors of the similar kind in multiple ways depending on the available information or the circumstances in which state the object is available, there by executing the operation.

Polymorpshism are available in two forms

- static polymorphism (use method overloading)

- dynamic polymorphism (use method overriding)

Method Overloading

Method Overloading is one of the useful concept, by virtue of which a class can provide multiple definition of the methods with same name but different signatures.

The signature of the methods may get differs as per no of arguments, types or order of specification in method declaration.

```
class Calculator
```

```
{  
    public static int Adder(int a, int b)  
    {  
        return a + b;  
    }
```

```
  
    public static int Adder(int a, int b, int c)  
    {  
        return a + b + c;  
    }
```

```
  
    public static float Adder(float a, int b)  
    {  
        return a + b;  
    }
```

```
  
    public static float Adder(int a, float b)  
    {  
        return a + b;  
    }  
}
```

Method Overloading

Program.cs

```
Console.WriteLine("10 + 5 = {0}", Calculator.Adder(10, 5));
```

```
Console.WriteLine("10 + 20 + 5 = {0}", Calculator.Adder(10, 20, 5));
```

```
Console.WriteLine("10.75 + 5 = {0}", Calculator.Adder(10.75f, 5));
```

```
Console.WriteLine("10 + 5.75 = {0}", Calculator.Adder(10, 5.75f));
```


Inheritance

Inheritance is one of the important concept of OOP. As per inheritance a given class can inherit the common set of attributes, properties and behaviors from it's parent class.

Inheritance are of two kinds --- single inheritance and multiple inheritance.

C# programs only support the use of single inheritance.

Single inheritance are available in many kinds --- single level, multi level and hierarchical level.

```
class AB
{
    declares attributes / properties / methods
}

class EF : AB
{
    re-use common attributes / properties / methods of AB
}
```

Inheritance [Single Level]

Inheritance

Inheritance [Multi Level]

```
class AB
{
    declares public/protected attributes / properties / methods
}
class EF : AB
{
    re-use common attributes / properties / methods of AB
}
class JK : EF
{
    re-use common attributes / properties / methods of EF and AB
}
```

Inheritance

Inheritance [Hierarchical Level]

```
class AB
{
    declares public/protected attributes / properties / methods
}

class EF : AB
{
    re-use common attributes / properties / methods of AB
}

class JK : AB
{
    re-use common attributes / properties / methods of AB
}
```

Benefits of inheritance

Inheritance provides two important benefits to the program of sub class.

The derived class can re-use the common set of properties or behaviors by inheriting them from it's parent.

The child class can remodel the concept of behaviors or properties by using the concept of Versioning or Overriding.

Versioning using 'new' keyword

Versioning is the concept by virtue of which a subclass can maintain the new version of properties or methods (by declaring them using "new" keyword) there by shadowing the inherited ones.

Example:

```
public class Cube : Square
{
    ...
    protected new int GetArea()
    {
        return 6 * base.GetArea();
    }
}
```

Exercise

Develop a program in C# that define the parent class Square and it's child class Cube with respective field, properties and constructor to initialize them.

The program must create the instance of Square and Cube classes respectively and invoke Area property to return the calculated area of type using the mechanism of inheritance and versioning whichever required.

```
D:\Somenath\Education\Dotnet C#>  
Select the shape :  
Square  
Cube  
Square  
Input the value of side  
10  
Area of Square is 100
```

```
D:\Somenath\Education\Dotnet Core\I>  
Select the shape :  
Square  
Cube  
Cube  
Input the value of side  
10  
Surface Area of Cube is 600
```

Overriding

Overriding is the useful concept, by virtue of which a child class gets the ability to override the definition of the property or method that it inherits from its base class with same name and signatures but with different context.

The base class must declare the property or method as 'virtual' or 'abstract' by setting the rule that subclass can override it in future. The subclass can override the definition of base class (virtual/abstract) method or property using 'override' keyword.

Exercise

Write a program in C#, that explain the concept of method overriding using the examples of Chair.

using System;

```
class Chair
{
    public virtual void Display()
    {
        Console.WriteLine("A chair has 4 legs and a backrest");
    }
}

class ChairWithWheels : Chair
{
    public override void Display()
    {
        base.Display();
        Console.WriteLine("The wheels are fixed with legs to enable quick motion");
    }
}

class Program{
    public static void Main(string[] args)
    {
        Chair chair = new Chair();
        chair.Display();
        ChairWithWheels chair2
            = new ChairWithWheels();
        chair2.Display();
    }
}
```


Abstract class

Abstract class is a kind of class that is declared using an abstract keyword. An abstract class cannot create an instance or an object like the concrete class.

An abstract class can provide the declaration of zero, one or many abstract methods. An abstract method is declared using abstract keyword with no implementation of method body.

In future, we can derive one or many child classes from the abstract class which must override its abstract method with the implementation of the method body.

Exercise

Write a program in C# that create an abstract class Arithmetic having abstract method Calculate(). The class must be inherited by subclasses like Adder and Divider which should implement the Calculate().

Exercise

Develop a program in C# that define the class Player as abstract with three protected members name, country and game.

The class also define a three argument constructor for respective protected properties Name, Country and Game with getters and setters and public virtual void Display() method showing details.

This class must declare an abstract method as -

protected abstract void GameDetails()

Exercise

Define a subclass CricketPlayer that accept additional details like skill, runsScored and wicketsTaken. The class further define six argument constructor to initialize all members, including the additional properties like Skill, Runs and Wickets with getter and setter.

Define a subclass HockeyPlayer that accept additional details like position and goalsScored. The class further define a five argument constructor to initialize all members, including the additional properties like Position and GoalsScored with getter and setter.

The subclasses must provide the implementation of abstract methods to show GameDetails. They must further override Display() method and invoke GameDetails() method in it.

Develop a program that either create an instance of CricketPlayer or HockeyPlayer based on Game and there by accept and display details by invoking overridden methods.

Microsoft Visual Studio Debug Console

```
Enter player's name, country and associated game
Virat Kohli
India
Football
Choose the game either as Cricket or Hockey
```

Microsoft Visual Studio Debug Console

```
Enter player's name, country and associated game
Virat Kohli
India
Cricket
Enter skill, runs scored and wickets taken
Batsman
12752
1
Name: Virat Kohli
Country: India
Game: Cricket
Skill: Batsman
Runs Scored: 12752
Wickets Taken: 1
```



Enter player's name, country and associated game

Akashdeep Singh

India

Hockey

Enter position and goals scored

Center Forward

566

Name: Center Forward

Country: India

Game: Hockey

Position: Center Forward

Goals Scored: 566

Interface

Interface is a kind of reference type that provide the declaration of properties and methods (like abstract method).

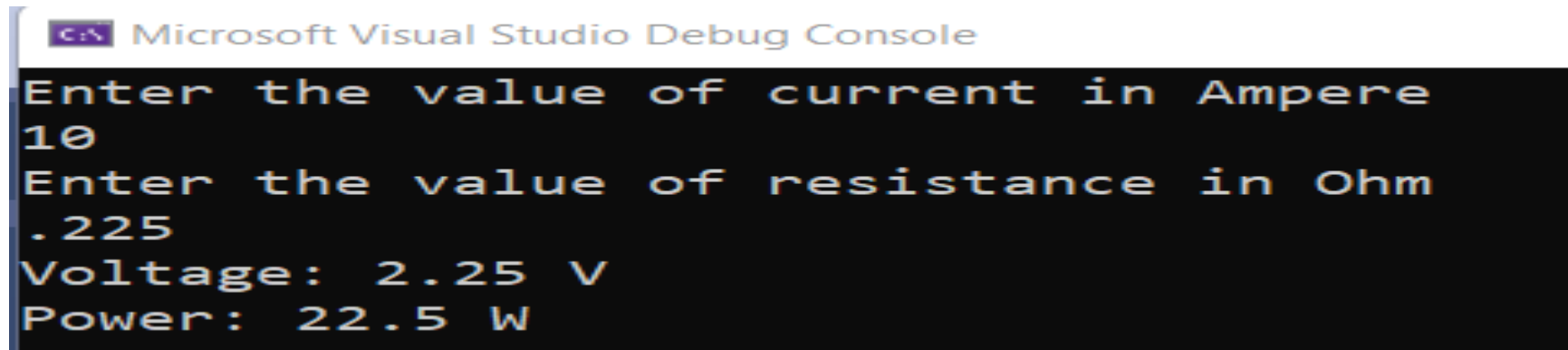
We can declare an interface using 'interface' keyword. Any class which implements an interface must keep the promise to provide the implementation of all of the methods or properties as declared by the interface.

An interface can be implemented by one or many classes. Moreover, a class can implements one or many interfaces.

Exercise

Develop a program in C# that define two interfaces IVolt and IPower consisting of method declarations as “double CalculateVoltage(int current, double resistance)” and “double CalculatePower(double voltage,int current)”.

Define a class Electricity which implements the interfaces and provide the implementation of the methods. Inside Main() create the instance of Electricity type and invoke the methods as published by the interfaces.



```
Microsoft Visual Studio Debug Console
Enter the value of current in Ampere
10
Enter the value of resistance in Ohm
.225
Voltage: 2.25 V
Power: 22.5 W
```

Exceptions

Exception is a type of runtime error which may be get occur due to abnormal incident during the course of the program execution, which ultimately disrupts the normal flow of the program execution.

In order to tackle with the situation of exception, we must use the concept of exception handlers and trap the generated exception within an object of appropriate Exception type.

Exception handlers allows us to make use of try, catch, finally handler blocks and throw clause.

Exceptions

```
try{
    program_statement;          // may generate exception
    if(boolean_expr_abnormal_condition)
    {
        throw new Exception("Error Message");
    }
}
catch(Exception ex)
{
    program_stmt_ex.Message;    // display cause of exception and error message
}
finally{
    // program_stmt_finish_code;
}
```

Exercise

Write a program in C# that implement the concept of exception handling mechanism while the program execute the task of dividing a number by another.

```
Microsoft Visual Studio Debug Console  
Input first number  
10  
Input second number  
4  
Division result : 2.5  
End of the process
```

```
Microsoft Visual Studio Debug Console
```

```
Input first number  
Ten  
Error due to FormatException, Error Message: Input string was not in a correct format.  
End of the process
```

```
Microsoft Visual Studio Debug Console
```

```
Input first number  
10  
Input second number  
0  
Error due to DivideByZeroException, Error Message: Error occurred while trying dividing a number by zero  
End of the process
```

Custom Exception

It is a type of user defined exception that is defined as subclass of Exception. The instance of custom exception is intentionally created and raised using throw keyword whenever an abnormal condition is said to be met.

The general syntax ---

```
public class UserDefinedException : Exception
{
    public UserDefinedException(string errorMessage):base(errorMessage)
    {
    }
}
```

Exercise

Write a program in C# that accept the value of temperature in degree Fahrenheit and prints the temperature in degree centigrade. The program must raise Temperature Exception if the temperature is 50 degree centigrade or above or it is below -2 degree centigrade.

C# Microsoft Visual Studio Debug Console

```
Input temperature in degree fahrenheit
75
Fahrenheit: 75 deg F
Celcius: 23.89 deg C
```

C# Microsoft Visual Studio Debug Console

```
Input temperature in degree fahrenheit
20
TemperatureException: Temperature is very low...
Fahrenheit: 20 deg F
Celcius: -6.67 deg C
```

C# Microsoft Visual Studio Debug Console

```
Input temperature in degree fahrenheit
144
TemperatureException: Temperature is very high...
Fahrenheit: 144 deg F
Celcius: 62.22 deg C
```

Strings

String is an array of characters that can store the value of text or any data as string.

Declaration:

```
String firstName = "Ashok";
```

```
String firstName = new String("Ashok");
```

Length of string: `firstName.Length;` `// 5`

Display data: `Console.WriteLine("{0}", firstName);`

String converted to character array

```
char[ ] letters = firstName.ToCharArray();
```

```
foreach(char ch in letters) {
```

```
    Console.Write("{0}",ch);
```

```
}
```

Strings

Split()

```
Console.WriteLine("Input the name of four cities separated by comma");  
string data = Console.ReadLine();           // Chennai,Bangalore,Hyderabad,Bhopal  
string[ ] cities = data.Split(',');  
    foreach(string city in cities) {  
        Console.WriteLine("{0}",city);  
    }
```

Concatenation or Join()

```
string firstName = "Ashok";    string lastName = "Kanitkar"  
string fullName = firstName + " " + lastName;  
string[ ] names = {"Ashok","Kumar","Kanitkar"}  
string fullName = string.Join(" ", names);           // Ashok Kumar Kanitkar
```

Strings

IndexOf(), LastIndexOf(), Contains(), StartsWith(), EndsWith(), Substring()

```
Console.WriteLine("Index of K : {0}", fullName.IndexOf("K"));           // Index of K : 6
```

```
Console.WriteLine("Index of K : {0}", fullName.LastIndexOf("K"));        // Index of K : 12
```

```
bool answer = fullName.Contains('h')? true : false;                     // true
```

```
bool answer = fullName.StartsWith('Kumar')? true : false;               // false
```

```
bool answer = fullName.EndsWith('Kanitkar')? true : false;              // true
```

Substring(startIndex, Length)

```
string firstName = fullName.Substring(0, fullName.IndexOf(" "));        // firstName="Ashok"
```

Substring(startIndex)

```
string lastName = fullName.Substring(fullName.LastIndexOf(" ")+1);      // lastName="Kanitkar"
```

Strings

ToLower(), ToUpper(), Equals(), StringBuilder

```
string fullName = fullName.ToLower();           // fullName = ashok kumar kanitkar
string fullName = fullName.ToUpper();           // fullName = ASHOK KUMAR KANITKAR
string psw = "password123";
string confPsw = Console.ReadLine();            // Password123
bool answer = (psw.Equals(confPsw))? true : false; // false
bool answer = (psw.Equals(confPsw, StringComparison.InvariantCultureIgnoreCase))? true : false;
StringBuilder sb = new StringBuilder();         // []
    string firstName = "Ashok";    sb.Append(firstName);    // ["Ashok"]
    string lastName = "Kanitkar";  sb.Append(" Kumar ");    // ["Ashok Kumar "]
    sb.Append(lastName);           // ["Ashok Kumar Kanitkar"]
    Console.WriteLine("{0}", sb.ToString());           // Ashok Kumar Kanitkar
```


Regular Expression

```
string pattern = @"[A-Z]{5}[0-9]{4}[A-Z]{1}";  
System.Text.RegularExpressions.Regex regexp = new System.Text.RegularExpressions.Regex(pattern);  
  
Console.WriteLine("Input PAN no");  
string PAN = Console.ReadLine();  
  
bool answer = regexp.IsMatch(PAN) ? true : false;           // true
```

Exercise

Write a program in C# that accept the vehicle registration number and ensure whether it is the valid one or not.

 Microsoft Visual Studio Debug Console

```
Enter the vehicle registration number  
WBJKL098762  
The vehicle registration number is not valid
```

 Microsoft Visual Studio Debug Console

```
Enter the vehicle registration number  
WB04HB8297  
The vehicle registration number is valid
```

Collections and Generics

We can use the instances of collections or generics specific classes to store or maintain the data or elements dynamically.

We must import `System.Collections` and `System.Collections.Generic` namespaces for accessing the collections or generics specific classes or interfaces.

Classes or Interfaces imported from `System.Collections` namespace:

`ArrayList`, `Stack`, `Queue`, `NamedValueCollection`, `IComparable`, `IComparer`, `Hashtable`

Classes or Interfaces imported from `System.Collections.Generic` namespace:

`List<T>`, `Stack<T>`, `Queue<T>`, `IComparable<T>`, `IComparer<T>`, `HashSet<T>`, `SortedSet<T>`, `Dictionary<K,V>`, `SortedDictionary<K,V>`

ArrayList

ArrayList represents an ordered collection of an object that can be indexed individually. It also allows dynamic memory allocation, adding, searching and sorting items in the list.

Exercise

Write a program in C# that use the concepts of collections of elements like Book and Pen using ArrayList

List<T>, HashSet<T>

List<T> It is a type of generic collection which can store and manipulate with the elements of specified type <T> dynamically. It maintains the elements at specific index. It allows addition of any duplicate elements.

HashSet<T> It is a type of generic collection which can store and manipulate with the elements of specified type <T> dynamically. It does not maintain the elements at specific index. It does not allow addition of any duplicate elements.

Exercise

Write a program in C# that create an instance of Batsman, accept name, scores of last 5 matches in List<int> and display the total score and average score.

 Microsoft Visual Studio Debug Console

```
Input the name of the batsman
Suresh Raina
Enter the scores of last five matches
23
18
73
56
12
Total runs scored by Suresh Raina: 182
Average score: 36.4
```

Exercise

Write a program in C# that accept the names of 5 players in an instance of HashSet and display the names of the players in collection.

 Microsoft Visual Studio Debug Console

```
Enter the names of 5 players
PT Usha
Mary Kom
Sania Mirza
Saina Nehwal
Mary Kom
The name of the players in the collection are ...
PT Usha
Mary Kom
Sania Mirza
Saina Nehwal
```

Stack<T>, Queue<T>

Stack<T> is a type of generic collection which can store and manipulate with the elements of specified type <T> dynamically. It can add and remove the elements in last in first out mode.

Exercise

Write a program in C# that use the concept of stack to add or remove plate items.

Queue<T> is a type of generic collection which can store and manipulate with the elements of specified type <T> dynamically. It can add and remove the elements in first in first out mode.

Exercise

Write a program in C# that use the instance of queue to add, show and remove Bus with details like BusNo, StartLocation, Destination from collection. The program must display the first bus details and should release the bus from the queue. The program should show the count of the bus and all bus details in queue.

```
No of buses in queue?  
3  
Enter BusNo, StartLocation, Destination  
81  
Barasat  
Barrackpore  
Enter BusNo, StartLocation, Destination  
206  
SaltLake  
Santoshpur  
Enter BusNo, StartLocation, Destination  
3  
Karunamoyee  
Shrirampur  
Total buses waiting in queue: 3  
Displaying first bus details  
BusNo: 81, Start Location: Barasat, Destination: Barrackpore  
Releasing first bus ...  
Total buses waiting in queue: 2  
Displaying bus details  
BusNo: 206, Start Location: SaltLake, Destination: Santoshpur  
BusNo: 3, Start Location: Karunamoyee, Destination: Shrirampur
```

Dictionary<K, V>

Dictionary<K,V> is a type of generic collection which can store the elements in the pattern of Key-Value pair, where Key is unique text or no use to identify element or value of simple type or reference type.

Exercise

Write a program in C#, that use the instance of dictionary to add and display the details of World Cup Football Final matches in KeyValuePair<int,WorldCup> collection format. The program can also search the World Cup Event details of particular year.

The program must define the class “WorldCup” having auto implemented properties like EventYear, Venue, Winner, RunnersUp, ScoreDetail.

No of World Cup football events

2

Event Year

1990

Venue details

Rome,Italy

Winner

West Germany

Runners up

Argentina

Score details

West Germany 1:0 Argentina

Event Year

2014

Venue details

Rio de Janeiro,Brazil

Winner

Germany

Runners up

Argentina

Score details

Germany 1:0 Argentina

All world cup football event details

World Cup Football 1990

Event year: 1990, Venue: Rome,Italy, West Germany defeated Argentina, Details: West Germany 1:0 Argentina

World Cup Football 2014

Event year: 2014, Venue: Rio de Janeiro,Brazil, Germany defeated Argentina, Details: Germany 1:0 Argentina

Input event year

1990

Event year: 1990, Venue: Rome,Italy, West Germany defeated Argentina, Details: West Germany 1:0 Argentina

Exercise

Write a program in C#, that accept the name of the student, the names of 5 subjects and their respective scores, each in single line input. The program must display the performance of the student having name and set of subjects with their marks in ascending order on the basis of subject. Define the class Performance with auto implemented properties string Subject and int Marks respectively.

The class must implements `IComparable<Performance>` and there by provide the implementations of `CompareTo()`, which help arrange a list of performances in sorted order on the basis of name of the subjects.

Define another class Student with properties for string Name and `List<Performance>` respectively.

Input student's name:

ABCD

Input names of five subjects as single line input separated by comma

English,Maths,Science,Hindi,Language

Input marks of five subjects as single line input separated by comma

73,95,82,66,68

Performance of ABCD

English	73
Hindi	66
Language	68
Maths	95
Science	82

Exercise

Write a program in C#, that accept the name of the student, the names of 5 subjects and their respective scores, each in single line input. The program must display the performance of the student having name and set of subjects with their marks in descending order with respect to marks. Define the class Performance with auto implemented properties of string Subject and int Marks respectively.

The class must implements IComparable<Performance> and IComparer<Performance> interfaces and there by provide the implementations of CompareTo() and Compare(), which help arrange a list of performances in sorted order on the basis of marks.

Define another class Student with properties for string Name and List<Performance> respectively.

Input student's name:

ABCD

Input names of five subjects as single line input separated by comma

Maths,English,Science,Hindi,Language

Input marks of five subjects as single line input separated by comma

100,71,86,57,66

Performance of ABCD

Maths	100
Science	86
English	71
Language	66
Hindi	57

Delegates

Delegate can be defined as a kind reference type, whose instance can be use to keep reference of the method belonging to a specific type.

1> Single cast delegate ---> It can keep reference of single method at a time.

```
delegate return_type singleDele(type1 pm1, type2 pm2);    // single cast  
delegate
```

2> Multi cast delegate ---> It can keep reference of multiple method at a time.

```
delegate void multiDele(type1 pm1, type2 pm2);    // multi cast delegate
```

Exercise

Write a program in C# that use the instance of single cast delegate that target the Add and Subtract method to execute the process.

The program must also use another instance of multi cast delegate that target the Product and Divide methods and execute them together.

Events

Event is a kind of reference type that have subscription of the type of multicast delegate. Whenever the event is raised or invoked it will fire all the target methods together to execute the task.

```
public static event MulticastDelegateType eventName;
```

```
class Language
```

```
{  
    public static void SpeakEnglish()  
    {  
        Console.WriteLine("Good Morning...");  
    }  
  
    public static void SpeakHindi()  
    {  
        Console.WriteLine("Namaskar...");  
    }  
  
    public static void SpeakTamil()  
    {  
        Console.WriteLine("Vanakkam...");  
    }  
  
    public static void SpeakBengali()  
    {  
        Console.WriteLine("Suprabhat...");  
    }  
}
```

```
public delegate void SpeakDele();
```

```
class Program
```

```
{  
    public static event SpeakDele Speak;  
    static void Main(string[] args)  
    {  
        Speak = Language.SpeakEnglish;  
        Speak += Language.SpeakTamil;  
        Speak += Language.SpeakHindi;  
        Speak += Language.SpeakBengali;  
  
        Speak();  
        Console.Read();  
    }  
}
```

Lambda expressions and anonymous functions

We use a *lambda expression* to create an anonymous function. We must use the lambda declaration operator `=>` to separate the lambda's parameter list from its body.

A lambda expression can be of any of the following two forms:

- Expression lambda that has an expression as its body:
 - Statement lambda that has a statement block as its body:

Lambda expressions and anonymous functions

Expression lambda

```
Func<int, int> square = x => x * x;  
Console.WriteLine(square(5));
```

Statement lambda

```
Action<string> greet = name =>  
{  
    string greeting = $"Hello {name}!";  
    Console.WriteLine(greeting);  
};  
greet("World");
```

Files and Streams

A **file** is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a **stream**.

The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the **input stream** and the **output stream**.

The **input stream** is used for reading data from file (read operation) and the **output stream** is used for writing into the file (write operation).

The System.IO namespace has many classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.

Sr.No. Some important I/O Class & Description

- 1 **BinaryReader:** Reads primitive data from a binary stream.
- 2 **BinaryWriter:** Writes primitive data in binary format.
- 3 **Directory:** Helps in manipulating a directory structure with static methods.
- 4 **DirectoryInfo:** Used for performing operations on directories with instance methods.
- 5 **DriveInfo:** Provides information for the drives.
- 6 **File:** Helps in manipulating files using static methods.
- 7 **FileInfo:** Used for performing operations on files using instance methods.
- 8 **FileStream:** Used to read from and write to any location in a file.
- 9 **Path:** Performs operations on path information.
- 10 **StreamReader:** Used for reading characters from a byte stream.
- 11 **StreamWriter:** Is used for writing characters to a stream.

StreamReader and StreamWriter

The **StreamReader** and **StreamWriter** classes are used for reading from and writing data to text files.

The **StreamWriter** class inherits from the abstract class `TextWriter` that represents a writer, which can write a series of character.

The **StreamReader** class also inherits from the abstract base class `TextReader` that represents a reader for reading series of characters.

```
string[] names = new string[] { "Rohit", "Anwar Ali" };  
using (StreamWriter sw = new StreamWriter("names.txt"))  
{  
    foreach (string s in names)  
    {  
        sw.WriteLine(s);  
    }  
}
```

```
string line = "";  
using (StreamReader sr = new StreamReader("names.txt"))  
{  
    while ((line = sr.ReadLine()) != null)  
    {  
        Console.WriteLine(line);  
    }  
}
```

FileStream

The **FileStream** class in the System.IO namespace helps in reading from, writing to and closing files. This class derives from the abstract class Stream.

The syntax for creating a FileStream object is as follows –

```
FileStream <object_name> = new FileStream( <file_name>, <FileMode Enumerator>,  
    <FileAccess Enumerator>, <FileShare Enumerator>);
```

For example, we create a FileStream object F for reading a file named sample.txt as shown –

```
FileStream F = new FileStream("sample.txt", FileMode.Open, FileAccess.Read, FileShare.Read);
```

FileMode Enumerator – Create | CreateNew | Open | OpenOrCreate | Append

FileAccess Enumerator – Read | Write | ReadWrite

FileShare Enumerator – Read | Write | ReadWrite | None | Inheritable

BinaryReader and BinaryWriter

The **BinaryReader** and **BinaryWriter** classes are used for reading from and writing to a binary file.

The **BinaryReader** class is used to read binary data from a file. A **BinaryReader** object is created by passing a **FileStream** object to its constructor.

The **BinaryWriter** class is used to write binary data to a stream. A **BinaryWriter** object is created by passing a **FileStream** object to its constructor.

BinaryWriter bw;

int i = 25; double d = 3.14157; bool b = true; string s = "I am happy";

//create the file

try {

 bw = new BinaryWriter(new FileStream("mydata", FileMode.Create));

}

catch (IOException e) {

 Console.WriteLine(e.Message + "\n Cannot create file.");

 return;

}

//writing into the file

try {

 bw.Write(i); bw.Write(d); bw.Write(b); bw.Write(s);

}

catch (IOException e) {

 Console.WriteLine(e.Message + "\n Cannot write to file.");

 return;

}

bw.Close();

```
BinaryReader br;  
//reading from the file  
try {  
    br = new BinaryReader(new FileStream("mydata", FileMode.Open));  
} catch (IOException e) {  
    Console.WriteLine(e.Message + "\n Cannot open file.");    return;  
}  
try {  
    i = br.ReadInt32();  
    Console.WriteLine("Integer data: {0}", i);  
    d = br.ReadDouble();  
    Console.WriteLine("Double data: {0}", d);  
    b = br.ReadBoolean();  
    Console.WriteLine("Boolean data: {0}", b);  
    s = br.ReadString();  
    Console.WriteLine("String data: {0}", s);  
} catch (IOException e) {  
    Console.WriteLine(e.Message + "\n Cannot read from file.");  
    return;  
}  
br.Close();
```

Exercises

Write a program in C# that accept any string and write the content of the string to the file.

The program must read and display all of the data recorded in the file.

Write a program in C# that accept the rollno, name and average marks of the student and write the data in the binary file using BinaryWriter instance.

The program must read the information back from the file using the instance of BinaryReader.

DirectoryInfo and FileInfo

//creating a DirectoryInfo object

```
DirectoryInfo mydir = new DirectoryInfo(@"c:\Windows");
```

// getting the files in the directory, their names and size

```
FileInfo [] f = mydir.GetFiles();
```

```
foreach (FileInfo file in f) {
```

```
    Console.WriteLine("File Name: {0} Size: {1}", file.Name, file.Length);
```

```
}
```


Filestream, BinaryWriter and BinaryReader – More Examples

```
class Car
{
    public int Distance { set; get; }
    public int Time { set; get; }
    public double Speed { get; private set; }
    public void Calculate()
    {
        Speed = ((double)Distance * 60) / Time;
    }
}
```

```
Car obj = new Car(){ Distance = 20; Time = 30; }
obj.Calculate();

FileStream fs = new FileStream(@"E:\CarDetails.dat",
                               FileMode.Create, FileAccess.Write);

BinaryWriter bw = new BinaryWriter(fs);
bw.Write(obj.Distance);
bw.Write(obj.Time);
bw.Write(obj.Speed);
bw.Close();
fs.Close();

Console.WriteLine("Details about Car Speed written to file
.....");
```

FileStream, BinaryWriter and BinaryReader – More Examples

```
Console.WriteLine("Reading back details about Car Speed from file .....");
```

```
    FileStream fs1 = new FileStream(@"E:\CarDetails.dat", FileMode.Open, FileAccess.Read);
```

```
    BinaryReader br = new BinaryReader(fs1);
```

```
    String str=String.Format("Distance : {0} KM, Time : {1} Mins, Speed : {2:0.00} KM/HR",  
                             br.ReadInt32(),br.ReadInt32(),br.ReadDouble());
```

```
    Console.WriteLine("{0}",str);
```

```
    br.Close();
```

```
    fs1.Close();
```

Serialization, Deserialization

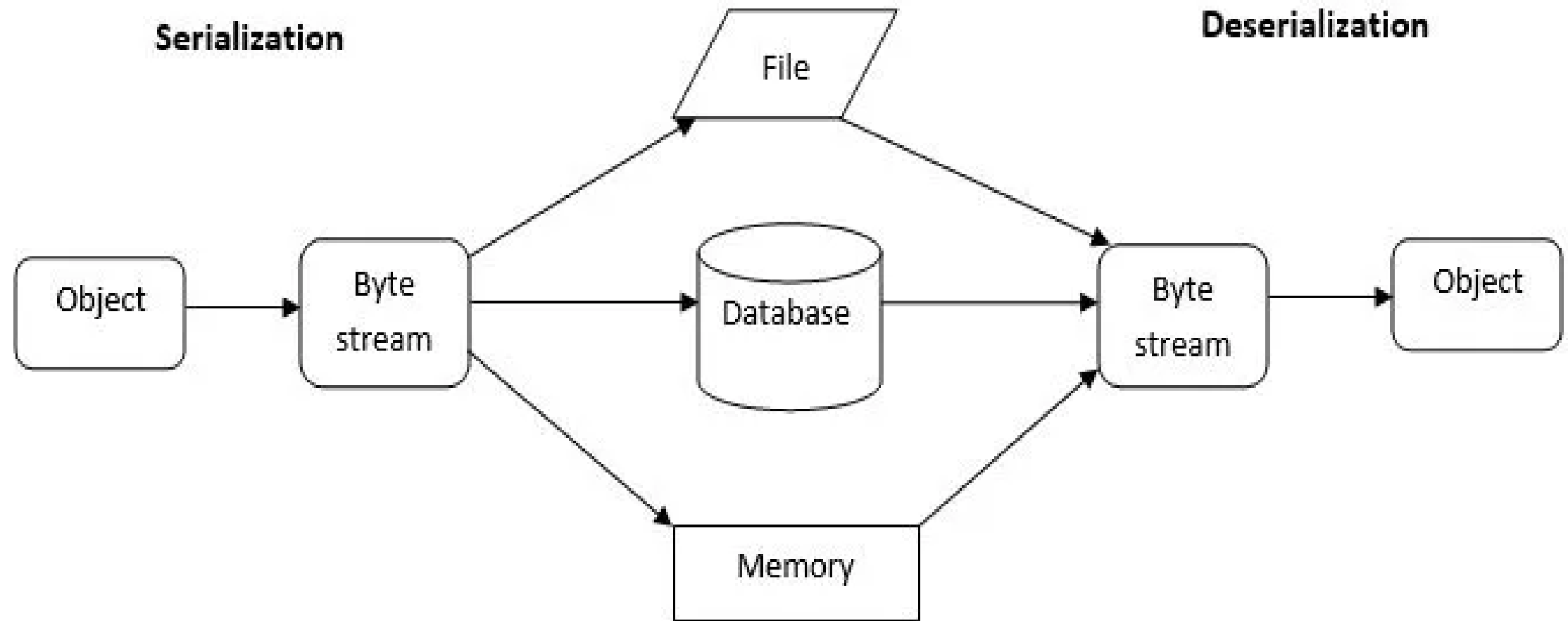
Serialization

It is an important process by virtue of which the state of the object can be transformed into sequence of bytes which can be written to the destination file using an instance of File stream.

We can implement two kinds of serialization - Binary Serialization and XML Serialization

Deserialization

It is the reverse process by virtue of which the program can read the bytes of streams of the file content and transform it back into an Object using unboxing process to provide the actual information of the object.



Exercises

Write a program in C# that can write the state information of the object of Car using Serialization. The program must use the concept of deserialization to read back the data from the file. [Use an instance of Binary Formatter]

Write a program in C# that can write the state information of the object of Car using Serialization in XML Document. The program must use the concept of deserialization to read back the data from the XML file. [Use an instance of Xml Serializer]

Multi-threading in C#

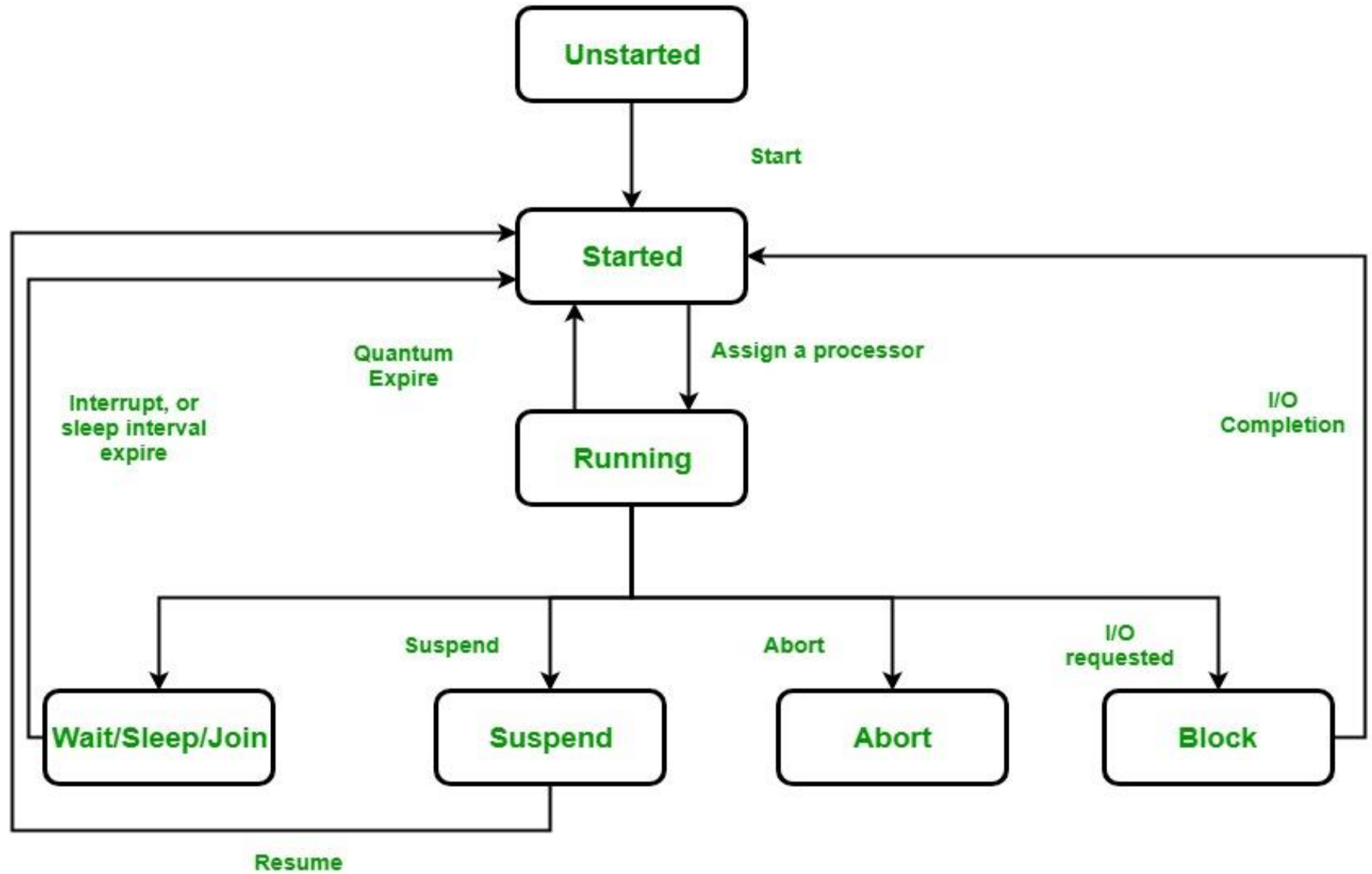
Thread is a kind of lightweight component that can execute unit segment of the work belonging to the heavyweight process.

A Program may consist of single or many process, whereas the works belonging to a process can be break up into many smaller unit of segments, where each of the segment of work can be controlled by one or many threads.

The threads executing the task on behalf of a process continue to share the same resources of memory that is already claimed for the process.

In C#, we can create an instance of the Thread class by importing it from System.Threading namespace.

Thread Life-cycle



Exercises

Write a program in C# that work with an instance of the thread which print the letters of English alphabets within every interval of one second.

Write a program in C# that work with the 3 instances of the thread which print the letters of English alphabets within a gap of CPU generated random interval. [Use Join()]

Write a program in C# that accept the scores of last 5 overs in cricket match and use the instance of thread which accept the array of scores as parameter and call the target method for the displaying the scores of each over in bar chart format

Write a program in C# that work with the 'n' instances of the thread which must print the letters of English alphabets within a gap of CPU generated random interval. [Use lock]

Synchronization

Synchronization is the process by virtue of which we can implement the mechanism of lock using the object for the particular segment of work that is already being started by one of the multiple executable threads.

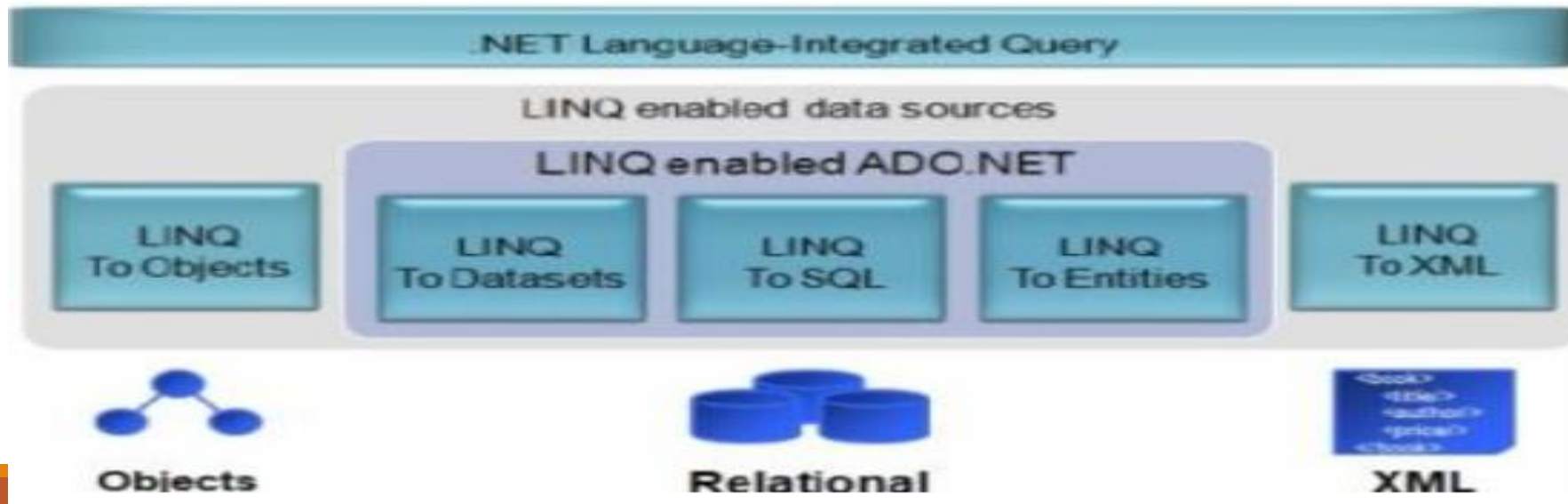
The first thread which executes the segment of work, gains the lock on it using an object, and thereby gets the opportunity to complete the work till end and then it gets itself free by releasing the lock.

The other participating threads will not be able to execute the segment task that is already being executed by one of the threads which has acquired the lock and hence wait for their chances as per CPU response time till the lock is released by the previous busy thread.

LINQ

LINQ stands for Language Integrated Query. LINQ is a data querying API with SQL like query syntaxes.

LINQ provides functions to query cached data from all kinds of data sources. The data source could be a collection of objects, cached data in dataset, database or XML files. We can easily retrieve data using LINQ from any object that implements the `IEnumerable<T>` or `IQueryable<T>` interface.



Examples

```
int[] numbers = { 8, 5, -2, 9, 19, 14, 16, 7, 3, 22 };
```

```
var nos = from n in numbers
          select n;
```

```
//Using LINQ to fetch and display all numbers....
```

```
foreach(var a in nos)
{
    Console.WriteLine("{0,5}",a);
}
```

```
var oddnos = from n in numbers
              where n % 2 == 1
              select n;
```

```
//Using LINQ to fetch and display odd numbers....
```

```
foreach (var a in oddnos)
{
    Console.WriteLine("{0,5}", a);
}
```

```
//Using LINQ query extentension method with query operator to fetch and display even numbers....
numbers.ToList().Where(v => v % 2 == 0).ToList().ForEach(v => Console.WriteLine("{0,5}", v));
```

Examples

```
int[] numbers = { 8, 5, -2, 9, 19, 14, 16, 7, 3, 22 };
```

```
var nos = from n in numbers           // Fetch all elements in ASC Order....");
orderby n ascending
select n;
```

```
var nosInDesc = from n in numbers    // Displaying all odd elements in DESC Order...
where n%2 != 0
orderby n descending
select n;
```

```
// Displaying all odd elements using LINQ Standard Query Operator in ASC Order...
numbers.ToList().Where(v => v % 2 != 0).OrderBy(v => v).ToList().ForEach(v => Console.Write("{0,5}", v));
```

```
// Displaying all even elements using LINQ Standard Query Operator in DESC Order...
numbers.ToList().Where(v => v % 2 == 0).OrderByDescending(v => v).ToList().ForEach(v=>Console.Write("{0,5}",v));
```

Examples

```
int[] numbers = { 8, 5, -2, 9, 19, 14, 16, 7, 3, 22 };
```

```
numbers.Select(n => n).ToList().ForEach(n=>Console.Write("{0,5}", n));
```

```
int noOfElements = (from n in numbers select n).Count();
```

```
int maxValue = (from n in numbers select n).Max();
```

```
int minValue = (from n in numbers select n).Min();
```

```
int sum = (from n in numbers select n).Sum();
```

```
double avgValue = (from n in numbers select n).Average();
```

```
Console.WriteLine("\n Using LINQ, No of Elements - {0}, Max - {1}, Min - {2}, Sum - {3}, Avg - {4}",noOfElements,  
maxValue,minValue,sum,avgValue);
```

Examples

```
var customerDetails = from cust in customers           // Fetch all customer instance
                        select cust;
```

```
var customerLocations = from cust in customers         // Fetch customer with name and location
                        select new { Name = cust.CustomerName, Location = cust.Location };
```

```
var healthFoods = from p in products                  // Fetch all product of category "Health Foods"
                   where p.Category.Equals("Health Foods")
                   select p;
```

Example

```
var purchases = from cust in customers                                // Linq using join
                 join ord in orders on cust.CustomerId equals ord.CustomerId
                 join prd in products on ord.ProductId equals prd.ProductId
                 select new
                 {
                     CustomerName = cust.CustomerName,
                     ProductName = prd.ProductName,
                     Brand = prd.Brand,
                     Category=prd.Category,
                     OrderDate = ord.OrderDate.ToShortDateString(),
                     Price = prd.Price,
                     Quantity = ord.Quantity,
                     Amount = prd.Price * ord.Quantity
                 };

```

ADO.NET

ADO.NET is an integral component of .NET Framework. It provides a sophisticated model by virtue of which any .NET Application can communicate with the database and execute the SQL commands for querying information or updating the data.

ADO. Net Model mainly depends on two major components

DataProvider

and DataSet

Data Provider

It is the important component of ADO. Net Model that mainly comprises of four in-built objects namely - Connection, Command, DataReader and DataAdapter.

It can use Connection instance to create open the communication channel with the database with the help of connection string.

It can use Command instance for preparing SQL Statements and execute them in the database either for retrieving or updating data.

It can use the DataReader for accessing the information in read-only and forward mode.

It can use Data Adapter that work with it's own commandbuilder for retrieving or updating the data. The data adapter can fetch the records from the table using select command and fill the retrieved cached data into the dataset, which can be used by the application.

If required the DataAdapter can update the changes made in the datatables by modifying them in the original tables in database.

Connection

It is one of the most important object of the DataProvider. The instance of the connection can use the Connection string to create open a communication channel with the database.

```
using System.Configuration;
using System.Data.SqlClient;
using System.Data;
...
```

```
string conString = ConfigurationManager.ConnectionStrings["sqlcon"].ConnectionString;
SqlConnection sqlcon = new SqlConnection(conString);
sqlcon.Open();
```

```
if(sqlcon.State == ConnectionState.Open)
{
    Console.WriteLine("Connection Open")
}
```

Command

Command is one of the useful built in object of the Data Provider. The ADO. Net model use the Command object to set the SQL Statement either in the form SQL Text instructions or stored procedure. The command object need the reference of connection object in open state, before it can pass and execute the SQL commands in the database.

The command can invoke three important methods for the execution of the SQL commands.

ExecuteScaler(), ExecuteReader() and ExecuteNonQuery()

Using ExecuteScaler(), the command object can execute the simple select statement that can return a single value as an object.

Using ExecuteNonQuery(), the command object can execute the updation statement like insert, delete or update that can update the record of table in database.

Using ExecuteReader(), the command object can execute the Select statement which can retrieve the records in the form of Result Set and thereby returns the reference to DataReader.

```
SqlCommand cmd = new SqlCommand("Select count(*) from Employee",sqlcon);  
    object data = cmd.ExecuteScalar();  
    Console.WriteLine("There are {0} employees", (int)data);
```

```
string commandText="Insert into Author values (@authId,@name,@contact)";  
SqlCommand cmd=new SqlCommand(commandText,sqlcon);  
    cmd.Parameters.AddWithValue("@authId",authId);  
    cmd.Parameters.AddWithValue("@name",name);  
    cmd.Parameters.AddWithValue("@contact",contact);  
int r = cmd.ExecuteNonQuery();  
if(r>0)  
{  
    Console.WriteLine("Record Inserted");  
}
```

DataReader

DataReader is an in built object of the DataProvider which can get reference of the object of result set as returned by execution of the select query and thereby it helps us to access each record of the result set in read-only and forward mode.

```
SqlConnection sqlcon = new SqlConnection(conString);    sqlcon.Open();
string authId = Console.ReadLine();
SqlCommand cmd = new SqlCommand("Select * from Author where authorId = @authId", sqlcon);
cmd.Parameters.AddWithValue("@authId", authId);
SqlDataReader sqlldr = cmd.ExecuteReader();
if (sqlldr.Read())
{
    string ald = sqlldr[0].ToString();                string name = sqlldr[1].ToString();
    string email = sqlldr[2].ToString();                long contact = Int64.Parse(sqlldr[3].ToString());
    Console.WriteLine("Auth Id - {0}, Name - {1}, Email - {2}, Contact - {3}", ald, name, email, contact);
}
sqlldr.Close();
```

Data Adapter

DataAdapter is one of the integral component of data provider. It can use it's own commandbuilder to execute the SQL statements either for updating of the database or retrieving the records from the table.

The DataAdapter can fill the cached set of retrieved records in the dataset, which can be further used by .NET application irrespective of whether the connection to the database remains open or not, as the dataset is purely based on the concept of disconnected architecture.

If the .NET application update the record in the dataset, the actual change in the database is done by the data adapter.

```
DataSet ds = new DataSet();  
SqlDataAdapter sda = new SqlDataAdapter("Select * from Author", sqlcon);  
sda.Fill(ds, "Author");
```

Thanks
