.NET Introduction

.NET

- NET is an open source development platform developed by Microsoft to make end-to-end applications.

Applications are divided into

* Console application
* Window application
* Web application
* Mobile application

.NET Framework

- The .NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft windows.
- .NET includes a large class library called Framework class library.
- FCL provides user interface, database connection, cryptography, web development and many more.
- .NET Framework applications execute in CLR

History of .NET

- 1997 : Microsoft wants to develop a new program language to develop desktop, web, mobile apps.

- 1998 : Microsoft has developed “SMC” language.

  Simple Managed code. It contains three features:

  1. Light weight : Easy to implement
  2. Hardware Independent : To able to run on different hardware.
  3. Shortcut code : For perform common activities, some predefined libraries are available.

     Ex: Arrays.Sort(arrayname);

- 1999: SMS → renamed as → C#

  ASP(ASP)

  CE

  C# + ASP + CE are together , it is called as NGWS(Next Generation Windows Services).

2000 : NGWS → renamed as → .NET

2001: Development..

2002: .NET 1.0 was released

2003: .NET 1.1

2005 : .NET 2.0

2006 : .NET 3.0

2007: .NET 3.5

2007: .NET3.5

2010: .NET4.0

2012: .NET4.5

2013: .NET 4.5.1

2014: .NET 4.5.2

2015: .NET 4.6

2016: .NET 4.6.2

2017: .NET 4.7

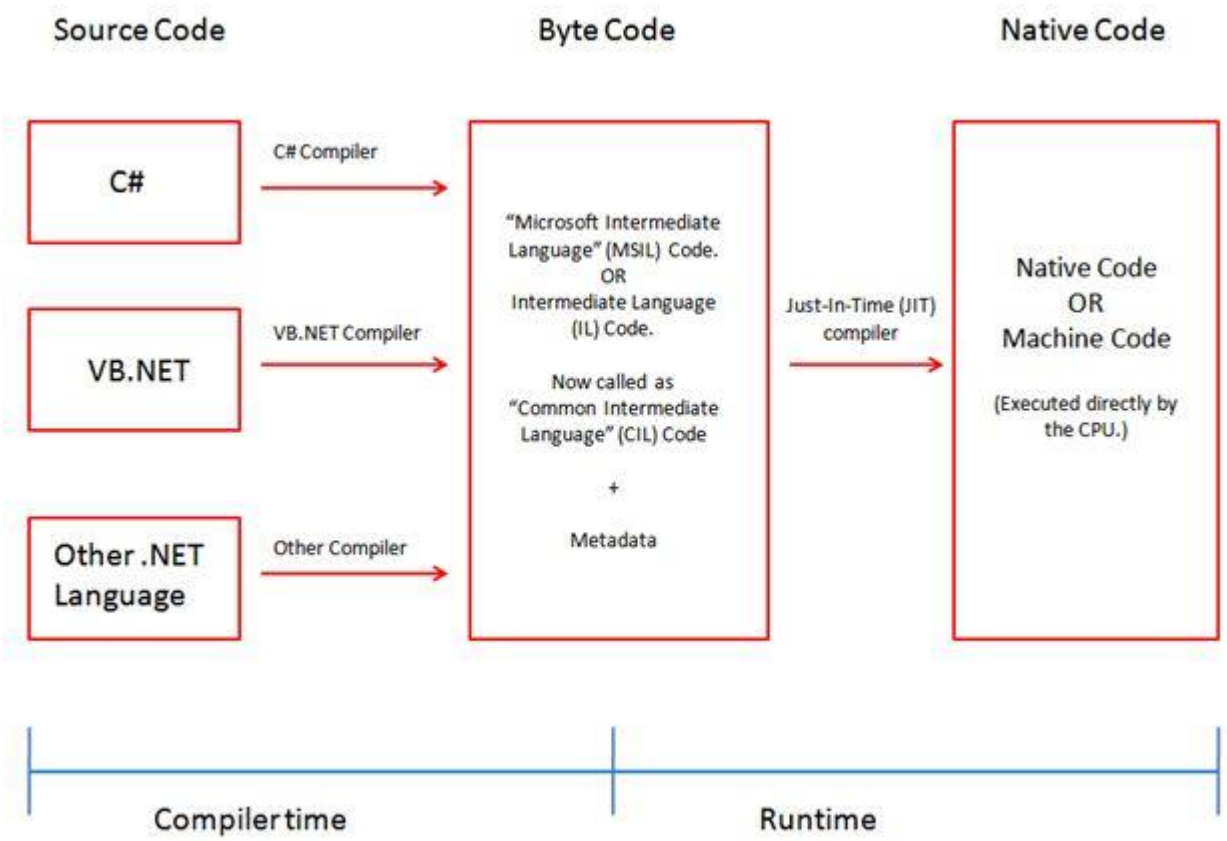
2019: .NET 4.8

Version of .NET Framework

- Version — Release Date

Version	Release Date
1.0	2002 →We cannot develop web application, only develop base application
1.1	2003 ->Inbuilt web applications
2.0	20005 -> add Generic collection
3.0	2006 →Add advance technologies(WPF(windows presentation foundation),WCF,Wf)
3.5	2007→ Linq
4.0	2010 -> Integrated silver light application
4.6	2012→ Metro apps using window 8 operation system
4.7	->Performance and reliability improvement
4.8	

Compilation process



Compilation process is done in two phases:

- First phase is done by language compilers.
- Second phase is done by JIT(Just-In-Time).

Components of .NET Framework

1. CLR(Common Language Runtime)
2. BCL(Base class Library)

1. CLR :

- CLR is runtime execution engine of .NET. It acts as interface between .NET application and operation system.
- CLR provides many features like manage memory, handle exception.
- Without CLR, we cannot execute application.

CLR provides services like CLS, CTS

CLS(Common language specification):

The code which is there in one .NET framework language can be used in another .NET framework language.

CTS(Common Type System):

* In CTS, it deals with datatypes. Here we have several languages, each & every language have own datatype, one language datatype cannot understand to other language.

- CLR understands all language because CLR having own datatype.

2. BCL(Base class Library):

- It is also known as class library. Class library is collection of reusable types that are closely integrated with CLR.
- Class library provides classes and types that are helpful in performing day to day operation.
- Ex: Dealing with string , primitives types, Database connection, IO operations.

BCL can be divided into two:

* User defined class library : unit part of deployment is called Assembly

* Predefined class library : collection of predefined class & methods.

Memory Management

- Memory management is a specific programming language is responsible for allocating and deallocating memory used by applications without involving developers.
- .NET manages memory automatically.

  * Create objects onto managed memory blocks(heaps).
- Allocates objects onto one of two heaps:

  1. Simple Object Heap(SOH)
  2. Large Object Heap(LOH)
- You allocate onto a heap whenever you the “new” keyword in code.

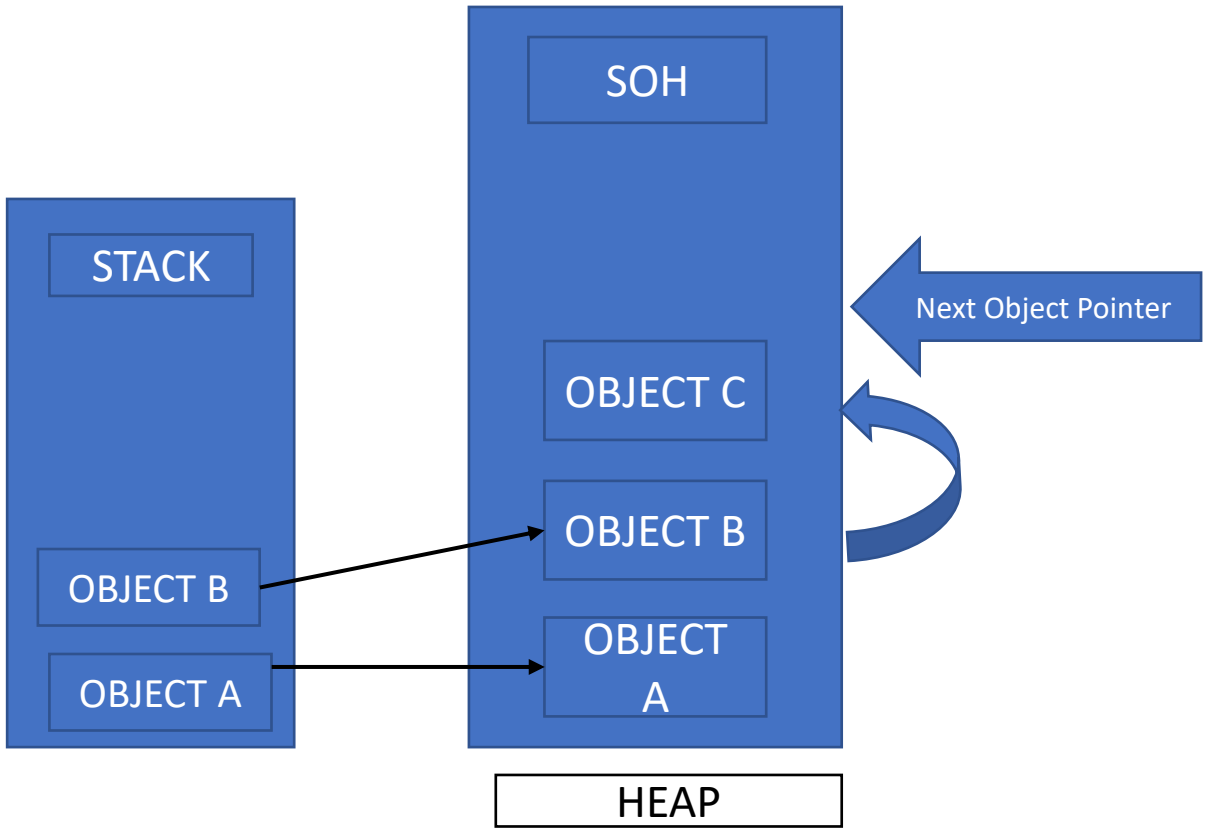
Simple Object Heap:

- Allocation of objects < 85KB, that objects are allocated in SOH.
- Contiguous Heap

* objects allocated consecutively

* Next object pointer maintained.

• SOH

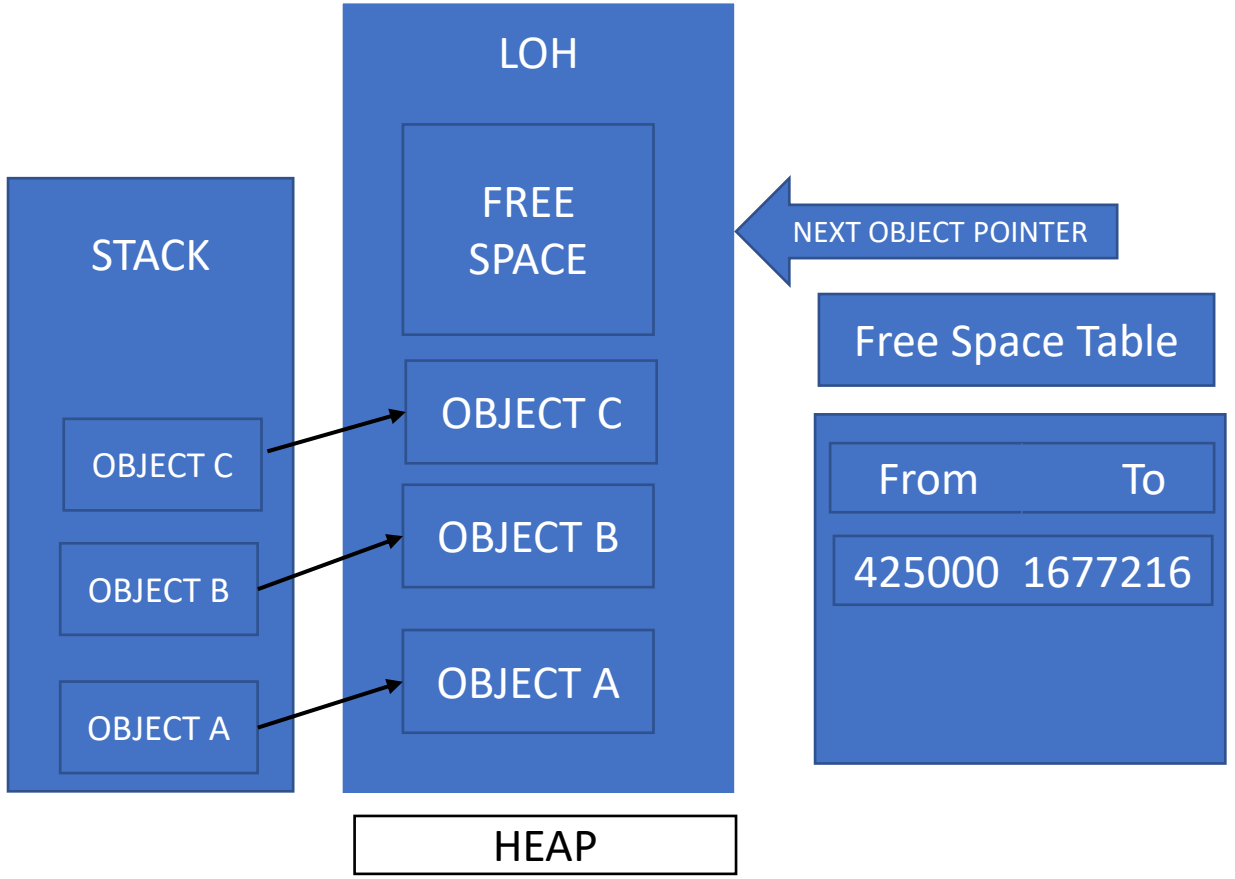


Large Object Heap:

- Allocation of objects >=85KB, that objects are allocated in LOH.
- Non contiguous Heap

  * Objects allocated using Free Space Table
- Garbage collected when LOH Threshold is reached.
- LOH uses a Free Space Table to find where to allocate.

- LOH



Garbage Collection:

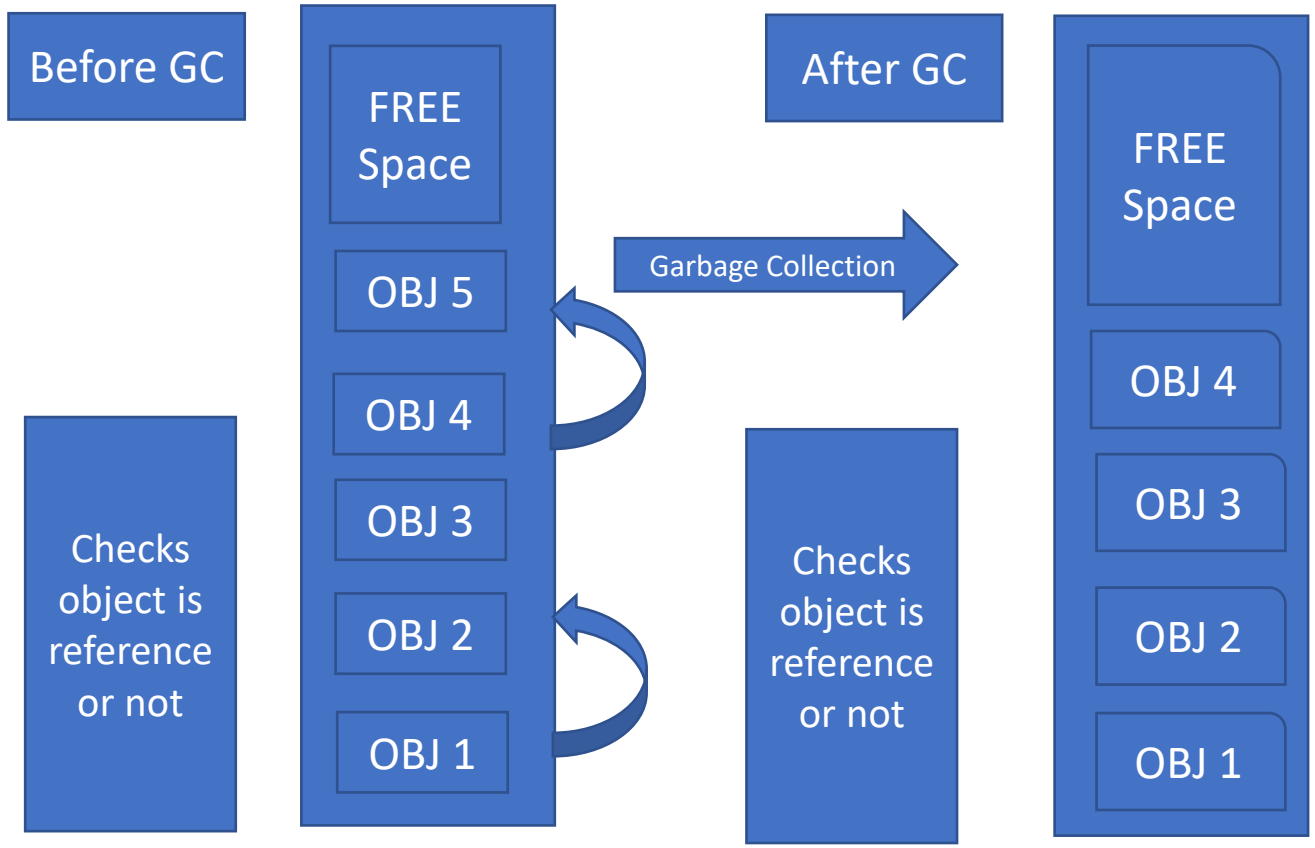
- Garbage Collection is one of the services that provided by CLR.
- Manages the allocation and release of memory for an application.
- Purpose of Garbage collector is AMM(Automatic memory management).
- To clear memory in two ways:
  1. Manually → If you go for manually, it will leads to the errors.
  2. Automatic → We go for AMM, Which will be provided by Garbage collector.

Garbage Collection

Every object holds memory in the managed heap. When the heap is full, the garbage collector(GC) is called. The GC goes through every object to examine

- First, Gc checks for each object. That object refers to reference or null.
- It only remove dead objects.

Garbage collection:

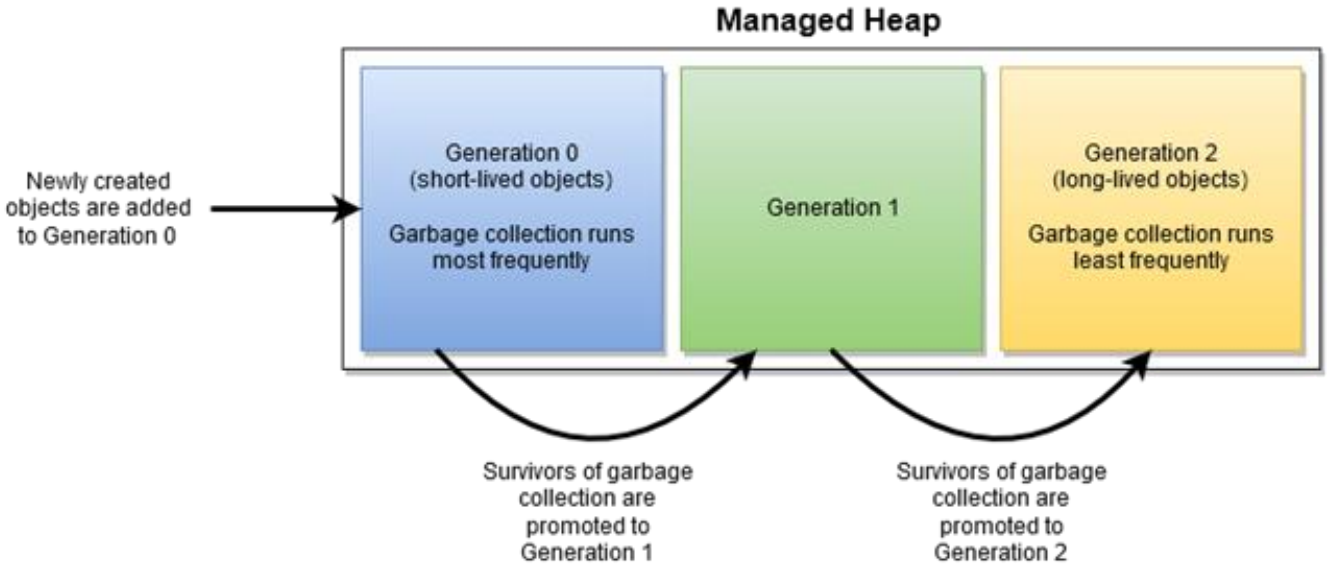


Heap memory is divided into 3 generations:

1. Generation 0
2. Generation 1
3. Generation 2

Garbage collector has 3 phases:

1. Marking phase : It create list of live object.
2. Relocating phase : In this phase, it updates the reference objects.
3. Compacting phase : It clears unuse objects from heap memory.



Static class

In C#, one is allowed to create, by using static keyword. A static class con only contain static data members, static methods and static constructors. It is not allowed to create objects of the static class. Static classes are sealed, means you cannot inherit a static class from another class.

Syntax:

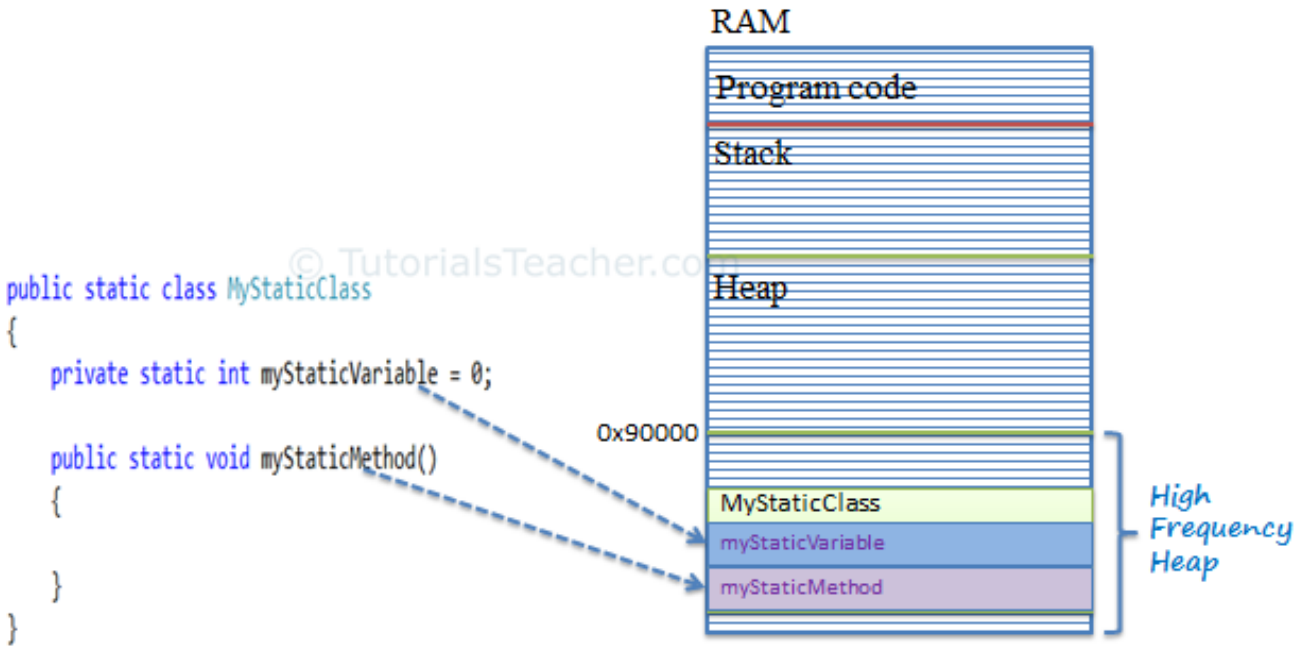
Static class Class Name

{

    //static data members

    // static methods

}



GAC :

- GAC is short version of “Global Assembly Cache”. It is a common place in the OS where assemblies that are going to be shared between different applications can be stored. The .NET assemblies are there, for one

GAC = Global Assembly Cache

Let’s break it down:

- Global – applies to the entire machine
- Assembly –what .NET calls its code-libraries(DLLs)
- Cache – a place to store things for faster/common access

So, the GAC must be a place to store code libraries so they’re accessible

To all applications running on the machine.

Assembly

- Assembly is .NET term for a deployment on unit.
- Files with extensions .exe/.DLL are known as assembly.
- Assemblies includes metadata.
- Assembly can be classified into two
  * private Assembly
  * Shared Assembly

Public Assembly:

- Follows 3 steps:
- Building strong name by using command is sn.exe –k keyname.snk
- Singing Assembly
- Copying into GAC by using GACUTIL.exe -i dllfilename.dll

Private Assembly:

- It is used by multiple applications.
- Drawback of private assembly is, when DLL file use in every application, each & every time copy of DLL file will be placed.
- Location of private assemblies in own folder.

Shared Assembly:

- It is also called as public assembly.
- In this, we can use DLL is reusability in several application.
- But , DLL file is placed in GAC(Global Assembly Cache).
- Shared Assembly must have a version number & unique strong name.

String pooling in .NET

- The string intern pool is a table that contains a single reference to each unique literal string declared or created programmatically in your application. The CLR uses the intern pool to minimize string storage requirements. As a result, an instance of a literal string with

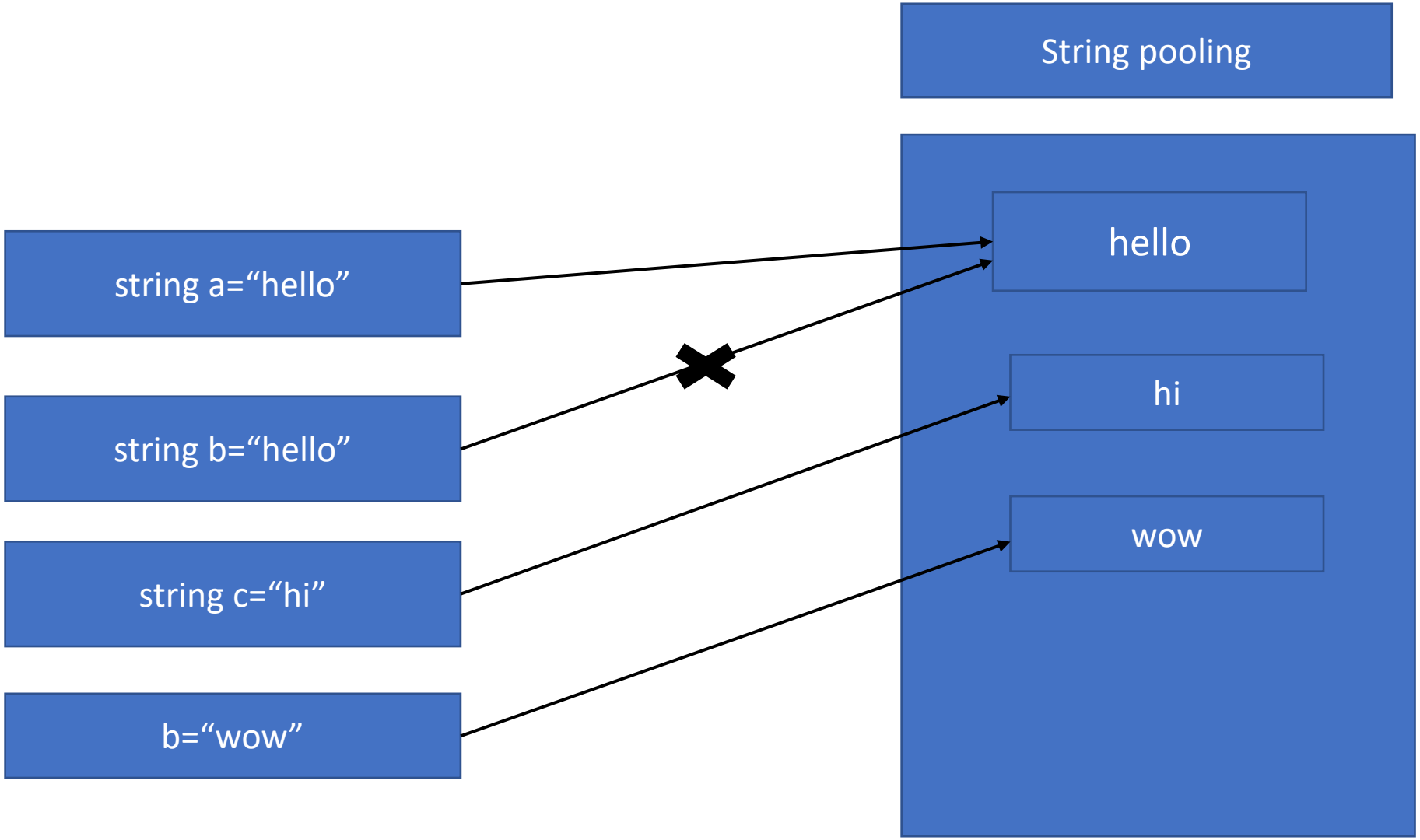
a particular value only exist once in the system.

Eg: string a="hello";

string b="hello"

In the above case when the variable 'b' is assigned value, the CLR first checks it in the string pool for an object with same value, if such a variable exists in the pool, the reference to the same object is assigned to variable 'b'.

String pooling in C#



Visual studio

- Visual studio is an Advance integrated Development Environment developed by Microsoft corporation in the year 2000.
- It is actively used to develop computer programs, websites, Desktop applications, Games and much more.
- It supports many languages like C, C++,C#, F#, python etc…

Features of Visual Studio:

- Code Editor
- Debugger
- Designer
- Other Tools

Visual studio is Most power IDE, that reason behind

- Light weight
- Fast
- Open source
- Cross-platform
- Extensible

Visual Studio Code

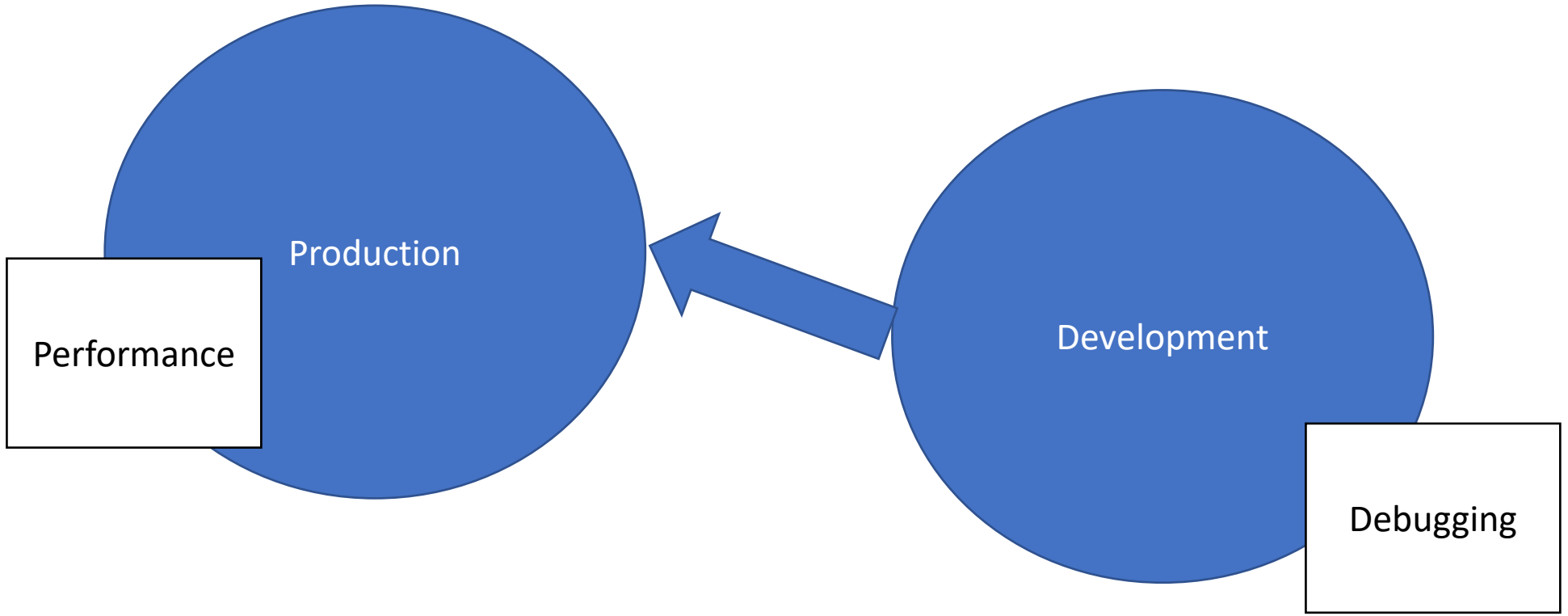
- Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for windows, macOS, Linux.
- It has a rich ecosystem of extensions for other languages(such as c, C++, C#, python etc..)
- It is combines the simplicity of a source code editor with powerful developer tooling, like debugging.
- And also it provides lot of features like syntax highlighting, bracket-matching, auto-indentation, box-selection and , more.
- By using visual studio code , easily handle with shortcut keys.

- The main features that make VS code unique is the sidebar that hosts

the core features you will be interacting with to code . Everything else you need is likely an extension you just need to install.

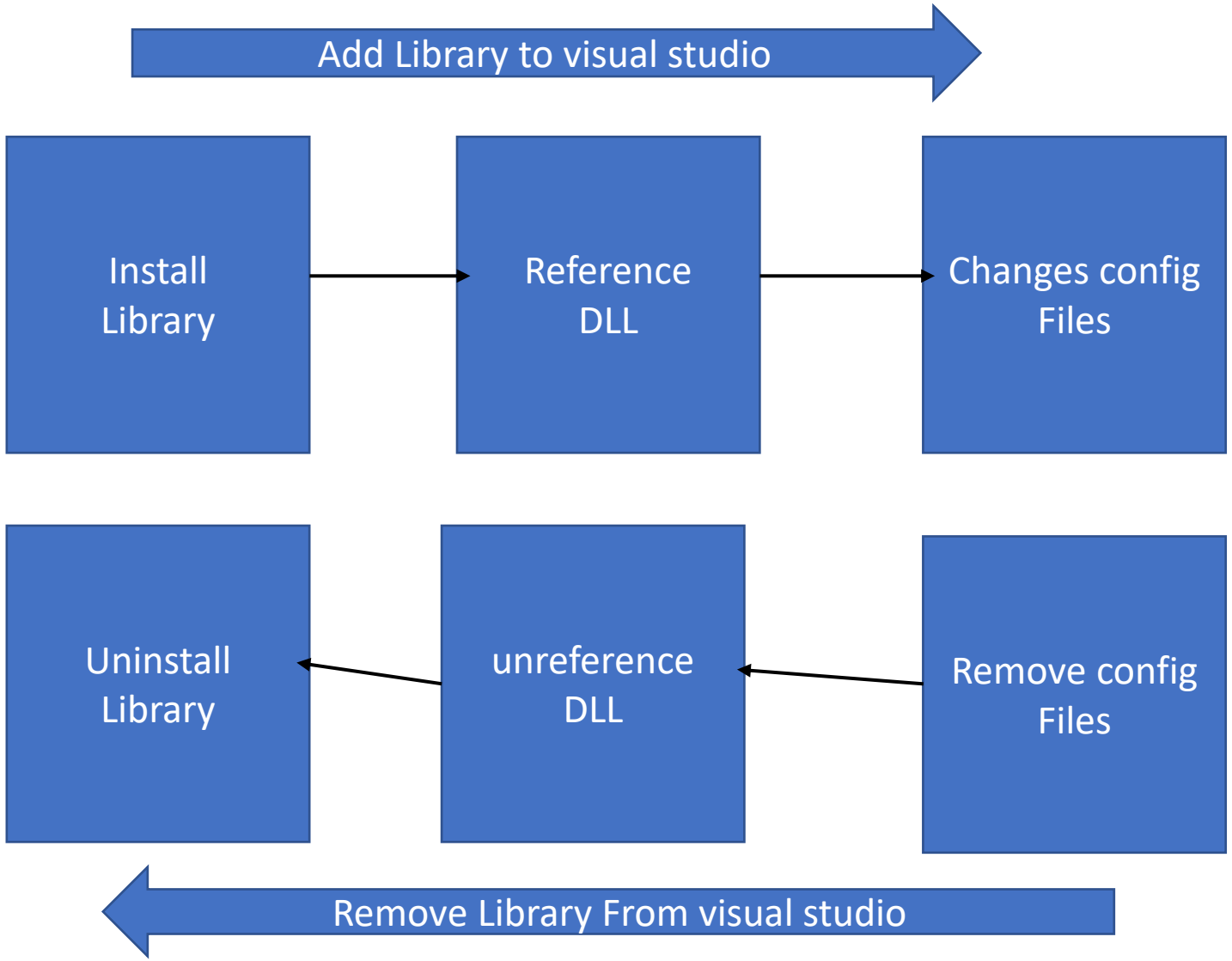
Difference between Debug and Release

Debug & Release



- In Debug mode, code is not optimized.
  In Release mode, code is optimized.
- In Debug mode, shows full stack trace.
- In release mode, directly show errors.
- In Release mode, Debug symbols are removed.

NuGet


Add Library to visual studio
Install Library
Reference DLL
Changes config Files
Uninstall Library
unreference DLL
Remove config Files
Remove Library From visual studio

NuGet:

- NuGet is a visual studio extension which helps us to search, locate the library, download them, reference them in our visual studio project

  And make appropriate changes.
- NuGet packages contain reusable code that other developers make available to you for use in your projects.
- Packages are installed into visual studio project using the NuGet package Manager or package Manager Console.
- Once installed, refer to the package in code with using <namespace> where <namespace> is specific to the package you are using.

- NuGet is the official Package manager for .NET
- Package is compiled library and descriptive metadata.

The Package Manager Console provides a PowerShell interface within Visual Studio on Windows to interact with NuGet through the specific commands listed below:

- get-help NuGet : show all command details
- Find-Package : Searches a package source using a package ID or keyword
- Get-Package : Retrieves the list of packages installed in the local repository.
- Install-Package : Install a package into the project.
- Uninstall-Package : Uninstall a package. If other packages depend on this package, the command will fail unless the –Force option is specified.

- Update-Package : Updates a package to a newer version.
- Sync-Package : Get the version of installed package from specified project and syncs the version to the rest of projects in the solution.
- Get-Project : Displays information about the default or specified project.

GitHub

- GitHub is a code hosting platform for collaboration.

GitHub essentials are:

* Repositories
* Branches
* Commits
* Pull Request
* Git

Repository:

- Storage space for your project.
- It can contains folders and any type of files(HTML, CSS, JavaScript , Documents etc..)
- It also include README file about project.

Branch:

- Branches allow you to work on other features.
- By default a repository has master branch.

Commits:

- At GitHub, changes are called commits.
- Each commit has a description explaining why a change was made.

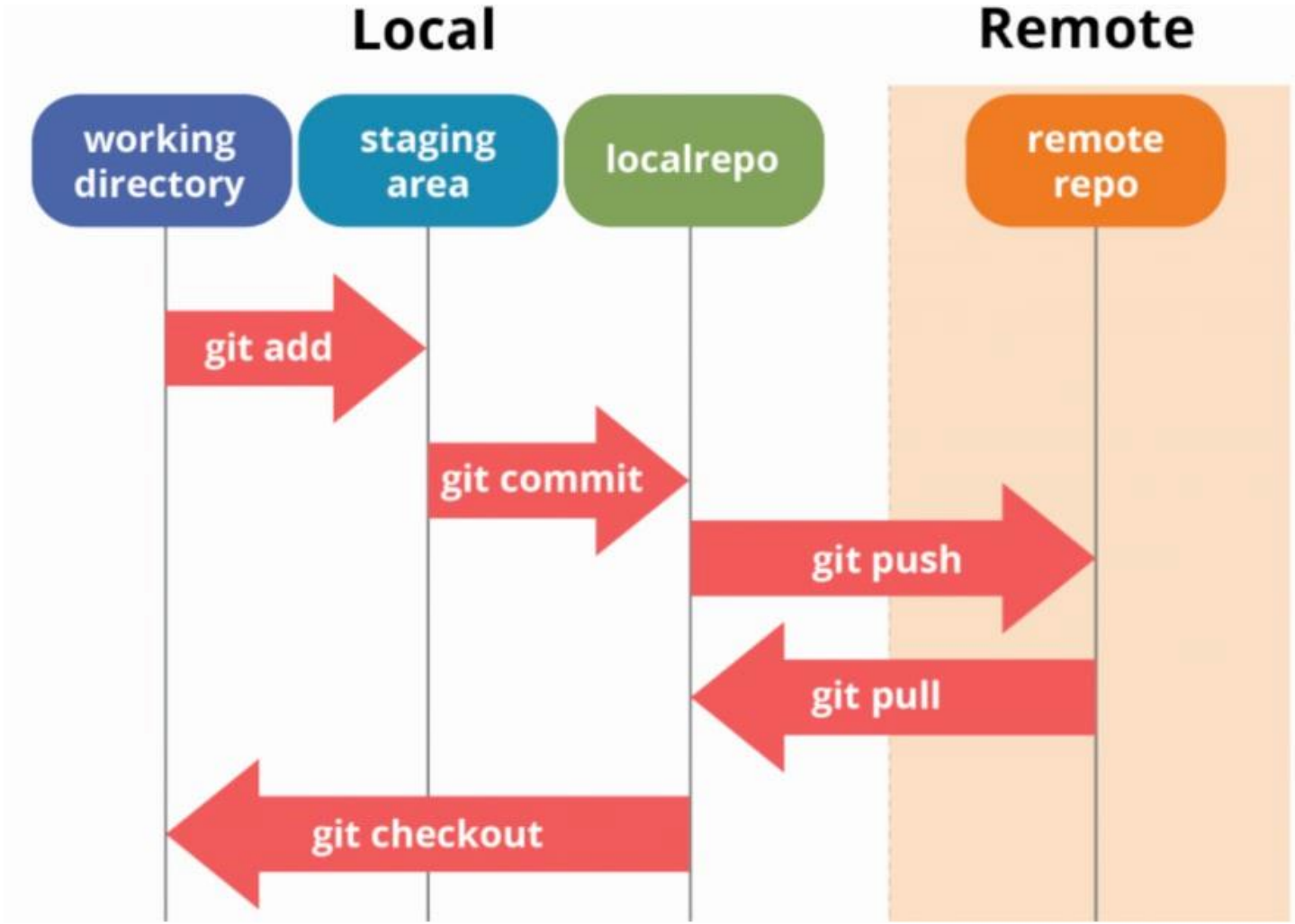
Pull Request :

- Pull requests are the heart of GitHub collaboration.
- With a pull request you are proposing that your changes should be merged with master.
- Pull request show content differences, changes, additions and subtractions.

Git:

- Git is a revision control system, a tool to manage your code history.
- Installed & maintained in your local system.
- Git push or pull data from central server.

GitHub:


Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull
git checkout

Git Repository structure consists of 4 parts:

- Working directory : This is your local directory where you make the project(write code) and make changes to it.
- Staging Area : This is an area where you first need to put your project before committing. This is used for code review by other team members.
- Local Repository : This is your local repository where you commit changes to the project before pushing them to central repository on GitHub. This what is provided by distributed version control system. This corresponds to the .git folder in our directory.
- Central Repository : This is the main project on the central server, a copy of which is with every team member as local repository.

Git commands

- git config : This command sets the author name and email address respectively to be used with your commits.

Usages: git config –global user.name “name”

git config –global user. Email “email address”

- git init : This command is used to start a new repository.

Usages : git init [repository name]

- git clone : This command is used to obtain a repository from an existing URL.

Usages : git clone [URL]

- git add : This command adds a file to the staging area.

Usage : git add [file]

git add *.txt

git add .

- git add *: This command adds one or more to the staging area.
- git commit : This command records the file permanently in the version history.

Usages : git commit –m [“message”]

git commit –a –m [“message”]

git commit –A(for all file and folders move to staging area)

- git status : This command lists all the files that have to be committed.

Usages: git status

- git rm : This command deletes the file from your working directory and stages the deletion.

Usages : git rm [file]

- git branch : This command lists all the local branches in the current repository.

Usages : git branch [branch name]

git branch –d [branch name] →To delete branch

git branch → shows all branches in current repository.

- git checkout : This command is used to switch from one branch to another.

Usages: git checkout [branch name]

git checkout –b [branch name]→to make and switch to branch.

- git remote : This command is used to connect your local repository to the remote server.

Usage: git remote add [variable name] [remote server link]

- git push : This command sends the committed changes of master branch to your remote repository.

Usage : git push [variable name] master

git push [variable name] [branch]→specific branch

git push –all [variable name]→pushes all branches to your remote repository.

git push [variable name] : [branch name]→deletes a branch on your remote repository.

- git pull: This command fetches and merges changes on the remote server to your working directory.

Usages : git pull [repository Link]

git pull [variable name] [branch name]

Bitbucket

- Bitbucket is a Git-based source code repository hosting service.
- In Bitbucket, to create unlimited private repositories.
- Bitbucket is mostly used for code and code review.
- Bitbucket Cloud is a Git based code hosting and collaboration tool, built for teams.
- We provide one place for your team to collaborate on code from concept to cloud, build quality code through automated testing .

TFS

- Previously known as Team Foundation Server, Azure DevOps Server is a set of collaborative software development tools.

C# :

- C# is a programming language of .NET Framework.
- C# is pronounced as “C-Sharp”. It is an object-oriented programming language provided by Microsoft that runs on .Net Framework.

- By using C#, to develop different types of application
- Windows applications
- Web applications
- Distributed applications

Java Vs C#

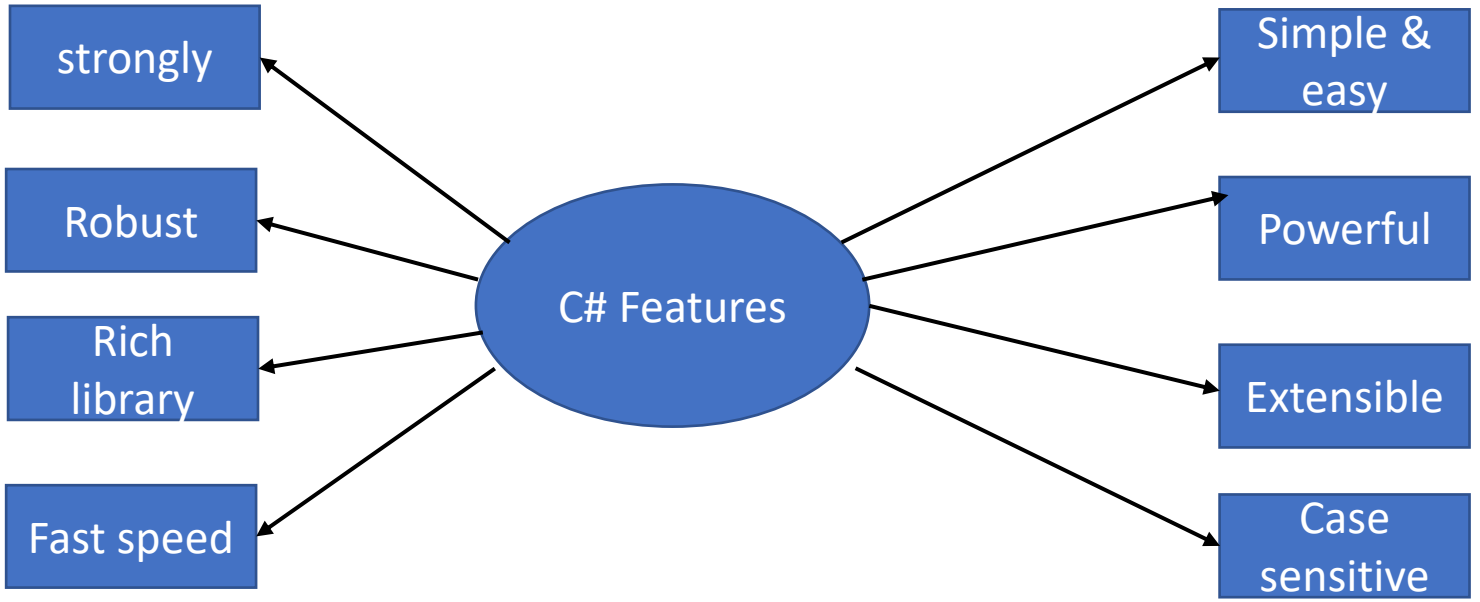
Java

- Java programming language is designed to be run on a java platform , by help of JVM.
- Java doesn't support goto statement.
- Java doesn't support structure and unions.
- In java, built-in data types that are passed by value are called primitive types.

C#

- C# programming language is designed to be run on on the CLR.
- C# supports goto statement.
- C# supports structure and unions.
- In C#, built-in data types that are passed by value are called simple types.

C# features:


strongly
Robust
Rich library
Fast speed
C# Features
Simple & easy
Powerful
Extensible
Case sensitive

- Simple and Easy: C# is a simple and easy to learn programming language. C# programs contain English like commands/instructions, hence easy to learn, code, understand, modify, debug and test.
- Powerful: C# is a powerful language. In our today life we find various types of data for number, text, list etc. C# provides rich set of data types, which help us to store variety of data and perform operations on them.
- Extensible and more flexible: C# programs are extensible to any extent from a small and simple program to a large and complex application and C# programs or applications more flexible to modify. C# programs are extended by creating methods, structures, classes, interfaces, etc..
- Case sensitive: C# treats lower case letters and upper case letters separately. If we create three identifiers num, NUM and Num then they are considered as three different identifiers.

- Strongly or strictly typed language: In C# programs while creating variables, constants or parameters, programmers must and should explicitly define their data type.
- Robust: means reliability

  Application should never get crash after deploying on client machine.

That is reliability of application indicates robustness of the language.

- Rich Library: C# provides a lot of inbuilt functions that makes the development fast.
- Fast Speed: The compilation and execution time of C# language is fast.

- C# is an object-oriented programming language. In Object-Oriented Programming methodology, a program consist of various objects that interact with each other by means of actions. The actions that an object may take are called methods.
- Ex: using System

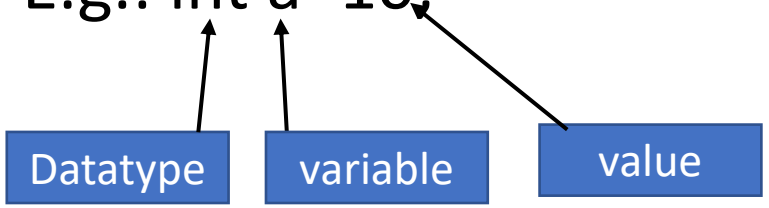
```
namespace  sample
{
        class Demo
      {
            static void Main(String[] args)
             {
                Console. WriteLine(“hello”);
             }
      }
}
```

- The **using** keyword is used for including the namespace in the program. A program can include multiple using statements.
- **Class** is a keyword which is used to define class.
- **Program** is the a class name. A class is a blueprint or template from which objects are created. It contains data members and data methods.
- **static** is a keyword which means object is not required to access static members.
- **Void** is the return type of the method. It doesn’t return any value.
- **Main** is the method name. It is the entry point for any C# program. Whenever we run the C# program, Main() method is invoked first before any other method. It represents start up of the program.

- **String[] args** is used for command line arguments in C#. While running the C# program, we can pass values. These values are known as arguments which we can use in the program.
- **Console. WriteLine** is printing statement. Console is a class defined in System namespace. WriteLine() is a static method of Console class which is used to write the text on the console.

Variable:

- A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.
- E.g.: int a=10;

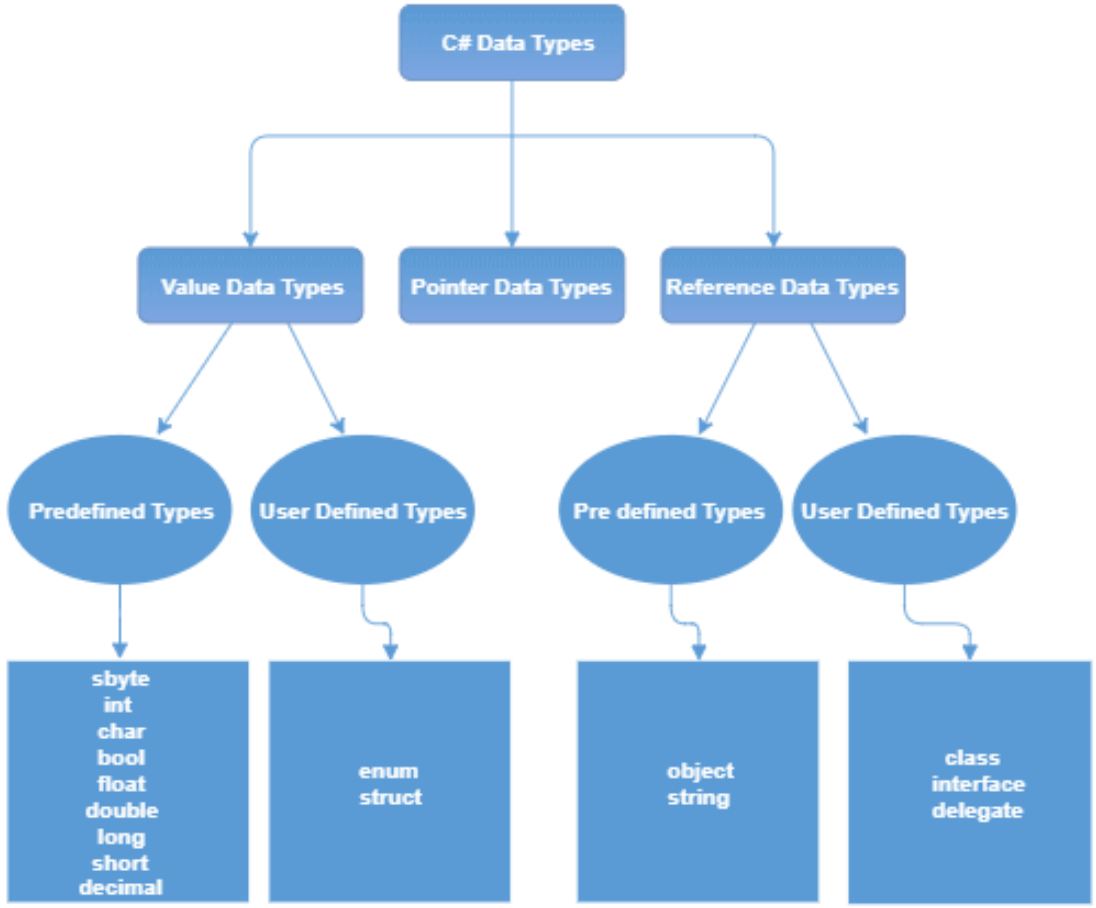


Rules to Declare C# variable

- You can define a variable name with a combination of alphabets, numbers and underscore.
- A variable name must always start with either alphabet or underscore but not with numbers.
- While defining the variable, no white space is allowed within the variable name.
- Don’t use any reserved keywords such as int, float , char etc.. For a variable name.
- In c#, once the variable is declared with a particular data type, it cannot be re-declared with a new type.

Datatype:

A data type specifies what type of data store in variable such as integer, float, character etc..



There are 2 types of datatypes in C#

- Value Data type : short , int, char, float, double etc
- Reference Data type : String, Class, Object and Interface

Operators

- An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc..

Types of operations:

- Arithmetic operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Unary Operators
- Ternary Operators
- Misc. Operators

Arithmetic Operators

Operator	Description	Example
+	Adds two operands	A+B
-	Subtraction	A-b
*	Multiplication	A*B
/	Division	A/B
%	Modulo Division	A%B
++	Increment	A++
--	Decrement	A--

Relational Operators

Operator	Description	Example
==	Equal	A==B
!=	Not equal	A!=B
>	Greater than	A>B
<	Less than	A<B
>=	Greather than or equal	A>=B
<=	Less than or equal	A<=B

Logical Operators

Operator	Description	Example
&&	Logical AND	A&&B
||	Logical OR	A||B
!	Logical Not	!(A&&B)

Bitwise Operators

Operator	Description	Example
&	Bitwise AND	A&B
|	Bitwise OR	A|B
^	Bitwise XOR	A^B
~	complementary	
<<	Left shift	A<<2
>>	Right Shift	A>>2

Assignment Operators

Operator	Description	Example
=	Equal	C=A+B
+=	Add and assign	C+=a
-=		

Miscellaneous Operators

- Sizeof() Returns the size of a data type
- Typeof() Returns the type of a class
- & Returns address of an variable
- * Pointer to a variable
- ?: Conditional Expressiong
- Is Determines whether an object is of a certain type.
- As cast without raising an exception if the cast fails.

Conditional Statement:

- C# provides following types of decision making statements.
- If
- If else statement
- Nested if statement
- Switch statement
- Nested switch statement

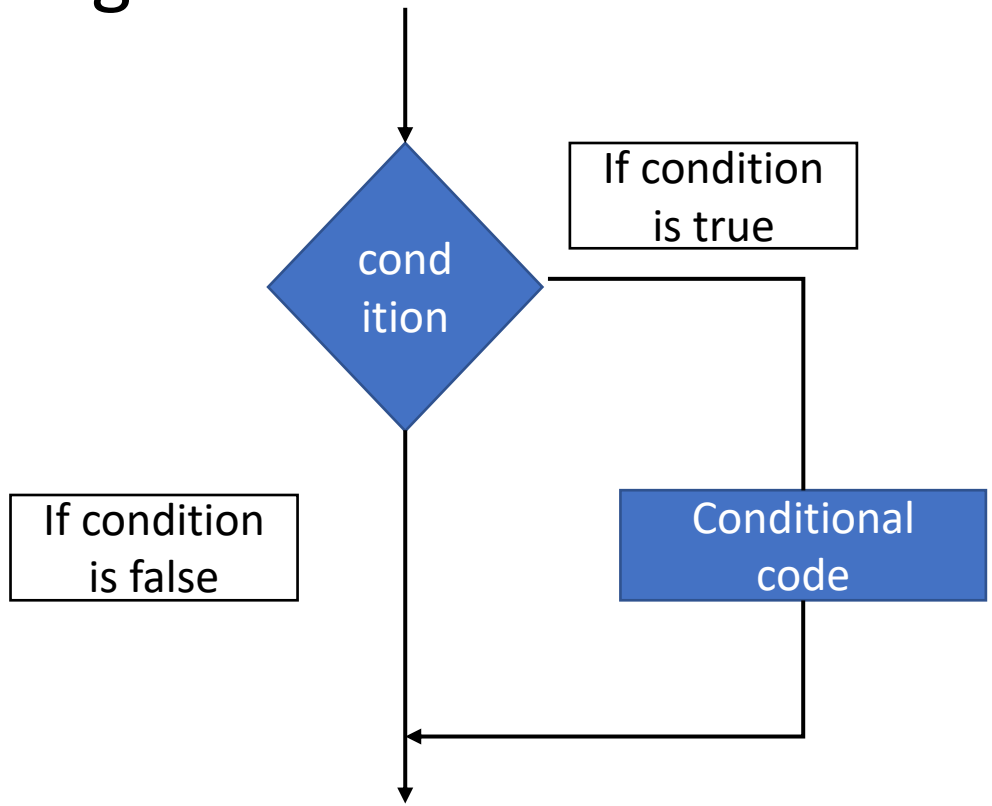
If statement:

An if statement consists of a Boolean expression followed by one or more statements.

Syntax:

```
If(bool_expression)
{
  // some code
}
```

- Flow Diagram

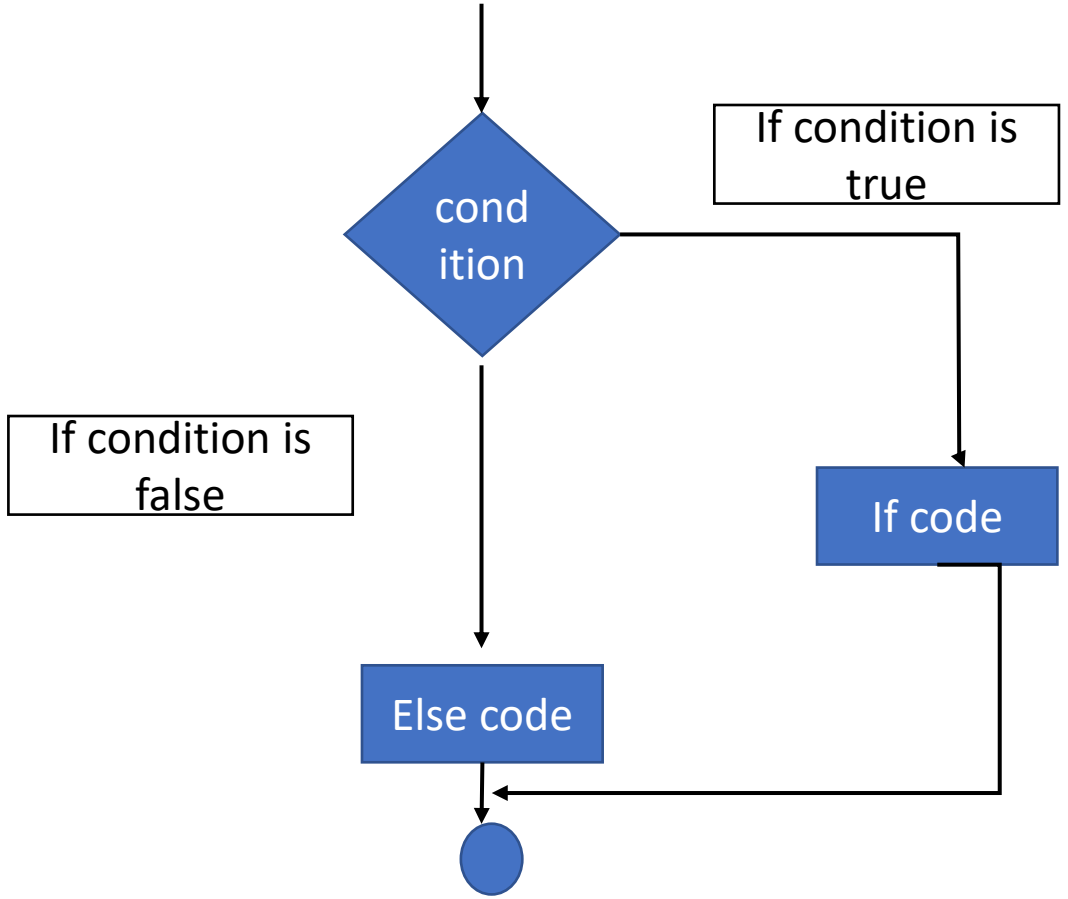


If else statement

An If statement can be followed by an optional else statement, which executes when the Boolean expression is flase.

Syntax:

```
If(Boolean_expression)
{
  //code
}
Else{
  //code
}
```



Nested if statements

- It is always in c# to nest if-else statement, which means you can one if or else if statement inside another if or else if statement.
- Syntax:

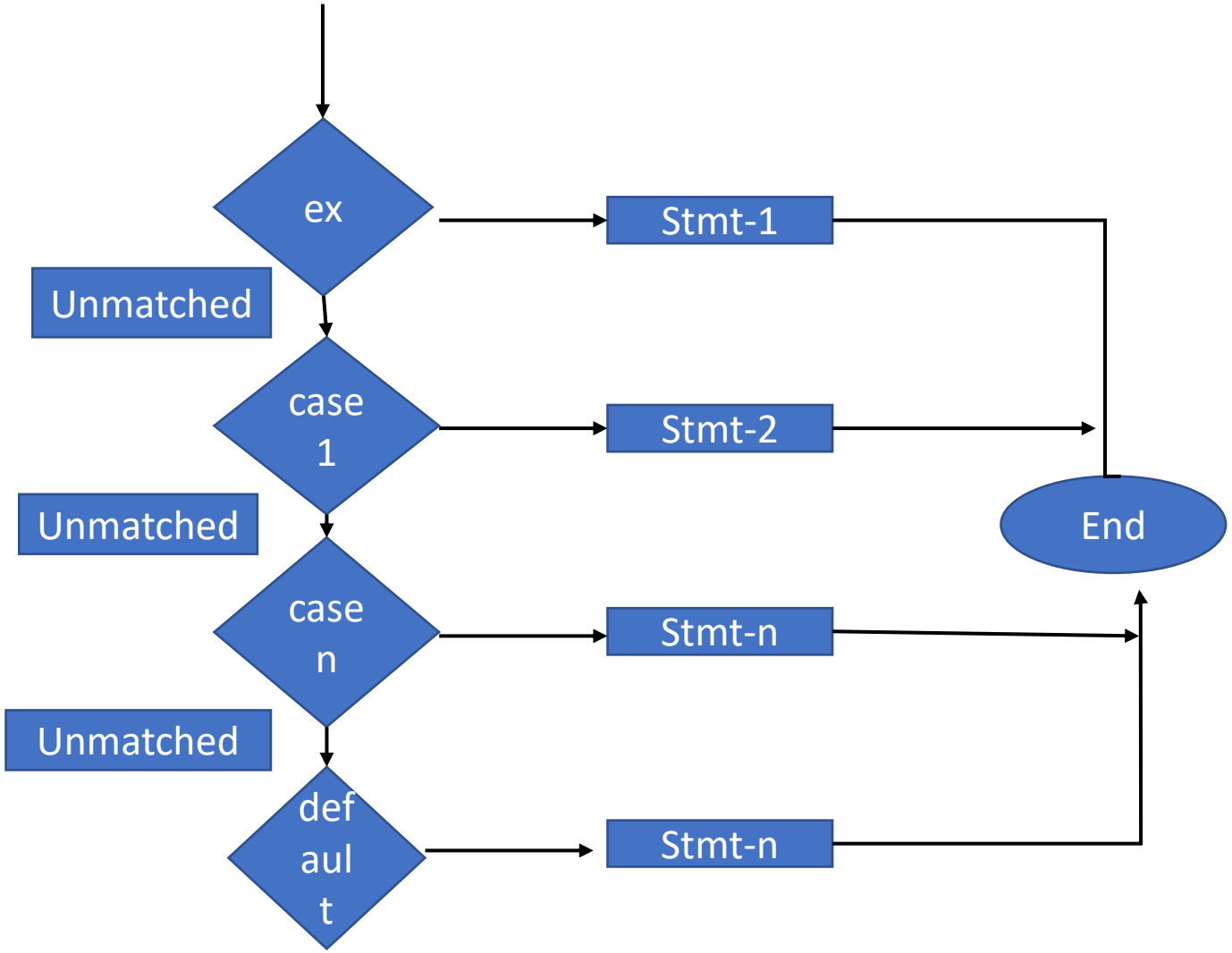
```
If(codition){
        if(condition){
        }
        else{
       }
}
Else{
}
```

Switch statement

- In C#, Switch is a selection statement, and it will execute a single case statement from the list of multiple case statements based on the pattern match with the defined expression.
- Syntax:
- Switch(variable/expressiong)

```
 {
        case 1://statements
                    break;
         :
         default://statements
                        break;
}
```

Flow Diagram:



Iteration Loops in C#

- For loop
- While loop
- Do-while loop
- Foreach loop

For loop

- In C#, for loop is useful to execute a statement or a group of statements repeatedly until the defined condition returns true.
- Syntax:

```
for(Initialization; condition; inc/dec){
        //statements
}
```

While loop

- In C#, while loop is used to execute a block of statements until the specified expression return as true.
- Syntax:

```
while(condition){
   //statements
 }
```

Do-while:

- In C#, the Do-while loop is used to execute a block of statements until

The specified expression return as ture.

Only the difference is while loop will exectute the statements only when the defined condition returns true, but the do-while will execute the statements

At least once .

Foreach

- In C#, the foreach loop is useful to loop through each item in an array or collection object to execute the block of statements repeatedly.
- Syntax:

```
foreach(Type var_name in collection_object){
        //statements to execute
}
```

Jumping Statements in C#

- In C#, Break statement is useful to break or terminate the execution of loops.
- In C#, Continue statement is used to pass control to the next iteration of loops.
- In C#, the Goto statement is used to transfer program control to the defined labelled statement and it is useful to get out of the loop or exit from deeply nested loops based on our requirements.
- In C#, Return statement is useful to terminate the execution of method in which it appear and return the control back to the calling method.

Methods

- In C#, Method is a separate code block, and that contains a series of statements to perform particular operations. Methods must be declared either in class or struct by specifying the required parameters.
- Generally, methods are useful to improve code reusability by reducing code duplication.
- Syntax:

<Access_specifier> <Return Type> Method_name()

{

    //statements

}

Types of Methods:

In C#, we have different ways to pass parameters to methods; those are:

- Pass by value
- Pass by reference
- Out parameter
- Params keyword

Pass by reference

- Pass by reference: In c#, passing a value type parameter to a method by reference means passing a reference of the variable to the method.
- So the changes made to the parameter inside the called method will affect the original data stored in the argument variable.
- Syntax:

```
int x=10;//variable need to initialized
Multiplication(ref x)
```

Pass by out

- In C#, out keyword is used to pass arguments to the method as a reference type. The out keyword same as the ref keyword , but the only difference is out doesn’t require a variable to be initialized before we pass it as an argument to the method.
- Still, the variable must be initialized in called method before it returns a value to the calling method.

syntax:

```
int x; //No need to initialize variable
Multiplication(out x);
```

Pass by in

- In C#, “in parameter” has been introduced which allows passing read-only reference of a variable.
- Before , C# we used “ref” and “out” keywords for passing reference of a variable. “out” is meant for output only whereas “ref” is meant for input and output both.
- Syntax;
- Int x=10;

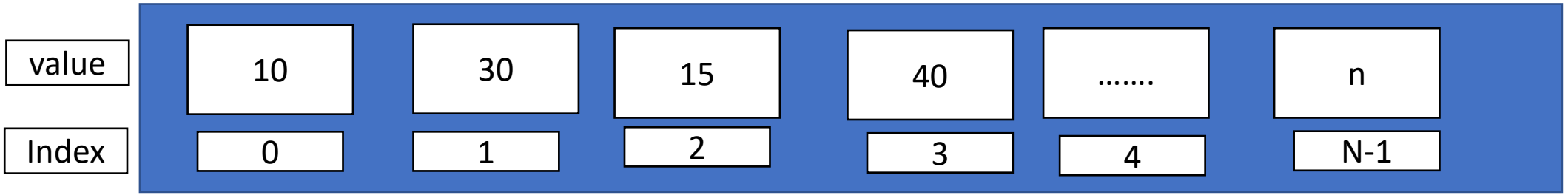
  Multiplication(x)

Params

- In C#, params keyword is useful to specify a method parameter that takes a variable number of arguments. The params keyword is useful when we are not sure about the number of arguments to send as a parameter.
- In C#, during method declaration, only one params keyword is allowed, and no additional parameters are permitted after the params keyword in a method declaration.

Arrays

- In c#, Arrays are useful for storing multiple elements of the same data type at contiguous memory location and arrays. It will store a fixed number of elements sequentially based on the predefined number of items.
- An array can start storing the values from index 0. If we have an array with n elements, it will start storing the elements from index 0 to n-1.



Declaration of array

//Declaring and Initializing an array with size of 4

- Int[] array=new int[4]

//Defining and assigning an elements at the same time

- Int[] array2=new int[5]{1,2,3,4,5};

//Initialize with 5 elements

- Int[] array3=new int[]{1,2,3,5};
- Int[] array4={1,2,3,4,5};
- Int[] array5;
- array5=new int[]{1,2,3};

Array Types

- Sigle-Dimensional Arrays
- Multi-Dimensional Arrays
- Jagged Arrays

Multi-Dimensional Array

- In c#, Multidimensional Arrays can be declared by specifying the data type of elements followed by the square brackets [] with comma(,) separator.
- Syntax:
- int[,] arr=new int[4,2]
- //three Dimensional

int[, ,] arr1=new int[4,2,3]

Jagged Array

- In c#, Jagged Array is an array whose elements are arrays with different dimensions and sizes. Sometimes a jagged array is called an array of arrays and can store arrays instead of a particular data type value.
- Syntax:
- Int[][] a=new int[2][];
- Int[][,] a=new int[3][,];//jagged Array with two dimensional array.

List

- In c#, List is a generic type of collection, so it will allow storing only strongly typed object, i.e,. Elements of the same data type.
- The size of the list will vary dynamically based on our application requirements, like adding or removing elements from the list.
- List<T> l=new List<T>();
- Ilist<T> l=new List<T>();

Dictionary

- In c#, Dictionary is a generic type of collection, and it is used to store a collection of key/value pairs organized based on the key. The dictionary in c# will allow to store only strongly-typed objects, i.e, the key/value pairs of the specified data type.
- In C#, while storing the elements in the dictionary object, you need to make sure that the keys are unique because the dictionary object will allow us to store duplicate values, but the keys must be unique.
- The size of the dictionary object will vary dynamically so that you can add or remove elements from dictionary based on our requirements.
- Syntax:
- Dictionary<Tkey, Tvalue> d=new Dictionary<Tkey, Tvalue>();

Structure

- In c#, structures are same as classes, but the only difference is classes are the reference types and structures are the value types.
- As a value type, the structures directly contain their value, so their object or instance is stored on the stack, and structures are faster than classes.
- Syntax:
- Public struct users{

  //Methods etc..

  }
- In c#, structures can be instantiated with or without new keyword.

Enum(Enumerator)

- In c#, enum is a keyword that is useful to declare an enumeration. In c#, the enumeration is a type that consists of a set of named constants as a list.
- By using an enumeration, we can group constants that are logically related to each other.
- Syntax:

```
enum enum_name{
  //enumeration list
}
```

Properties

- Property is a member that provides a flexible mechanism to read, write or compute the value of private field.
- In c#, properties can contain one or two code blocks called accessors, and those are called a get accessor and set accessor.
- Generally, in object-oriented programming languages like c# you need to define fields as private and then use properties to access their values in a public way with get and set accessors.
- Syntax:

<access_modifier><return_type><property_name>

{

    get{ return property value}

    set{set a new value}

}

C# Access Modifiers

- In c#, Access Modifiers are the keywords used to define an accessibility level for all types and type members.
- By specifying an access level for all types and type member, we can control whether they can be accessed in other classes or the current assembly or other assemblies based on our requirements.
- Different types of Access Modifiers:
- Public : It is used to specifies that access is not restricted.
- Private: It is used to specifies that access is limited to the containing type.

- Protected : It is used to specifies that access is limited to the current class or types derived from the containing class.
- Internal : It is used to specifiers that access is limited to the current assembly.
- Protected internal : It specifies that access is limited to the current assembly or types derived from the containing class.
- Private protected: It is used to specifies that access is limited to the containing class or types derived from the containing class within the current assembly.

OOPs in c#

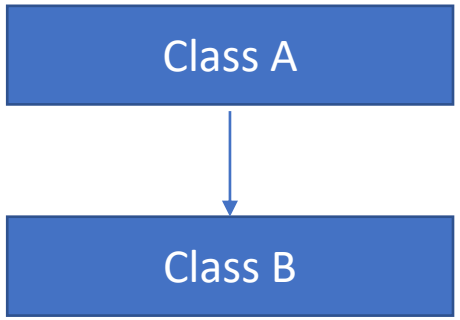
- Encapsulation : It is a process of binding the data members and member functions into a single unit. In c# , the class is the real-time example for encapsulation because it will combine various types of data members and member functions into a single unit.
- Abstraction : It is a principle of object-oriented programming language and it is used to hide the implementation details and display only essential features of the object.
- Inheritance : It is one of the primary concepts of object- oriented programming and it is used to inherit the properties from one class to another class.

Types of Inheritance

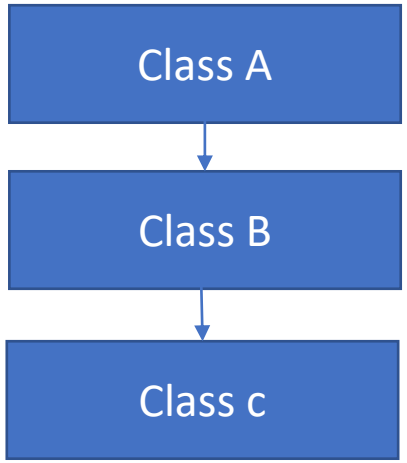
- Single
- Multi-level
- Hierarchical
- Hybrid
- Multiple
- Multipath

Single Inheritance

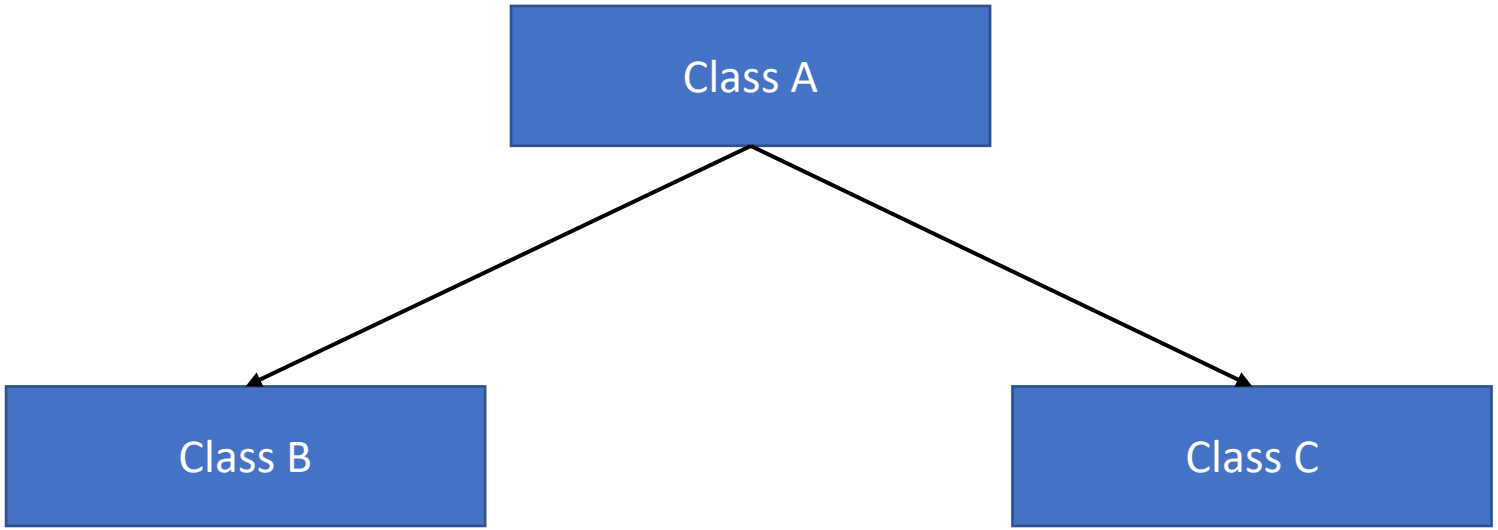
- In this inheritance, a derived class is created from a single base class.



- Multi-level : A derived class is created from another derived class.



- Multiple : If class has more than 1 immediate parent class to it we call it as multiple inheritance.


Class A
Class B
Class C

- Hierarchical : More than one derived classes are created from a single base class.
- Hybrid : It is combination of more than one inheritance. It may be a combination of multilevel and Multiple or Hierarchical.
- Advantages of Inheritance:
- Reduce code redundancy.
- Provides code reusability.
- Reduces source code size and improves code readability.

- Method overloading : It means defining multiple methods with the same name but different parameters. Using method overloading, we can perform different tasks with the same method name by passing different parameters.
- Method overriding: It means override a base class method in derived class by creating a method with the same name and signatures to perform a different task. The Method overriding in c# can be achieved using override and virtual keywords and inheritance principle.

Polymorphism

- Polymorphism means to ability to create more than one form.
- Generally, poly means multiple & morphism means forms.
- Two types of polymorphism.
- Compile Time polymorphism : It can be achieved by using method overloading and it is also called early binding.
- Run Time polymorphism : It can be achieved by using method overriding and it is also called as late binding.

Class and object

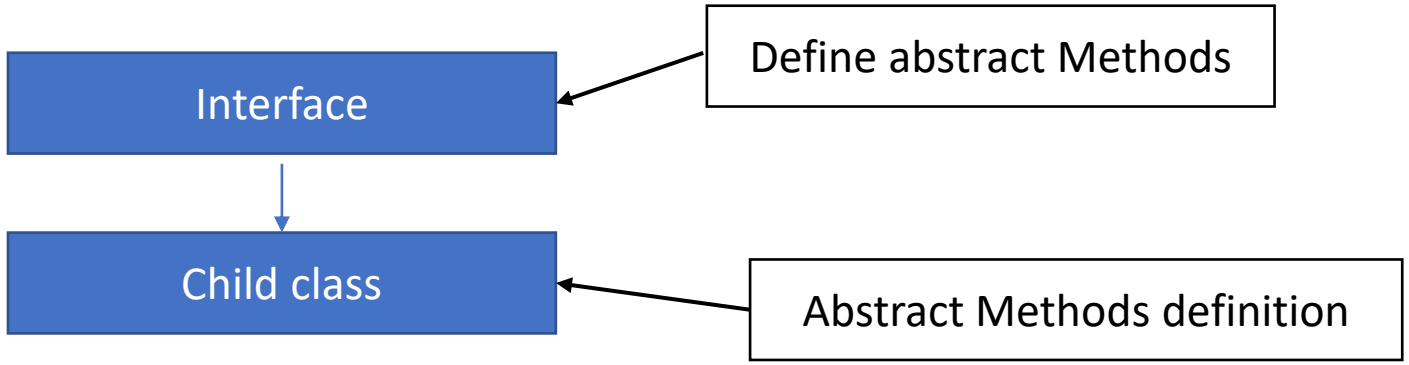
- Class is user defined type. It is nothing but a collection of various data members.
- Object : Object is instance of class.

- Abstract Class : It contains Non-Abstract methods(with definition) and Abstract Methods(without definition).
- If abstract method is exist in any class, then that class is called as abstract class.

- Abstract Method : A method without any method definition is known as an abstract method.

Interface

- Interface is also user-defined type.
- It contains only Abstract methods (without definition).
- Every abstract method of an interface should be implemented by the child class of the interface without fail.



- [<modifier>] interface <Name>

  {

  //Abstract Member declaration here….

  }

Constructor

- Constructor is special method that will invoke automatically whenever an instance of class or struct is created.
- Constructor will have the same name as the class or struct and it useful to initialize and set default values for the data members of the new object.
- Syntax:
- public class User{

```
        //constructor
        public User(){
        }
}
```

Types of constructors

- Default constructor
- Parameterized constructor
- Copy constructor
- Static constructor
- Private constructor…

Namespaces

- Namespaces is container. It contains class and methods.
- It is used to organize the multiple classes in our applications.
- The global namespace is the root namespace: system.

Delegates

- It is a type safe function pointer.
- A delegate holds the reference of a method and then calls the method for execution.
- Define a delegate

  [<modifier>] delegate void/type <Name>([<parameterList>])

  Ex: public delegate string sayDelegate(string name);

Two types of Delegates:

1. Single Cast Delegate
2. Multicast Delegate.

- Single cast Delegate : It is points to a single method then it is called as single cast Delegate. It is used to hold the reference of a single method.
- Multicast Delegate : It is points to multiple methods then it is called as multicast delegate. It is used to hold the reference of multiple methods..

C# String Format

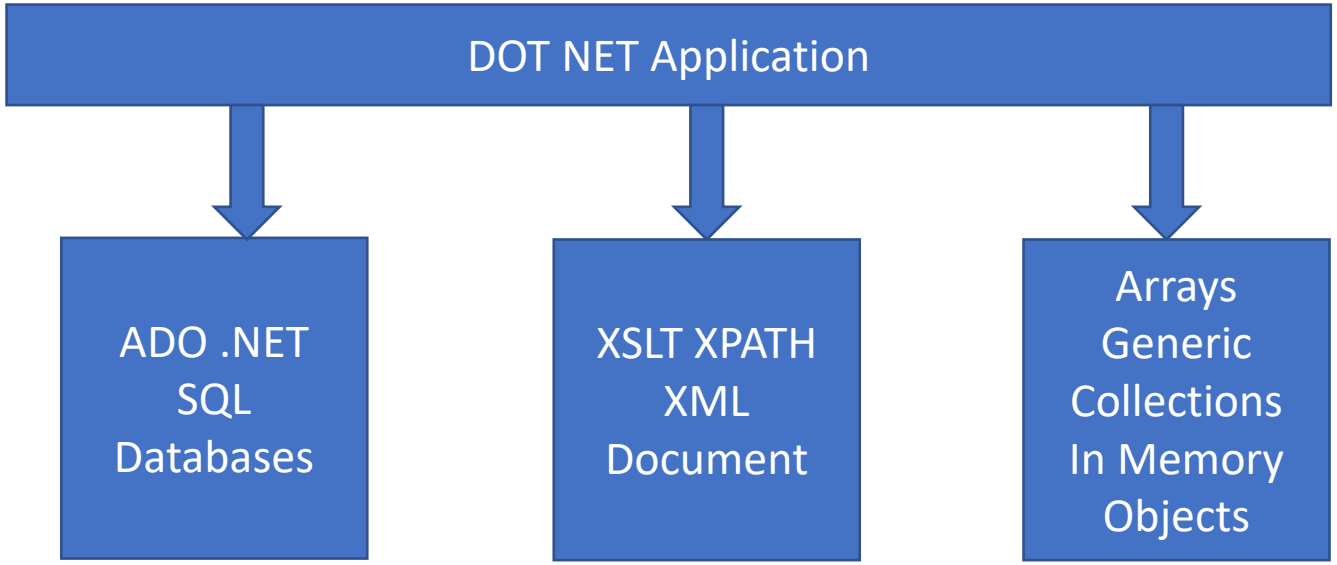
- In C#, the string Format method is used to insert the variable's value or an object or expression into another string. Using the string Format method, we can replace the format items in the specified string with the string representation of specified objects.
- Syntax:
- Public string Format(string, object)
- Public string Format(string, object, object)
- Public string Format(IFormatProvider, string, object)

- Ex: string s=“age is {0}\n” + “Name is {1}\n”;
- Console. WritLine(string.Format(s,22,”veerababu”));
- Output: 22

  veerababu

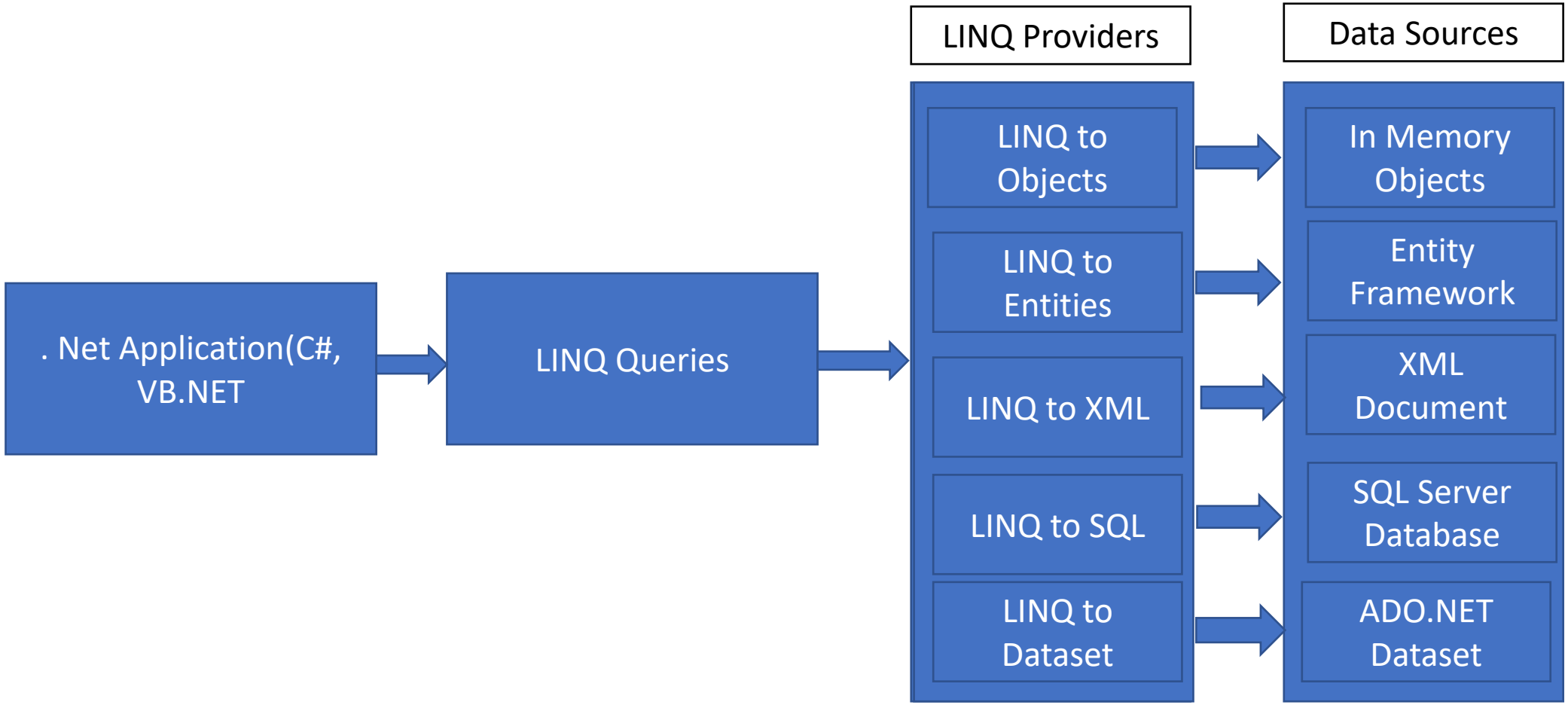
LINQ

- The LINQ(Language Integrated Query) is a part of a language but not a complete language. It was introduced by Microsoft with .NET Framework 3.5 and C# 3.0 and is available in System.Linq namespace.
- LINQ provides us common query syntax which allows us to query the data from various . That means using a single query we can get or set the data from various data sources such as SQL Server database, XML documents, ADO.NET Datasets, and any other in-memory objects such as Collections, Generics, etc.


DOT NET Application
ADO .NET
SQL
Databases
XSLT XPATH
XML
Document
Arrays
Generic
Collections
In Memory
Objects

- LINQ provides a uniform programming model(i.e. common query syntax) Which allows us to work with different data sources but using a standard or you can say unified coding style. As a result, we don’t require to learn different syntaxes to query different data sources.

How LINQ works ?



- The LINQ provider is a software component that lies between the LINQ queries and the actual data source. The Linq provider will convert the LINQ queries into a format that can be understood by the underlying data source.
- Ex: LINQ to SQL provider will convert the LINQ queries to SQL statements which can be understood by the SQL server database.

LINQ Providers

- A LINQ provider is software that implements the IQueryProvider and IQueryable interface for a particular data source.
- In other words, it allows us to write LINQ queries against that data source.
- If you want to create your custom LINQ provider then it must implement IQueryProvider and Iqueryable interface.
- Without LINQ provider we cannot execute our LINQ Queries.

- LINQ to Objects: The LINQ to objects provider allows us to query an in-memory object such as an array, collection, and generics types. It provides many built-in functions that we can use to perform many useful operations such as filtering, ordering, grouping with minimum code.
- LINQ to SQL: The LINQ to SQL provider is designed to work with only SQL Server database.
- LINQ to Datasets : The LINQ to Datasets provider provides us the flexibility to query data cached in a Dataset in an easy and faster way.
- LINQ to Entities : The LINQ to Entities provider looks like LINQ to SQL.

- LINQ to XML: The LINQ to XML provider is basically designed to work with an XML document. So, it allows us to perform different operations on XML data sources such as querying or reading, manipulating, modifying, and saving the changes to XML documents. The System.Xml.Linq namespace contains the required classes for LINQ to XML.
- Parallel LINQ (PLINQ): The Parallel LINQ was introduced with .NET Framework 4.0 . This provider provides the flexibility of parallel implementation of LINQ to Objects.

Advantages of LINQ:

- We don’t need to learn new query language syntaxes for different data sources as it provides common query syntax to query different data sources.
- Less code as compared to the traditional approach. That means using LINQ we can minimize our code.
- It provides compile time error checking as well as intelligence support in visual studio. This powerful feature helps us to avoid run-time errors.
- Its query can be reused.

Disadvantages of LINQ:

- Using LINQ it’s very difficult to write complex queries like SQL.
- LINQ doesn’t take the full advantage of SQL features like cached execution plan for the stored procedure.ss

Different ways to write LINQ Query

In order to write a LINQ query, we need the following three things

- Data Source (in-memory objects, SQL, XML)
- Query
- Execution of the query

What is Query ?

A query is nothing but a set of instructions which is applied on a data source (i.e. in-memory objects, SQL, XML, etc.) to perform certain operations (i.e. CRUD operations) and then tells the shape of the output from that query. That means the query is not responsible for what will be the output rather it is responsible for the shape of the output. Means what is going to return from that query whether it is going to return a particular value, or a particular list or an object.

Each query is a combination of three things. They are as follows:

1.Initialization (to work with a particular data source)
2.Condition (where, filter, sorting condition)
3.Selection (single selection, group selection or joining)

What are the different ways to write a LINQ query?

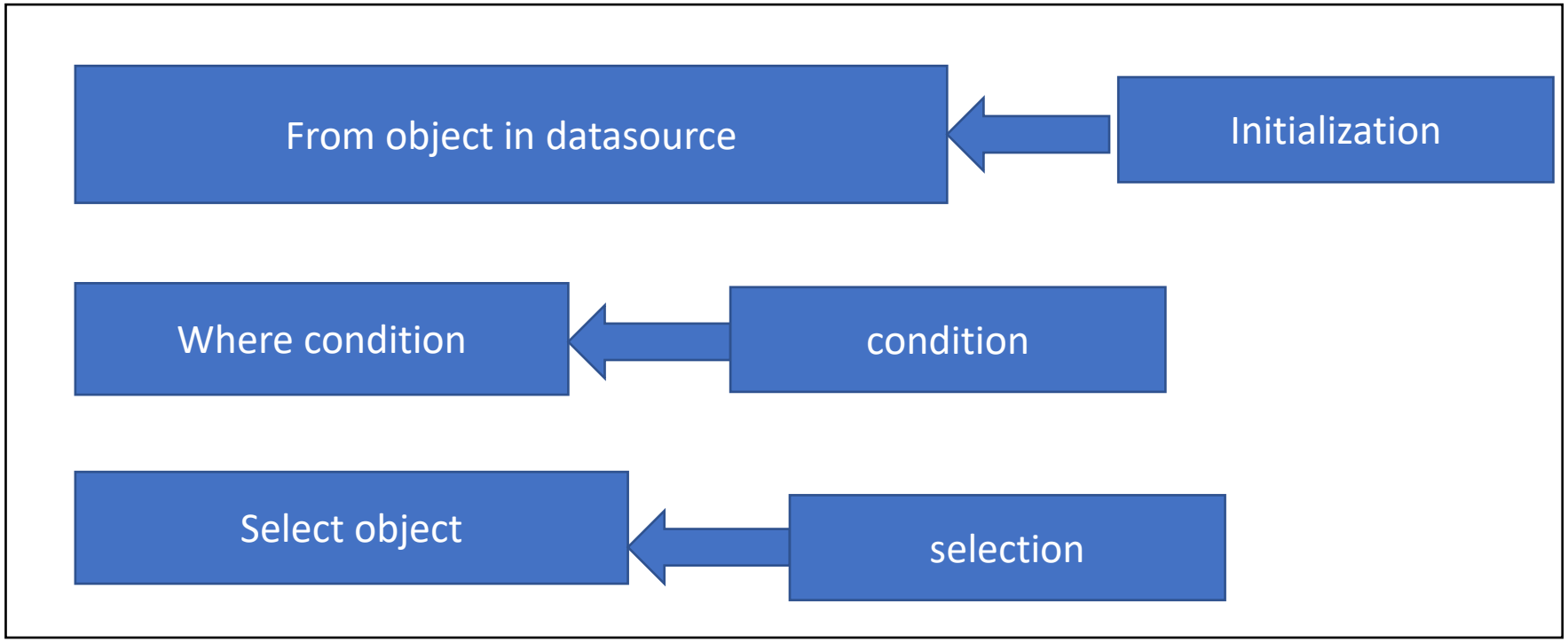
- We can write the LINQ query in three different ways. They are as follows

1.Query Syntax
2.Method Syntax
3.Mixed Syntax (Query + Method)

LINQ Query syntax

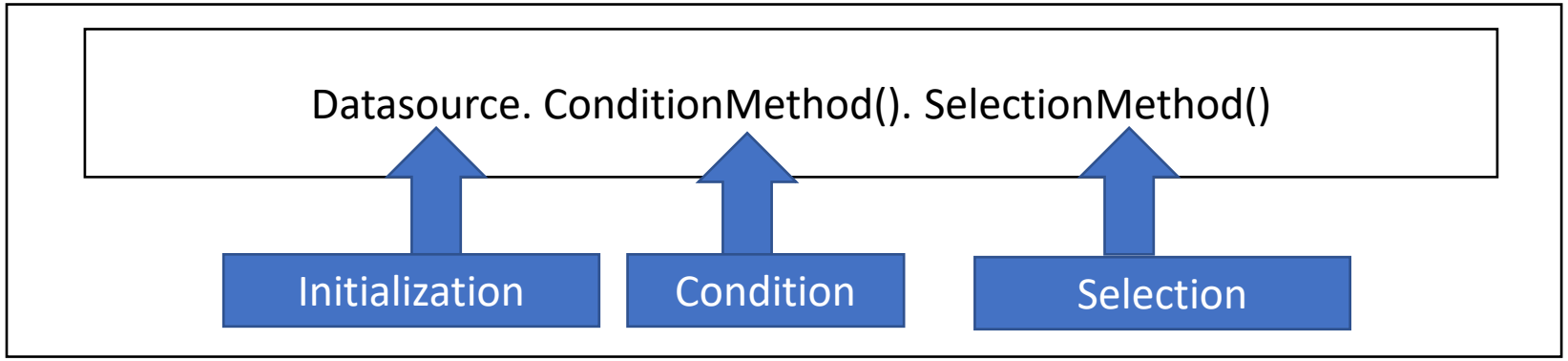
- This is one of the easy ways to write complex LINQ queries in an easy and readable format. The syntax for this type of query is very much similar to SQL Query. If you are familiar with SQL queries then it is going to be easy for you to write LINQ queries using this query syntax. The syntax is given below.

- Syntax


From object in datasource
Initialization
Where condition
condition
Select object
selection

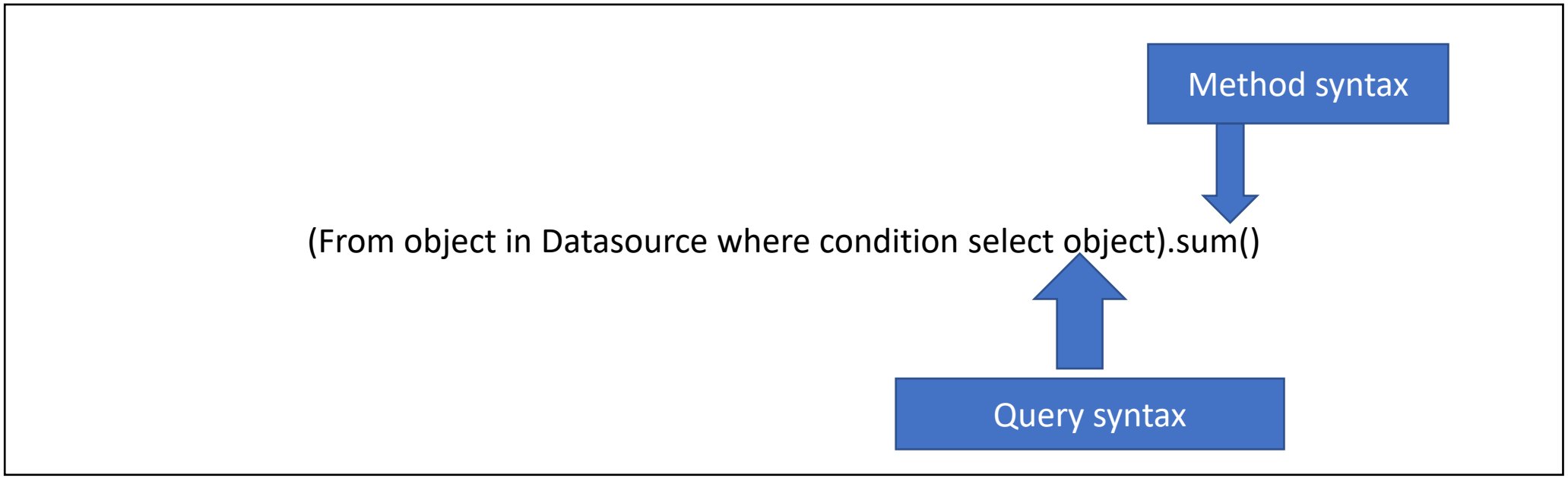
LINQ Method syntax

- Method syntax becomes most popular now a day to write LINQ queries. It uses a lambda expression to define the condition for the query. Method syntaxes are easy to write simple queries to perform read-write operations on a particular data source. But for complex queries Method syntaxes are a little hard to write as compared to query syntax.



Mixed syntax:

- This is a combination of both Query and Method syntax.



IEnumerable and IQueryable in C#

- What is IEnumerable
- IEnumerable is an interface that is available in System. Collection namespace. The IEnumerable interface is a type of iteration design pattern. It means we can iterate on the collection of the type IEnumerable.
- IEnumerable has one method called GetEnumerator which will return an IEnumerator that iterates through collection.
- The IEnumerable or IEnumerable<T> interface should be used only for in-memory data objects.

IQueryable:

- The **IQueryable** is an interface and it is available in **System.Linq** namespace.
- The IQuerable interface is a child of the IEnumerable interface.
- So we can store IQuerable in a variable of type IEnumerable.
- The IQuerable interface has a property called **Provider** which is of type **IQueryProvider** interface.
- Let us see the definition of IQueryProvider.
- The methods provided by the **IQueryProvider** are used to create all Linq Providers. So, this is the best choice for other data providers such as **Linq to SQL, Linq to Entities**, etc. Why that we will discuss in our next article.

IQueryable

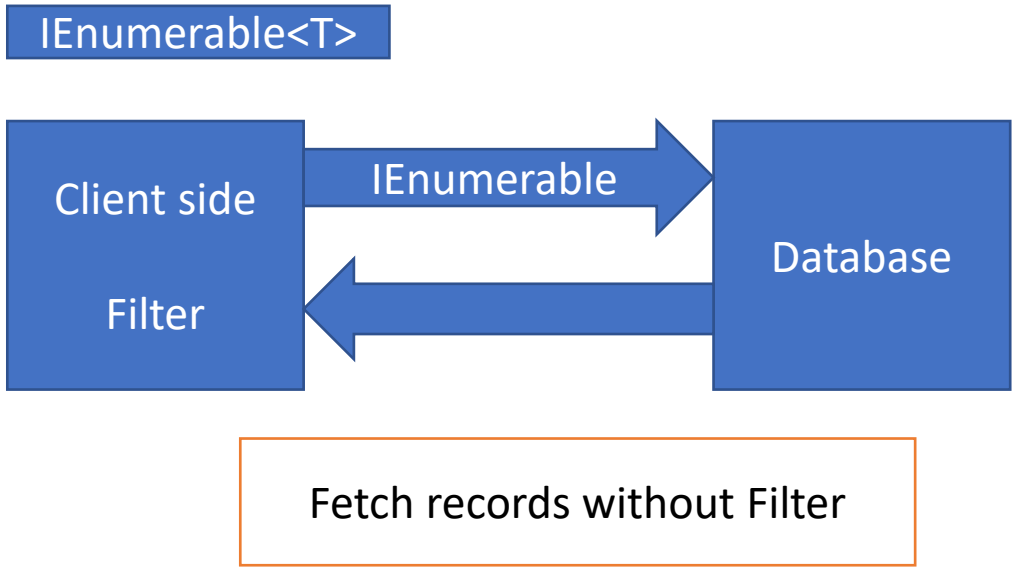
Let us see the definition of IQueryProvider.

- The methods provided by the **IQueryProvider** are used to create all Linq Providers.
- So, this is the best choice for other data providers such as Linq to SQL ,Linq to entities, etc.

Difference between IEnumerable and IQueryable

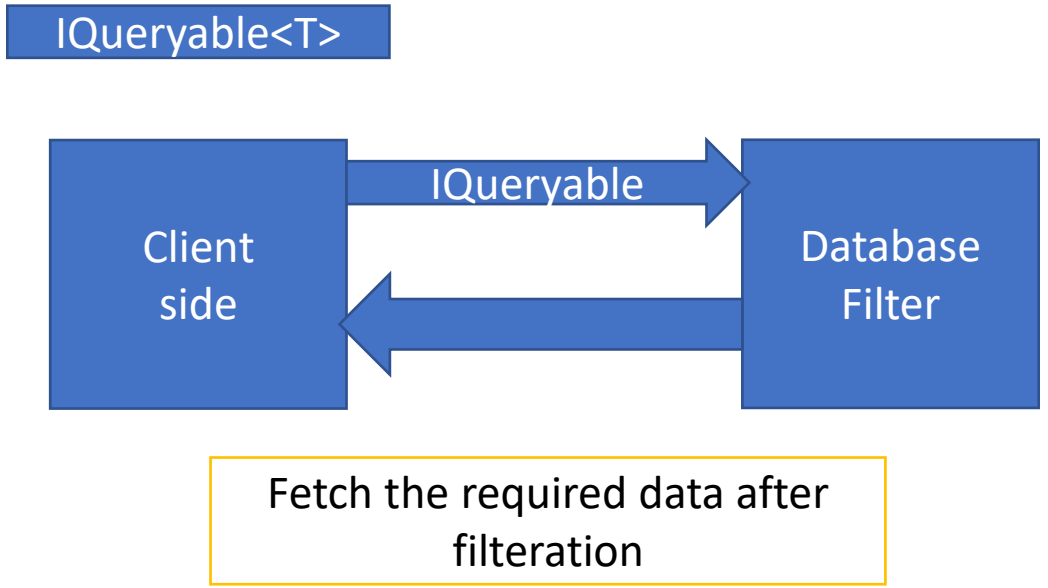
- The IEnumerable and IQueryable are used to hold a collection of data and also used to perform data manipulation operations such as filtering, Ordering, Grouping, etc. based on the business requirements.

- IEnumerable



Fetch records without Filter

- IQueryable



Fetch the required data after filteration

LINQ Extension Methods:

- Extension methods allow us to add methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.
- In simple words, we can say that the Extension methods can be used as an approach of extending the functionality of a class by adding new methods in the future if the source code of the class is not available or if we don’t have any permission in making changes to the class.
- All the LINQ Extension methods are implemented in the Enumerable class, so ,the syntax to call those methods.

LINQ Operators

- The LINQ operators are nothing but a set of extension methods that are used to write LINQ query. These LINQ extension methods provide lots of very useful features which we can apply to the data source.
- Some of the features are filtering the data, sorting the data, grouping the data, etc.

Categories of LINQ Operators

- Projection operators
- Ordering operators
- Filtering operators
- Set operators
- Quantifier operators
- Partitioning operators
- Equality operators
- Element operators
- Conversion operators
- Concatenation operators

- Aggregation operators
- Generation operators
- Join operators
- Custom Sequence operators
- Miscellaneous operators

LINQ Select Projection Operator

Projection:

- Projection is nothing but the mechanism which is used to select the data from a data source.
- You can select the data in the same form(i.e. the original data in its original state.)
- It is also possible to create new form of data by performing some operations on it.
- There are two methods or operators available in projection.

  1. Select
  2. SelectMany

Select operator:

- As we know the Select clause in SQL allows us to specify what columns we want to retrieve, whether you want to retrieve all the columns or some of the columns that you need to specify the select clause.
- In the same way, the LINQ Select operator also allows us to specify what properties we want to retrieve, whether you want to retrieve all the properties or some of the properties that you need to specify in the select operator. The standard LINQ Select Operator also allows us to perform some calculations.

Query Syntax:

```
IEnumerable<Employee> basicQuery = (from emp in Employee.GetEmployees()
                                    select emp);
```

At this point the query is just created not executed. In order to execute the query we need to applied some methods to the query such as ToList() method or foreach loop. If you are not applying the ToList() method on the query then the data type is going to be IEnumerable<Employee>

```
List<Employee> basicQuery = (from emp in Employee.GetEmployees()
                             select emp)
                            .ToList();
```

⇦ Query

⇦ Execution

At this time the query is created as well as executed as we applied the ToList() method on the query.

Method Syntax:

```
IEnumerable<Employee> basicMethod = Employee.GetEmployees().ToList();
```

```
//Query Syntax
IEnumerable<Employee> selectQuery = (from emp in Employee.GetEmployees()
                                     select new Employee()
                                     {
                                         FirstName = emp.FirstName,
                                         LastName = emp.LastName,
                                         Salary = emp.Salary
                                     });
```

Here we are selcting the First Name, Last Name and Salary properties to the same Employee class by excluding the ID property using Query Syntax

```
//Method Syntax
List<Employee> selectMethod = Employee.GetEmployees().
                              Select(emp => new Employee()
                              {
                                  FirstName = emp.FirstName,
                                  LastName = emp.LastName,
                                  Salary = emp.Salary
                              }).ToList();
```

Here we are selcting the First Name, Last Name and Salary properties to the same Employee class by excluding the ID property using Method Syntax

How to Select Data to another class using LINQ Projection Operator?

- It is also possible to select the data to another class using the LINQ Select operator. In our previous example, we have selected the First Name, Last Name, and Salary properties to the same Employee class. Let us create a new class with the above three properties. So, add a new class file with the name EmployeeBasicInfo.cs .

```
//Query Syntax
IEnumerable<EmployeeBasicInfo> selectQuery = (from emp in Employee.GetEmployees()
                                              select new EmployeeBasicInfo()
                                              {
                                                  FirstName = emp.FirstName,
                                                  LastName = emp.LastName,
                                                  Salary = emp.Salary
                                              });

//Method Syntax
List<EmployeeBasicInfo> selectMethod = Employee.GetEmployees().
                                       Select(emp => new EmployeeBasicInfo()
                                       {
                                           FirstName = emp.FirstName,
                                           LastName = emp.LastName,
                                           Salary = emp.Salary
                                       }).ToList();
```

The only difference is that instead of using Employee type, now we are using EmployeeBasicInfo type

Anonymous Type using Select Linq operator:

```
//Query Syntax
var selectQuery = (from emp in Employee.GetEmployees()
                                        select new
                                        {
                                            FirstName = emp.FirstName,
                                            LastName = emp.LastName,
                                            Salary = emp.Salary
                                        });

//Method Syntax
var selectMethod = Employee.GetEmployees().
                                    Select(emp => new
                                    {
                                        FirstName = emp.FirstName,
                                        LastName = emp.LastName,
                                        Salary = emp.Salary
                                    }).ToList();
```

Index in Linq

```
//Query Syntax
var query = (from emp in Employee.GetEmployees().Select((value, index) => new { value, index })
             select new
             {
                 IndexPosition = emp.index,
                 FullName = emp.value.FirstName + " " + emp.value.LastName,
                 Salary = emp.value.Salary
             }).ToList();

//Method Syntax
var selectMethod = Employee.GetEmployees().
                                Select((emp, index) => new
                                {
                                    IndexPosition = index,
                                    FullName = emp.FirstName + " " + emp.LastName,
                                    Salary = emp.Salary
                                });
```

SelectMany

- The SelectMany in LINQ is used to project each element of a sequence to an **IEnumerable<T>** and then flatten the resulting sequences into one sequence. That means the SelectMany operator combines the records from a sequence of results and s then converts it into one result.

Where Filtering operators in LINQ:

- Filtering is nothing but the process to get only those elements from a data source that satisfied the given condition. It is also possible to fetch the data from a data source with more than one condition as per our business requirement.

There are two methods provided by LINQ in C# which are used for filtering.

1. Where
2. OfType

Where Filtering operators in LINQ:

- The standard query operator “**where**” comes under the **Filtering Operators** category in LINQ.

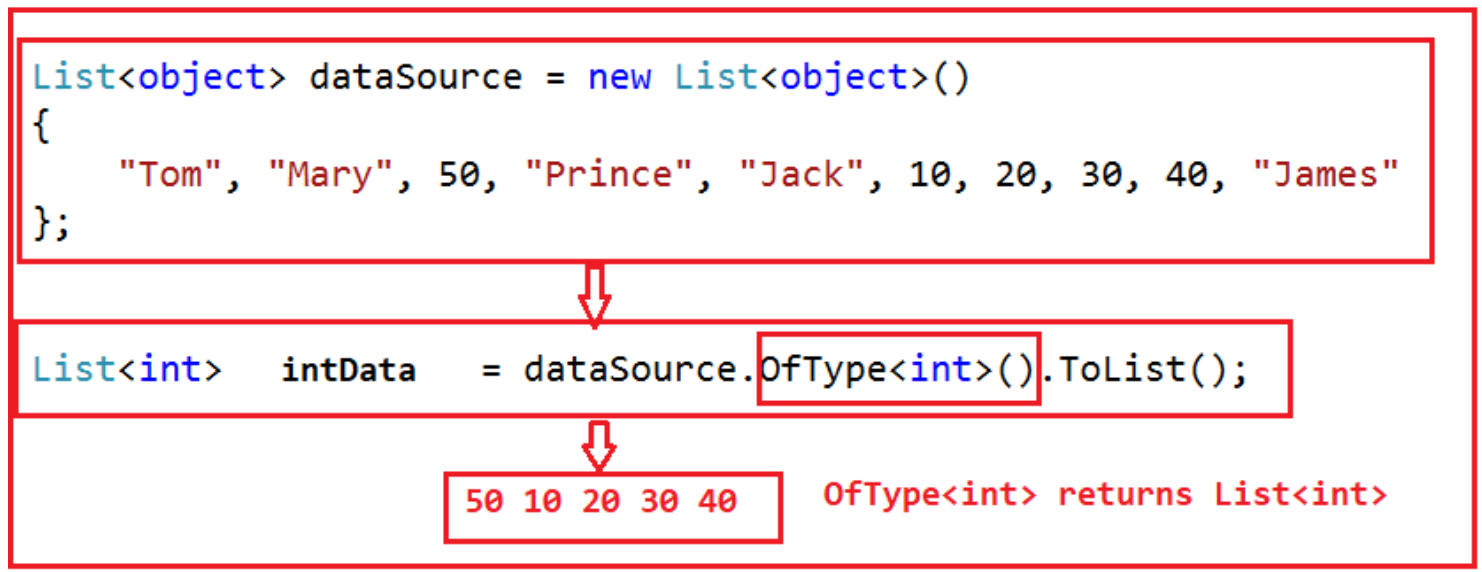
- We need to use the where standard query operator in LINQ when we need to filter the data from a data source based on some condition(s) just like as we did in SQL using the where clause. So in simple words, we can say that it is used to filter the data from a data source based on some condition(s).

- The “where” always expects at least one condition and we can specify the condition(s) using predicates. The conditions can be written using the following symbols

- **==, >=, <=, &&, ||, >, <, etc.**

OfType Operator in LINQ

- The OfType Operator in LINQ is used to filter specific type data from a data source based on the data type we passed to this operator. For example, if we have a collection that stores both integer and string values and if we need to fetch either only the integer values or only the string values from that collection then we need to use the oftype operator.


List<object> dataSource = new List<object>()
{
"Tom", "Mary", 50, "Prince", "Jack", 10, 20, 30, 40, "James"
};
List<int> intData = dataSource.OfType<int>().ToList();
50 10 20 30 40
OfType<int> returns List<int>

Set Operators in LINQ using C#

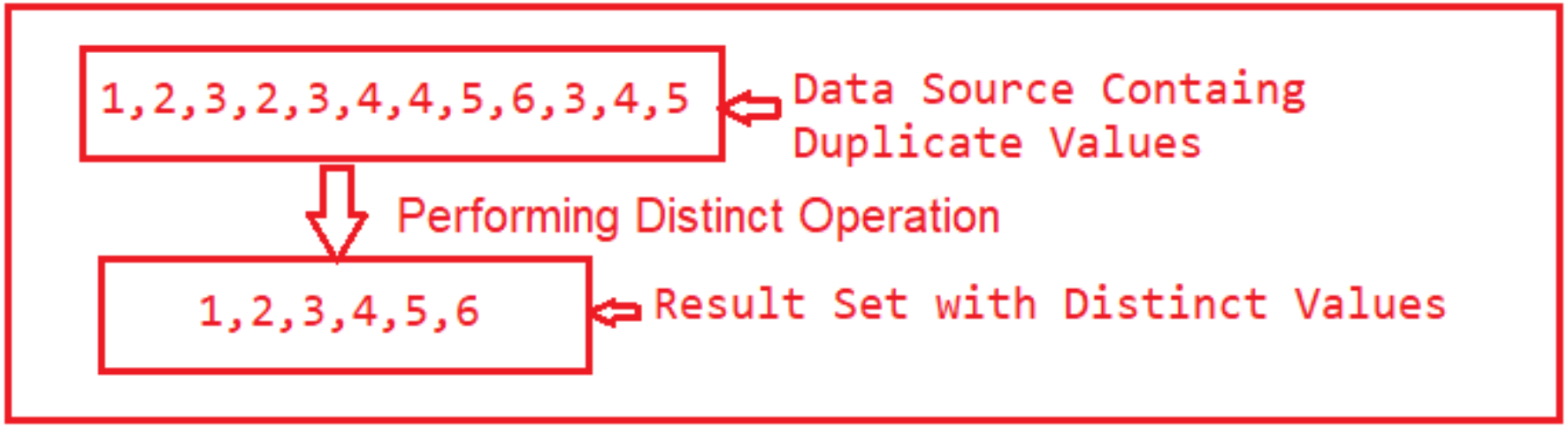
- Set operators in LINQ:
- The set operators in LINQ are used to produce the result set based on the presence and absence of elements within the same or different data sources.
- That means these operations are performed either on a single data source or on multiple data sources and in the output some of the data are present and some of the data are absent.

LINQ Set operation Methods in C#:

- Distinct: we need to use the Distinct() method when we want to remove the duplicate data or records from a data source. This method operates on a sing data source.
- Except : We need to use the Except() LINQ method when we want to return all the elements from the first data source which do not exists in the second data source. This method operates on two data sources.
- Intersect : This method is used to return the common elements from both the data sources i.e. the elements which exist in both the data set are going to return as output.

- Union : This method is used to return all the elements which are present in either of the data sources. That means it combines the data from both the data sources and produce a single result set.

LINQ Distinct Method



Difference between IEqualityComparer<T> &IEquatable<T>

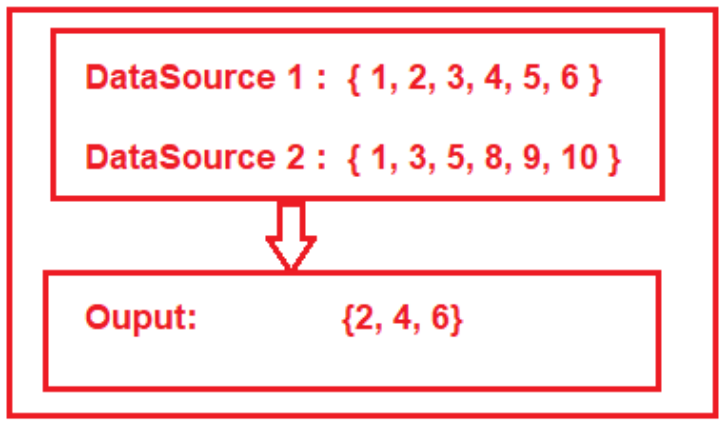
- The IEqualityComparer<T> is an interface for an object that performs the comparison on two objects of the type T whereas the IEquatable<T> is also an interface for an object of type T so that it can compare itself to another.

LINQ Except in C#

- What is LINQ Except:
- The LINQ Except method in c# is used to return the elements which are present in the first data source but not in the second data source.
- There are two overloaded versions available for the LINQ Except Method.

```
public static IEnumerable<TSource> Except<TSource>(this IEnumerable<TSource> first,
IEnumerable<TSource> second);

public static IEnumerable<TSource> Except<TSource>(this IEnumerable<TSource> first,
IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
```



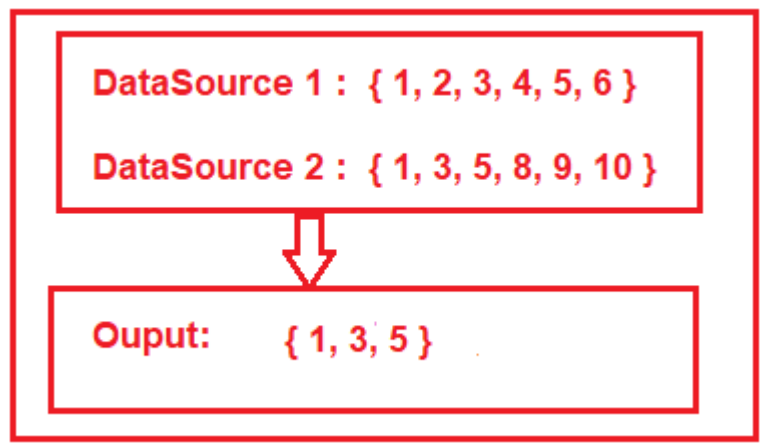
The one and the only difference between the above two methods is the second overloaded version takes IEqualityComparer as an argument. That means the Except Method can also be used with comparer also.

LINQ Intersect Method in C#:

- The LINQ Intersect Method in c# is used to return the common elements from both the collections. The elements that are present in both the data sources. There are two overloaded versions available for LINQ Intersect Methods.

```
public static IEnumerable<TSource> Intersect<TSource>(this IEnumerable<TSource> first,
IEnumerable<TSource> second);

public static IEnumerable<TSource> Intersect<TSource>(this IEnumerable<TSource> first,
IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
```



LINQ Union

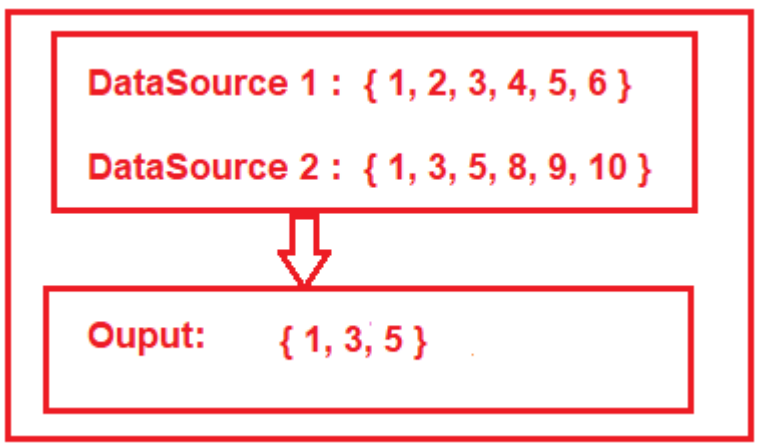
- The LINQ Union Method in c# is used to combine the multiple data sources into one data source by removing the duplicate elements.
- There are two overloaded versions available for the LINQ Union Methods.

.NET Core

- .NET Core is the open-source, free development platform maintained by Microsoft. It is a cross-platform framework that runs on windows, macOS & Linux operating system.
- .NET Core Framework can be used to build different types of applications such as mobile, desktop, web, game etc.

```
public static IEnumerable<TSource> Intersect<TSource>(this IEnumerable<TSource> first,
IEnumerable<TSource> second);

public static IEnumerable<TSource> Intersect<TSource>(this IEnumerable<TSource> first,
IEnumerable<TSource> second, IEqualityComparer<TSource> comparer);
```



LINQ Concat Method in c#

- The Linq Concat Method in c# is used to concatenate two sequences into one sequence. There is only one version available for this method whose signature is given below.

```
public static IEnumerable<TSource> Concat<TSource>(this IEnumerable<TSource> first,
IEnumerable<TSource> second);
```

What is the difference between Concat and union operators in Linq ?

- The Concat operator is used to concatenate two sequences into one sequence without removing the duplicate elements. That means it simply returns the elements from the first sequence followed by the elements from the second sequence.
- On the other hand, Linq union operator is also used to concatenate two sequences into one sequence by removing the duplicate elements.
- While working with the concat operator if any of the sequence is null then it will throw an exception.

Ordering Operators in LINQ :

- In simple terms, we can say that ordering is nothing but a process to manage the data in a particular order.
- It is not changing the data or output rather this operation arranges the data in a particular order i. e. either ascending order or descending order.

What are the Methods available in Linq for sorting the data:

- There are five methods provided by LINQ to sort the data. They are as follows

1. OrderBY
2. OrderByDescending
3. ThenBy
4. ThenByDescending
5. Reverse

What is Linq OrderBy Method

- The Linq OrderBy method in C# is used to sort the data in ascending order. The most important point that you need to keep in mind is this method is not going to change the data rather it is just changing the order of the data.
- You can use the OrderBY method on any data type i.e. you can use character, string, decimal, integer, etc. Let us understand the use of the LINQ OrderBy method in c# using both query syntax and method syntax.

- In query syntax, while we are sorting the data in ascending order then the use of ascending operator is optional. That means if we are not specifying anything then by default it is ascending. So the following two statements are the same.

```
var QS1 = (from name in stringList
           orderby name ascending
            select name).ToList();
```

Use of ascending operator is optional. Default is ascending.

```
var QS1 = (from name in stringList
           orderby name
            select name).ToList();
```

Both the above query will give us the same output.

- The most important point that you need to remember is, you need to use the Where method before the OrderBy method in both query syntax and method syntax.

LINQ OrderByDescending Method

- The LINQ OrderByDescending method in c# is used to sort the data in descending order. The point that you need to remember is , the OrderByDescending method is not going to change the data, it is just changing the order of the data.
- Like the OrderBY method, you can also use the OrderBYDescending method on any data type such as string, character, float, integers etc.

Why we need the LINQ ThenBy and ThenByDescending Method in c#

- The LINQ OrderBy or OrderByDescending method works fine when you want to sort the data based on a single value or a single expression.
- But if you want to sort the data based on multiple values or multiple expressions then you need to use the LINQ ThenBy and ThenByDescending Method along with OrderBy or OrderByDescending Method.

What are the LINQ ThenBy and ThenByDescending Method in c#:

- The Linq ThenBy Method in c# is used to sort the data in ascending order from the second level onwards. On the other hand, the Linq ThenBYDescending Method in c# is used to sort the data in descending order also from the second level onwards.
- These two methods are used along with OrderBy or OrderByDescending method . You can use the ThenBy or ThenByDescending method more than once in the same LINQ query.
- The OrderBy or OrderBy method is generally used for primary sorting.

ThenBy or ThenByDescending are used for secondary sorting and so on.

What is Reverse Method in C#:

- The LINQ Reverse method is used to reverse the data stored in a data source. That means this method will not change the data rather it simple reverse the data stored in the data source. So, as a result, we will get the output in reverse order.
- Reverse Method Signature:
- The Reverse Method is implemented in two different namespaces such as System.Linq and System.Collections.Generic namespaces.

```
namespace System.Linq

public static IEnumerable<TSource> Reverse<TSource>(this IEnumerable<TSource> source);
```

```
namespace System.Collections.Generic

public void Reverse();
```

- As you can see the Reverse method which belongs to System.Linq namespace implemented as an extension method on Ienumberable<TSource> interface and more importantly this method also returns an IEnumberable<Tsource> type.
- On the other hand, the Reverse method which belongs to the System.Collections.Generic namespace is not returning any value as the return type is void.

LINQ Aggregate Functions in c#

- The Linq aggregate functions are used to group together the values for multiple rows as the input and then return the output as a single value. So, Simple word, we can say that the aggregate function in C# is always going to return a single value.
- When to use the Aggregate Functions in C#

Whenever you want to perform some mathematical operations such as Sum , Count, Max, Min, Average, and Aggregate on the numeric property of a collection then you need to use the Linq Aggregate Functions.

what are the Aggregate methods provided by Linq

- Sum() : This method is used to calculate the total value of the collection.
- Max(): This method is used to find the largest value in the collection.
- Min(): This method is used to find the smallest value in the collection.
- Average(): This method is used to calculate the average value of the numeric type of the collection.
- Count(): This method is used to count the number of elements present in the collection.
- Aggregate(): This method is used to performs a custom aggregation operation on the values of a collection.

Quantifier Operations

- We need to use the LINQ Quantifier operators on a data source when we want to check if some or all of the elements of that data source satisfy a condition or not.
- That means, here we have a data source and also we have a condition. Then we need to check whether all or some of the elements of that data source satisfied the condition or not.
- All the methods in quantifier operations are always going to return a Boolean value. That means if the all or some of the elements in the data source satisfy the given condition then it is going to return true else it is going to return false.
- The condition that we specify may be for some or all of the elements.

What methods are available in this category?

- All : This specifies whether all the elements of a data source satisfy a given condition or not.
- Any : This specifies whether at least one of the elements of a data source satisfies the condition or not.
- Contains : This method is used to check whether the data source contains a specified element or not.
- All of the above three methods return Boolean true or false depending on whether all or some of the elements in a data source satisfy a condition.

LINQ GroupBY in C#

- The Linq GroupBy in c# belongs to the Grouping Operators category and exactly does the same thing as the Group By clause does in SQL Query. This method takes a flat sequence of elements and then organizes the elements into group(i.e Igrouping<Tkey, Tsource>)based on a given key.
- If you go the definition of GroupBy method then you will see that it return an Ienumerable<Igrouping<Tkey, Tsource>> where Tkey is nothing but the key value on which the grouping has been formed and Tsource is the collection of elements that matches the grouping key value.

Why do we need to group the data based on Multiple keys

- In real-time applications, we need to group the data based on multiple keys.
- How to group the data based on multiple keys.
- But before that, you just need to remember one thing that when you are using multiple keys in Group By operator then the data returned is an anonymous type.

What is Linq ToLookup operator

- The Linq ToLookup Method in c# exactly does the same thing as the GroupBy operator does in Linq.
- The only difference between these two methods is the GroupBy method uses deferred execution whereas the execution of the ToLookup method is immediate.

Linq Joins in c#

What Linq join operations:

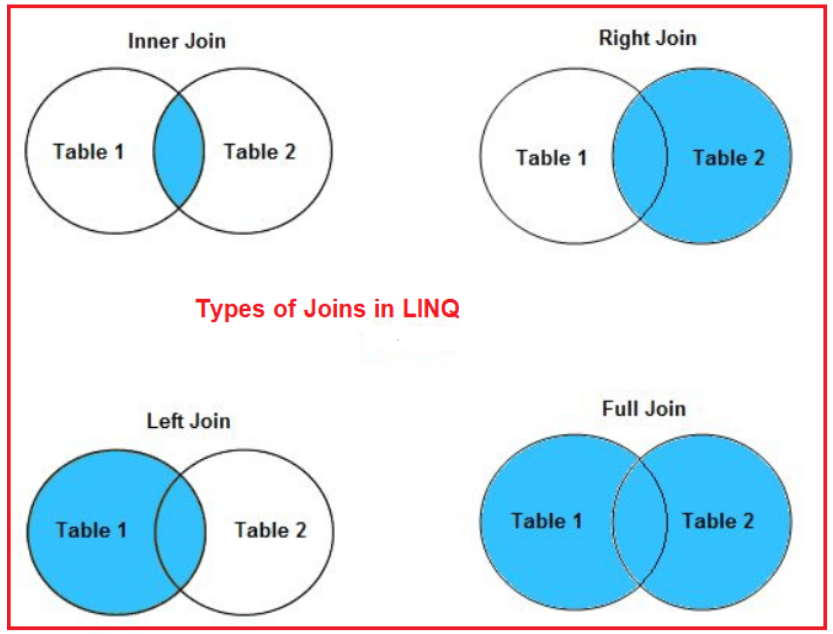
- A join of two data sources is the association of objects in one data source with objects that share a common attribute in another data source.
- We can simplify the above definition as “ join operations are used to fetch the data from two or more data sources based on some common properties present in the data sources”.

What are the methods available in Linq to perform the join operations

- There are two methods available in Linq to perform Join operations. They are
- Join : This operator is used to join two data sources or collection based on common property and return the data as a single result set.
- GroupJoin : This operator is also used to join two data sources or collections based on a common key or property but return the result as a group of sequences.

What are the different types of Linq joins available in c#.

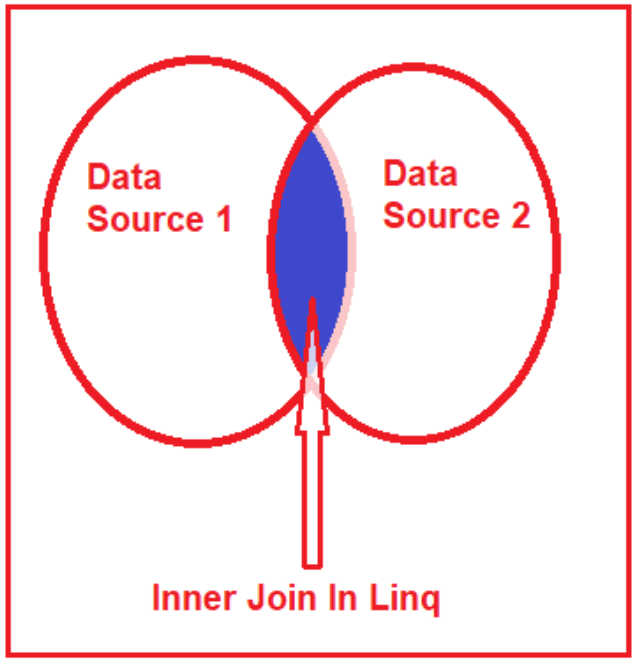
- We can perform different types of joins such as Inner join, Left join, Right Join, Full join and Cross join in Linq.



What is Linq Inner Join in c#

- An inner join produces a result set in which each element of the first collection appears one time for every matching element in the second collection. If an element in the first collection does not have any matching element in the second collection, then it does not appear in the result set.
- In simple words, we can say that the Linq inner join is used to return only the matching elements from both the data sources while the non-matching elements are removed from the result set.
- So, if you have two data sources, and when you perform the LINQ inner join, then only the matching elements which exist in both the data sources are included in the result set.

- While performing the Linq inner join then there should exist a common element or property in both the data sources.



What is Linq Join Method in c#

- The Linq join method in c# operates on two data sources or you can say two collections such as inner collection and outer collection. This operator returns a new collection which contains the data from both collections and it is the same as the SQL join operator.

Group Join in C#

- In Linq, we can apply the Group Join on two or more data sources based on a common key (the key must exist in both the data sources) and then it produces the result set in the form of groups.
- In simple words, we can say that Linq Group Join is use to group the result sets based on a common key.
- So, the Group join is basically used to produces hierarchical data structures. Each item from the first data source is paired with a set of correlated items from the second data source.

Method syntax

```
Department.GetAllDepartments(). //Outer Data Source
GroupJoin(
    Employee.GetAllEmployees(), //Inner Data Source
    dept => dept.ID, //Outer Key Selector
    emp => emp.DepartmentId, //Inner key Selector
    (dept, emp) => new {dept, emp}  //Result Selector
);
```

Query Syntax

```
from dept in Department.GetAllDepartments()
join emp in Employee.GetAllEmployees()
on dept.ID equals emp.DepartmentId
```

Inner Join

```
into EmployeeGroups
```

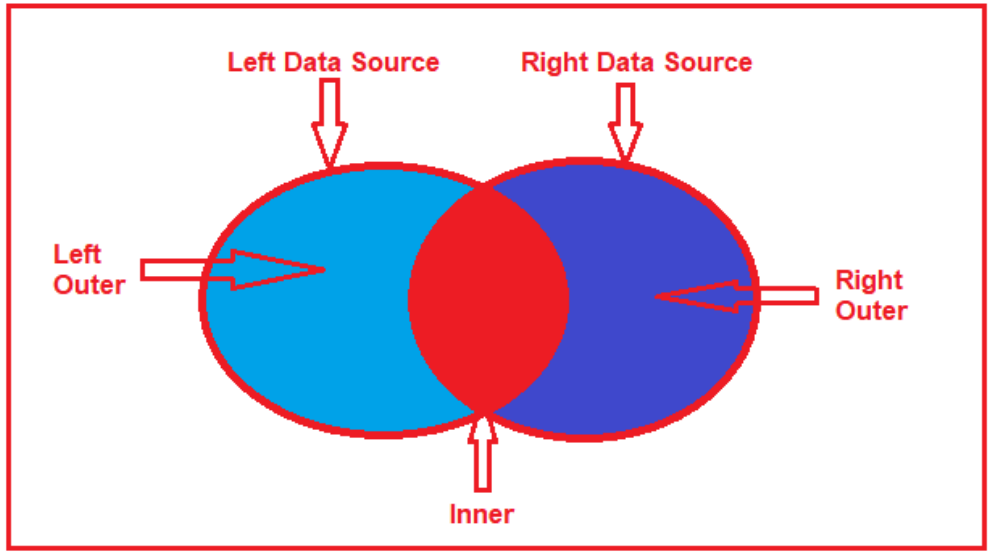
Grouping the Result set

```
select new { dept, EmployeeGroups };
```

Result selector

Left Join in Linq

- The left join or left outer join is a join in which each data from the first data source is going to be returned irrespective of whether it has any correlated data present in the second data source or not.



- So, in simple words, we can say that the Left Outer join is going to return all the matching data from both the data sources as well as all the non-matching data from the left data source. In such cases, for the non-matching data, it will take null values for the second data source.
- In order to implement the linq Left Join in c#l, it’s mandatory to use the “INTO” keyword along with the “DefaultEmpty()” method.

Cross Join

- When combining two data sources(or you can two collections) using Linq Cross Join, then each element in the first data source will be mapped with each and every element in the second data source.
- So, in simple words, we can sya that the Cross join produces the cartesian products of the collections or data sources involved in the join.
- In cross join we don’t require the common key or property as the “on” keyword which is used to specify the Join key is not required. And moreover, there is no filtering of data.

Element Operators in LINQ

- The Element Operators in Linq are used to return a single element from a data source using the element index or based on a predicate i.e. a condition.
- These Element Operators can be used with a single data source or on a query of multiple data sources.

What methods are available in the Element Operators category

- The folloDefwing methods are provided by Linq to perform element operations.

1. ElementAt and ElementAtOrDefault
2. First and FirstOrDefault
3. Last and LastOrDefault
4. Single and SingleOrDefault
5. DefaultEmpty.

ElementAt and ElementAtOrDefault

- ElementAt : The ElementAt operator is used to return an element from a specific index. If the data source is empty or if the provide index value is out of range, then we will get ArgumentOutOfRangeException.

```
public static TSource ElementAt<TSource>(this IEnumerable<TSource> source,
int index);
```

- If the data source is empty or if you specify a negative value for the index position or if you specify the index position which is out of range then you will get a runtime Exception.
- If you don’t want that exception then you need to use the ElementAtOrDefault method.
- ElementAtOrDefault method: The ElementAtOrDefault method does the same thing as the ElementAt method except that this method does not thrown an exception when the data source is empty or when the supplied index value is out of range. In such cases, it will return the default value based on the data type of the element the data source contain.

First and FirstOrDefault Methods in LInq

- Both First and FirstOrDefault methods in Linq are used to return the first element from a data source.
- But if the data source is empty or if the specified condition does not return any data, then First method will throw an exception while FirstOrDefault method will not throw an exception instead it returns a default value based on the data type of the element.

Last and LastOrDefault methods in Llnq

- Both Last and LastOrDefault methods in Linq are used to return the last element from a data source.
- But If the data source is empty or if no element is satisfied with the specified with the specified condition, then the Last method will throw an exception while the LastOrDefault method will not throw an exception instead it returns a default value based on the data type of the element.

Single and SingleOrDefault methods

- Both Single and SingleOrDefault methods in Linq are used to returns a single element from a sequence.
- But if the sequence is empty or if no element is satisfied with the given condition, then the Signle method will throw an exception while SingleOrDefault method will not throw an exception instead it returns a default value.
- The most important point that you need to keep in mind is, like the Single method, the SingleOrDefault method still throws an exception when the sequence contains more than one matching element for the given condition.

Linq DefaultIfEmpty method:

- If the sequence or data source on which the DefaultIfEmpty method is called is not empty, then the values of the original sequence or data source are going to be returned.
- On the other hand, if the sequence or data source is empty, then it returns a sequence with the default values based on the data type.

SequenceEqual Operator in LINQ

- The SequenceEqual operator in Linq is used to check whether two sequences are equal or not. If two sequences are equal then it returns true else it returns false.
- Two sequences are considered to be equal when both the sequences have the same number of elements as well as the same values should be present in the same order.
- Both the sequence contains the same data but here we are getting the output as False. This is because when comparing the complex types, the default comparer which is used by the SequenceEqual method will only check if the object references are equal or not.

How to solve the above problem?

- There are many ways we can solve the above problem as follows.

1. We need to use the other overloaded version of the SequenceEqual method to which we can pass a custom class that implements the IEqualityComparer interface.
2. Project the properties into a new anonymous type, which overrides Equals() and GetHashCode() methods.
3. In the Student class override the Equals() and GetHashCode() methods.

What are partitioning operations in Linq

- The partitioning operations in Linq are used to divide a sequence or you can say data source into two parts and then return one of them as output without changing the positions of the elements.
- Partitioning Methods provided by Linq:
- Take
- Skip
- TakeWhile
- SkipWhile

Take operator

- The Take operator in Linq is used to fetch the first “n” number of elements from the data source where “n” is an integer which is passed as a parameter to the Take method.
- The Take method takes one integer count parameter and this method going to return that number of contiguous elements from the data source. If the data source is null then it is going to throw an exception.
- You can use the Take method with both the Method and Query syntax but most important point that you need to remember is it will not make any changes to the positions of the elements.

TakeWhile Method

- The TakeWhile method in linq is used to fetch all the elements from a data source or sequence until a specified condition is true. Once the condition is failed, then the TakeWhile method will not check the rest of the elements presents in the data source even though the condition is true for the remaining elements.
- The TakeWhile method will not make any changes to the positions of the elements.

Difference between TakeWhile and Where

- The difference is that the TakeWhile method checks the conditions from the beginning of the data source. As long as the condition is true it fetches the data.
- On the other hand, the Where method checks each and every element present in the collection. The elements which satisfy the condition will be returned. It does not matter the position of the elements in the sequence.

Skip Method

- The Skip Method in Linq is used to skip or bypass the first “n” number of elements from a data source or sequence and then returns the remaining elements from the data source as output.
- Here “n” is an integer value passed to the Skip method as a parameter.
- If the Data source is null then it is going to throw an exception.

SkipWhile Method

- The SkipWhile Method in Linq is used to skip or bypass all the elements from a data source or sequence as long as the given the condition specified by the predicate is true and then returns the remaining element from the sequence as an output.
- The most important point that you need to remember is, once the condition is failed then the SkipWhile method will not check the resto of the elements even though the condition is true for some of the remaining elements.

Why we need paging in real-time application

- Suppose we have a data source with lots of records and we need to display those records in a view. We can display all the records in a view at once. If we do so then we will get the following disadvantages:
- Network Issue(as huge data is travelled)
- Memory Management
- Performance

What is paging

- Paging is nothing but a process in which we will divide a large data source into multiple pages. At one page we need to display a certain number of recodes. And next records can be visible with next- previous buttons or scroll or using any other techniques.
- Advantages:
- Faster data transfer.
- Improve memory management.
- Better performance

Drawback: In a client-server architecture, the number of requests between the client and server is increased. In such cases, you may get the data at once and store it locally and then implement the paging at the client side.

How to implement paging?

- We can implement the paging using the Linq Skip and Take method. Here we need to understand two things one is PageNumber and Other one is the number of records per page.

what is Generation operator

- The Enumberable class also provides three non-extension methods are as follows.

1. Range()
2. Repeat<T>()
3. Empty<T>()

These methods allow us to create some specific type of array, sequence, or collection using a single expression instead of creating them manually and populating them using some kind of loops.

That means these methods return a new sequence or collection that implements the Ienumberable<T> interface.

Methods

- Range : This method is used to generates a sequence of integral numbers within a specified range.
- Repeat : The LINQ Repeat Method is used to generate a sequence or collection that with a specified number of elements and each element contains the same value.
- The LINQ Repeat method is implemented using the deferred execution. So, the immediate return value is an object which stores all the required information to perform an action. The query represented by this method is not executed until the object is enumerated either by calling its GetEnumerator method directly or by using a foreach loop.

- Empty: The Empty method belongs to the Generation operator category. It is a static method included in the static Enumberable class. The LINQ Empty Method is used to return an empty collection(i.e. IEnumberable<T>) of a specified type.
- Append : The LINQ Append method is used to append a value to the end of the sequence. This Append method does not modify the elements of the sequence. Instead, it creates a copy of the sequence with the new element.
- Prepend : The Linq Prepend method is used to add one value to the beginning of a sequence. This Prepend method like the Append method does not modify the elements of the sequence. Instead, it creates a copy of the sequence with new element.

- Zip : The LINQ Zip Method in c# is used to apply a specified function to the corresponding elements of two sequences and producing a sequence of the results.
- The Zip method merges each element of the first sequence with an element in the second sequence that has the same index position. If both the sequences do not have the same number of elements, then the zip method merges sequences until it reaches the end of the sequence which contains fewer elements.

Deferred & Immediate Execution in LINQ

- The LINQ queries are executed in two different ways as follows:

1. Deferred Execution
2. Immediate Execution

Based on the above two types of execution, the LINQ operators are divided into 2 categories. They are as follows:

Deferred or Lazy operators: These query operators are used for deferred execution. For example: select, SelectMany, where, Take, Skip etc. are belongs to Deferred or Lazy operators category.

* Immediate or Greedy operators: These query operators are used for immediate execution. For example: count, average, min,max, First, Last, ToArray, ToList etc. are belongs to the immediate or Greedy Operators category.

.NET Core Characteristics

- Open-source Framework: .NET Core is an open-source framework maintained by Microsoft.
- Cross-platform: .NET Core runs on windows, macOS and Linux operating system. There are different runtime for each operating system that executes the code and generates the same output.
- Wide-range of applications: Various types of applications can be developed and run on .NET Core platform such as mobile, web, game etc..
- Supports Multiple languages: You can use C#, F#, and visual Basic programming languages to develop .NET Core applications.

.NET Core

- .NET Core is an open source.
- It is compatible with various operating system like windows, Linux and Mac OS. It is cross-platform.
- .NET Core is packaged and installed independently of the underlying operating system as it is cross-platform.
- .NET Core is shipped as a collection of NuGet packages.

.NET Framework

- Certain components of the .NET framework are open source.
- It is compatible with only windows.
- .NET Framework is installed as a single package for windows operating system.
- All the libraries of .NET Framework are packaged and shipped together.