**.NET**

1. **What is .NET?**

.NET is a framework tool that supports many programming languages and many technologies. It supports 60+ programming languages. In 60+ programming languages, 9 are designed by Microsoft and the remaining are designed by non-Microsoft.

1. **What .NET represents?**

* NET stands for Network Enabled Technology
* In .NET dot (.) refers to object-oriented and NET refers to the internet.

So the complete .NET means through object-oriented we can implement internet applications.

1. **What is a Framework?**

A framework is a software. Or we can say that a framework is a collection of many small technologies integrated together to develop applications that can be executed anywhere.

1. **Which technologies are supported by the .NET framework?**

* ASP.NET (Active Server Pages.NET)
* ADO.NET (Active Data Object.NET)
* WCF (Windows Communication Foundation)
* WPF (Windows Presentation Foundation)
* WWF (Windows Workflow Foundation)
* AJAX (Asynchronous JavaScript and XML)
* LINQ (Language Integrated Query)
* ASP.NET MVC (Model View Controller)
* ASP.NET WEB API

1. **What does the .NET ecosystem provide?**

The .NET ecosystem is a comprehensive platform developed by Microsoft for building and running various types of applications, ranging from web applications and services to desktop applications, mobile apps, cloud-based solutions, and more. The .NET ecosystem provides a rich set of tools, frameworks, libraries, and services to support application development across different domains.

1. **What are the components of .NET?**

Key components and offering within the .NET ecosystem:

* .NET Framework – building windows app, web app and services.
* .NET Core – cross platform, framework for building cloud, cross platform app.
* ASP.NET – development for dynamic web app, web APIs and services
* ASP.NET Core – building modern, cross platform, and cloud based web app.
* Entity Framework – ORM for simplifying database interaction in .NET app.
* Xamarin – for mobile app using .NET languages like c#.
* Blazor – building web app like C# and .NET instead of JS. Used for Client & Server dev.
* Azure Devops
* Visual Studio
* NuGet - manage dependencies

1. **How does the .NET Framework work?**

* .NET framework-based applications that are written in supportive languages like C#, F#, or VB are compiled to Common Intermediate Language (CIL).
* Compiled code is stored in the form of an assembly file that has a .dll or .exe file extension.
* When the .NET application runs, Common Language Runtime (CLR) takes the assembly file and converts the CIL into machine code with the help of the JIT (Just in Time) compiler.
* Now, this machine code can execute on the specific architecture of the computer it is running on.

A diagram of a language code

Description automatically generated

1. **What are the major components of the .NET Framework?**

* Common Language Runtime
* Framework Class Library
* Base class library

User defined class library (Assembly)

Pre defined class library (Namespace)

* Common Type System
* Common Language Specification

1. **Discuss various major components of the .NET?**

COMMON LANGUAGE RUNTIME

* It is an execution engine that runs the code and provides services that makes the development process easier.
* Services provided by CLR are memory management, garbage collection, type safety, exceptional handling, security and thread management. It also makes it easier for designing the app and components whose objects interact across the language.
* The programs written for the .NET framework are executed by the CLR regardless of programming language.

A blue rectangular sign with white text

Description automatically generated

FRAMEWORK CLASS LIBRARY

* It has pre – defined method and properties to implement common and complex functions that can be used by .NET applications.
* It will also provide types for dates, string, numbers, etc.
* This class includes APIs for database connections, file reading, and writing, drawing etc.

BASE CLASS LIBRARY

* It is a huge collection of libraries features and functions that are helpful in implementing various programming languages such as C#, F#.
* BCL is divided in 2 parts
  + User Defined class library : It includes Assemblies. Assembly is a logical unit of code, which is used for security, deployment, and versioning. Assembly can be defined in two forms namely dll or exe.
  + Predefined class library: it contains namespace. Namespace is the collections of pre-defined method and classes that are present in the .NET framework. A namespace can be added to a .NET program with the help of “using System”, where using represents a keyword and system represents a namespace.

COMMON TYPE SYSTEM

* CTS stands for Common Type System. It follows a set of structured rules according to which a data type should be declared and used in the program code. It is used to describe all the data types that are going to be used in the application.
* We can create our own classes and functions by following the rules in the CTS. It helps in calling the data type declared in one programming language by other programming languages.

COMMON LANGUAGE SPECIFICATION

* It is a subset of CTS and defines a set of rules and regulations to be followed by every .NET framework language.
* A CLS supports inter-operability or cross language integration, which means it provides a common platform for interacting and sharing information. For ex, every programming language (C#, F#, VB, VB.NET etc.) under the .NET framework has its own syntax. So, when statements belonging to different languages get executed, a common platform, will be provided by the CLS to interact and share the information.

1. **What are the different languages that are supported with .NET?**

Applications are written with .NET in several languages such as-

* C# which is a simple and modern language. This programming language is a type-safe language and is object oriented thus used with .NET easily.
* F# which helps to write code in a compact and easy way.
* Visual Basic is an approachable language and is used with .NET to build type-safe applications.

1. **What is JIT?**

* JIT stands for Just-in-time.
* JIT is the component of CLR that is responsible for converting MSIL code into Native code or Machine code.
* This Native code or Machine code is directly understandable by the operating system.

1. **What is Microsoft Intermediate Language?**

MSIL is the Microsoft Intermediate Language, which provides instructions for calling methods, memory handling, storing, and initializing values, exception handling and so on.

The instructions provided by MSIL are platform independent and are generated by the language specific compiler from the source code. JIT.

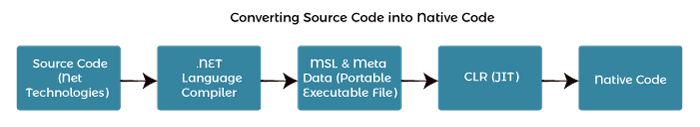
1. **What is CLR?**

CLR is the core component under the .NET framework which is responsible for converting MSIL code into native code and then execution.

That language's compiler compiles the source code of applications developed using .NET compliant languages into CLR's intermediate language called MSIL, i.e., Microsoft intermediate language code. This code is platform-independent. It is comparable to byte code in java.

Metadata is also generated during compilation and MSIL code and stored in a file known as the Manifest file. This metadata is generally about members and types required by CLR to execute MSIL code.

A just-in-time compiler component of CLR converts MSIL code into native code of the machine. This code is platform-dependent. CLR manages memory, threads, exceptions, code execution, code safety, verification, and compilation.



1. **What is the execution process of CLR?**

In .NET, the code is compiled twice.

In 1st compilation source code (High-Level Code) is compiled by the respective language compiler and the language compiler generates intermediate code which is also known as MSIL (Microsoft Intermediate Language) or IL (Intermediate language code) Or Managed code.

In the 2nd compilation, MSIL is converted into Native Code (Machine code) using CLR.

Always 1st compilation is slow and 2nd compilation is fast.

1. **What is metadata?**

Metadata describes every type and member defined in our code in a Multilanguage form. Metadata stores the following information.

* Description of assembly.
* Identity (name, version, culture, public key).
* The types that are exported
* Other assemblies that this assembly depends on.
* Security permissions are needed to run.

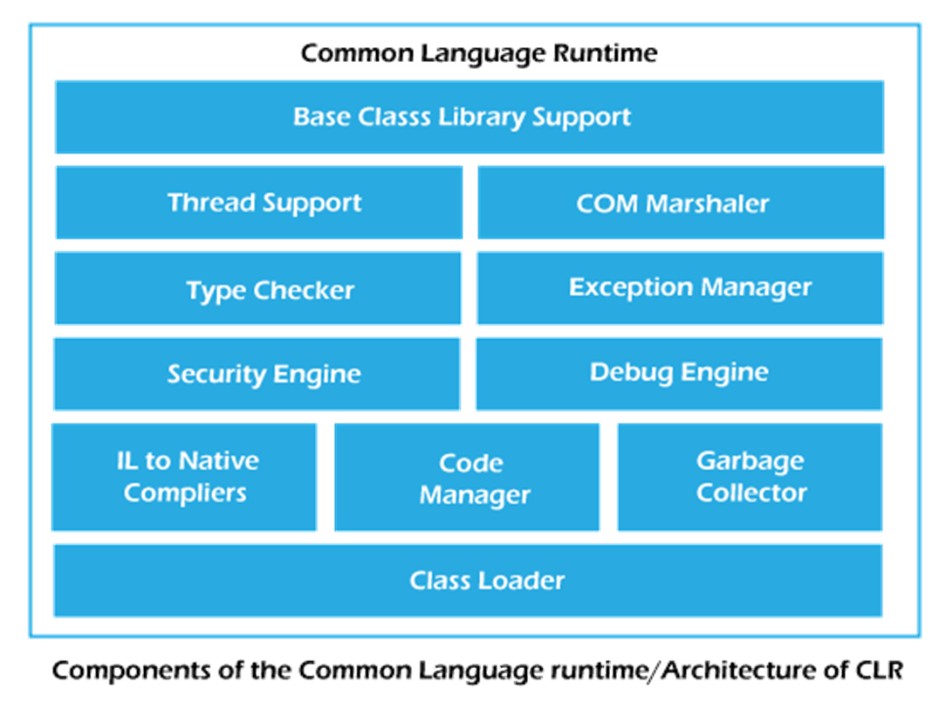
1. **What is the components of CLR?**

* Common Type System
* Common Langugae Specification
* Garbage Collector
* Jut in Time Compiler
* Metadata and Assemblies

A diagram of a computer process

Description automatically generated

1. What is the .NET CLR Structure?



BASE CLASS LIBRARY SUPPORT

It is a class library that supports classes for the .NET application.

THREAD SUPPORT

The CLR provides thread support for managing the parallel execution of multiple threads.

The System.Threading class is used as the base class for this.

COM Marshaler

* COM Marshaler refers to the component responsible for facilitating communication between managed code (code written in languages like C#) and unmanaged code (code written in languages like C++ or COM components). COM stands for Component Object Model, a binary interface standard for software components.
* When managed code needs to interact with unmanaged code or COM components, the COM Marshaler plays a crucial role in managing the transition between the two environments.
* Unmanaged code - requires only one complication code

SECURITY ENGINE

* The security engine in the CLR handles the security permissions at various levels such as the code level, folder level, and machine level. This is done using the various tools that are provided in the .NET framework.

TYPE CHECKER

* Type safety is provided by the type checker by using the Common Type System (CTS) and the Common Language Specification (CLS) that are provided in the CLR to verify the types that are used in an application.

CODE MANAGER

* It manages code at execution runtime.
* The code manager in CLR manages the code developed in the .NET framework i.e. the managed code. The managed code is converted to intermediate language by a language-specific compiler and then the intermediate language is converted into the machine code by the Just-In-Time (JIT) compiler.

GARBAGE COLLECTOR

* It releases the unused memory and allocates it to a new applications.
* Automatic memory management is made possible using the garbage collector in CLR. The garbage collector automatically releases the memory space after it is no longer required so that it can be reallocated.

EXCCEPTION HANDLER

* It handles the exception to runtime to avoid application failure.
* The exception manager in the CLR handles the exceptions regardless of the *.NET Language* that created them. For a particular application, the catch block of the exceptions are executed in case they occur and if there is no catch block then the application is terminated.

CLASS LOADER

* It is used to load all classes at runtime.
* Various modules, resources, assemblies, etc. are loaded by the CLR loader. Also, this loader loads the modules on demand if they are actually required so that the program initialization time is faster and the resources consumed are lesser.

1. **Difference between Managed and Unmanaged Code?**

Managed code/methods:

Machine instructions are in MSIL format and located in assemblies that will be executed by the CLR and will have the following advantages

* Memory management to prevent memory leaks in program code.
* Thread execution
* Code safety verification
* Compilation.

Unmanaged code/ methods:

The Unmanaged codes are the instructions that are targeted for specific platforms. Unmanaged code will exist in any of the formats.

* COM/COM+ components
* Win 32 Dlls/system DLLs
* As these codes are in native formats of OS, these instructions will be executed faster compared with JIT compilation and execution of managed code.

1. **Difference between .NET 7 and .NET 8?**

* Performance Improvements
* Cross Platform compatibility
* Programming Language Compatibility – With the release of .NET8, Microsoft has also upgraded the C# programming language used the Microsoft dotnet framework. Helps in writing clean code, helping to maintain structure, efficiently debug and update the app per business needs.
* Integration - In .NET 7, integration takes more time, as developers have to create custom codes for APIs and other components. However, in .NET 8, Microsoft has provided built-in support for major integrated development environments, APIs, and services.

1. **What are the .NET new features?**

.NET 8 New Features

* Better source generator
* Native AOT (Ahead of time) compilation
* Advanced AI capabilities
* Improved Blazor
* Elevated .NET MAUI
* SHA-3 Hashing Primitives
* HTTPS Proxy Support

1. What is Visual Studio?

Visual Studio is a Microsoft IDE tool that is needed to develop applications with the .NET framework. The IDE integrates 3 features:

* Editor
* Compiler
* Interpreter

ASSEMBLY

1. **What is .NET assembly?**

All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. It is a compiled unit of code, that can be executed by .NET runtime.

1. **What is the use of manifest in the .NET framework?**

Manifest stores the metadata of the assembly. It contains metadata which is required for many things as given below:

* Assembly version information.
* Scope checking of the assembly.
* Reference validation to classes.
* Security identification.

1. **Explain the different parts of an Assembly.**

The different parts of an assembly are:

Manifest – Every static or dynamic assembly holds a data collection that gives details about how the elements in the assembly relate to each other. An assembly manifest consists of complete metadata required to specify version requirements and security identity of an assembly, and also the metadata required for defining the assembly scope and resolving references to classes and resources.

The assembly manifest will be stored in either a standalone PE(Portable Executable) file that holds only assembly manifest information, or in a PE file (a .exe or .dll) with MSIL(Microsoft intermediate language) code.

Type Metadata – Metadata gives you additional information such as types, type names, method names, etc about the contents of an assembly. Metadata will be automatically generated by the Compilers from the source files and the compiler will embed this metadata within target output files like .exe, .dll, or a .netmodule(in the case of multi-module assembly).

MSIL – Microsoft Intermediate Language(MSIL) is a code that implements the types. It includes instructions to load, store, initialize, and call the methods on objects. Along with this, it also includes instructions for control flow, direct memory access, arithmetic and logical operations, exception handling, etc. This is generated by the compiler using one or more source code files. During the runtime, the JIT(Just In Time) compiler of CLR(Common Language Runtime) converts the MSIL code into native code to the Operating System.

Resources – Resources can be a list of related files such as .bmp or .jpg files. These resources are static, which means they do not change during run time. Resources are not executable items.

1. **How many ways assemblies can be classified?**

.NET assemblies can be broadly classified into 2 types:

Weak Named Assemblies

Strong named assemblies

1. **How to generate key pair?**

To generate the key pair, use strong naming tool

sn.exe -k C:\KeyFile.snk

To strong name an assembly

[assembly: AssemblyKeyFile(“KeyFile.snk”)]

1. **What should a strong name assembly should have ?**

A strong named assembly should have all of the following:

* The textual assembly name.
* The assembly version number
* The assembly should have been signed with private/public key pair.

1. **What does assemblies consist of?**

A assembly name consists of 4 parts

* Simple textual name
* Version number
* Culture information (otherwise the assembly is language neutral)
* Public Key Token

We use AssemblyVersion attributes to specify the assembly version. The default is 1.0.0.0. The version number of an assembly consists of the following 4 parts.

* Major version
* Minor version
* Build number
* Revision number

1. **What is versioning managed in assembly?**

The version information for an assembly typically consists of four values: a major version number, a minor version number, and optionally, a build number and a revision number. These version numbers play a crucial role in identifying and managing different versions of the assembly, enabling developers to track and control changes made to the assembly over time.

Major Version

This is the internal version of the product and is assigned by the application team. It should not change during the development cycle of a product release.

Minor Version

This should only change when there is a small changes to existing features. It is assigned by the application team, and it should not be changed during the development cycle of a product release.

Build Number

Typically incremented automatically as part of every build performed on the Build Server. Using the build number in conjunction with the source number allows you to identify what was built and how. This allows each build to be tracked and tested.

Revision

This is the number taken from source control to identify what was actually built. This is set to zero for the initial release of any major or minor version of the solution.

1. **What is the steps to add an assembly to GAC?**

* Make a class library project to create a dll or assembly
* Make a .NET console project and use or add reference the assembly into this project.
* Generate a key pair using the command: **sn.exe -k c:\MyStrongKeys.snk**
* Go to Properties of .NET console project and add key pair for strong named assembly: **[assembly:AssemblyKeyFile("C:\\MyStrongKeys.snk")]**
* Go to VS Command Prompt and add the assembly to GAC : **gacutil.exe -I MathFunctionDLL.dll**
* You can find the assembly added to GAC : C:\Windows\Microsoft.NET\assembly\GAC\_MSIL\ {Your assembly name}

1. **How to uninstall assembly ?**

* **gacutil -u MathFunctionDLL** (no mention the extension)

1. **Features of Assembly**

* They are self-describing. They consist of metadata which tells what are the methods, properties etc present in the assembly.
* Assembly can be loaded side-by side thus achieving side by side execution.
* Installation of an assembly is easier.
* Assemblies solve the DLL HELL problem.

1. **What are the types of Assembly?**

They are implemented as .exe or .dll.

1. **Difference between DLL and EXE ?**

|  |  |
| --- | --- |
| DLL | EXE |
| Dynamic Lin k Library with .dll extension. | Executable with .exe extension. |
| Can have many entry point. | Has only one entry point |
| Does not contain UI | Contains UI |
| It is an in-process file, that means it runs in someone else memory. | It is out process file. It can run independently. Its standalone. |

1. **What the physical location of assembly**

2 Paths:

1. C:\Windows\Assembly – For .NET 2.0 – 3.5 assemblies
2. C:\Windows\Microsoft.NET\assembly - For .NET 4.0 assemblies.
3. **Different types of Assembly?**

Private Assembly

An assembly used by a single application is called private assembly. Its present in the bin folder.

Shared Assembly

An assembly used by more than one application. Its present in GAC.

Satellite Assembly

These are resource files which are compiled to assemblies.

1. **What is DLL Hell ?**

DLL HELL is a problem which arises when a new version of application with a new set of DLL overrides the older version.The application which are using older version of DLL crashes because those have been replaced by new DLL which are not compatible with the old applicaitons.

1. **How is DLL Hell solved in .Net?**

It is solved with the help of Assemblies.Assemblies allow different versions of DLL to co-exist among themselves.This feature is called Versioning.

1. **What are the contents of an assembly?**

Manifest

IL Code

Resource Files

Metadata

1. **What is manifest?**

Manifest of an assembly contains the following-

Identity

List of file present in assembly

A list of referenced assemblies

A set of permissions request.

1. **What is metadata?**

Contains all the types and member information present in the code.

1. **What is ILDASM?**

Its a tool used to view contents of an assembly.

1. **What is GAC ?**

GAC is a central repository in a system in which assemblies are registered to share between application.

gacutil.exe is used to view and change the content of GAC in system

GAC can contain multiple versions on .net assemblies

The gautil.exe/I<assembly\_name> is used to install assembly in GAC

1. **How does the GAC work ? What is the prupose in .NET Framework? Pros and Cons?**

Its purpose is to reduce duplication of code, promote reusability, and simplify versioning.

When an application requests an assembly, the CLR checks the GAC before probing other locations. If found, it loads the appropriate version based on the application’s configuration or the latest available version.

Pros of using the GAC include:

1. Reduced disk space usage by sharing assemblies.

2. Simplified version management with side-by-side execution.

3. Enhanced security through strong-named assemblies.

Cons of using the GAC are:

1. Increased complexity in deployment due to additional steps for registering assemblies.

2. Potential conflicts when multiple versions coexist.

3. Dependency on strong-named assemblies may hinder development flexibility.

1. **How do install and uninstall assembly in GAC ?**

Two ways-

Simply darg and drop

Use GacUtil.exe (GAC Utility Tool)

Install - gacutil -i <assembly name>

Uninstall -gacutil -u <assembly name>

1. **How do you add/remove assembly from GAC?**

you can add assembly by using below syntax:

gacutil /i [assemblyName | assemblyPath]

you can remove assembly by using below syntax:

gacutil /u [assemblyName | assemblyPath]

1. **What is Satellite Assembly?**

Satellite assemblies are assemblies that is used to deploy culture and language for an application. A separate product id is assigned to each language and a satellite assembly is installed in language specific sub directory.

OOPs and PROGRAMMING FUNDAMENTALS

1. **What is OOPs?**

Object oriented programming is a methodology or paradigm, to develop program using some classes and object. It simplifies software development and life cycle using some OOPs concept.

1. **What is Programming Paradigm?**

In simple words it means coding style, or fundamental style, or approach to program a given problem statement. Different paradigm has different type of approaches and solution to the problem statement.

1. **Difference between Top Down and Bottom Up approach?**

|  |  |
| --- | --- |
| Top Down Approach | Bottom Up Approach |
| We focus on breaking up the problem into smaller parts. | We solve smaller problems and integrate it as a whole and complete the solution. |
| It contains redundant information. | It does not contain redundant information. |
| The implementation depends on the programming language and platform. | Data encapsulation and data hiding is implemented in this approach. |

1. **What are the advantages of Top Down approach?**

Clarity: Provides a clear understanding of the overall structure and goals.

Efficiency: Allows for the identification of key components early in the process.

Alignment: Aligns with high-level organizational objectives.

1. **What are the advantages of Bottom Down approach?**

Advantages of Bottom-up approach:

Detail-Oriented: Captures fine-grained details and nuances.

Adaptability: Allows for flexibility and adjustments as more details emerge.

Incremental Progress: Immediate progress on smaller components can be achieved.

1. **What is the difference between top down and bottom down approaches?**

The main difference between the top-down and bottom-up approaches is the process's starting point and focus. The top-down approach prioritizes high-level planning and decision-making, while the bottom-up approach prioritizes the execution of individual tasks and the development of detailed knowledge. Both approaches have advantages and disadvantages, and the best approach will depend on the specific context, including the nature of the problem, the resources available, the timeline, and the desired outcome.

1. **What are the types of Programming Paradigm and explain?**
2. Imperative
3. Declarative

Imperative Paradigm

Giving explicit instruction on how to achieve a task.

Advantages of Imperative

* Very simple to implement.
* It contains loops, variable, etc.

Disadvantages of Imperative

* Complex problem cannot be solved.
* Less efficient and less productive
* Parallel Programming in not possible

**Imperative Classification**

1. Structed
2. Procedural
3. Object Oriented
4. Parallel Processing

Structed Programming

* It emphasizes the use of structed control flow constructs, such as loops, conditional and subroutine, to improve the clarity, quality and maintainability of software.
* Based on the idea that program should be divided into small, manageable pieces that can be easily understood and modified.

Procedural Programming

* It focuses on the use of procedures, subroutines, to organize and structure code.
* Ability to reuse the code.
* Ex. C, C++, Java, ColdFusion

A screenshot of a computer code

Description automatically generated

Object Oriented Programming

* Code written using classes and object.
* Advantages – Data Security, Inheritance, Code Reusability, Abstraction

Parallel Processing Programming

* Parallel processing is a technique for improving the performance of programs by executing multiple tasks simultaneously.
* It involves breaking down a large task into smaller sub-tasks that can be executed in parallel, either on multiple processors or on a single processor with multiple cores.
* Used in wide range of applications, including scientific simulations, data analysis, and machine learning.

1. **Classify Declarative Classification**
2. Functional programming
3. Logic programming
4. Aspect Oriented programming
5. **What is Declarative paradigm?**

It focuses on describing what a program should do, rather than how it should do it.

1. Functional Programming

* It has its root in mathematics and is language independent.
* It is meant for some specific computation and not the data structure.

1. Logical Programming

* It focuses on using logical statements to represent and manipulate data. It is based on the idea that a program should be organized around logical statements that describe the relationships between inputs and outputs.

1. Aspect Oriented Programming

* Focus on logging, security and error handling, from the main logic of a program.

1. **What are the factors leading to the development and adoption of OOPs?**

* Complexity of software development
* Code reusability
* Data and code encapsulation
* Real world modelling
* Improved maintenance and security
* Adoption to modern software challenges

1. **What are the drawbacks of OOPs?**

* Complexity
* Performance Overhead
* Difficulty in debugging

C# Basics

1. **What is C#?**

C# is a high-level object-oriented programming language. It is used for building secure and robust applications.

1. **What is the process of code compilation in C#?**

There exist four steps in the process of code compilation:

* Compilation of Source code in managed code
* Clubbing the newly created code into assembly
* Loading the CLR (Common Language Runtime)
* Execution of assembly through CLR

1. **Can you name the types of comments in C#?**

There are two types of comments in C#:

Single line:

//contains only 1 line of code

Multiple line (/\* \*/):

/\*Line 1

Line 2

Last line\*/

1. **What are the different types of operators in C#?**

There are a total of seven operators in C#, which include:

Arithmetic Operators: These are the kind of operators that are used for performing arithmetic operations.

Relational Operators: These operators are used for comparing values. These always result in true or false (>,<,!=, etc).

Logical/Boolean Operators: These operators are also used for comparing values. These also always result in true or false (! , &&).

Bitwise Operators: These operators perform a bit-by-bit operation.

Assignment Operators: These operators help in assigning a new value to the variable.

Type Information Operators: These operators provide information about a certain type.

Miscellaneous Operators: ?: , => , & , && , \*

1. **Mention the various types of statements in C#**

Block statements – Block statements are a collection of multiple statements and are represented using {} [curly braces].

Declaration statements – Declaration statements are used to assign a value and a data type to variables.

Expression statements – Expression statements are statements that end with a semicolon (;).

Selection statements – Selection statements are also known as conditional statements and are statements that validate certain conditions and run the code according to the validation.

Iteration statements – Iteration statements are also known as loop statements that run multiple times according to the loop state.

1. **What is ternary operator in C#?**

A ternary operator is used for a conditional expression that returns a Boolean value. It is a short form of if-else.

1. **Difference Between Break and Continue Statement?**

Break: Terminates the nearest enclosing loop or switch statement.

Continue: Skips the rest of the loop’s code and proceeds to the next iteration of the loop.

1. **What is keywords in C#?**

Keywords are reserved words that have some predefined actions. They are special words that hold special meaning to the compiler.

1. **What is variables? And classify?**

It is an identity given to a memory location.

They are containers to store data values.

4 Classification:

Static variables - Access static variables using class-name.

Instance variables – access using object reference.

Method Params – access directly and only inside the method

Local variables – access directly and only inside the method.

1. **What is the difference between object, var and dynamic keyword in C#?**

Object

It is a type which is base of all the types and one of the oldest features of the language.

It means values of any types can be stored in object type variable. But type conversion

(un-boxing) is required to get original type when the value of variable is retrieved in

order to use it.

For Example:

Object o;

o = 20;

VAR

This was introduced in .net 3.5 framework. This also can have any type of values like

object but it is must to initialize the value at time of declaration of the variable. The

keyword is best suited for linq queries in C#.

For example:

Var a = 10;

Var b = "string value";

Var c = new Product(); // object of product type

Var d;//Error, as the value is not assigned at time of declaration of

variable d

d = 10;

DYNAMIC

This keyword was introduced in .net 4.0 framework. This also can be used to store any

type of value like object and var but unlike object un-boxing is not required at time of

using the variable.

For example:

Dynamic d1 = 10;

Dynamic d2;

D2=new Product();

Dynamic d3="string value";

1. **What is enums in C#?**

It is used to store a set of named constants such as season, days, month, size etc.

Why use Enums?

Code is easier to maintain, because only expected values of variables are determined.

Code is easier to read, because easily identifiable names are assigned.

A screen shot of a computer

Description automatically generated

1. **What is static keyword?**

* Static keyword is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested class.
* It belongs to the class than an instance of the class
* It gets memory only once at the time of class loading.
* Used to refer to the common properties of all object.

Advantage

There is only one copy of static field created in the memory. It is shared to all the objects. We need not create instance for accessing the static members, so it saves memory.

1. **What is this keyword?**

* this is a keyword that refers to the current instance of the class.
* Used to refer to the current class instance variable.
* Used to refer to the current class method.
* Used to pass current object as a parameter to another method
* Used to declare indexers

1. **What is this()?**

* This() is same as constructor chaining
* Invoking constructor from another constructor is called Chaining.
* this() method is used to invoke constructor

1. **What is the difference between this and this()?**

|  |  |
| --- | --- |
| **This** | **This()** |
| This keyword is used with the objects only. | This() is used with constructors only. |
| It refers to the current object. | It refers to the constructor of the same class whose parameters matches with the parameters passed to this(parameters). |
| Dot(.) operator is used to access the members.  For EG:  This.variableName; | No Dot(.) operator is used. Only the matching parameters are passed. |
| It is used to differentiate between the local variable and the instance variable in the method. | It is used to refer to the constructor belonging to the same class. |

1. **Define a local variable in C#.**

Local variables are referred to as variables that are defined in a code block. They are only visible in the code block they’re declared in.

1. **What is method overriding in C#?**

Method overriding means having two methods with the same name and same signature, one method in the base class and the other method in the derived class.

1. **Define method overloading in C#?**

Method overloading essentially is creating multiple methods with the same name and unique signatures with the same class. When you compile, the compiler makes use of overload resolution to determine the specific method which will be invoked.

1. **Difference between method overloading and overriding?**

|  |  |
| --- | --- |
| **Method Overloading** | **Method Overriding** |
| Creating more than one method or function having same name but different signatures or the parameters in the same class is called method overloading. | Creating a method in the derived class with the same signature as a method in the base class is called as method overriding |
| It is called the compile time polymorphism | It is called runtime polymorphism |
| Method overloading doesn’t need inheritance | Method overriding needs inheritance |
| Method overloading is also called early binding. | Method overriding is also called late binding. |

1. **Difference between Compile Time & Run time Polymorphism?**

|  |  |
| --- | --- |
| **Compile Time Polymorphism** | **Runtime Polymorphism** |
| It is method overloading technique. | It is a Method overriding technique |
| Defining multiple methods with same name and different signature(parameters). | Defining a method in the Child class with the same name and same signature of its Parent class. |
| Provides fast execution since the method to be executed is known as compile time | Provides slow execution since the method to be executed is known at runtime. |
| Less flexible since all things execute at compile time. | More flexible since all things execute at runtime. |

1. **What is the difference between the Virtual method and the Abstract method?**

A Virtual method must always have a default implementation. However, it can be overridden in a derived class by the keyword override.

An Abstract method doesn’t have any implementation. It resides in the abstract class, and also it is mandatory that the derived class must implement abstract class. Use of override keyword is not necessary.

1. **What is class?**

A class is a group of objects. It is a logical entity rather than a physical one. It takes up no memory and are also called templates for objects.

1. **What is Object?**

An object is an instance of a class, and a program may contain multiple instances of a given class. These objects have physical properties and behaviour.

1. **What are the Four Piller of OOPs?**

* Inheritance
* Encapsulation
* Abstraction
* Polymorphism

Inheritance - When one object acquires all the properties and behaviour of another object, it is called inheritance.

Encapsulation - Binding or wrapping up of code into a single unit, it is called encapsulation.

Abstraction - Hinding internal details and showing only the functionality is called as abstraction.

Polymorphism - If one task is performed in many ways, it is called as polymorphism.

1. **What are the various types of relationships in OOPs?**

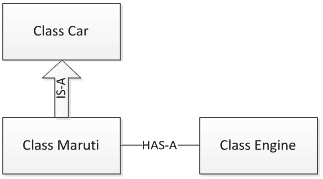
One of the advantages of an OOP is code reuse.

2 ways we can do code reuse.

- Inheritance (IS-A relationship)

- Object composition (HAS-A relationship).

Example to understand.



IS-A relationship.

* Based on inheritance.
* Example – Apple is a fruit, Car is a Vehicle
* Advantages – Code Reusability, Reduce Redundancy, Logical Hierarchy
* Disadvantages – Overhead (code complexity)

HAS-A relationship (composition/Aggregation)

* Whenever an instance of one class is used in another class
* Or we can say use of instance variables that are references to other objects.
* Example – Maruti has Engine, House has Bathroom.
* Advantages – Flexibility, Better Encapsulation
* Disadvantages – Learning curve due to deep class hierarchy, runtime overhead

1. **Why C# not supports Multiple Inheritance?**

C# does not support multiple inheritance through classes to avoid certain complications and ambiguities that can arise with this programming paradigm.

1. Diamond Problem

One of the main issues with multiple inheritance is the "diamond problem." This occurs when a class inherits from two classes that have a common ancestor. If both base classes have a method with the same name, it can create ambiguity when trying to call that method in the derived class.

2. Code Complexity

3. Implementation Challenges

1. **What is an Array?**

An array is a group of similar types of elements that have contiguous memory locations.

In C# array is an object of base type System.Array.

Advantages:

* Code Optimization
* Random Access
* Easy to manipulate data
* Easy to sort and traverse data

Disadvantages:

* Fixed Size

1. **What are the various types of Array?**

* Single Dimensional Array
* Multi-Dimensional Array
* Jagged Array

1. **What is multi dimensional array?**

A multidimensional array contains more than one level or dimension of array such as 2D and 3D array.

1. **What is Jagged Array?**

A Jagged array is referred to as an “array of arrays”. It is an array whose elements are arrays, the element of the same can be of different dimensions and sizes. The length of each array index can differ.

Example:

int[][] jagArray = new int[5][];

1. **How do we declare single dimensional array?**

int[] arr = new int[5];

int[] arr = new int[5]{10,20,30,40,50};

int[] arr = new int[]{10,20,30,40,50};

int[] arr = {10,20,30,40,50};

1. **What is access modifiers? What are the types of access modifiers?**

It is used to specify the accessibility or scope of a variable functions. C# supports 5 types of access specifiers.

1. Public
2. Protected
3. Internal
4. Protected Internal
5. Private

|  |  |
| --- | --- |
| Public | It specifies that access is not restricted. |
| Protected | It specifies that access is limited to the containing. |
| Internal | It specifies 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 | It specifies that access is limited to the containing type. |



1. **What is value type and reference type in C#?**

Value Type

* Directly contain their data, and they are stored in the memory where that variables is declared.
* Examples of value type include simple types like int, float, double, char and user defined structs.
* When you assign a value type to another variable, a copy of the value is made.

For example:

int a =10;

int b = a; // b gets a copy of the value in a

Reference types

* Stores a reference to the memory location where the data is actually stored. They don’t store the data directly.
* Example of reference type includes classes, interfaces, arrays, and delegates.
* When you assign a reference type to another variable, both variable reference the same object in memory.

1. **What is C# Collections?**

Collection represents group of objects. By the help of collection, we can perform various operations on objects such as store, update, delete, retrieve, search and sort object. We can store a objects in array or collection. Collection has advantages over array. Array has size limit but objects stored in collection can grow or shrink dynamically.

1. **What are the types of collection in C#?**

* System.Collections.Generic
* System.Collections
* System.Collections.Concurrent

1. **What is System.Collections.Generic classes?**

They have the following classes:

* List
* Stack
* Queue
* LinkedList
* HashSet
* SortedSet
* Dictionary
* SortedDictionary
* SortedList

1. **What is System.Collections?**

* ArrayList
* Stack
* Queue
* Hashtable

1. **What is System.Collections.Concurrent classes?**

They provide classes for thread-safe operations.

* BlockingCollection
* ConcurrentBag
* ConcurrentStack
* ConcurrentQueue
* ConcurrentDictionary
* Partitioner
* Partitioner
* OrderablePartitioner

1. **What is C# List?**

It is used to store and fetch elements. It can have duplicate elements. It is found in System.Collections.Generic namespace.

Var names = new List<string>();

1. **What is C# HashSet?**

C# HashSet class can be used to store, remove or view elements. It does not store duplicate elements. It is suggested to use HashSet class if you have to store only unique elements. It is found in System.Collections.Generic namespace.

Example:

Var names = new HashSet<string>();

1. **What is C# SortedSet?**

C# SortedSet class can be used to store, remove or view elements. It maintains ascending order and does not store duplicate elements. It is suggested to use SortedSet class if you have to store unique elements and maintain ascending order. It is found in System.Collections.Generic namespace.

1. **What is C# Stack?**

C# Stack<T> class is used to push and pop elements. It uses the concept of Stack that arranges elements in LIFO (Last In First Out) order. It can have duplicate elements. It is found in System.Collections.Generic namespace.

**What is C# Queue?**

C# Queue<T> class is used to Enqueue and Dequeue elements. It uses the concept of Queue that arranges elements in FIFO (First In First Out) order. It can have duplicate elements. It is found in System.Collections.Generic namespace.

1. **What is C# LinkedList?**

C# LinkedList<T> class uses the concept of linked list. It allows us to insert and delete elements fastly. It can have duplicate elements. It is found in System.Collections.Generic namespace.

It allows us to add and remove element at before or last index.

1. **What is C# Dictionary <TKey, TValue>?**

C# Dictionary<TKey, TValue> class uses the concept of hashtable. It stores values on the basis of

key. It contains unique keys only. By the help of key, we can easily search or remove elements. It

is found in System.Collections.Generic namespace.

1. **What is C# SortedDictionary?**

C# SortedDictionary<TKey, TValue> class uses the concept of hashtable. It stores values on the

basis of key. It contains unique keys and maintains ascending order on the basis of key. By the

help of key, we can easily search or remove elements. It is found in System.Collections.Generic

namespace.

1. **What is SortedList?**

C# SortedList<TKey, TValue> is an array of key/value pairs. It stores values on the basis of key. The

SortedList<TKey, TValue> class contains unique keys and maintains ascending order on the basis

of key. By the help of key, we can easily search or remove elements. It is found in

System.Collections.Generic namespace.

It is like SortedDictionary<TKey, TValue> class.

1. **What is the difference between C# SortedList<TKey, TValue> vs SortedDictionary<TKey,**

**TValue>?**

SortedList<TKey, TValue> class uses less memory than SortedDictionary<TKey, TValue>. It is

recommended to use SortedList<TKey, TValue> if you have to store and retrieve key/valye pairs.

The SortedDictionary<TKey, TValue> class is faster than SortedList<TKey, TValue> class if you

perform insertion and removal for unsorted data.

1. **What is Delegates?**

A delegate is a variable that holds the reference to a method or pointer to a method. It can refer to more than one method of same return types and parameters. Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System.Delegate class.

1. **What to use Delegate?**

When we need to pass a method as a parameter.

1. **What is a Multicast Delegate?**

A multicast delegate is a delegate that holds the references of more than one functions.

1. **What is anonymous Delegate?**

Delegates pointing methods without name are anonymous delegate.

1. **What is the difference between Events and Delegates?**

* The events is a notification mechanism that depends on delegate.
* An event is dependent on a delegate and cannot be created without delegates.
* Event is like a wrapper over the delegate to improve its security.

1. **What is Events?**

Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence

such as system generated notifications. Applications need to respond to events when they occur.

For example, interrupts. Events are used for inter-process communication.

The events are declared and raised in a class and associated with the event handlers using

delegates within the same class or some other class. The class containing the event is used to

publish the event. This is called the publisher class. Some other class that accepts this event is

called the subscriber class. Events use the publisher-subscriber model.

For example:

public Delegate void TestEvent();

public TestEvent TestEvent1;

1. **What is Generics in C#?**

Generics allows us to make classes and methods type independent or type safe. Generics are used to make reusable code classes which decrease the code redundancy. Using Generics one can do a variety of things like create collections. To create Generic collection, System.namespace.

1. **What is parameter types in C#?**

In C#, parameters are used to pass information into methods or constructors. There are different

types of parameters in C# based on how they are declared and used. Here are the common

parameter types:

Value Parameters (By Value)

* Passed by value, meaning a copy of the actual value is passed to the method.
* Changes made to the parameters inside the method do not affect the original value.

Reference Parameters (By Reference)

* Passed by reference meaning the actual memory address(reference) of the variable is passed.
* Changes made to the parameter inside the method affect the original value.
* Declared using the ref keyword.

Out Parameters

* Similar to reference parameters but used when a method is expected to assign a value to the parameter.
* The parameter must be assigned a value inside the method before it returns.
* Declared using the out keyword.

Params Parameter

* Allows a method to accept a variable number of parameters of the same type.
* The params keyword is used along with an array type.

1. **What is C# out parameter?**

C# provides out keyword to pass arguments as out type. It is like reference-type, except that it does not require variable to initialize before passing. We must use out keyword to pass argument as out type. It is useful when we want a function to return multiple values.

1. **Difference between ref and out keyword?**

|  |  |
| --- | --- |
| Ref Keyword | Out keyword |
| The passing of value through ref parameter is useful when the called method also need to change the value of passed parameters. | The declaring of parameters through out parameter is useful when a method return multiple values. |
| It is necessary the parameters should initialize before it passes to ref. | It is not necessary to initialize parameter before it pass it out. |
| It is not necessary to initialize the value of a parameter before returning to the calling method. | It is necessary to initialize the value of a parameter before returning to the calling method. |

1. **What is Type Casting in C#?**

Typecasting in C# is a mechanism that covers a certain type of value to another value. It is only probable when both the data types are well-suited with each other.

There are 2 main types of casting.

* Implicit Casting (Widening Conversion)
* Explicit Casting (Narrowing Conversion)

Implicit Casting

This type of casting is performed automatically by the compiler when there is no risk of data loss.

It is also known as widening conversion because you are converting to a larger data type.

For example, converting from int to long or from float to double

converting a smaller type to a larger type size

char -> int -> long -> float -> double

Explicit Casting

This type of casting requires the explicit use of the cast operator.

It is necessary when there is a risk of data loss, such as when converting from a larger data type to a smaller one.

Also known as narrowing conversion.

converting a larger type to a smaller size type

double -> float -> long -> int -> char

1. **What is upcasting and downcasting?**

Upcasting is implicitly converting the derived classes into a base class. Downcasting is just the opposite in which the base class is explicitly converted into a derived class.

1. **What is a safe and unsafe code in C#?**

A safe code is the one that runs by the management of CLR; and, an unsafe code does not run by the management of CLR.

1. **What is Boxing?**

The process of converting a Value Type variable (char, int etc.) to a Reference Type variable (object) is called Boxing. Value Type variables are always stored in Stack memory, while Reference Type variables are stored in Heap memory.

For example:

int num = 23; // 23 will assigned to num

Object obj = num; // Boxing

1. **What is Unboxing?**

The process of converting a Reference Type variable into a Value Type variable is known as Unboxing.

It is an explicit conversion process.

For example:

int num = 23; // value type is int and assigned value 23

Object Obj = num; // Boxing

int i = (int)Obj; // Unboxing

1. **What is using keyword in c#?**

In C#, the using keyword serves multiple purposes, and its meaning can vary depending on the context in which it is used. Here are the main uses of the using keyword:

Namespace Import:

The using keyword is commonly used to import namespaces in C#. This allows you to use types from a particular namespace without having to fully qualify their names.

Resource Management with using Statement:

The using keyword is also used to create a block of code within which a resource is allocated and automatically released when the block is exited. This is often used with objects that implement the IDisposable interface.

The using statement ensures that the Dispose() method of the IDisposable object (in this case, FileStream) is called, providing a convenient way to manage resources.

Alias for Namespace or Type:

The using keyword can be used to create an alias for a namespace or a type, especially when there might be naming conflicts.

A screen shot of a computer

Description automatically generated

Here, MyAlias is an alias for MyLongNamespace.MyLongTypeName, making it easier to use a shorter name within the code.

In summary, the using keyword in C# is versatile. It's used for importing namespaces, managing resources with the using statement, and creating aliases for namespaces or types.

1. **What is async and await?**

Async and Await are the two keywords that help us to program asynchronously. An async keyword is a method that performs asynchronous tasks such as fetching data from a database, reading a file, etc, they can be marked as “async”. Whereas await keyword making “await” to a statement means suspending the execution of the async method it is residing in until the asynchronous task completes. After suspension, the control goes back to the caller method. Once the task completes, the control comes back to the states where await is mentioned and executes the remaining statements in the enclosing method.

For example:

public async Task>CalculateCount()

{

await Task.Delay(2000);

return 1;

}

public async task mytestmethod()

{

Task> count = CalculateCount();

int result = await count;

1. **How do you do Exception Handling in C#?**

The following four keywords are used for Exception Handling in C#:

Try - The try block recognizes which block of code has particular exceptions activated.

Catch - The catch keyword signifies a program for catching an exception using an exception handler.

Finally - The finally block executes a given block of code whether or not an exception is caught.

Throw - Using the throw keyword, the program throws an exception in the event of a problem.

A screen shot of a computer program

Description automatically generated

1. **When and why to use Interface?**

* To achieve security
* And to implement multiple inheritance.

1. **What is the benefit of using statement in C#?**

The “using” statement can be used to obtain a resource for processing before automatically disposing it when execution is completed.

1. **What is String?**

String is a sequence of characters (i.e. zero or more characters) enclosed in between pair of double quotations. In C#, string is keyword which is an alias for System.String class.

Eg: “”, “A”, “Hello World”, “123”

1. **Why are strings in C# immutable?**

In C# Arrays have a fixed size, which means that once an array is created of some size, it cannot be dynamically increased or decreased in size. The CLR (Common Language Runtime) in C#, stores the strings as an array of characters. So whenever we delete or add a character or characters to a string, the original arrays of characters are fixed, and hence a new array of characters is created to accommodate the change. This is known as the immutability of strings in C#.

1. **What are sealed classes in C#?**

We create sealed classes when we want to restrict the class to be inherited. Sealed modifier used to prevent derivation from a class. If we forcefully specify a sealed class as base class, then a compile-time error occurs.

1. **Differentiate between Struct and Class in C#.**

|  |  |
| --- | --- |
| Struct | Class |
| Struct is a value type in C# that inherits values from System.Value | Class is a reference type in C#. Since it refers to objects, it inherits from System.Object |
| It is mostly used for small quantities of data | Classes are mostly used for large quantities of data |
| It cannot be inherited to any other type | Classes can be inherited to other classes |
| A Struct cannot have abstract values. | Classes can have abstract values. |

1. **What is Constructor? And types?**

Constructor can be defined as a special member function of a class that is responsible for initializing a newly created object of the class. They are called automatically when we create an instance of a class.

3 Types of constructor

Default – When there is no constructor defined, the compiler automatically creates a no argument constructor at runtime.

Parameterized - Constructor having any parameters are known as parametrized cons and not having params called as non-parameterized constructor.

Static constructor - It is used to initialize the static members of a class. It is called automatically before any static members are accessed or any static methods are called, and it is executed only once per type, typically when the type is first accessed or instantiated.

static Program(){

}

1. **What is the use of private constructor?**

A private constructor is a special instance constructor. It is generally used in classes that contain static members only. If a class has one or more private constructors and no public constructors, other classes (except nested classes) cannot create instances of this class.

1. **What is Constructor chaining in C#?**

When you want to share initialization code among multiple constructors, you can do it using constructor chaining. In Constructor chaining, the constructor calls another constructor in its class using the this().

For example:

public Test( bool a, int b, string c ): this( a, b )

{

this.m\_C = c;

}

public Test( bool a, int b, float d ) this( a, b )

{

this.m\_D = d;

}

private Test( bool a, int b )

{

this.m\_A = a;

this.m\_B = b;

}

1. **What is a Destructor in C# and when is it used?**

A destructor is a special method in C# that is automatically called when an object is destroyed. It is used to free up any resources that the object may have been using, such as memory or files. Destructors are usually implemented in a class and are denoted by the keyword ~ followed by the class name.

For example, if a class called MyClass was to have a destructor, it would be declared as follows: ~MyClass().

1. **What is Getter and Setter?**

Getter

- Returns the value [accessors]

- They are called accessors because they provide a way to access the value of a private variable from outside the class.

- Getters are used to retrieve the value of a private variable without allowing direct access to the variable itself.

Setter

- Sets / updates the value [mutators]

- They are called mutators because they provide a way to modify the value of a private variable from outside the class.

1. **What is fully encapsulated class?**

A fully encapsulated class is a class in object-oriented programming that hides its internal data and methods from the outside world, and only exposes a public interface for other classes to interact with. This means that the internal workings of the class are hidden and cannot be accessed or modified directly by other classes.

Encapsulation is important because it helps to ensure that the internal state of an object is not accidentally or maliciously modified by other parts of the program. It also allows for better organization and abstraction of code, making it easier to maintain and modify in the future.

To achieve full encapsulation, a class typically uses private member variables and methods, and provides public methods or properties to access and modify the data in a controlled way. This way, the class can enforce its own rules and constraints on how its data is used, without relying on external code to do so.

1. **Difference between Abstraction & Encapsulation.**

|  |  |
| --- | --- |
| 1. ABSTRACTION | ENCAPSULATION |
| It solves the problem in the design level. | It solves the problem in the implementation level. |
| It is used for hiding the unwanted data and giving relevant data. | It hides the code and data into a single unit to protect the data from outside world. |
| For simplicity, abstraction means hiding implementation using Abstract class and Interface. | It means hiding data using getter and setter. |
| It partitions the entity into many independent parts. Hence it is easy to read the abstract of entity. | It helps developer to organize the code better. |

For example

Class User

{

private string username;

private string passWd;

public void **registerUser**() // partitioning the entities

{

validateUser();

saveUserToDatabase();

}

public void validateUser()

{

//validation logic

}

}

Here registerUser(), validateUser() functions are partitioning entities, and inside these functions the methods, logic implementations is handled at the encapsulation level. So we can say just defining the function is called abstraction, and the implementation is the encapsulation.