SUNBEAM

SUNBEAM

Institute of Information Technology

PLACEMENT INITIATIVE

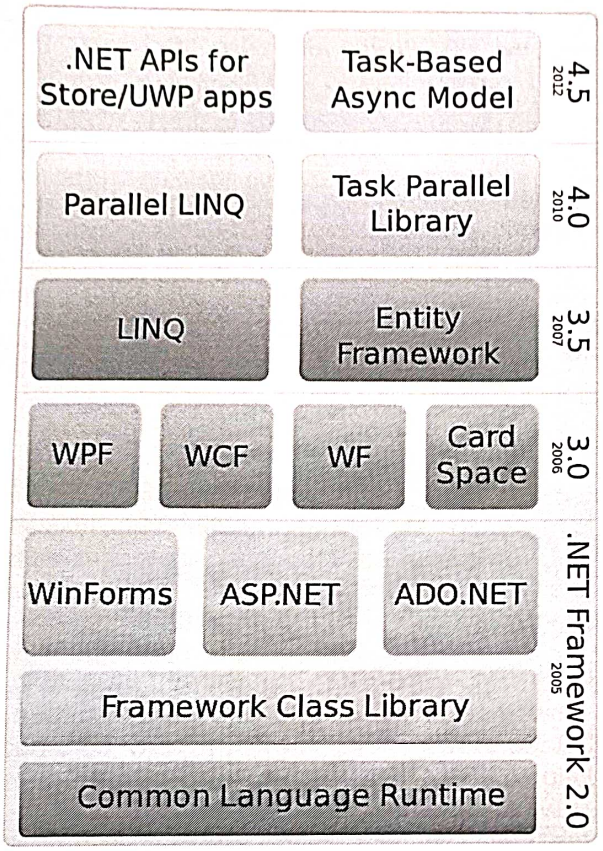
MS . NET NOTES

DAC

Course _____ Batch _____

Student Name _____ Reg ID _____

What is the .NET Framework?



The .NET Framework is a development platform for building apps for web, Windows, Windows Phone, Windows Server, and Microsoft Azure. It consists of the common language runtime (CLR) and the .NET Framework class library, which includes a broad range of functionality and support for many industry standards.

The .NET Framework provides many services, including memory management, type and memory safety, security, networking, and application deployment. It provides easy-to-use data structures and APIs that abstract the lower-level Windows operating system. You can use different programming languages with the .NET Framework, including C#, F#, and Visual Basic.

The .NET Framework is a managed execution environment for Windows that provides a variety of services to its running apps. It consists of two major components: the common language runtime (CLR), which is the execution engine that handles running apps, and the .NET Framework Class Library, which provides a library of tested, reusable code that developers can call from their own apps.

The services that the .NET Framework provides to running apps include the following:

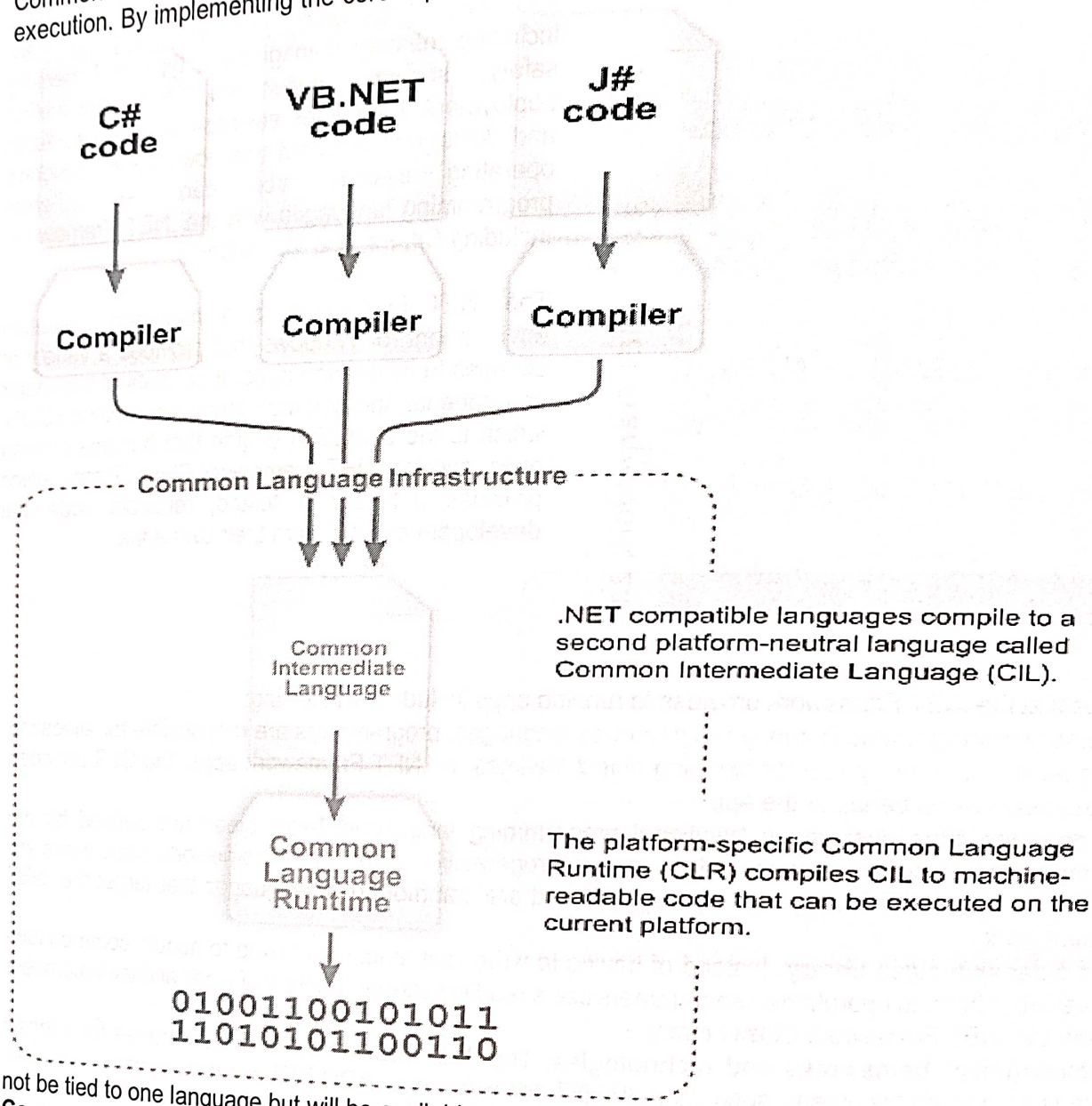
- **Memory management.** In many programming languages, programmers are responsible for allocating and releasing memory and for handling object lifetimes. In .NET Framework apps, the CLR provides these services on behalf of the app.
- **A common type system.** In traditional programming languages, basic types are defined by the compiler, which complicates cross-language interoperability. In the .NET Framework, basic types are defined by the .NET Framework type system and are common to all languages that target the .NET Framework.
- **An extensive class library.** Instead of having to write vast amounts of code to handle common low-level programming operations, programmers use a readily accessible library of types and their members from the .NET Framework Class Library.
- **Development frameworks and technologies.** The .NET Framework includes libraries for specific areas of app development, such as ASP.NET for web apps, ADO.NET for data access, Windows Communication Foundation for service-oriented apps, and Windows Presentation Foundation for Windows desktop apps.
- **Language interoperability.** Language compilers that target the .NET Framework emit an intermediate code named Common Intermediate Language (CIL), which, in turn, is compiled at runtime by the common language runtime. With this feature, routines written in one language are accessible to other languages, and programmers focus on creating apps in their preferred languages.
- **Version compatibility.** With rare exceptions, apps that are developed by using a particular version of the .NET Framework run without modification on a later version.
- **Side-by-side execution.** The .NET Framework helps resolve version conflicts by allowing multiple versions of the common language runtime to exist on the same computer. This means that multiple versions of apps can coexist and that an app can run on the version of the .NET Framework with which

it was built. Side-by-side execution applies to the .NET Framework version groups 1.0/1.1, 2.0/3.0/3.5, and 4/4.5.x/4.6.x/4.7.x/4.8.

- **Multitargeting.** By targeting .NET Standard, developers create class libraries that work on multiple .NET Framework platforms supported by that version of the standard. For example, libraries that target the .NET Standard 2.0 can be used by apps that target the .NET Framework 4.6.1, .NET Core 2.0, and UWP 10.0.16299.

Common Language Infrastructure (CLI)

Common Language Infrastructure (CLI) provides a language-neutral platform for application development and execution. By implementing the core aspects of .NET Framework within the scope of CLI, these functions will



not be tied to one language but will be available across the many languages supported by the framework.

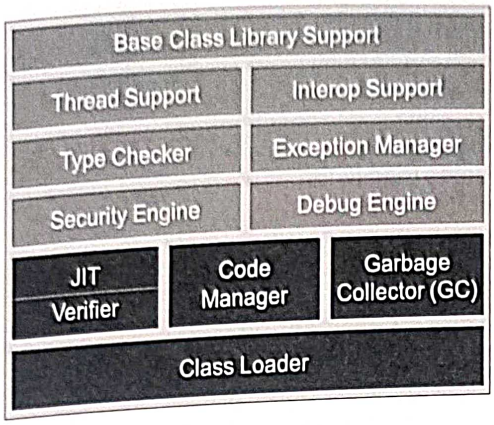
Common Language Runtime (CLR)

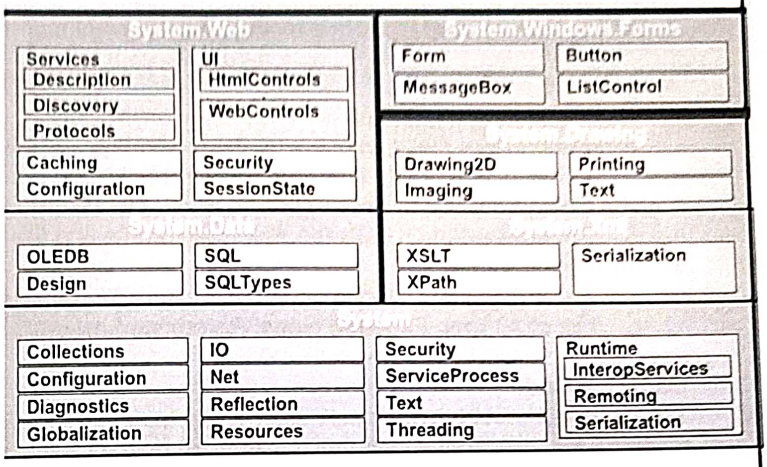
.NET Framework includes the Common Language Runtime (CLR). It serves as the execution engine of .NET Framework and offers many services such as memory management, type safety, exception handling, garbage collection, security and thread management. All programs written for .NET Framework are executed by the CLR.

Programs written for .NET Framework are compiled into Common Intermediate Language code (CIL), as opposed to being directly compiled into machine code. During execution, an architecture-specific just-in-time compiler (JIT) turns the CIL code into machine code.

With Microsoft's move to .NET Core, the CLI Virtual Execution System (VES) implementation is known as CoreCLR instead of CLR.

.NET Framework Classes





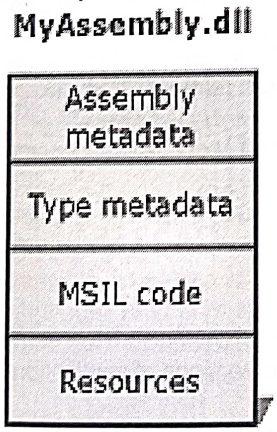
The .NET Framework Class Library (FCL) is organized in a hierarchy of namespaces. Most of the built-in application programming interfaces (APIs) are part of either ***System.**** or ***Microsoft.**** namespaces. These class libraries implement many common functions, such as file reading and writing, graphic rendering, database interaction, and XML document manipulation. The class libraries are available for all CLI compliant languages. The FCL implements the CLI Base Class Library (BCL) and other class libraries—some are specified by CLI and other are Microsoft specific.

BCL includes a small subset of the entire class library and is the core set of classes that serve as the basic API of CLR..NET Framework most classes considered being part of BCL reside in ***mscorlib.dll, System.dll and System.Core.dll***. BCL classes are available in .NET Framework as well as its alternative implementations including .NET Compact Framework, Microsoft Silverlight, .NET Core and Mono. FCL refers to the entire class library that ships with .NET Framework. It includes an expanded set of libraries, including BCL, Windows Forms, ASP.NET, and Windows Presentation Foundation (WPF) but also extensions to the base class libraries ADO.NET, Language Integrated Query (LINQ), Windows Communication Foundation (WCF), and Workflow Foundation (WF). FCL is much larger in scope than standard libraries for languages like C++

Assemblies

Source code written in C# is compiled into an intermediate language (IL) that conforms to the CLI specification. The IL code, along with resources such as bitmaps and strings, is stored on disk in an executable file called an assembly(.exe/.dll).

An assembly contains a manifest that provides information on the assembly's types, version, culture, and security requirements.



Assembly Functions

It contains IL code that CLR executes. It contains Manifest that makes PE files executables.An assembly is the unit at which permissions are requested and granted.Every type's identity includes the name of the assembly in which it resides. The assembly's manifest contains assembly metadata that is used for resolving types and satisfying resource requests.The assembly is the smallest versionable unit in the common language runtime; all types and resources in the same assembly are versioned as a unit.

It forms a deployment unit. When an application starts, only the assemblies that the application initially calls must be present. Other assemblies, such as localization resources or assemblies containing utility classes, can be retrieved on demand. This allows applications to be kept simple and thin when first downloaded.

It is the unit at which side-by-side execution is supported thus Assemblies are a fundamental part of programming with the .NET Framework

Parts of Assembly

Static assembly can consist of four elements (as shown in diagram). Assemby can be Single file assembly or multifile assembly.Multifile assembly is linked through Manifest and managed as single unit by CLR.multifile assembly allow to combine modules written in different languages and to optimize downloading an application by putting seldom used types in a module that is downloaded only when needed.

Assembly Manifest

- Each assembly's manifest –
  - Enumerates files that make up the assembly.
  - Governs how references to the assembly's types and resources map to the actual files
  - Enumerates other assemblies on which the assembly depends.
  - Provides isolation between assembly clients and the assembly's implementation details.
- Manifest Contains
  - Assembly name, Version number, Culture for sattelite assembly
  - List of All files, Exported types, Referenced assemblies

Managed Code

The CLR also provides other services related to automatic garbage collection, exception handling, and resource management. Code that is executed by the CLR is sometimes referred to as "managed code," in contrast to "unmanaged code" which is compiled into native machine language that targets a specific system.

Types of Assembly

Assemblies can be static or dynamic. Static assemblies can include .NET Framework types and resources. Static assemblies are stored on disk in portable executable (PE) files.
You can also use the .NET Framework to create dynamic assemblies, which are run directly from memory and are not saved to disk before execution.
You can also use common language runtime APIs, such as System.Reflection.Emit, to create dynamic assemblies.

Execution of Assembly

When the C# program is executed, the assembly is loaded into the CLR, which might take various actions based on the information in the manifest.
Then, if the security requirements are met, the CLR performs just in time (JIT) compilation to convert the IL code into native machine instructions.These can be passed to OS.

Global Assembly Cache

It is recommended that only users with Administrator privileges be allowed to delete files from the global assembly cache. When an assembly is added to the global assembly cache, integrity checks are performed on all files that make up assembly.
The cache performs these integrity checks to ensure that assembly has not been tampered with. Assembly registered with GAC is shared assembly while assembly placed in application folder is local or private assembly.
Assemblies deployed in the global assembly cache must have a strong name.

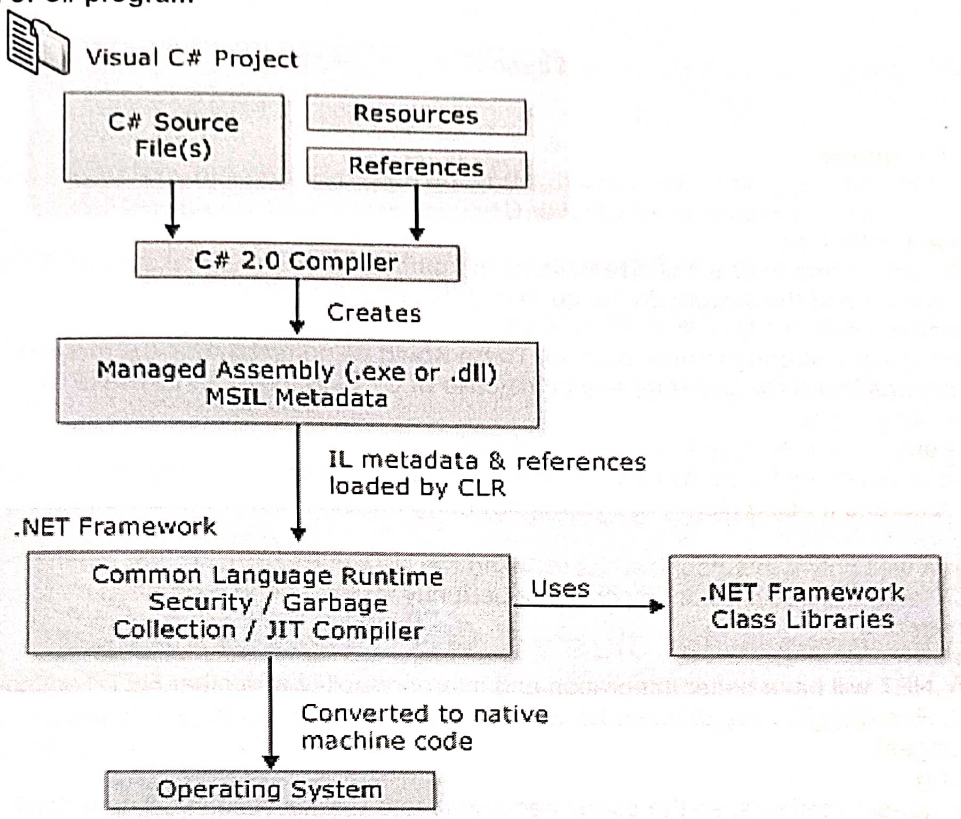
Strong named assembly

A strong name consists of the assembly's identity – its simple text name, version number, and culture information – plus a public key and a digital signature.
It is generated from an assembly file using the corresponding private key.Strong names guarantee name uniqueness by relying on unique key pairs.
When you reference a strong-named assembly, you expect to get certain benefits, such as versioning and naming protection.

Side-by-side Execution

Side-by-side execution is the ability to store and execute multiple versions of an component on the same computer. This means that you can have multiple versions of the runtime, and multiple versions of applications that use a version of the runtime, on the same computer at the same time.
Side-by-side execution gives you more control over what versions of a component and runtime an application binds to. Support for side-by-side storage and execution of different versions is possible due to strong naming and is built into the infrastructure of the runtime.

Execution of C# program



Application domains

Application domains provide an isolation boundary for security, reliability, and versioning, and for unloading assemblies. Application domains are typically created by runtime hosts, which are responsible for bootstrapping the common language runtime before an application is run.

Language Interoperability

Language Interoperability simply means a function in VB .NET can be called by C# and vice-versa. When .NET code is compiled, it results in MSIL, regardless of what language was used to write it, be it C#, VB.NET, F#, etc. The resulting MSIL code can be used by a different language than the one it was written in. Language interoperability is the ability of code to interact with code that is written using a different programming language. Language interoperability can help maximize code reuse and, therefore, improve the efficiency of the development process. Every code written in any .NET language and compiled could be reused from other .NET language.

The IL disassembler

The IL Disassembler is a companion tool to the IL Assembler (*Ilasm.exe*). *Ildasm.exe* takes a portable executable (PE) file that contains intermediate language (IL) code and creates a text file suitable as input to *Ilasm.exe*.

The text file produced by *Ildasm.exe* can be used as input to the IL Assembler (*Ilasm.exe*). This is useful, for example, when compiling code in a programming language that does not support all the runtime metadata attributes. After compiling the code and running its output through *Ildasm.exe*, the resulting IL text file can be hand-edited to add the missing attributes. You can then run this text file through the IL Assembler to produce a final executable file.

C# Language

C# language was created by Anders Hejlsberg at Microsoft and launched in 2000. C# is a simple, modern, flexible, object-oriented, safe, and open source programming language

C# syntax simplifies many of the complexities of C++ while providing powerful features such as nullable value types, enumerations, delegates, anonymous methods and direct memory access, which are not found in Java. The C# build process is simple compared to C and C++ and more flexible than in Java. A C# source file may define any number of classes, structs, interfaces, and events.

Streanth of C#

1. Object- Oriented Language
C# is pure object-oriented language, this allows you to create modular maintainable applications and reusable codes. This is one of the biggest advantages of C# over C++.

2. Automatic Garbage Collection
C# has got a very efficient system to erase and remove all the garbage present on the system. C# doesn't create a mess in the system and the system do not get hanged during execution.

3. No Problem if Memory Leak
C# has a major advantage of a strong memory backup. There would be no problem of the memory leak and other such type of problems in the C# as it happens in the case of C++ language. In this case C# has a very clear edge on all other languages.

4. Easy-to-Development
The rich class libraries make many functions easy to be implemented. C# has influence on most of the programmers of the world and it has a history in the programming world.

5. Cross Platform
Your application will run well only if the machine has installed the NET framework. This is the most important requirement for the C#. also this could be an important opportunity for the young programmers to get them trained with .NET framework.

6. Better Integration
Applications written in .NET will have better integration and interperatability with other NET Technologies. Actually C# runs on CLR, making it easy to integrate with components written in other languages (specifically, CLR-compatible languages)

7. More Legible Coding
Formalized concept of get-set methods, so the codes becomes more legible. Also in C#, you don't need to worry about header files. Coding would be a worth to do in C#.

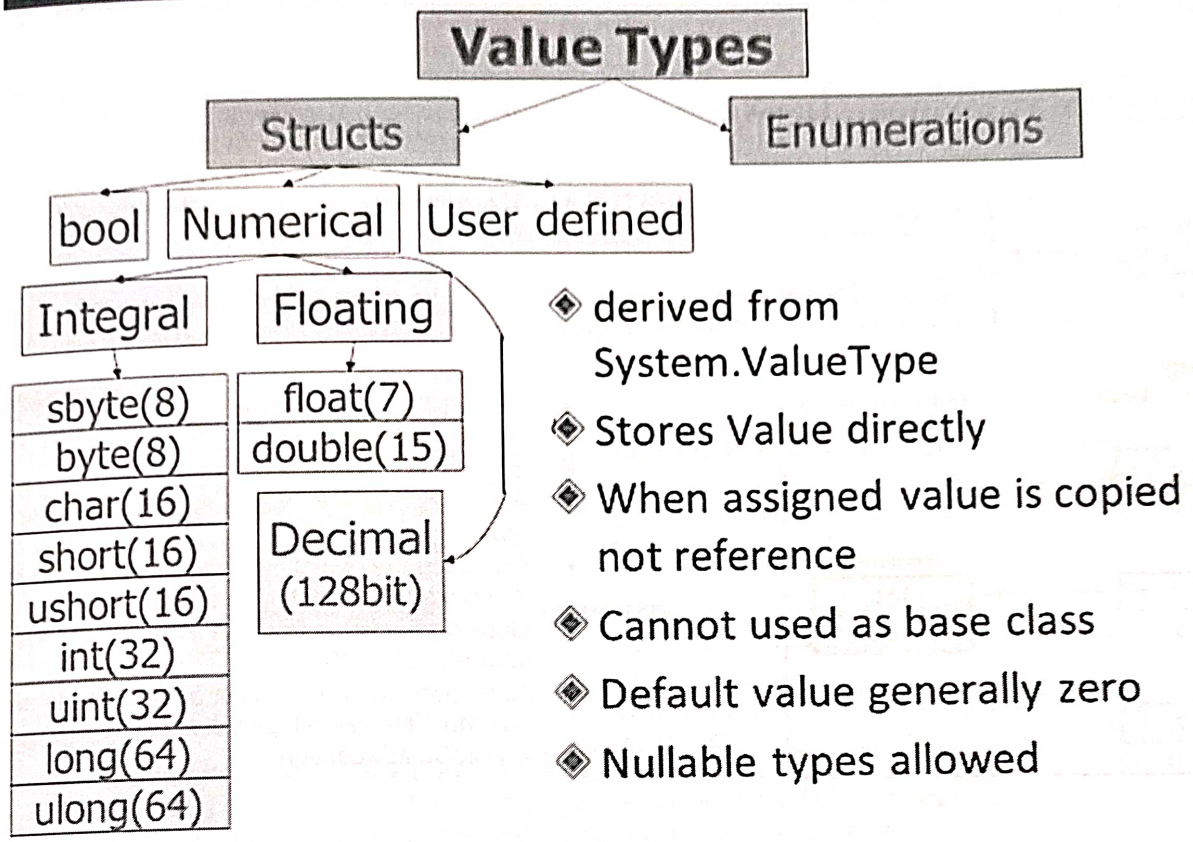
8. Scarcity of Choice
When you are in Microsoft stack, you have a tool for everything. So, basically you match your needs to the tool, and you use it. That's why I recommend C# is very supportive kind of language specially for the beginners.

9. Programming Support
You can buy support from the Microsoft in C# (.NET framework) unlike Java where community is your support. So if things get wrong then you can solve your issues with the support of Microsoft.

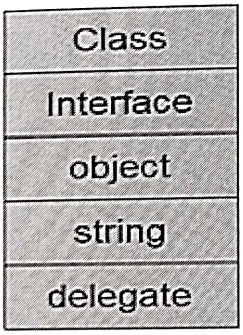
10. Backward Compatibility
.NET applications work on windows platforms only and Microsoft keeps retiring support for old windows platforms. So always there would be a need to upgrade your .NET framework if you are going on a new version of the windows

Data Types In C#

- C# is a strongly typed language
- Classified as Built-in type and User-defined type
- Or Classified as Value type – stores values and Reference type – stores references to data.



Reference Types

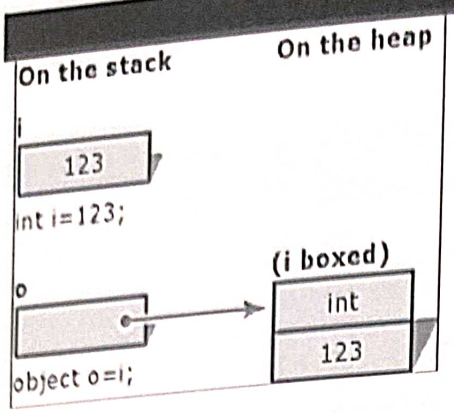


- Classes and Interfaces are user defined.
- object is cosmic super class
- string is also built in type represents immutable strings
- delegate represent object oriented function pointer
- store references to actual data

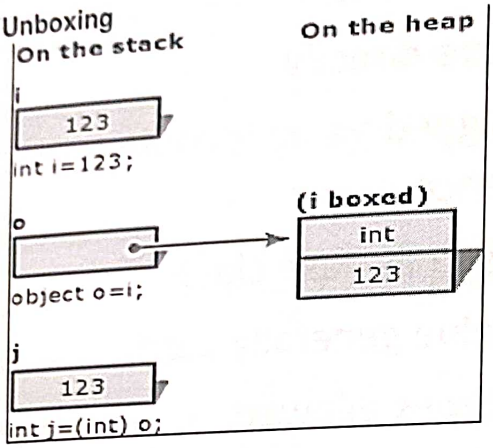
Boxing and Unboxing

- By Boxing Value types can be treated as objects.
- This allows the value type to be stored on the garbage collected heap as an Object.
- Unboxing extracts the value type from the object.
- example, the integer variable i is *boxed* and assigned to object o.
- int i = 123;
- object obj = (object) i; // boxing
- The object o can then be unboxed and assigned to integer variable i:
- object obj = 123;
- i = (int) obj; // unboxing

Boxing



- object o = i; // implicit boxing
- object o=(object) i; //explicit boxing(not required)
- The result of this statement is creating an object reference o, on the stack, that references a value of the type int, on heap.
- This value is a copy of the value-type value assigned to the variable



- int i = 123; // a value type
- object o = i; // boxing
- int j = (int) o; // unboxing
- explicit conversion from type object to value type
- Checking the object instance to make sure it is a boxed value of the given value type.
- Copying the value from the instance into the value-type variable.
- Attempting to unbox null or a reference to an incompatible value type will result in an InvalidCastException

Arrays

The Array class gives methods for creating, manipulating, searching, and sorting arrays. The Array class is not part of the System.Collections namespace, but it is still considered as a collection because it is based on the IList *interface*. The Array class is the base class for language implementations that support arrays.

- int[] a={1,2,3,4,5}; int[] a=new int[5];
- int[] a=new int[5]{1,2,3,4,5};
- string[] weekDays = new string[] { "Sun", "Mon", "Tue";
- int[,] a=new int[4,2];
- int[,] a=new int[,]{{1,2},{3,4},{5,6}};

```
int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };
foreach (int i in numbers) {   System.Console.WriteLine(i); }
Console.WriteLine(numbers.Length);
int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };
foreach (int i in numbers)
    System.Console.WriteLine(i);
Console.WriteLine(numbers.Length);
```

- A jagged array is an array whose elements are arrays.
- The elements of a jagged array can be of different dimensions and sizes.
- A jagged array is sometimes called an "array of arrays."

string type [mscorlib.dll]

- sequence of Unicode characters
- alias for reference type System.String
- == and != operators check for values even though it is reference type
- +, [], += operators can be used
- To avoid escape seq: str=@"c:\file.txt";

- To modify the same string object use System.Text.StringBuilder class string[] split(char[] seperators);

Objects, classes and structs

- Objects are instances of a given data type. The data type provides a blueprint for the object that is instantiated when application is executed.
- New data types are defined using classes & structs.
- Classes and structs form the building blocks of C# applications, containing code and data. C# application will always contain of at least one class.
- A struct can be considered a lightweight class, ideal for creating data types that store small amounts of data, and does not represent a type that might later be extended via inheritance.
- C# classes support inheritance, meaning they can derive from a previously defined class.

Class

- class may have
  - Methods, Fields, Properties, Operators, Indexers, Events & other user defined types
- Access specifiers
  - private, protected, public
  - internal, protected internal
- Class definitions can be splitted in files
- Static classes are sealed classes with static methods
- Only single inheritance is allowed
- Multiple interface inheritance is allowed

Structure

- Structs are value types; classes are reference types
- When passing a struct to a method, it is passed by value instead of as a reference.
- Unlike classes, structs can be instantiated without using a new operator.
- Structs can declare constructors, but they must take parameters.
- A struct cannot inherited from other stuct.A struct can implement interfaces.
- All structs are inherit from System.ValueType that inherits System.Object
- It is an error to initialize an instance field in struct

Inheritance & Polymorphism

- is operator checks for hierarchy relation
- bool flag = der_obj is base;
- as operator is used for dynamic type casting
- derived newobj = obj as derived;
- abstract classes & members are solely used for inheritance - to define features of derived, non-abstract classes.
- sealed members cannot be overridden
- sealed classes cannot be further inherited; they are final.
- public class derived : base { ... }

Operator overloading

- By operator overloading, we can extend the meaning of the operator. By this we can use operator with user defined classes.
- The operators are implemented as static functions of the class.
- &&, ||, [ ], ( ), shorthand,=, ., ?:, →, new, is, sizeof operators cannot be overloaded.
- Relational operators like ==, != should be overloaded in pair.
- public static Complex operator +(Complex c1, Complex c2);
- Conversion operators are used to convert a class object into any other type and vice versa.
- Conversions declared as implicit occur automatically when required.
- Conversions declared as explicit require a cast to be called.

new Keyword

- new operator is used to allocated memory.
- When function is declared as new, it hides all the functions in base class with the same name. Using a derived class reference we can't access any function with same name in base class.
- We can even declare a function as new virtual, which makes function as a new virtual function hence forth in hierarchy that can be overridden in further derived classes using override keyword. Using a base class reference we cannot access any such function with same signature in the derived class.

sealed Keyword

- Function declared as sealed cannot be overriden in the derived class.Thus sealed functions finalizes functionality provided by the base class, that cannot be modified or extended in the derived class.
- The specifier sealed override tells that function is overridden in this class but cannot be overridden further
- If we wish to have a new function in the derived class with same name we should use new.
- If a class is declared as sealed, then that class cannot be further inherited.

const, readonly Keywords

- The variable declared as const, is a constant variable and cannot be modified anywhere. The const field's value should be given at compile time, where it is declared in class.
- const double PI = 3.142;
- Now value of PI is finalized to 3.142.
- The variable declared as readonly, is assigned in constructor and cannot be modified in any other method afterwards.The readonly field's value may not be known at the compile time.

abstract Keyword

- If function is declared as abstract within a class then we cannot create object of that class. Such class must be declared as abstract.
- Functions declared abstract in base class must be overriden in the derived class. Otherwise we cannot even create object of the derived class.
- Abstract classes are used to implement some common functionality that can be shared by the objects of the derived classes.
- Function declared as abstract override overrides function in base class but make it abstract in derived class.

Use of abstract classes

- Abstract class references are used to refer to objects of the derived classes. Thus they work like generic references and give specific implementation depending on type of object. Also they implement the common functionality used by the derived class objects.
- Generally we create collection like array of base class references and now we can store any objects derived from this common base class.Using this technique we can achieve polymorphic behavior.

Interface

- Interface is full abstraction of a class. In interface all methods are by default public & abstract.Interface cannot have any non-abstract method or fields.
- The fields can be declared only as public static const.
- Multiple interface inheritance is allowed, even though implementation inheritance is not allowed.
- If class A is implementing I1 and I2 interfaces, A should override all methods from I1 & I2.

Fragile Base Class Problem

- If base class defination is modified, then all derived classes must be recompiled. To avoid such problem we can go for interafaces, as interfaces are immutable.
- Interfaces are service based. It is a contract between service provider and service user.It declares set of rules in form of its methods, that will be implemented by the service provider.Interfaces ensure that rules or services are implemented by the service provider. Using interface you can collect irrelevent objects together, by getting a service common to both.Example: IPrintable interface has Print() method, that can be implemented by Point or Employee class. The implementation will differ according to actual class.

IComparable interface, ICloneable are some of the standard interfaces.Interface can have only static and const members.const should be known at compile time.readonly can be assigned during intialization or constructor

IComparable Interface

- Defines a generalized type-specific comparison method that a value type or class implements to order or sort its instances. This interface is implemented by types whose values can be ordered or sorted.
- It requires that implementing types define a single method, CompareTo(Object), that indicates whether the position of the current instance in the sort order is before, after, or the same as a second object of the same type.
- The instance's IComparable implementation is called automatically by methods such as Array.Sort and ArrayList.Sort.

IComparer Interface

- This interface is used in conjunction with the Array.Sort and Array.BinarySearch methods. It provides a way to customize the sort order of a collection.
- See the Compare method for notes on parameters and return value. Its generic equivalent is the

ICloneable

- Supports cloning, which creates a new instance of a class with the same value as an existing instance. The ICloneable interface enables you to provide a customized implementation that creates a copy of an existing object.
- The ICloneable interface contains one member, the Clone method, which is intended to provide cloning support beyond that supplied by Object.MemberwiseClone. For more information about cloning, deep versus shallow copies, and examples, see the Object.MemberwiseClone method.

User-defined conversions

- User-defined conversions are performed by special methods that you can define to enable explicit and implicit conversions between custom types that do not have a base class–derived class relationship.

Namespaces and Assemblies

- Assemblies are for physical scope and namespaces are for logical scope, so namespaces can be expanded over assemblies but the converse is not possible. The namespace keyword is used to declare a scope. This namespace scope lets you organize code and gives you a way to create globally unique types.
- Even if you do not explicitly declare one, a default namespace is created. This unnamed namespace, sometimes called the global namespace, is present in every file. Any identifier in the global namespace is available for use in a named namespace. Namespaces are hierarchical. They implicitly have public access, which you cannot modify.

enum

- enum is a value type data type. The enum is used to declare a list of named integer constants.
- It can be defined using the *enum* keyword directly inside a namespace, class, or structure.
- The enum is used to give a name to each constant so that the constant integer can be referred using its name.

Example

```
enumWeekDays {
Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
}
Console.WriteLine((int)WeekDays.Monday);
Console.WriteLine((int)WeekDays.Friday);
int dayNum = (int)WeekDays.Friday;
Console.WriteLine(dayNum);
```

Properties

- A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field.
- Properties can be used as if they are public data members, but they are actually special methods called *accessors*.
- This enables data to be accessed easily and still helps promote the safety and flexibility of methods.

Auto-implemented properties

- In some cases, property get and set accessors just assign a value to or retrieve a value from a backing field without including any additional logic.
- By using auto-implemented properties, you can simplify your code while having the C# compiler transparently provide the backing field for you.

Indexer

- An Indexer is a special type of property that allows a class or structure to be accessed the same way as array for its internal collection.
- It is same as property except that it defined with **this** keyword with square bracket and parameters.
- Indexers enable objects to be indexed in a similar way to arrays.
- Indexers do not have to be indexed by an integer value.
- Indexers can be overloaded.Indexers can have more than one formal parameter, for example, when accessing a two-dimensional array.
- It can be declared as abstract & can be interface member.

Preprocessor

- The preprocessor directives give instruction to the compiler to preprocess the information before actual compilation starts.
- All preprocessor directives begin with #, and only white-space characters may appear before a preprocessor directive on a line. Preprocessor directives are not statements, so they do not end with a semicolon (;).

Sr.No.	Preprocessor Directive & Description
1	**#define:** It defines a sequence of characters, called symbol.
2	**#undef:** It allows you to undefine a symbol.
3	**#if:** It allows testing a symbol or symbols to see if they evaluate to true.
4	**#else:** It allows to create a compound conditional directive, along with #if.
5	**#elif:** It allows creating a compound conditional directive.
6	**#endif:** Specifies the end of a conditional directive.
7	**#line:** It lets you modify the compiler's line number and (optionally) the file name output for errors and warnings.
8	**#error:** It allows generating an error from a specific location in your code.
9	**#warning;** It allows generating a level one warning from a specific location in your code.
10	**#region:** It lets you specify a block of code that you can expand or collapse when using the outlining feature of the Visual Studio Code Editor.

11	**#endregion:** It marks the end of a #region block.

Delegates

- Delegate is a type that references a method. It can be treated as object oriented function pointer.
- public delegate int PerformCalculation(int x, int y);
- Any method that matches the delegate's signature, which consists of the return type and parameters, can be assigned to the delegate.
- It makes delegates ideal for defining callback methods.
- Delegates can be chained together called multicast delegate. Delegates are mainly used with Eventing design pattern.

Events

- Ex: delegate void TestEventDelegate();
- public event TestEventDelegate TestEvent;
- .NET framework provide guideline for creating event. The first parameter should be sender object while second should provide information about event derived from EventArgs.
- Event may have more than one handlers that are executed sequentially like multicast delegate.
- Add or remove event accessors can written that are executed when += or -= operator is used on event.
- Event can be declared in interface and overridden into derived class.

Reflection

- CLR loads the assembly into appdomain.
- Assemblies contain modules, modules contain types, and types contain members.
- Reflection provides objects that encapsulate assemblies, modules, and types.
- Reflection can be used to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object.
- The related classes are declared in System.Reflection namespace.
- The classes of the System.Reflection.Emit namespace provide a specialized form of reflection that enables you to build types at run time.
- The GetType method or typeof operator provides Type structure that keeps information of type.
- Providing access to nonpublic information involves security risks. Code that is allowed to discover nonpublic information on a type can potentially access code, data, and other information you want to keep private.
- Therefore, .NET Framework security enforces rules that determine to what degree reflection can be used to discover type info and access types.
- Depending on the operation being performed, a ReflectionPermission or the SecurityPermission for serialization might be required.

Reflection classes

- Use Assembly to define and load assemblies, load modules that are listed in the assembly manifest, and locate a type from this assembly and create an instance of it.Use Module to discover information such as the assembly that contains the module and the classes in the module, all global methods or other specific, nonglobal methods defined on the module.
- Use ConstructorInfo to discover information such as the name, parameters, access modifiers and implementation details (abstract or virtual) of a constructor. Use the GetConstructors or GetConstructor method of a Type to invoke a specific constructor.Use PropertyInfo to discover information such as the name, data type, declaring type, reflected type, and read-only or writable status of a property, and to get or set property values.Use ParameterInfo to discover information such as a parameter's name, data type, whether a parameter is an input or output parameter, and the position of the parameter in a method signature.
- Use CustomAttributeData to discover information about custom attributes when you are working in the reflection-only context of an application domain. CustomAttributeData allows you to examine attributes without creating instances of them.

Attributes

- Attributes provide a powerful method of associating declarative information with types, methods, properties, etc.
- Attributes add metadata to your program. Metadata is information embedded in your program such as compiler instructions or descriptions of data.
- They are classified as predefined(given in framework) and user defined attributes.
- Predefined attributes are used in PInvoke, Serialization, Interoperability, etc.

Custom Attributes

You can create your own custom attributes by defining an attribute class, a class that derives directly or indirectly from Attribute, which makes identifying attribute definitions in metadata fast and easy.
System.AttributeUsage attribute decides usage of the attribute.

- AttributeTargets – Class, Struct, Field, Property, All, etc.
- AllowMultiple = true / false
- Inherited = true / false

Regular Expression

- A regular expression is a pattern that could be matched against an input text. The .Net framework provides a regular expression engine that allows such matching. A pattern consists of one or more character literals, operators, or constructs.
- There are various categories of characters, operators, and constructs that lets you to define regular expressions The **Regex class** is used for representing a regular expression

Generics

- For every new value type parameter specialized version is created at runtime
- For reference types only version in created for storing refernces as refereces of any type have same size.
- To use any collection with foreach loop it should support iterator or IEnumerable interface.
- Enumerator can be returned by "yield" keyword. We can write more than one yield return statement.
- Yield statement cannot be there in unsafe blocks or try-catch blocks, anonymous methods, method with ref & out params.

Generics in C#

- Generics are template classes or functions written by programmer.C# generic type substitutions are performed at runtime. Use generic types to maximize code reuse, type safety, and performance.
- Collection classes declared in System.Collections.Generic Generic interfaces, classes, methods, events and delegates can be created.

Collection classes and Interfaces

- Collection classes are defined as part of the System.Collections or System.Collections.Generic namespace
- Most collection classes derive from the interfaces **ICollection, IComparer, IEnumerable, IList, IDictionary, and IDictionaryEnumerator**.
- Some important classes are **ArrayList, SortedList, Stack, Queue, HashTable**, etc.
- Using generic collection classes provides increased type-safety and in some cases can provide better performance, especially when storing value types

ArrayList Class

- It represents an ordered collection of an object that can be indexed individually. It is basically an alternative to an array.
- However, unlike array you can add and remove items from a list at a specified position using an **index** and the array resizes itself automatically. It also allows dynamic memory allocation, adding, searching and sorting items in the list.

Example

```
ArrayList arrlist = new ArrayList();
arrlist.Add(11);
arrlist.Add(22);
arrlist.Add(9);
arrlist.Add(34);
arrlist.Add(887);
Console.WriteLine("Capacity: {0} ", al.Capacity);
Console.WriteLine("Count: {0}", al.Count);
foreach (int i in arrlist)
Console.Write(i + "");
```

Stack class

- It represents a last-in, first out collection of object. It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.

Hashtable class

The Hashtable class represents a collection of key-and-value pairs that are organized based on the hash code of the key. It uses the key to access the elements in the collection.

Example

```
Hashtable hashtb = new Hashtable();
hashtb.Add("abc1","Krishna");
hashtb.Add("abc2","Kaveri");
hashtb.Add("abc3","Venna");
if (hashtb.ContainsValue("Krishna"))
        Console.WriteLine("This name is already present in the list");
else
        hashtb.Add("abc4", "Nitin");
ICollection key = hashtb.Keys;
foreach (string i in key)
        Console.WriteLine(i + ": " + hashtb [i]);
```

Dictionary class

- The Dictionary<TKey, TValue> class is a generic collection class in the System.Collection.Generics namespace. TKey denotes the type of key and TValue is the type of TValue.

Example

```
IDictionary<int, string> d = newDictionary<int, string>();
d.Add(1,"SunbeamKarad");
d.Add(2,"SunbeamPuneMarketYard");
d.Add(3,"ThreePuneHinjawadi");
foreach (KeyValuePair<int, string> i in d)
Console.WriteLine("Key: {0}, Value: {1}", i.Key, i.Value); //accessing Dictionary using foreach

for (int i = 0; i < d.Count; i++)
Console.WriteLine("Key: {0}, Value: {1}", d.Keys.ElementAt(i), d[ dict.Keys.ElementAt(i)]);

Console.WriteLine(d [1]); //accessing Dictionaryindividual element
Console.WriteLine(d [2]);
```

HashSet Class

- A set is a collection that contains no duplicate elements, and whose elements are in no particular order.
- A **HashSet<T>** is an unordered collection of the **unique elements**. It comes under ***System.Collections.Generic*** namespace.The HashSet<T> class provides high-performance set operations.

- The capacity of a HashSet<T> object is the number of elements that the object can hold.A HashSet<T> collection is not sorted and cannot contain duplicate elements.
- HashSet<T> provides many mathematical set operations, such as set addition (unions) and set subtraction.

Example

```
HashSet<int> num = new HashSet<int>();
for (int n = 0; n < 7; n++)
        num.Add(n);
foreach(int n in odd)
        Console.WriteLine(i);
```

List class

List<T> collection is the same as an ArrayList except that List<T> is a generic collection whereas ArrayList is a non-generic collection.The List<T> class implements eight different interfaces that provide different functionalities, those are IEnumerable<T>, IEnumerable, ICollection<t>, ICollection,IReadOnlyList<T>, IList<T>, IList, IReadOnlyCollection<T>

```
IList<Employee> emp = newList<Student>() {
newEmployee (){empID=1, empName="Nitin"},
newEmployee (){empID=2, empName="Prashant"},
newEmployee (){empID=3, empName="Nilesh"},
newEmployee (){empID=4, empName="Sandip"}
foreach(Employee e in emp)
        Console.WriteLine(e.ToString());
```

Exceptions

- Exception is flexible way to deal with any unexpected or exceptional situations that arise while a program is running. Exception handling uses the try, catch, and finally keywords to attempt actions that may not succeed, to handle failures and to clean up resources at endWhen exceptional circumstance is encountered, such as a division by zero or low memory warning, an exception is generated by CLR.
- Once an exception occurs, the flow of control immediately jumps to an associated exception handler, if one is present program stops executing with an error message.try block contain code that may result in exception. An exception handler is catch block of code that is executed when an exception occurs. One try may have mutiple catch blocks. The first catch statement that can handle the exception is executed. Exceptions can be explicitly generated by a program using the throw keyword.
- Exception objects contain detailed information about the error, including the state of the call stack and a text description of the error. Code in a finally block is executed even if an exception is thrown, thus allowing a program to release resources

Checked & Unchecked Exceptions

C# statements can execute in either checked or unchecked context.
In a checked context, arithmetic overflow raises an exception.
Expressions using the following predefined operators on integral types: ++ — - (unary) + - * /
Explicit numeric conversions between integral types.
In unchecked context, arithmetic overflow is ignored & result is truncated.

Example:

```
int z = 0;
short x=32767, y=32767;
try {
        z =     checked((short)(x + y));
}
catch (System.OverflowException e) {
        Console.WriteLine(e.ToString());
}
```

blocks can be used for expressions

try...catch...finally

- Put the suspected code in the **try block** that may raise an exception followed by a catch or finally block. Exception raised within try block can be handled using the **catch block**. Code in the catch block will only execute when an exception occurs. A **multiple catch block** can also be specified with a different exception type is called exception filters.
- The **finally block** must come after a **try orcatch block**. The finally block will always be executed whether or not an exception is thrown. The finally block is generally used for cleaning-up code e.g. for disposing an unmanaged objects.

Example:

```
staticvoid Main(string[] args) {
int num1=12,num2=0;
try {
int result = num1/num2;  // this will throw an exception
}
catch(Exception err)  {
Console.WriteLine("Inside catch block Type of Exception= {0}", err.Message );
}
finally {
Console.WriteLine("Inside finally block");
}
}
```

Threading

- Operating systems use processes to separate the different applications that they are executing.
- Threads are the basic unit to which an OS allocates processor time, and more than one thread can be executing code inside that process.
- The .NET Framework subdivides an OS process into lightweight managed subprocesses, called application domains(System.AppDomain).
- One or more threads (System.Threading.Thread) can run in one or any number of application domains within the same managed process.
- EachAppdomain is having at least one thread.
- Thread can easily switch into Appdomains
- It can create additional application domains and additional threads.
- Managed thread can move freely between appdomains inside the same managed process

Thread synchronization

Thread synchronization is done by lock keyword. It is internally using Monitor object for that.

```
lock(obj) {  }
```

This is equivalent to

- Monitor. Enter or Monitor.TryEnter
- Monitor.Exit

The object passed to Enter() should be same as Exit(), otherwise Monitor throws SynchronizationLockException.You can also use extra methods like Wait(), Pulse() and PulseAll().To avoid conflicts, you must synchronize, or control the access to, shared resources.Failure to synchronize access properly can lead to problems such as deadlocks and race conditionsThe system provides synchronization objects that can be used to coordinate resource sharing among multiple threads.Resources that require synchronization include:

- System resources (communications ports)
- Resources shared by multiple processes (such as file handles).
- The resources of a single application domain accessed by multiple threads.

Life Cycle of a Thread

- States: Highest, AboveNormal, Normal, BelowNormal, Lowest

WPF

- WPF, stands for Windows Presentation Foundation is a development framework and a sub-system of .NET Framework. WPF is used to build Windows client applications that run on Windows operating system. WPF uses XAML as its frontend language and C# as its backend languages.
- WPF was introduced as a part of .NET Framework 3.0 as the Windows library to build Windows client apps and the next generation of Windows Forms.
- WPF is the engine that is responsible for creating, displaying, and manipulating user-interfaces, documents, images, movies, and media in Windows 7 and later Windows operating systems. WPF is a set of libraries that have all functionality you need to build, run, execute, and manage Windows client applications.

XAML

- XAML is a new descriptive programming language developed by Microsoft to write user interfaces for next-generation managed applications. XAML is used to build user interfaces for Windows and Mobile applications that use Windows Presentation Foundation (WPF), UWP, and Xamarin Forms.
- The purpose of XAML is simple, to create user interfaces using a markup language that looks like XML. Most of the time, you will be using a designer to create your XAML but you're free to directly manipulate XAML by hand. XAML uses the XML format for elements and attributes. Each element in XAML represents an object which is an instance of a type.

WPF Controls

- WPF has a set of rich UI controls. These controls support visual actions such as drag and drop, setting properties and events, data binding, and setting up resources and templates.

Data Binding in WPF

Binding allows you to link a source object to some control. There are the following two types of bindings:

- One-Way Binding: Binds a source to the interface.
- Two-Way Binding: Binds a source to the interface and back to the source.

INotifyPropertyChanged interface allows sources to interact with the interface and update it as the values change.

- To bind an object or a list to an element you must set the Data Context property.
- It is possible to bind an object or a list of objects and you can bind one element to another.
- To customize how bound data will be displayed you can set the DataTemplate from a control.
- It is possible to set Data Converters to convert the source type to another type.

LINQ

- LINQ (Language Integrated Query) is uniform query syntax in C# and VB.NET to retrieve data from different sources and formats. It is integrated in C# or VB, thereby eliminating the mismatch between programming languages and databases, as well as providing a single querying interface for different types of data sources.
- For example, SQL is a Structured Query Language used to save and retrieve data from a database. In the same way, LINQ is a structured query syntax built in C# and VB.NET to retrieve data from different types of data sources such as collections, ADO.Net Dataset, XML Docs, web service and MS SQL Server and other databases.
- Most of us are good at firing queries on database. Ever imagined how would it be if one can fire SQL like queries on collection, arrays or XML?
- What do we do to find a Customer class object from the collection whose address starts with 'M'?
- Consider below collection of customer objects:

```
List<Customer> customers = new List<Customer>() {
new Customer(){ No =1 , Name = "Amit", Address = "Pune"},
new Customer(){ No =2 , Name = "Mahesh", Address = "Mumbai"},
new Customer(){ No =3 , Name = "Nilesh", Address = "Manglore"},
```

```
new Customer(){ No =4 , Name = "Nitin", Address = "Panji"}
};
```

- As we can see the sample data; we have two customer objects with Address starting with 'M'
- It means after looping through entire collection; we shall get two objects. But this is sample data, may get any number of objects or one or none based on real time data. How to hold the data then? Should we go ahead and declare a collection or single object?
- That's the last thing to worry about. First let us be concerned about- "Can we make this easy"?
- Yes!
- Let us use LINQ to collection by firing SQL like queries on the collection. In order to hold the result let us use 'VAR' type discussed in earlier chapter.

- "Why 'VAR'?"
- "We don't know what we are going to get!"
- Here goes the code for the same:

```
var result = from c in customers
where c.Address.StartsWith("M")
select c;
```

- In the code above the SQL like syntax used is similar to Hibernate Query Language (HQL). Hibernate is one of the popular ORM framework.
- Syntax is not straight forward like SQL SELECT statement. Had it been SQL one would have written query like:
- "SELECT * FROM CUSTOMERS WHERE ADDRESS = 'M%' "
- Why do we have such a weird syntax?
- Answer is - In the LINQ query used above; 'FROM', 'WHERE', and 'SELECT' are not keywords! Those are actually methods! Output of one method becomes an input to another method. It is more of chain of methods.

- Good point about this query is the query does not execute at the mentioned statement. It means if one tries to execute only LINQ statement above; result will actually have nothing!

- LINQ query executes only when result is going to get used for the first time. It is called differed execution.
- It means when one actually tries to use 'result' variable e.g. may be iteration of the result; then the query executes.
- However if one changes the query like given below; one may see the result with data immediately:

```
var result = (from c in customers
where c.Address.StartsWith("M")
select c).ToList();
```

- In above statement; change is we have asked to convert the entire result of the query in to LIST format. It means we have asked for immediate result! Hence in this case query executes immediately.

File IO – Serialization:

Files and Directories:

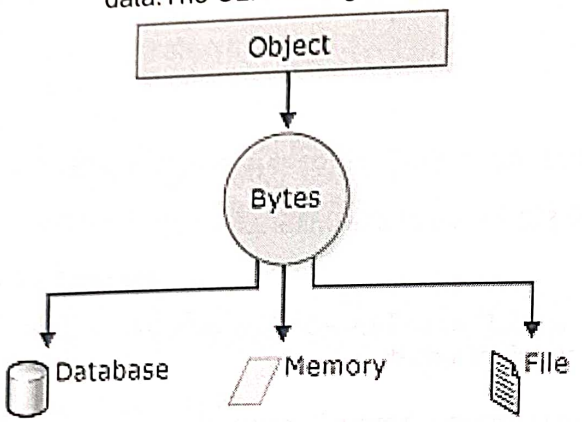
Some of the very commonly used classes are as follows:

- File - provides static methods for creating, copying, deleting, moving, and opening files, and helps create a FileStream object.
- FileInfo - provides instance methods for creating, copying, deleting, moving, and opening files, and helps create a FileStream object.
- Directory - provides static methods for creating, moving, and enumerating through directories and subdirectories.
- DirectoryInfo - provides instance methods for creating, moving, and enumerating through directories and subdirectories.
- Path - provides methods and properties for processing directory strings in a cross-platform manner.

- Streams: Stream is nothing but a sequence of bytes. Stream in .net is an abstract base class. It could be a file, an input output device, an IPC, or a TCP/IP socket. The programmer is thus isolated from the specific details of operating system. Basic operations that can be performed on a stream are read, write and seek.

Serialization

- Serialization is the process of converting the state of an object into a form that can be persisted or transported. The complement of serialization is deserialization, which converts a stream into an object.Binary serialization preserves type fidelity, which is useful for preserving the state of an object between different invocations of an application.
  - XML serialization serializes only public properties and fields and does not preserve type fidelity. This is useful when you want to provide or consume data without restricting the application that uses the data.The CLR manages object memory management and provides an automated serialization.



- When an object is serialized, the name of the class, the assembly, and all the data members of the class instance are written to storage. When the class is serialized, the serialization engine ensure that the same object is not serialized more than once, if object has references. The only requirement placed on object graphs is that all objects, referenced by the serialized object, must also be marked as Serializable.
- When the serialized class is deserialized, the class is recreated and the values of all the data members are automatically restored.
- Serialization can be done as Binary or XML or SOAP serialization.
- If you don't want to store any member then it should be marked with NonSerialized attribute. You can also use XMLSerializer object with StreamWriter object to write in file.
- Steps for serialization
  - Mark the class with Serializable attribute.
  - Create a formatter object – Binary or SOAP
  - Create FileStream object to store data
  - Serialize object into file using formatter
  - Close the stream

Types of serialization:

There are basically 3 types of serialization:

- **Binary:** Binary serialization uses binary encoding to produce compact serialization for uses such as storage or socket-based network streams.
- **XML:** XML serialization serializes the public fields and properties of an object, or the parameters and return values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document.

- **SOAP Serialization:** XML serialization can also be used to serialize objects into XML streams that conform to the SOAP specification. SOAP is a protocol based on XML, designed specifically to transport procedure calls using XML. As with regular XML serialization, attributes can be used to control the literal-style SOAP messages generated by an XML Web service.

Now consider the following code, in which an Employee class is marked as Serializable

```
[Serializable]
class Emp
{
        private int Eid;
        private string Ename;
        public int _Eid
        {
                get     {       return this.Eid;        }
                set     {       this.Eid = value;       }

        }
        public string _Ename
        {
                get     {       return this.Ename;      }
                set     {       this.Ename = value;     }

        }
}
```

Consider serialization code below:

Binary Serialization Code Snippet:

```
string path=@"D:\demo\data.bin";
FileStream fs=new FileStream(path,FileMode.OpenOrCreate,FileAccess.Write);
BinaryFormatter bf = new BinaryFormatter();
Emp e = new Emp();
e._Eid = 101;
e._Ename = "Sharma";
bf.Serialize(fs, e);
Console.WriteLine("Object serialized..");
Console.ReadLine();
fs.Close();
Emp enew = null;
fs=new FileStream(path,FileMode.Open,FileAccess.Read);
enew = (Emp)bf.Deserialize(fs);
Console.WriteLine("Emp id: " + enew._Eid.ToString() + " Emp name : " +    enew._Ename.ToString());
Console.ReadLine();
```

FOR SOAP add reference to assembly and import namespace with the name "System.Runtime.Serialization.Formatters.Soap" Consider below code for serialization & deserialization.

SOAP Serialization Code Snippet:

```
string path = @"D:\demo\data.xml";
FileStream fs =new FileStream(path, FileMode.OpenOrCreate, FileAccess.Write);
SoapFormatter sf = new SoapFormatter(); //Binary serialization
Emp e = new Emp();
e._Eid = 101;
e._Ename = "Sharma";
```

```
sf.Serialize(fs, e); // Serialize the object in a file
Console.WriteLine("Object serialized..");
Console.ReadLine();
fs.Close();
//SOAP De-serialization
Emp enew = (Emp)sf.Deserialize(fs);
Console.WriteLine("Emp id: " + enew._Eid.ToString() + " Emp name : " + enew._Ename.ToString());
Console.ReadLine();
```

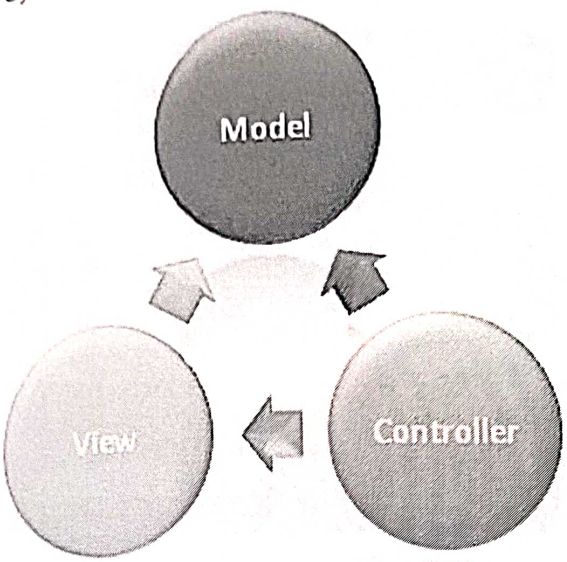
FOR XML Serialization; import namespace : "System.Xml.Serialization" from System.Xml.dll

XML Serialization Code Snippet:

```
string path = @"D:\demo\data.xml";
FileStream fs =new FileStream(path, FileMode.OpenOrCreate, FileAccess.ReadWrite);
XmlSerializer xs = new XmlSerializer(typeof(Emp));
Emp e = new Emp();
e._Eid = 101;
e._Ename = "Sharma";
xs.Serialize(fs, e); // Serialize the object in a file
Console.WriteLine("Object serialized..");
Console.ReadLine();
fs.Close();
//XML De-serialization
Emp enew = (Emp)xs.Deserialize(fs);
Console.WriteLine("Emp id: " + enew._Eid.ToString() + " Emp name : " + enew._Ename.ToString());
Console.ReadLine();
```

ASP.NET MVC

MVC is a standard design pattern that is Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller. The ASP.NET MVC framework is a lightweight, highly testable presentation framework that (as with Web Forms-based applications) is integrated with existing ASP.NET features, such as master pages and membership-based authentication.
The MVC framework is defined in the System.Web.Mvc namespace and is a fundamental, supported part of the System.Web namespace.



Model: Model is data and business logic. It maintains the data of the application. Model objects retrieve and store model state in a database. Model is nothing but a data and business logic.

View: View is a user interface. View display data using model to the user and also enables them to modify the data.

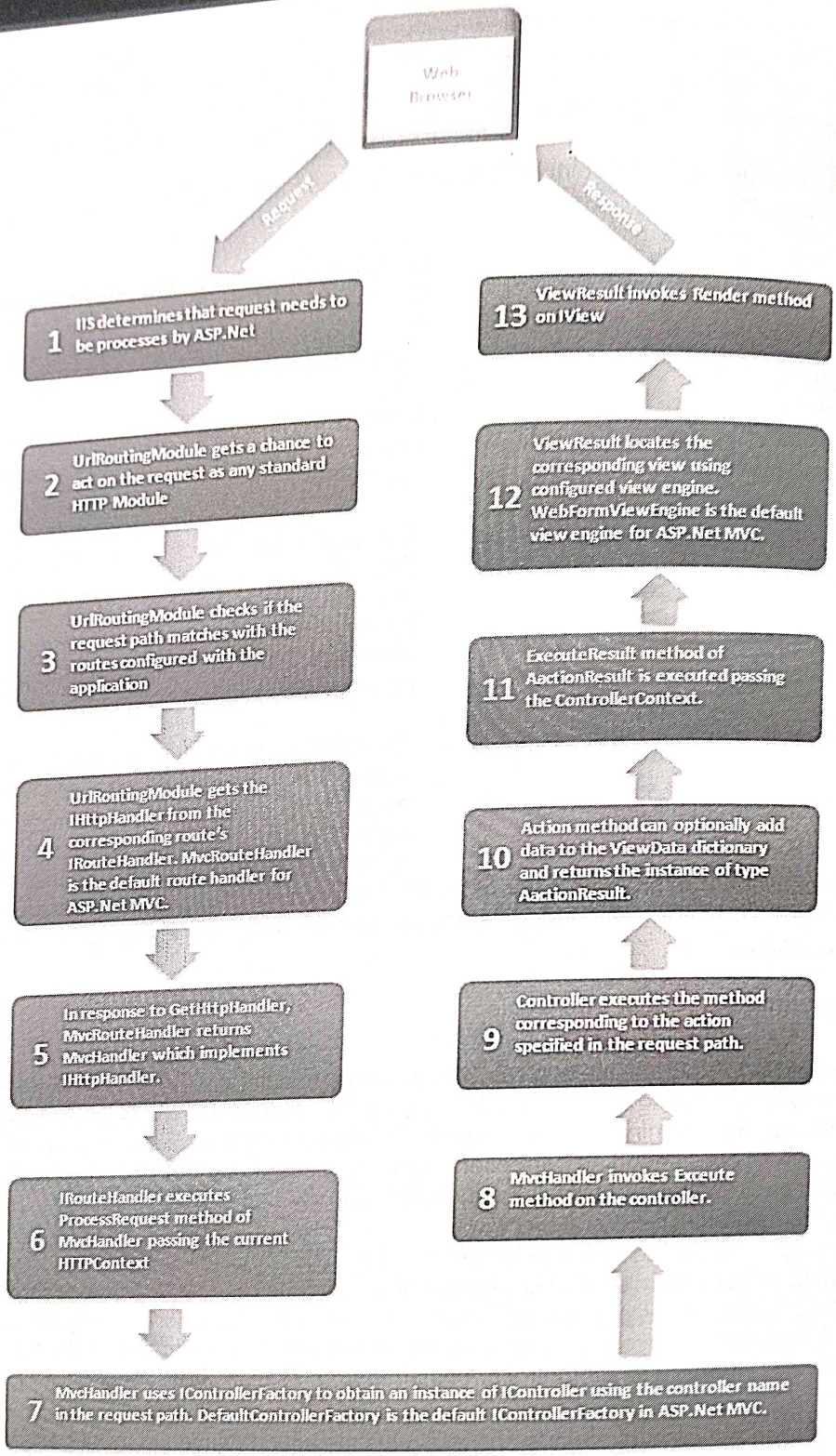
Controller: Controller handles the user request. Typically, user interact with View, which in-turn raises appropriate URL request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response. Controller is nothing but a request handler.

The MVC pattern helps you create multi tier applications that separate the different aspects of the application i.e. input logic , business logic, and UI logic. The MVC pattern helps you to reducde the complexity of your application, easy to test

The loose coupling between the three main tier of an MVC application also promotes parallel development i.e. one cae develope View, 2nd can work on Models and at the same time 3rd developer can work on Controller part for same functionality.

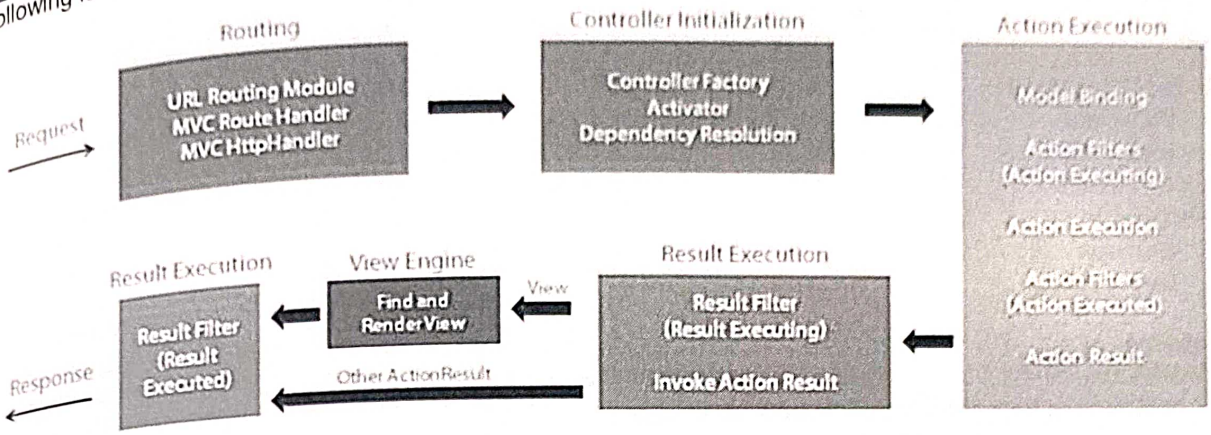
Understanding Routing & Request Life Cycle

- The entry point of MVC Request life cycle is URL Routing module, the incoming request from IIS pipeline is handed over to URL Routing module which analyses the request and looks up Routing table to figure out which controller the incoming request maps to. Routing Table is a static container of routes defined in MVC application with corresponding controller action mapping. If the route is found in the routing table MVCRouteHandler executes and brings the instance of MVCHttpHandler. Together they act as a gateway into the MVC Framework.
- MVC handler begins initializing and executing controller. The MVCHttpHandler also takes of converting route data into concrete controller that is capable of serving the request. MVC handler does all this with the help of MVC Controller factory and activator which are responsible for creating an instance of the controller. This is also the place where the Dependency Injection is performed if the application has been designed to invoke parameterized controller constructor and satisfy its dependencies.
- After the controller instance is created the next major step is to find and execute the corresponding action. A component called ActionInvoker finds and executes the action defined in routing table. Before the action method is called model bindings takes place which maps data from http request to action method parameters. After the model binding, action filters are invoked which includes OnActionExecuting filter. This is followed by action execution and Action Executed filter execution and finally preparing Action Result.



- Once the Action method has been finished executing the next step is Result execution. MVC separates the action declaration from Result execution. If the Result from action execution is view, then depending upon configuration, ASPX or Razor view engine will be called to find and render the html view as a response of http request. If the result was not view then it's passed as-is to http response.

Following is the conceptual view of MVC request life cycle.



Bundling and Minification:

Bundling is a new feature in ASP.NET 4.5 that makes it easy to combine or bundle multiple files into a single file. You can create CSS, JavaScript and other bundles. Fewer files means fewer HTTP requests and that can improve first page load performance.

Minification performs a variety of different code optimizations to scripts or css, such as removing unnecessary white space and comments and shortening variable names to one character. Consider the following JavaScript function.

HTML Helpers:

HtmlHelper	Strogly Typed HtmlHelpers	Html Control
Html.ActionLink		Anchor link
Html.TextBox	Html.TextBoxFor	Textbox
Html.TextArea	Html.TextAreaFor	TextArea
Html.CheckBox	Html.CheckBoxFor	Checkbox
Html.RadioButton	Html.RadioButtonFor	Radio button
Html.DropDownList	Html.DropDownListFor	Dropdown, combobox
Html.ListBox	Html.ListBoxFor	multi-select list box
Html.Hidden	Html.HiddenFor	Hidden field
Password	Html.PasswordFor	Password textbox
Html.Display	Html.DisplayFor	Html text
Html.Label	Html.LabelFor	Label
Html.Editor	Html.EditorFor	Generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int, double or other numeric type.

Data Validation in MVC:

Data Annotations:

Attribute	Description
Required	Indicates that the property is a required field
StringLength	Defines a maximum length for string field
Range	Defines a maximum and minimum value for a numeric field
RegularExpression	Specifies that the field value must match with specified Regular Expression
CreditCard	Specifies that the specified field is a credit card number
CustomValidation	Specified custom validation method to validate the field
EmailAddress	Validates with email address format
FileExtension	Validates with file extension
MaxLength	Specifies maximum length for a string field

Attribute	Description
MinLength	Specifies minimum length for a string field
Phone	Specifies that the field is a phone number using regular expression for phone numbers

State Management

- **ViewData**
- **ViewBag**
- **TempData**
- **Session**
- **Cookies**
- **QueryString**

ViewData

- ViewData is used to pass data from controller to view
- It is derived from ViewDataDictionary class
- It is available for the current request only
- Requires typecasting for complex data type and checks for null values to avoid error
- If redirection occurs, then its value becomes null

ViewBag

- ViewBag is also used to pass data from the controller to the respective view
- ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
- It is also available for the current request only
- If redirection occurs, then its value becomes null
- Doesn't require typecasting for complex data type

TempData

- TempData is derived from TempDataDictionary class
- TempData is used to pass data from the current request to the next request
- It keeps the information for the time of an HTTP Request. This means only from one page to another. It helps to maintain the data when we move from one controller to another controller or from one action to another action
- It requires typecasting for complex data type and checks for null values to avoid error. Generally, it is used to store only one time messages like the error messages and validation messages

Session:

- In ASP.NET MVC Session manages to store and retrieve values for a user when user navigates between views.
- Generally session is used to store user's information to uniquely identify a user.
- The server maintains the state of user information by using a session ID.
- When users makes a request without a session ID, ASP.NET creates a session ID and sends it with every request and response to the same user.'
- Session state is stored in the Session key/value dictionary.

Query Strings:

- A Query String is a string variable which is appended to the end of the Page URL.
- It can be used to send data across pages. It stores information in a key/value pair.
- A "?" signature is used to append the key and value to the page URL
- In MVC application we can pass query string values along with a route parameter.

Cookies:

- Cookie is a small text file which is created by the server and stored on the client hard disk by the browser. It does not use server memory.
- Generally a cookie is used to identify users. When user sends a request to the server, the server creates a cookie and attaches a header and sends it back to the user along with the response.

- The browser accepts the cookie and stores it on the client machine either permanently or temporarily.
- The next time the user makes a request for the same site, the browser checks the existence of the cookie for that site in the folder.
- If the cookie exists it sends a request with the same cookie, else that request is treated as a new request.

Query Strings:

- A Query String is a string variable which is appended to the end of the Page URL. It can be used to send data across pages. It stores information in a key/value pair. A "?" signature is used to append the key and value to the page URL. In MVC application we can pass query string values along with a route parameter id like

Partial View:

- Partial view is a reusable view, which can be used as a child view in multiple other views. It eliminates duplicate coding by reusing same partial view in multiple places. You can use the partial view in the layout view, as well as other content views.

Action Method and child action:

All the public methods of a Controller class are called Action methods. They are like any other normal methods with the following restrictions:

1. Action method must be public. It cannot be private or protected
2. Action method cannot be overloaded
3. Action method cannot be a static method.

In our projects, when we want an action method to not be called from a URL request we should make the method a child action method.An action method can be a child or a normal action method, but child actions are action methods invoked from within a view, you cannot invoke a child action method via user request (URL)annotate an action method with the [ChildActionOnly] attribute to create a child action.
An action method can be a child or a normal action method, but child actions are action methods invoked from within a view, you cannot invoke a child action method via user request (URL).

Filters: In ASP.NET MVC, a user request is routed to the appropriate controller and action method. However, there may be circumstances where you want to execute some logic before or after an action method executes.
Filters can be applied to an action method or controller in a declarative or programmatic way. Declarative means by applying a filter attribute to an action method or controller class and programmatic means by implementing a corresponding interface.

Filter Type	Description	Built-in Filter	Interface
Authorization filters	Performs authentication and authorizes before executing action method.	[Authorize], [RequireHttps]	IAuthorizationFilter
Action filters	Performs some operation before and after an action method executes.		IActionFilter
Result filters	Performs some operation before or after the execution of view result.	[OutputCache]	IResultFilter
Exception filters	Performs some operation if there is an unhandled exception thrown during the execution of the ASP.NET MVC pipeline.	[HandleError]	IExceptionFilter

Authentication Filters: Authentication filters are new in MVC version 5 and provide the means to provide fine-grain control over how users are authenticated for controllers and actions in an application Authentication filter runs before any other filter or action method. Authentication confirms that you are a valid or invalid user. Action filters implements the IAuthenticationFilter interface.

Authorization Filters: The AuthorizeAttribute and RequireHttpsAttribute are the examples of Authorization Filters. Authorization Filters are responsible for checking User Access; these implement the IAuthorizationFilterinterface in the framework. These filters used to implement authentication and authorization for controller actions. For example, the Authorize filter is an example of an Authorization filter. The **OnAuthenticationChallenge** method is called by the MVC Framework whenever a request has failed the authentication or authorization policies for an action method. The OnAuthenticationChallenge method is passed an AuthenticationChallengeContext object, which is derived from the ControllerContext class

Action Filters: Action Filter is an attribute that you can apply to a controller action or an entire controller. This filter will be called before and after the action starts executing and after the action has executed. Action filters implement the IActionFilter interface that have two methods OnActionExecuting andOnActionExecuted. OnActionExecuting runs before the Action and gives an opportunity to cancel the Action call. These filters contain logic that is executed before and after a controller action executes, you can use an action filter, for instance, to modify the view data that a controller action returns. The ActionFilterAttribute class implements both the IActionFilter and IResultFilter interfaces. This class is abstract, which forces you to provide an implementation

OnActionExecuting Method:
The OnActionExecuting method is called before the action method is invoked. You can use this opportunity to inspect the request and elect to cancel the request, modify the request, or start some activity that will span the invocation of the action.

OnActionExecuted Method: You can also use the filter to perform some task that spans the execution of the action method. As a simple example, I have created a class file called ProfileActionAttribute.cs in the Infrastructure folder, and used it to define a class that measures the amount of time that an action method takes to execute

Result Filters: The OutputCacheAttribute class is an example of Result Filters. These implement the IResultFilter interface which like the IActionFilter has **OnResultExecuting and OnResultExecuted**. These filters contains logic that is executed before and after a view result is executed. Like if you want to modify a view result right before the view is rendered to the browser.

ExceptionFilters: The HandleErrorAttribute class is an example of ExceptionFilters. These implement the IExceptionFilter interface and they execute if there are any unhandled exceptions thrown during the execution pipeline. These filters can be used as an exception filter to handle errors raised by either your controller actions or controller action results.The OnException method is called when an unhandled exception arises. The parameter for this method is an ExceptionContext object, which is derived from ControllerContext and provides a number of useful properties that you can use to get information about the request

Entity Framework
Entity Framework was first released in 2008, Microsoft's primary means of interacting between .NET applications and relational databases. Entity Framework is an Object Relational Mapper (ORM) which is a type of tool that simplifies mapping between objects in your software to the tables and columns of a relational database.
An ORM takes care of **creating database connections** and **executing commands**, as well as taking query results and automatically materializing those results as yourapplication objects.

Advantages Of Entity FrameWork:

- **Cross-platform:** EF Core is a cross-platform framework which can run on Windows, Linux and Mac.
- **Modelling:** EF (Entity Framework) creates an EDM (Entity Data Model) based on POCO (Plain Old CLR Object) entities with **get/set properties** of different data types. It uses this model when querying or saving entity data to the underlying database.

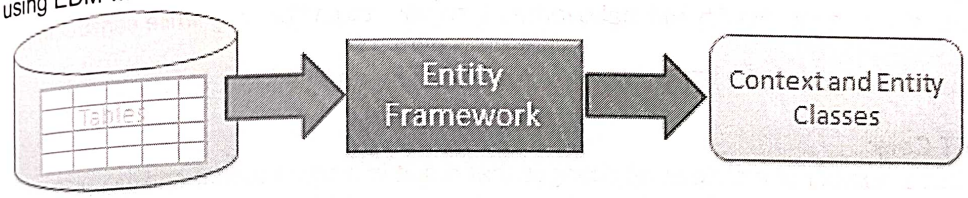
- **Change Tracking:** EF keeps track of changes occurred to instances of your entities (Property values) which need to be submitted to the database.
- **Saving:** EF executes INSERT, UPDATE, and DELETE commands to the database based on the changes occurred to your entities when you call the SaveChanges() method. EF also provides the asynchronous SaveChangesAsync() method.
- **Concurrency:** EF uses Optimistic Concurrency by default to protect overwriting changes made by another user since data was fetched from the database.
- **Transactions:** EF performs automatic transaction management while querying or saving data. It also provides options to customize transaction management.
- **Caching:** EF includes first level of caching out of the box. So, repeated querying will return data from the cache instead of hitting the database.
- **Built-in Conventions:** EF follows conventions over the configuration programming pattern, and includes a set of default rules which automatically configure the EF model.
- **Configurations:** EF allows us to configure the EF model by using data annotation attributes or Fluent API to override default conventions.
- **Migrations:** EF provides a set of migration commands that can be executed on the NuGet Package Manager Console or the Command Line Interface to create or manage underlying database Schema.

Different Approaches in Entity Framework:

1) Database First
2) Model First
3) Code First

Database-First Approach:

In the database-first development approach, you generate the context and entities for the existing database using EDM wizard integrated in Visual Studio or executing EF commands.

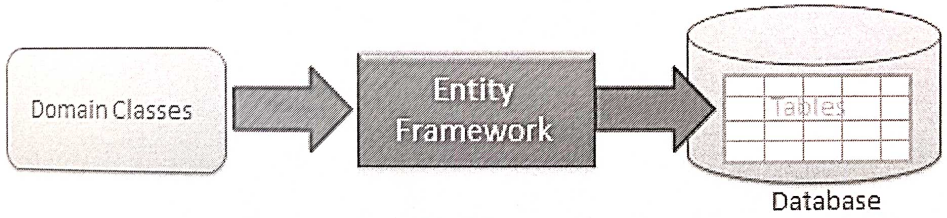


Database-First Approach

Code-First Approach:

Use this approach when you do not have an existing database for your application. In the code-first approach, you start writing your entities (domain classes) and context class first and then create the database from these classes using migration commands.

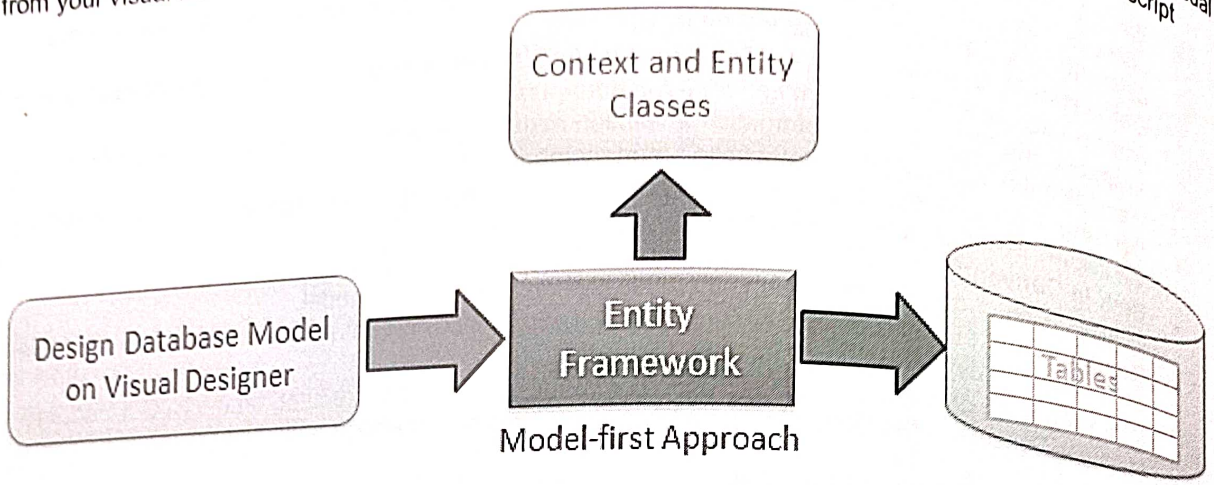
Developers who follow the Domain-Driven Design (DDD) principles, prefer to begin with coding their domain classes first and then generate the database required to persist their data.



Code-First Approach

Model-First Approach:

In the model-first approach, you create entities, relationships, and inheritance hierarchies directly on the visual designer integrated in Visual Studio and then generate entities, the context class, and the database script from your visual model.



What is ASP.NET Core:

ASP.NET Core is an open source and cloud-optimized web framework for developing modern web applications that can be developed and run on Windows, Linux and the Mac. It includes the MVC framework, which now combines the features of MVC and Web API into a single web programming framework.

- ASP.NET Core apps can run on .NET Core or on the full .NET Framework.
- It was architected to provide an optimized development framework for apps that are deployed to the cloud or run on-premises.
- It consists of modular components with minimal overhead, so you retain flexibility while constructing your solutions.
- You can develop and run your ASP.NET Core apps cross-platform on Windows, Mac and Linux.

Advantages of ASP.NET Core:

- ASP.NET Core has a number of architectural changes that result in a much leaner and modular framework.
- ASP.NET Core is no longer based on System.Web.dll. It is based on a set of granular and well factored NuGet packages.
- This allows you to optimize your app to include just the NuGet packages you need.
- The benefits of a smaller app surface area include tighter security, reduced servicing, improved performance, and decreased costs