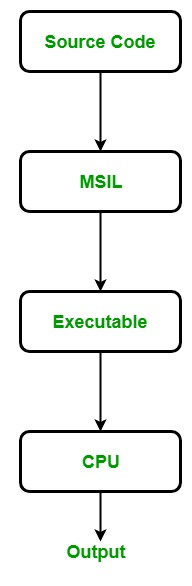
1) **Differences between var and dynamic keyword in C#:**

|  |  |
| --- | --- |
| Var | Dynamic |
| It is introduced in C# 3.0. | It is introduced in C# 4.0 |
| The variables are declared using var keyword are statically typed. | The variables are declared using dynamic keyword are dynamically typed. |
| The type of the variable is decided by the compiler at compile time. | The type of the variable is decided by the compiler at run time. |
| The variable of this type should be initialized at the time of declaration. So that the compiler will decide the type of the variable according to the value it initialized. | The variable of this type need not be initialized at the time of declaration. Because the compiler does not know the type of the variable at compile time. |
| If the variable does not initialize it throw an error. | If the variable does not initialize it will not throw an error. |
| It supports intelliSense in visual studio. | It does not support intelliSense in visual studio |
| var myvalue = 10; // statement 1 myvalue = “GeeksforGeeks”; // statement 2 Here the compiler will throw an error because the compiler has already decided the type of the myvalue variable using statement 1 that is an integer type. When you try to assign a string to myvalue variable, then the compiler will give an error because it violating safety rule type. | dynamic myvalue = 10; // statement 1 myvalue = “GeeksforGeeks”; // statement 2 Here, the compiler will not throw an error though the type of the myvalue is an integer. When you assign a string to myvalue it recreates the type of the myvalue and accepts string without any error. |
| It cannot be used for properties or returning values from the function. It can only used as a local variable in function. | It can be used for properties or returning values from the function. |

# 2) **Managed code and Unmanaged code in .NET.**

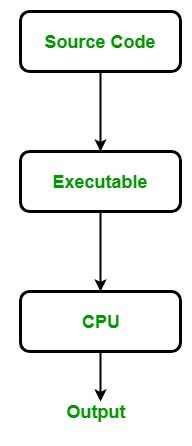
MSIL :- Microsoft Intermediate Language



* It improves the security of the application like when you use runtime environment, it automatically checks the memory buffers to guard against buffer overflow.
* It implement the garbage collection automatically.
* It also provides runtime type checking/dynamic type checking.
* It also provides reference checking which means it checks whether the reference point to the valid object or not and also check they are not duplicate.

**What are the disadvantages of Managed Code?**

The main disadvantage of managed language is that you are not allowed to allocate memory directly, or you cannot get the low-level access of the CPU architecture.



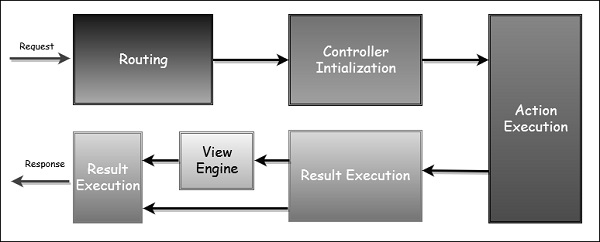
**What are the advantages of using Unmanaged Code?**

* It provides the low-level access to the programmer.
* It also provides direct access to the hardware.
* It allows the programmer to bypass some parameters and restriction that are used by the managed code framework.

**What are the disadvantages of Unmanaged Code?**

* It does not provide security to the application.
* Due to the access to memory allocation the issues related to memory occur like memory buffer overflow, etc.
* Error and exceptions are also handled by the programmer.
* It does not focus on garbage collection.

# **ASP.NET MVC - Life Cycle**



**Seek, scan, lookup**

Let's consider that you have two tables. TableA and TableB. Both of these tables contains more than 1000 000 rows, and both has clustered indexes on Id column. TableB has also nonclustered index on code column. (Remember that your nonclustered index always pointing at pages of clustered one...)

**seek:**

Let's consider that you want only 1 record from TableA and your clustered index is on column Id. Query should seems like:

SELECT NameFROM TableAWHERE Id = 1

Your result contains fewer than 15% (it is something between 10-20, depends on situation) of your full data set... Sql Server performs index seek in this scenario. (optimizer has found a useful index to retrieve data)

**scan:**

For example your query needs more than 15% of data from TableA , then it is necessary scan whole index to satisfy query. Let's consider that TableB has TableA Id column as foreign key from TableA, and TableB contains all Ids from TableA. Query should seems like:

SELECT a.IdFROM TableA aJOIN TableB b ON a.Id = b.TableAId

Or just

SELECT \*FROM TableA

For index on TableA SQL Server performs use index scan. Because all data (pages) needs satisfy query...

**lookup:**

Let's consider that TableB has column dim and also column code and nonclustered index on code (as we mentioned). SQL Server will use lookup when it needs to retrieve non key data from the data page and nonclustered index is used to resolve the query. For example key lookup could be used in query like:

SELECT id, dimFROM TableBWHERE code = 'codeX'

* You can resolve it by covering index (include dim to nonclustered one)

**1. Clustered Index :**   
Clustered index is created only when both the following conditions satisfy –

1. The data or file, that you are moving into secondary memory should be in sequential or sorted order.
2. There should be non key value, meaning it can have repeated values.

**2. Non-clustered Index :**   
Non-Clustered Index is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any topic or chapter then you can directly go to that page by using index of that book. No need to go through each and every page of a book.

The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

**Difference between Clustered and Non-clustered index :**

|  |  |
| --- | --- |
| CLUSTERED INDEX | NON-CLUSTERED INDEX |
| Clustered index is faster. | Non-clustered index is slower. |
| Clustered index requires less memory for operations. | Non-Clustered index requires more memory for operations. |
| In clustered index, index is the main data. | In Non-Clustered index, index is the copy of data. |
| A table can have only one clustered index. | A table can have multiple non-clustered index. |
| Clustered index has inherent ability of storing data on the disk. | Non-Clustered index does not have inherent ability of storing data on the disk. |
| Clustered index store pointers to block not data. | Non-Clustered index store both value and a pointer to actual row that holds data. |
| In Clustered index leaf nodes are actual data itself. | In Non-Clustered index leaf nodes are not the actual data itself rather they only contains included columns. |
| In Clustered index, Clustered key defines order of data within table. | In Non-Clustered index, index key defines order of data within index. |
| A Clustered index is a type of index in which table records are physically reordered to match the index. | A Non-Clustered index is a special type of index in which logical order of index does not match physical stored order of the rows on disk. |

# State Management In ASP.NET MVC

1. Hidden Field
2. Cookies
3. Query String
4. ViewData
5. ViewBag
6. TempData

**Differences between ViewState and SessionState:**

|  |  |
| --- | --- |
| ViewState | SessionState |
| Maintained at page level only. | Maintained at session level. |
| View state can only be visible from a single page and not multiple pages. | Session state value availability is across all pages available in a user session. |
| It will retain values in the event of a postback operation occurring. | In session state, user data remains in the server. Data is available to user until the browser is closed or there is session expiration. |
| Information is stored on the client’s end only. | Information is stored on the server. |
| used to allow the persistence of page-instance-specific data. | used for the persistence of user-specific data on the server’s end. |
| ViewState values are lost/cleared when new page is loaded. | SessionState can be cleared by programmer or user or in case of timeouts. |
|  |  |

**Usage:**

* **SessionState:** It can be used to store information that you wish to access on different web pages.
* **ViewState** It can be used to store information that you wish to access from same web page.

**Scaffolding**

ASP.NET Scaffolding is a code generation framework for ASP.NET Web applications.

You add scaffolding to your project when you want to quickly add code that interacts with data models. Using scaffolding can reduce the amount of time to develop standard data operations in your project.

**iqueryable and ienumerable difference**

* IEnumerable exists in System.Collections Namespace.
* IQueryable exists in System. Linq Namespace.
* Both IEnumerable and IQueryable are forward collection.
* IEnumerable doesn’t support lazy loading
* IQueryable support lazy loading
* Querying data from a database, IEnumerable execute a select query on the server side, load data in-memory on a client-side and then filter data.
* Querying data from a database, IQueryable execute the select query on the server side with all filters.

# **Difference Between dispose() and finalize() in C#**

### Comparison Chart

|  |  |  |
| --- | --- | --- |
| **BASIS FOR COMPARISON** | **DISPOSE( )** | **FINALIZE( )** |
| Defined | The method dispose( ) is defined in the interface IDisposable interface. | The method finalize( ) id defined in java.lang.object class. |
| Syntax | public void Dispose( ){ // Dispose code here } | protected void finalize( ){ // finalization code here } |
| Invoked | The method dispose( ) is invoked by the user. | The method finalize( ) is invoked by the garbage collector. |
| Purpose | Method dispose( ) is used to free unmanaged resources whenever it is invoked. | Method finalize( ) is used to free unmanaged resources before the object is destroyed. |
| Implementation | The method dispose( ) is to be implemented whenever there is a close( ) method. | The method finalize( ) is to be implemented for unmanaged resources. |
| Access specifier | The method dispose( ) is declared as public. | The method finalize( ) is declared as private. |
| Action | The method dispose( ) is faster and instantly disposes an object. | The method finalize is slower as compared to dispose |
| Performance | The method disposes( ) performs the instantaneous action hence, does not effect the performance of websites. | The method finalize( ) being slower affects the performance of the websites. |

## **SOLID Principles**

Single Responsibility Principle (SRP)

Open Closed Principle (OCP)

Liskov Substitution Principle (LSP)

Interface Segregation Principle (ISP)

Dependency Inversion Principle (DIP)

A class should have only one reason to change.

Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

We can check using LSP that inheritance is applied correctly or not in our code.

we can create separate interfaces for each operation or requirement rather than having a single class to do the same work.

High-level modules should not depend on low-level modules. Both should depend on abstractions.

**Can “this” be used within a static method c#**

No, we can not used "this" keyword within a static method. because "this" keyword refers to the current instance of the class. Static Member functions do not have a this pointer (current instance). Note - we can also not used "base" keyword within a static method.

# **Difference between Abstract Class and Interface in C#**

|  |  |
| --- | --- |
| Abstract Class | Interface |
| It contains both declaration and definition part. | It contains only a declaration part. |
| Multiple inheritance is not achieved by abstract class. | Multiple inheritance is achieved by interface. |
| It contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). | It does not contain [constructor](https://www.geeksforgeeks.org/c-sharp-constructors/). |
| It can contain static members. | It does not contain static members. |
| It can contain different types of access modifiers like public, private, protected etc. | It only contains public access modifier because everything in the interface is public. |
| The performance of an abstract class is fast. | The performance of interface is slow because it requires time to search actual method in the corresponding class. |
| It is used to implement the core identity of class. | It is used to implement peripheral abilities of class. |
| A class can only use one abstract class. | A class can use multiple interface. |
| If many implementations are of the same kind and use common behavior, then it is superior to use abstract class. | If many implementations only share methods, then it is superior to use Interface. |
| Abstract class can contain methods, fields, constants, etc. | Interface can only contain methods . |
| It can be fully, partially or not implemented. | It should be fully implemented. |

**Abstraction**

**Abstraction** is an important part of object oriented programming. It means that only the required information is visible to the user and the rest of the information is hidden. **Abstraction** can be implemented using abstract classes in **C#**. Abstract classes are base classes with partial implementation.

**Const**

* Compile-time constant
* Can't be declared static
* Can't be modified or changed
* Can be of any type of Access Modifier
* Local scope
* Needs to get initialized
* Declared at the time of declaration

**ReadOnly**

* Run-time constant
* It can be static
* Global scope
* Can be declared in the constructer class
* Generally public

**Static**

* Can't be used with indexers
* Works with constructors too
* By default it is private
* Can be parameterized or public too
* If its applied to a class then all the class members need to be static

**Access Modifiers**

|  |  |
| --- | --- |
| Modifier | Description |
| public | There are no restrictions on accessing public members. |
| private | Access is limited to within the class definition. This is the default access modifier type if none is formally specified |
| protected | Access is limited to within the class definition and any class that inherits from the class |
| internal | Access is limited exclusively to classes defined within the current project assembly |
| protected internal | Access is limited to the current assembly and types derived from the containing class. All members in current project and all members in derived class can access the variables. |
| private protected | Access is limited to the containing class or types derived from the containing class within the current assembly. |

**Virtual method**

The main objective of virtual method is that, you can redefine it in one or more derived classes. You can use virtual keyword with method, property, indexer, or event. It allows these members to be overridden in a derived class. The implementation of a virtual member can be changed in a derived class by overriding these members (method, property, indexer, event). The process of redefining a virtual method in a derived class is called method overriding. It is also called as runtime polymorphism, late binding or dynamic binding. When overriding a method, the name, return type, and signature of the overriding method should be the same as the virtual method.

**Generics**

* It helps you in code reuse, performance and type safety.
* You can create your own generic classes, methods, interfaces and delegates.
* You can create generic collection classes. The .NET framework class library contains many new generic collection classes in System.Collections.Generic namespace.
* You can get information on the types used in generic data type at run-time.

**Reusability:** You can use a single generic type definition for multiple purposes in the same code without any alterations.

**Type Safety:** Generic data types provide better type safety, especially in the case of collections. When using generics you need to define the type of objects to be passed to a collection. This helps the compiler to ensure that only those object types that are defined in the definition can be passed to the collection.

**Performance:** Generic types provide better performance as compared to normal system types because they reduce the need for boxing, unboxing, and typecasting of variables or objects.

**Difference between Api controller and controller**

They work similarly in Web API, but controllers in Web API derive from the API controller class instead of Controller class.  
The first major difference you will notice is that actions on Web API controllers do not return views, they return serialized data.  
The MVC controller shows URL examples matching the default route pattern of “{controller}/{action}/{id}”.  
The Web API controller shows URL examples matching it’s default route pattern of “api/{controller}/{id}”.  
In Web API controller no need to convert data to json format.  
Use Controller to render your normal views. API controller action only return data that is serialized and sent to the client.

**methods involved in global aspx c#**

**Application\_Init:** Fired when an application initializes or is first called. It is invoked for all HttpApplication object instances.  
  
**Application\_Disposed:** Fired just before an application is destroyed. This is the ideal location for cleaning up previously used resources.  
  
**Application\_Error:** Fired when an unhandled exception is encountered within the application.  
  
**Application\_Start:** Fired when the first instance of the HttpApplication class is created. It allows you to create objects that are accessible by all HttpApplication instances.  
  
**Application\_End:** Fired when the last instance of an HttpApplication class is destroyed. It is fired only once during an application's lifetime.  
  
**Application\_BeginRequest**: Fired when an application request is received. It is the first event fired for a request, which is often a page request (URL) that a user enters.  
  
**Application\_EndRequest:** The last event fired for an application request.  
  
**Application\_PreRequestHandlerExecute**: Fired before the ASP.NET page framework begins executing an event handler like a page or Web service.  
  
**Application\_PostRequestHandlerExecute**: Fired when the ASP.NET page framework has finished executing an event handler.  
  
**Applcation\_PreSendRequestHeaders**: Fired before the ASP.NET page framework sends HTTP headers to a requesting client (browser).  
  
**Application\_PreSendContent:** Fired before the ASP.NET page framework send content to a requesting client (browser).  
  
**Application\_AcquireRequestState**: Fired when the ASP.NET page framework gets the current state (Session state) related to the current request.  
  
**Application\_ReleaseRequestState:** Fired when the ASP.NET page framework completes execution of all event handlers. This results in all state modules to save their current state data.  
  
**Application\_ResolveRequestCache:** Fired when the ASP.NET page framework completes an authorization request. It allows caching modules to serve the request from the cache, thus bypassing handler execution.  
  
**Application\_UpdateRequestCache:** Fired when the ASP.NET page framework completes handler execution to allow caching modules to store responses to be used to handle subsequent requests.  
  
**Application\_AuthenticateRequest:** Fired when the security module has established the current user's identity as valid. At this point, the user's credentials have been validated.  
  
**Application\_AuthorizeRequest:** Fired when the security module has verified that a user can access resources.  
  
**Session\_Start:** Fired when a new user visits the application Web site.

**Session\_End:** Fired when a user's session times out, ends, or they leave the application Web site.

**Bundling**

The bundle is a logical group of physical files, which loads in a single HTTP request. We have separate CSS files, which can be loaded in a single request with the help of bundling. The bundling also can create for JavaScript files separately. A bundle can’t contain both CSS and JavaScript files. We need to create a separate bundle for CSS and JavaScript files. We create a bundle, based on the use of CSS or JS files in the Application. For example, an Application uses both the bootstrap and site CSS for UI design, due to which we create a common bundle for them, such as a core bundle. The following figure shows a logical grouping of a file to create a bundle.

**BundleConfig.cs:**

This is used to create and register bundles for CS and JS files.By default various bundles are added in this file including jQuery,jQueryUI,jQuery validation,Modernizer and Site Css..

**FilterConfig.cs-**

This is used to create and register global MVC filter error filter,action filter etc.By default it contains HandleErrorAttribute filter.

**RouteConfig.cs-**

This is used to register various route patterns for your Asp.Net MVC application. By default,one route is registered here named as Default Route.

**WebApiConfig.cs-**

This is used to register various WEB API routes like as Asp.Net MVC,as well as set any additional WEB API configurations settings.

**Triggers**

**DML**

* Instead of Trigger: An Instead of trigger is fired instead of the triggering action such as an insert, update, or delete
* After Trigger: An After trigger executes following the triggering action, such as an insert, update or delete

**DDL Trigger**

This type of trigger is fired against DDL statements like Drop Table, Create Table or Alter Table. DDL Triggers are always After Triggers.

**Logon trigger**

This type of trigger is fired against a LOGON event before a user session is established to the SQL Server.

**Filters**

**Authentication Filters**

Authentication filter runs before any other filter or action method. Authentication confirms that you are a valid or invalid user. Action filters implement the IAuthenticationFilter interface.

**Authorization Filters**

The AuthorizeAttribute and RequireHttpsAttribute are 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.

**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 has 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.

**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 contain 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.

**Circular reference**

Circular reference occurs when two or more interdependent resources cause lock condition. This makes the resource unusable.

**Sealed class**

Sealed classes are used to restrict the users from inheriting the class. A class can be sealed by using the ***sealed*** keyword. The keyword tells the compiler that the class is sealed, and therefore, cannot be extended. No class can be derived from a sealed class.

### **String vs. StringBuilder**

* String
  + Under System namespace
  + Immutable (readonly) instance
  + Performance degrades when continuous change of value occurs
  + Thread-safe
* StringBuilder (mutable string)
  1. Under System.Text namespace
  2. Mutable instance
  3. Shows better performance since new changes are made to an existing instance

## **What is Authentication?**

Authentication is a process to ensure and confirms a user’s identity and whether the user is registered or not to access particular data or web pages. In other words, we can say that it is a process to validate someone against some data source.

## **What is Authorization?**

Authorization is a security mechanism which is used to determine whether the user has access to a particular resource or not. The main point that you need to remember is, authentication happens first, then only authorization.

# **Difference between JWT and OAuth**

Basically, JWT is a token format.

OAuth is an standardized **authorization** protocol that can use JWT as a token.

OAuth uses server-side and client-side storage. If you want to do real logout you must go with OAuth2. Authentication with JWT token can not logout actually. Because you don’t have an Authentication Server that keeps track of tokens.

If you want to provide an API to 3rd party clients, you must use OAuth2 also. OAuth2 is very flexible. JWT implementation is very easy and does not take long to implement. If your application needs this sort of flexibility, you should go with OAuth2. But if you don’t need this use-case scenario, implementing OAuth2 is a waste of time.

## **Anti-Forgery Tokens**

To help prevent CSRF attacks, ASP.NET MVC uses anti-forgery tokens, also called *request verification tokens*.

The client requests an HTML page that contains a form.

The server includes two tokens in the response. One token is sent as a cookie. The other is placed in a hidden form field. The tokens are generated randomly so that an adversary cannot guess the values.

When the client submits the form, it must send both tokens back to the server. The client sends the cookie token as a cookie, and it sends the form token inside the form data. (A browser client automatically does this when the user submits the form.)

If a request does not include both tokens, the server disallows the request.

# **Encapsulation**

Technically in encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of own class in which they are declared.

As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.

**Area**

Area allows us to partition the large application into smaller units where each unit contains a separate MVC folder structure, same as the default MVC folder structure.

**delegate**

The delegate is a reference type data type that defines the method signature. You can define variables of delegate, just like other data type, that can refer to any method with the same signature as the delegate.