## Collections

<https://www.codeproject.com/Articles/832189/List-vs-IEnumerable-vs-IQueryable-vs-ICollection-v>

## stack vs heap memory

## Delegate amd multi cast delegate

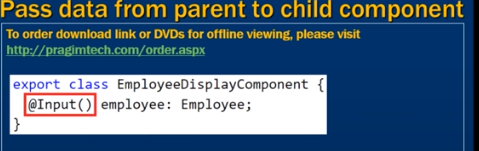
## Abtraction v/s encapsulation

## How to pass from parent to child component

<https://www.c-sharpcorner.com/blogs/update-the-child-component-and-parent-component-using-input-output-in-angular-4>

pass data from a parent component to a child directive or component using the @Input() decorator in the child component/directive.

To let Angular know that a property in a child component or directive can receive its value from its parent component we must use the @Input() decorator in the said child.



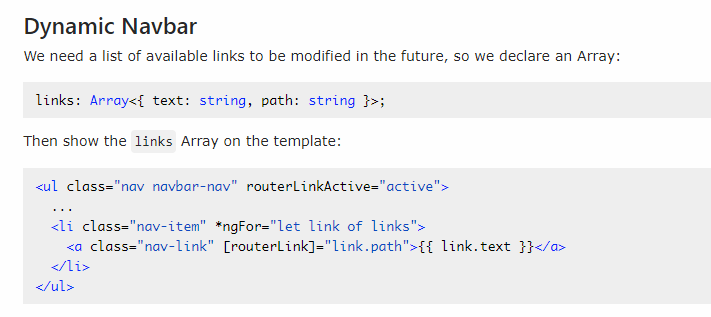
# Sharing Data from Child to Parent in Angular 8 Using @viewchild

<https://dzone.com/articles/sharing-data-from-child-to-parent-in-angular-8-usi>



## How to pass from parent to child grid

## How to dynamically implemnt naviagtion bar in angular



## What is docker and containers

## What is kubernates

## How to host web api and angular apps in azure

## Design Patterns

## Design pattern for toggle features

<https://www.c-sharpcorner.com/UploadFile/questpond/design-pattern-interview-questions/>

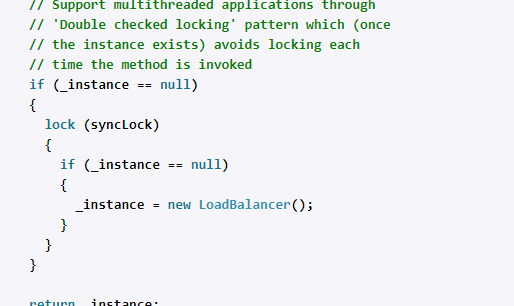
## Which are the three main categories of design patterns?

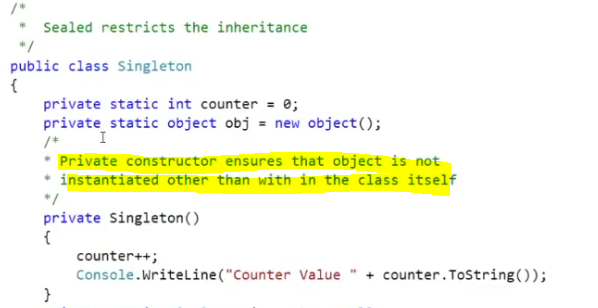
There are three basic classifications of patterns Creational, Structural, and Behavioral patterns.  
***Creational Patterns***

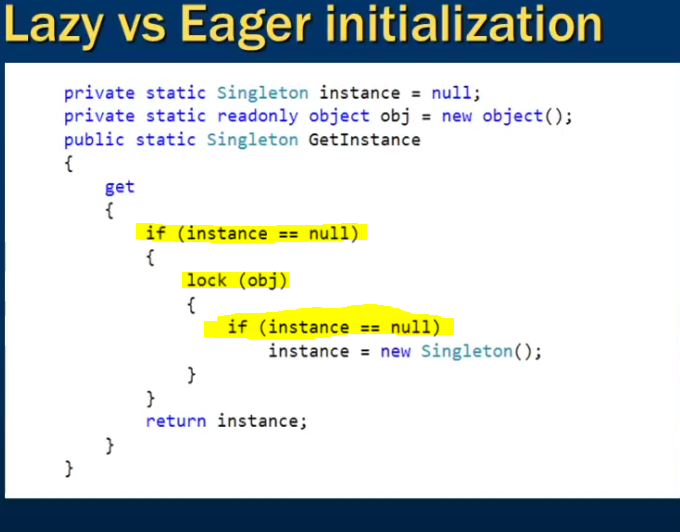
Factory Method : Creates an instance of several derived classes  
Abstract Factory : Creates an instance of several families of classes/superset of factory pattren  
Builder : Separates object construction from its representation  
Prototype : A fully initialized instance to be copied or cloned  
Singleton : A class in which only a single instance can exist  
 **Note**   
  
The best way to remember Creational pattern is by remembering ABFPS (Abraham Became First President of States).  
  
***Structural Patterns***  
Adapter : Match interfaces of different classes .  
Bridge : Separates an object's abstraction from its implementation.  
Composite : A tree structure of simple and composite objects.  
Decorator : Add responsibilities to objects dynamically.  
Flyweight : A fine-grained instance used for efficient sharing.  
Proxy : An object representing another object.  
  
**Note**  
To remember structural pattern best is (ABCDFFP)  
  
***Behavioral Patterns***  
Mediator : Defines simplified communication between classes.  
Memento : Capture and restore an object's internal state.  
Interpreter : A way to include language elements in a program.  
Iterator : Sequentially access the elements of a collection.  
Chain of Resp : A way of passing a request between a chain of objects.  
Command : Encapsulate a command request as an object.  
State : Alter an object's behavior when its state changes.  
Strategy : Encapsulates an algorithm inside a class.  
Observer : A way of notifying change to a number of classes.  
Template Method : Defer the exact steps of an algorithm to a subclass.  
Visitor : Defines a new operation to a class without change.

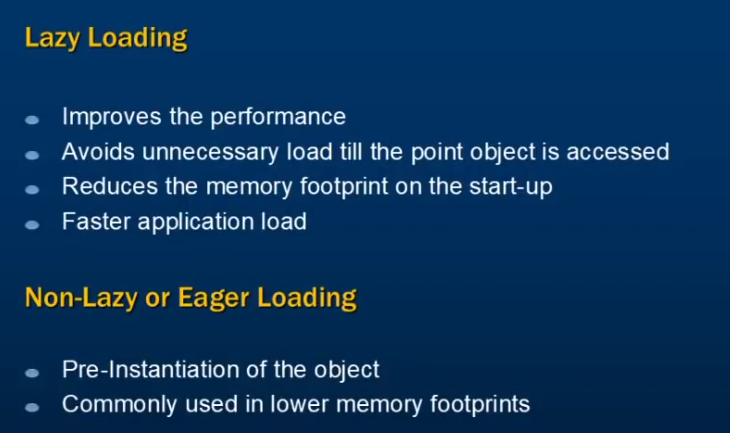
Singleton

<https://www.dofactory.com/net/singleton-design-pattern>

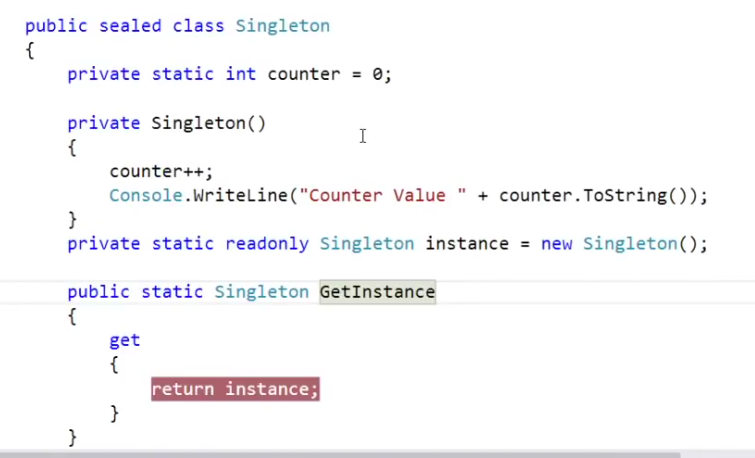


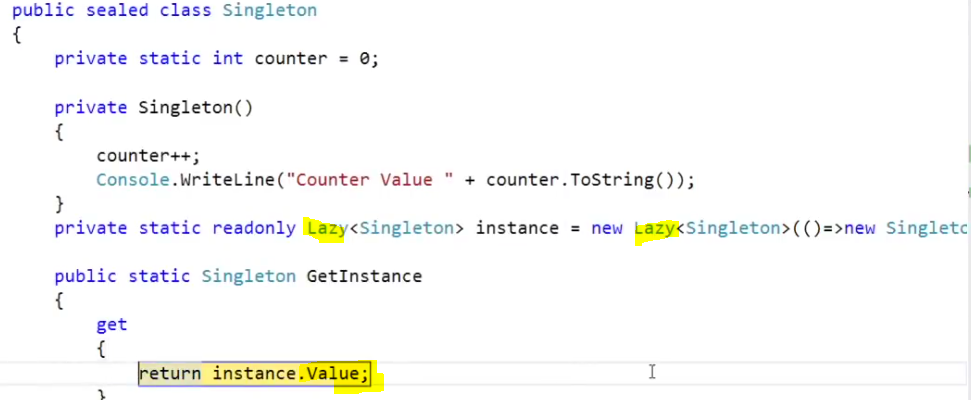




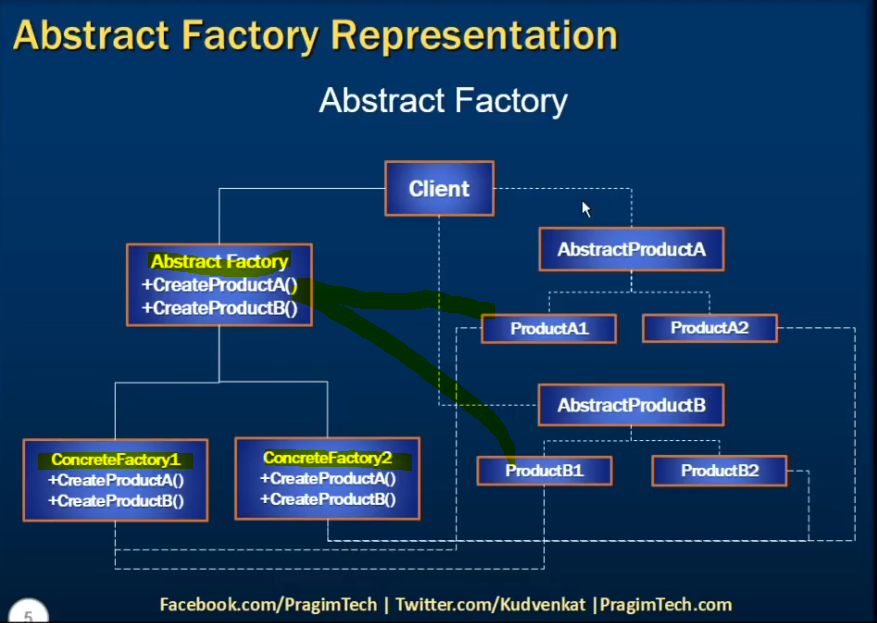


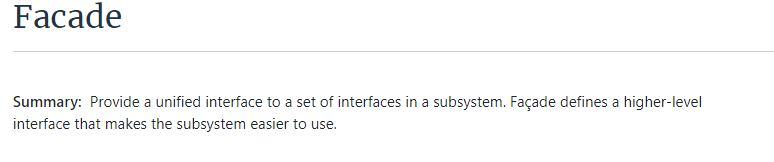
Non Lazy/Eager Loading











<https://www.dofactory.com/net/facade-design-pattern>

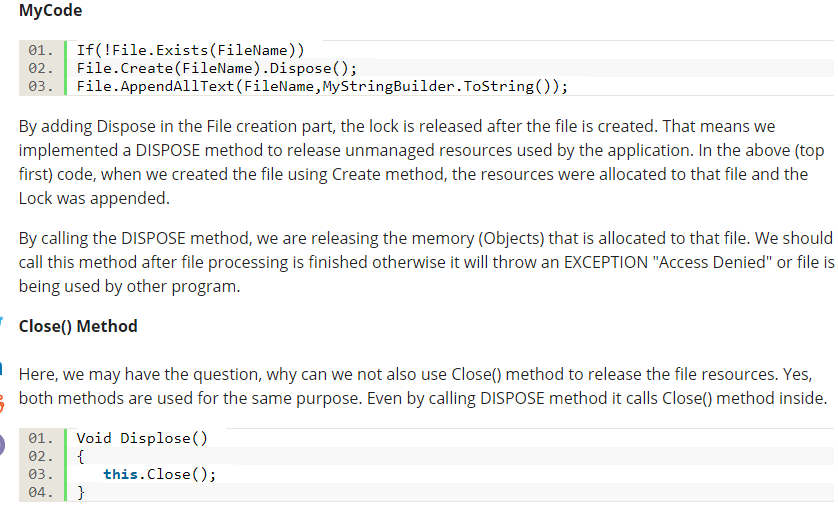
### Calling Store proc in EF

1. / Creating Custom class to hold result of Stored Procedure
2. **public** **class** EmployeeDetail
3. {
4. **public** **int** EmployeeID { get; set; }
5. **public** string EmployeeName { get; set; }
6. **public** string DepartmentName { get; set; }
7. }
9. // using Object Context (EF4.0)
10. using (Entities context = **new** Entities())
11. {
12. IEnumerable<EmployeeDetails> empDetails  =  context.ExecuteStoreQuery<EmployeeDetails>
13. ("exec GetEmployeeData").ToList();
14. }
16. // using DBContext (EF 4.1 and above)
17. using (Entities context = **new** Entities())
18. {
19. IEnumerable<EmployeeDetails> empDetails  =  context. Database.SqlQuery
20. < EmployeeDetails >("exec GetEmployeeData ", **null**).ToList();
21. }

<https://stackoverflow.com/questions/18690537/multi-processes-readwrite-one-file>



<https://www.c-sharpcorner.com/article/c-sharp-to-overcome-the-process-cannot-access-the-file-xyz-because-it-is-being-used/>



<https://bytes.com/topic/c-sharp/answers/493277-two-diffrent-process-tries-open-file-same-time>

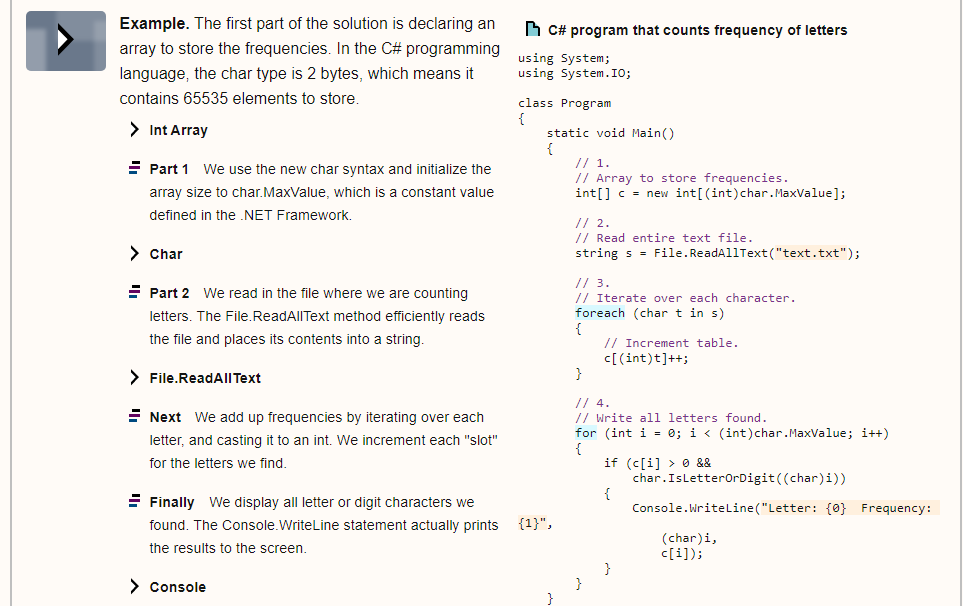
<https://www.tutorialspoint.com/csharp/csharp_arrays.htm>

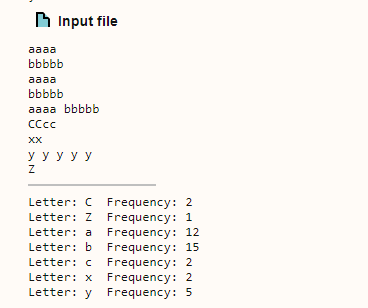
<https://www.tutorialspoint.com/csharp/csharp_jagged_arrays.htm>

**C# Count Letter Frequencies**

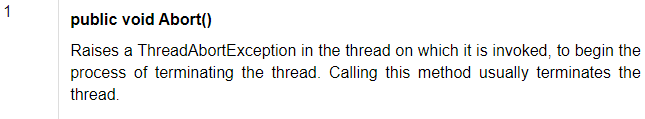
Implement a method that counts the frequencies of letters in text.

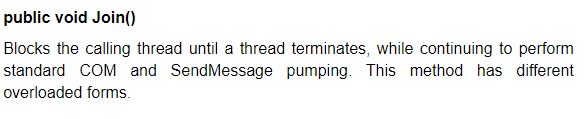
<https://www.dotnetperls.com/count-letter-frequencies>





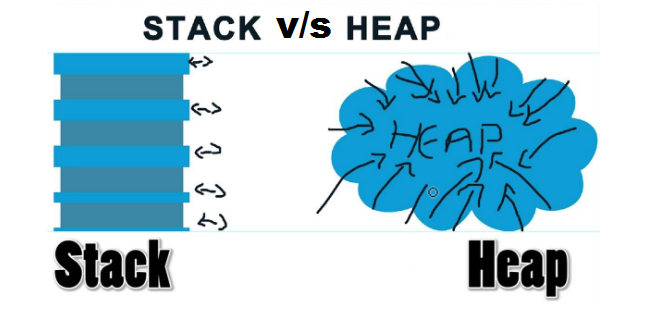
<https://www.tutorialspoint.com/csharp/csharp_delegates.htm>

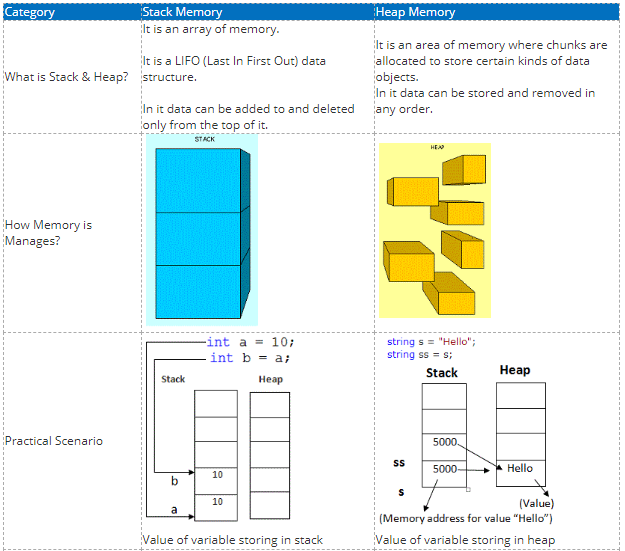


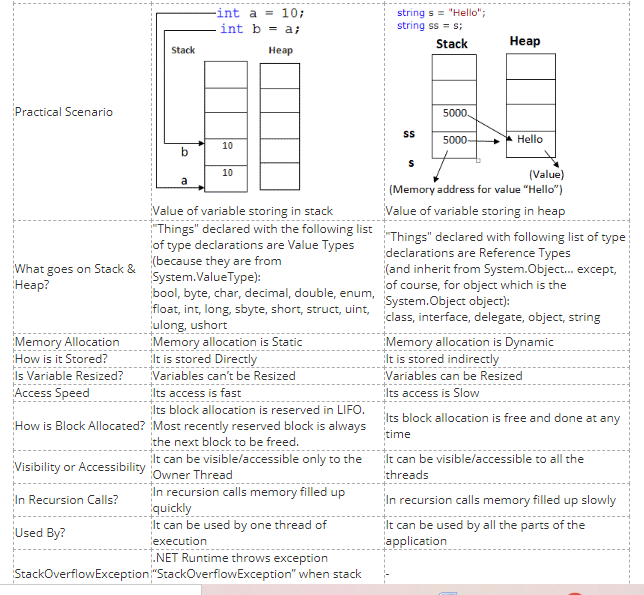


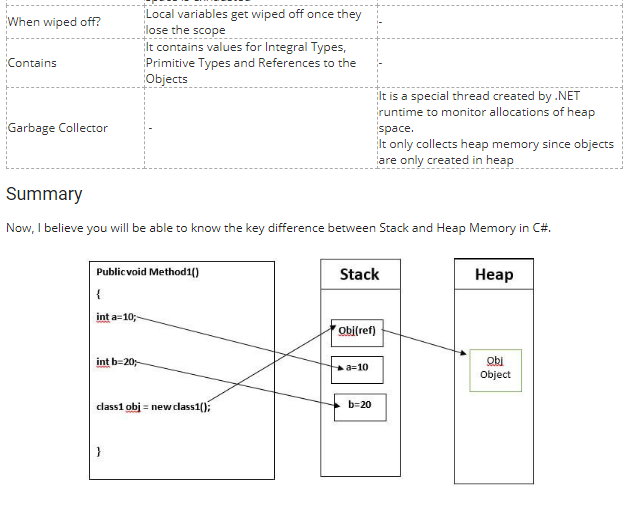
Compile time – Converts to execuable

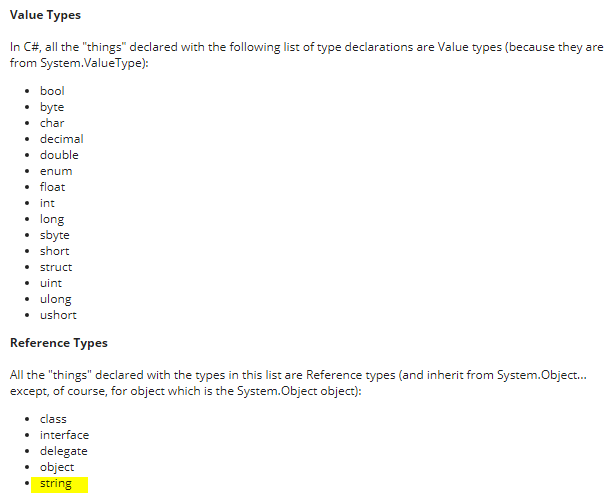
Run Time – Runs the executable









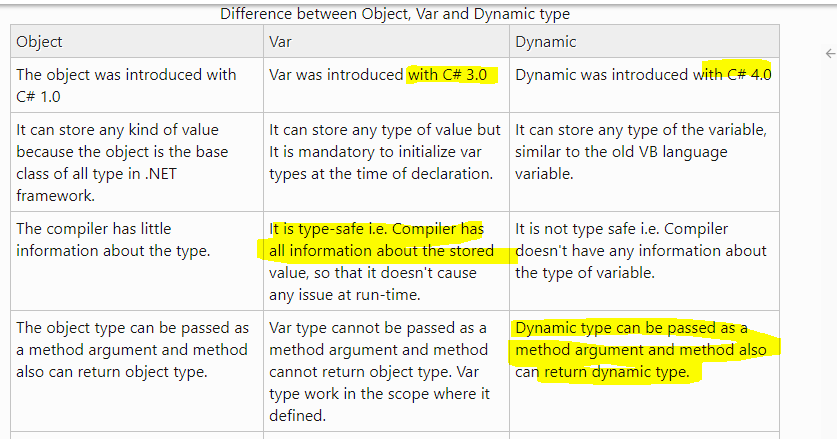


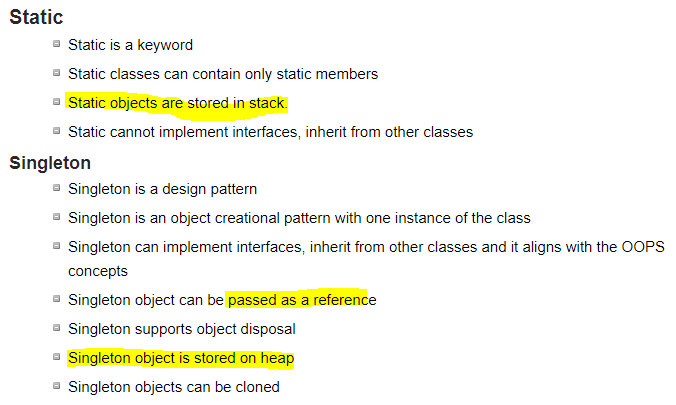
<https://www.c-sharpcorner.com/article/C-Sharp-heaping-vs-stacking-in-net-part-i/>

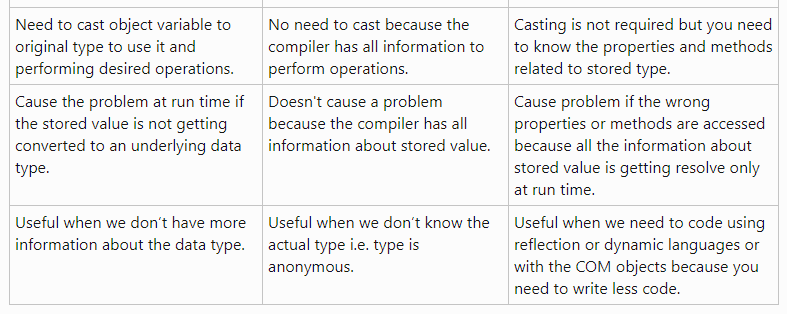
<https://www.tutorialspoint.com/Stack-and-Queue-in-Chash>

|  |  |
| --- | --- |
| Stacks | Queue |

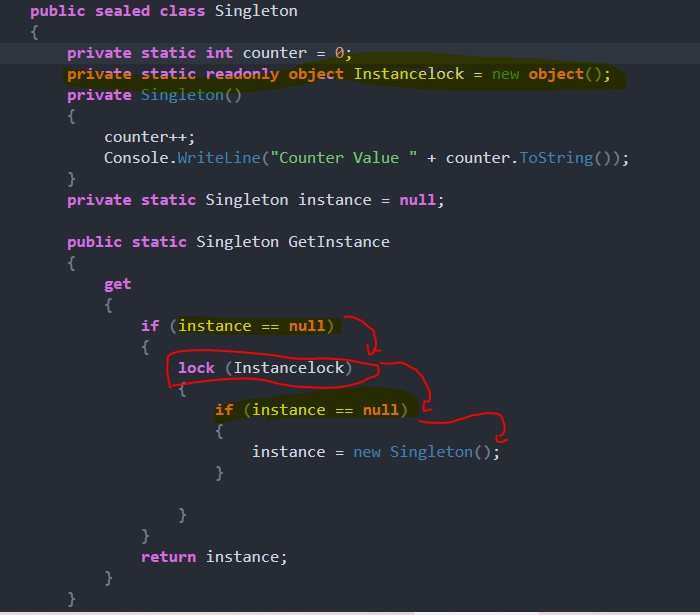
|  |  |
| --- | --- |
| Stacks | Queues |
| Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. | Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list. |
| Insertion and deletion in stacks takes place only from one end of the list called the top. | Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list. |
| Insert operation is called push operation. | Insert operation is called enqueue operation. |
| Delete operation is called pop operation. | Delete operation is called dequeue operation. |
| In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list. | In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element. |
| Stack is used in solving problems works on recursion. | Queue is used in solving problems having sequential processing. |



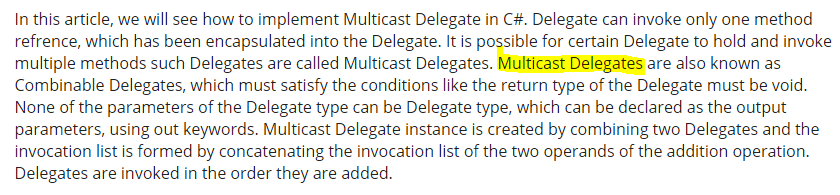


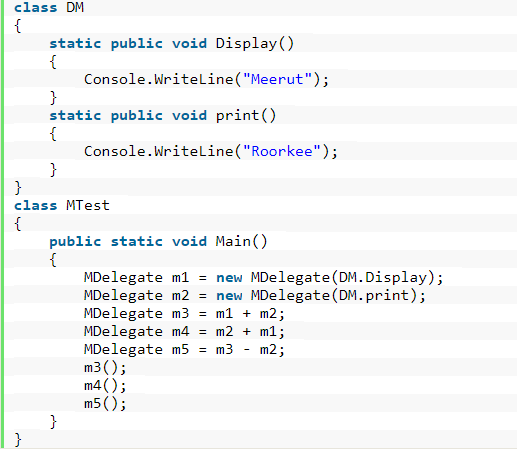


Thread Safe Singleton :

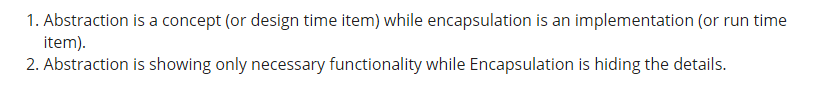


**Multi Cast Delegate**

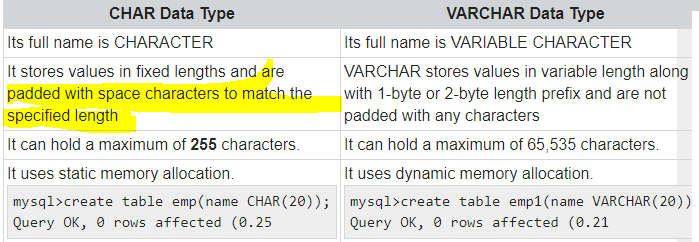




Exception handling in C#



<https://www.c-sharpcorner.com/article/abstraction-vs-encapsulation-in-oops/>



## \*\*\*Differences between a task and a thread.

1. The Thread class is used for creating and manipulating a [thread](http://msdn.microsoft.com/en-us/library/windows/desktop/ms684841%28v=vs.85%29.aspx) in Windows. A [Task](http://msdn.microsoft.com/en-us/library/vstudio/system.threading.tasks.task) represents some asynchronous operation and is part of the [Task Parallel Library](http://msdn.microsoft.com/en-us/library/dd460717%28v=vs.110%29.aspx), a set of APIs for running tasks asynchronously and in parallel.
2. The task can return a result. There is no direct mechanism to return the result from a thread.
3. Task supports cancellation through the use of cancellation tokens. But Thread doesn't.
4. ***A task can have multiple processes happening at the same time. Threads can only have one task running at a time.***
5. We can easily implement Asynchronous using ’async’ and ‘await’ keywords.
6. A new Thread()is not dealing with Thread pool thread, whereas Task does use thread pool thread.
7. A Task is a higher level concept than Thread.

*Is thread uses multicore ? no, its one task running at a time with context swtching*

IN testing /TDD considerring Dependency Injection in class to be tested is very important factor

## Sql peformance Improvements

<https://www.c-sharpcorner.com/UploadFile/ff2f08/tips-to-improve-sql-database-performance/>

<https://www.c-sharpcorner.com/UploadFile/f0b2ed/tips-to-increase-sql-server-query-performance-part-1/>

<https://www.c-sharpcorner.com/UploadFile/f0b2ed/tips-to-increase-sql-server-query-performance-part-2/>

## Preventing Sql Injection

1. Implement strong server side validation for all user inputs including cookie values.
2. Escape or filter the special characters in user inputs.
3. Use store procedures whenever possible.
4. Use parameterized queries or ORM-Entity Framework.
5. Avoid building SQL statements either in a class file or inside a procedure.
6. Avoid using exec command in SQL Server.
7. Avoid using sa account to connect database from the application.
8. Use low privileged account to execute queries.
9. Configure generic error page for the application and don’t display error information to user.
10. Catch all possible exceptions, implement global exception handler.

https://www.codeproject.com/Tips/706692/Preventing-SQL-Injection-Attacks-2

## ASP.NET performance improvement

<https://stackify.com/asp-net-performance-tuning/>

## Angular Performance Improvement

<https://www.simform.com/angular-performance/>

<https://medium.com/swlh/angular-performance-optimization-techniques-5b7ca0808f8b>

## token expiration and refresh web api angular 8

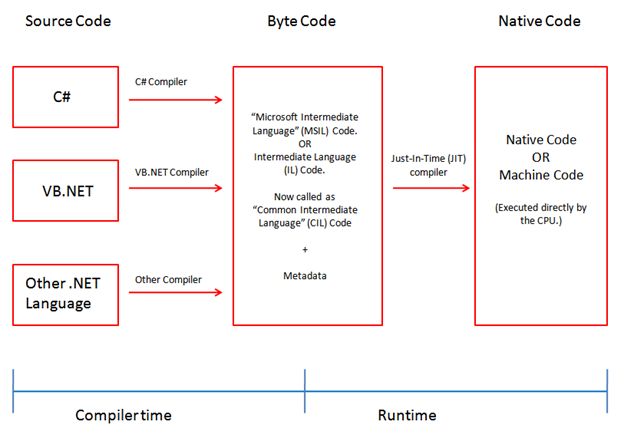
<https://jasonwatmore.com/post/2020/05/22/angular-9-jwt-authentication-with-refresh-tokens>

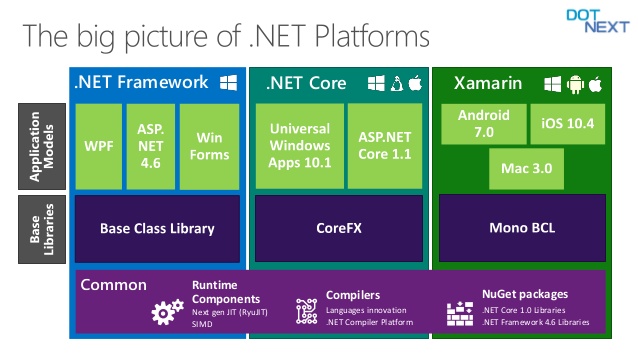
<https://www.codemag.com/article/1809031/Security-in-Angular-Part-2>

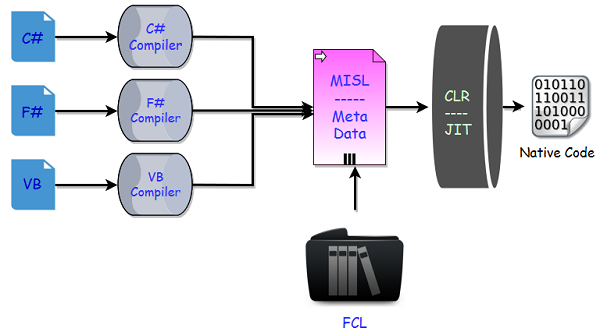
## MVC.net core life cycle

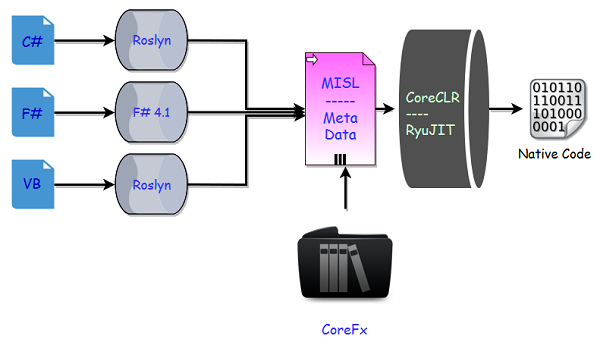
<https://www.c-sharpcorner.com/article/asp-net-core-mvc-request-life-cycle/>

.net execution





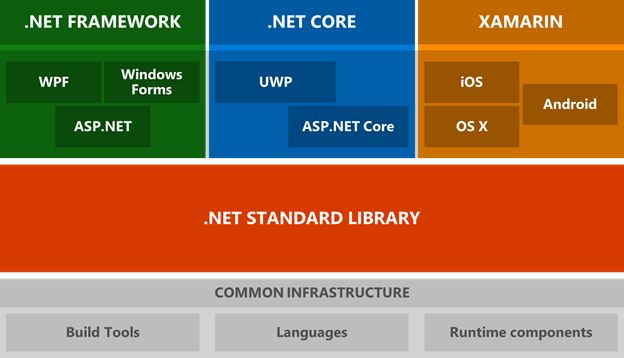




* In .NET Core now we have a new series of compilers, like we have **Roslyn** for C# and VB.
* You can also make use of the new F# 4.1 compiler if you want to use F# with .NET Core.
* Actually these tools are different and we can use Roslyn with .NET Framework as well if we are using C# 6 or later, because C# compiler can only support up to C# 5.
* In .NET Core, we don’t have a framework class libraries (FCL), so a different set of libraries are used and we now have **CoreFx**.
* CoreFx is the reimplementation of the class libraries for .NET Core.
* We also have a new run time with .NET Core known as CoreCLR and leverages a JIT Compiler.
* Now the question is why do we have the reimplementation of all these components that we already have in .NET framework.
* So the answer is the same as why Microsoft implemented .NET Core

.net core v/s .net framework

<https://www.c-sharpcorner.com/article/difference-between-net-framework-and-net-core/>



## The .NET Framework

supports Windows and Web applications. Today, you can use Windows Forms, WPF, and UWP to build Windows applications in .NET Framework. ASP.NET MVC is used to build Web applications in .NET Framework.

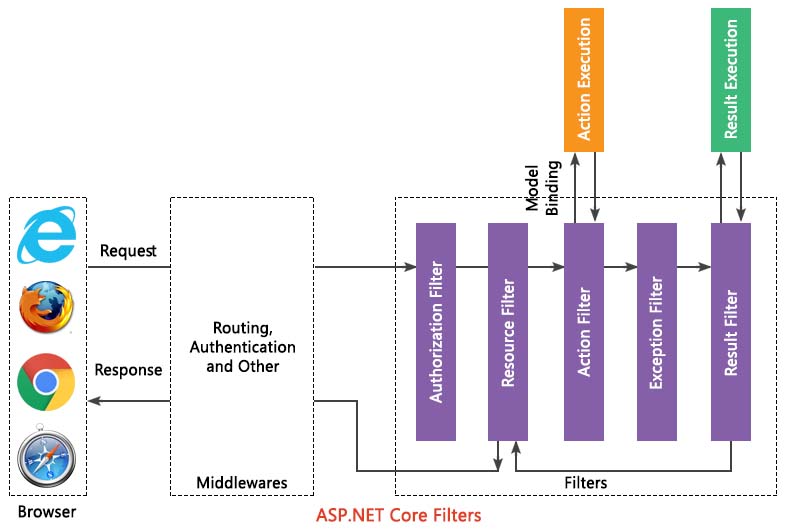
## .NET Core

is the new open-source and cross-platform framework to build applications for all operating systems including Windows, Mac, and Linux. .NET Core supports UWP and ASP.NET Core only. UWP is used to build Windows 10 targets Windows and mobile applications. ASP.NET Core is used to build browser-based web applications.

The following table may help you make your decision.

|  |  |
| --- | --- |
| A high-performance and scalable system without UI | .NET Core is much faster. |
| Docker containers support | Both, but .NET Core is born to live in a container. |
| Heavily rely on the command line | .NET Core has better support. |
| Cross-platform needs | .NET Core |
| Using Microservices | Both, but .NET Core is designed to keep today's needs in mind. |
| User interface centric Web applications | .NET Framework is better now until .NET Core catches up. |
| Windows client applications using Windows Forms and WPF | .NET Framework |
| Already have a pre-configured environment and systems | .NET Framework is better. |
| Stable version for an immediate need to build and deploy | .NET Framework has been around since 2001. .NET Core is just a baby. |
| Have existing experienced .NET team | .NET Core has a learning curve. |
| Time is not a problem. Experiments are acceptable. No rush to deployment. | .NET Core is the future of .NET. |

Web api filters



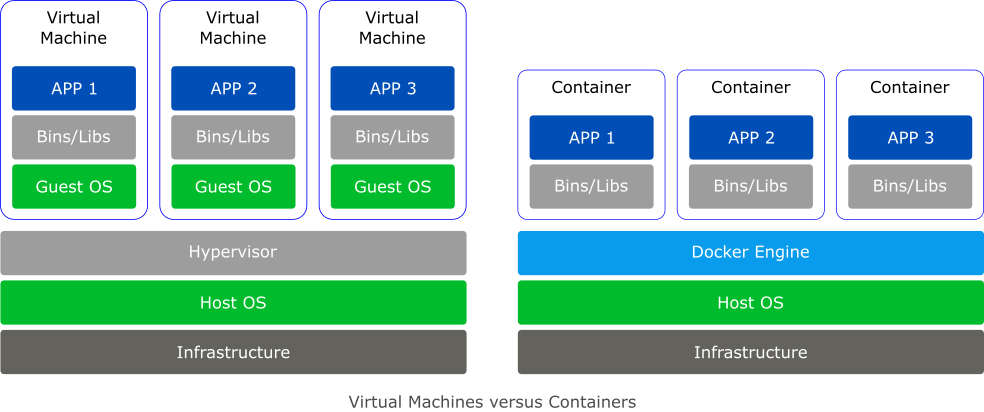
## [Deploying C# Web Applications with Docker](https://platform.uno/blog/deploying-c-web-applications-with-docker/)

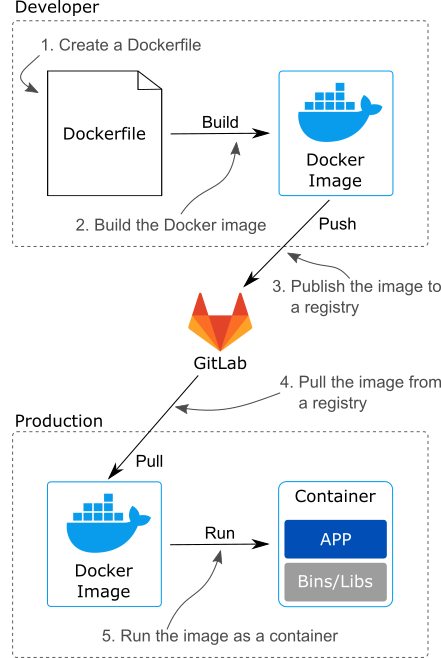
<https://platform.uno/blog/deploying-c-web-applications-with-docker/>

Docker is a type of container.

A container is a package that contains an application’s code and all of its dependencies, so that it can run quickly and reliably from one computing environment to the next.

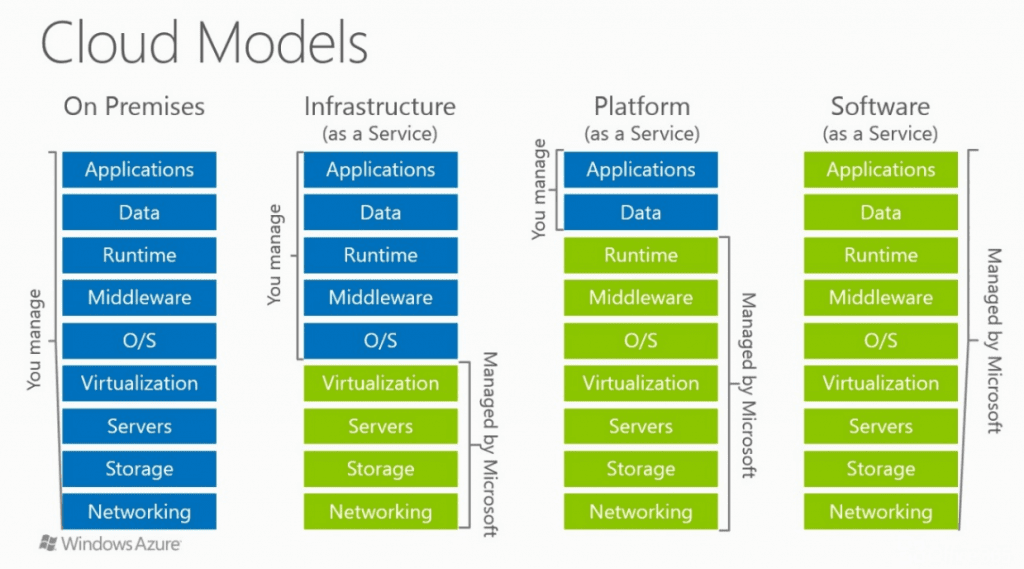
Like a virtual machine (VM), containers are isolated and act as if they have their own file system, CPU, and RAM. Unlike a VM, however, they don’t have an OS. Instead, a container shares the kernel of the host OS.

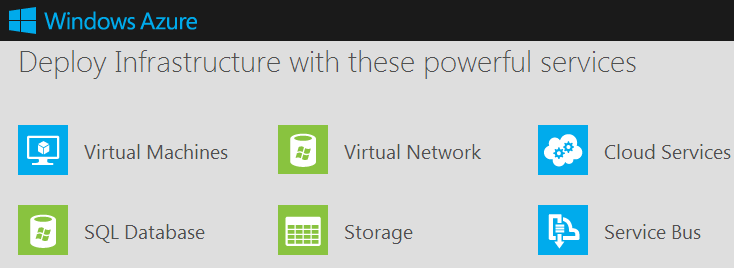


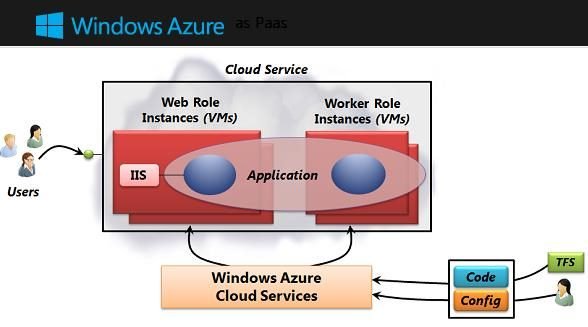


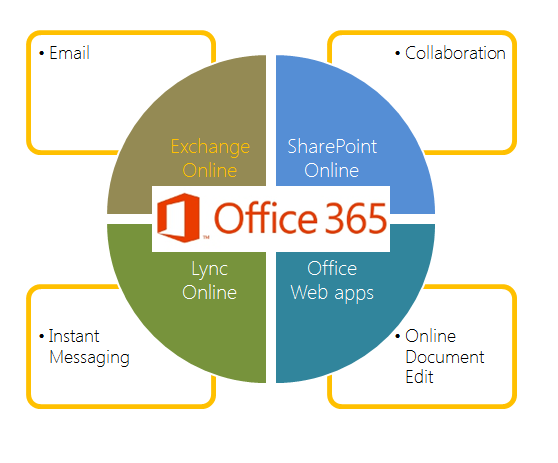
* Docker is a type of container. Containers are isolated packages that hold your code and dependencies so that they can run quickly and reliably from one computing environment to the next.
* Unlike a virtual machine, a container shares the kernel of the host OS allowing it to be smaller, faster, and use fewer resources.
* There are Linux and Windows Docker containers. To use Windows containers, you need to switch Docker Desktop to Windows containers. You also need to use Windows 10, Windows Server 2016, or Windows Server 2019 with either Hyper-V or WSL 2.
* Microsoft has four main base images (Windows Server Core, Nano Server, Windows, and Windows IoT Core) but they’ve also created several images derived from the four main ones including images with IIS, the .NET Framework, the .NET SDK, and the .NET Runtime.
* A Dockerfile is a special file that serves as a blueprint for creating a Docker image.
* A Docker image is a read-only template for creating a container. An image is built based on the instructions in a Dockerfile using the docker build command.
* The docker push command is used to push an image to a registry and docker pull is used to pull an image from a registry.

Cloud Service Models:









## how do microservices communicate with each other

efining the communication mechanism for the micro-services is the main challenge while developing the architecture design for an application. Micro-services design depends on business aspects of a system, however communication links is technical aspect for that particular system.  
  
There are two modes by which micro services use to communicate with each other which is based on the receiver configuration whether it is single or multiple:

**Synchronous**:  
In this approach, client sends a request and waits for the response from a particular service. Multiple services are communicating with each other through HTTP sync.  
  
**Asynchronous**:  
This kind of communication is established with the help of message broker queues which flows the messages flowing from one service to another service. It depends on the concept of message flowing through the services.

Message queues involved two components i.e Message produce and message consumer.

**Message producer:** It is responsible for sending out the requests and data and just waits for acknowledgement of data reception from consumer ones only.  
One to one communication is handled through queues and One to many communication is handled through Topics.  
  
Popular Message brokers used recently are RabbitMQ, Kafka etc. Software developers use these brokers as per their convenience.