**Master Reference**

This is the book for my personal reference when I learn something interesting and the things that will be valuable to learn. This document will have never ending updates. The goal is to update this document a.k.a book once in every day.

Contents

[OOPS 4](#_Toc27550939)

[C# Keywords and Types 5](#_Toc27550940)

[Multi Threading 6](#_Toc27550941)

[Design Patterns 7](#_Toc27550942)

[Data Structures 8](#_Toc27550943)

[Algorithms 9](#_Toc27550944)

[Collections 11](#_Toc27550945)

[General Things to know 13](#_Toc27550946)

[Buzz words 14](#_Toc27550947)

[Azure 15](#_Toc27550948)

**OOPS Basics**

**Primitive Data types**

**C# Keywords**

**Yield**

**C# necessary Features**

**Generics**

**Delegates**

**Events**

**Design Patterns/ Architecture**

**Data Structures**

**Garbage Collection**

**Collections**

**MultiThreading**

**Task Parallel Library**

**Async Await**

**General things to know**

**Buzz skills**

**Wishes**

**70 483 – C#**

**70 486**

**98-361 – Software Fundamentals**

**.NetCore**

**ASP.Net**

**REST API**

**WPF**

**Azure**

# OOPS

**Abstraction**

**Polymorphism**

**Encapsulation**

**Inheritance**

# C# Keywords and Types

**CallerInfo** Attribute: This attribute gives the information about the caller method like callermembername, callersourcepath.

# Multi Threading

**Concurrency vs Parellelism**

Async and await does the asynchonousy not concurrency

It does not spawn any threads

The code in async method will be run in state machine.

After async method is called the control goes back to caller method and execute the rest of the code.

Once the await method done executing it call back the main method through worker thread

Async method gives Task or Task<T> back.

For void Task return back Task.FromResult(null)

**Thread** : A small unit of program flow to process the work

**TPL**

Task parallel library spawns the threads in optimized manner. It user all the processors with the same load.

**TASK**

Task.Run creates a thread to run

Semaphore: It initializes how many minimum and maximimum threads can be spawned.

SemaphoreSlim : this is lighter version of semaphore class

# Design Patterns

**Creational**

* + 1. Factory
    2. Abstract Factory
    3. Builder
    4. singleton \*
    5. Prototype

**Structural**

* + 1. Adapter
    2. Bridge
    3. Composite
    4. Decorator
    5. Façade : Where you need to instantiate many instances in one call from the client application. Example may be the cacheaggregator.
    6. Proxy

**Behavioral**

* + 1. Command
    2. Interpreter
    3. Iterator
    4. Memento
    5. Observer \*:
    6. **Strategy\*** : This has set of algorithms and the object picks up the algorithm by its type IQueryService.
    7. Example: BhLibrary (where the connection type is supplied and the DBConnection base decides which one to connect) / CacheService/ IQueryService
    8. Template
    9. Visitor
    10. Chain of Responsibilities

Data Structures

**Abstract Data structures**

**Stacks : LIFO**

**Push**

**Pop**

**Peek**

**Queues : FIFO**

**Enqueue**

**DeQueue**

**Array**

**LinkedLists :** The node connects to the next element

**Double Linked List :** The left node connects to previous element and the right node connects to the next element

**Circular Double Linked List :** The left most node connects to the Right most element

**Hash Tables :** Hash tables holds key value pairs, It uses a hash function to compute an index into an array in which an element will be inserted or searched.

* **Collision Resolution**
* **Has tables are loosely typed which means it can have any types as values. Where as dictionary is strongly typed data structure/**

**Search Tree**

**BST (Binary Search Tree)**

**Graphs**

# Algorithms

Sorting

Search

**Linear : Find until you find it**

**Quadratic**

**Binary Search :** can only done on sorted array

# .NET core

.Net core is open source cross platform , cloud based framework to build modern apps

Abilities

1. High performance
2. Testability is included
3. Easy creation of pages using Razor
4. Blazor is included where we can have C# code in the browser
5. Built in dependency injection. No need to use Unity or Autofac
6. Ability to host on different servers . Kestrel, IIS, Nginx, Apache, Docker, HTTP.Sys
7. High performance and modular HTTP pipeline

When to use

1. If you are planning to host the application on multiple platforms like windows. Linux or macos
2. When your architecture targets to microservices architecture
3. Using Docker container
4. More importantly Microsoft is going to have .NET core as it standard framework.

Fundamentals

Startup.cs

Program.cs : This is where the .Net core will be spin up and make necessary configurations such as web host configurations

MiddleWare

Host

Dependency Injection

Options

Logging

.NetCore Routing

1. Conventional routing : In app.UseMvc in startup.cs , we will be configuring the routes. We typically do this in MVC web application
2. Attributr routing : app.UseMvc(). And then we configure the routes as attributes on each of the controller action methods.

# Collections

IList

What:

When to Use:

IEnumerable :

What:

When to Use: To modify when the collection is in memory

IQueryable

What: intended to be run against a “queryable” collection (ex. a database). Which means that, whenever we call methods on IQueryable , it calles Expression.Call() method. Expression.Call() method work as converting LINQ statements to SQL statements.

When to Use: To query any of the database collections

ICollections

What:

When to Use:

List

What: A collections with all the methods available

When to Use: For all CRUD operations

ReadOnlyDictionary

What:

When to Use:

ObservableCollection

What:

When to Use:

# General Things to know

**Throw:** throw the previous exception that bubbled from the last method and preserving the stack trace.

**Throw ex :** resets the stack trace and throw the exception that occurred in the current method.

**Unmanaged Code**: database connections, GDI+ objects, COM objects, and other system objects which basically operate with OS/ Kernel. As a developer we need to take care of cleaning up these. Whereas GC would take care of managed resources/ Code.

**Finalizers** : ~ClassName{} : Finalizers would be executed any point of time after the code exits the class. GC will clean up at some point of time and developers don’t know when. Avoid using finalizers because it is performance drag on the GC. Only use if you have to release very expensive resource.

<https://hownot2code.com/2016/10/07/why-using-finalizers-is-a-bad-idea/>

Static Variables always stored in Heap irrespective of where they have been declared.

Static classes can not be instantiated or not inherited. These are basically ‘Abstract Sealed’ classes. Static classes can be accessed by anywhere and can not be passed as parameter.

Limit type visibility when it is needed

Use interfaces than abstract classes. Interfaces are used when you need ‘Behave like’ and abstract classes are used ‘inherited from’

Regex Match Groups - DomainViewExtensions

Yield return

Intercepting interface when logging. Aceserver-> UnityConfig

Reverse proxy settings : Reverse proxy is an additional server that sits between Firewall and the web server to make another configuration needs.

Invariance

COVariance : Assigning more derived type to less derived type

IEnumerable<Animal> animals = new List<Dog>();

CONtravariance

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Add Unsubscribe to the event on CleanUp to avoid memory leaks

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Convert.Toint32 🡪 it will try to convert , if the parameter is null it will not throw argumentnull exception

Int.TryParse 🡪 It will try to convert the value and if it can not convert it will give ‘out’ value. However if the value is null , it throws exception.

# Buzz words

API Gateway

Topology Types

Docker Container

# Azure

**On Premises :** The servers will be hosted at the client location. This became expensive and maintenance cost is also increasing.

**IaaS**: Infrastructure as a service

This facilitates only the infrastructure like spinning up Virtual server and if the failover happens it spin up another server as needer

**PaaS**: Platform as a service

This is extensive to IaaS which means the cloud provider provides many services to the users that they need such as firewall, load balancing, Database , storage buckets, caching services etc