# C# and dotnet

## Keyword

### Sealed

### Where

Specifies constraints on the types that are used as arguments for type parameters in a generic type, method, delegate, or local function.

Sample code in Github

## Operator

**Msdn reference:** https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/

### Nameof

Converts name of member to string

**nameof(List<int>.Count) - Output is count**

### Typeof

*Type t =* ***typeof****(String);*

***MethodInfo*** *substr =* ***t.GetMethod****("Substring", new Type[] { typeof(int), typeof(int) });*

*Object result =* ***substr.Invoke****("Hello, World!", new Object[] { 7, 5 });*

*Console.WriteLine("{0} returned \"{1}\".", substr, result);*

//Output: System.String Substring(Int32, Int32) returned "World".

## Dependency Injection

Software design pattern and a technique used in object-oriented programming to achieve loose coupling between components and manage dependencies between classes. The process of obtaining the dependencies required by a class is moved out of the class itself, and the dependencies are provided from the outside (typically by a DI container) when the class is instantiated.

Example in Github: DependencyInjection folder

## Covariance and Contravariance

## Generics

Introduces concept of type paramters.

To create reusable, type-safe code by defining classes, interfaces, and methods that can work with different data types. Avoids the need for unnecessary type casting.

Signature:

***public class GenericList<T>***

***public interface IRepository<T, TKey>***

Applicable for interface and struct as well

Opposite of this is heterogeneous collection one of the example is array list

## LINQ - Language Integrated Query

Name for a set of technologies based on the integration of query capabilities directly into the C# language.

| **Clause** | **Description** |
| --- | --- |
| **[from](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/from-clause)** | Specifies a **data source** and a range variable (similar to an iteration variable). |
| **[where](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/where-clause)** | **Filters** source elements **based on** one or more **Boolean expressions** separated by logical AND and OR operators ( && or || ). |
| **[select](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/select-clause)** | **Specifies** the **type and shape** that the elements in the returned sequence will have when the query is executed. |
| **[group](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/group-clause)** | **Groups** query results **according to a specified key value**. |
| **[into](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/into)** | Provides an identifier that can serve as a reference to the results of a join, group or select clause. |
| **[orderby](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/orderby-clause)** | Sorts query results in ascending or descending order based on the default comparer for the element type. |
| **[join](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/join-clause)** | **Joins two data sources** based on an **equality comparison** between two specified matching criteria. |
| **[let](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/let-clause)** | Introduces a range variable to store sub-expression results in a query expression. |
| **[in](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/in)** | Contextual keyword in a [join](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/join-clause) clause. |
| **[on](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/on)** | Contextual keyword in a [join](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/join-clause) clause. |
| **[equals](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/equals)** | Contextual keyword in a [join](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/join-clause) clause. |
| **[by](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/by)** | Contextual keyword in a [group](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/group-clause) clause. |
| **[ascending](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/ascending)** | Contextual keyword in an [orderby](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/orderby-clause) clause. |
| **[descending](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/descending)** | Contextual keyword in an [orderby](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/orderby-clause) clause. |

## Method to suppress code analysis warning

try { ... }

catch (Exception e)

{

**#pragma warning disable** CA2200 // Rethrow to preserve stack details

throw e;

**#pragma warning restore** CA2200 // Rethrow to preserve stack details

}

## Memory

### Stack vs Heap Memory

### Dotmemory

Memory profiler tool by jet brains.



Unmanaged memory

Heap generation 0,1,2

Large object heap

## Unit testing

**Msdn reference:** https://learn.microsoft.com/en-us/dotnet/core/testing/

**Arrange, Act, Assert** is a common pattern when unit testing. As the name implies, it consists of three main actions:

* Arrange your objects, create and set them up as necessary.
* Act on an object.
* Assert that something is as expected.

***[Fact]***

*public void Add\_EmptyString\_ReturnsZero()*

*{*

*// Arrange*

*var stringCalculator = new StringCalculator();*

*// Act*

*var actual = stringCalculator.Add("");*

*// Assert*

*Assert.Equal(0, actual);*

*}*

### Attributes

**[Fact]:** Used to mark a method that contains a test

**[Theory]:** Used in conjunction with [InlineData], [ClassData], or [MemberData] to run a test multiple times with different input values.

**[InlineData]:** Specify the parameters of a test.

**[ClassData]:** Specify a class that returns the parameters of a test.

**[MemberData]:** Specify a property, field, or method that returns the parameters of a test.

**[Trait]:** Used to categorize tests, allowing you to filter which tests are run.

**[Collection]:** Used to indicate that a test class is part of a collection that shares a test context.

**[CollectionDefinition]:** Used to define a collection of tests that can share a context.

**[Output]:** Used to capture and output additional information during test execution. This requires an instance of ITestOutputHelper to be passed into the test class's constructor.

**[Skip]:** An argument you can add to the [Fact] and [Theory] attributes to skip the execution of a particular test, usually with a reason.

**[BeforeAfterTest]:** An attribute that can be applied to a class, allowing actions to be performed before and after each test within the class.

Sample code in github

### xUnit

### NUnit

### MSTest

## Misc and questions

### Can we have multiple awaits in asynchronous methods? Yes