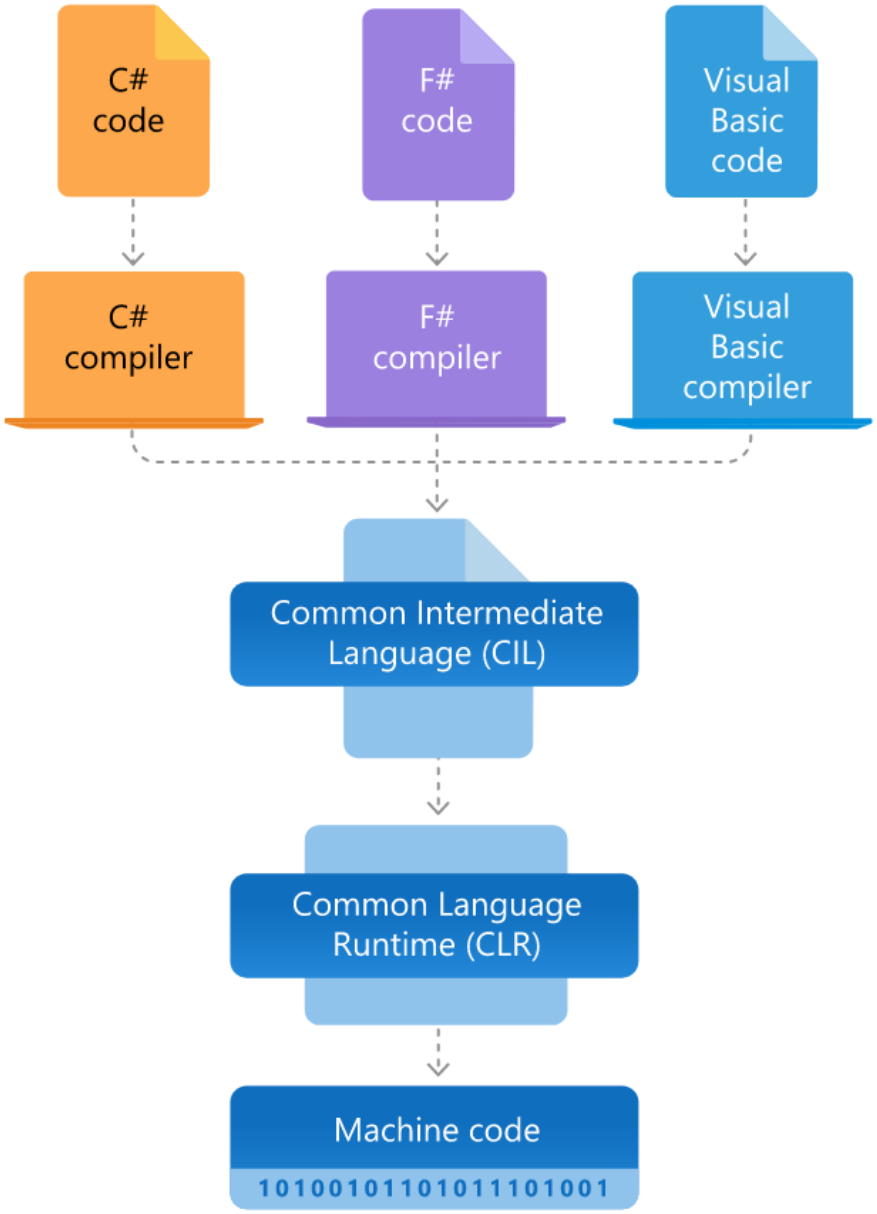
Day 1 : .NET & C# Basics

.NET Framework

- A runtime execution environment to manage apps targeting the framework
- Original implementation of .NET
- Supports running websites, services, desktop apps on Windows platforms only
- Includes two major components
  - Common Language Runtime(CLR)
  - Base Class Library

- Common Type System
- Common Language Specification
- Current version – 4.8

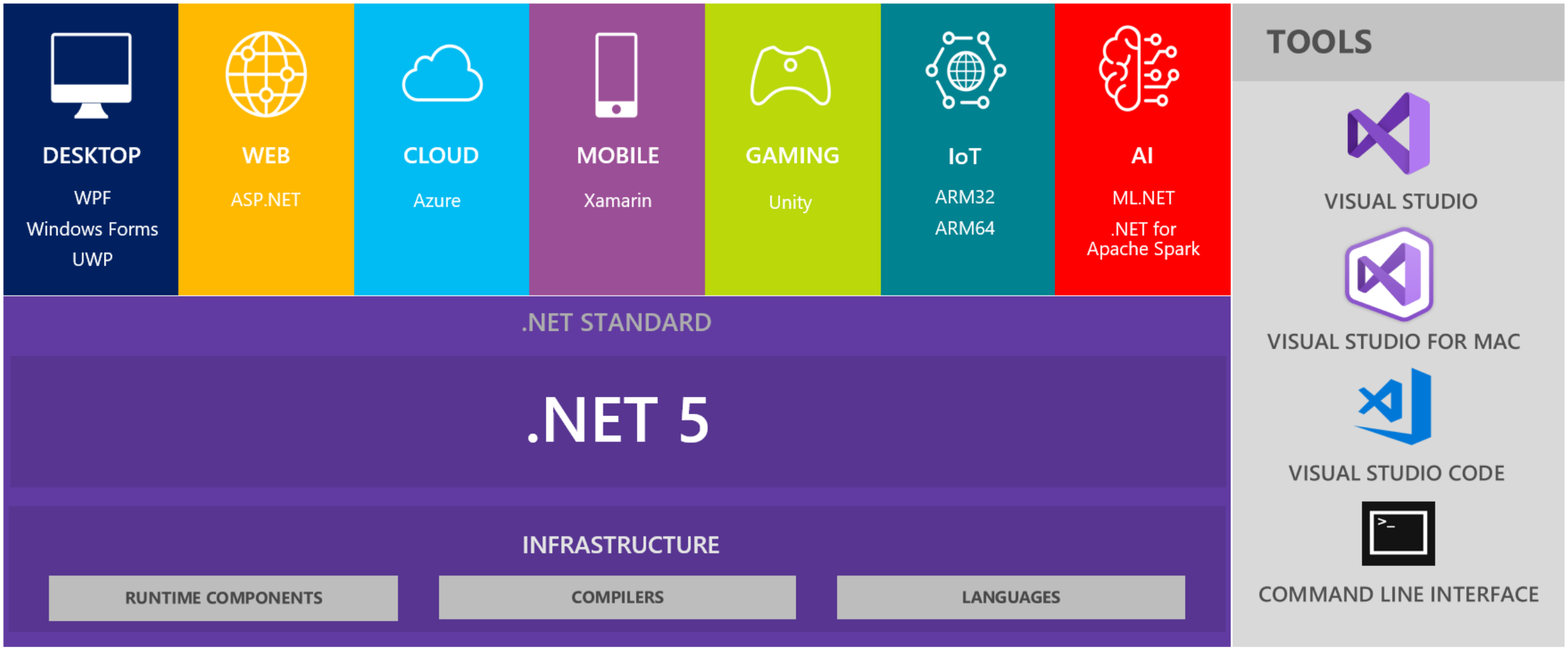
.NET Framework



.NET 5.0

- Is a free, open-source development platform
- Cross platform
  - Operating Systems: Windows, macOS, Linux, Android, iOS, tvOS, watchOS
  - Processor Architecture: x64, x86, ARM32, ARM64
- Can be used to build Web Apps, Web APIs, Cloud native apps, Mobile Apps, Desktop Apps, Games, IoT
- Supports C#, F#, Visual Basic
- Supports AOT(Ahead of time) compilation
- Automatic memory management

.NET – A unified platform



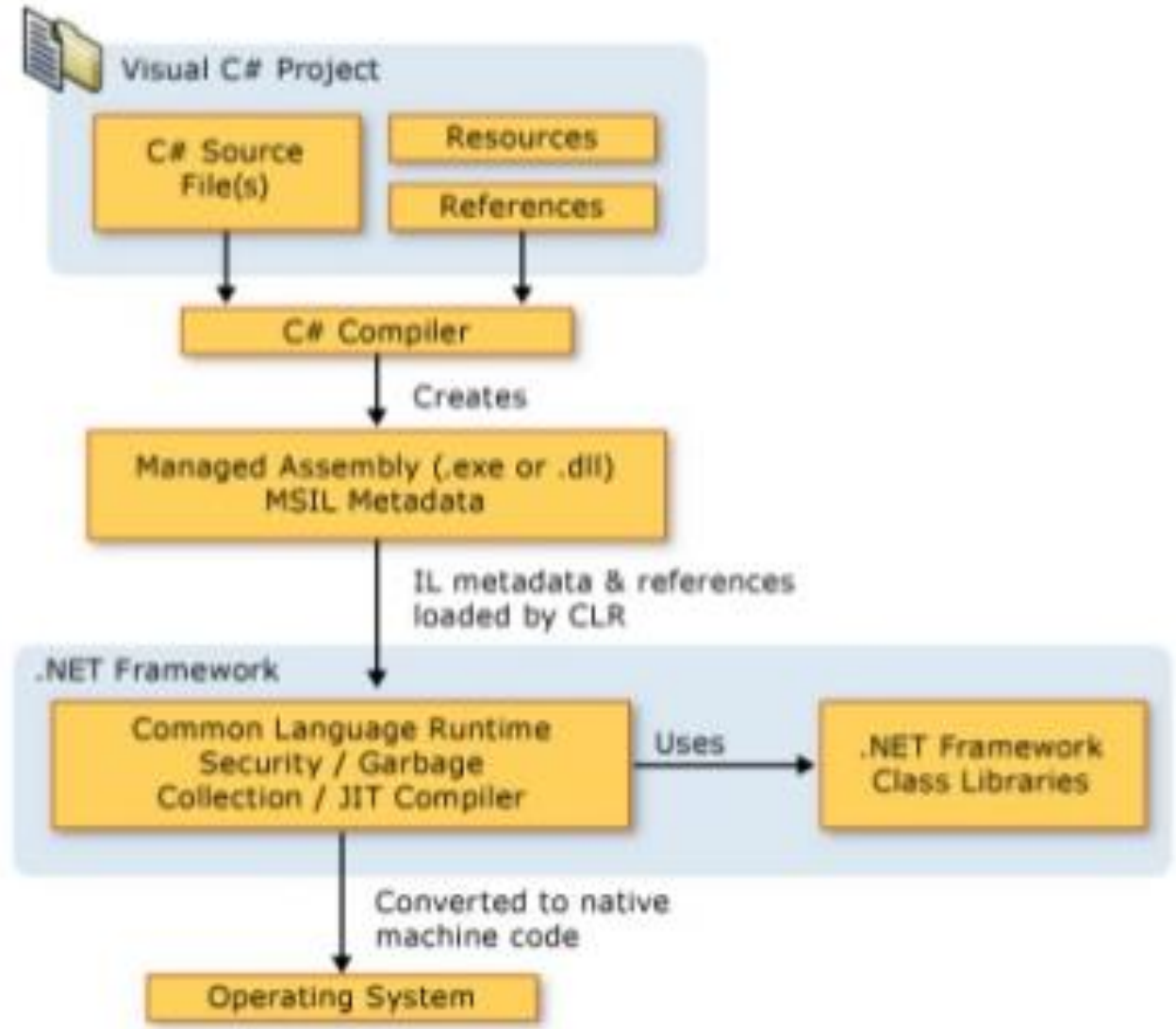
C#

- Developed by Microsoft as part of .NET Framework initiative
- Approved as a standard by ECMA -ECMA-334
- Designed by Anders Hejlsberg and is now lead by Mads Torgersen
- Major Versions

Version	Year	Framwork	IDE
1.0	2002	.NET 1.0	Visual Studio.NET 2002
2.0	2005	.NET 2.0	Visual Studio 2005, 2008
3.0	2008	.NET 3.0	Visual Studio 2008
4.0	2010	.NET 4.0	Visual Studio 2010
5.0	2012	.NET 4.5	Visual Studio 2012, 2013
6.0	2015	.NET 4.6, .NET Core 1.0	.NET Core 1.0, Visual Studio 2015
7.0	2017	.NET 4.7, .NET Core 2.0	.NET Core 2.0, Visual Studio 2017 v15.0
8.0	2019	.NET 4.8, .NET Core 3.0	.NET Core 3.0, Visual Studio 2017 v16.3

https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history

Execution Cycle



Assembly

- Fundamental unit of deployment
- Collection of types and resources
- Assemblies take the form of a .DLL or .EXE
- Assemblies that are part of .NET framework is put in Global Assembly Cache(GAC)

Structure

```
using System;
namespace SampleNamespace
{
    class SampleClass
    {
        static void Main(string[] args)
        {
            //Your program here...
        }
    }
}
```

Type System

Value Types

- Stores data directly
- Cannot be null

Reference Types

- Stores references to objects
- Can be null

Dynamic Types

- Stores any type of value
- Type checking take place at runtime

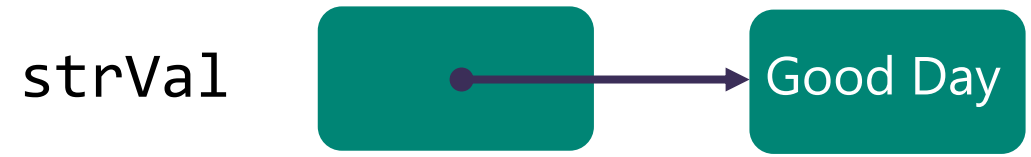
Type System

Value Type

```
int inputVal = 12345;
```

Reference Type

```
string strVal = “Good Day”;
```



Value Types

- Primitives
- Enums
- Structs

- `int i;`
- `enum Selected { “off”, “on” }`
- `struct Point { int x, int y; }`

Reference Types

- Classes
- Interfaces
- Arrays
- Delegates

- `class Foo : Ifoo { … }`
- `interface Ifoo : Ibar { … }`
- `string[] arr = new string[10];`
- `delegte void OnClicked()`

Predefined Types

- Reference
- Signed
- Unsigned
- Character
- Floating Point
- Logical

- `object, string`
- `sbyte, short, int, long`
- `byte, ushort, uint, ulong`
- `char`
- `float, double, decimal`
- `bool`

Comments

```
//Single line comment


/*
Multi line
…
…
Comments
*/


/// <summary>
/// Documentation single line comment.
/// </summary>
public string FirstName { get; set; }


/**
* <summary> Documentation Multi line comment</summary>
*/
public string Last Name { get; set; }
```

Statements

- A single line of code that ends with semi colon(;)
- Series of single line of statements in a block
- A statement block is enclosed in {} brackets
- Can contain nested blocks

```
//Declaration
int age;

//Assignment
age = 25;
```

Variables

Syntax

```
<access specifier> <data type> <name>;
```

Example

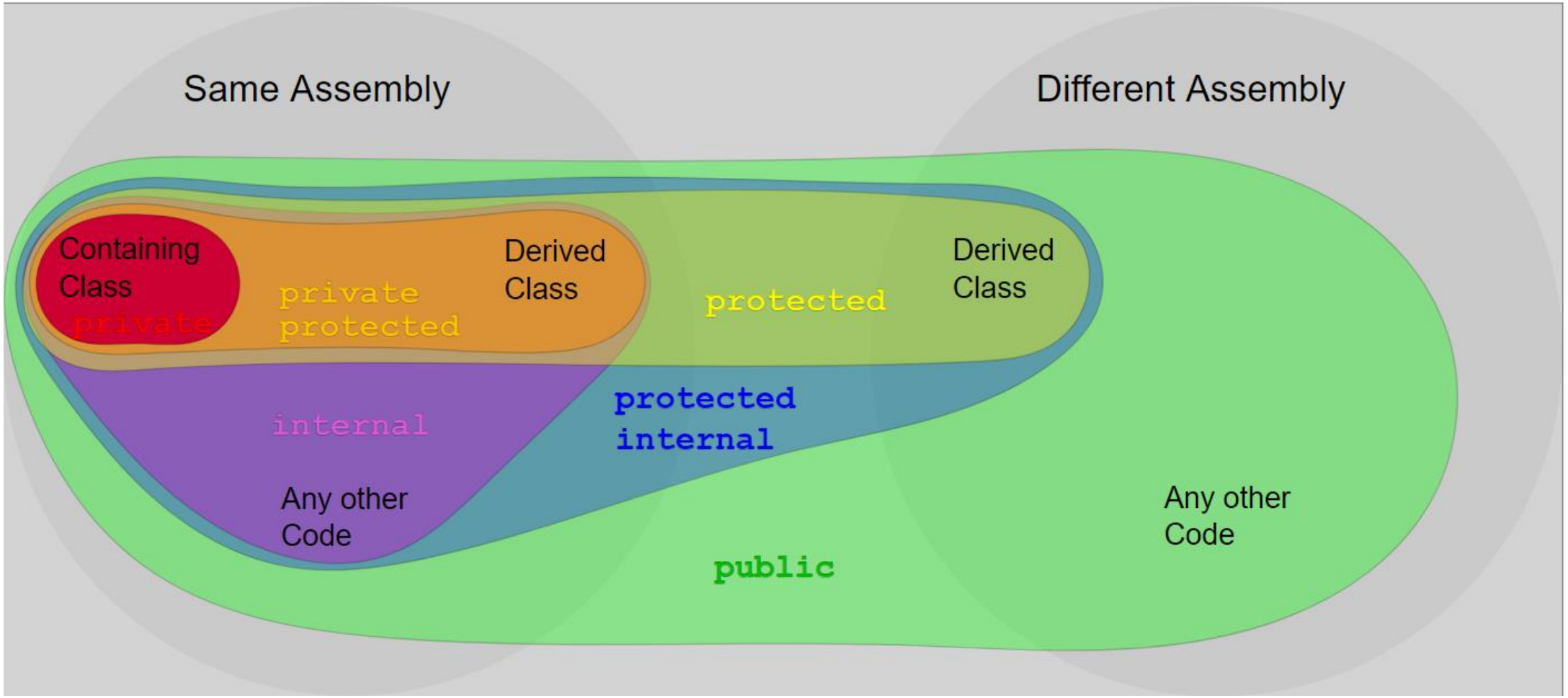
```
private int age; //declaration

private int age = 10; //declaration and assignment
```

Access Specifiers

public	can be accessed by any other code in the same assembly or another assembly that references it
private	can be accessed only by code in the same class or struct
protected	can be accessed only by code in the same class, or in a class that is derived from that class.
internal	can be accessed by any code in the same assembly, but not from another assembly.
protected internal	can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly
private protected	can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.

Access Specifiers



Day 2 : .NET & C# Basics

Functions

Syntax

```
<access specifier> <return type> <name>(params){ //code
}
```

Example

```
public void PrintMessage(string message) //function definition
{
        Console.WriteLine(“Hello “ + message);
}

PrintMessage(“World”); //Invoking function
```

Function Parameters

- Parameters can be passed by value or by reference
- By value is the default

```
public static void Main()
{
   int a = 10, b = 15;
   int c = Add(a, b);
   Console.WriteLine($"Main method : a -> {a}, b -> {b}");
   Console.WriteLine($"Sum of {a} + {b} is {c}");
}

private static int Add(int a, int b)
{
   a = 20;
   b = 25;
   Console.WriteLine($"Add method : a -> {a}, b -> {b}");
   return a + b;
}
```

Output

```
Add method : a -> 20, b -> 25

Main method : a -> 10, b -> 15

Sum of 10 + 15 is 45
```

Function Parameters

- ref modifier is used to pass object as reference

```
public static void Main()
{
   int a = 10, b = 15;
   int c = Add(ref a, b);
   Console.WriteLine($"Main method : a -> {a}, b -> {b}");
   Console.WriteLine($"Sum of {a} + {b} is {c}");
}

private static int Add(ref int a, int b)
{
   a = 20;
   b = 25;
   Console.WriteLine($"Add method : a -> {a}, b -> {b}");
   return a + b;
}
```

Output

```
Add method : a -> 20, b -> 25

Main method : a -> 20, b -> 15

Sum of 20 + 15 is 45
```

Function Parameters

- out modifier is also used to pass object as reference, but without initializing the param

```
public static void Main()
{
   int a , b = 15;
   int c = Add(out a, b);
   Console.WriteLine($"Main method : a -> {a}, b -> {b}");
   Console.WriteLine($"Sum of {a} + {b} is {c}");
}

private static int Add(out int a, int b)
{
   a = 20;
   b = 25;
   Console.WriteLine($"Add method : a -> {a}, b -> {b}");
   return a + b;
}
```

Output

```
Add method : a -> 20, b -> 25

Main method : a -> 20, b -> 15

Sum of 20 + 15 is 45
```

Function Parameters

- params modifier can be used to specify arbitrary number of params

```
public static void Main()
{
   Console.WriteLine("First method ");
   ChangeToLowerCase();
   Console.WriteLine("\nSecond method ");
   ChangeToLowerCase("Germany");
   Console.WriteLine("\nThird method ");
   ChangeToLowerCase("India", "USA", "UAE", "United Kingdom");
}

private static void ChangeToLowerCase(params string[] words)
{
   foreach(var itm in words)
   Console.WriteLine(itm.ToLower());
}
```

Output

```
First method

Second method
germany

Third method
india
usa
uae
united kingdom
```

Function Parameters

- Default values for parameters can be specified in the function definition

```
public static void Main()
{
  int a = 10;
  Console.WriteLine($"Sum = {Add(a)} ");
}

private static int Add(int a , int b = 2)
{
  return a + b;
}
```

Output

```
Sum = 12
```

Function Parameters - Named Arguments

- Default values for parameters can be specified in the function definition

```
public static void Main()
{
Console.WriteLine($"Sum = {Add(secondArg: 20, firstArg:
  15)} ");
}


private static int Add(int firstArg, int secondArg)
{
return a + b;
}
```

Output

```
Sum = 32
```

Expression-Bodied Members

```
private static int Add(int firstArg, int secondArg)
{
   return a + b;
}
```

Can be written as

```
private static int Add(int firstArg, int secondArg)()
   => a + b;
```

```
private static void PrintMessage()
{
   Console.WriteLine(“Hello World”);
}
```

Can be written as

```
private static void PrintMessage()()
  => Console.WriteLine(“Hello World”);
```

Class

Syntax

```
<access specifier> class <name>{
 //code
}
```

Example

```
public class Student{ //function definition
{
        int ID;
        string Name;
        private int string GetStudentName(int Id)
        {
        }
}
```

Properties

- A member of a class which is used to set and get the data from a data field of a class

- Types available are
  - Read-only property
  - Write only property
  - Read Write property
  - Auto-implemented property

Properties

```
//Read only

 public class Employee
 {
    public string FirstName
    {
        get { return firstName; }
    }
    private string firstName;
 }
```

```
//Write only

 public class Employee
 {
    public string FirstName
    {
        set { firstName = value; }
    }
    private string firstName;
 }
```

Properties -- continued

```
//Read and Write

  public class Employee
  {
    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }
    private string firstName;
  }
```

```
//auto property syntax
public class Employee
 {
   public string FirstName
   {
       get ;
       set ;
   }

 }
```

Properties -continued

```
//initialize string property to empty string rather than null
public string FirstName { get; set; } = string.Empty;


//expression bodied syntax
public string FirstName
{
   get => firstName;
   set => firstName = value;
}
private string firstName;

//readonly property
public string FirstName { get; private set; }
```

Strings

- Anything within a set of double quotes is considered as string
- It can be empty as well as null

```
string numbers = "";
for (int i = 0; i < 10000; i++)
    numbers += i.ToString();

/*when this code is executed, the framework creates a
new string 10000 times which means that in need to
allocate memory and assign old variable that many
times */
```

Better approach

```
//use string builder

StringBuilder numbers = new StringBuilder();
for (int i = 0; i < 10000; i++)
    numbers.Append(i);
Console.WriteLine(numbers.ToString());
```

String Operations

```
string msg = "Hello" + " " + "World"; //concatenation
Console.WriteLine(msg.Length); //gets length of the string output : 11

string newMsg = msg.Replace("World","India");
Console.WriteLine(newMsg); //replaces string with another output: Hello India

if (newMsg.Contains("India"))
        Console.WriteLine(newMsg.Replace("India", "World")); //replaces all occurances of a string if a match is found, output -> Hello World
Console.WriteLine("Hello \"World\""); //escaping double quotes , output : Hello "World"

Console.WriteLine("Yes \\ No"); //escaping back slash , output : Yes \ No

Console.WriteLine(@"In a verbatim string \ everything is literal: \n & \t"); //is notified by the @ symbol, output : In a verbatim string \ everything is literal: \n & \t
```

String Interpolation

```
string msg1 = "hello";
string msg2 = "World";

Console.WriteLine(msg1 + " " + msg2); //old way

Console.WriteLine($"{msg1} {msg2}"); //using interpolation

Console.WriteLine($"Insert \"{msg1}\" between curly braces: {{message here}} {msg2}");
//output -> Insert "hello" between curly braces: {message here} World
```

Exception Handling

- Compilation Errors
  - Occurs during compile time
  - Occurs due to syntactical mistakes
- Runtime Errors
  - Occurs during the execution of the program
  - Due to this program terminates abnormally

Exception Handling

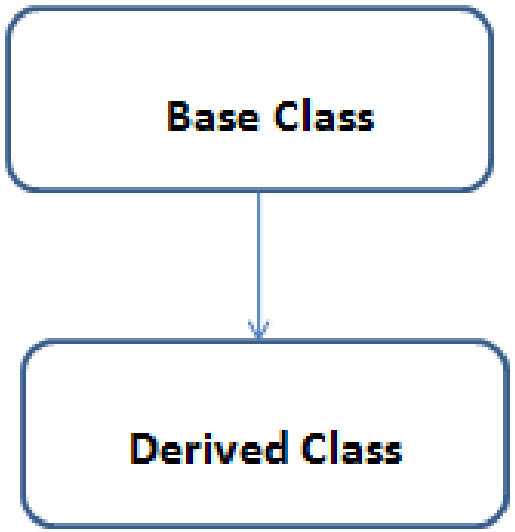
```
//Single line comment
try
{
   // statements causing exception
}
catch( ExceptionName e1 )
{
   // error handling code
}
finally
{
   // statements to be executed
}
```

```
try
{
   result = num1 / num2;
}
catch (DivideByZeroException e)
{
   Console.WriteLine("Exception caught: {0}", e);
}
finally
{
   Console.WriteLine("Result: {0}", result);
}
```

Day 3 : .NET & C# Basics

Inheritance - Single

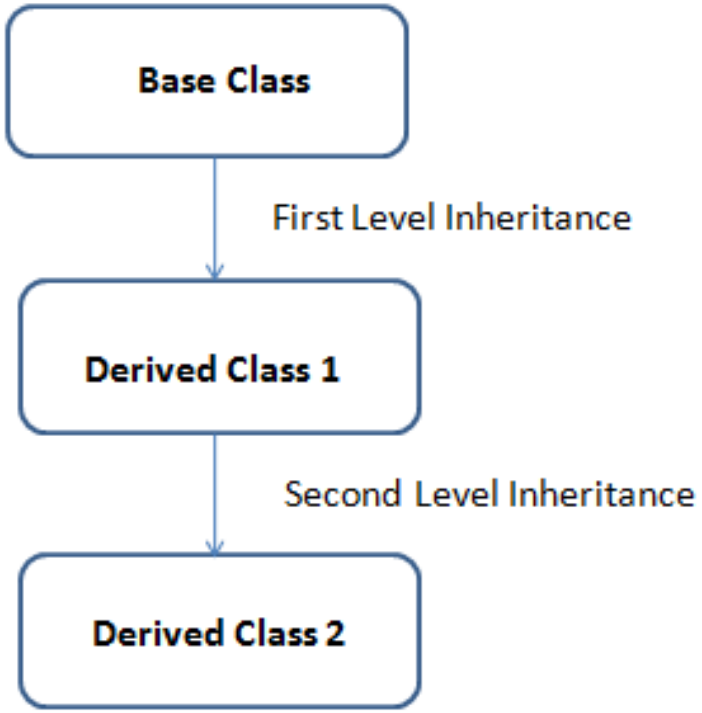
- Derived Class inherits properties and behavior from a single base class




```
//Base Class
class Parent {
  public void Method1() {
    //TO DO:
  }
}
//Derived Class
class Child: Parent {
  public void Method2() {
    //TO DO:
  }
}
```

Inheritance - Multilevel

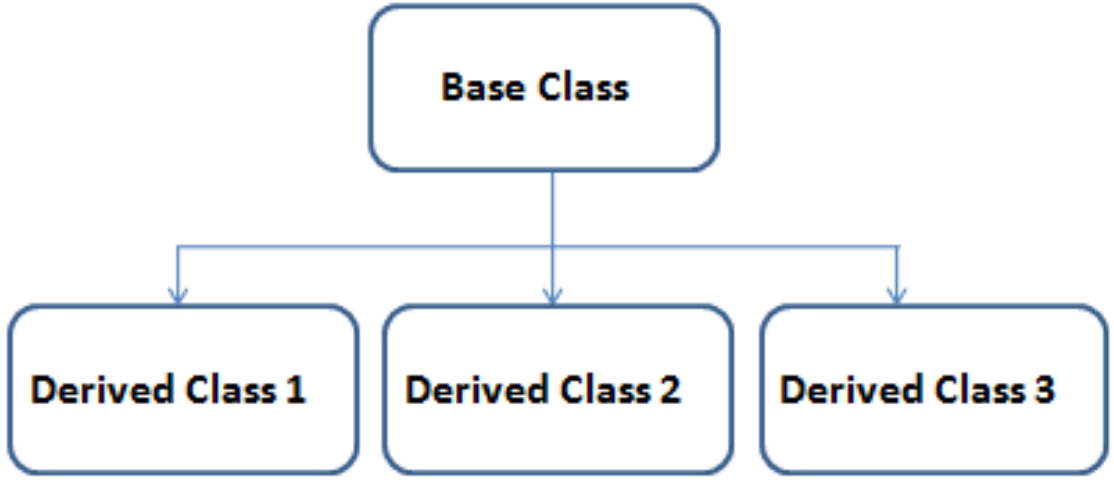
- Multiple derived classes are there
- Lower-level derived classes are derived from another derive class




```
class Vehicle {
  public void Method1() {
    //TO DO:
  }
}
//Derived Class - level1
class FourWheeler: Vehicle {
  public void Method2() {
    //TO DO:
  }
}
//Derived Class - level 2
class Car: FourWheeler {
  public void Method2() {
    //TO DO:
  }
}
```

Inheritance - Hierarchical

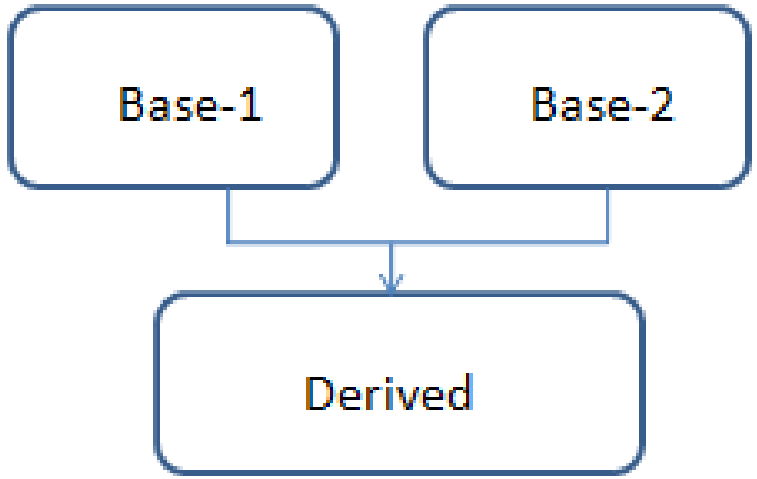
- More than one derived classes are created from a single base class




```
//Base Class
class Parent {
  public void Method1() {
    //TO DO:
  }
}
//Derived Class
class Child1: Parent {
  public void SubMethod1() {
    //TO DO:
  }
}
//Derived Class
class Child2: Parent {
  public void SubMethod1() {
    //TO DO:
  }
```

Inheritance - Multiple

- More than one base classes are used to from a single derived class
- Classes are only allowed to inherit from one parent class, but can inherit from multiple Interfaces

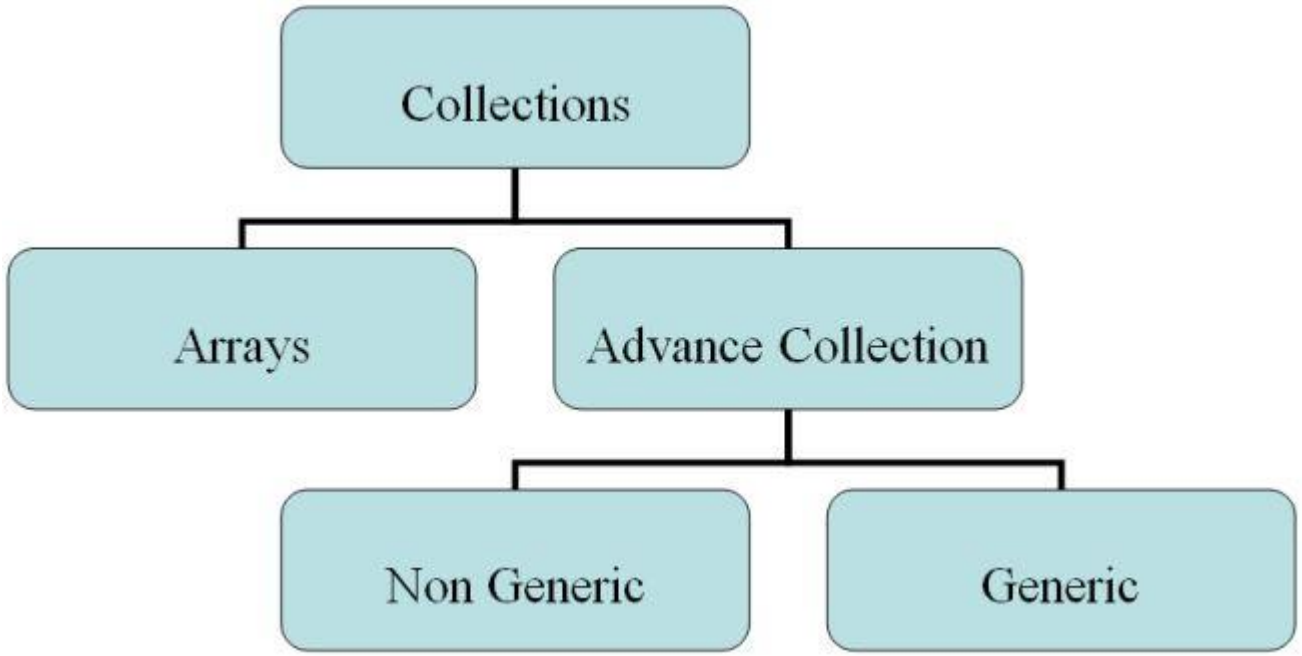



```
//Base Class
class Parent {
  public void Method1() {
    //TO DO:
  }
}
interface Interface1 {}
interface Interface2 {}

//Derived Class
class Child: Parent, Interface1, Interface2
{
  public void Method2() {
    //TO DO:
  }
}
```

Collections

- Is a set of similar objects grouped together
- System.Collections contains specialized classes




```
//Base Class
class Parent {
  public void Method1() {
    //TO DO:
  }
}
//Derived Class
class Child1: Parent {
  public void SubMethod1() {
    //TO DO:
  }
}
//Derived Class
class Child2: Parent {
  public void SubMethod1() {
    //TO DO:
  }
```

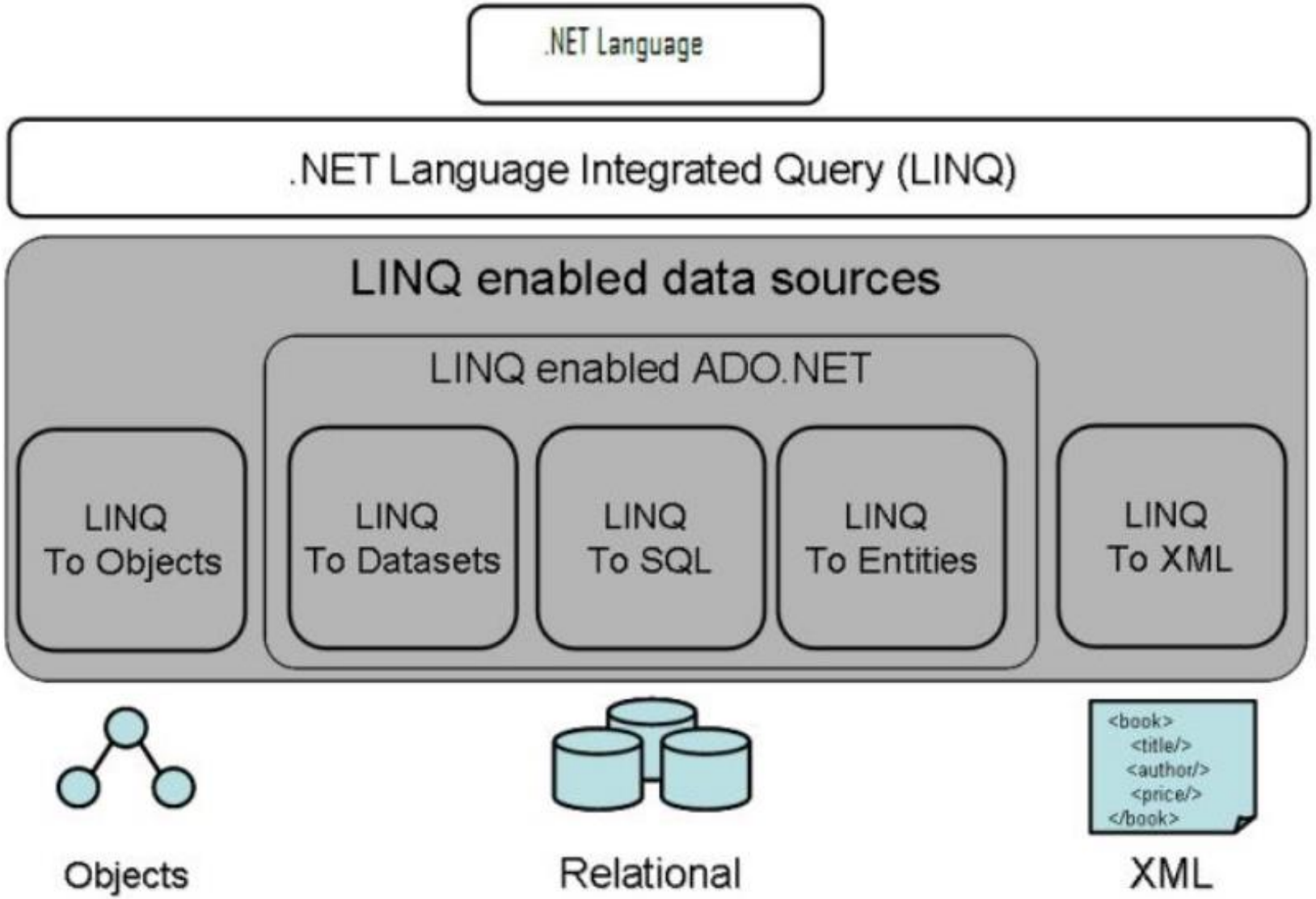
Language Integrated Query(LINQ)

- Offers a common syntax for querying any type of datasource
- Is a data querying API with SQL like syntax
- Types
  - LINQ to Object – iterating in-memory collections
  - LINQ to XML – for querying XML datasoures
  - LINQ to ADO.NET – for querying datasources supported by ADO.NET such as SQL

Example

```
int[] nums = new int[] {0,1,2};
var res = from a in nums where a < 3 orderby a select a;
foreach(int i in res)
 Console.WriteLine(i);
```

LINQ - Architecture



Lambda Expressions

- Are anonymous functions that contains expressions or sequence of operators
- All Lambda expressions uses **=>** operator
- Syntax - *Parameter => expression*
- Left side of the lambda operator specifies the parameters
- Right side holds an expression or a code block

Example

```
int[] nums = new int[] {0,1,2};
var res = nums.Where(a => a < 3).OrderBy(b=>b);
foreach(int i in res)
 Console.WriteLine(i);
```

LINQ vs Lambda Expression

Query Syntax

```
var vendorQuery = from v in vendors
                  where v.CompanyName.Contains("Toy")
                  orderby v.CompanyName
                  select v;
```

Method Syntax

```
var vendorQuery = vendors
                  .Where(v => v.CompanyName.Contains("Toy"))
                  .OrderBy(v=> v.CompanyName);
```