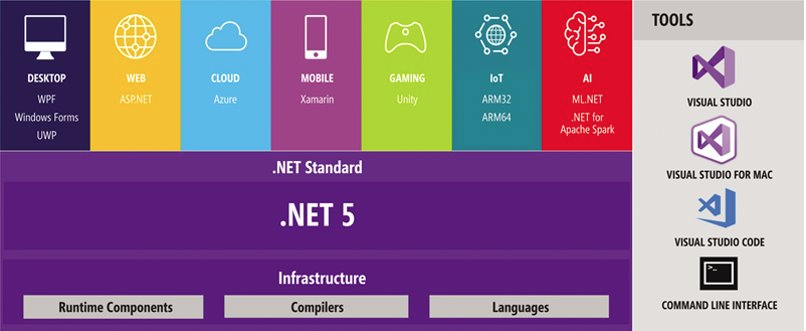
# .Net Architecture Components

* There are many frameworks that is already included in our .Net 5.0
  + A framework just has predefined classes and libraries to help us start an application.
  + One framework we will be using in the future is ASP .Net (used to develop web application)

## SDK vs. Runtime

* Software Development Kit (SDK) it includes everything we need to build and run a .Net application.
  + You can run them either using Command Line interface (CLI) or you can use an Integrated development environment (IDE).
* Runtime
  + It includes all the resource we need to run an existing .Net application
  + A lot less storage required to download and install
    - Mostly used for end-users.



## .Net standard

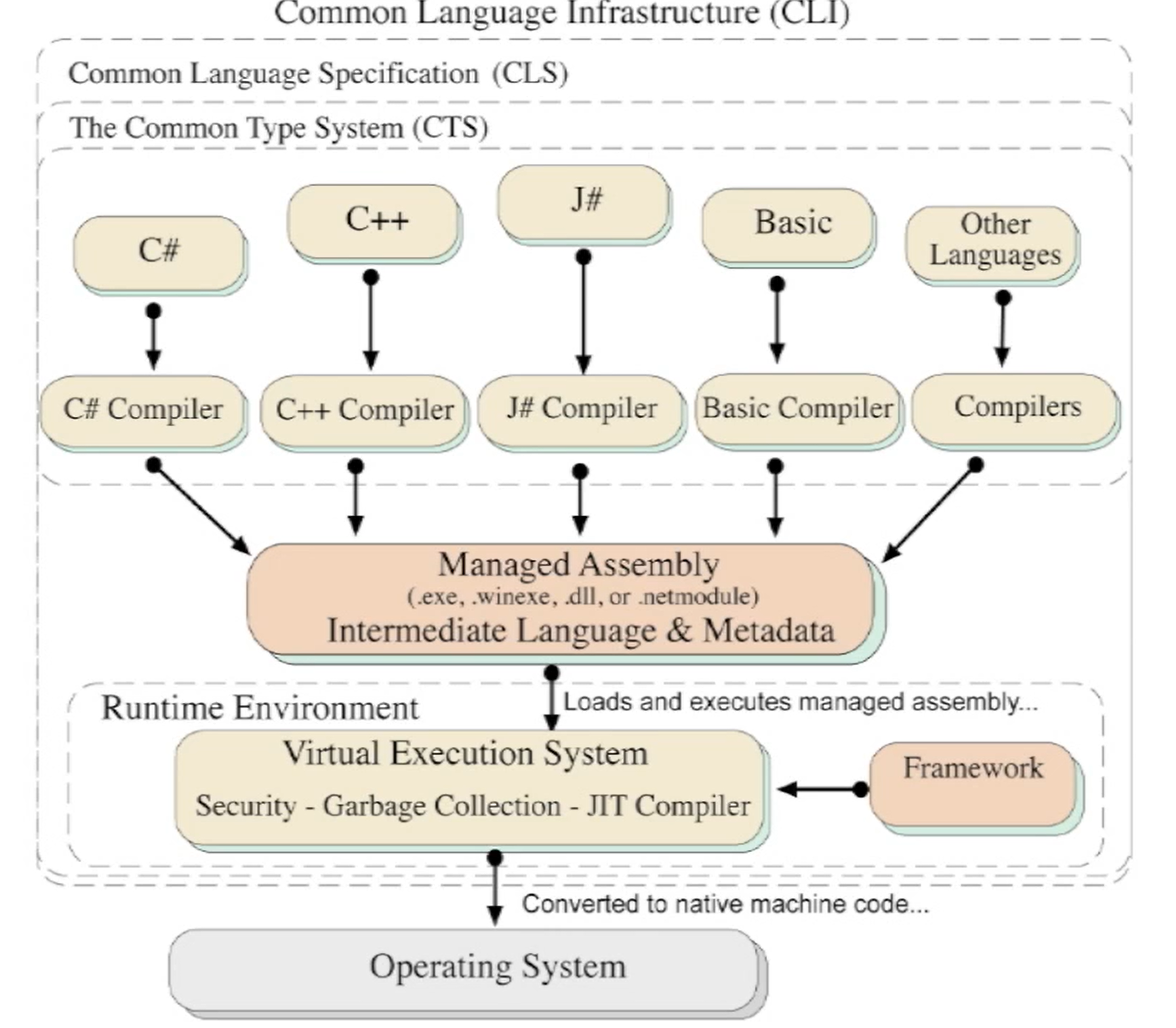
* It is a set of APIs that are implemented by the **Base Class Library** (BCL) which provides the classes and types that are helpful in performing day to day operations.
  + Ex: dealing with strings, primitives types (value types), database connection, etc.
* With .Net standard, you can use multiple language to create your application.
  + Your apps can run in F#, Visual Basics, and other languages
    - They are .Net compliant language
* Each version of .Net standard are additive
  + Any previous versions of .Net will be included to the current version of .Net

## Managed Code

* It is a code whose execution is managed by runtime.
* The CLR uses the managed code and compiles into machine code and then executes it.

## Unmanaged Code

* Code developed outside of the .Net environment.
* Not manage by the CLR.
* Can be executed with the help of wrapper classes.



# CLI

* Stands for Common Language Infrastructure.
* This enables an application program to be written in numerous programming language to be run on any operating system.
* It is comprised of:
  + CLS
    - CTS
    - VES
      * CLR

## CLS

* Stands for Common Language Specification.
* They will define the rules and restriction that every language must follow for it to be able to run the .Net framework.
* **CTS**
  + Common Type System
  + Provides a library of the basic primitive data types.
  + Ensures that every data type in each programming language is talking about the same thing.
    - i.e. int data type in C#,J#, VB has to be 32-bit
    - This lets us create **Language Interoperability**.

## VES

* Run-time system of CLI
* It provides an environment for executing managed code
* **CLR**
  + Stands for Common Language Runtime
  + It is essentially .Net framework’s implementation of VES
  + This lets us able to run our application across multiple systems (Windows, Mac, Linux, etc.)
    - Automatic memory management (older language you have to manually release unused resources).
    - JIT compilation – Just-in-time compilation involves compiling the code while executing for optimization.
    - Exception handling support

## Garbage Collection

* CLR provides automatic memory management of your heap memory.
* It checks for objects in the managed heap that are no longer being used by the app.
* However, resources that are unmanaged code needs to be clean up manually.
  + We can use IDisposable interface to clean up those external resources.
* You can also use *using* statement for clean up.
* Pros:
  + No memory leaks.
  + Efficient memory allocation.
  + Garbage collection ensures that one object cannot use the memory of another object.
* Cons:
  + Since this process is automated, it can negatively impact the performance of your code especially when garbage collection must regularly run through the program and checking if the object is being referenced and needs cleaning up.

# 

# Data types

* We can use data types to structure the data and tell the compiler to how we intend to use that data.
* Fun fact! Unlike java, all types inherit from the Object base class.
* They are two major types:
  + Value types
  + Reference types

## Value types

* They are same as Java primitive types.
* They derived from the System.ValueType which inherites System.Object.
* Stores in the **stack** and not the heap.
* It means you get the value directly and not a reference to where the value is stored.
* There is no garbage collection for value-type variables.
  + Memory is automatically reclaimed when a stack frame is removed.

## Reference types

* They do not have the value directly, but they have a variable that reference that value.
  + Ex: classes, delegates, arrays, or interfaces, etc.
* When you declare a variable of a reference type and not have it point to anything in the beginning, it will have a null value.
* They are stored in the **heap** and the reference variable is stored in the **stack.**
* Like in Java, string are also classes, but main difference is we can use “==” to compare two strings since .Net is smart enough to not compare the reference value and compare the actual value.

# Collections

* It is a data structure that can hold many values.
* All collections have methods add, remove, or find items since they all inherit from Innumerable Interface.
* In C#, there are two major types of collection: Generic and non-generic.

## Generic

* They allow you to type safe collection.
  + Meaning you can add a data type in the collection and only that data type is allowed to be stored.
* They come from System.Collections.Generic namespace.
* The “T” you see in documentation is where you can put that data type that collection will hold.
* List of Generic Collections:
  + List<T> - It grows automatically as you add element, zero-based index
  + Stack<T> - LIFO
  + Queue<T> - FIFO
  + Dictionary<TKey, TValue> - A collection that represents a key/value pair
  + SortedList<TKey, TValue> - same as dictionary buuut it automatically order the key in (you decide)
  + HashSet<T> - contains non-duplicated elements

## Non-generic

* They allow you to store multiple data types.
* They come from System.Collection namespace.
* List of non-generic collections
  + ArrayList – non-generic version of list
  + SortedList – non-generic version of sortedlist
  + Queue – FIFO
  + Stack
  + Hashtable

# Object Oriented Programming

* It is a methodology we use to design our programs just using classes and objects.
* It makes it easier to develop and maintain your project as it gets bigger.

## Objects

* It is any entity that has a state and behavior
* Can be defined as an instance of a class

## Classes

* They are templates that are used to create objects and defines the object data types, methods, constructor, etc.
* They are blueprint from which you can create multiple objects from

# Inheritance

* It is a mechanism in which an object can acquire all the properties, methods, constructor of a parent object.
* It allows us to create classes that are built upon existing classes.
* It creates the is-a relationship, also known as parent-child relationship between two classes.
* You can inherit one class and only one class but you can implement multiple interfaces

## Advantages

* Code reusability and organization
* Allows us to use polymorphism (method overriding)
* Covariance

## Terminology

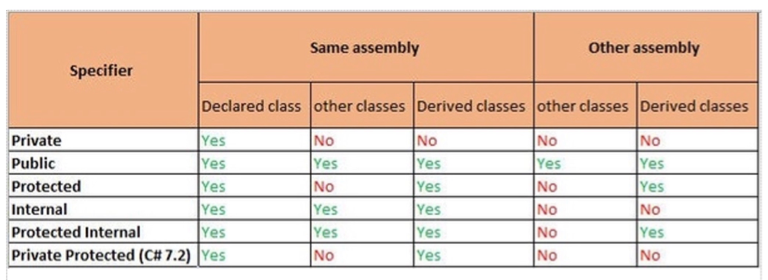
* Base class
  + It is the parent class in which the child class inherits from.
* Derived class
  + It is the child class that inherits from the parent class.

## Different types of inheritance

* Single inheritance
  + Where the base class only has one derived class
* Multi-level inheritance
  + Like the single inheritance, but the derived class will also have another derived class that inherits from.
* Hierarchical inheritance
  + Where the base class has multiple derived classes and those derived classes have their own multiple derived classes
  + It is similar to how family trees work

# Access Modifiers

* They restrict the scope of the classes, methods, fields to be only accessible in certain areas



# Polymorphism

* The ability of an object to take on many forms.
* It allows you to substitute different implementation details for different needs.

## Method Overriding

* When a derived class changes the implementation details of a method from a base class.
* You need to use virtual (for the base class) and override (for the derived class) for method overriding in C#.

## Method Overloading

* When there are multiple methods but with different parameters and most of time, different implementation details.

# Abstraction

* The process of hiding the implementation details and only showing the functionality to that user.
* The purpose is to both simplify the life of the end-user and to add security since the end-user doesn’t know how it works.
* Ex: You know how to send text in your phone but you don’t know the entire details on how exactly that text is being sent and received by the other person.

## Interface

* It contains nothing but abstract methods and properties.
* It means it has no implementation details.
* You can also implement multiple interfaces.

## Abstract class

* You can have some methods and properties without implementation.
* May also contain concrete methods with implementation.
* You can only inherit one abstract class like a normal class.
  + Mostly because there is a chance that diamond problem will occur.
* You can have constructors in abstract classes.

# Encapsulation

* The process of wrapping our code and data together into a single unit.
* So essentially, the validation and any process of data in your class will be handled by the class itself.
* They prevent unauthorized access to your object’s properties and setting values that should not be there.
* You can also manipulate your models to either be just read-only or write-only.

# Non-access Modifier

## Abstract

* Applies to: classes, methods, properties, etc.
* They enable you to create incomplete implementation of whatever you applied to it and the derived class must give it an implantation details

## Const

* Applied to fields
* Constant field means you cannot modify the value
* Constant can be an int, Boolean, string, or null
* Do not create a constant field to represent an information that you expect to change at any time.
* You must initialize at declaration.
* They are implicity static.

## Readonly

* Applied to fields, structs, class members, etc.
* Readonly fields
  + That field can only be instantiated once
    - Can only be instantiated in the constructor or at the declaration.
    - They can be initialized at a later time.
* Readonly structs
  + Once you create that struct, it is immutable.

## Sealed

* Applied to: class, class members
* Sealed classes cannot be inherit by other classes.
* Sealed methods and properties aren’t overridable in any classes that inherit those members.
* Similar to final

## Static

* Applied class, class members
* Static class cannot be instantiated or inherited, its class members must be static also
* Static class member belongs to that class itself rather than a specific object
* Static fields belongs to that class itself

## Virtual

* It allows for it to be overridden in a derived class.
* By default, methods are non-virtual therefore you cannot override.
* You can’t make virtual methods private

## Override

* Override methods must do method overriding or it gives an error
  + Ensures that the method you created is overriding a method from the base class

## Partial

* Applied to: classes, structs, interfaces
* It will split the class into two or more source files.
* Advantage:
  + Let’s you work on the same class with another person without interrupting their workflow.
* At compile time, it will merge those partial class into one.