# Introduction to .NET

* It is a free, open-source development platform for a variety of applications (games, websites, consoles, etc.)
* Essentially, it is a collection of languages and libraries that can work together to build different types of software/application

## Different types of open-source developer platform

* .NET 5.0
  + The one you have right now
  + We are using this since it supports the most types of applications and platforms compared to the other two
* .NET core
  + Essentially the older version of .NET 5.0
* .NET framework
  + It is used to create windows-only desktop or server-based applications

## What is C#?

* It is an object-oriented and type-safe programming language
  + Object-oriented just means that everything is based on objects and the relationships between
  + Type-safe just means once you set a type you can’t change it
* It is one of the main languages that we will deal with when using the .NET 5.0

# Application Architecture

## Separation of Concern

* The concept of organizing our code
* We want our code to follow a certain theme of functionality
* We can do this by leveraging classes and other grouping mechanism to group data and logic together
* This is first but **important** step to writing readable, extendable, and maintainable

## Classes

* They are the building blocks of your code
* They are the blueprints of creating objects that you process in your program

## Namespace

* Logical grouping of types that follow a certain theme of functionality
* Grouping of classes

## Projects

* They contain all the files that are compiled into an executable, library, UI component, database.

## Solution

* It is a container for one or more related project
* Grouping of projects

## Assembly

* It is a group of files that was generated by the SDK that the computer understands and can execute (for windows, it will be the .exe and .dll files)
* It is in the bin and obj folders

# .NET Architecture Components

* There are many frameworks that is already include in .NET 5.0
  + Frameworks just have predefined classes and libraries to help us start an application
  + One framework we will be using in the future ASP.NET MVC (Used to developed web application)

## SDK vs Runtime

* Software Development Kit (SDK) includes everything we need to build and run a .NET application
  + You can run them either by using Command-line interface (CLI) or you can use Integrated development environment (IDE)
* Runtime
  + It just includes all the resources we need to run existing .NET application
  + A lot less storage required to download and installed

## .NET standard

* It is set of APIs (Application Programming Interface) that are implemented by the **Base Class Library** of a .NET implementation
* Base Class Library (BCL) provides the classes and types that are helpful in performing day to day operations in your SDK
  + Dealing strings
  + Primitive types
  + Database connections
  + IO operations
* With .NET standard, you can use multiple language to create your application
  + You can use F#, Visual basics, and other language
  + We will just use exclusive C#
* Each version of .Net standard is additive
* Whenever a language is compatible in the .NET 5.0, they are known as a .NET compliant language

## ×Managed Code

* Code whose execution is managed by a runtime
* The Common Language Runtime (CLR) uses the managed code and compiles it into machine code and then execute

## ×Unmanaged Code

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

# ×CLI

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

## ×CLS

* Common Language Specification
* The will defined rules and restriction that every language must follow for it to be able to run in the .NET framework
* CTS
  + Common Type System
  + Provides a library of the basic primitive types
    - Primitive types are just int, double, Boolean, float
* VES
  + Run-time system of CLI
  + It provides an environment for executing managed code
  + CLR
    - Common Language Runtime
    - .NET framework’s implementation of VES
    - Run-time environment that provides services that makes the development process easier
    - Some services/features
      * Automatic memory management
      * JIT compilation – Just in Time compilation involved compilation during execution for optimization
      * Exception handling support

# ×Garbage Collection

* CLR provides the automatic memory management of your heap memory
* It checks for objects in the manage heap that are no longer being used by the application
* Resources that are unmanaged code needs to be clean up manually
  + IDisposable interface will clean up those external resources

## Pros

* No memory leaks
* Efficient memory allocation
* Garbage collection ensures that one object cannot use the memory of another object

## Cons

* This process is automated and can negative impact performance of your code
* You cannot call upon the garbage collection to start reclaiming memory

# Datatype

* We can use data types to structure the data and tell the compiler to how we intend to use that data
* Fun fact! C# all data types inherit from System.Object class
* There are two major types:
  + Values types:
    - Int – holds whole numbers with 32 bits of memory
    - Char – holds a single character with 16 bits of memory
    - Double – holds whole numbers and decimals with 64 bits of memory
    - Float – like double but only holds 32 bits of memory
    - Etc. – there are a lot more
  + Reference types:
    - Objects
    - Strings – since it is a reference type, it is immutable/cannot change

## Values Types

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

## Reference Types

* They do now have the value directly, but they have a variable that reference that value
* Ex: classes, strings, delegates, arrays, or interface.
* 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
  + Meaning it points to nothing
* Using the new operator assigned it an “address” to point to an existing object that the new operator has created
* The object itself is stored **heap** and the reference variable is stored in the **stack**

# Collections

* It is a data structure that can hold many values
* All collection methods to add, remove, find items since they all inherit from IEnumerable interface
* In C#, there are two major types of collection

## Look into Arrays

## Generic

* They allow you to type safe collection
* They come from System.Collections.Generic namespace
* The “<T>” you see in the documentation is where you put the data type that collection will hold
* Generic collections:
  + List<T> - it grows automatically as you add elements to it, zero-based index
  + Stack<T> - LIFO
  + Queue<T> - FIFO
  + **Dictionary<TKey, TValue>** - a collection that represents a key/value pair

## Non-Generic

* They allow you to store multiple data types
* They come from the System.Collections namespace
* Non-generic collections:
  + ArrayList
  + SortedList
  + Queue
  + Stack

# Introduction to Serialization

* It is just the process of converting an object into a stream of bytes or a **JSON** or XML for storage or transfer.
* The reverse process called is deserialization

## JSON Serialization

* Popular choice in .NET and it is provided by using System.Text.Json namespace
* All public properties in a class are serialized (transferred into a JSON file) and you can specify which one to exclude
* Fields are excluded in JSON (not supported)
* It converted into string, byte array, or some sort of standardization.