Apuntes Clase 1

¿Qué es una PLATAFORMA DE DESARROLLO?

Una plataforma de desarrollo es un entorno de software que cuenta conjunto de herramientas que nos permite construir determinadas aplicaciones de software.
Por ejemplo: editores de código, compiladores, entornos de ejecución, lenguajes de programación, bibliotecas, etc.

Introducción a .NET

Historia e introducción a .NET

.NET es una plataforma de desarrollo que nace en 2002 de la mano de Microsoft, en esa época el negocio eran las licencias de Windows y Software para Windows, más adelante este paradigma se vería afectado y Microsoft trasladaría su atención a proveer soporte en la nube.

Lo que se quería lograr con .NET era facilitar a los desarrolladores la posibilidad de programar aplicaciones para Windows utilizando múltiples lenguajes de alto nivel. Fue pensado como multiplataforma pero finalmente se lanzó en Windows (Hoy día es multiplataforma).

A fines del año 2000 se publicaron una serie de especificaciones de cómo construir un código ejecutable y un entorno de ejecución que permita programar con distintos lenguajes de alto nivel de forma agnóstica a la maquina donde se terminaría ejecutando el programa (Common Language Infrastructure), esto se logra a partir del entorno de ejecución (Common Language Runtime) que a partir de su software compila la información y permite que ese código sea multiplataformas y no haya necesidad de reescribirlo.

Originalmente, Microsoft sólo ofrecía un entorno de ejecución para Windows, el Common Language Runtime. El proyecto de código abierto Mono se interesó en la plataforma y empezó a evaluar la posibilidad de desarrollar un entorno de ejecución para Linux. Luego de varias adquisiciones de la empresa, Miguel de Icaza (ex dueño de la empresa que desarrollaba Mono) funda Xamarin donde continua el proyecto de Mono.

En 2016, la empresa es adquirida por Microsoft que liberó el código fuente del SDK de Xamarin y comenzó a ofrecer gratuitamente las integraciones y características para Visual Studio.
Fue así como Mono dio origen a la plataforma para desarrollo mobile Xamarin y otras como Unity, que se utiliza para el desarrollo de videojuegos multiplataforma. La versión final de Mono es la 5.0.

Con la compra de Xamarin comienza el trabajo de implementar un Common Language Runtime multiplataforma. Es así como nace .NET Core.
.NET Core es una implementación de .NET Framework y Mono, pero fue hecha de cero, modularizada, debugeada y optimizada. Así es como con .NET se puede desarrollar en multiplataformas (Linux, Mac, Mobiles).
Así fue como comenzó la era multiplataforma y open source de .NET.

Niveles de Soporte de .NET

Long-Term Support (LTS): Son versiones estables y que se actualizarán con poca frecuencia. Se garantiza su soporte por 3 años desde su lanzamiento. Por ejemplo, la versión 3.1 que fue lanzada en diciembre de 2019 tiene soporte hasta diciembre de 2022.
Current: Contienen las últimas mejoras y tienden a actualizarse con frecuencia. Reciben actualizaciones hasta 18 meses después de su lanzamiento. Por ejemplo, la versión 5.0 que fue lanzada en noviembre de 2020 tiene soporte hasta mayo de 2022.

Características de .NET

- Multiplataforma
- Multilenguaje
- Open Source

Hoy soporta 3 lenguajes: Visual Basic, F# y C#

Componentes de .NET

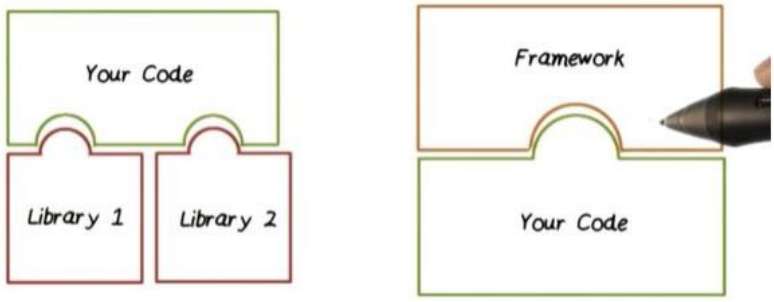
Todas las implementaciones de .NET incluyen los siguientes componentes:

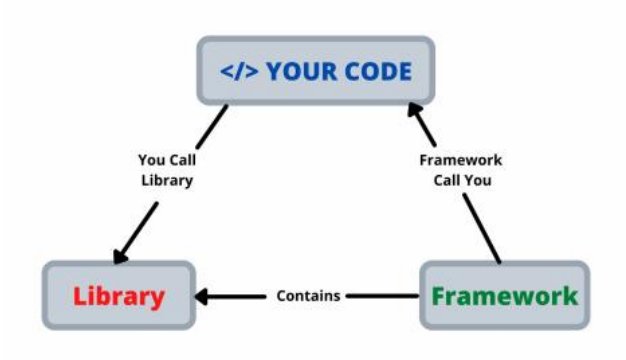
- Common Language Runtime (CLR): Entorno de ejecución, es el encargado de reservar, asignar y liberar memoria según la necesidad.
- Base Class Library (BCL): Son una serie de bibliotecas bases que van a tener funcionalidades bases de .NET
- Componentes de Infraestructuras común: Lenguajes de Programación, sus compiladores, sistemas de proyectos, etc.
- Frameworks: Se utilizan para desarrollar aplicaciones específicas, ejemplo: Windows Form, WPF, ASP.NET.
- Herramientas de Desarrollo: Editores de código, IDEs, interfaz de líneas de comando, etc.

Frameworks vs Bibliotecas

Las bibliotecas son una serie de funciones o de funcionalidades para realizar tareas específicas. Los marcos de trabajo (frameworks) nos dan un conjunto de herramientas mucho mas grandes que las bibliotecas y define un modelo de trabajo, nos va a indicar como desarrollar. El framework va llamar al código, a diferencia de las bibliotecas que nosotros teníamos que invocar las funciones.

BIBLIOTECAS (LIBRARIES)	MARCOS DE TRABAJO (FRAMEWORKS)
Se trata de una serie de funcionalidades para realizar operaciones específicas, bien definidas.	Definen una forma de trabajo y nos brindan un conjunto de herramientas de alto nivel que permiten desarrollar ciertos tipos de aplicaciones con facilidad.
Se componen de una colección de funciones y objetos auxiliares.	Se componen de múltiples bibliotecas y otras herramientas.
Nosotros invocamos las funciones de la biblioteca a necesidad.	El framework invoca al código y maneja el flujo de ejecución.
El desarrollador tiene libertad y control para usar la biblioteca como desee.	El framework tiene un comportamiento por defecto y define un estándar para el desarrollo.





Proceso de Compilación

Se compilan los archivos que contienen el código fuente a lenguaje intermedio. Este lenguaje intermedio NO es lenguaje máquina.

Al ejecutarse la aplicación, el lenguaje intermedio se compila a **lenguaje nativo** (máquina) por el CLR

Cuando ejecutemos la aplicación este lenguaje intermedio va a ser levantado por el CLR y lo va a compilar a lenguaje nativo.
La primera compilación que sucede del código fuente al lenguaje intermedio se llama compilación estática y la segunda, que es la que hace el CLR se llama compilación just-in-time, que sucede en tiempo de ejecución.

Tiempo de Compilación

Se denomina tiempo de compilación (compile-time) al intervalo de tiempo en el que un compilador compila código escrito en un lenguaje de programación a una forma de código ejecutable por una máquina.
El compilador normalmente realiza un chequeo de sintaxis y una optimización del código generado.
El tiempo de compilación no sucede en los lenguajes interpretados debido a que estos no necesitan compilarse.

Tiempo de Ejecución

Se denomina **tiempo de ejecución (*runtime*)** al intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo. Este tiempo se inicia con la puesta en memoria principal del programa, por lo que el sistema operativo comienza a ejecutar sus instrucciones. El intervalo finaliza en el momento en que este envía al sistema operativo la señal de terminación, sea esta una terminación normal, en que el programa tuvo la posibilidad de concluir sus instrucciones satisfactoriamente, o una terminación anormal, en el que el programa produjo algún error y el sistema debió forzar su finalización.

Conceptos clave

- Archivos con lenguaje intermedio (.exe, .dll)
- Tiempo de ejecución y tiempo de compilación.

Introducción a C#

Características de C#

- **Compilación Hibrida:** No es un compilado directo a un lenguaje nativo, no es un lenguaje interpretado (No es un lenguaje cuyo código fuente es interpretado línea a línea el código fuente y lo va a ir ejecutando línea a línea). **C# primero compila a lenguaje intermedio y en tiempo de ejecución el CLR va a compilar a lenguaje nativo.** Pros: Capacidad de desarrollar sin pensar donde se va a ejecutar. Contra: Retraso al ejecutar el programa
- **Orientado a objetos:** Se enfoca en el paradigma que se basa en las relaciones entre objetos y los mensajes que se envían entre sí, sus características y comportamiento.
- **Orientado a Componentes:** Me permite desarrollar unas bibliotecas de clases y utilizarla o reutilizarla en una aplicación de consola, Windows form, pagina web, etc.
- **Seguridad de tipos (tipado estático):** C# es un lenguaje principalmente de tipado estático. Sus tipos de datos son definidos.
- **Garbage collection:** Administra la memoria desatendida. La asignación de memoria la va a realizar el CLR, un módulo del CLR es el garbage collector. El programador no debe estar pendiente en cómo se libera o se utiliza la memoria. Se encarga del segmento HEAP de memoria.
- **Sistema de tipos unificado:** Todos los tipos con los que trabajemos (primitivos, propios, de .NET, etc) todos se heredan o derivan de una clase pobre que se llama object. Esto significa que todos los tipos tienen una serie de métodos en común que se van a compartir
- **Case sensitive:** La misma variable, pero escrita con mayúsculas o minúsculas no es la misma.

Common Type Systema (CTS)

Todos los lenguajes de .NET tienen que cumplir esta especificación.

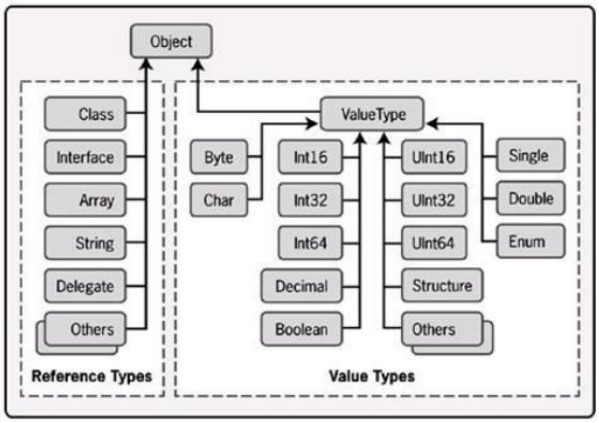
- Define un conjunto común de “tipos” de datos orientados a objetos.
- Todo lenguaje de programación .NET debe implementar los tipos definidos por el CTS.
- Todo tipo hereda directa o indirectamente del tipo System.Object.
- El CTS define tipos de VALOR y de REFERENCIA

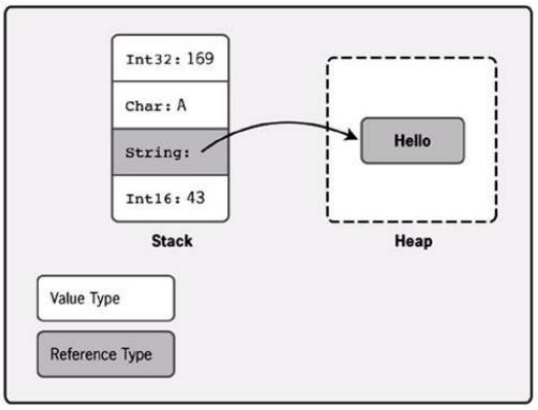
Common Type System (CTS)

UTNFra Avellaneda

- Definición Common Type System.
- Tipos de valor y referencia. *null.*
- Categorías de tipos.
- Aliases

- Literales y declaración
- Tipos char, string y bool
- Numéricos. *sizeOf(). MinValue. MaxValue.*
- Valores por defecto





Tipos de Datos

- Las variables escalares son constantes o variable que contiene un dato atómico y unidimensional.
- Las variables no escalares son array (vector), lista y objeto, que pueden tener almacenado en su estructura más de un valor.

Enteros: 0 (cero)

Punto flotante: 0 (cero)

Lógicos: False

Referencias: Null

Conversiones de tipos de datos

Implícitas: No interviene el programador (no requieren casteo). No deberían implicar pérdida de datos.

Explicitas: Interviene el programador (se quiere un casteo). Podrían implicar pérdida de datos.

Entry Point

El punto de entrada para los programas en C# es la función Main, es el primer método en ejecutarse.

- **static:** Es un modificador que permite ejecutar un método sin tener que instanciar a una variable (sin crear un objeto). El método Main() debe ser estático.
- **void:** Indica el tipo de valor de retorno del método Main(). No necesariamente tiene que ser void.
- **string [] args:** Es un Array de tipo string que puede recibir el método Main() como parámetro. Este parámetro es opcional.

Otros datos

Console

- Es una clase pública y estática.
- Representa la entrada, salida y errores de Streams para aplicaciones de consola.
- Es miembro del NameSpace System.

Métodos

- **ReadLine():** Lee la siguiente línea de caracteres de la consola. Devuelve un string. Equivalente a gets() de C.
- **Write():** Escribe el string que se le pasa como parámetro a la salida estándar. Equivalente a printf() de C.
- **WriteLine():** Ídem método Write, pero introduce un salto de línea al final de la cadena.

Propiedades

- **BackGroundColor**: Obtiene o establece el color de fondo de la consola.
- **ForeGroundColor:s**Obtiene o establece el color del texto de la consola
- **Title:** Obtiene o establece el título de la consola.

Formato de Salida de Texto

Fomato completo: { N [, M][: Formato] } (*)

- N será el número del parámetro, empezando por cero.
- M será el ancho usado para mostrar el parámetro, el cual se rellenará con espacios. Si M es negativo, se justificará a la izquierda, y si es positivo, se justificará a la derecha.
- Formato será una cadena que indicará un formato extra a usar con ese parámetro.

Tipo object

Todos los tipos de datos en .NET derivan de un tipo de dato padre, la clase System.Object. object es un alias de System.Object.
Esto nos permite almacenar en una variable de tipo object cualquier valor.

Tipo dynamic

dynamic es otro tipo especial que también puede almacenar cualquier valor. La diferencia con object radica en que nos permite utilizar los atributos y métodos del valor almacenado sin necesidad de un casteo.