Programación II

Table of contents:

Programa

Primer parcial

Clase 01 - Introducción a .NET y C#

Características de .NET. - Entorno de ejecución. - Common Language Runtime. - Características de C#. - Proceso de compilación de C#. - Tiempos de compilación y ejecución. - Entry point. - Tipos de referencia y de valor. - Variables escalares y no escalares.

Clase 02 - Manejo de fechas, matrices y strings

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 03 - Programación orientada a objetos

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 04 - Clases y miembros estáticos

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 05 - Sobrecarga

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 06 - Windows Forms

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 07 - Colecciones

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 08 - Encapsulamiento, propiedades e indexadores.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 09 - Herencia.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 10 - Polimorfismo.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 11 - Clases y métodos abstractos.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Segundo parcial

Clase 15 - Excepciones.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 16 - Pruebas unitarias.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 17 - Tipos Genéricos.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 18 - Interfaces.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 19 - Archivos y Serialización.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 20 - Introducción a SQL.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 21 - Conexión a bases de datos.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 22 - Delegados y expresiones lambda.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 23 - Programación multi-hilo y concurrencia.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 24 - Eventos.

> *Clase Math. - Uso de DateTime. - Strings. Métodos principales. Template strings. Clase StringBuilder. - Matrices y arrays.*

Clase 25 - Herramientas adicionales de C#.

> *Métodos de extensión. - Auto-properties. - Parámetros opcionales. -*

Clase Extra - Introducción a la programación web.

> *Métodos de extensión. - Auto-properties. - Parámetros opcionales. -*

*Last updated on **7/24/2021** by **mauriciocerizza***

Antes de empezar

Apuntes

Ejercicios prácticos

Todas las secciones cuentan con ejercicios de resolución práctica como ejercitación para el parcial de Laboratorio de Computación II y los trabajos prácticos.

Los mismos se encuentran categorizados según la siguiente tabla:

Categorías	Prefijo	Descripción
Introductorio	I##	Se enfocan en el contenido de la sección, no dependen de ejercicios previos y vienen acompañados de un diagrama de clase o descripción paso a paso de lo que hay que hacer.
Análisis funcional	A##	Plantean un requerimiento funcional y no especifican en profundidad cómo resolverlo. Sirven para ejercitar la creatividad junto con las capacidades de análisis y diseño.
Corrección de errores	E##	Son ejercicios de detección y corrección de errores, una tarea habitual en el ambiente profesional.
Consolidación	C##	Sirven para ejercitar el tema de la sección en conjunto con los aprendidos en clases anteriores. Pueden tener dependencias con ejercicios previos.
Investigación	R##	Para resolverlos se requerirá investigar contenido no abarcado por el temario. Son desafiantes y refuerzan las habilidades de búsqueda de soluciones en internet, aprendizaje auto-didacta e interpretación de documentación.

Recomendaciones sobre el uso de los ejercicios prácticos

- Consulte a su docente cuáles ejercicios le recomienda realizar y en qué orden.
- Cuantos más ejercicios haga, más aprenderá. Lleve la guía al día.
- Ante una duda o problema primero investigue en internet y trate de resolverlo por su cuenta. Como segunda instancia puede preguntar a un docente o en el canal de Slack, detallando cuál es la situación y qué intentó hacer previamente.

Cuestionarios

Todas las secciones cuentan con un cuestionario para reforzar lo aprendido y como ejercitación para el parcial teórico, recomendamos ir realizando los cuestionarios semana a semana. En dicha sección también encontrará referencias a bibliografía de consulta.

Recomendaciones sobre el uso de los cuestionarios

1. Repase la clase en cuestión.
2. Realice la ejercitación práctica.
3. Estudie como si fuera a dar un parcial sobre ese tema específico.
4. Responda el cuestionario tratando de no ayudarse con ninguna otra fuente que sus propios conocimientos.
5. Si lo intentó y aun así no puede responder la pregunta, puede ayudarse del material teórico o investigar.
6. Verifique sus respuestas contrastando con los apuntes y el material teórico.
7. Refuerce sus conocimientos y respuestas en clase. De ser necesario, consulte sus dudas con el profesor.
8. Repase lo aprendido antes del parcial.

Indice

En esta sección introduciremos algunas formas de trabajar en C# con procesos que se ejecutan en simultáneo. Introduciremos los conceptos de concurrencia, hilos, programación paralela, entre otros.

Descargar PDF de la sección

Apuntes

Ejercicios

Consolidación

Bibliografía

- Cleary, S. (2019). *Concurrency in C# Cookbook* (2nd edition). O'Reilly.

Introducción a .NET

En esta asignatura haremos uso de la plataforma de desarrollo .NET y del lenguaje de programación C# para entender conceptos y prácticas que son comunes a muchas otras herramientas que se utilizan al desarrollar software.

.NET (*pronunciado como "dot net"*) es una plataforma gratuita y de código abierto que nos provee una serie de herramientas y programas para construir fácilmente una gran variedad de software, así como el entorno necesario para ejecutarlo sobre distintas arquitecturas y sistemas operativos.

Características de .NET

En esta sección se apuntará a introducir las principales características de la plataforma para que puedan alcanzar un entendimiento general sobre las herramientas que utilizaremos durante la cursada.

Multi-plataforma

Existió una época donde esta plataforma sólo nos permitía trabajar para Windows, pero esos tiempos quedaron muy atrás. Desde la salida de .NET Core en 2016, podemos implementar nuestros sitios web, aplicaciones para servidores y programas de consola también en Linux y macOS.

Open Source

El código fuente de .NET es público y es mantenido por miles de desarrolladores y compañias. Es soportado por .NET foundation, una organización sin fines de lucro, la cual se encarga de promover el desarrollo y la colaboración alrededor del ecosistema de .NET.

Multi-lenguaje

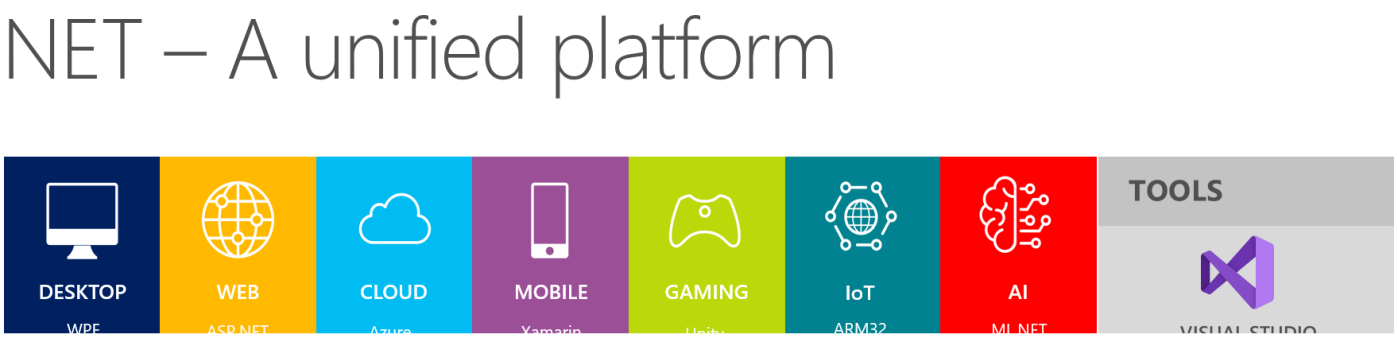
.NET admite varios lenguajes de programación, los cuales se pueden utilizar para programar sobre la plataforma:

C# → Lenguaje orientado a objetos con una sintaxis similar a C y JAVA.

F# → Lenguaje orientado principalmente a la programación funcional, de sintaxis liviana.

Visual Basic → La sintaxis de este lenguaje es la que más se asemeja al lenguaje humano (inglés), lo que facilita el trabajo para personas sin experiencia en el desarrollo de software.

Componentes de .NET



Runtime

Un runtime (entorno de ejecución, en español) es un programa encargado de ejecutar y administrar

Base Class Library

Cuando trabajamos con .NET contamos con

Frameworks

Herramientas

Lo que hace a una plataforma de desarrollo es que nos otorga todas las herramientas necesarias para llevar el diseño de un sistema a su implementación en la realidad.

Además de las antes mencionadas, .NET integra las siguientes herramientas:

- **.NET CLI** → Una interfaz de línea de comandos que nos provee una serie de instrucciones de consola que nos permitirán desarrollar, construir, ejecutar y públicar aplicaciones construidas con .NET.
- **Compiladores** para los lenguajes soportados.
- **MSBuild** → Un motor para cargar y construir nuestras aplicaciones.

- **NuGet** → Un administrador de paquetes desde donde podremos incorporar distintas librerías a nuestros proyectos, muchas de ellas desarrolladas por la comunidad.

Common Type System

El Common Type System (CTS) define un conjunto de tipos de datos común a todos los lenguajes soportados por .NET.

- Establece un marco de herramientas que habilita la ejecución de los distintos lenguajes de la plataforma.
- Provee un modelo orientado a objetos.
- Define un conjunto de reglas que todos los lenguajes deben seguir en lo que refiere a tipos.
- Provee una biblioteca que contiene los tipos primitivos básicos (Boolean, Int32, Byte, Char, etc).
- Define tipos de dato en dos categorías: de valor y de referencia.

Tipos de valor y tipos de referencia

Los **tipos de valor** son tipos de dato representados por su valor real. Si son asignados a una variable, esa variable obtendrá una nueva copia del valor.

Los **tipos de referencia**, al contrario, son tipos de dato representados por una referencia que apunta a un sector de memoria donde se encuentra el valor real. Si son asignados a una variable, esa variable almacenará la referencia y apuntará al valor original. No se realiza ninguna copia del valor. La referencia se almacena en

Categorías de tipos

.NET define cinco categorías de tipos de datos.

Categoría	Valor/Referencia	Descripción
Clases	Tipo de referencia	
Estructuras	Tipo de valor	
Enumerados	Tipo de valor	
Interfaces	Tipo de valor	

Categoría	Valor/Referencia	Descripción
Delegados	Tipo de referencia	

Entraremos en el detalle de cada una de estas categorías a lo largo de la cursada.

Introducción a C#

Durante nuestro viaje a través de las características comunes de los lenguajes de alto nivel y la programación orientada a objetos nos acompañará el lenguaje de programación C#.

Tal vez algunas de las características de este lenguaje les parezcan similares a otros lenguajes que conozcan, y están en lo correcto. El equipo de C#, desde sus inicios, no dudó en tomar grandes ideas de otros lenguajes y reformarlas para incorporarlas a C#. Las principales influencias han sido Java (sobre todo en los inicios), y más cerca en el tiempo el mismísimo F#.

¡Hello world!

Exploremos el entorno de trabajo y pongamos en ejecución nuestra primera aplicación de consola.

Visual Studio

Podríamos usar Notepad o algún otro editor de texto para escribir código C#, pero Visual Studio es una mejor opción ya que es un IDE.

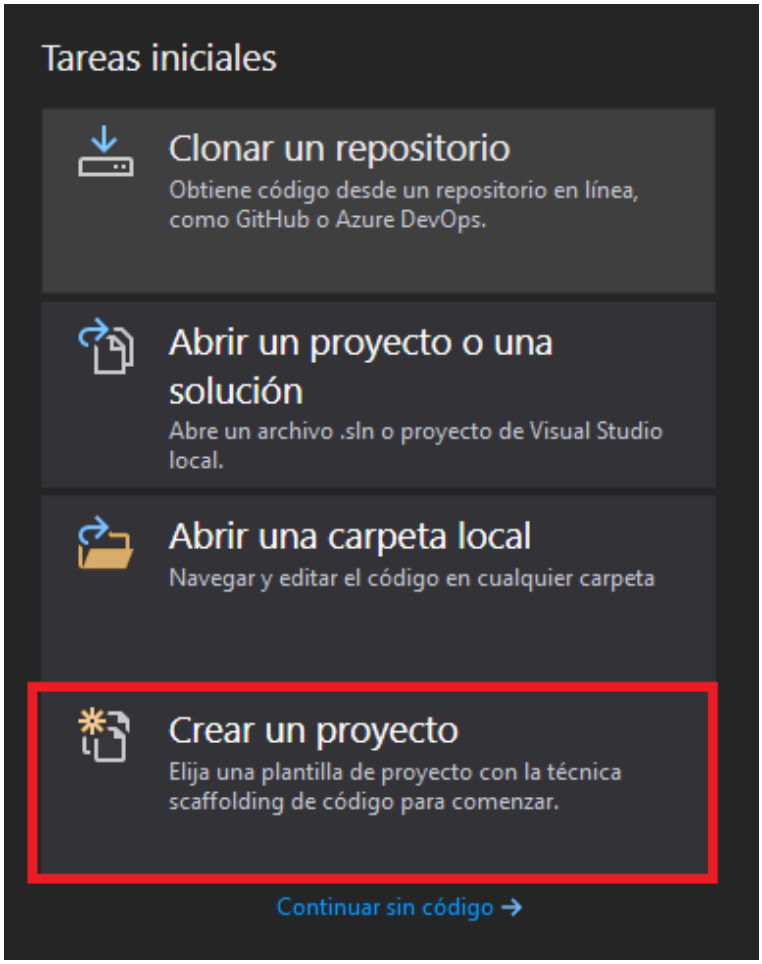
IDE es un acrónimo para **Integrated Development Environment** (Entorno de Desarrollo Integrado, en español), una aplicación que proporciona todos los servicios y herramientas necesarios para desarrollar software (editor de texto, diseñador visual, administrador de archivos, debugger, entre otras).

> **NOTA**
>
> NO confundir con ***Visual Studio Code***, el cual es un excelente **editor de código** open source, gratuito y multiplataforma.
>
> Si bien se puede programar con C# en editores de código, Visual Studio (*no Code*) es más completo y especializado en el desarrollo sobre la plataforma .NET.

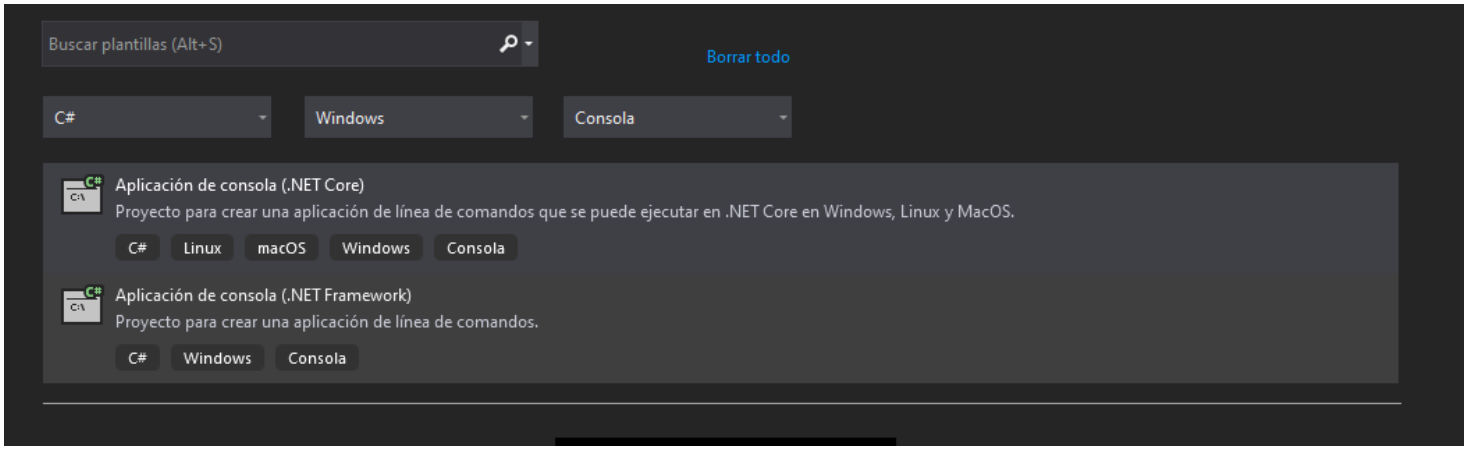
Instalación

Lo primero que tendremos que hacer es instalar Visual Studio Community Edition, si es que aún no lo tenemos. Se puede descargar tanto para Windows como para Mac. Aún no existe una versión para Linux.

Una vez instalado y ejecutado, nos mostrará la siguiente pantalla con acciones rápidas para arrancar:

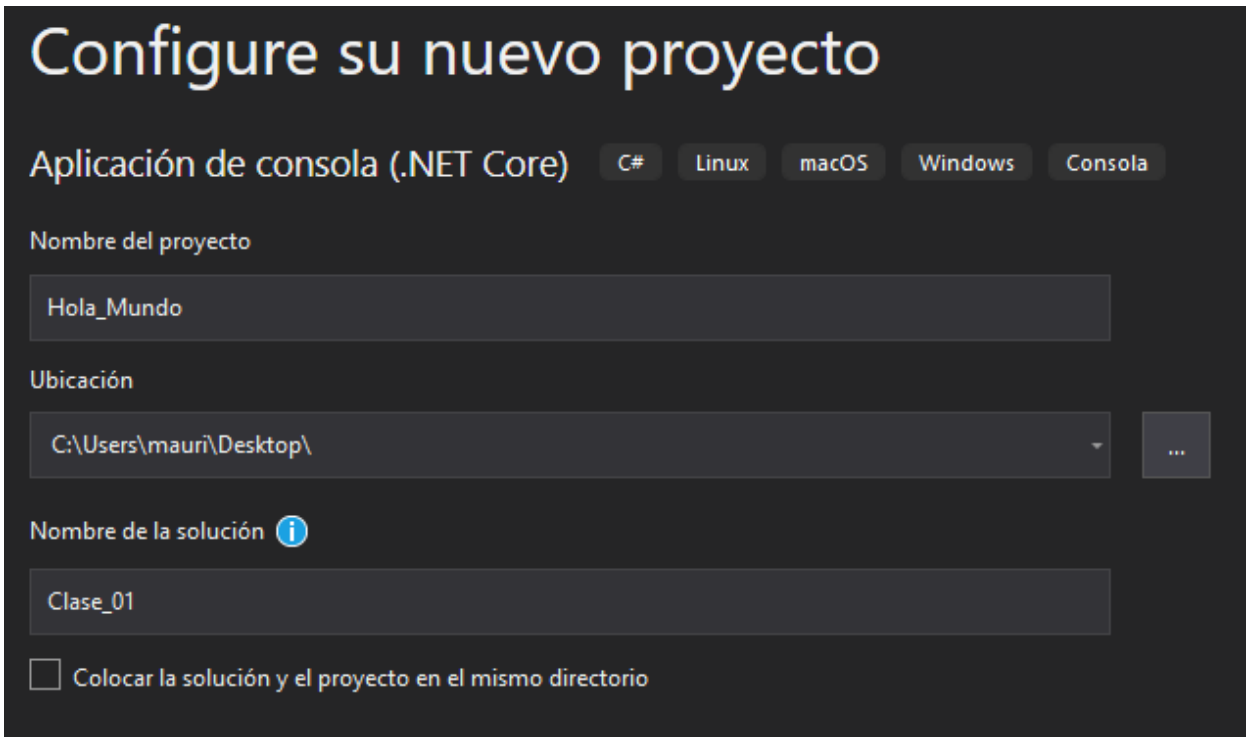


Elegiremos "Crear un proyecto". Lo siguiente que encontraremos es una serie de plantillas o templates de los distintos tipos de proyectos que podemos encarar con la plataforma. Elegiremos, para arrancar, "Aplicación de consola". *Asegúrense de consultar con su docente la versión indicada para la cursada.*



Lo siguiente será elegir un nombre para el proyecto y otro para la solución (o el mismo) y presionar el botón "Crear".

Un ***proyecto*** es una estructura que nos permitirá construir nuestros programas con .NET y compilarlos como una unidad independiente, mientras que a una ***solución*** la podemos ver como una agrupadora de proyectos. Normalmente un programa de .NET está compuesto por varios proyectos inter-relacionados dentro de una solución.



Módulos de Visual Studio

Como podemos observar rápidamente, Visual Studio se presenta como una interfaz de única página (SPI) compuesta por múltiples ventanas anidadas. Está de más decir que podremos administrar estas ventanas a gusto, todas soportan funcionalidades básicas como mover, cambiar el tamaño, ocultar y fijar.

Hagamos un repaso rápido de cada sección.

Barra lateral derecha

Tipado estático y tipado dinámico

Recordemos la diferencia entre tipado estático y tipado dinámico.

Tipado estático → Decimos que un lenguaje es de tipado estático, porque los tipos tienen que definirse en tiempo de compilación para que el programa funcione.

Proceso de compilación

Conclusión

Cuestionario

1. Enumere y describa las características de C#.

2. ¿Qué funciones cumple el entorno de ejecución o runtime?
3. Explique y compare tiempo de compilación y tiempo de ejecución.
4. Describa el proceso de compilación de C#.
5. ¿Qué es el Common Type System (CTS)?
6. Explique las diferencias entre los tipos por referencia y los tipos por valor.
7. Explique las diferencias entre variables escalares y no escalares. Ejemplifique.
8. ¿Qué es un “entry point”? ¿Cuál es el entry point de los programas construidos con C#?

Bibliografía

- “.NET Architectural Components.” Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/standard/components.
- "Common Type System & Common Language Specification." Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/standard/common-type-system

On this page

Cuestionario y bibliografía

Cuestionario

1. Enumere y describa las características de C#.
2. ¿Qué funciones cumple el entorno de ejecución o runtime?
3. Explique y compare tiempo de compilación y tiempo de ejecución.
4. Describa el proceso de compilación de C#.
5. ¿Qué es el Common Type System (CTS)?
6. Explique las diferencias entre los tipos por referencia y los tipos por valor.
7. Explique las diferencias entre variables escalares y no escalares. Ejemplifique.
8. ¿Qué es un “entry point”? ¿Cuál es el entry point de los programas construidos con C#?

Bibliografía

- “.NET Architectural Components.” Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/standard/components.
- "Common Type System & Common Language Specification." Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/standard/common-type-system

Créditos

Autor/es: Prof. Mauricio Cerizza **Revisor/es: Fecha de la última revisión:**

Introducción a .NET

En esta asignatura haremos uso de la plataforma de desarrollo .NET y del lenguaje de programación C# para entender conceptos y prácticas que son comunes a muchas otras herramientas que se utilizan al desarrollar software.

Introducción a .NET

.NET (*pronunciado como "dot net"*) es una plataforma gratuita y de código abierto que nos provee una serie de herramientas y programas para construir fácilmente una gran variedad de software, así como el entorno necesario para ejecutarlo sobre distintas arquitecturas y sistemas operativos.

Características de .NET

En esta sección se apuntará a introducir las principales características de la plataforma para que puedan alcanzar un entendimiento general sobre las herramientas que utilizaremos durante la cursada.

Multi-plataforma

Existió una época donde esta plataforma sólo nos permitía trabajar para Windows, pero esos tiempos quedaron muy atrás. Desde la salida de .NET Core en 2016, podemos implementar nuestros sitios web, aplicaciones para servidores y programas de consola también en Linux y macOS.

Open Source

El código fuente de .NET es público y es mantenido por miles de desarrolladores y compañias. Es soportado por .NET foundation, una organización sin fines de lucro, la cual se encarga de promover el desarrollo y la colaboración alrededor del ecosistema de .NET.

Multi-lenguaje

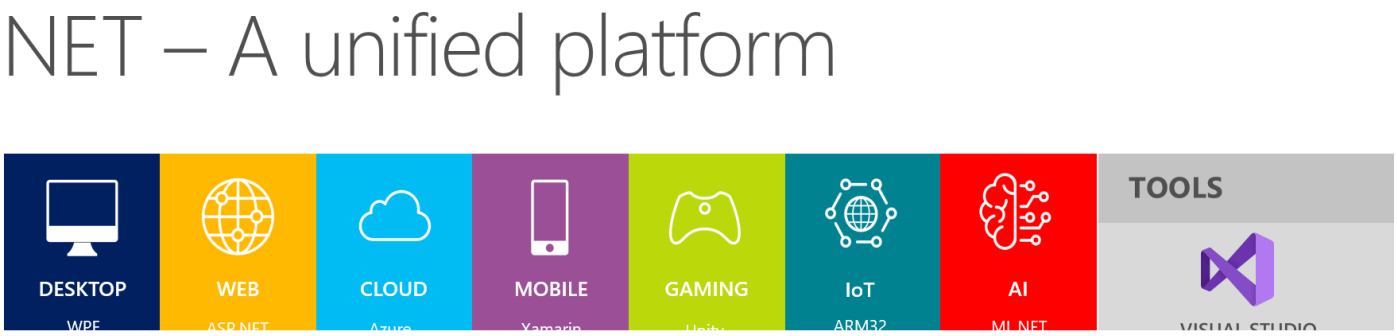
.NET admite varios lenguajes de programación, los cuales se pueden utilizar para programar sobre la plataforma:

C# → Lenguaje orientado a objetos con una sintaxis similar a C y JAVA.

F# → Lenguaje orientado principalmente a la programación funcional, de sintaxis liviana.

Visual Basic → La sintaxis de este lenguaje es la que más se asemeja al lenguaje humano (inglés), lo que facilita el trabajo para personas sin experiencia en el desarrollo de software.

Componentes de .NET



Runtime

Un runtime (entorno de ejecución, en español) es un programa encargado de ejecutar y administrar

Base Class Library

Cuando trabajamos con .NET contamos con

Frameworks

Herramientas

Lo que hace a una plataforma de desarrollo es que nos otorga todas las herramientas necesarias para llevar el diseño de un sistema a su implementación en la realidad.

Además de las antes mencionadas, .NET integra las siguientes herramientas:

- **.NET CLI** → Una interfaz de línea de comandos que nos provee una serie de instrucciones de consola que nos permitirán desarrollar, construir, ejecutar y públicar aplicaciones construidas con .NET.
- **Compiladores** para los lenguajes soportados.
- **MSBuild** → Un motor para cargar y construir nuestras aplicaciones.

- **NuGet** → Un administrador de paquetes desde donde podremos incorporar distintas librerías a nuestros proyectos, muchas de ellas desarrolladas por la comunidad.

Introducción a C#

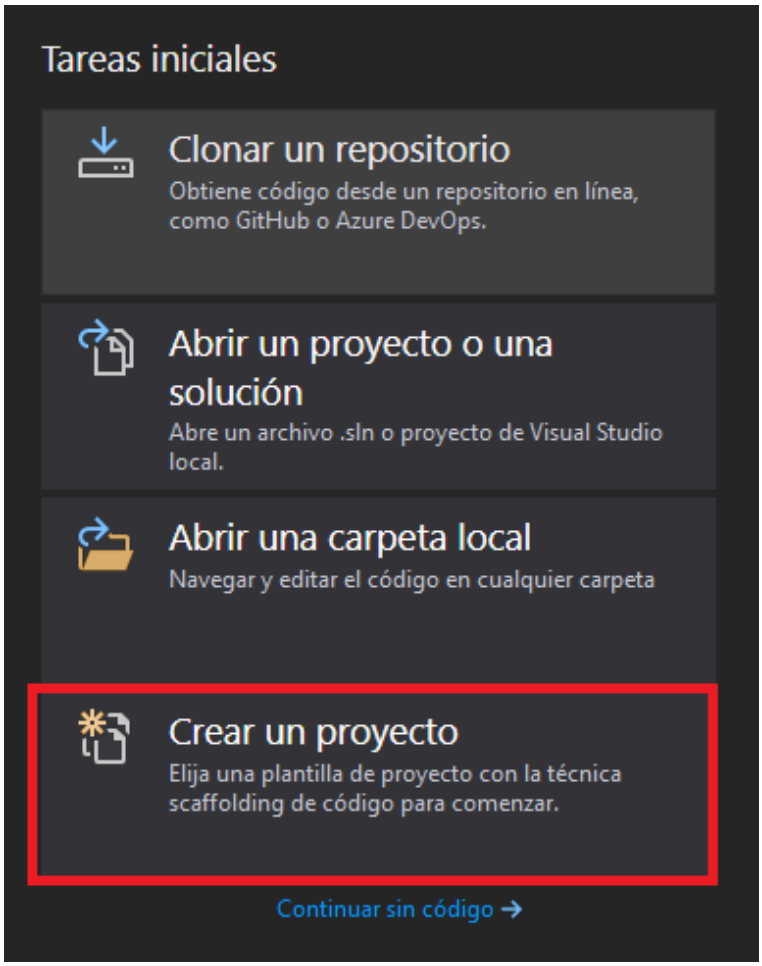
Durante nuestro viaje a través de las características comunes de los lenguajes de alto nivel y la programación orientada a objetos nos acompañará el lenguaje de programación C#.

Tal vez algunas de las características de este lenguaje les parezcan similares a otros lenguajes que conozcan, y están en lo correcto. El equipo de C#, desde sus inicios, no dudo en tomar grandes ideas de otros lenguajes y reformarlas para incorporarlas a C#. Las principales influencias han sido Java (sobre todo en los inicios), y más cerca en el tiempo el mismísimo F#.

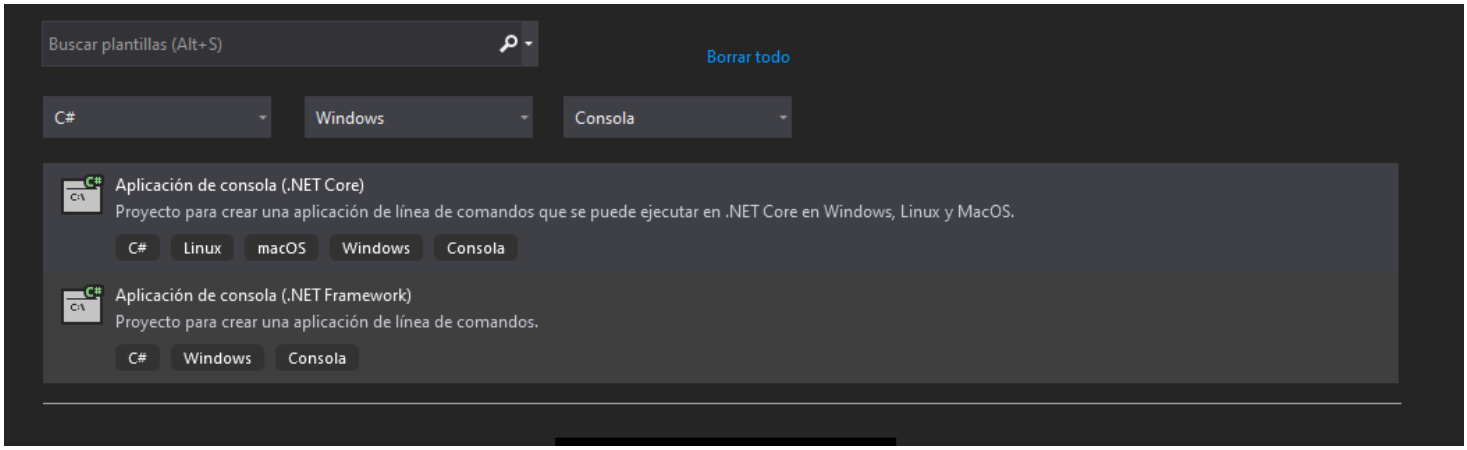
¡Hello world!

Exploremos el entorno de trabajo y pongamos en ejecución nuestra primera aplicación de consola.

Lo primero que tendremos que hacer es instalar el IDE Visual Studio Community, si es que aún no lo tenemos. Una vez instalado y ejecutado, nos mostrará la siguiente pantalla con acciones rápidas para arrancar:

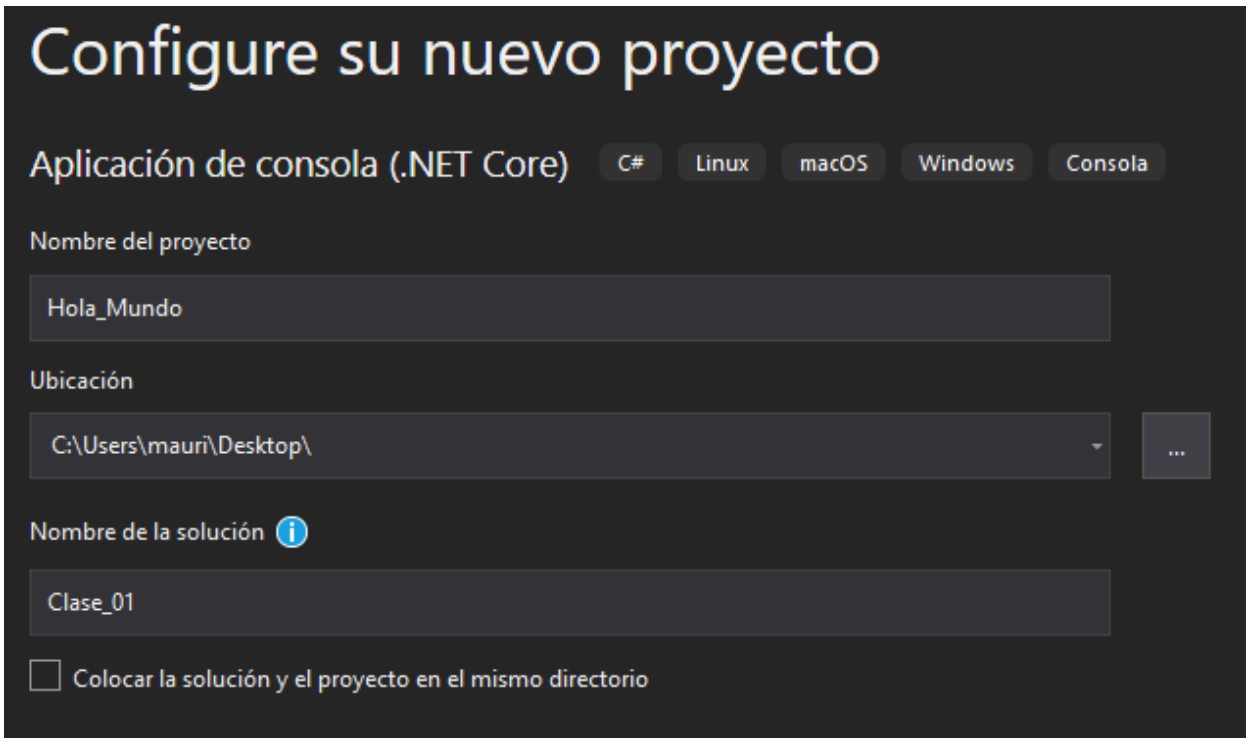


Elegiremos "Crear un proyecto". Lo siguiente que encontraremos es una serie de plantillas o templates de los distintos tipos de proyectos que podemos encarar con la plataforma. Elegiremos, para arrancar, "Aplicación de consola". *Asegúrense de consultar con su docente la versión indicada para la cursada.*



Lo siguiente será elegir un nombre para el proyecto y otro para la solución (o el mismo) y presionar el botón "Crear".

Un ***proyecto*** es una estructura que nos permitirá construir nuestros programas con .NET y compilarlos como una unidad independiente, mientras que a una ***solución*** la podemos ver como una agrupadora de proyectos. Normalmente un programa de .NET está compuesto por varios proyectos inter-relacionados dentro de una solución.



Módulos de Visual Studio

Como podemos observar rápidamente, Visual Studio se presenta como una interfaz de única página (SPI) compuesta por múltiples ventanas anidadas. Está de más decir que podremos administrar estas ventanas a gusto, todas soportan funcionalidades básicas como mover, cambiar el tamaño, ocultar y fijar.

Hagamos un repaso rápido de cada sección.

Barra lateral derecha

Common Type System

El Common Type System (CTS) define un conjunto de tipos de datos común a todos los lenguajes soportados por .NET.

- Establece un marco de herramientas que habilita la ejecución de los distintos lenguajes de la plataforma.
- Provee un modelo orientado a objetos.
- Define un conjunto de reglas que todos los lenguajes deben seguir en lo que refiere a tipos.
- Provee una biblioteca que contiene los tipos primitivos básicos (Boolean, Byte, Char, etc).

Tipado estático y tipado dinámico

Recordemos la diferencia entre tipado estático y tipado dinámico.

Tipado estático → Decimos que un lenguaje es de tipado estático, porque los tipos tienen que definirse en tiempo de compilación para que el programa funcione.

Proceso de compilación

Conclusión

Cuestionario

1. Enumere y describa las características de C#.
2. ¿Qué funciones cumple el entorno de ejecución o runtime?
3. Explique y compare tiempo de compilación y tiempo de ejecución.
4. Describa el proceso de compilación de C#.
5. ¿Qué es el Common Type System (CTS)?
6. Explique las diferencias entre los tipos por referencia y los tipos por valor.
7. Explique las diferencias entre variables escalares y no escalares. Ejemplifique.
8. ¿Qué es un “entry point”? ¿Cuál es el entry point de los programas construidos con C#?

Bibliografía

- ".NET Architectural Components." Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/standard/components.
- "Common Type System & Common Language Specification." Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/standard/common-type-system

*Last updated on **7/24/2021** by **mauriciocerizza***

Matrices

Las matrices, también llamadas arrays, arreglos o vectores, son estructuras de datos que nos permiten almacenar múltiples variables **del mismo tipo**.

Características de las matrices

Dimensionalidad

Las matrices pueden ser **unidimensionales**, **bidimensionales** o **multidimensionales**.

Las matrices unidimensionales tienen una sola dimensión. , es decir, tener una sola dimensión. multidimensionales.

Tamaño fijo

La cantidad y el tamaño de las dimensiones son establecidos al momento de instanciar la matriz **y no pueden ser cambiados**. Si necesitamos alterar el tamaño de un array, tendremos que instanciar uno nuevo con distinta longitud y migrar los datos almacenados en el array original.

Indexación base-cero

Se puede acceder a los elementos de una matriz a través de un índice numérico entero. En el caso de las matrices con más de una dimensión será un sub-índice por dimensión, y se accederá a un elemento específico con una determinada combinación de los distintos sub-índices.

Se dice que las matrices tienen indexación de base-cero (zero indexed) porque sus elementos pueden ser accedidos a través de un índice numérico que comienza en el número cero y se incrementa de uno en uno.

Si tenemos un array con tamaño ***n***, su primera posición se accederá con el ***índice 0*** y la última con el ***índice n-1*** (el tamaño menos uno).

Seguridad de tipos

Los elementos de una matriz pueden ser de cualquier tipo, incluyendo el tipo Array.

Todos los elementos de una matriz son inicializados en su valor por defecto.

Las matrices tienen indexación base-cero (zero indexed). Sus elementos pueden ser accedidos a través de un índice numérico que comienza en el número cero y se incrementa de uno en uno.

Todos los elementos de una matriz son inicializados en su valor por defecto. Los elementos pueden ser de cualquier tipo, incluyendo el tipo Array.

Multi-plataforma

Existió una época donde esta plataforma sólo nos permitía trabajar para Windows, pero esos tiempos quedaron muy atrás. Desde la salida de .NET Core en 2016, podemos implementar nuestros sitios web, aplicaciones para servidores y programas de consola también en Linux y macOS.

Open Source

El código fuente de .NET es público y es mantenido por miles de desarrolladores y compañias. Es soportado por .NET foundation, una organización sin fines de lucro, la cual se encarga de promover el desarrollo y la colaboración alrededor del ecosistema de .NET.

Multi-lenguaje

.NET admite varios lenguajes de programación, los cuales se pueden utilizar para programar sobre la plataforma:

C# → Lenguaje orientado a objetos con una sintaxis similar a C y JAVA.

F# → Lenguaje orientado principalmente a la programación funcional, de sintaxis liviana.

Visual Basic → La sintaxis de este lenguaje es la que más se asemeja al lenguaje humano (inglés), lo que facilita el trabajo para personas sin experiencia en el desarrollo de software.

Componentes de .NET

.NET – A unified platform

Runtime

Un runtime (entorno de ejecución, en español) es un programa encargado de ejecutar y administrar

Base Class Library

Cuando trabajamos con .NET contamos con

Frameworks

Herramientas

Lo que hace a una plataforma de desarrollo es que nos otorga todas las herramientas necesarias para llevar el diseño de un sistema a su implementación en la realidad.

Además de las antes mencionadas, .NET integra las siguientes herramientas:

- **.NET CLI** → Una interfaz de línea de comandos que nos provee una serie de instrucciones de consola que nos permitirán desarrollar, construir, ejecutar y públicar aplicaciones construidas con .NET.
- **Compiladores** para los lenguajes soportados.
- **MSBuild** → Un motor para cargar y construir nuestras aplicaciones.
- **NuGet** → Un administrador de paquetes desde donde podremos incorporar distintas librerías a nuestros proyectos, muchas de ellas desarrolladas por la comunidad.

Common Type System

El Common Type System (CTS) define un conjunto de tipos de datos común a todos los lenguajes soportados por .NET.

- Establece un marco de herramientas que habilita la ejecución de los distintos lenguajes de la plataforma.
- Provee un modelo orientado a objetos.
- Define un conjunto de reglas que todos los lenguajes deben seguir en lo que refiere a tipos.
- Provee una biblioteca que contiene los tipos primitivos básicos (Boolean, Int32, Byte, Char, etc).
- Define tipos de dato en dos categorías: de valor y de referencia.

Tipos de valor y tipos de referencia

Los **tipos de valor** son tipos de dato representados por su valor real. Si son asignados a una variable, esa variable obtendrá una nueva copia del valor.

Los **tipos de referencia**, al contrario, son tipos de dato representados por una referencia que apunta a un sector de memoria donde se encuentra el valor real. Si son asignados a una variable, esa variable almacenará la referencia y apuntará al valor original. No se realiza ninguna copia del valor. La referencia se almacena en

Categorías de tipos

.NET define cinco categorías de tipos de datos.

Categoría	Valor/Referencia	Descripción
Clases	Tipo de referencia	
Estructuras	Tipo de valor	
Enumerados	Tipo de valor	
Interfaces	Tipo de valor	
Delegados	Tipo de referencia	

Entraremos en el detalle de cada una de estas categorías a lo largo de la cursada.

*Last updated on **7/24/2021** by **mauriciocerizza***

On this page

Colecciones

Los sistemas de hoy manejan grandes cantidades de datos y por eso requerimos herramientas poderosas para trabajar con ellos. Es ahí cuando las colecciones entran en acción.

Las colecciones son **objetos** especializados en almacenar, organizar y administrar una gran cantidad de datos.

En esta asignatura haremos uso de la plataforma de desarrollo .NET y del lenguaje de programación C# para entender conceptos y prácticas que son comunes a muchas otras herramientas que se utilizan al desarrollar software.

.NET (*pronunciado como "dot net"*) es una plataforma gratuita y de código abierto que nos provee una serie de herramientas y programas para construir fácilmente una gran variedad de software, así como el entorno necesario para ejecutarlo sobre distintas arquitecturas y sistemas operativos.

Características de .NET

En esta sección se apuntará a introducir las principales características de la plataforma para que puedan alcanzar un entendimiento general sobre las herramientas que utilizaremos durante la cursada.

Multi-plataforma

Existió una época donde esta plataforma sólo nos permitía trabajar para Windows, pero esos tiempos quedaron muy atrás. Desde la salida de .NET Core en 2016, podemos implementar nuestros sitios web, aplicaciones para servidores y programas de consola también en Linux y macOS.

Open Source

El código fuente de .NET es público y es mantenido por miles de desarrolladores y compañias. Es soportado por .NET foundation, una organización sin fines de lucro, la cual se encarga de promover el desarrollo y la colaboración alrededor del ecosistema de .NET.

Multi-lenguaje

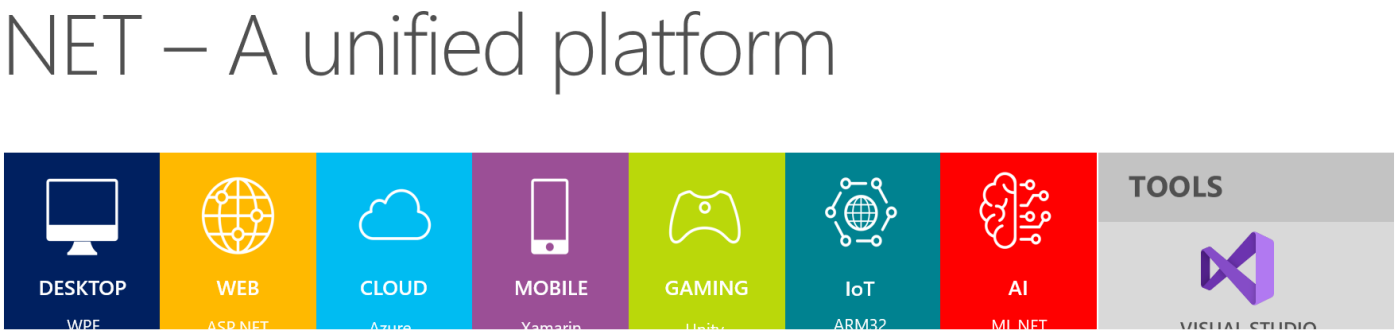
.NET admite varios lenguajes de programación, los cuales se pueden utilizar para programar sobre la plataforma:

C# → Lenguaje orientado a objetos con una sintaxis similar a C y JAVA.

F# → Lenguaje orientado principalmente a la programación funcional, de sintaxis liviana.

Visual Basic → La sintaxis de este lenguaje es la que más se asemeja al lenguaje humano (inglés), lo que facilita el trabajo para personas sin experiencia en el desarrollo de software.

Componentes de .NET



Runtime

Un runtime (entorno de ejecución, en español) es un programa encargado de ejecutar y administrar

Base Class Library

Cuando trabajamos con .NET contamos con

Frameworks

Herramientas

Lo que hace a una plataforma de desarrollo es que nos otorga todas las herramientas necesarias para llevar el diseño de un sistema a su implementación en la realidad.

Además de las antes mencionadas, .NET integra las siguientes herramientas:

- **.NET CLI** → Una interfaz de línea de comandos que nos provee una serie de instrucciones de consola que nos permitirán desarrollar, construir, ejecutar y públicar aplicaciones construidas con .NET.
- **Compiladores** para los lenguajes soportados.
- **MSBuild** → Un motor para cargar y construir nuestras aplicaciones.
- **NuGet** → Un administrador de paquetes desde donde podremos incorporar distintas librerías a nuestros proyectos, muchas de ellas desarrolladas por la comunidad.

Common Type System

El Common Type System (CTS) define un conjunto de tipos de datos común a todos los lenguajes soportados por .NET.

- Establece un marco de herramientas que habilita la ejecución de los distintos lenguajes de la plataforma.
- Provee un modelo orientado a objetos.
- Define un conjunto de reglas que todos los lenguajes deben seguir en lo que refiere a tipos.
- Provee una biblioteca que contiene los tipos primitivos básicos (Boolean, Int32, Byte, Char, etc).
- Define tipos de dato en dos categorías: de valor y de referencia.

Tipos de valor y tipos de referencia

Los **tipos de valor** son tipos de dato representados por su valor real. Si son asignados a una variable, esa variable obtendrá una nueva copia del valor.

Los **tipos de referencia**, al contrario, son tipos de dato representados por una referencia que apunta a un sector de memoria donde se encuentra el valor real. Si son asignados a una variable, esa variable almacenará la referencia y apuntará al valor original. No se realiza ninguna copia del valor. La referencia se almacena en

Categorías de tipos

.NET define cinco categorías de tipos de datos.

Categoría	Valor/Referencia	Descripción
Clases	Tipo de referencia	
Estructuras	Tipo de valor	

Categoría	Valor/Referencia	Descripción
Enumerados	Tipo de valor	
Interfaces	Tipo de valor	
Delegados	Tipo de referencia	

Entraremos en el detalle de cada una de estas categorías a lo largo de la cursada.

*Last updated on **7/24/2021** by **mauriciocerizza***

Cuestionario y bibliografía

Cuestionario

1. ¿El método Array.Resize cambia el tamaño de la instancia de array proveída o genera una nueva con distinto tamaño? ¿Por qué?
2. ¿Cuál es la diferencia entre colecciones y matrices?
3. ¿Cuál es la diferencia entre las colecciones genéricas y las no-genéricas?
4. ¿Cuál es la diferencia entre una cola (queue) y una pila (stack). Asocie con los conceptos "FIFO" y "LIFO".
5. Enumere y describa las siguientes colecciones: List, Dictionary, SortedList, Arraylist, Hashtable. Clasifíquelas en genéricas o no-genéricas.
6. ¿Por qué piensa que las colas y pilas no traen un método para ordenarlas? Piense en el uso de dichas colecciones.
7. ¿Cuál es la diferencia entre colas y pilas genéricas y no-genéricas?
8. ¿Cual es la salida del siguiente código?

Bibliografía

- “Arrays (C# Programming Guide)” Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/.

Créditos

Autor/es: Prof. Mauricio Cerizza **Revisor/es: Fecha de la última revisión:**

*Last updated on **7/24/2021** by **mauriciocerizza***

Indice

En esta sección introduciremos algunas formas de trabajar en C# con procesos que se ejecutan en simultáneo. Introduciremos los conceptos de concurrencia, hilos, programación paralela, entre otros.

Descargar PDF de la sección

Apuntes

Concurrencia

Cuestionario

Ejercicios

Consolidación

C01 - El relojero

C02 - Simulador de atención a clientes

C03 - Simulador de llamadas

Bibliografía

- Cleary, S. (2019). *Concurrency in C# Cookbook* (2nd edition). O'Reilly.

Concurrencia

¿Qué es concurrencia?

En programación hablamos de **concurrencia** cuando se ejecuta más de una tarea o proceso al mismo tiempo.

Esta habilidad es útil cuando necesitamos que una aplicación haga alguna cosa *mientras* está trabajando en algo más.

La concurrencia es un aspecto clave de las aplicaciones modernas, permite que:

- Los usuarios finales puedan interactuar con la interfaz de la aplicación de manera no-bloqueante.
- Un servidor pueda atender varias peticiones en simultáneo y no afectar los tiempos de respuesta ante periodos de alta demanda.
- Realizar tareas de computo complejas de manera más rápida y haciendo un uso más eficaz los recursos de la computadora.

En esta sección veremos las **dos formas de concurrencia** más comunes: **programación multi-hilo <*multithreaded programming*>** y **programación asincrónica <*asynchronous programming*>**.

Programación multi-hilo

La **programación multi-hilo <*multithreaded programming*>** es una forma de concurrencia

Un **hilo <*thread*>**, también llamado hebra o subproceso, es un ejecutor de tareas independiente.

Un **proceso** está compuesto por múltiples hilos y cada uno de esos hilos puede estar realizando una tarea distinta en paralelo. Todos los hilos de un mismo proceso comparten los mismos recursos del sistema operativo.

Cada hilo tiene una **pila de ejecución <*call stack*>** independiente, esto significa que cada uno maneja su propia secuencia de funciones a ejecutar.

En algunos tipos de aplicación existen hilos especiales, por ejemplo un hilo para la interfaz de usuario <*UI Thread*> o el hilo principal en los programas de consola <*Main Thread*>.

IMAGEN - proceso, hilos, call stack y memoria

Todas las aplicaciones de .NET tienen un **conjunto de hilos <*thread pool*>** que se encarga de mantener un número de hilos activos esperando para ejecutar cualquier trabajo que se requiera. Lo podemos ver como un lugar donde podemos poner en cola tareas a realizar y que se ajustará automáticamente de acuerdo a la demanda.

En .NET se solía utilizar la clase ***Thread*** para trabajar con hilos, la cual es una abstracción de bajo nivel. El *conjunto de hilos* es una abstracción de un nivel un poco más alto, ya que se encargará por si mismo de instanciar un hilo si existe la necesidad. Actualmente no se recomienda crear instancias de ***Thread*** ya que existen nuevas soluciones que fueron afinadas para cubrir de forma eficiente y sencilla la gran mayoría de los escenarios reales.

Las clases con las que trabajaremos son abstracciones de alto nivel que ponen en cola trabajo para que sea resuelto por el *conjunto de hilos*.

IMAGEN - thread vs thread pool vs high level abstractions

Programación en paralelo

La **programación en paralelo <*parallel programming*>** es un tipo de programación multi-hilo, que a su vez es una forma de concurrencia. Se utiliza cuando se necesita dividir una gran carga de trabajo computacional en partes independientes y ejecutarlas en paralelo, maximizando el uso de los núcleos de la CPU.

> **IMPORTANTE**
>
> Cuando procesamos en paralelo cada fragmento de trabajo de ser tan independiente del resto como sea posible.

IMAGEN - ejemplo en la cocina

Existen dos formas de paralelismo: data parallelism y task parallelism

IMAGEN - tipos de concurrencia y subtipos

https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming

Parallel.ForEach

Task.Run

Resumen de la sección

- ¿Qué significa concurrencia?
- ¿Qué formas de concurrencia existen y cuándo se usa cada una?

Cuestionario

Todas las secciones finalizan con un cuestionario para reforzar lo aprendido y como ejercitación para el parcial teórico, recomendamos ir realizando los cuestionarios semana a semana. En dicha sección también encontrará referencias a bibliografía de consulta.

Recomendaciones sobre el uso de los cuestionarios

1. Repase la clase en cuestión.
2. Realice la ejercitación práctica.
3. Estudie como si fuera a dar un parcial sobre ese tema específico.
4. Responda el cuestionario tratando de no ayudarse con ninguna otra fuente que sus propios conocimientos.
5. Si lo intentó y aun así no puede responder la pregunta, puede ayudarse del material teórico o investigar.
6. Verifique sus respuestas contrastando con los apuntes y el material teórico.
7. Refuerce sus conocimientos y respuestas con la puesta en común en clase. De ser necesario, consulte sus dudas con el profesor.
8. Repase lo aprendido antes del parcial.

Ejercicio C01 - El relojero

Consigna

Crear un proyecto de Windows Forms con un RichTextBox y un Label dentro.

1. Crear el método AsignarHora que se encargará de imprimir la hora en la Label lblHora.
2. En el Label se deberá mostrar la fecha y hora actual, con segundos incluidos, y refrescándose una vez por segundo.
3. Generar tres prácticas, independientes, en el orden planteado:
    i. Realizar la actualización de la hora 1 vez por segundo utilizando alguna estructura de control dada en clase.
    ii. Agregar un objeto del tipo Timer para refrescar la hora actual cada 1 segundo.
    iii. Resolver el mismo ejercicio utilizando hilos.

Negocio:

- Tendrá como atributos una lista de tipo String “clientes” y dos atributos de tipo Caja.
- Crear propiedades de lectura para todos sus atributos.
- El constructor recibirá por parámetro las dos cajas e inicializará la lista de clientes.
- El método AsignarCaja deberá imprimir el mensaje "Asignando cajas..." cuando sea invocado, recorrer la lista de clientes y asignar a cada cliente en la fila de la caja que menos clientes tenga en ese momento.
- La asignación de cada cliente a una caja tardará 1 segundo.

Main:

- La asignación de cada cliente a una caja tardará 1 segundo.
- Crear un thread para asignar las cajas a los clientes, uno para atender los clientes de la caja1 y otro para atender los clientes de la caja2. Los threads destinados a atender a los clientes deberán tener en su propiedad “Name” el nombre de la caja que está atendiendo.
- Se deberán iniciar los 3 threads uno a continuación del otro.
- Utilizar el método Join del objeto de la clase Thread para asegurar que se hayan asignado todos los clientes a alguna caja antes de comenzar a atender.

Diagrama de clases

Resolución

	**Video**		**Código**

Ejercicio C02 - Simulador de atención a clientes

Consigna

Crear un proyecto de tipo consola para simular la atención paralela de clientes en 2 cajas de un negocio.

Para ello se pide crear las siguientes clases:

Caja:

- Tendrá como único atributo una lista de tipo String "filaClientes".
- El constructor de la clase Caja deberá inicializar dicha lista.
- El método AtenderClientes deberá recorrer la fila de clientes e ir imprimiendo el nombre del cliente que se está atendiendo junto con el número de caja que será previamente seteado en la propiedad "Name" del thread.
- Por cada cliente que se atiende en cada caja se tardará 2 segundos.

Negocio:

- Tendrá como atributos una lista de tipo String “clientes” y dos atributos de tipo Caja.
- Crear propiedades de lectura para todos sus atributos.
- El constructor recibirá por parámetro las dos cajas e inicializará la lista de clientes.
- El método AsignarCaja deberá imprimir el mensaje "Asignando cajas..." cuando sea invocado, recorrer la lista de clientes y asignar a cada cliente en la fila de la caja que menos clientes tenga en ese momento.
- La asignación de cada cliente a una caja tardará 1 segundo.

Main:

- La asignación de cada cliente a una caja tardará 1 segundo.
- Crear un thread para asignar las cajas a los clientes, uno para atender los clientes de la caja1 y otro para atender los clientes de la caja2. Los threads destinados a atender a los clientes deberán tener en su propiedad “Name” el nombre de la caja que está atendiendo.
- Se deberán iniciar los 3 threads uno a continuación del otro.
- Utilizar el método Join del objeto de la clase Thread para asegurar que se hayan asignado todos los clientes a alguna caja antes de comenzar a atender.

Diagrama de clases

Resolución

	**Video**		**Código**

Ejercicio C03 - Simulador de llamadas

Consigna

Crear un simulador de llamadas para Centralita, agregando una nueva llamada a la lista en intervalos aleatorios de tiempo.

Comentar las líneas del método Main que generaban las llamadas y agregarle un menú para consultar los diferentes listados.

Resolución

	**Video**		**Código**