Introducción a C#

Manual del estudiante

Miguel Muñoz Serafín

Agosto, 2017

TI Capacitación

Introducción a C#

Manual de estudiante

Primera edición

Agosto de 2017

Soporte técnico:
soporte@mail.ticapacitacion.com
https://ticapacitacion.com/promociones/introcs.html

Contenido

Introducción a C#

Introducción

Acerca del curso

Este curso proporciona a los participantes, los conocimientos y habilidades requeridas para crear aplicaciones utilizando el lenguaje de programación C#.

El curso inicia con una introducción a la estructura básica de una aplicación C# y la sintaxis del lenguaje para posteriormente describir las distintas características proporcionadas por el .NET Framework, así como las técnicas y tecnologías empleadas para desarrollar aplicaciones de escritorio y empresariales.

Al final del curso, los participantes tendrán los conocimientos básicos del lenguaje de programación C# para desarrollar aplicaciones .NET.

Audiencia

Este curso está dirigido a todas las personas que se encuentren interesadas en aprender el lenguaje de programación C# y que cuenten con conocimientos básicos en algún lenguaje de programación como C, C++, JavaScript, Objective-C o Java entre otros.

Objetivos

Al finalizar este entrenamiento los participantes serán capaces de:

- Describir la sintaxis fundamental y características de C#.
- Crear e invocar métodos.
- Administrar Excepciones.
- Describir los requerimientos para monitoreo de aplicaciones de escala empresarial.
- Implementar la estructura básica y los elementos esenciales de una aplicación de escritorio tradicional.
- Crear Clases.
- Definir e implementar Interfaces.
- Crear y utilizar colecciones genéricas.
- Utilizar herencia para crear una jerarquía de Clases.
- Extender una Clase del .NET Framework.
- Crear Clases y métodos genéricos.

Requerimientos

Para poder practicar los conceptos y realizar los ejercicios del curso, se recomienda trabajar con un equipo de desarrollo con Windows 10 y Visual Studio 2015 o posteriores. Puede utilizarse la versión gratuita **Visual Studio Community** que puede descargarse desde el siguiente enlace:

https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx

Los videos de demostración de este curso fueron realizados con Visual Studio 2015.

Contenido del curso

El contenido de este entrenamiento está dividido en 5 módulos.

Módulo 1: Introducción a C#

En este módulo, se examina la sintaxis y características principales del lenguaje de programación C#, proporcionando también, una introducción al depurador de Visual Studio.

Al finalizar este módulo, los participantes podrán:

- Describir la arquitectura de las aplicaciones .NET y utilizar las características que Visual Studio y C# proporcionan para dar soporte al desarrollo de aplicaciones con el .NET Framework.
- Utilizar los tipos de datos básicos, operadores y expresiones proporcionadas por C#.
- Utilizar las sentencias principales del lenguaje de programación C#.

Los temas que forman parte de este módulo son:

- Lección 1: Desarrollando aplicaciones con C#
- Lección 2: Tipos de datos, Operadores y Expresiones
- Lección 3: Sentencias principales del lenguaje de programación C#

Módulo 2: Creación de Métodos, Manejo de Excepciones y Monitoreo de Aplicaciones

En este módulo se explica la forma de crear e invocar métodos, así como la manera de capturar y manejar Excepciones. Se describe también los requerimientos para monitorear aplicaciones de alta escalabilidad.

Al finalizar este módulo, los participantes podrán:

- Crear e invocar métodos, pasar parámetros a los métodos y devolver valores desde los métodos.
- Crear métodos sobrecargados y utilizar parámetros opcionales y parámetros de salida (output parameters).

- Capturar y manejar Excepciones, así como escribir información al log de eventos de Windows.
- Explicar los requerimientos para implementar el registro (Logging), seguimiento (Tracing) y análisis de rendimiento (Profiling) cuando se construyen aplicaciones de gran escala.

Los temas que se cubren en este módulo son:

- Lección 1: Creando e invocando Métodos
- Lección 2: Creando Métodos sobrecargados y utilizando parámetros opcionales y de salida (output)
- Lección 3: Manejo de Excepciones
- Lección 4: Monitoreo de aplicaciones

Módulo 3: Desarrollando el código para una aplicación gráfica

En este módulo, se describe la forma de implementar la estructura básica y los elementos esenciales de una aplicación gráfica típica, incluyendo el uso de estructuras, enumeraciones, colecciones y eventos.

Al finalizar este módulo, los participantes podrán:

- Definir y utilizar estructuras y enumeraciones.
- Crear y utilizar colecciones simples para almacenar datos en memoria.
- Crear, suscribir y lanzar eventos.

Los temas que se cubren en este módulo son:

- Lección 1: Implementando Estructuras y Enumeraciones
- Lección 2: Organizando datos dentro de colecciones
- Lección 3: Manejando Eventos

Módulo 4: Creando Clases e implementando colecciones de Tipos Seguros (Type-safe collections)

En este módulo se explica cómo crear Clases, definir e implementar Interfaces, así como la forma de crear y utilizar colecciones genéricas. Se describe también las diferencias entre Tipos Valor (Value Type) y Tipos Referencia (Reference Type) en C#.

Al finalizar este módulo, los participantes podrán:

- Crear y utilizar Clases personalizadas.
- Definir e implementar Interfaces personalizadas.
- Utilizar Tipos Genéricos para implementar colecciones de tipos seguros (Type-safe Collections).

Los temas que se cubren en este módulo son:

- Lección 1: Creando Clases
- Lección 2: Definiendo e implementando Interfaces
- Lección 3. Implementando colecciones de tipos seguros (Type-safe Collections)

Módulo 5: Creando una jerarquía de Clases utilizando Herencia

En este módulo se explica cómo utilizar herencia para crear una jerarquía de Clases y la manera de extender una Clase del .NET Framework. Este módulo también describe la forma de crear Clases Genéricas y la forma de definir Métodos de Extensión.

Al finalizar este módulo los participantes podrán:

- Definir Clases Abstractas.
- Heredar desde Clases Base para crear una jerarquía de Clases.
- Heredar desde Clases del .NET Framework.
- Utilizar Métodos de Extensión para agregar funcionalidad personalizada a las clases heredadas.
- Crear Clases y Métodos Genéricos.

Los temas que se cubren en este módulo son:

- Lección 1: Creando Jerarquías de Clases
- Lección 2: Extendiendo Clases del .NET Framework

Introducción a C#

Módulo 1: Introducción a C#

Acerca del módulo

El Microsoft .NET Framework proporciona una plataforma de desarrollo que podemos utilizar para construir, desplegar y administrar aplicaciones y servicios. En este módulo, conoceremos las principales características proporcionadas por el .NET Framework y Microsoft Visual Studio.

Examinaremos algunas de las sentencias clave de Visual C# que nos permitirán empezar a desarrollar aplicaciones para el .NET Framework.

Objetivos

Al finalizar este módulo, los participantes contarán con las habilidades y conocimientos para:

- Describir la arquitectura de las aplicaciones .NET
- Utilizar las características que Visual Studio y C# proporcionan para dar soporte al desarrollo de aplicaciones para el .NET Framework.
- Utilizar los tipos de datos básicos, operadores y expresiones proporcionadas por C#
- Utilizar las sentencias principales del lenguaje de programación C#.

Los temas que se cubren en este módulo son:

- Lección 1: Desarrollando aplicaciones con C#.
- Lección 2: Tipos de datos, Operadores y Expresiones.
- Lección 3: Sentencias principales del lenguaje de programación C#.

Lección 1:
Desarrollando aplicaciones con C#

El .NET Framework y Visual Studio proporcionan diversas características que podemos utilizar cuando desarrollamos nuestras aplicaciones.

En esta lección, aprenderemos acerca de las características que Visual Studio y el .NET Framework proporcionan y que nos permiten crear nuestras propias aplicaciones.

Objetivos de la lección

Al finalizar esta lección, los participantes contarán con los conocimientos y habilidades para:

- Describir la plataforma Microsoft .NET.
- Describir el propósito del .NET Framework.
- Describir la arquitectura de las aplicaciones .NET.
- Describir las características clave de Visual Studio.
- Describir las plantillas de proyectos proporcionadas en Visual Studio.
- Crear una aplicación .NET Framework.
- Describir XAML.

¿Qué es Microsoft .NET?

Microsoft .NET es una propuesta de Microsoft que proporciona una plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma de hardware y que permite un rápido desarrollo de aplicaciones. Con esta propuesta, se facilita la interconexión de distintas plataformas de hardware, software, información y usuarios. Basado en esta plataforma, se ha desarrollado una estrategia horizontal que integra productos que van desde sistemas operativos, herramientas de desarrollo hasta las aplicaciones para usuario final.

La plataforma .NET no es un producto empaquetado que se pueda comprar como tal, sino que es una plataforma que engloba distintas aplicaciones, servicios y conceptos y que en conjunto permiten el desarrollo y la ejecución de aplicaciones.

Microsoft .NET puede considerarse como una respuesta de Microsoft al creciente mercado de los negocios en entornos Web. La propuesta es ofrecer una manera rápida y económica, a la vez que segura y robusta, de desarrollar aplicaciones permitiendo una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

La plataforma .NET proporciona un modelo de programación consistente e independiente del lenguaje en todas las capas de una aplicación. Permite la interoperabilidad y una fácil migración de las tecnologías existentes.

La plataforma .NET proporciona nuevas formas de construir aplicaciones a partir de colecciones de servicios Web. Soporta completamente la infraestructura actual de Internet incluyendo HTTP, XML y SOAP entre otros estándares.

La plataforma .NET está compuesta por un conjunto de tecnologías diseñadas para transformar a Internet en una plataforma de cómputo distribuido. Proporciona Interfaces de programación y herramientas para diseñar, crear, ejecutar y distribuir soluciones para la plataforma .NET, por ejemplo, Visual Studio.

Dentro de la plataforma .NET se encuentran dispositivos ejecutando sistemas operativos que se integran e interactúan con otros elementos .NET. Se incluyen dispositivos móviles, Navegadores Web, Xbox, PCs, etc.

Los servidores también forman parte de la plataforma Microsoft .NET y proporcionan la infraestructura para administrar, construir y distribuir soluciones para la plataforma .NET, por ejemplo, Windows Server, SQL Server o Exchange Server.

Los Servicios Web, que son un conjunto de servicios predefinidos que realizan tareas específicas y que permiten a los desarrolladores crear sus propios servicios, también forman parte de la plataforma .NET

Las experiencias de usuario también forman parte de la plataforma Microsoft .NET y se componen de Software integrado con servicios XML Web que presentan al usuario la información que necesita y en la forma en que la necesita.

Componentes Principales

Los elementos principales de la plataforma .NET incluyen:

- **El .NET Framework.** El .NET Framework proporciona un entorno de ejecución de aplicaciones denominado **Common Language Runtime** o **CLR**. Este entorno es un componente de software cuya función es la de ejecutar las aplicaciones .NET e interactuar con el sistema operativo, ofreciendo sus servicios y recursos a estas aplicaciones .NET. Este entorno es común a todos los lenguajes .NET. Con esto podemos decir que la plataforma .NET no sólo nos brinda todas las herramientas y servicios que se necesitan para desarrollar modernas aplicaciones empresariales y de misión crítica, sino que también nos provee de mecanismos robustos, seguros y eficientes para asegurar que la ejecución de las mismas sea óptima.

- **Common Languaje Specification**. CLS define los estándares comunes a los que los lenguajes y desarrolladores deben adherirse para que sus componentes y aplicaciones puedan ser utilizados por otros lenguajes compatibles con .NET. El CLS permite que desarrolladores de Visual Basic puedan trabajar en equipo con otros desarrolladores de Visual C# sin que esto genere problemas de interoperabilidad entre el código generado por ambos lenguajes. Un desarrollador Visual Basic, por ejemplo, podrá consumir clases desarrolladas en Visual C# y viceversa.

- **Servidores empresariales .NET**. Los servidores empresariales .NET proporcionan escalabilidad, confiabilidad, administración e integración dentro de la empresa o incluso entre distintas empresas. Dentro de los servidores .NET podemos encontrar a Windows Server, SQL Server, Bistalk Server, Exchange Server, Sharepoint Server, etc.

- **Bloque de Servicios**. Los bloques de servicios están formados por Servicios Web que ofrecen diversas funcionalidades. Como ejemplo de estos bloques de servicios podemos mencionar al servicio *Microsoft Account* (anteriormente conocido como Windows Live) o al servicio OneDrive (anteriormente conocido como *SkyDrive)* entre otros.

- **Visual Studio**. Visual Studio .NET proporciona un ambiente para desarrollar aplicaciones para el .NET Framework. Visual Studio simplifica el proceso entero de desarrollo de software, desde el diseño hasta la implementación. Visual Studio tiene soporte para múltiples monitores, desarrollo de aplicaciones para Sharepoint, permite desarrollar aplicaciones para distintas versiones del .NET Framework, Soporte Intellisense y mucho más.

Como parte de la plataforma .NET, también contamos con un conjunto de lenguajes de programación de alto nivel, junto con sus compiladores y linkers, que permiten el desarrollo de aplicaciones para la plataforma .NET.

¿Qué es el .NET Framework?

El .NET Framework es la plataforma de desarrollo de código administrado de Microsoft. Está formado por una serie de herramientas y librerías con las que se pueden crear todo tipo de aplicaciones, desde las tradicionales aplicaciones de escritorio hasta aplicaciones para XBOX pasando por desarrollo Web, desarrollo para el Windows Store y Windows Phone así como aplicaciones de servidor con WCF.

El .NET Framework está compuesto por un conjunto de tecnologías que forman una parte importante de la plataforma .NET. Constituye una infraestructura de programación para construir, distribuir y ejecutar aplicaciones y servicios para la plataforma .NET.

El .NET Framework soporta completamente las características de la programación orientada a objetos. Soporta el uso de Herencia, Polimorfismo, Clases, propiedades, métodos, eventos, constructores y otras estructuras de la programación orientada a objetos.

Proporciona un ambiente que minimiza los conflictos de versiones de los DLL´s (DLL Hell) a los que se enfrentan los programadores que utilizan los componentes COM y simplifica la distribución e instalación de las aplicaciones.

Proporciona un ambiente de portabilidad basado en estándares certificados que permiten que las aplicaciones puedan ser hospedadas por cualquier sistema operativo. Actualmente C# y la mayor parte del motor de ejecución de .NET conocido como CLR y que es una implementación del Common Language Infrastructure (CLI) han sido estandarizados por ECMA. (European Computer Manufacturers Association).

El .NET Framework proporciona un ambiente administrado (Managed) en el cual el código es verificado para realizar una ejecución segura. Ofrece un entorno de ejecución de código que fomenta la ejecución segura del mismo, incluso del código creado por terceras personas desconocidas o que no son de plena confianza.

Componentes Principales del .NET Framework

El .NET Framework proporciona 3 elementos principales, el **Common Language Runtime** o motor en tiempo de ejecución común para todos los lenguajes .NET, **El .Net Framework Class Library** o biblioteca de clases base del .NET Framework y una **Colección de Frameworks** de desarrollo.

El **Common Language Runtime** es el corazón del .NET Framework. El CLR como comúnmente se le conoce, es la implementación de Microsoft del CLI. Es el agente encargado de administrar la ejecución del código y simplifica el proceso de desarrollo, proporcionando un ambiente de ejecución robusto y altamente seguro con servicios centrales como la compilación en tiempo de ejecución, Administración de memoria, Seguridad, Administración de los hilos de ejecución (Threads) además

de encargarse de aplicar una seguridad estricta a los tipos de datos y la interoperabilidad con código no administrado.

El concepto de administración de código es un principio básico del motor en tiempo de ejecución. El código que corre sobre el **CLR** es conocido como código administrado (Managed Code). Existen aplicaciones tales como componentes COM y aplicaciones basadas en las APIs de Windows cuyo código no requiere del ambiente administrado que ofrece el CLR. Este tipo de código recibe el nombre de código no administrado (Unmanaged Code).

La **Biblioteca de clases base del .NET Framework**, proporciona una colección completa orientada a objetos de tipos reutilizables que contiene Clases y estructuras de datos que se pueden emplear para desarrollar prácticamente todo tipo de aplicaciones. Las clases proporcionan la base de la funcionalidad común y elementos que ayudan a simplificar el desarrollo de aplicaciones, eliminando la necesidad de reinventar la lógica constantemente. Por ejemplo, la clase **System.IO.File** contiene funcionalidad que nos permite manipular el sistema de archivos de Windows. Además de utilizar las clases de la biblioteca de clases base, podemos extender estas clases, creando nuestras propias bibliotecas de clases.

El .NET Framework proporciona varios Frameworks de desarrollo que podemos utilizar para construir los tipos de aplicaciones comunes, incluyendo:

- Aplicaciones de escritorio cliente mediante el uso de Windows Presentation Foundation (WPF).
- Aplicaciones de escritorio de Windows Store utilizando XAML.
- Aplicaciones Web del lado del servidor, mediante ASP.NET Web Forms o ASP.NET MVC.
- Aplicaciones Web orientadas a servicios, mediante el uso de Windows Communication Foundation (WCF) o ASP.NET MVC Web API.
- Aplicaciones de ejecución en segundo plano mediante el uso de servicios Windows.

Cada Framework proporciona los componentes y la infraestructura necesaria para desarrollar aplicaciones.

El .NET Framework, traducido como “Marco de Trabajo”, es el componente fundamental de la plataforma Microsoft .NET, necesario tanto para poder desarrollar aplicaciones como para poder ejecutarlas luego en entornos de prueba o producción.

El CLR se comunica con el sistema operativo para proporcionar recursos a las aplicaciones administradas.

La biblioteca de clases base del .NET Framework, proporciona una colección completa orientada a objetos de tipos reutilizables que contiene Clases y Estructuras de datos que se pueden emplear para desarrollar prácticamente todo tipo de aplicaciones administradas.

Un tipo especial de aplicaciones administradas son las bibliotecas de clases personalizadas.

Las aplicaciones administradas pueden acceder a la biblioteca de clases base y a las bibliotecas de clases personalizadas.

Las aplicaciones administradas no se comunican directamente con el sistema operativo.

Las aplicaciones no administradas se comunican con el sistema operativo y pueden convivir con las aplicaciones administradas.

Los servicios de **Internet Information Server (IIS)** dan soporte a las aplicaciones ASP.NET. ASP.NET es el soporte para las aplicaciones Web administradas.

El motor ASP.NET se comunica con Internet Information Server y no con el sistema operativo.

Para obtener más información sobre el .**NET Framework**, se recomienda visitar el siguiente enlace:

Overview of the .NET Framework
https://msdn.microsoft.com/en-us/library/zw4w595w.aspx

El .NET Framework y el estándar CLI

El **CLI** (Common Language Infrastructure) define el ambiente virtual de ejecución de código que es independiente de cualquier plataforma. Como mencionamos anteriormente, ha sido estandarizado por ECMA. No es específica de un sistema operativo en particular.

El **CLR** del .NET Framework es la implementación del Common Language Infrastructure. El .NET Framework contiene más características que las especificadas en la arquitectura del CLI.

Una de las preocupaciones que tiene un desarrollador que decide invertir su tiempo en aprender C# y .NET es saber si esos conocimientos le permitirán desarrollar aplicaciones para otras plataformas. La pregunta típica es ¿El .NET Framework es un producto Microsoft diseñado únicamente para el sistema operativo Windows o es una plataforma que permite transportar las aplicaciones .NET a otros sistemas operativos?

Debido a que el CLI es independiente de cualquier plataforma y no es específica de un sistema operativo en particular, una aplicación .NET podría ejecutarse transparentemente en un ambiente Windows o Linux.

La parte central del CLI es la definición de un lenguaje intermedio común – Common Intermediate Language (CIL) – que debe ser generado por los compiladores compatibles con el CLI y un sistema de tipos de datos que define los distintos tipos de datos soportados por cualquier lenguaje compatible.

El CLI incluye además los estándares para el lenguaje C# desarrollado por Microsoft. Otros proveedores que han adoptado el estándar CIL, han desarrollado compiladores .NET para lenguajes como Python, Pascal, Fortran, Cobol e Eiffel entre otros.

Caracteristicas principales de Visual Studio

Visual Studio proporciona un ambiente de desarrollo que permite diseñar, implementar, compilar, probar y desplegar rápidamente varios tipos de aplicaciones y componentes utilizando una amplia gama de lenguajes de programación.

Algunas de las principales características de Visual Studio son:

- **Un Entorno de desarrollo integrado (IDE) intuitivo**. El IDE de Visual Studio ofrece todas las características y herramientas que son necesarias para diseñar, implementar, desarrollar, probar e implementar aplicaciones y componentes.
- **Desarrollo rápido de aplicaciones**. Visual Studio ofrece vistas de diseño para componentes gráficos que permiten crear fácilmente interfaces de usuario complejas. Alternativamente, podemos utilizar las vistas del editor de código, que proporcionan un mayor control. Visual Studio también proporciona asistentes que ayudan a acelerar el desarrollo de componentes particulares.
- **Acceso a datos y servidores**. Visual Studio proporciona el Explorador de servidores, que nos permite iniciar sesión en los servidores y explorar sus bases de datos y servicios del sistema. También proporciona una manera familiar para crear, consultar y modificar bases de datos que la aplicación utiliza a través del diseñador de tablas.
- **Internet Information Server (IIS) Express**. Visual Studio proporciona una versión ligera de IIS como servidor Web predeterminado para la depuración de nuestras aplicaciones Web.
- **Funciones de Depuración**. Visual Studio proporciona un depurador que permite ejecutar paso a paso a través de código local o remoto, hacer una pausa en los puntos de interrupción y seguir rutas de ejecución.
- **Manejo de errores**. Visual Studio proporciona la ventana **Lista de errores**, que muestra los errores, advertencias o mensajes que se producen al editar y crear nuestro código.
- **Ayuda y Documentación**. Visual Studio ofrece ayuda y orientación a través del **Microsoft IntelliSense**, fragmentos de código y el sistema de ayuda integrada que contiene documentación y ejemplos.

Para obtener más información acerca de las novedades en Visual Studio, se recomienda visitar el siguiente enlace:

Novedades de Visual Studio 2015
https://msdn.microsoft.com/es-mx/library/bb386063.aspx

Plantillas de Visual Studio

Visual Studio nos proporciona un entorno de Desarrollo que permite crear aplicaciones Windows, Web, Servicios, Bibliotecas, o aplicaciones empresariales para la nube. Para ayudarnos a iniciar, Visual Studio proporciona una serie de plantillas de aplicación que proporcionan una estructura para los diferentes tipos de aplicaciones. Las plantillas:

- Proporcionan el código inicial para construir y crear rápidamente aplicaciones funcionales.
- Incluyen soporte a componentes y controles dependiendo del tipo de proyecto seleccionado.
- Configuran el IDE de Visual Studio dependiendo del tipo de aplicación que se desea desarrollar.
- Agregan referencias a los ensamblados que el tipo de aplicación requiera.

La siguiente tabla describe algunas de las plantillas comunes de aplicaciones que podríamos utilizar cuando desarrollemos aplicaciones .NET Framework utilizando Visual Studio.

Plantilla	Descripción
Console Application	Proporciona la configuración del ambiente, herramientas, referencias y código inicial para desarrollar una aplicación que se ejecuta en una interfaz de línea de comandos. Este tipo de aplicación es considerada ligera debido a que no tiene una interfaz de usuario gráfica.
Windows Forms Application	Proporciona la configuración del ambiente, herramientas, referencias y código inicial para desarrollar una aplicación gráfica Windows Form.
WPF Application	Proporciona la configuración del ambiente, herramientas, referencias y código inicial para desarrollar una aplicación Windows rica en interfaz de usuario. Una aplicación WPF nos permite crear la siguiente generación de aplicaciones Windows con mucho más control sobre el diseño de la interfaz de usuario.
Blank App (Universal Windows)	Proporciona la configuración del ambiente, herramientas, referencias y código inicial para desarrollar una aplicación de la Plataforma Universal de Windows (Universal Windows Platform).
Blank App (Universal Windows 8.1)	Proporciona la configuración del ambiente, herramientas, referencias y código inicial para desarrollar una aplicación Universal para Windows y Windows Phone que utiliza el Windows Runtime.
Class Library	Proporciona la configuración del ambiente, herramientas, referencias y código inicial para desarrollar una biblioteca de clases. Esta plantilla nos permite generar un Assembly .dll que podemos utilizar para almacenar funcionalidad que podríamos querer invocar desde otra aplicación.
ASP.NET Web Application (.NET Framework)	Proporciona la configuración del ambiente, herramientas, referencias y código inicial para desarrollar aplicaciones ASP.NET. Podemos crear aplicaciones Web Forms, MVC, Web API y otro tipo de aplicaciones ASP.NET.

Introducción a XAML

Extensible Application Markup Language (**XAML)**, es un lenguaje basado en XML que podemos utilizar para definir la interfaz de usuario de aplicaciones .NET. XAML también puede ser utilizado para desarrollar aplicaciones para la **Plataforma Universal de Windows (UWP)** o para desarrollar aplicaciones para iOS y Android con Xamarin.

XAML utiliza elementos y atributos para definir controles y sus propiedades en sintaxis XML. Cuando diseñamos una interfaz de usuario, podemos utilizar la caja de herramientas y el panel de propiedades de Visual Studio para crear visualmente la interfaz de usuario. Podemos utilizar el panel XAML para crear la interfaz de usuario de forma declarativa. También podemos utilizar **Blend** para Visual Studio o cualquier herramienta de terceros. Algunos desarrolladores pueden sentirse más cómodos al escribir directamente código **XAML** sobre el *panel XAML* en lugar de arrastrar los controles desde el cuadro de herramientas hacia la ventana.

El siguiente ejemplo muestra la declaración XAML de una etiqueta, una caja de texto y un botón.

```
<TextBlock Text="Name:" HorizontalAlignment="Left" Margin="72,43,0,0"
    VerticalAlignment="Top" />
<TextBox HorizontalAlignment="Left" Height="23" Margin="141,43,0,0" Text=""
      VerticalAlignment="Top" Width="120" />
<Button Content="Click Me!" HorizontalAlignment="Left" Margin="119,84,0,0"
     VerticalAlignment="Top" Width="75" />
```

El definir la interfaz de usuario en XAML, nos permite que esta sea más portable y separa la interfaz de usuario de la lógica de la aplicación. Por ejemplo, es posible migrar una gran parte del código XAML de una aplicación WPF y reutilizarla en una aplicación UWP o Xamarin.

Podemos utilizar la sintaxis XAML para generar interfaces de usuario simples o construir interfaces de usuario mucho más complejas. La sintaxis de XAML proporciona la funcionalidad para enlazar datos a controles, utilizar gradientes, texturas, plantillas para dar formato a los datos y enlazar a eventos de los controles.

También es posible utilizar distintos contenedores para posicionar los controles de nuestra interfaz de usuario de forma apropiada y dar una apariencia uniforme ajustándose al tamaño de la pantalla.

Lección 2:
Tipos de datos, Operadores y Expresiones

La mayoría de las aplicaciones utilizan datos, estos datos podrían ser proporcionados por el usuario a través de una interfaz de usuario, desde una base de datos, desde un servicio de red, o desde alguna otra fuente. Para almacenar y utilizar datos en nuestras aplicaciones, debemos familiarizarnos con la forma de definir y utilizar variables, así como en la forma de crear y utilizar expresiones con la variedad de operadores que Visual C# proporciona.

En esta lección, conoceremos la forma de utilizar algunos de los elementos fundamentales de la programación en Visual C#, tales como variables, miembros de los tipos de datos, conversiones y manipulación de cadenas.

Objetivos de la lección

Al finalizar esta lección, los participantes contarán con las habilidades y conocimientos para:

- Describir los tipos de datos proporcionados por Visual C#.
- Crear y utilizar expresiones.
- Declarar y asignar variables.
- Acceder a los miembros de la instancia de un tipo de dato.
- Convertir datos de un tipo a otro.
- Concatenar y validar cadenas.

¿Qué son los tipos de Datos?

Una variable contiene datos de un tipo específico. Cuando declaramos una variable para almacenar los datos en una aplicación, debemos elegir un tipo de dato adecuado para los datos. Visual C# es un lenguaje de tipos seguros "**Type-Safe**", esto significa que el compilador garantiza que los valores almacenados en las variables siempre son del tipo apropiado.

La siguiente tabla muestra los tipos de datos más comunes que son utilizados en Visual C#.

Tipo	Descripción	Tamaño en bytes	Rango
int	Números enteros.	4	–2,147,483,648 a 2,147,483,647
long	Números enteros.	8	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
float	Números de punto flotante.	4	+/–3.4 × 10^38
double	Números de punto flotante de doble precisión (más precisos).	8	+/–1.7 × 10^308
decimal	Valores de moneda.	16	-7.9 x 10^28 a 7.9 x 10^28
char	Un simple carácter Unicode.	2	N/A
bool	Valor booleano.	1	Falso o Verdadero.
DateTime	Momentos en el tiempo	8	0:00:00 del 01/01/2001 a 23:59:59 del 12/31/9999
string	Secuencia de caracteres.	2 por carácter.	N/A

Para mayor información acerca de los tipos de datos, se recomienda visitar el siguiente enlace:

Reference Tables for Types (C# Reference)
https://msdn.microsoft.com/en-us/library/1dhd7f2x.aspx

Expresiones y Operadores en Visual C#

Las expresiones son el componente central de prácticamente cualquier aplicación Visual C# debido a que las expresiones son construcciones fundamentales que utilizamos para evaluar y manipular datos.

Las expresiones son colecciones de operandos y operadores que podemos definir de la siguiente manera:

- Los operandos son valores por ejemplo números y cadenas. Los operandos pueden ser valores constantes (literales), variables, propiedades o valores devueltos por las llamadas a métodos.
- Los operadores definen operaciones a realizar sobre los operandos, por ejemplo, la suma o la multiplicación. Los operadores existen para todas las operaciones matemáticas básicas, así como para operaciones más avanzadas tales como comparaciones lógicas o la manipulación de bits de datos que constituyen un valor.

Todas las expresiones son evaluadas a un simple valor cuando la aplicación es ejecutada. El tipo de valor que una expresión genera depende de los tipos de operandos y operadores que utilicemos. No existe un límite en la longitud de las expresiones en las aplicaciones Visual C#, aunque en la práctica estamos limitados por la memoria de la computadora y nuestra paciencia al escribir. La recomendación es utilizar expresiones cortas y ensamblar el resultado de las expresiones individuales. Esto nos facilita ver lo que el código está realizando y facilita también la depuración del código.

Operadores en Visual C#

Los operadores se combinan con los operandos para formar expresiones. Visual C# proporciona una amplia gama de operadores que podemos utilizar para realizar las operaciones matemáticas y lógicas fundamentales más comunes. Los operadores caen dentro de una de las siguientes tres categorías:

- **Unario**. Este tipo de operador, opera sobre un solo operando. Por ejemplo, podemos utilizar el operador **–** como un operador unario. Para hacer esto, lo colocamos inmediatamente antes de un operando numérico y el operador convierte el valor del operando a su valor actual multiplicado por **-1**.
- **Binario**. Este tipo de operador opera sobre 2 valores. Este es el tipo más común de operador, por ejemplo, *****, el cual multiplica el valor de dos operandos.
- **Ternario**. Existe un solo operador ternario en Visual C#. Este es el operador **? :** que es utilizado en expresiones condicionales.

La siguiente tabla muestra los operadores más comunes que podemos utilizar en Visual C# agrupados por tipo.

Tipo	Operadores
Aritméticos	+, -, *, /, %
Incremento, decremento	++, --
Comparación	==, !=, <, >, <=, >=, is
Concatenación de cadenas	+
Operaciones lógicas de bits	&, |, ^, !, ~, &&, ||
Indizado (el contador inicia en el elemento 0)	[ ]
Conversiones	( ), as
Asignación	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, ??
Rotación de Bits	<<, >>
Información de Tipos de datos	sizeof, typeof
Concatenación y eliminación de Delegados	+, -
Control de excepción de Overflow	checked, unchecked
Apuntadores y direccionamiento en código No seguro (Unsafe code)	*, ->, [ ], &
Condicional (operador ternario)	?:

Para mayor información acerca de los operadores en Visual C#, se recomienda visitar el siguiente enlace:

C# Operators
https://msdn.microsoft.com/en-us/library/6a71f45d.aspx

Declaración y asignación de variables

Antes de utilizar una variable, debemos declararla. Al declararla podemos especificar su nombre y características. El nombre de la variable es referido como un **Identificador**. Visual C# tiene reglas específicas relacionadas con el uso de los identificadores:

- Un identificador solo puede contener letras, dígitos y el carácter guion bajo.
- Un identificador debe iniciar con una letra o un guion bajo.
- Un identificador no debería ser una de las palabras clave que visual C# reserva para su propio uso.

Visual C# es sensible a mayúsculas y minúsculas, por ejemplo, la variable **IDEmpleado** es distinta a la variable **idempleado**. Nosotros podemos declarar 2 variables llamadas **IDEmpleado** y **idempleado** al mismo tiempo y Visual C# no se confundirá, sin embargo, esta no es una buena práctica de codificación.

Al declarar una variable, debemos elegir un nombre que tenga significado respecto a lo que almacena, de esta forma, el código será más fácil de entender. Debemos también adoptar una convención de nombres y utilizarla.

Microsoft ofrece una nomenclatura que podríamos seguir en caso de no contar con una nomenclatura propia. Para mayor información, se recomienda visitar el siguiente enlace:

Naming Guidelines
https://msdn.microsoft.com/en-us/library/ms229002(v=vs.110).aspx

Cuando se declara una variable, se reserva un espacio de almacenamiento en memoria para esa variable y el tipo de datos que va a contener. Podemos declarar múltiples variables en una sola declaración utilizando el separador coma (,), todas las variables declaradas de esta manera, son del mismo tipo de datos. El siguiente ejemplo, muestra como declarar una nueva variable.

```
int Precio;
int Impuesto, Descuento;
```

Después de declarar la variable, podemos asignarle un valor utilizando una operación de asignación. Durante la ejecución de la aplicación, podemos cambiar el valor de una variable tantas veces como queramos. El operador de asignación **=** nos permite asignar un valor a una variable.

El siguiente código muestra cómo utilizar el operador **=** para asignar el valor a una variable.

```
Precio = 120;
```

El valor del lado derecho de la expresión es asignado a la variable que se encuentra en el lado izquierdo de la expresión.

También es posible declarar una variable y asignar su valor al mismo tiempo. El siguiente ejemplo declara una variable de tipo entero llamada **Precio** y le asigna el valor 120.

```
int Precio = 120;
```

Cuando declaramos una variable, esta contiene un valor aleatorio hasta que le asignamos un valor. Este comportamiento fue una poderosa fuente de errores en C y C++ que nos permitían declarar variables y utilizarlas accidentalmente sin haberles asignado previamente un valor. Visual C# no nos permite utilizar una variable que no haya sido previamente asignada. Debemos asignar un valor a una variable antes de que la utilicemos, de lo contrario, la aplicación podría no compilar.

Al declarar las variables, también podemos utilizar la palabra clave **var** en lugar de especificar un tipo de datos explícitamente, como **int** o **string**. Cuando el compilador encuentra la palabra clave **var**, este utiliza el valor que se asigna a la variable para poder determinar el Tipo correspondiente.

En el siguiente ejemplo, se muestra la forma de utilizar la palabra clave **var** para declarar una variable.

```
var Perimetro = 1000;
```

En este caso, la variable **Perimetro** es una variable cuyo tipo fue establecido de forma implícita. Sin embargo, la palabra clave **var** no indica que posteriormente podamos asignarle un valor de un tipo diferente. El tipo de **Perímetro** es fijo de la misma forma que lo sería si lo hubiéramos declarado explícitamente.

Las variables con tipo implícito son útiles cuando no sabemos, o es difícil establecer de forma explícita el tipo de una expresión que se desea asignar a una variable.

Otro tipo común de variables que podemos utilizar, son las variables que almacenan objetos.

Cuando declaramos una variable **objeto**, está se encuentra inicialmente sin asignar. Para utilizar una variable objeto, debemos crear una instancia de la clase correspondiente mediante el operador **new** y asignarla a la variable objeto.

El operador **new** hace dos cosas:

1. Hace que el CLR asigne memoria para el objeto.
2. Después de asignar la memoria para el objeto, invoca un constructor para inicializar los campos de dicho objeto. La versión del constructor que se ejecuta depende de los parámetros que especifiquemos en el operador **new**.

El siguiente ejemplo muestra la forma de crear la instancia de una clase mediante el uso del operador **new**.

```
var Coleccion = new System.Collections.ArrayList();
```

Para mayor información acerca de la declaración y asignación de variables, se recomienda visitar el siguiente enlace:

Implicitly Typed Local Variables (C# Programming Guide)
https://msdn.microsoft.com/en-us/library/bb384061.aspx