C# Guía básica de programación

Pre-requisitos

- Instalar .NET

  https://dotnet.microsoft.com/download

  Para probar la instalación, ejecutar en la línea de comandos:

  ```
  dotnet --version
  ```

.NET Command Line Interface (CLI) Comandos Báscos

Compilación

- Clean

```
dotnet clean
```

  Limpia las compilaciones previas, borrando los archivos de los directorios /bin /obj

- Restore

```
dotnet restore
```

  Restaura los paquetest NuGet instalados en el proyecto, haciendo que estén disponibles para compilarse

- Debug

```
dotnet build
```

  Es la compilación en modo de desarrollo. Sirve localmente para pruebas.

- Release

```
dotnet build -c Release
```

  Es la compilación en modo productivo. Sirve para enviar el proyecto al cliente listo para el entorno real.

Ejecución de pruebas automática

```
dotnet test
```

Ejecución

- Clasica

```
dotnet run
```

- Con HotReload (actualiza automáticamente los cambios)

```
dotnet watch
```

Publicación

```
dotnet publish
```

Assemblies

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/standard/assembly/

Un `Assembly` (Ensamblado) es un archivo generado tras la compilación de un proyecto de código .NET. Los nombres de `namespace` se escriben en `PascalCase`. Por ejemplo, una librería de clases genera un Assembly nombrado `ClassLibrary.dll`

Si ejecutamos en el directorio de un proyecto .NET de consola:

`dotnet build`
se producirá la compilación dentro del directorio `/bin` generando los archivos `.dll` y `.exe` que contienen la compilación del Assembly .NET.

Namespaces

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/types/namespaces

Dentro de un Assembly .NET existe idealmente un `namespace` (Espacio de nombres) que contiene todo el código de ese Assembly. Podrían haber más de uno, pero no es lo recomendado.

Un `namespace` es una jerarquía que define una estructura similar a la de directorios y archivos, que sirve para organizar el código dentro de un Assembly. Por lo tanto, idealmente los namespaces deberían coincidir con la estructura de directorios de la solución. Los namespaces se nombran en `PascalCase`.

Clases

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/types/classes

- Tutorial

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/tutorials/classes

Las clases definen los objetos que el programa contendrá. Una clase podría pensarse como la analogía entre una receta para hacer una galletita y una galletita. La clase sería la receta (el diseño de la galletita) y la galletita sería la instancia de esa clase (que será contenida en una variable). Los nombres de clases se escriben en `PascalCase`.

- Clase

```
public class Cookie
{
    public double ChocolateAmount { get; set; }
    public double SugarAmount { get; set; }
}
```

- Instancia

```
var cookie = new Cookie();
```

Propiedades

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/properties

Las propiedades son definiciones de características de un objeto que permiten almacenar valores y lo describen. Pueden ser de solo lectura (`get`) o de lectura y escritura (`get` y `set`). Sus nombres se escriben en `PascalCase`.

- Ejemplos

```
public class Person
{
    public string FirstName { get; set; }

    public string LastName { get; set; }

    public string FullName { get { return $"{FirstName} {LastName}"; } }

    public DateTime DateOfBirth { get; set; }

    public int Age { get { return (DateTime.Now - DateOfBirth).TotalDays; } }
}
```

Variables

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/language-reference/statements/declarations

Una variable es un espacio de memoria en el programa que contiene valores que pueden asignarse o leerse. Las variables en C# se asignan con el operador = La declaración de la variable comprende desde el igual hacia la izquierda y la asignación comprende desde el igual hacia la derecha. La palabra clave `new` de C# genera una instancia nueva de la clase que se defina.

- Variable definida con `var`

  ```
  var cookie = new Cookie();
  var name = "Juan";
  ```

  Son variables cuyo tipo es inferido automáticamente (implícitamente) por C#

- Variable definida explícitamente

  ```
  Cookie cookie = new Cookie();
  int count = 10;
  ```

  Son variables cuyo tipo es declarado explícitamente por el programador.

Modificadores de acceso

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/access-modifiers

- Publicos, son de acceso irrestricto

  ```
  public string Name { get; set; }
  ```

- Privados, son de acceso limitado al tipo que las contiene y no son visibles fuera del mismo

  ```
  private string Name { get; set; }
  ```

Ejemplo

```
public class Cookie
{
    public string Name { get; set; }
    private double ChocolateAmount { get; set; }
    private double SugarAmount { get; set; }
}
```

Tipos

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/types/

  https://learn.microsoft.com/es-es/dotnet/csharp/language-reference/language-specification/types

- Tipos por valor Sus valores estan directamente asignados en memoria. Son los tipos fundamentales como `int`, `byte`, `char`, etc.
- Tipos por referencia Sus valores estan asignados por la definición de una clase que representa una referencia en memoria al objeto, por ejemplo, las clases definidas por el programador como `Cookie`, `DateTime`, etc.
- Tipos genéricos Son tipos compuestos por uno o varios parámetros de tipo, por ejemplo, `List<Cookie>`

Cadenas de texto

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/tutorials/exploration/interpolated-strings

- La manera más simple de trabajar con cadenas de texto es la siguiente

  ```
  var fullName = customer.Surname + ", " + customer.Name;
  ```

- Otra manera más práctica se llama interpolación de cadenas

  ```
  var fullName = $"Hello {customer.Surname}, {customer.Name}. It's a pleasure to meet you!";
  ```

Métodos

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/methods

Firma de un método

Los métodos pueden declararse en una clase, o interfaz especificando el nivel de acceso, como `public` o `private`, el valor devuelto, el nombre del método y cualquier parámetro de método. Todas estas partes forman la firma del método.

Declaración de la firma de un método

- Modificador de acceso

  Define el tipo de acceso que el método tendrá, por ejemplo, `public` o `private`.

- Tipo de retorno

  Define el tipo de por valor o referencia que el método devolverá como resultado

- Nombre

  Define el nombre del método. Los nombres de método deben ser escritos en `PascalCase`.

- (parámetro)

  Define un valor que ingresará al método como parámetro del mismo. Deben ser escritos en `camelCase` Los parámetros se definen por tipo (espacio) nombre y se separan por comas.

  ```
  (tipo nombre, tipo nombre)
  (int  number1, int number2)
  ```

Ejemplo

```
namespace SampleProject
{
    public class SimpleMath
    {
        /*

        Mod. acceso   Tipo de ret.   Nombre          (parámetro, parámetro)
        public        int            AddTwoNumbers   (int number1, int number2)

        */
        public int AddTwoNumbers(int number1, int number2)
        {
            return number1 + number2;
        }

        public int SquareANumber(int number)
        {
            return number * number;
        }
    }
}
```

Programación Orientada a Objetos en .NET

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/tutorials/oop

- Herencia

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/tutorials/inheritance

- Interfaces

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/types/interfaces

- Generics

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/types/generics

- Tipos anónimos

  https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/types/anonymous-types

- Abstracción: modelar los atributos e interacciones pertinentes de las entidades como clases para definir una representación abstracta de un sistema.
- Encapsulación: ocultar el estado interno y la funcionalidad de un objeto y permitir solo el acceso a través de un conjunto público de funciones.
- Herencia: capacidad de crear nuevas abstracciones basadas en abstracciones existentes.
- Polimorfismo: capacidad de implementar propiedades o métodos heredados de maneras diferentes en varias abstracciones.

LINQ (Language Integrated Query / Consulta Integrada del Lenguaje)

https://learn.microsoft.com/es-es/dotnet/csharp/tutorials/working-with-linq

Perimite realizar consultas a objetos de .NET Por ejemplo `.Where()` `.First()` `.Select()` etc..

Ejemplo

```
public List<MeasureDto> GetByCustomerId(int customerId)
{

    var list = _context.Measures.Where(m => m.Reefer.Customer.Id ==
    customerId).Include(t =>  t.Reefer).ToList();

     var dtoList = list.Select(t => new MeasureDto
     {

         Temperature = t.Temperature,
         Humidity = t.Humidity,
         MeasureDateTime = t.MeasureDateTime,
         Pressure = t.Pressure,
         ReeferId = t.Reefer.Id,
         Id = t.Id,

     }).ToList();

     return dtoList;
}
```

Servicio REST

- Referencia teórica

  https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio-code

Cliente REST

- Referencia teórica

  https://learn.microsoft.com/es-es/dotnet/csharp/tutorials/console-webapiclient