PROGRAMACION WEB AVANZADA

Actividad 4: Consulta - Fundamentos de C

Author
Jairo Quilumbaquin
Sangolqui, Ecuador
Miércoles 19 de Noviembre del 2023

Contents

1 Introducción a C#

1.1 ¿Qué es C#?

El lenguaje informático C está orientado a objetos y componentes. Dado que C ofrece estructuras de lenguaje que soportan explícitamente estas ideas, es un lenguaje ideal para desarrollar y utilizar componentes de software.[1]

1.2 Evolución e historia

Microsoft lanzó el marco de desarrollo .NET en 2002. En aquel momento, la mayoría de los desarrolladores que trabajaban con tecnologías Microsoft creaban programas de escritorio utilizando la plataforma Visual Basic. Cuando surgió.NET, se lanzaron varios lenguajes para la plataforma, siendo C# y VB.NET los primeros en salir a la luz. Microsoft necesitaba facilitar al máximo el cambio a su amplia comunidad de desarrolladores de Visual Basic, así que creó VB.NET.[2]
La necesidad de un lenguaje con una sintaxis como la de C, al igual que la mayoría de los lenguajes más utilizados, incluido Java, uno de sus principales rivales, se cumplió con C#, dado que era un lenguaje más serio, robusto y estándar del sector que VB.NET, la comunidad de desarrolladores optó finalmente por promoverlo, esto facilitó enormemente el cambio desde otros lenguajes.
No obstante, C# se convirtió rápidamente en el lenguaje estándar del marco .NET de Microsoft. Actualmente es uno de los lenguajes más utilizados en la industria del software, sobre todo ahora que Microsoft ha hecho posible ejecutar y desarrollar programas .NET en cualquier sistema operativo.

1.3 Plataformas compatibles

De acuerdo con al documentacion de C#[3] las plataformas compatibles son:

- Windows
- Linux
- macOS
- Android y IOs haciendo uso de herramientas de herramientas de Xamarin.

1.4 Configuración del Entorno de Desarrollo

Para la instalación de visual studio descargamos el instalador y lo ejecutamos: En el asis-

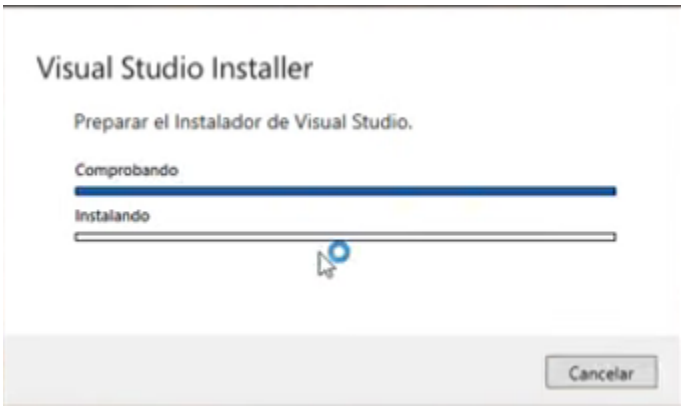

Figure 1: instalador de C#

tente de instalación seleccionamos los paquetes que deseamos instalar de acuerdo a nuestras necesidades y esperamos a que presionamos el botón de instalar:

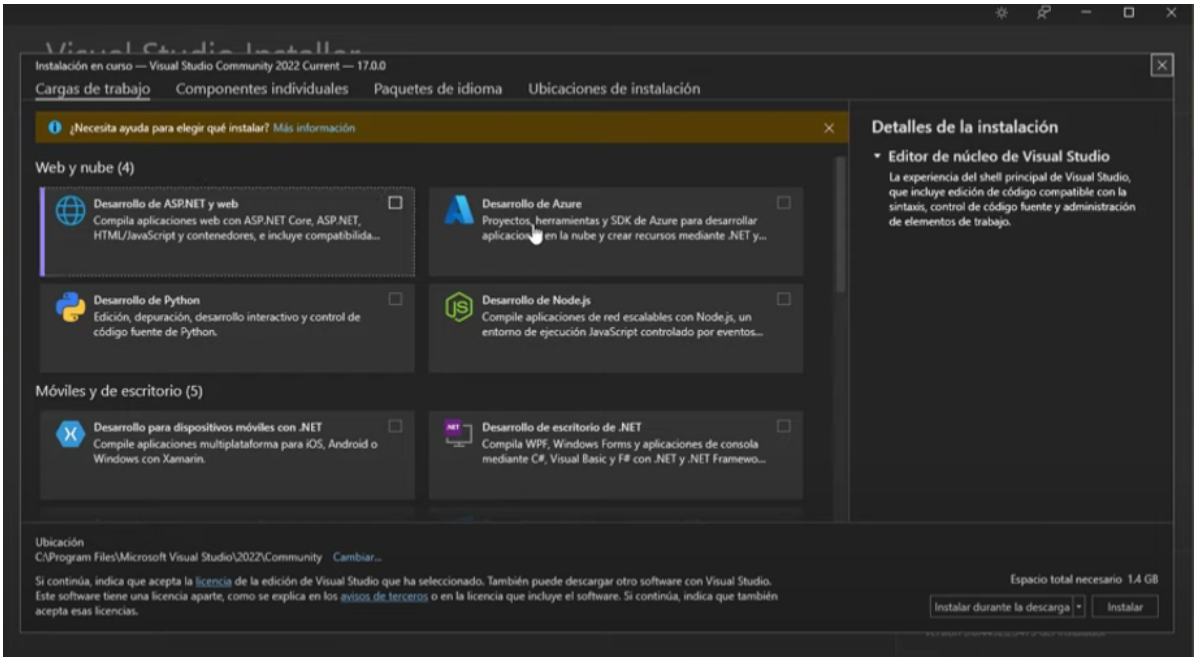

Figure 2: asistente de instalación de paquetes

Una vez descargados e instalados los paquetes se abre la ventana de inicio donde podemos escoger nuestros proyectos existentes o iniciar uno nuevo:

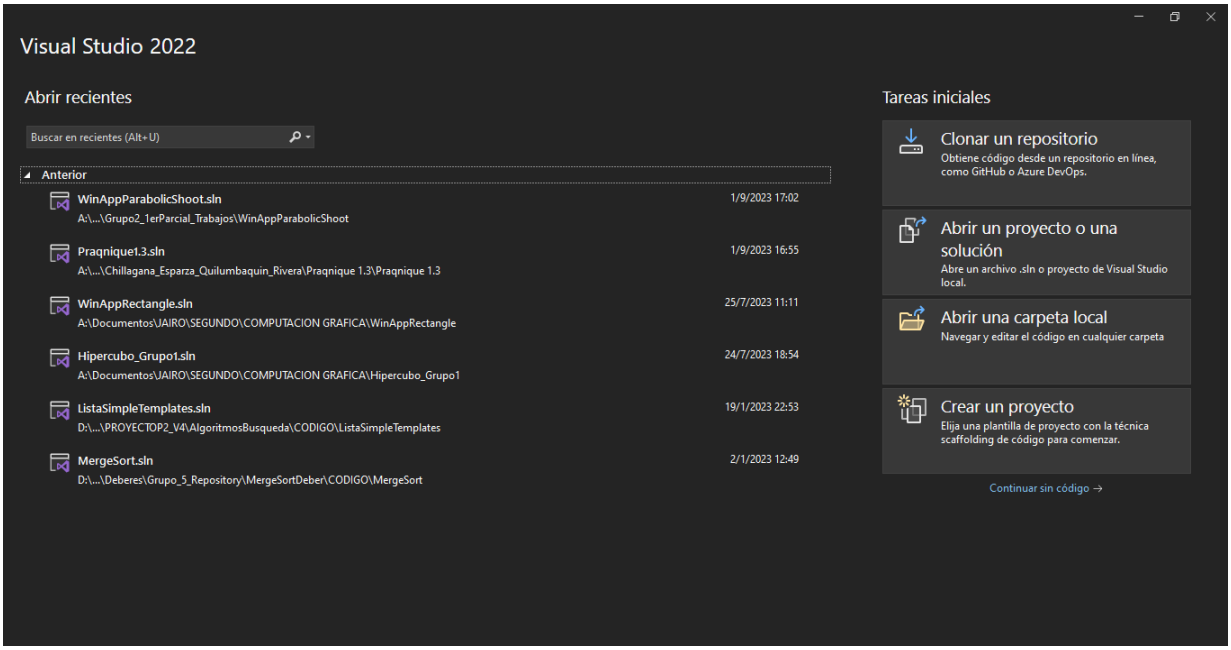

Figure 3: Ventana de inicio de visual studio

1.5 Sintaxis Básica de C#

1.5.1 Estructura de un programa en C#

Las ideas fundamentales de la organización de C# son los miembros, los ensamblados, los programas, los espacios de nombres y los tipos. Los tipos son declarados por los programas, y pueden agruparse en espacios de nombres y tener miembros. Ejemplos de tipos son interfaces, clases y estructuras. Los miembros incluyen cosas como propiedades, eventos, métodos, campos, etc. Los programas C# se agrupan físicamente en ensamblados durante la compilación. Los ensamblados suelen terminar en.exe o.dll, dependiendo de si implementan bibliotecas o aplicaciones, respectivamente.[4]

A continuación se muestra un ejemplo de programa en C#:

```
namespace Acme.Collections;

public class Stack<T>
{
    Entry _top;

    public void Push(T data)
    {
        _top = new Entry(_top, data);
    }

    public T Pop()
    {
        if (_top == null)
        {
            throw new InvalidOperationException();
        }
        T result = _top.Data;
        _top = _top.Next;

        return result;
    }

    class Entry
    {
        public Entry Next { get; set; }
        public T Data { get; set; }

        public Entry(Entry next, T data)
        {
            Next = next;
            Data = data;
        }
    }
}
```

1.5.2 Variables y tipos de datos

los tipos de datos de acuerdo a la documentacion oficial son:

- Tipos de valor
  - Tipos simples
    - Entero con signo: sbyte, short, int, long
    - Entero sin signo: byte, ushort, uint, ulong
    - Caracteres Unicode: char, que representa una unidad de código UTF-16
    - Punto flotante binario IEEE: float, double.
    - Punto flotante decimal de alta precisión: decimal
    - Booleano: bool, que representa valores booleanos, valores que son true o false
- Tipos de enumeración
  - Tipos definidos por el usuario con el formato enum E .... Un tipo enum es un tipo distinto con constantes con nombre. Cada tipo enum tiene un tipo subyacente, que debe ser uno de los ocho tipos enteros. El conjunto de valores de un tipo enum es igual que el conjunto de valores del tipo subyacente.
- Tipos de estructura
  - Tipos definidos por el usuario con el formato struct S ...
- Tipos de valores que aceptan valores NULL
  - Extensiones de todos los demás tipos de valor con un valor null
- Tipos de valor de tupla
  - Tipos definidos por el usuario con el formato (T1, T2, ...)
- Tipos de referencia
  - Tipos de clase
    - Clase base definitiva de todos los demás tipos: object
    - Cadenas Unicode: string, que representa una secuencia de unidades de código UTF-16
    - Tipos definidos por el usuario con el formato class C ...
  - Tipos de interfaz
    - Tipos definidos por el usuario con el formato interface I ...
  - Tipos de matriz
    - Unidimensional, multidimensional y escalonada. Por ejemplo, int[], int[,] y int[][].
  - Tipos delegados
    - Tipos definidos por el usuario con el formato delegate int D(...)

1.6 Control de Flujo y Estructuras de Control

A continuación se muestran ejemplos de la implementación de las estructuras de control en C#:

1.6.1 Condicionales

```
// Sentencia if else
DisplayWeatherReport(15.0);  // Output: Cold.
DisplayWeatherReport(24.0);  // Output: Perfect!

void DisplayWeatherReport(double tempInCelsius)
{
    if (tempInCelsius < 20.0)
    {
        Console.WriteLine("Cold.");
    }
    else
    {
        Console.WriteLine("Perfect!");
    }
}
```

```
// Sentencia switch
DisplayMeasurement(-4);  // Output: Measured value is -4; too low.
DisplayMeasurement(5);  // Output: Measured value is 5.
DisplayMeasurement(30);  // Output: Measured value is 30; too high.
DisplayMeasurement(double.NaN);  // Output: Failed measurement.

void DisplayMeasurement(double measurement)
{
    switch (measurement)
    {
        case < 0.0:
            Console.WriteLine($"Measured value is {measurement}; too low.");
            break;

        case > 15.0:
            Console.WriteLine($"Measured value is {measurement}; too high.");
            break;

        case double.NaN:
            Console.WriteLine("Failed measurement.");
            break;

        default:
            Console.WriteLine($"Measured value is {measurement}.");
            break;
    }
}
```

1.6.2 Bucles

```
// Sentencia for
for (int i = 0; i < 3; i++)
{
    Console.Write(i);
}
// Output:
// 012
```

```
// Sentencia while
int n = 0;
while (n < 5)
{
    Console.Write(n);
    n++;
}
// Output:
// 01234
```

```
// Sentencia do while
int n = 0;
do
{
    Console.Write(n);
    n++;
} while (n < 5);
// Output:
// 01234
```

1.7 Funciones y Métodos en C#

1.7.1 Declaración y llamada de funciones

A continuación se muestra un ejemplo de declaración de métodos:

```
// Declaracion de funciones
private static string GetText(string path, string filename)
{
     var reader = File.OpenText($"{AppendPathSeparator(path)}{filename}");
     var text = reader.ReadToEnd();
     return text;

     string AppendPathSeparator(string filepath)
     {
        return filepath.EndsWith(@"\") ? filepath : filepath + @"\";
     }
}
```

1.7.2 Parámetros y argumentos

Una breve explicación sobre parámetros y argumentos es que los parámetros son parte interna de una función y los argumentos son los valores que recibe una función como se muestra en el ejemplo anterior los parámetros son la variable reader y los argumentos son las variables path y filename.

1.7.3 Retorno de valores

oara retornar un valor producto de una operacion realizada por una función se usa la palabra clave **return** en el ejemplo de la función esta devuelve la cadena filepath.

1.8 Estructuras de Datos Básicas

1.8.1 Arreglos

La implementación de un arreglo en C# se muestra a continuación:

```
// creacion de una arreglo
// Declare a single-dimensional array of 5 integers.
int[] array1 = new int[5];

// Declare and set array element values.
int[] array2 = [1, 2, 3, 4, 5, 6];

// Declare a two dimensional array.
int[,] multiDimensionalArray1 = new int[2, 3];

// Declare and set array element values.
int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

// Declare a jagged array.
int[][] jaggedArray = new int[6][];

// Set the values of the first array in the jagged array structure.
jaggedArray[0] = [1, 2, 3, 4];
```

1.8.2 Lista

la implementación de una lista en C# se muestra a continuación:

```
// creacion de una lista
List<int> numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

// Remove odd numbers.
for (var index = numbers.Count - 1; index >= 0; index--)
{
    if (numbers[index] % 2 == 1)
    {
        // Remove the element by specifying
        // the zero-based index in the list.
        numbers.RemoveAt(index);
    }
}

// Iterate through the list.
// A lambda expression is placed in the ForEach method
// of the List(T) object.
numbers.ForEach(
    number => Console.Write(number + " "));
// Output: 0 2 4 6 8
```

1.9 Programación Orientada a Objetos (POO)

1.9.1 Clases y objetos

Implementación de una clase:

```
// creacion de una clase
class Child
{
    private int age;
    private string name;

    // Default constructor:
    public Child()
    {
        name = "N/A";
    }

    // Constructor:
    public Child(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    // Printing method:
    public void PrintChild()
    {
        Console.WriteLine("{0}, {1} years old.", name, age);
    }
}
```

Instanciación de un objeto:

```
class StringTest
{
    static void Main()
    {
        // Create objects by using the new operator:
        Child child1 = new Child("Craig", 11);
        Child child2 = new Child("Sally", 10);

        // Create an object using the default constructor:
        Child child3 = new Child();

        // Display results:
        Console.Write("Child #1: ");
        child1.PrintChild();
        Console.Write("Child #2: ");
        child2.PrintChild();
        Console.Write("Child #3: ");
        child3.PrintChild();
    }
}
/* Output:
```

```
    Child #1: Craig, 11 years old.
    Child #2: Sally, 10 years old.
    Child #3: N/A, 0 years old.
*/
```

1.9.2 Herencia y polimorfismo

la creación de herencia se realiza haciendo uso del signo **":"**, como se muestra a continuación:

```
class GraphicsClass
{
    public virtual void DrawLine() { }
    public virtual void DrawPoint() { }
}
```

Implementación de la herencia:

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public void DrawRectangle() { }
}
```

Para crear polimorfismo se usa la palabra override:

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public override void DrawRectangle() { }
}
```

References

[1] Microsoft. Tour de c# - .net. Recuperado el [fecha de acceso]. [Online]. Available: https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/

[2] ——. Historial de versiones de c# - .net. Recuperado el [fecha de acceso]. [Online]. Available: https://learn.microsoft.com/es-es/dotnet/csharp/whats-new/csharp-version-history

[3] ——. Platform compatibility analyzer. Recuperado el [fecha de acceso]. [Online]. Available: https://learn.microsoft.com/es-es/dotnet/standard/analyzers/platform-compat-analyzer

[4] ——. Tour de c# - estructura del programa. Recuperado el [fecha de acceso]. [Online]. Available: https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/#program-structure