Programación I

ING. ROBERTO ELIAS

Paradigmas de Programación

- Programación orientada a objetos (POO) se utilizan en lenguajes modernos (c#, c++, java, entre otros)
- Programación orientadas a Procedimientos se utilizan en lenguajes antiguos como ser (Fortran, Cobol, Basic, etc)

Programación Orientada Procedimientos.

- Unidades de códigos muy grandes en aplicaciones muy complejas.
- En aplicaciones complejas el código resulta difícil de descifrar.
- Poco reutilizable
- Si existe fallo en alguna línea de código línea de código, es muy probable que el programa caiga.
- Difícil de depurar por otros programadores en caso de necesidad de Error.

Programación Orientada a Objetos

- ¿En que consiste ?

- Trasladar la naturaleza de los objetos de la vida real al código de programación.
- La programación orientada a objetos es un paradigma de programación que se basa en la creación de objetos que contienen tanto datos como métodos para manipular esos datos. En C#, los objetos se definen mediante clases, que actúan como plantillas para crear instancias de objetos.

- ¿cual es la naturaleza de un objeto de la vida real?

- Los objetos tienen un estado, un comportamiento (que pueden hacer?), y propiedades

Por ejemplo un objeto Auto

- ¿Cuál es el estado de un auto?
- Un auto puede estar parado o circulando
- ¿Qué propiedades tiene un auto?
- Un auto tiene un color, un peso, un largo y un ancho.
- ¿Qué comportamiento tiene un auto? Un auto puede arrancar, frenar, acelerar, girar, etc

Ventajas de la programación orientadas a objetos (POO)

- Los programas se pueden dividir en partes, módulos o clases.
- Muy reutilizables.
- Si existe un fallo en alguna de las líneas de código, el programa continua su funcionamiento.
- Encapsulamiento.

Vocabulario de la Programación Orientada Objetos (POO)

- Clase
- Objetos
- Instancia de una clase
- Modularizacion
- Encapsulamiento
- Polimorfismo

Concepto de Clase

- Una clase en C# es un tipo de dato definido por el usuario que describe un conjunto de datos y funciones relacionadas (porción estructurada de código). Las clases se utilizan para representar objetos en el mundo real o abstracto. Por ejemplo, puedes crear una clase "Persona" que tenga atributos como "nombre", "edad", "género", etc. y métodos que realizan acciones relacionadas con las personas, como "caminar" o "hablar".

Clase

- Modelo donde se redactan las características comunes de un grupo de objetos.
- Lo podemos mirar como una plantilla o modelo para obtener las propiedades de los objetos
- Ejemplo
- Chasis de un auto
- Citroen Peugeot

Ejemplo de Clase.

```
2     // Definición de la clase
3   class Persona
4   {
5       // Variables de instancia
6       private string nombre;
7       private int edad;
8
9       // Constructor
10      public Persona(string nombre, int edad)
11      {
12          this.nombre = nombre;
13          this.edad = edad;
14      }
15
16      // Método de instancia
17      public void Saludar()
18      {
19          Console.WriteLine("Hola, mi nombre es " + nombre + " y tengo " + edad + " años.");
20      }
21  }
22
```

Método

- Un método en C# es un bloque de código que se ejecuta cuando se llama a una instancia de una clase o un objeto. Los métodos pueden aceptar parámetros y devolver valores. Pueden ser públicos o privados, lo que determina si pueden ser llamados desde fuera de la clase o sólo desde dentro de la misma. Los métodos son esenciales para la encapsulación de datos y el comportamiento de una clase.

Función en C#

- Por último, una función en C# es un bloque de código que realiza una tarea específica y devuelve un valor. Las funciones pueden ser utilizadas en cualquier parte de un programa para realizar cálculos o manipulaciones de datos. En C#, las funciones son definidas fuera de una clase, mientras que los métodos son definidos dentro de una clase.

Funciones y Métodos

- Las funciones o métodos son trozos de código que nos permiten modularizar nuestro programa, permitiendo dividir funcionalidades para poder usarlos en otros programas.
- Una función devuelve un valor y sino lo hace lo llamaremos método.
- Hay veces que nuestra funciones o métodos necesitan datos para realizar su operación. A esto lo llamaremos parámetros, en la definición de la función o método deberemos declarar esos parámetros.
- En C# suelo llamar función a lo que está definido por fuera de la clase y método si está definido dentro de la clase. Entendiendo que todo esto es conceptual y desde otros lenguajes pueda definirse de otra manera.

Ejemplo de Método o Función

```
1
2     // Definición de la clase
3   class Persona
4   {
5       // Variables de instancia
6       private string nombre;
7       private int edad;
8
9       // Constructor
10      public Persona(string nombre, int edad)
11      {
12          this.nombre = nombre;
13          this.edad = edad;
14      }
15
16      // Método de instancia
17      public void Saludar()
18      {
19          Console.WriteLine("Hola, mi nombre es " + nombre + " y tengo " + edad + " años.");
20      }
21      // Creación de objetos
22      Persona persona1 = new Persona("Juan", 25);
23      Persona persona2 = new Persona("María", 30);
24
25      // Llamada al método de instancia
26      persona1.Saludar();
27      persona2.Saludar();
28  }
29
```

Herencia

- La herencia es un concepto fundamental en la programación orientada a objetos que permite crear nuevas clases basadas en clases existentes. La clase derivada hereda los atributos y comportamientos de la clase base, lo que facilita la reutilización del código y la creación de jerarquías de clases.

- La herencia permite a una clase derivar características de una clase base. Una clase derivada hereda los campos y métodos de su clase base y puede agregar sus propios campos y métodos. La herencia es útil para la reutilización de código y la definición de jerarquías de clases.

Ejemplo de herencia

```
3    // Clase base
4   class Vehiculo
5   {
6       public string Marca { get; set; }
7       public string Modelo { get; set; }
8
9       public void Arrancar()
10      {
11          Console.WriteLine("El vehículo ha arrancado.");
12      }
13  }
14
15  // Clase derivada
16  class Coche : Vehiculo
17  {
18      public void Acelerar()
19      {
20          Console.WriteLine("El coche está acelerando.");
21      }
22  }
23
```

- Se define una clase base llamada Vehículo con dos propiedades (Marca y Modelo) y un método Arrancar(). Luego, se define una clase derivada llamada Coche, que hereda de la clase base Vehículo y agrega un nuevo método Acelerar()

Polimorfismo

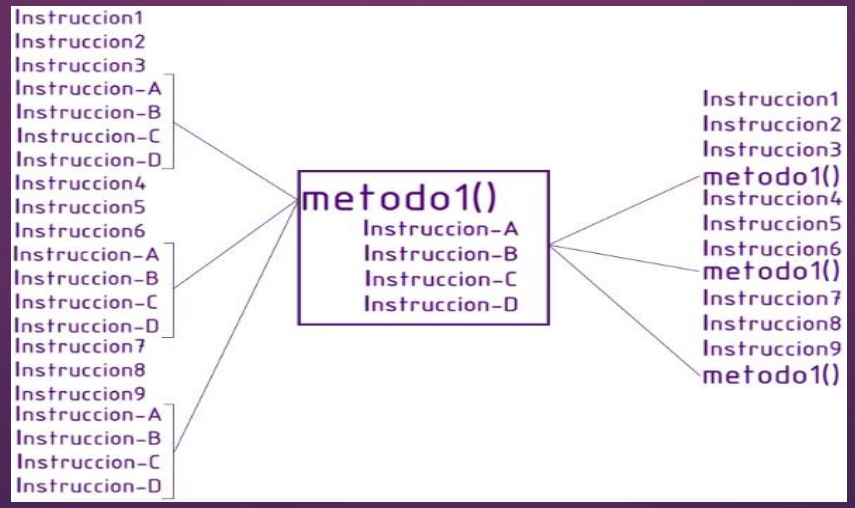
- El polimorfismo se refiere a la capacidad de objetos de diferentes clases de responder al mismo mensaje o método de diferentes maneras. Por ejemplo, una clase "Animal" puede tener un método "hacerSonido()", que será implementado de forma diferente en subclases como "Perro" y "Gato".

```
using System;
class ejemploPolimorfismo {
  static void Main() {

  // Clase base
        class Animal
        {
            public virtual void Sonido()
            {
             Console.WriteLine("El animal hace un sonido.");
            }
        }
// Clase derivada
        class Perro : Animal
        {
            public override void Sonido()
            {
                Console.WriteLine("El perro ladra.");
            }
        }
// Clase derivada
        class Gato : Animal
        {
            public override void Sonido()
            {
                Console.WriteLine("El gato maulla.");
            }
        }
}
}
```

Modularización

- Su función es dividir a tu programa en módulos
- La modularización es el proceso por el cual seleccionamos y agrupamos instrucciones de programación que cumplen una función específica.



Encapsulamiento

- Al dividir el programa en módulos o en varias partes el encapsulamiento permite bloquear el acceso al interior del modulo creado en el programa.
- El encapsulamiento en C#, como en otros lenguajes orientados a objetos, nos permite proteger o crear límites de acceso a las variables y métodos de una clase, controlando con ello el comportamiento fuera y dentro de la misma clase.
- Los tipos de encapsulamiento que nos permite el lenguaje C# son:
- Public
- Private
- Protected

Concepto de Objeto

- En el contexto de la programación orientada a objetos, un objeto es una instancia específica de una clase. Una clase es una plantilla o estructura que define las características y comportamientos de un tipo de objeto en particular. Un objeto se crea a partir de una clase utilizando el proceso de instanciación.
- Cada objeto tiene un estado, que es determinado por los valores de sus variables o propiedades, y puede tener comportamientos definidos por los métodos asociados a la clase. Los objetos también pueden interactuar entre sí a través de mensajes, es decir, llamando a los métodos de otros objetos.
- Por ejemplo, consideremos la clase Persona que define las características y comportamientos de una persona:

Objeto

- Tiene propiedades o atributos (objeto auto):
- Color
- Peso
- Ato
- Ancho
- Tiene un comportamiento (métodos o funciones) ¿Qué son capaces de hacer?
- Arrancar
- Frenar
- Girar

Objetos

- Para acceder a la propiedades de los objetos desde e código.
- Puegeot.color = “Rojo”;
- Puegeot.peso = “1000”;
- Puegeot.alto = “1.75”;
- Accediendo a comportamiento de objetos desde e código.
- Peugeot.arrancar();
- Peugeot.frenar();
- Peugeot.girar();

Ejemplo de Objeto

```
4
5  class Persona
6  {
7      public string Nombre { get; set; }
8      public int Edad { get; set; }
9
10     public void Saludar()
11     {
12         Console.WriteLine("Hola, mi nombre es " + Nombre + " y tengo " + Edad + " años.");
13     }
14 }
15
```

- A partir de esta clase, podemos crear objetos de tipo Persona:

```
5  class Persona
6  {
7      public string Nombre { get; set; }
8      public int Edad { get; set; }
9
10     public void Saludar()
11     {
12         Console.WriteLine("Hola, mi nombre es " + Nombre + " y tengo " + Edad + " años.");
13     }
14 }
15     // a partir de la clase creada podemos crear nuestros objetos, persona1 y persona2.
16     Persona persona1 = new Persona();
17     persona1.Nombre = "Juan";
18     persona1.Edad = 25;
19
20     Persona persona2 = new Persona();
21     persona2.Nombre = "María";
22     persona2.Edad = 30;
```

Ejemplo de Objeto

- En este ejemplo, creamos dos objetos persona1 y persona2 a partir de la clase Persona. Cada objeto tiene su propio estado (nombre y edad) y puede llamar al método Saludar() para mostrar un mensaje personalizado.
- Los objetos son fundamentales en la programación orientada a objetos, ya que permiten modelar y manipular entidades del mundo real de manera eficiente y estructurada. Cada objeto encapsula su estado y comportamiento, y la interacción entre objetos permite construir sistemas complejos.

Métodos Get y Set

- En C#, los get y set son palabras clave utilizadas para implementar las propiedades en una clase. Las propiedades permiten acceder y modificar los valores de las variables privadas de una clase de manera controlada, proporcionando una interfaz más sencilla y segura para interactuar con los datos de un objeto.
- El get se utiliza para obtener el valor de una propiedad, mientras que el set se utiliza para asignar un valor a una propiedad.

```
1  class Persona
2  {
3      private string nombre;  // Variable privada
4
5      public string Nombre   // Propiedad pública
6      {
7          get { return nombre; }   // Obtener el valor de la variable
8          set { nombre = value; }  // Asignar un valor a la variable
9      }
10 }
11
```

Ejemplo de Métodos Get y Set

```
1  class Persona
2  {
3      private string nombre;  // Variable privada
4
5      public string Nombre   // Propiedad pública
6      {
7          get { return nombre; }   // Obtener el valor de la variable
8          set { nombre = value; }  // Asignar un valor a la variable
9      }
10 }
11
```

En este ejemplo, se define una clase Persona con una variable privada nombre. Luego, se define una propiedad pública Nombre que encapsula el acceso a esa variable. Ahora puedes utilizar la propiedad Nombre de la siguiente manera:

```
1  class Persona
2  {
3      private string nombre;  // Variable privada
4
5      public string Nombre   // Propiedad pública
6      {
7          get { return nombre; }   // Obtener el valor de la variable
8          set { nombre = value; }  // Asignar un valor a la variable
9      }
10 }
11
12 Persona persona = new Persona();
13 persona.Nombre = "Juan";   // Asignar un valor a la propiedad
14
15 string nombrePersona = persona.Nombre;   // Obtener el valor de la propiedad
16 Console.WriteLine(nombrePersona);        // Salida: Juan
17
```

Ejemplo Métodos Get y Set

```
1  class Persona
2  {
3      private string nombre;  // Variable privada
4
5      public string Nombre   // Propiedad pública
6      {
7          get { return nombre; }   // Obtener el valor de la variable
8          set { nombre = value; }  // Asignar un valor a la variable
9      }
10 }
11
12 Persona persona = new Persona();
13 persona.Nombre = "Juan";   // Asignar un valor a la propiedad
14
15 string nombrePersona = persona.Nombre;   // Obtener el valor de la propiedad
16 Console.WriteLine(nombrePersona);        // Salida: Juan
17
```

En este caso, puedes asignar un valor a la propiedad Nombre utilizando la sintaxis persona.Nombre = "Juan". El set de la propiedad se encarga de asignar ese valor a la variable privada nombre.
Luego, puedes obtener el valor de la propiedad utilizando persona.Nombre, y el get de la propiedad se encarga de devolver el valor actual de la variable privada nombre.
Las propiedades con get y set permiten un mayor control sobre cómo se accede y se modifica el estado de un objeto, lo que facilita el mantenimiento del encapsulamiento y la consistencia de los datos.

Arreglos Unidimensionales (Vectores)

- En programación, los arreglos unidimensionales, también conocidos como vectores, son estructuras de datos que permiten almacenar una colección ordenada de elementos del mismo tipo. Los elementos se almacenan en posiciones contiguas de memoria y se acceden mediante un índice numérico.

```
9  using System;
10 class HelloWorld {
11   static void Main() {
12       // como crear un vector
13    tipo[] nombreArreglo = new tipo[tamaño];
14   }
15 }
16
```

Arreglos Unidimensionales (Vectores)

- Donde tipo es el tipo de dato de los elementos que se almacenarán en el arreglo, nombreArreglo es el nombre que le das al arreglo y tamaño es el número de elementos que puede contener el arreglo.
- Un ejemplo de cómo crear y trabajar con un arreglo unidimensional en C#:

```
using System;
class HelloWorld {
  static void Main() {

    int[] numeros = new int[5]; // Crear un arreglo de enteros con capacidad para 5 elementos

numeros[0] = 10;  // Asignar un valor al primer elemento del arreglo
numeros[1] = 20;  // Asignar un valor al segundo elemento del arreglo
numeros[2] = 30;  // Asignar un valor al tercer elemento del arreglo
numeros[3] = 40;  // Asignar un valor al cuarto elemento del arreglo
numeros[4] = 50;  // Asignar un valor al quinto elemento del arreglo

Console.WriteLine(numeros[2]);  // Salida: 30 (acceder al valor del tercer elemento del arreglo)

// Recorrer el arreglo e imprimir todos los elementos
for (int i = 0; i < numeros.Length; i++)
{
    Console.WriteLine(numeros[i]);
}
  }
}
```

Arreglos Unidimensionales (Vectores)

```
using System;
class HelloWorld {
  static void Main() {

    int[] numeros = new int[5]; // Crear un arreglo de enteros con capacidad para 5 elementos

numeros[0] = 10;  // Asignar un valor al primer elemento del arreglo
numeros[1] = 20;  // Asignar un valor al segundo elemento del arreglo
numeros[2] = 30;  // Asignar un valor al tercer elemento del arreglo
numeros[3] = 40;  // Asignar un valor al cuarto elemento del arreglo
numeros[4] = 50;  // Asignar un valor al quinto elemento del arreglo

Console.WriteLine(numeros[2]);  // Salida: 30 (acceder al valor del tercer elemento del arreglo)

// Recorrer el arreglo e imprimir todos los elementos
for (int i = 0; i < numeros.Length; i++)
{
    Console.WriteLine(numeros[i]);
}
  }
}
```

En este ejemplo, creamos un arreglo números de tipo entero con capacidad para 5 elementos. Luego, asignamos valores a cada uno de los elementos del arreglo utilizando el índice correspondiente (0 al 4). También podemos acceder y mostrar el valor de un elemento específico utilizando el índice. Además, utilizamos un bucle **for** para recorrer el arreglo y mostrar todos los elementos en la consola. Los arreglos unidimensionales son útiles cuando necesitas almacenar y manipular una colección de elementos del mismo tipo de manera eficiente. Puedes realizar operaciones como asignar valores, acceder a elementos individuales o recorrer el arreglo para realizar alguna acción en cada elemento.

Métodos de ordenamiento de un vector

- Existen varios algoritmos populares para ordenar los elementos de un vector en C#. A continuación, te explicaré dos de los métodos de ordenamiento más comunes: el ordenamiento de burbuja (Bubble Sort) y el ordenamiento rápido (Quick Sort).
- Ordenamiento de burbuja (Bubble Sort): El ordenamiento de burbuja es un algoritmo simple que compara repetidamente pares de elementos adyacentes y los intercambia si están en el orden incorrecto. Este proceso se repite hasta que el vector esté completamente ordenado.

```
using System;
class HelloWorld {
  static void Main() {

   static void BubbleSort(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
  }
}
```

Ordenamiento rápido (Quick Sort):

- En este ejemplo, definimos dos funciones: QuickSort y Partition. La función QuickSort toma un arreglo de enteros arr, un índice bajo low y un índice alto high. Divide el arreglo en subarreglos más pequeños utilizando un elemento pivote y aplica recursivamente el algoritmo QuickSort a cada subarreglo. La función Partition se encarga de seleccionar el pivote y reorganizar los elementos del arreglo de manera que los elementos menores al pivote estén a su izquierda y los elementos mayores estén a su derecha.

```
static void QuickSort(int[] arr, int low, int high)
{
    if (low < high)
    {
        int pivotIndex = Partition(arr, low, high);
        QuickSort(arr, low, pivotIndex - 1);
        QuickSort(arr, pivotIndex + 1, high);
    }
}
static int Partition(int[] arr, int low, int high)
{
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            // Intercambiar elementos
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    // Colocar el pivote en su posición correcta
    int temp2 = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp2;
    return i + 1;
}
```

Caso practico

- Realizar un ejemplo de cada concepto, en el mismo se debe observar las clases, objetos, funciones o métodos implementados, para el código c#.
- Resolverlos siguientes Ejercicios.
- Realizar un programa en c# que me permita ingresar 4 nombres de personas recorrer y mostrar cada uno de los nombres ingresados.
- Realizar un programa en c# que me permita ingresar 5 números enteros, recorrer y mostrar cada uno de los números ingresados.
- A) la solución debe utilizar el concepto de arreglos.
- B) Marcar en la resolución del problema, cuando crea el vector, cuantos vectores se crean, nombre de la clase utilizada.