Unidad 1. Programación Avanzada

1.1. Clases y Métodos Genéricos

¿Porque necesitamos genéricos?

- Maximizar la reutilización del código.
- Cuando implementamos un algoritmo , nosotros quisiéramos utilizarlo para diferentes tipos de datos.
- Ejemplo: Nosotros escribimos un método genérico para ordenara un arreglo de objetos entonces llamamos a un método genérico con un arreglo de un tipo determinado.
- El compilador realiza una verificación de tipos de datos para asegurarse de que el arreglo pasado al método de clasificación solo contiene elementos del mismo tipo.
- Los genéricos proporcionan seguridad de tipo de tiempo de compilación.

Donde se puede utilizar genéricos

- Parámetros
- Clases
- Métodos
- Interfaces
- Delegados

MotivacionMetodosGenericos.sln

```
static void Main(string[] args)
{
    // crear un arreglo de int, double y char
    int[] intArray = { 1, 2, 3, 4, 5, 6 };
    double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
    char[] charArray = { 'H', 'O', 'L', 'A' };
    Console.WriteLine("intArray contiene:");
    MuestraArreglo(intArray);
    Console.WriteLine("doubleArray contiene:");
    MuestraArreglo(doubleArray);
    Console.WriteLine("charArray contiene:");
    MuestraArreglo(charArray);
    Console.ReadKey();
}
// muestra arreglo int
private static void MuestraArreglo(int[] arreglo)
{
    foreach (int element in arreglo)
        Console.Write(element + " ");
    Console.WriteLine("\n");
}
// muestra arreglo double
private static void MuestraArreglo(double[] arreglo)
{
    foreach (double element in arreglo)
        Console.Write(element + " ");
    Console.WriteLine("\n");
}
// muestra arreglo char
private static void MuestraArreglo(char[] arreglo)
{
    foreach (char element in arreglo)
        Console.Write(element + " ");
    Console.WriteLine("\n");
}
```

Métodos Genéricos

- Los métodos genéricos le permiten especificar, con una sola declaración de método, un conjunto de métodos relacionados.
- Si reemplazamos los tipos de elementos en cada método con un nombre genérico T, luego deberíamos de indicar el tipo genérico en los parámetros

MetodosGenericos.sln

```
static void Main(string[] args)
{
    // crear un arreglo de int, double y char
    int[] intArray = { 1, 2, 3, 4, 5, 6 };
    double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
    char[] charArray = { 'H', 'O', 'L', 'A' };
    Console.WriteLine("intArray contiene:");
    MuestraArreglo(intArray);
    Console.WriteLine("doubleArray contiene:");
    MuestraArreglo(doubleArray);
    Console.WriteLine("charArray contiene:");
    MuestraArreglo(charArray);
    Console.ReadKey();
}
// método genérico para mostrar un arreglo
private static void MuestraArreglo<T>(T[] arreglo)
{
    foreach (T elemento in arreglo)
        Console.Write(elemento + " ");
    Console.WriteLine("\n");
}
```

Métodos Genéricos

- Cada lista de parámetros de tipo contiene uno o más parámetros de tipo.
- Un parámetro de tipo es un identificador que se utiliza en lugar de los nombres de tipo reales.
- Los parámetros de tipo se pueden usar para declarar el tipo de retorno, los tipos de parámetros y los tipos de variables locales en una declaración de método genérico.
- El cuerpo de un método genérico se declara como el de cualquier otro método..
- Un parámetro de tipo se puede declarar solo una vez en la lista de parámetros de tipo, pero puede aparecer más de una vez en la lista de parámetros del método.

Declaración de un método genérico

{modificador de acceso} {tipo retorno} {nombre método}<T[,T,…]>({parámetros})
{restricciones}
{
//Implementación
}

MetodosGenericos3.sln

```
using System;

namespace MetodosGenericos3
{
    class Program
    {
        static void Main(string[] args)
        {
            string maxString = Maximo<string>("Carlos", "Pedro");
            Console.WriteLine(maxString);
            int maxInt = Maximo<int>(2, 99);
            Console.WriteLine(maxInt);
            Console.ReadKey();
        }
        public static T Maximo<T>(T uno, T otro) where T : IComparable
        {
            if (uno.CompareTo(otro) > 0) return uno;
            return otro;
        }
    }
}
```

Restriciones

- where T: struct, indica que el argumento debe ser un tipo valor.
- where T: class, indica que T debe ser un tipo referencia.
- where T: new(), fuerza a que el tipo T disponga de un constructor público sin parámetros; es útil cuando desde dentro del método se pretende instanciar un objeto del mismo.
- where T: nombredeclase, indica que el argumento debe heredar o ser de dicho tipo.
- where T: nombredeinterfaz, el argumento deberá implementar el interfaz indicado.
- where T1:T2, indica que el argumento T1 debe ser igual o heredar del tipo, también argumento del método, T2.

Clases Genericas

- Una clase genérica describe una clase de una manera independiente del tipo.
- Encapsulan operaciones que no son específicas de un tipo de datos concreto
- Se utilizan frecuentemente con colecciones como listas
- Cuantos más tipos se puedan parametrizar, más flexible y reutilizable será el código

ClasesGenericas.sln

```
public class Pila<T>
{
    private int tope;
    private T[] datos = new T[10];

    public void Insertar(T objeto)
    {
        datos[tope++] = objeto;
    }

    public T Eliminar()
    {
        return datos[--tope];
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Pila<double> stack = new Pila<double>();
        stack.Insertar(5.5);
        stack.Insertar(10.1);

        double x = stack.Eliminar();
        double y = stack.Eliminar();

        Console.WriteLine("x: {0}, y: {1}", x, y);
    }
}
```

Declaración de una clase genérica

{modificador de acceso} **class** {**nombre clase**}**<T[,T,…]>**{**restricciones**}
{
/ / Cuerpo de la declaración
}

Delegados

Delegados

- Es un nuevo tipo que hace referencia a un método
- Permiten pasar los métodos como parámetros

Sintaxis:

modificador delegate tipo nombre(parámetros);

Modificadores:
- private
- protected
- public
- internal

- Un delegado es una referencia a un método.
- Una variable creada de un tipo delegado representa a un método determinado
- Los delegados se utilizan para enviar métodos como parámetros a otros métodos.

```
public delegate bool TipoOrdenamiento(double x, double y);
```

- Cualquier método puede asignarse a un delegado.
- Deben coincidir…
  - El prototipo del delegado
  - Parámetros
  - Tipo de dato del valor devuelto