Colecciones

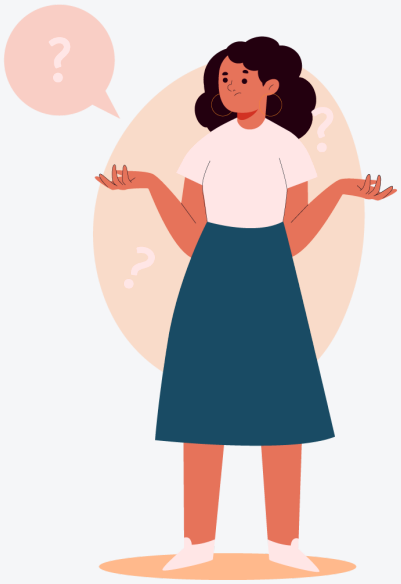
Sitio: Agencia de Habilidades para el Futuro
Curso: Desarrollo de Sistemas Orientado a Objetos 1° D
Libro: Colecciones

Imprimido por: Eduardo Moreno
Día: miércoles, 23 de abril de 2025, 23:21

Tabla de contenidos

Preguntas orientadoras

- ¿Qué pasa cuando tenemos, por ejemplo 10 automóviles? ¿Debemos crear 10 variables del tipo automóvil? ¿Se pueden asociar a todas con un mismo nombre?
- ¿Cuáles son las operaciones comunes que se pueden realizar con un List?
- ¿Qué métodos se utilizan para obtener el tamaño de un List y comprobar si está vacío?



Introducción

Muchas veces las aplicaciones necesitan agrupar las variables ya que tienen un sentido en sí mismas. Para ello hemos visto que podemos tener clases.

Pero los programas van creciendo en complejidad, y necesitamos operar sobre las variables y los objetos, agruparlos por aquellos que tienen un sentido común… Por ejemplo tener una lista de alumnos de una cátedra.

Eso lo haremos con las listas, vamos a declarar e inicializar un List de objetos, dejándolo listo para instanciar cada uno de sus elementos y agregarlos a la lista.

Luego poder recorrer esa lista e ir accediendo a los atributos y métodos de cada objeto que hay en ella.

Colecciones

Conceptualizando

Una colección de datos se refiere a una estructura de datos homogéneo, es decir, todos ellos del mismo tipo (<T>) permitiendo almacenar y manipular múltiples elementos de manera organizada y eficiente. Estas colecciones proporcionan una variedad de métodos y propiedades para realizar operaciones comunes, como agregar, eliminar, buscar y ordenar elementos.

Tipos de colecciones

Algunos tips

Las colecciones de datos en C# se encuentran en el espacio de nombres System.Collections y System.Collections.Generic.

Los tipos de colecciones más comunes son List y Array. En C#, tanto List<T> como los arreglos (array) son estructuras de datos utilizadas para almacenar colecciones de elementos. Sin embargo, existen algunas diferencias clave entre ellos:

Tamaño dinámico: Una de las principales diferencias es que List<T> tiene un tamaño dinámico en la cantidad de elementos a guardar en la colección, lo que significa que puede crecer o reducirse según sea necesario. Puedes agregar, insertar y eliminar elementos fácilmente en un List<T> en cualquier momento. Por otro lado, los arreglos tienen un tamaño fijo el cual debemos establecer en tiempo de diseño (al momento de estar escribiendo el c ódigo) y no se pueden cambiar después de su creación.

Tipo de datos: Los arreglos pueden contener elementos de cualquier tipo de datos, incluyendo tipos primitivos (int, char, etc.) y tipos de referencia a objetos de clases. Por otro lado, List<T> se utiliza con tipos de datos genéricos y solo puede contener elementos del tipo especificado cuando se declara, lo que proporciona seguridad de tipos durante la compilación.

Funcionalidad adicional: List<T> proporciona una amplia gama de métodos y propiedades que facilitan la manipulación de la lista, como Add, Remove, Find, Count, entre otros. Además, se puede acceder a los elementos de un List<T> mediante índices utilizando la notación de corchetes ([]). Los arreglos también tienen algunas funcionalidades básicas, como la capacidad de acceder a los elementos mediante índices y la capacidad de obtener el tamaño del arreglo utilizando la propiedad Length. Sin embargo, los arreglos no tienen métodos adicionales integrados como List<T>, por lo que la manipulación de los elementos puede requerir más código manual.

Veamos en este video algunos ejercicios de creación de Arrays:

Ejercicios C# - Arrays #1 - Recorriendo arrays

En general, si necesitas una colección que pueda cambiar de tamaño dinámicamente y proporcionar una funcionalidad más amplia, como agregar, eliminar o buscar elementos, es más común y recomendado utilizar List<T>. Por otro lado, si tienes una colección de tamaño fijo o necesitas trabajar con elementos de tipos específicos de manera más eficiente, los arreglos pueden ser una opción adecuada.

La elección de la colección a utilizar depende de los requisitos específicos de tu programa, como la eficiencia en la inserción, eliminación o búsqueda, así como las características particulares de los datos que deseas almacenar y manipular.

En nuestro ejemplo trabajaremos con list<T>

¡Para tener en cuenta! Es posible y muy útil crear listas de datos que son clases. Estas listas son ampliamente usados para almacenar datos complejos como registros de empleados, artículos, libros, etc.

Así, se logra representar por un lado estructuras complejas con diferentes tipos de datos a través de una clase y además a través de las colecciones se tiene un medio para agrupar una gran cantidad de datos de esa clase.

A continuación, trabajemos con un ejemplo.

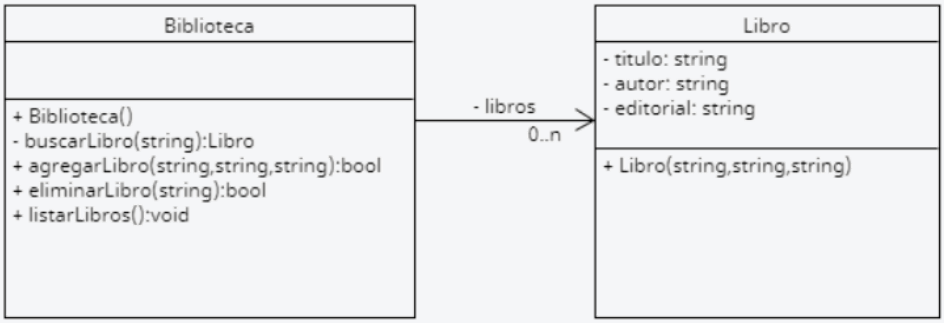
Trabajando un ejemplo

Enunciado

En un determinado pueblo tienen una biblioteca que cuenta con libros. De cada libro se conoce su titulo, genero y autor. La biblioteca dispone de un ejemplar de cada uno de ellos y los identifica por su título.

¿Cuál es la acción a procesar? En el siguiente ejemplo se muestra como se declara la variable biblioteca con una lista de libros y cómo se trabaja en forma encapsulada con la lista. Cada elemento de la lista representa un libro, y cada libro tiene sus atributos y sus datos.

Nuestro UML quedaría de la siguiente manera:



Desglosemos este ejemplo general en otras acciones.

Ejemplo clase Test

Retomando nuestro ejemplo, partimos de una clase Test que será la encargada de probar nuestro sistema.

¿Por qué estamos haciendo este procedimiento? De esta forma simulamos la interacción con el usuario ya que al trabajar con colecciones necesitamos varios datos como para realizar pruebas. Por ejemplo tenemos el método cargarLibros que le pasamos la cantidad de libros a cargar y evita escribir cada uno de los campos simulando como si se los pidiese al usuario.

¡Veámoslo en pantalla! Creamos un objeto biblioteca del tipo clase Biblioteca.

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

namespace Colecciones
{
    0 referencias
    internal class Test
    {
        0 referencias
        static void Main(string[] args)
        {
            Biblioteca biblioteca = new Biblioteca();
            cargarLibos(10);
            cargarLibos(2);
            biblioteca.listarLibros();
            biblioteca.eliminarLibro("Libro5");
            biblioteca.listarLibros();
            void cargarLibos(int cantidad){
                bool pude;
                for (int i = 1; i <= cantidad; i++)
                {
                    pude = biblioteca.agregarLibro("Libro" + i, "Autor" + i, "Editorial" + i);
                    if (pude)
                        Console.WriteLine("libro" + i + " agragado correctamente.");
                    else
                        Console.WriteLine("libro" + i + " Ya existe en la biblioteca.");
                }
            }
        }
    }
}
```

A continuación, profundicemos sobre la clase Biblioteca.

Ejemplo clase Biblioteca

Continuemos con nuestro ejemplo base. Ahora, la clase Biblioteca consta de una lista del tipo Libros llamada libros.

Observemos que la lista, como cualquier otro atributo de una clase como vimos en el tema de encapsulamiento, siempre será private ya que ninguna otra clase debe manipular la lista de libros más que la biblioteca.

¡Veámoslo en pantalla!

En el constructor de Biblioteca instanciamos la lista de libros para poder utilizarla:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Colecciones
{
    3 referencias
    internal class Biblioteca
    {
        private List<Libro> libros;

        1 referencia
        public Biblioteca()
        {
            this.libros = new List<Libro>();
        }
```

Luego creamos un método para realizar búsquedas de un libro por su nombre:

```
private Libro buscarLibro(string titulo)
{
    Libro libroBuscado = null;
    int i = 0;
    while (i < libros.Count && !libros[i].getTitulo().Equals(titulo))
        i++;
    if (i != libros.Count)
        libroBuscado = libros[i];
    return libroBuscado;
}
```

¿Qué podemos visualizar? En la búsqueda ciclamos con un while, de esta forma una vez que se encuentra el elemento buscado (en este caso un libro por su titulo) salimos del ciclo sin la necesidad de seguir recorriendo.

¿Cómo continuamos? Utilizamos el Count de la lista de libros para que nos diga la cantidad de elementos que tiene la lista.

Otra cosa a tener en cuenta, para acceder a cada elemento de la lista se lo hace a traves de su indice con corchetes [ ] siendo siempre el primer elemento de la lista el índice cero 0.

Una vez que encontramos el libro, retornamos la referencia al mismo para poderlo trabajar o bien un valor null.

Los métodos

¡Relacionemos ideas!

¿Te acordás lo que vimos en la semana anterior? ¿Los métodos? ¡Pongamos lo en práctica!

Luego de encontrar el libro y retomar la referencia, creamos utilizando métodos. ¿Qué métodos vamos a utiliizar para esta acción? Los métodos agregarLibro que recibe por parámetro los datos de un libro a agregar, lo busca en la lista y si no lo encuentra, crea una nueva instancia de Libro y lo agrega utilizando el método Add

Por otra parte, el método listarLibros que recorre toda la colección de libros y los muestra por pantalla.

Y por último, el método eliminarLibro que recibe por parámetro el titulo de un libro a eliminar, lo busca en la lista y si lo encuentra, lo elimina de la lista utilizando el método Remove.

Tanto en el método agregarLibro como en eliminarLibro devolvemos un valor booleano que nos indica el resultado de la operación y así poderle informar al usuario si se pudo realizar o no.

```
public bool agregarLibro(string titulo, string autor, string editorial)
{
    bool resultado = false;
    Libro libro;
    libro = buscarLibro(titulo);
    if (libro == null)
    {
        libro = new Libro(titulo, autor, editorial);
        libros.Add(libro);
        resultado = true;
    }
    return resultado;
}

2 referencias
public void listarLibros()
{
    foreach (var libro in libros)
        Console.WriteLine(libro);
}

1 referencia
public bool eliminarLibro(string titulo)
{
    bool resultado = false;
    Libro libro;
    libro = buscarLibro(titulo);
    if (libro != null)
    {
        libros.Remove(libro);
        resultado = true;
    }
    return resultado;
}
```

A continuación, ampliemos sobre la clase Libro.

Ejemplo clase Libro

En este caso lo único que necesitamos son sus atributos (siempre privados), el constructor parametrizado y el método ToString utilizado desde el método listarLibros de la clase Biblioteca.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Colecciones
{
    8 referencias
    internal class Libro
    {
        private string titulo;
        private string autor;
        private string editorial;

        1 referencia
        public Libro(string titulo, string autor, string editorial)
        {
            this.titulo = titulo;
            this.autor = autor;
            this.editorial = editorial;
        }

        1 referencia
        public string getTitulo()
        {
            return titulo;
        }

        0 referencias
        public override string ToString()
        {
            return "Titulo: " + titulo + " Autor: " + autor + " Editorial: " + editorial;

        }
    }
}
```

En resumen

En esta semana hemos visto cómo definir un List de objetos.

Hemos aprendido a declararlos, luego a inicializarlos (instanciarse), para luego asignar los objetos.

Teniendo ya los objetos instanciados y agregados a la lista, aprendimos a acceder a sus propiedades y métodos.

Cliqueá sobre las tarjetas para leer el resumen de los conceptos claves de este módulo.

Las colecciones y las listas en C# son herramientas esenciales para trabajar con conjuntos de datos de manera flexible y eficiente. Aprovechar sus funcionalidades integradas y su capacidad de expansión nos permite escribir código más limpio y efectivo al manipular y procesar conjuntos de datos en nuestros programas.

¡Explora y experimenta con las colecciones y las listas en C# para descubrir todas las posibilidades que ofrecen en tus proyectos de programación!