Colecciones

<https://www.youtube.com/watch?v=_f255TeqteQ> Video 63

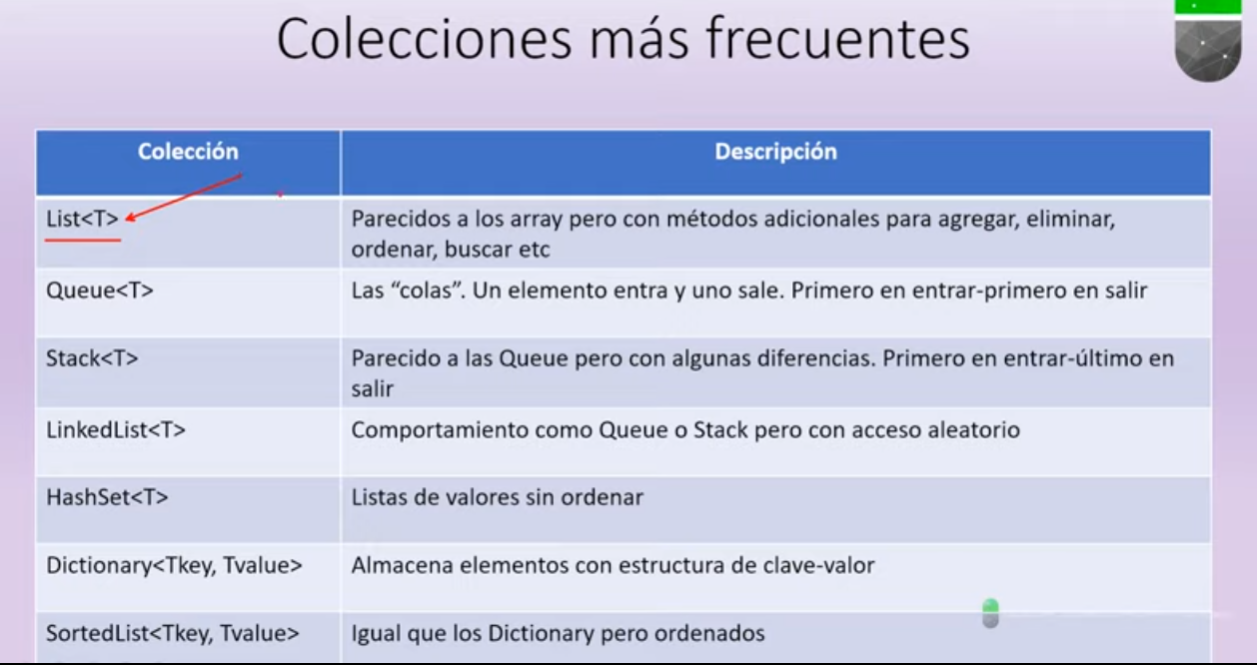
Pildorasinformaticas

Son clases que pertenecen al namespace System.Collection.Generic

Permiten almacenar elementos.

Son clases genéricas, como indica su propio namespace.

No tienen las limitaciones de los arrays, a cambio de un mayor consumo de recursos. Se pueden ordenar, añadir, eliminar, buscar, etc, sus elementos.



Busca “.net library” y va a docs.microsoft.com

Luego, ahí busca collections y le aparece el namespace System.Collection.Generic Ahí hay mucha data sobre las clases. Va a list y hay mucha data. “Es un rollo” dice.

Listas

Para crear una lista hace falta que se esté usando la clase System.Collection.Generic, que esté dentro de los using.

List<int> numeros = new List<int>(); //Así se declara una colección de tipo lista

numeros.Add(5); //Agregar elementos

numeros.Add(7);

/\*for (int i = 0; i < 3; i++) //Así se recorre una lista

{

Console.writeline (numeros[i]);

}\*/

//Hasta acá todo lo comparte con un array.

//Acá le sumo a la lista, los valores de un array.

int[] listanumeros = new int[] { 3, 6, 8, 10, 50 };

for (int i = 0; i < 5; i++)

{

numeros.Add(listanumeros[i]);

}

for (int i = 0; i < 7; i++) //Así se recorre una lista

{

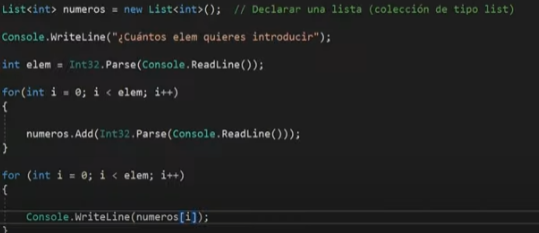
Console.WriteLine (numeros[i]);

}

“Las listas permite agregar elementos en tiempo de ejecución, cosa que el array no permite” (muchacho). Es decir, si se declaró un array con 5 elementos, eso luego no va a cambiar, distinto con la lista que arrancó con 2 y terminó con 7.



Primero agrega una línea de texto para preguntar al usuario.  
Despues una segunda para capturar lo que el usuario escriba. Convirtiendo lo que se escriba de string a int.



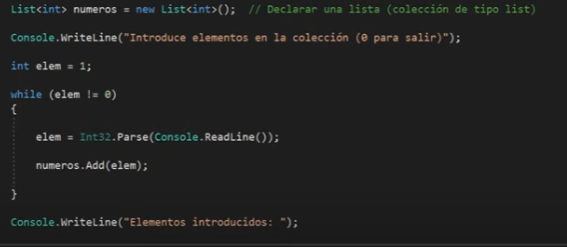
Con el primer for se suman los valores, con el segundo se printea.

El primer numero que escribe define la cantidad de elementos, los otros son ya los valores de la lista. Será así por defecto. El resultado final es desprolijo, pero bue.

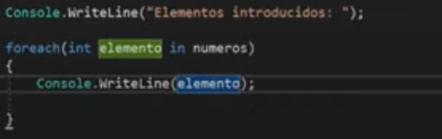
Dice luego que en el for, podría ser “i<números.Count;



“Los bucles for each son mas útiles con las colecciones”.



Antes de entrar al foreach, crea otra lista, en la que el usuario agregará elementos hasta que ingrese el número 0, ahí se termina el programa.



Por cada elemento que hay en la colección números, imprimirá el elemento. No se bien por qué es mejor.

La variable elemento, la declara ahí mismo.

El 0 queda dentro de la colección lo cual no está bien, ya que se lo puso para salir del bucle. 

Así se elimina el último elemento.

Linkedlist Video 64

Linkedlist. Es similar a list.

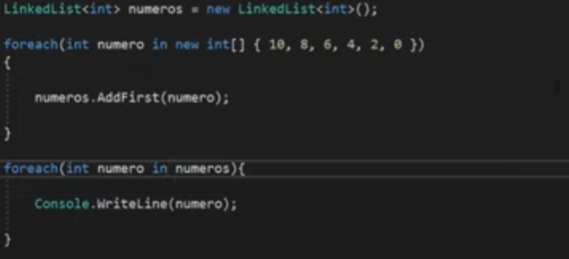
Cuando uno elimina un elemento de una list, como no pueden quedar espacios vacíos, los elementos con un índice superior se correrán uno hacia abajo para que no queden espacios vacíos. Eso hace, que al tener que moverse muchos números se consuma mas memoria.

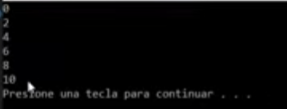
Cuando se van a usar colecciones a las que se le agreguen o eliminen elementos constantemente, será mas eficiente usar una linkedlist.

La declaración es idéntica pero en vez de list se usa linkedlist.

Luego dentro del foreach crea un array con sus respectivos valores.

El método addfirst pone al primer número en primer lugar, luego al segundo número lo va a primer lugar, y así. Por eso invierte el orden.





Addlast lo agrega al final. Conservaría el orden original.

Arrays video 36

Arreglos o matrices

Declaración Tipodedato [] nombredearray;

Iniciaciación: nombredearray= new int[numerodevalores];

En esto es igual que

Int edad;

Edad=27;

Nombredearray [0]=15;

Nombredearray [1]=23;

Nombredearray [2]=56;

Otra forma de declarar, iniciar y dar valores al mismo tiempo es:

Int [] nombreDeArray= {15,23,56};

Por defecto, si no se les pasa valores, estos son 0.

Así se printea:

int[] edades;

edades = new int[4];

edades[0] = 14;

edades[1] = 24;

edades[2] = 55;

edades[3] = 34;

for (int i = 0; i < edades.Length ; i++)

{

Console.WriteLine(edades[i]);

}

Video 37

Array implícito. No se especifica cantidad ni tipo de datos:

var datos = new[] { "Juan", "Díaz", "España", };

Array de objetos:

empleados[] arrayEmpleados = new empleados[2];

empleados Ana = new empleados("Ana", 27); //esto es \*

arrayEmpleados[0] = new empleados("Sarah", 36); // esta es una alternativa

arrayEmpleados[1] = Ana; //esta es la otra alternativa, requiere de \* ya que si no, daría error

}

class empleados //se van a construir objetos de tipo empleados

{

public empleados (string nombre, int edad) //Este public es el constructor de la clase

{

this.nombre = nombre;

this.edad = edad;

}

string nombre;

int edad;

//arrays de tipos anónimos

var personas = new[]

{

new {Nombre= "Juan", Edad=19},

new {Nombre="María", Edad=49},

new {Nombre="Diana", Edad=35}

};

Video 38

Repasa el for y las formas de printear un array. Nada nuevo

Video 39

Explica el .Length.

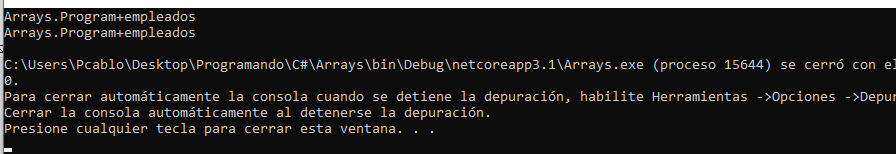
Acá se está printeando mal el array de objetos.

for (int i = 0; i < arrayEmpleados.Length; i++)

{

Console.WriteLine(arrayEmpleados[i]);

}



Hay 2 alternativas:  
una es poniendo arrayEmpleados [i].nombre

Previamente hay que definir nombre como variable pública, si no no se podría acceder.

Console.WriteLine(arrayEmpleados[i].nombre);

Esto no debería hacerse “casi nunca”. Los campos de clase deben ir encapsulados, no ser públicos. Ya que podría modificarse. Es mas, dice de ponerle private. Esto se relaciona con el tema de la herencia.

Getter.

public string getInfo ()

{

return "Nombre del empleado: "+ nombre +". Edad: " + edad;

}

Y luego:

for (int i = 0; i < arrayEmpleados.Length; i++)

{

Console.WriteLine(arrayEmpleados[i].getInfo());

}



For Each es útil para recorrer arrays donde no se sabe su extensión. Pierde flexibilidad ya que se recorren todos los elementos, no te permite modificarlos.

foreach (empleados iterador in arrayEmpleados) //la variable iteradora creada debe ser del mismo tipo que estamos almacenando en el array

{

Console.WriteLine(iterador.getInfo());

}

Así se imprime con un foreach el array personas.

var personas = new[]

{

new {Nombre= "Juan", Edad=19},

new {Nombre="María", Edad=49},

new {Nombre="Diana", Edad=35}

};

foreach (var iterador in personas)

{

Console.WriteLine(iterador);

}

Video 40

Arrays como parámetros de métodos.

No necesariamente los parámetros tienen que ser primitivos (double, string, etc).

static void Main(string[] args)

{

int[] numeros = new int[4];

numeros[0] = 7;

numeros[1] = 44;

numeros[2] = 53;

numeros[3] = 3;

procesaDatos(numeros); //este número se almacena en datosArray.

}

static void procesaDatos(int [] datosArray) //static porque solo se trabaja con una clase. Se llama a este desde el main

{

foreach (int i in datosArray)

{

Console.WriteLine(i);

}

Acá algo nuevo, con mas gracia. No se cuál es la novedad:

static void Main(string[] args)

{

int[] arrayElementos = LeerDatos();

Console.WriteLine("Esto se está imprimiendo desde el main");

for (int a = 0; a < arrayElementos.Length; a++)

{

Console.WriteLine(arrayElementos[a]);

}

}

static int[] LeerDatos ()

{

Console.WriteLine("¿Cuántos elementos quieres que tenga el array?");

string respuesta = Console.ReadLine();

int numElementos = Int32.Parse(respuesta);

int[] datos = new int[numElementos];

for (int i = 0; i < numElementos; i++)

{ Console.WriteLine($"Introduce el dato para la posición {i}");

respuesta = Console.ReadLine();

int datosElemento = Int32.Parse(respuesta);

datos[i] = datosElemento;

}

return datos;

}

El signo $ hace que {i} dentro del string devuelva el valor de i, y que no se printee {i}. También se podría hacer ”+ i.

Queue colas

Video 65

Filosofía FIFO. Primer en entrar es el primero en salir.

Sirven para: apps que maneje procesos secenciales (primero hago X, luego Y). Hasta que no se termina una tarea, el procesador no pasa a la siguiente.

class Program

{

static void Main(string[] args)

{

Queue<int> numeros = new Queue<int>();

//rellenar o agregar elementos a la cola

foreach (int numero in new int [5] { 2, 4, 6, 8, 10 })

{

numeros.Enqueue(numero);

}

//recorrer, imprimir la cola.

Console.WriteLine("Recorriende el Queue");

foreach (var numero in numeros)

{

Console.WriteLine(numero);

}

//Eliminando elementos

numeros.Dequeue(); //dequeue no admite argumentos. Quita el primer elemento ingresado e ingresa un elemento al ppio (FIFO)

Console.WriteLine("Eliminando elemento:");

Console.WriteLine("Recorriende el Queue");

foreach (var numero in numeros)

{

Console.WriteLine(numero);

}

Console.WriteLine("Agregando un elemento:");

numeros.Enqueue(769);

foreach (var numero in numeros)

{

Console.WriteLine(numero);

}

}

}

Stack

Filosofía LIFO. Al contrario de la queue, el último en entrar es el primero en salir.

En vez de queue va stack; en vez de enqeue va Push; en vez de dequeue va Pop:

class Program

{

static void Main(string[] args)

{

Stack<int> numeros = new Stack<int>();

//rellenar o agregar elementos al stack

foreach (int numero in new int [5] { 2, 4, 6, 8, 10 })

{

numeros.Push(numero);

}

//recorrer, imprimir el stack

Console.WriteLine("Recorriendo el Stack");

foreach (var numero in numeros)

{

Console.WriteLine(numero);

}

//Eliminando elementos

numeros.Pop(); //dequeue no admite argumentos. Quita el primer elemento ingresado e ingresa un elemento al ppio (FIFO)

Console.WriteLine("Eliminando elemento:");

Console.WriteLine("Recorriende el Queue");

foreach (var numero in numeros)

{

Console.WriteLine(numero);

}

Console.WriteLine("Agregando un elemento:");

numeros.Push(769);

foreach (var numero in numeros)

{

Console.WriteLine(numero);

}

}

}

}

Dictionary

El método Add pide 2 parámetros (Tkey, TValue);

Cada elemento está identificado con una clave, que no se puede repetir.

Util para manejar información de bases de datos. Es la colección que mas recursos consume.

static void Main(string[] args)

{

Dictionary<string, int> edades = new Dictionary<string, int>();

//rellenación de 2 formas

edades.Add("Pepe", 42);

edades.Add("Laucha", 35);

edades.Add("Maxi", 45);

edades["María"] = 25;

edades["Antonio"] = 29;

//recorrición

foreach(KeyValuePair<string,int> persona in edades)

{

Console.WriteLine("Nombre: "+ persona.Key+ " .Edad: "+ persona.Value);

}

foreach (KeyValuePair<string, int> persona in edades)

{

Console.WriteLine("Nombre: {0} Edad {1}", persona.Key, persona.Value);

}

