



2021

# C#: Estructuras dinámicas de memoria

ArrayList, List, Dictionary, Queue, Stack,  
Hashtable, SortedList, LinkedList

Rafael Alberto  
Moreno Parra

Contenido

Otros libros del autor ..... 2

Página web del autor y canal en Youtube ..... 3

Sitio en GitHub ..... 3

Licencia del software ..... 3

Marcas registradas ..... 3

Dedicatoria ..... 4

Introducción..... 5

El ArrayList ..... 6

    Adicionar, tamaño, buscar e imprimir ..... 6

    Borrar elemento ..... 8

    Cambiar Elemento ..... 9

    Insertar Elemento..... 10

    Referenciar con una variable, un rango de la lista ..... 11

    Tres formas de recorrer un ArrayList ..... 13

    Borrar completamente un ArrayList ..... 14

    Borrar un rango en un ArrayList..... 15

    Guardar el ArrayList en un arreglo estático ..... 16

    Agregar un arreglo estático a un ArrayList..... 17

    Inserta un arreglo estático en una determinada posición del ArrayList..... 18

    Insertar varios ArrayList dentro de un ArrayList..... 19

    Invertir un ArrayList ..... 20

    Invertir un rango de datos en el ArrayList..... 21

    Ordena un ArrayList..... 22

    Búsqueda Binaria ..... 23

    Capacidad del ArrayList ..... 24

    Detectar el tipo de dato del elemento del ArrayList ..... 25

List ..... 26

    Métodos similares a los de ArrayList ..... 26

    Comparativa de desempeño de ArrayList vs List vs Arreglo estático ..... 31

    Lista de objetos..... 42

    Listas en Listas ..... 44

Dictionary ..... 58

    Uso de llaves ..... 58

    Llaves tipo string..... 59

    Manejo de objetos en un Dictionary..... 60

Queue (Cola)..... 62

    Dato definido en la cola ..... 64

    Objetos en la cola..... 65

Stack (Pila) ..... 67

    Dato definido en la pila..... 69

    Objetos en la pila ..... 70

Hashtable ..... 72

    Manejo de objetos en un Hashtable ..... 74

SortedList ..... 76

LinkedList ..... 77

    Objetos en LinkedList..... 79

Bibliografía..... 81

## Otros libros del autor

Libro 16: "C#. Programación Orientada a Objetos". En Colombia 2020. Págs. 90. Libro y código fuente descargable en: <https://github.com/ramsoftware/C-Sharp-POO>

Libro 15: "C#. Estructuras básicas de memoria.". En Colombia 2020. Págs. 60. Libro y código fuente descargable en: <https://github.com/ramsoftware/EstructuraBasicaMemoriaCSharp>

Libro 14: "Iniciando en C#". En Colombia 2020. Págs. 72. Libro y código fuente descargable en: <https://github.com/ramsoftware/C-Sharp-Iniciando>

Libro 13: "Algoritmos Genéticos". En Colombia 2020. Págs. 62. Libro y código fuente descargable en: <https://github.com/ramsoftware/LibroAlgoritmoGenetico2020>

Libro 12: "Redes Neuronales. Segunda Edición". En Colombia 2020. Págs. 108. Libro y código fuente descargable en: <https://github.com/ramsoftware/LibroRedNeuronal2020>

Libro 11: "Capacitándose en JavaScript". En Colombia 2020. Págs. 317. Libro y código fuente descargable en: <https://github.com/ramsoftware/JavaScript>

Libro 10: "Desarrollo de aplicaciones para Android usando MIT App Inventor 2". En Colombia 2016. Págs. 102. Ubicado en: <https://openlibra.com/es/book/desarrollo-de-aplicaciones-para-android-usando-mit-app-inventor-2>

Libro 9: "Redes Neuronales. Parte 1.". En Colombia 2016. Págs. 90. Libro descargable en: <https://openlibra.com/es/book/redes-neuronales-parte-1>

Libro 8: "Segunda parte de uso de algoritmos genéticos para la búsqueda de patrones". En Colombia 2015. Págs. 303. En publicación por la Universidad Libre – Cali.

Libro 7: "Desarrollo de un evaluador de expresiones algebraicas. **Versión 2.0.** C++, C#, Visual Basic .NET, Java, PHP, JavaScript y Object Pascal (Delphi)". En: Colombia 2013. Págs. 308. Ubicado en: <https://openlibra.com/es/book/evaluador-de-expresiones-algebraicas-ii>

Libro 6: "Un uso de algoritmos genéticos para la búsqueda de patrones". En Colombia 2013. En publicación por la Universidad Libre – Cali.

Libro 5: Desarrollo fácil y paso a paso de aplicaciones para Android usando MIT App Inventor. En Colombia 2013. Págs. 104. Estado: Obsoleto (No hay enlace).

Libro 4: "Desarrollo de un evaluador de expresiones algebraicas. C++, C#, Visual Basic .NET, Java, PHP, JavaScript y Object Pascal (Delphi)". En Colombia 2012. Págs. 308. Ubicado en: <https://openlibra.com/es/book/evaluador-de-expresiones-algebraicas>

Libro 3: "Simulación: Conceptos y Programación" En Colombia 2012. Págs. 81. Ubicado en: <https://openlibra.com/es/book/simulacion-conceptos-y-programacion>

Libro 2: "Desarrollo de videojuegos en 2D con Java y Microsoft XNA". En Colombia 2011. Págs. 260. Ubicado en: <https://openlibra.com/es/book/desarrollo-de-juegos-en-2d-usando-java-y-microsoft-xna> . ISBN: 978-958-8630-45-8

Libro 1: "Desarrollo de gráficos para PC, Web y dispositivos móviles" En Colombia 2009. ed.: Artes Gráficas Del Valle Editores Impresores Ltda. ISBN: 978-958-8308-95-1 v. 1 págs. 317

Artículo: "Programación Genética: La regresión simbólica". Entramado ISSN: 1900-3803 ed.: Universidad Libre Seccional Cali v.3 fasc.1 p.76 - 85, 2007

## Página web del autor y canal en Youtube

Investigación sobre Vida Artificial: <http://darwin.50webs.com>

Canal en Youtube: <http://www.youtube.com/user/RafaelMorenoP> (dedicado principalmente al desarrollo en C#)

## Sitio en GitHub

El código fuente se puede descargar en <https://github.com/ramsoftware/CSharpDinamica>

## Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL “Lesser General Public License” [1]



## Marcas registradas

En este libro se hace uso de las siguientes tecnologías registradas:

Microsoft ® Windows ® Enlace: <http://windows.microsoft.com/en-US/windows/home>

Microsoft ® Visual Studio 2019 ® Enlace: <https://visualstudio.microsoft.com/es/vs/>

# Dedicatoria

A mis padres, a mi hermana....

Y a mi tropa gatuna: Sally, Suini, Vikingo, Grisú, Capuchina, Milú y mi recordada Tammy.

# Introducción

Siguiendo con la serie de libros de C# que he escrito anteriormente:

1. Iniciando en C# <https://github.com/ramsoftware/C-Sharp-Iniciando>
2. C#. Estructuras básicas de memoria <https://github.com/ramsoftware/EstructuraBasicaMemoriaCSharp>
3. C#. Programación Orientada a Objetos <https://github.com/ramsoftware/C-Sharp-POO>

Este libro finalizado en enero de 2021, se concentra en las estructuras de datos dinámicas de memoria en C# como ArrayList, List, Dictionary, Queue, Stack, Hashtable, SortedList y LinkedList.

El código fuente se puede descargar de GitHub en: <https://github.com/ramsoftware/CSharpDinamica>

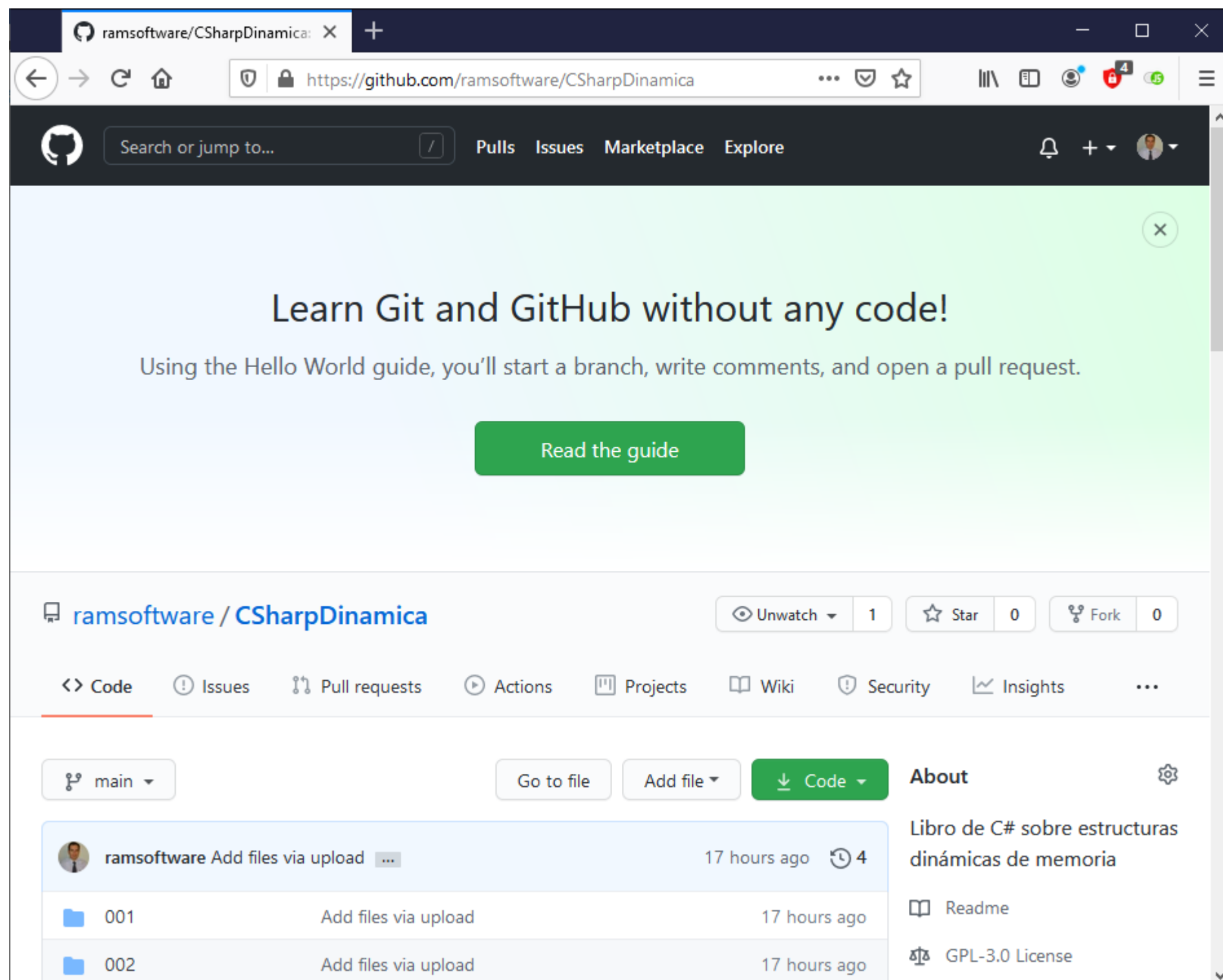


Ilustración 1: Sitio GitHub del libro y el código fuente:

# El ArrayList

## Adicionar, tamaño, buscar e imprimir

En C# tenemos el ArrayList [2], que es un contenedor de objetos de cualquier tipo.

Carpeta 001. Program.cs

```
using System;
using System.Collections;

namespace Estructuras {
    class Program {
        static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new ArrayList();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");
            ListaAnimales.Add("Calamar");
            ListaAnimales.Add("Gaviota");
            ListaAnimales.Add("Foca");
            ListaAnimales.Add("Manatíes");
            ListaAnimales.Add("Ballena con barba");
            ListaAnimales.Add("Peces Guppy");
            ListaAnimales.Add("Orca");
            ListaAnimales.Add("Medusas");
            ListaAnimales.Add("Mejillones");
            ListaAnimales.Add("Caracoles");

            //Tamaño la lista
            int tamano = ListaAnimales.Count;
            Console.WriteLine("Tamaño de la lista: " + tamano);

            //Traer un determinado elemento de la lista
            int posicion = 7;
            string texto = ListaAnimales[posicion].ToString();
            Console.WriteLine("Elemento en la posición " + posicion + " es: " + texto);

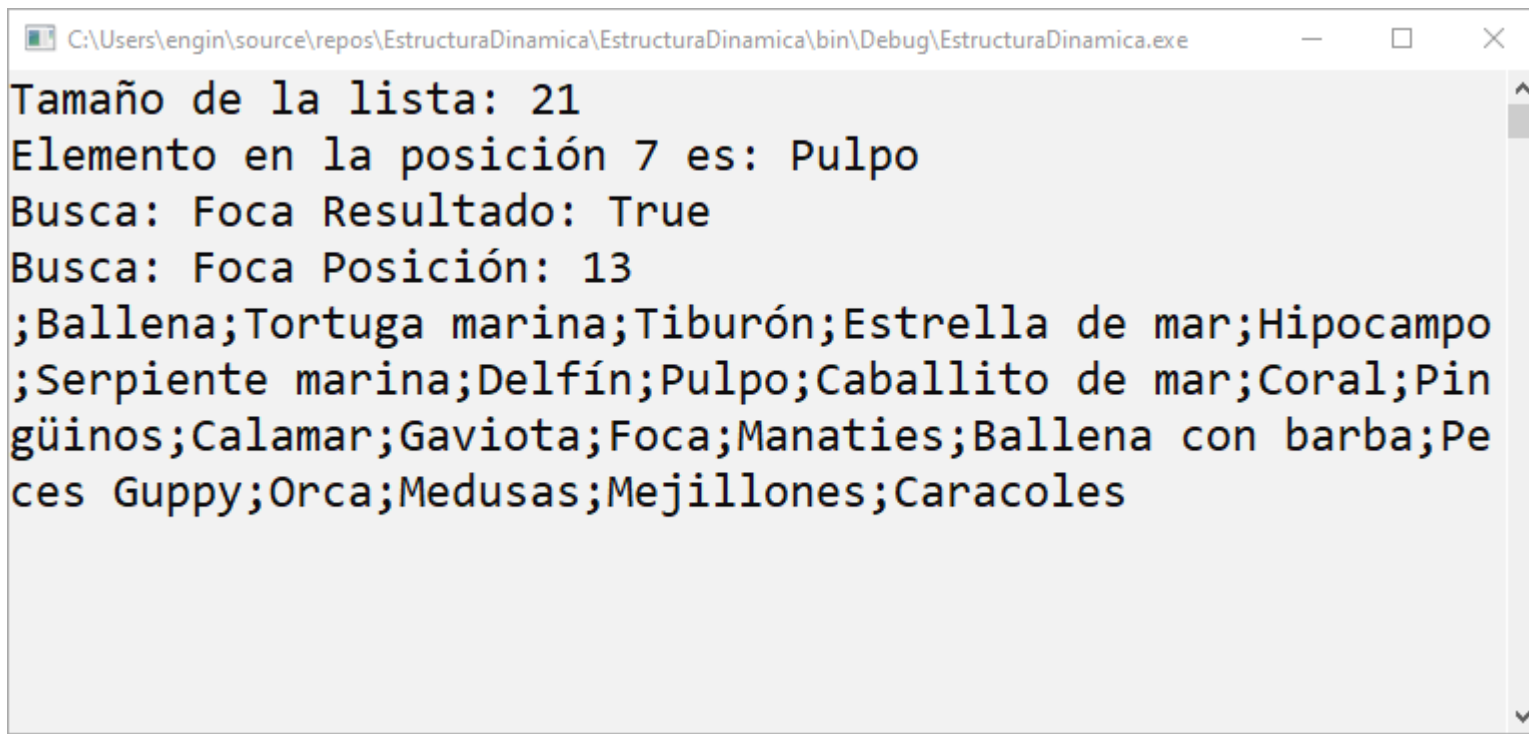
            //Nos dice si existe un determinado elemento en la lista
            string buscar = "Foca";
            bool Existe = ListaAnimales.Contains(buscar);
            Console.WriteLine("Busca: " + buscar + " Resultado: " + Existe);

            //Nos dice la posición donde encontró el elemento en la lista
            int posBusca = ListaAnimales.IndexOf(buscar);
            Console.WriteLine("Busca: " + buscar + " Posición: " + posBusca.ToString());

            //Imprime la lista
            foreach (object elemento in ListaAnimales)
                Console.Write("{0}{1}", ";", elemento);

            Console.ReadKey();
        }
    }
}
```

Los ArrayLists empiezan en cero(0). En el código anterior se muestran las funciones para adicionar al ArrayList, determinar el tamaño (número de elementos que tiene), decir si existe un determinado elemento y mostrar la lista usando foreach.



```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Debug\EstructuraDinamica.exe
Tamaño de la lista: 21
Elemento en la posición 7 es: Pulpo
Busca: Foca Resultado: True
Busca: Foca Posición: 13
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo
;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pin
güinos;Calamar;Gaviota;Foca;Manaties;Ballena con barba;Pe
ces Guppy;Orca;Medusas;Mejillones;Caracoles
```

Ilustración 2: Uso básico del ArrayList



```
using System;
using System.Collections;

namespace Estructuras {
    class Program {
        static void Main() {
            //Declara la lista
            ArrayList ListaAnimales = new ArrayList();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Imprime la lista
            foreach (object elemento in ListaAnimales) Console.Write("{0}{1}", ";", elemento);
            Console.WriteLine(" ");

            //Retira elemento de la lista
            ListaAnimales.Remove("Hipocampo");

            //Imprime de nuevo la lista
            foreach (object elemento in ListaAnimales) Console.Write("{0}{1}", ";", elemento);
            Console.WriteLine(" ");

            //Elimina el objeto de determinada posición.
            ListaAnimales.RemoveAt(5);

            //Imprime de nuevo la lista
            foreach (object elemento in ListaAnimales) Console.Write("{0}{1}", ";", elemento);

            Console.ReadKey();
        }
    }
}
```

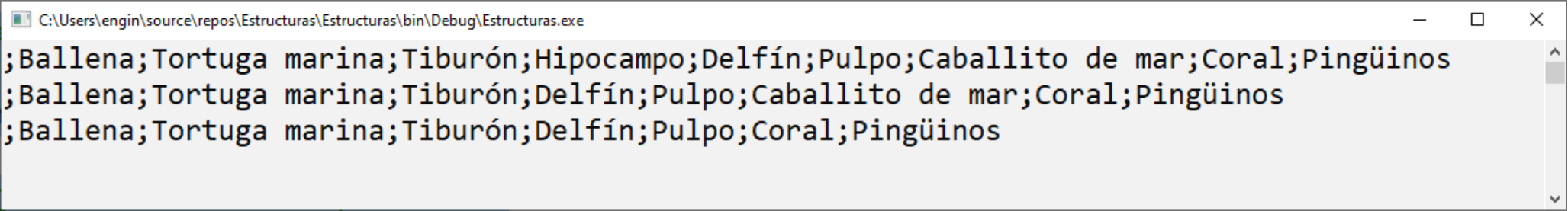


Ilustración 3: Imprime la lista completa, luego borra un elemento según su contenido y según su posición

Dos técnicas para eliminar elementos de un ArrayList, buscando el elemento y eliminándolo, o dada una posición se elimina lo que hay allí.

# Cambiar Elemento

Con la posición de la cadena en la lista, se puede cambiar directamente.

Carpeta 003. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");
            Ejemplo.Add("Calamar");
            Ejemplo.Add("Gaviota");
            Ejemplo.Add("Foca");
            Ejemplo.Add("Manaties");

            //Imprime valores
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Cambia una cadena en la lista
            Ejemplo[3] = "ZZZZZZZZZZ";

            //Imprime valores
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);

            Console.ReadKey();
        }
    }
}
```

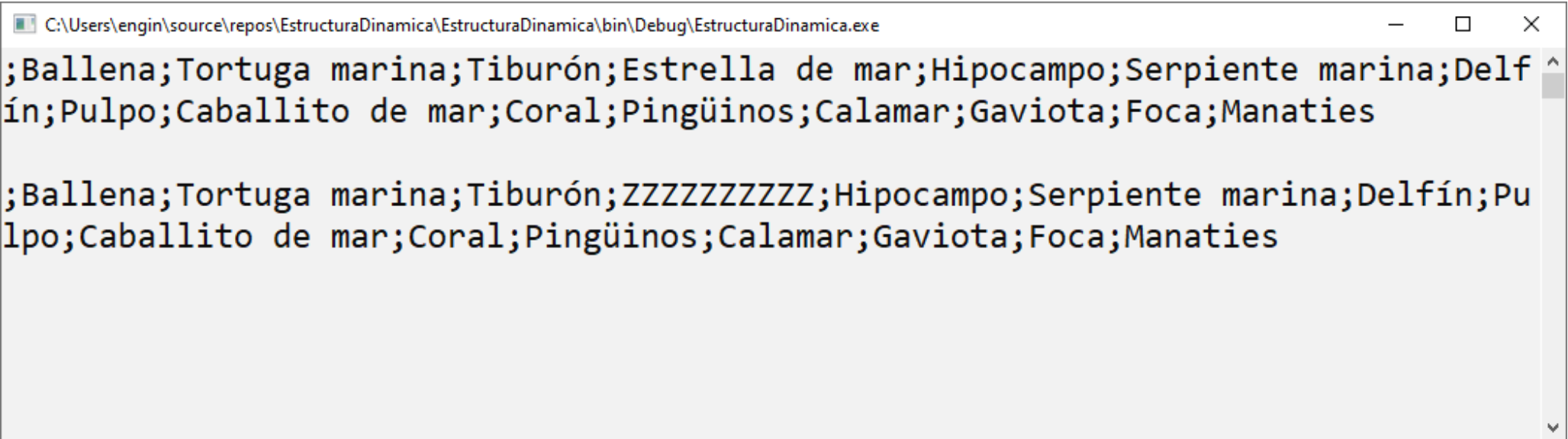


Ilustración 4: Modificar un elemento de la lista. En el ejemplo es el de la posición 3.

# Insertar Elemento

Se puede insertar un elemento en la lista simplemente señalando su posición.

Carpeta 004. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");
            Ejemplo.Add("Calamar");
            Ejemplo.Add("Gaviota");
            Ejemplo.Add("Foca");
            Ejemplo.Add("Manaties");

            //Imprime valores
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Inserta una cadena en la posición 4 de la lista
            Ejemplo.Insert(4, "ZZZZZZZZZZ");

            //Imprime valores
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);

            Console.ReadKey();
        }
    }
}
```

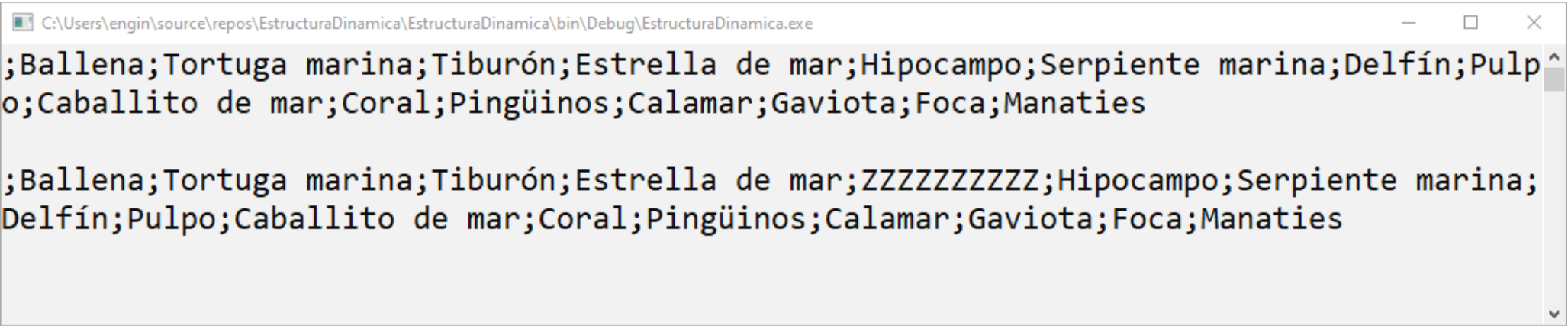


Ilustración 5: Inserta una cadena en la lista

## Referenciar con una variable, un rango de la lista

Dada una lista original, se crea una variable de tipo ArrayList que haga referencia a un rango de esa lista original. ¡OJO! Es una referencia, si se modifica un elemento de la variable tipo ArrayList, ese cambio será en la lista original.

Carpeta 005. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Imprime valores
            Console.WriteLine("Lista original");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Genera nueva lista
            int posicionInicial = 5;
            int cantidad = 3;
            ArrayList nuevaLista = Ejemplo.GetRange(posicionInicial, cantidad);

            //Imprime valores de esa nueva lista
            Console.WriteLine("Nueva lista");
            foreach (Object objeto in nuevaLista) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Modifica un valor de la nueva lista
            nuevaLista[0] = "HHHHHHHHHHHHHHHHHHHHHHHHHHHHHH";

            //Imprime la lista nueva con el valor alterado
            Console.WriteLine("Nueva lista con primer valor alterado");
            foreach (Object objeto in nuevaLista) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Imprime de nuevo la lista original
            Console.WriteLine("Lista original");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);

            Console.ReadKey();
        }
    }
}
```



```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Recorrido con foreach
            Console.WriteLine("Recorrido con foreach");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Recorrido con for
            Console.WriteLine("Recorrido con for");
            for (int cont = 0; cont < Ejemplo.Count; cont++)
                Console.Write("{0}{1}", ";", Ejemplo[cont]);
            Console.WriteLine("\r\n");

            //Recorrido con un IEnumerator
            Console.WriteLine("Recorrido con un IEnumerator");
            IEnumerator elemento = Ejemplo.GetEnumerator();
            while (elemento.MoveNext()) Console.Write("{0}{1}", ";", elemento.Current);

            Console.ReadKey();
        }
    }
}
```

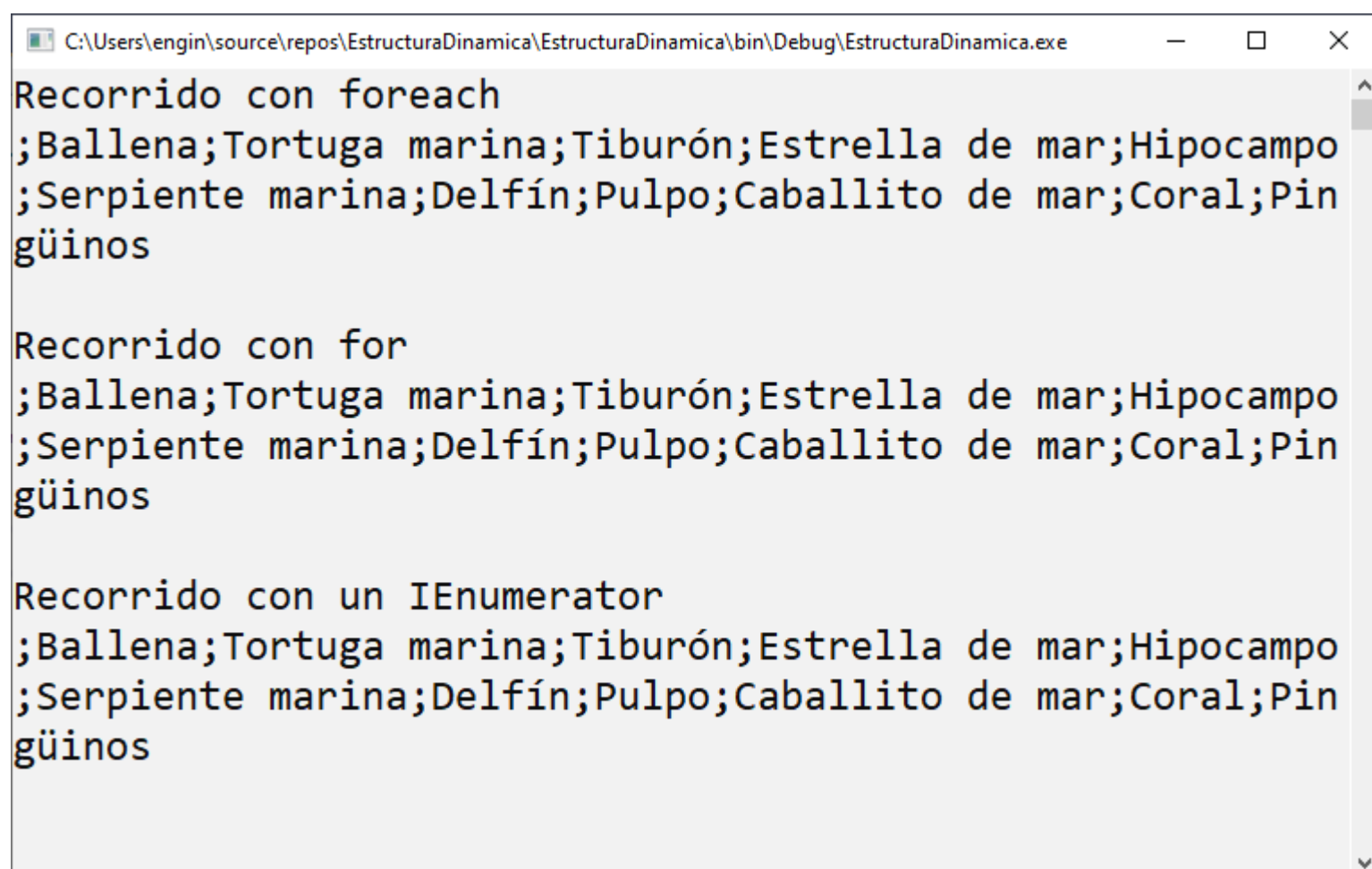


Ilustración 7: Tres formas de recorrer un ArrayList

## Borrar completamente un ArrayList

Se utiliza el método Clear()

Carpeta 007. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Limpia el ArrayList
            Console.WriteLine("(Antes) Total de elementos: " + Ejemplo.Count);
            Ejemplo.Clear();
            Console.WriteLine("(Después) Total de elementos: " + Ejemplo.Count);

            Console.ReadKey();
        }
    }
}
```

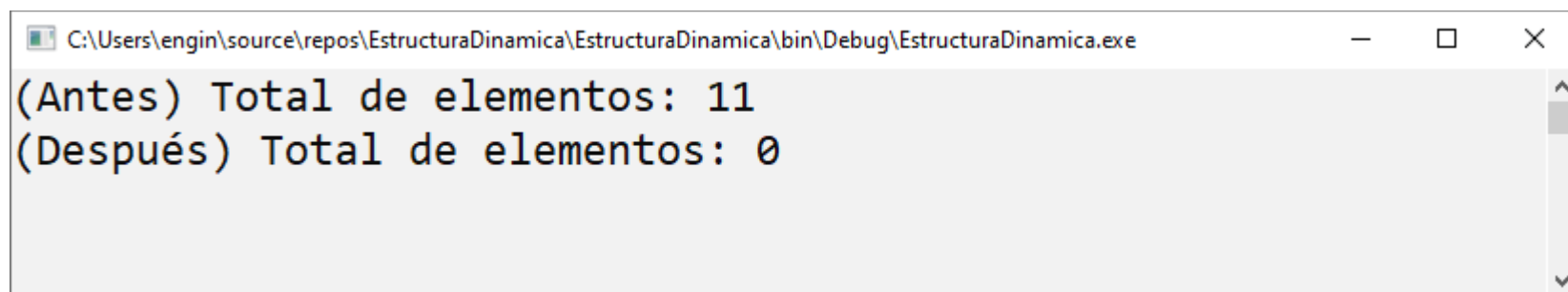


Ilustración 8: Borrar completamente un ArrayList



# Borrar un rango en un ArrayList

Se utiliza el método RemoveRange()

Carpeta 008. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Elimina un rango de elementos del ArrayList
            Console.WriteLine("Antes");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            int posicion = 1;
            int cantidad = 4;
            Ejemplo.RemoveRange(posicion, cantidad);

            Console.WriteLine("Después");
            foreach (Object objeto in Ejemplo) Console.Write("{0}{1}", ";", objeto);

            Console.ReadKey();
        }
    }
}
```

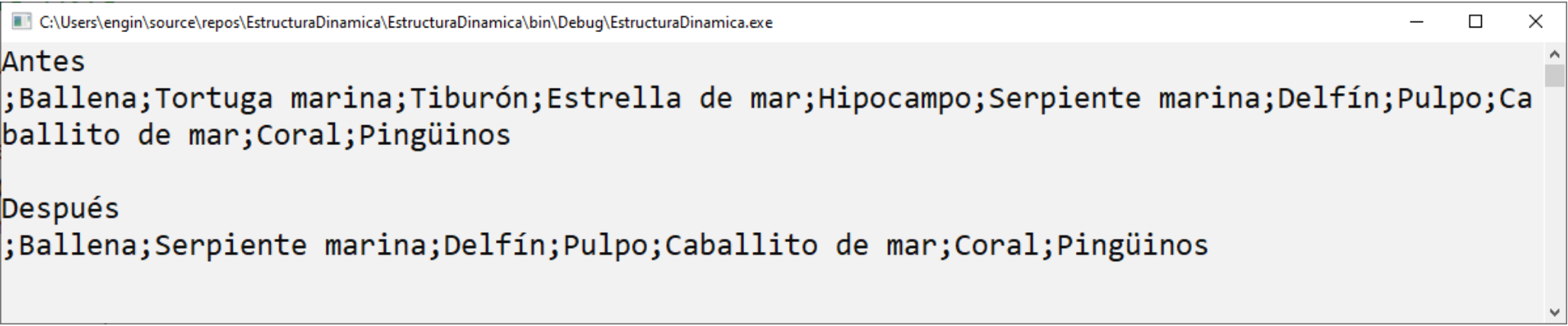


Ilustración 9: Borra un rango de elementos



# Guardar el ArrayList en un arreglo estático

Todos los elementos del ArrayList pasan a un arreglo estático, sea de tipo object() o de algún tipo de dato como string.

Carpeta 009. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Guarda el ArrayList en un arreglo estático de tipo objeto
            Console.WriteLine("Arreglo estático de tipo objeto");
            object[] arregloEstatico = Ejemplo.ToArray();
            foreach (Object objeto in arregloEstatico) Console.Write("{0}{1}", ";", objeto);
            Console.WriteLine("\r\n");

            //Guarda el ArrayList en un arreglo estático de tipo string
            Console.WriteLine("Arreglo estático de tipo cadena");
            string[] cadenas = Ejemplo.ToArray(typeof(string)) as string[];
            for (int cont = 0; cont < cadenas.Length; cont++)
                Console.Write(cadenas[cont] + ";");

            Console.ReadKey();
        }
    }
}
```

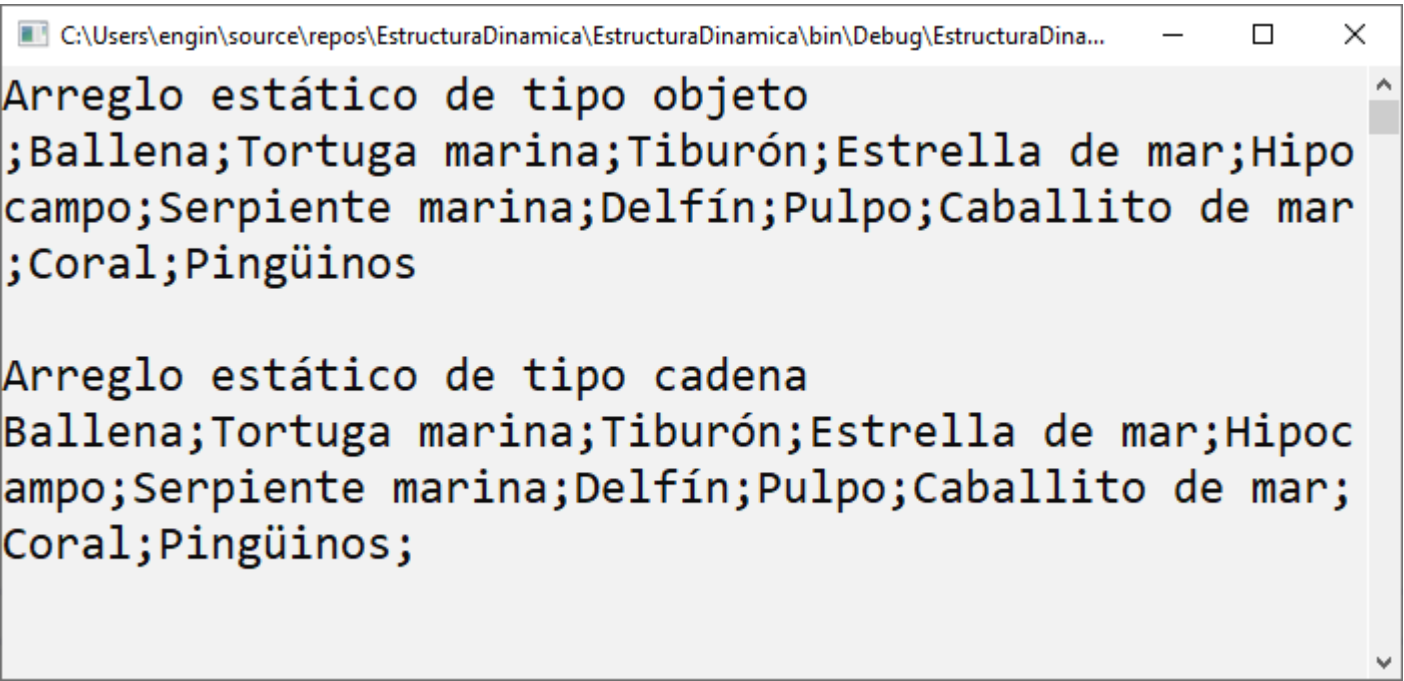


Ilustración 10: De ArrayList a arreglo estático

## Agregar un arreglo estático a un ArrayList

Toma el contenido del arreglo estático y lo copia en el ArrayList usando el método AddRange()

Carpeta 010. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Un arreglo estático
            string[] Cadenas = { "Gato", "Perro", "Conejo", "Liebre", "Oveja" };

            //Adiciona el arreglo estático al ArrayList
            Ejemplo.AddRange(Cadenas);

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");

            Console.ReadKey();
        }
    }
}
```

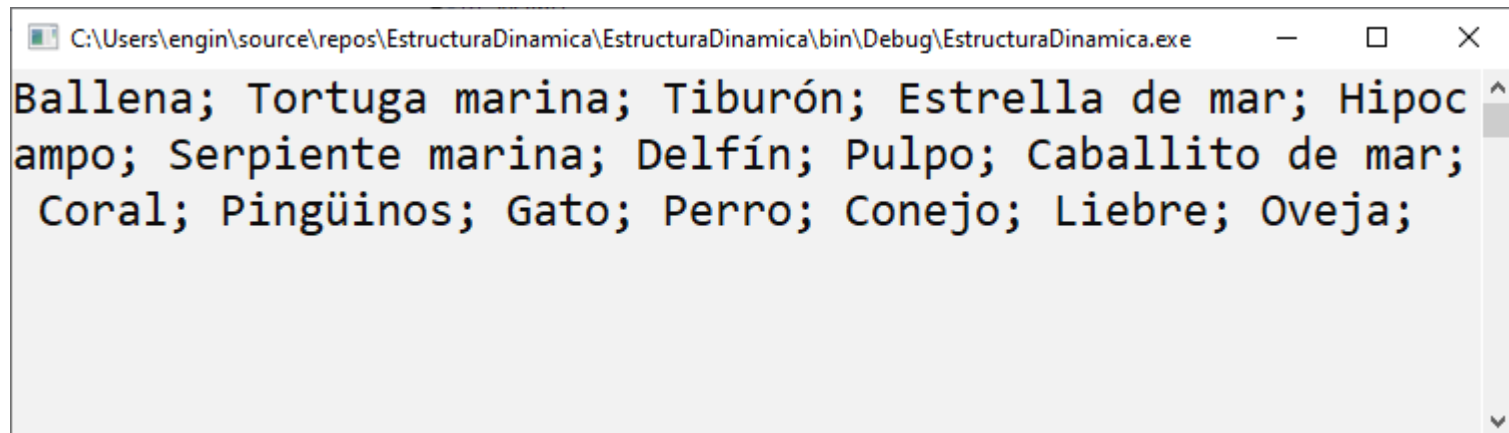


Ilustración 11: Agregar un arreglo estático a un ArrayList

## Inserta un arreglo estático en una determinada posición del ArrayList

Toma el contenido del arreglo estático, lo copia y lo inserta en alguna posición del ArrayList usando el método InsertRange()

Carpeta 011. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("Ballena");
            Ejemplo.Add("Tortuga marina");
            Ejemplo.Add("Tiburón");
            Ejemplo.Add("Estrella de mar");
            Ejemplo.Add("Hipocampo");
            Ejemplo.Add("Serpiente marina");
            Ejemplo.Add("Delfín");
            Ejemplo.Add("Pulpo");
            Ejemplo.Add("Caballito de mar");
            Ejemplo.Add("Coral");
            Ejemplo.Add("Pingüinos");

            //Un arreglo estático
            string[] Cadenas = { "Gato", "Perro", "Conejo", "Liebre", "Oveja" };

            //Inserta el arreglo estático en una determinada posición del ArrayList
            int posicionInserta = 4;
            Ejemplo.InsertRange(posicionInserta, Cadenas);

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");

            Console.ReadKey();
        }
    }
}
```

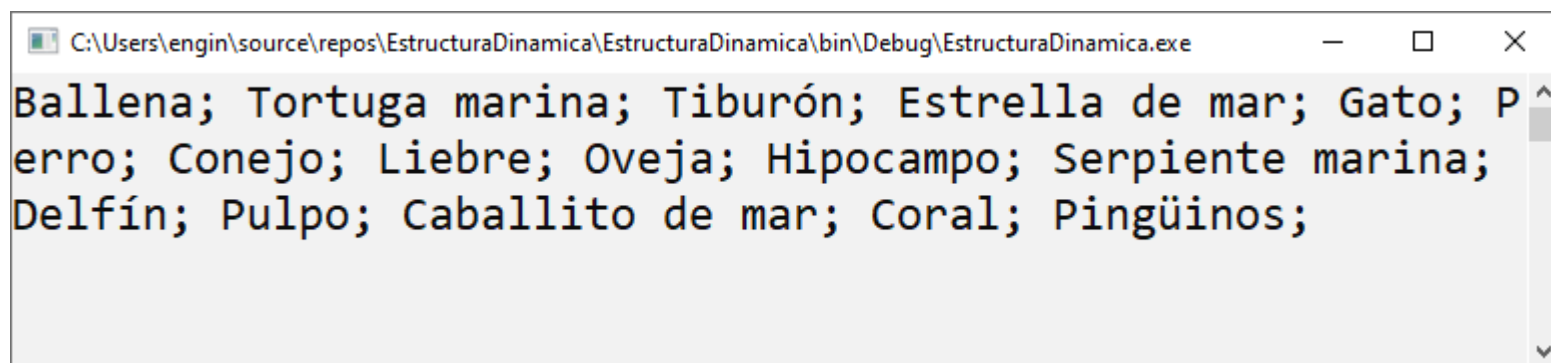


Ilustración 12: Inserta un arreglo estático en un ArrayList con InsertRange

## Insertar varios ArrayList dentro de un ArrayList

El contenido de varios ArrayList es copiado dentro de otro ArrayList. ¡OJO! Es una copia, no importa si se modifica el ArrayList fuente.

Carpeta 012. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara tres ArrayList
            ArrayList ListaA = new ArrayList();
            ArrayList ListaB = new ArrayList();
            ArrayList ListaC = new ArrayList();

            //Adiciona elementos a esos ArrayList
            ListaA.Add("A");
            ListaA.Add("B");
            ListaA.Add("C");
            ListaB.Add("7");
            ListaB.Add("8");
            ListaB.Add("9");
            ListaC.Add("qw");
            ListaC.Add("er");
            ListaC.Add("ty");

            //Inserta los dos primeros ArrayList en el tercero
            int posicionInserta = 1;
            ListaC.InsertRange(posicionInserta, ListaA);

            posicionInserta = 5;
            ListaC.InsertRange(posicionInserta, ListaB);

            //Imprime el ArrayList
            for (int cont = 0; cont < ListaC.Count; cont++) Console.Write(ListaC[cont] + "; ");
            Console.WriteLine("\r\n");

            //Modifica ListaA y se chequea si eso afectó a ListaC
            ListaA[0] = "RRRRRRRRRRRRR";
            for (int cont = 0; cont < ListaC.Count; cont++) Console.Write(ListaC[cont] + "; ");

            Console.ReadKey();
        }
    }
}
```

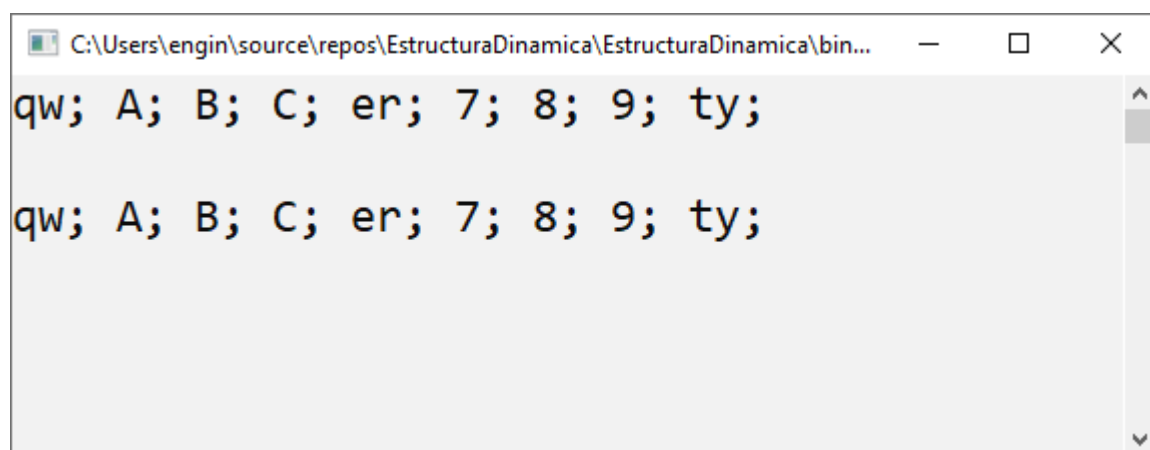


Ilustración 13: Inserta una copia de dos ArrayList en un tercero.

## Invertir un ArrayList

Le da la vuelta al ArrayList

Carpeta 013. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("AB");
            Ejemplo.Add("CD");
            Ejemplo.Add("EF");
            Ejemplo.Add("GH");
            Ejemplo.Add("IJ");
            Ejemplo.Add("KL");
            Ejemplo.Add("MN");
            Ejemplo.Add("OP");

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Aplica Reverse()
            Ejemplo.Reverse();

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");

            Console.ReadKey();
        }
    }
}
```

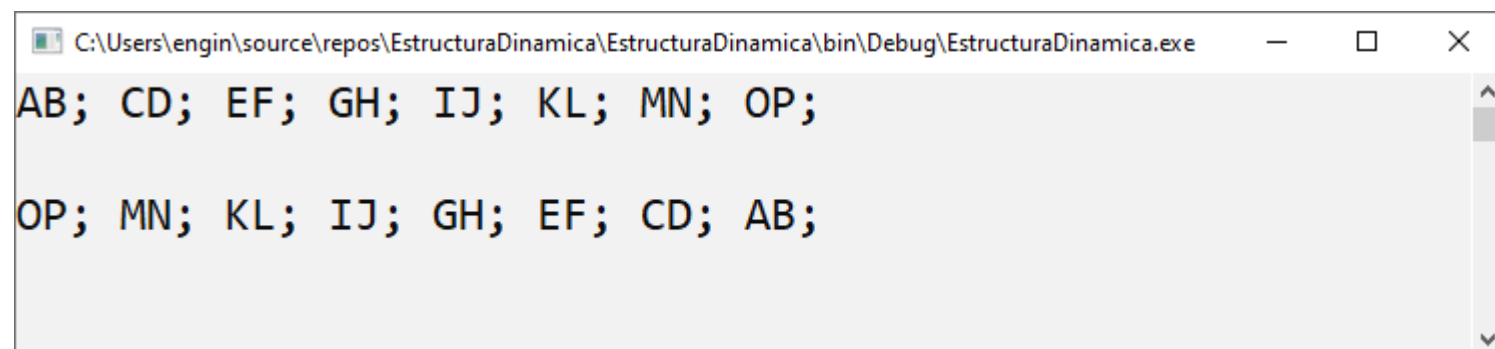


Ilustración 14: Invertir un ArrayList

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("AB");
            Ejemplo.Add("CD");
            Ejemplo.Add("EF");
            Ejemplo.Add("GH");
            Ejemplo.Add("IJ");
            Ejemplo.Add("KL");
            Ejemplo.Add("MN");
            Ejemplo.Add("OP");

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Aplica Reverse(posicion, cantidad)
            int posicion = 2;
            int cantidad = 4;
            Ejemplo.Reverse(posicion, cantidad);

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");

            Console.ReadKey();
        }
    }
}
```

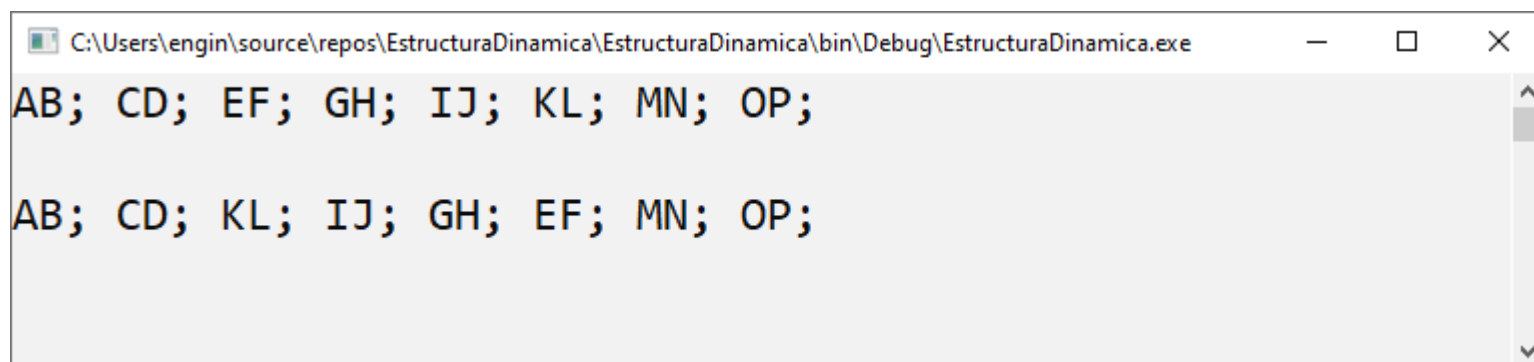


Ilustración 15: Invierte un rango de datos en el ArrayList

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("GH");
            Ejemplo.Add("MN");
            Ejemplo.Add("AB");
            Ejemplo.Add("OP");
            Ejemplo.Add("IJ");
            Ejemplo.Add("KL");
            Ejemplo.Add("CD");
            Ejemplo.Add("EF");

            //Imprime el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Ordena el ArrayList
            Ejemplo.Sort();

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");

            Console.ReadKey();
        }
    }
}
```

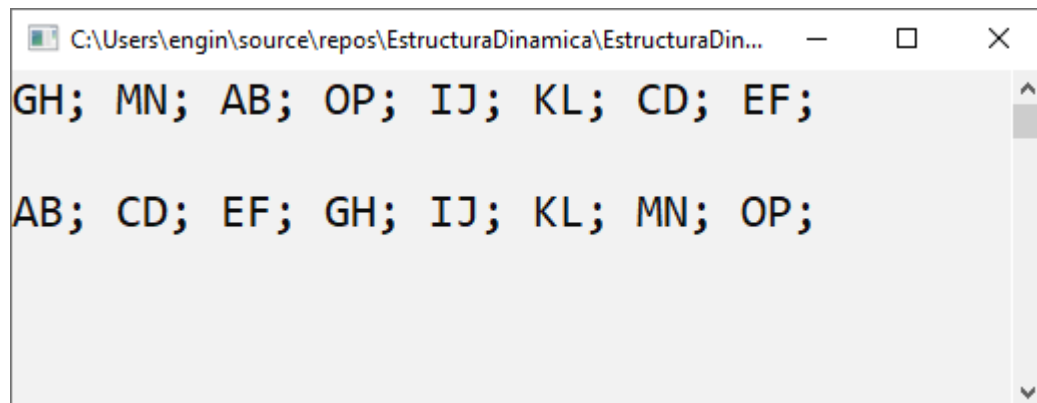


Ilustración 16: Ordena un ArrayList

## Búsqueda Binaria

Una vez el ArrayList es ordenado, se puede hacer una búsqueda binaria.

Carpeta 016. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Adiciona elementos a la lista
            Ejemplo.Add("GH");
            Ejemplo.Add("MN");
            Ejemplo.Add("AB");
            Ejemplo.Add("KL");
            Ejemplo.Add("OP");
            Ejemplo.Add("IJ");
            Ejemplo.Add("CD");
            Ejemplo.Add("EF");

            //Imprime el ArrayList
            Console.WriteLine("ArrayList Original");
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Ordena el ArrayList
            Ejemplo.Sort();

            //Imprime de nuevo el ArrayList
            Console.WriteLine("ArrayList Ordenado");
            for (int cont = 0; cont < Ejemplo.Count; cont++) Console.Write(Ejemplo[cont] + "; ");
            Console.WriteLine("\r\n");

            //Busca en forma binaria en el ArrayList
            string Buscar = "KL";
            int posicion = Ejemplo.BinarySearch(Buscar);
            Console.WriteLine("Buscando a: " + Buscar + " encontrada en: " + posicion.ToString());

            Console.ReadKey();
        }
    }
}
```

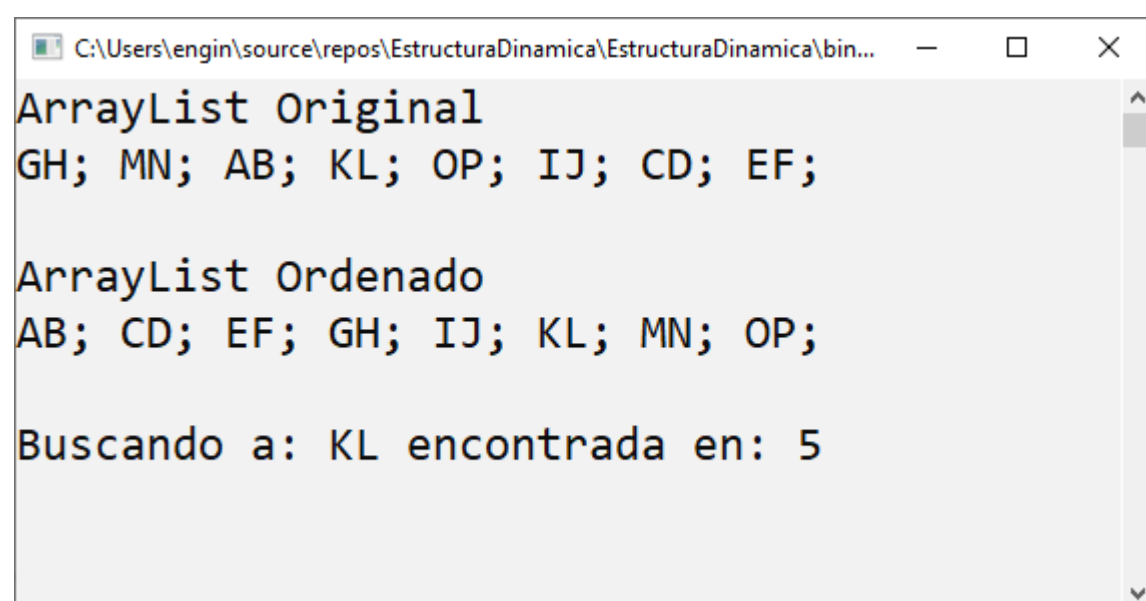


Ilustración 17: Búsqueda binaria de un elemento en un ArrayList previamente ordenado



# Capacidad del ArrayList

A medida que el ArrayList se le van adicionando elementos, su capacidad va aumentando siempre por encima del número de elementos almacenados.

Carpeta 017. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Para agregar elementos al azar
            Random azar = new Random();

            //Va agregando elementos al azar, imprime el tamaño y la capacidad
            for (int veces = 1; veces <= 50; veces++) {
                Console.WriteLine("Iteración: " + veces.ToString());
                Console.WriteLine("Tamaño del ArrayList: " + Ejemplo.Count);
                Console.WriteLine("Capacidad del ArrayList: " + Ejemplo.Capacity + "\r\n");
                for (int cont = 1; cont <= 30; cont++) {
                    Ejemplo.Add(azar.NextDouble());
                }
            }

            Console.ReadKey();
        }
    }
}
```

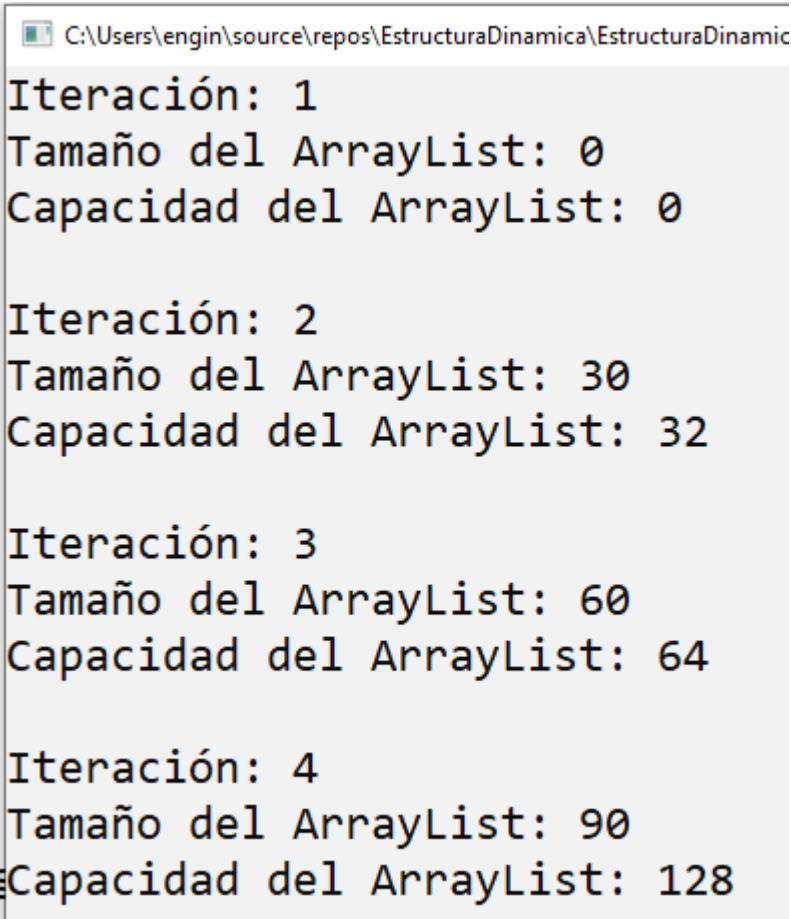


Ilustración 18: Tamaño del ArrayList y la capacidad de ese ArrayList

# Detectar el tipo de dato del elemento del ArrayList

Un ArrayList puede almacenar cualquier tipo de dato, luego si no se está seguro del tipo de dato, se hace uso de GetType( )

Carpeta 018. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Ejemplo = new ArrayList();

            //Agrega diferentes tipos de datos
            Ejemplo.Add("Rafael Alberto Moreno Parra");
            Ejemplo.Add(720626);
            Ejemplo.Add(1.6832929);
            Ejemplo.Add('J');
            Ejemplo.Add(true);

            //Muestra el contenido y el tipo de cada elemento
            for (int cont=0; cont < Ejemplo.Count; cont++) {
                Console.WriteLine(Ejemplo[cont] + " tipo es: " + Ejemplo[cont].GetType());
            }

            Console.WriteLine("\r\n");

            //Y compara
            for (int cont = 0; cont < Ejemplo.Count; cont++) {
                Console.Write(Ejemplo[cont]);
                if (Ejemplo[cont].GetType() == typeof(int)) Console.WriteLine(" es un entero");
                if (Ejemplo[cont].GetType() == typeof(char)) Console.WriteLine(" es un caracter");
                if (Ejemplo[cont].GetType() == typeof(double)) Console.WriteLine(" es un real");
                if (Ejemplo[cont].GetType() == typeof(string)) Console.WriteLine(" es una cadena");
                if (Ejemplo[cont].GetType() == typeof(bool)) Console.WriteLine(" es un booleano");
            }

            Console.ReadKey();
        }
    }
}
```

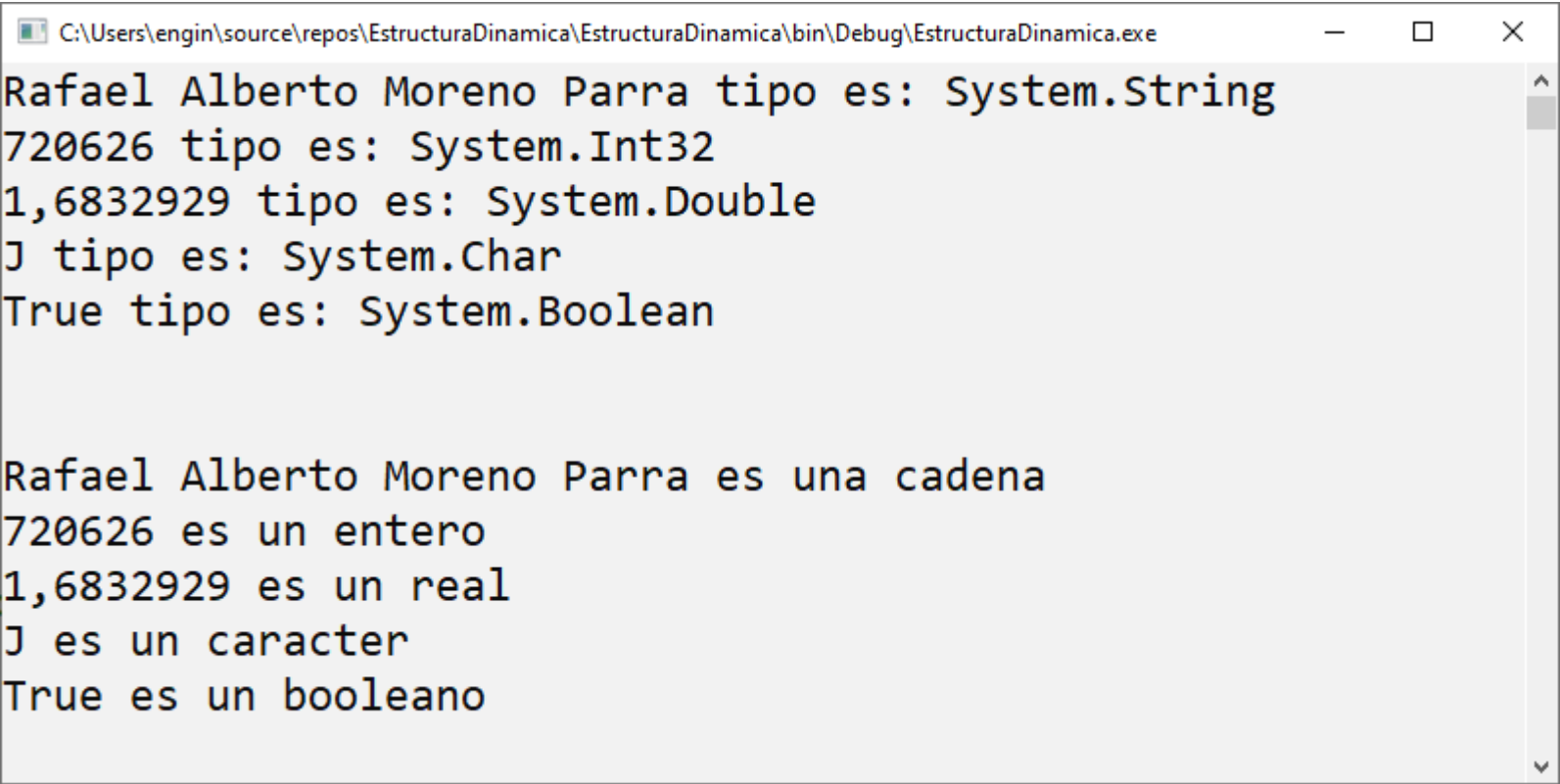


Ilustración 19: Tipos de datos en un ArrayList

# List

List [3] es similar a ArrayList. La diferencia es que List exige un mismo tipo de datos para todos los elementos. Los métodos de List son similares a ArrayList.

La sintaxis para definir un List es:

```
List<tipo de dato> Nombre = new List<tipo de dato>();
```

## Métodos similares a los de ArrayList

Los métodos que se documentaron anteriormente para ArrayList (excepto el de detectar el tipo de dato) se utilizan en List. Se muestra su uso a continuación:

Carpeta 019. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            List<string> ListaAnimales = new List<string>();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");
            ListaAnimales.Add("Calamar");
            ListaAnimales.Add("Gaviota");
            ListaAnimales.Add("Foca");
            ListaAnimales.Add("Manatíes");
            ListaAnimales.Add("Orca");
            ListaAnimales.Add("Medusas");
            ListaAnimales.Add("Mejillones");
            ListaAnimales.Add("Caracoles");

            //Tamaño de la lista
            int tamano = ListaAnimales.Count;
            Console.WriteLine("Tamaño de la lista: " + tamano);

            //Traer un determinado elemento de la lista
            int posicion = 7;
            string texto = ListaAnimales[posicion].ToString();
            Console.WriteLine("Elemento en la posición " + posicion + " es: " + texto);

            //Nos dice si existe un determinado elemento en la lista
            string buscar = "Foca";
            bool Existe = ListaAnimales.Contains(buscar);
            Console.WriteLine("Busca: " + buscar + " Resultado: " + Existe);

            //Nos dice la posición donde encontró el elemento en la lista
            int posBusca = ListaAnimales.IndexOf(buscar);
            Console.WriteLine("Busca: " + buscar + " Posición: " + posBusca.ToString() + "\r\n");

            //Imprime la lista
            Console.WriteLine("Lista Original");
            ImprimeLista(ListaAnimales);

            //Retira elemento de la lista
            Console.WriteLine("Retira HipoCampo");
            ListaAnimales.Remove("Hipocampo");
            ImprimeLista(ListaAnimales);

            //Elimina el objeto de determinada posición.
            Console.WriteLine("Retira Elemento posición 5");
            ListaAnimales.RemoveAt(5);
            ImprimeLista(ListaAnimales);
        }
    }
}
```

```

//Cambia una cadena en la lista
Console.WriteLine("Modifica elemento posición 3");
ListaAnimales[3] = "ZZZZZZZZZZ";
ImprimeLista(ListaAnimales);

//Inserta una cadena en la posición 4 de la lista
Console.WriteLine("Inserta elemento posición 4");
ListaAnimales.Insert(4, "RRRRRRRRRRR");
ImprimeLista(ListaAnimales);

//Genera nueva lista
int posicionInicial = 5;
int cantidad = 3;
List<string> nuevaLista = ListaAnimales.GetRange(posicionInicial, cantidad);
Console.WriteLine("Nueva lista");
ImprimeLista(nuevaLista);

//Recorrido con foreach
Console.WriteLine("Recorrido con foreach");
foreach (Object objeto in ListaAnimales) Console.Write("{0}{1}", ";", objeto);
Console.WriteLine("\r\n");

//Recorrido con for
Console.WriteLine("Recorrido con for");
for (int cont = 0; cont < ListaAnimales.Count; cont++)
    Console.Write("{0}{1}", ";", ListaAnimales[cont]);
Console.WriteLine("\r\n");

//Recorrido con un IEnumerator
Console.WriteLine("Recorrido con un IEnumerator");
IEnumerator Iobjeto = ListaAnimales.GetEnumerator();
while (Iobjeto.MoveNext()) Console.Write("{0}{1}", ";", Iobjeto.Current);
Console.WriteLine("\r\n");

//Guarda el List en un arreglo estático de tipo string
Console.WriteLine("Pasa el List a un arreglo estático de tipo cadena");
string[] cadenas = ListaAnimales.ToArray();
for (int cont = 0; cont < cadenas.Length; cont++) Console.Write(cadenas[cont] + ";");
Console.WriteLine("\r\n");

//Adiciona un arreglo estático al List
Console.WriteLine("Adiciona un arreglo estático al List");
string[] Cadenas = { "Gato", "Perro", "Conejo", "Liebre", "Oveja" };
ListaAnimales.AddRange(Cadenas);
ImprimeLista(ListaAnimales);

//Inserta un arreglo estático al List
Console.WriteLine("Inserta un arreglo estático al List");
string[] Aves = { "Azulejo", "Bichofue", "Paloma", "Gavilán", "Halcón" };
int posicionInserta = 4;
ListaAnimales.InsertRange(posicionInserta, Aves);
ImprimeLista(ListaAnimales);

//Invierte la lista
Console.WriteLine("Invierte la lista");
ListaAnimales.Reverse();
ImprimeLista(ListaAnimales);

//Ordena el List
Console.WriteLine("Ordena la lista");
ListaAnimales.Sort();
ImprimeLista(ListaAnimales);

//Busca en forma binaria en el List
Console.WriteLine("Búsqueda binaria en la lista");
string Buscar = "Gato";
int buscado = ListaAnimales.BinarySearch(Buscar);
Console.WriteLine("Buscando a: " + Buscar + " encontrado en: " + buscado.ToString() +
"\r\n");

//Elimina un rango de elementos del List
Console.WriteLine("Elimina un rango de elementos");
int PosBorra = 1;
int CantidadBorra = 4;
ListaAnimales.RemoveRange(PosBorra, CantidadBorra);
ImprimeLista(ListaAnimales);

//Limpia el List
Console.WriteLine("Borra el List");

```

```

        Console.WriteLine(" (Antes) Total de elementos: " + ListaAnimales.Count);
        ListaAnimales.Clear();
        Console.WriteLine(" (Después) Total de elementos: " + ListaAnimales.Count);

        Console.ReadKey();
    }

    static void ImprimeLista(List<string> listado) {
        for (int cont = 0; cont < listado.Count; cont++) Console.Write("{0}{1}", ";",
listado[cont]);
        Console.WriteLine("\r\n");
    }
}
}

```

```

C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Debug\EstructuraDinamica.exe
Tamaño de la lista: 16
Elemento en la posición 7 es: Pingüinos
Busca: Foca Resultado: True
Busca: Foca Posición: 10

Lista Original
;Ballena;Tortuga;Tiburón;Hipocampo;Delfín;Pulpo;Coral;Pingüinos;
Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Retira HipoCampo
;Ballena;Tortuga;Tiburón;Delfín;Pulpo;Coral;Pingüinos;Calamar;Ga
viota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Retira Elemento posición 5
;Ballena;Tortuga;Tiburón;Delfín;Pulpo;Pingüinos;Calamar;Gaviota;
Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

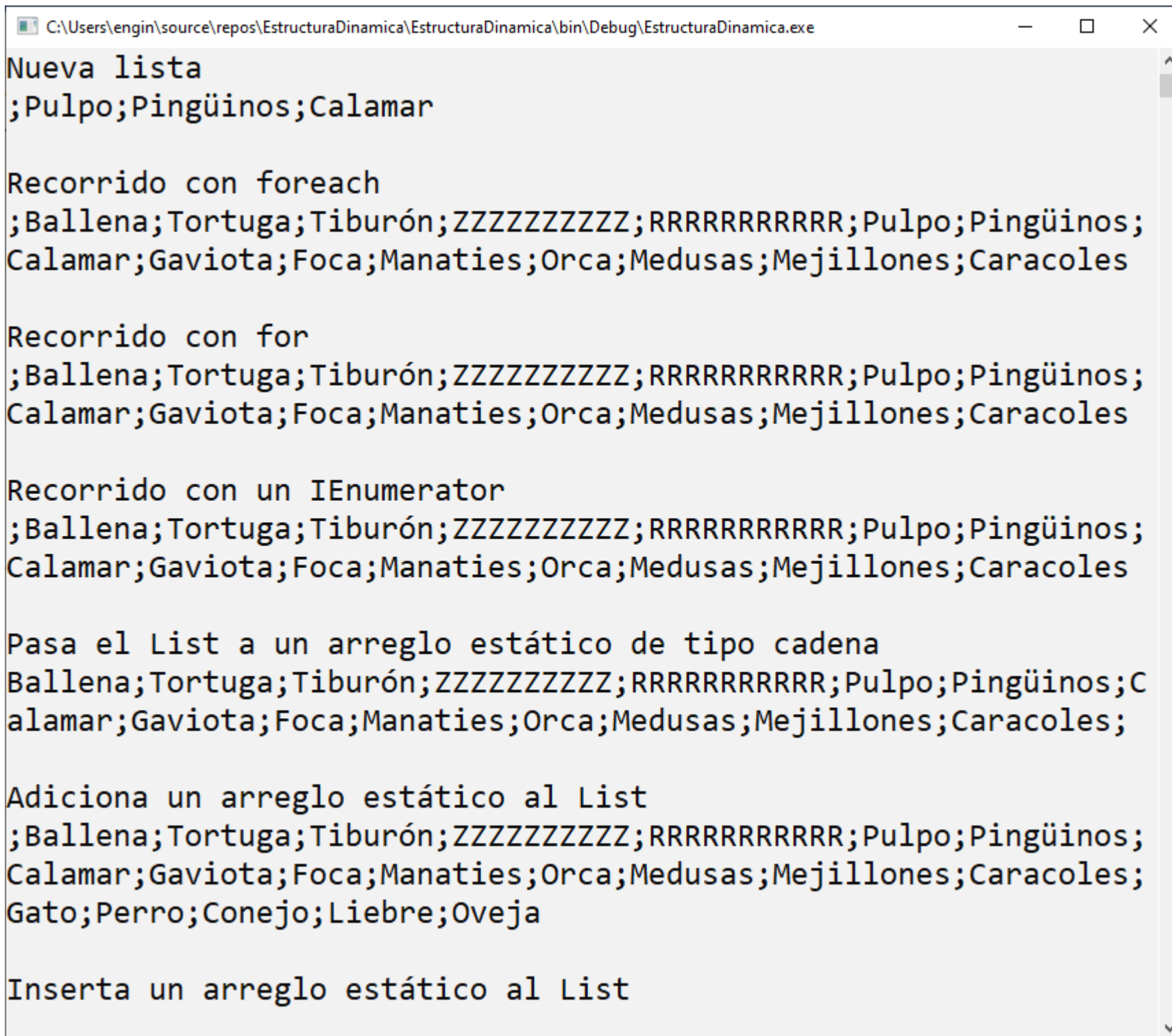
Modifica elemento posición 3
;Ballena;Tortuga;Tiburón;ZZZZZZZZZZ;Pulpo;Pingüinos;Calamar;Gavi
ota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Inserta elemento posición 4
;Ballena;Tortuga;Tiburón;ZZZZZZZZZZ;RRRRRRRRRRRRR;Pulpo;Pingüinos;
Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

```

Ilustración 20: Operaciones en List





```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Debug\EstructuraDinamica.exe

Nueva lista
;Pulpo;Pingüinos;Calamar

Recorrido con foreach
;Ballena;Tortuga;Tiburón;ZZZZZZZZZZ;RRRRRRRRRRR;Pulpo;Pingüinos;
Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Recorrido con for
;Ballena;Tortuga;Tiburón;ZZZZZZZZZZ;RRRRRRRRRRR;Pulpo;Pingüinos;
Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Recorrido con un IEnumerator
;Ballena;Tortuga;Tiburón;ZZZZZZZZZZ;RRRRRRRRRRR;Pulpo;Pingüinos;
Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Pasa el List a un arreglo estático de tipo cadena
Ballena;Tortuga;Tiburón;ZZZZZZZZZZ;RRRRRRRRRRR;Pulpo;Pingüinos;C
alamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles;

Adiciona un arreglo estático al List
;Ballena;Tortuga;Tiburón;ZZZZZZZZZZ;RRRRRRRRRRR;Pulpo;Pingüinos;
Calamar;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles;
Gato;Perro;Conejo;Liebre;Oveja

Inserta un arreglo estático al List
```

Ilustración 21: Operaciones con List

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Debug\EstructuraDinamica.exe
Inserta un arreglo estático al List
;Ballena;Tortuga;Tiburón;ZZZZZZZZZZ;Azulejo;Bichofue;Paloma;Gavi
lán;Halcón;RRRRRRRRRRR;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Mana
ties;Orca;Medusas;Mejillones;Caracoles;Gato;Perro;Conejo;Liebre;
Oveja

Invierte la lista
;Oveja;Liebre;Conejo;Perro;Gato;Caracoles;Mejillones;Medusas;Orc
a;Manaties;Foca;Gaviota;Calamar;Pingüinos;Pulpo;RRRRRRRRRRR;Hal
cón;Gavilán;Paloma;Bichofue;Azulejo;ZZZZZZZZZZ;Tiburón;Tortuga;Ba
llena

Ordena la lista
;Azulejo;Ballena;Bichofue;Calamar;Caracoles;Conejo;Foca;Gato;Gav
ilán;Gaviota;Halcón;Liebre;Manaties;Medusas;Mejillones;Orca;Ovej
a;Paloma;Perro;Pingüinos;Pulpo;RRRRRRRRRRR;Tiburón;Tortuga;ZZZZZ
ZZZZZ

Búsqueda binaria en la lista
Buscando a: Gato encontrado en: 7

Elimina un rango de elementos
;Azulejo;Conejo;Foca;Gato;Gavilán;Gaviota;Halcón;Liebre;Manaties
;Medusas;Mejillones;Orca;Oveja;Paloma;Perro;Pingüinos;Pulpo;RRRR
RRRRRRR;Tiburón;Tortuga;ZZZZZZZZZZ

Borra el List
(Antes) Total de elementos: 21
(Después) Total de elementos: 0
```

Ilustración 22: Operaciones con List

## Comparativa de desempeño de ArrayList vs List vs Arreglo estático

¿Cuál estructura tiene mejor desempeño? Para lograr esta comparativa, se utilizó el algoritmo de ordenación de burbuja, el cuál hace uso intensivo de operaciones de lectura y cambio de valores en la estructura de memoria (por eso es tan lento ese algoritmo), pero será útil para hacer la comparativa. El programa a continuación:

Carpeta 020. Entero.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            /* Prueba de velocidad de los diferentes tipos de estructuras:
             * arreglo estático, ArrayList, List
             * Se usará el método de ordenación de burbuja en el que
             * hace una gran cantidad de lectura y escritura sobre la estructura
             * (por eso es el más lento pero muy bueno para hacer esta comparativa)
             * */

            int minimoOrdena = 1000; //Mínimo número de elementos a ordenar
            int maximoOrdena = 10000; //Máximo número de elementos a ordenar
            int avanceOrdena = 1000; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
             * Luego el tiempo de ordenar N elementos es el promedio de esas pruebas
             * así se evita que por algún motivo los tiempos tengan picos o valles */
            int numPruebas = 10;

            //Limite es el tamaño de datos que se van a ordenar
            for (int Limite = minimoOrdena; Limite <= maximoOrdena; Limite += avanceOrdena)
                Ordenamiento(Limite, numPruebas);

            Console.WriteLine("\r\nFinal de la prueba");
            Console.ReadKey();
        }

        static void Ordenamiento(int Limite, int numPruebas) {
            Random azar = new Random();

            //Las estructuras usadas: arreglo estático, ArrayList, List
            int[] numerosA = new int[Limite];
            int[] numerosB = new int[Limite];
            ArrayList arraylist = new ArrayList();
            List<int> list = new List<int>();

            //Medidor de tiempos
            Stopwatch temporizador = new Stopwatch();

            //Almacena los tiempos de cada método de ordenación
            long TParreglo = 0, TParraylist = 0, TPlist = 0;

            //Para evitar una optimización agresiva del compilador, se acumula el valor de diferentes
            //posiciones del arreglo ordenado.
            long valor = 0;

            //Para disminuir picos o valles en el tiempo, se hacen varias pruebas
            for (int prueba = 1; prueba <= numPruebas; prueba++) {

                //Llena con valores al azar el arreglo
                LlenaAzar(numerosA, azar);

                //Ordenación por Burbuja ArrayList
                arraylist.Clear();
                arraylist.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaArrayList(arraylist);
                TParraylist += temporizador.ElapsedMilliseconds;
                valor += Convert.ToInt32(arraylist[0]);

                //Ordenación por Burbuja List
                list.Clear();
                list.AddRange(numerosA);
```



```

        temporizador.Reset();
        temporizador.Start();
        BurbujaList(list);
        TPlist += temporizador.ElapsedMilliseconds;
        valor += list[1];

        //Ordenación por Burbuja Arreglo estático unidimensional
        Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
        temporizador.Reset();
        temporizador.Start();
        BurbujaArreglo(numerosB);
        TParreglo += temporizador.ElapsedMilliseconds;
        valor += numerosB[2];
    }

    double Tarreglo = (double)TParreglo / numPruebas;
    double Tarraylist = (double)TParraylist / numPruebas;
    double Tlist = (double)TPlist / numPruebas;

    //Console.WriteLine("=====");
    /*Console.WriteLine("\r\nValor de control (contra optimización agresiva): " +
valor.ToString());
    Console.WriteLine("Número de elementos: " + Limite.ToString());
    Console.WriteLine("Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: "
+ Tarreglo.ToString());
    Console.WriteLine("Burbuja ArrayList, tiempo promedio en milisegundos: " +
Tarraylist.ToString());
    Console.WriteLine("Burbuja List, tiempo promedio en milisegundos: " + Tlist.ToString());*/

    //Para estudiarlo en Excel
    string CSV = Limite.ToString() + ";" + Tarreglo.ToString() + ";" + Tarraylist.ToString() + ";"
+ Tlist.ToString() + ";" + valor.ToString();
    Console.WriteLine(CSV);
}

//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(int[] numerosA, Random azar) {
    for (int cont = 0; cont < numerosA.Length; cont++)
        numerosA[cont] = azar.Next(0, 10000);
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (Convert.ToInt32(arraylist[j]) > Convert.ToInt32(arraylist[j + 1])) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

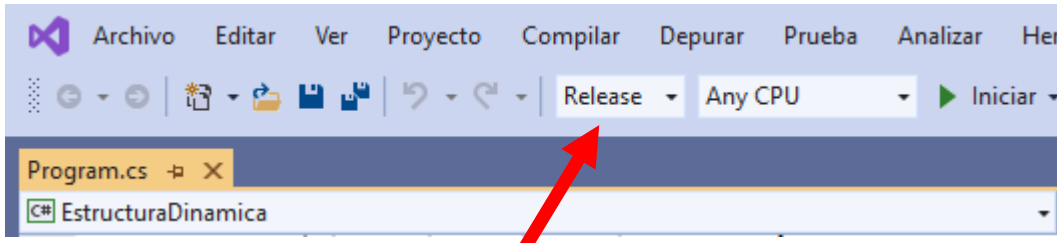
//Ordenamiento por burbuja usando List
static void BurbujaList(List<int> list) {
    int tamano = list.Count;
    int tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (list[j] > list[j + 1]) {
                tmp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando arreglo unidimensional estático
static void BurbujaArreglo(int[] arregloestatico) {
    int tamano = arregloestatico.Length;
    int tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (arregloestatico[j] > arregloestatico[j + 1]) {
                tmp = arregloestatico[j];
                arregloestatico[j] = arregloestatico[j + 1];
                arregloestatico[j + 1] = tmp;
            }
        }
    }
}

```

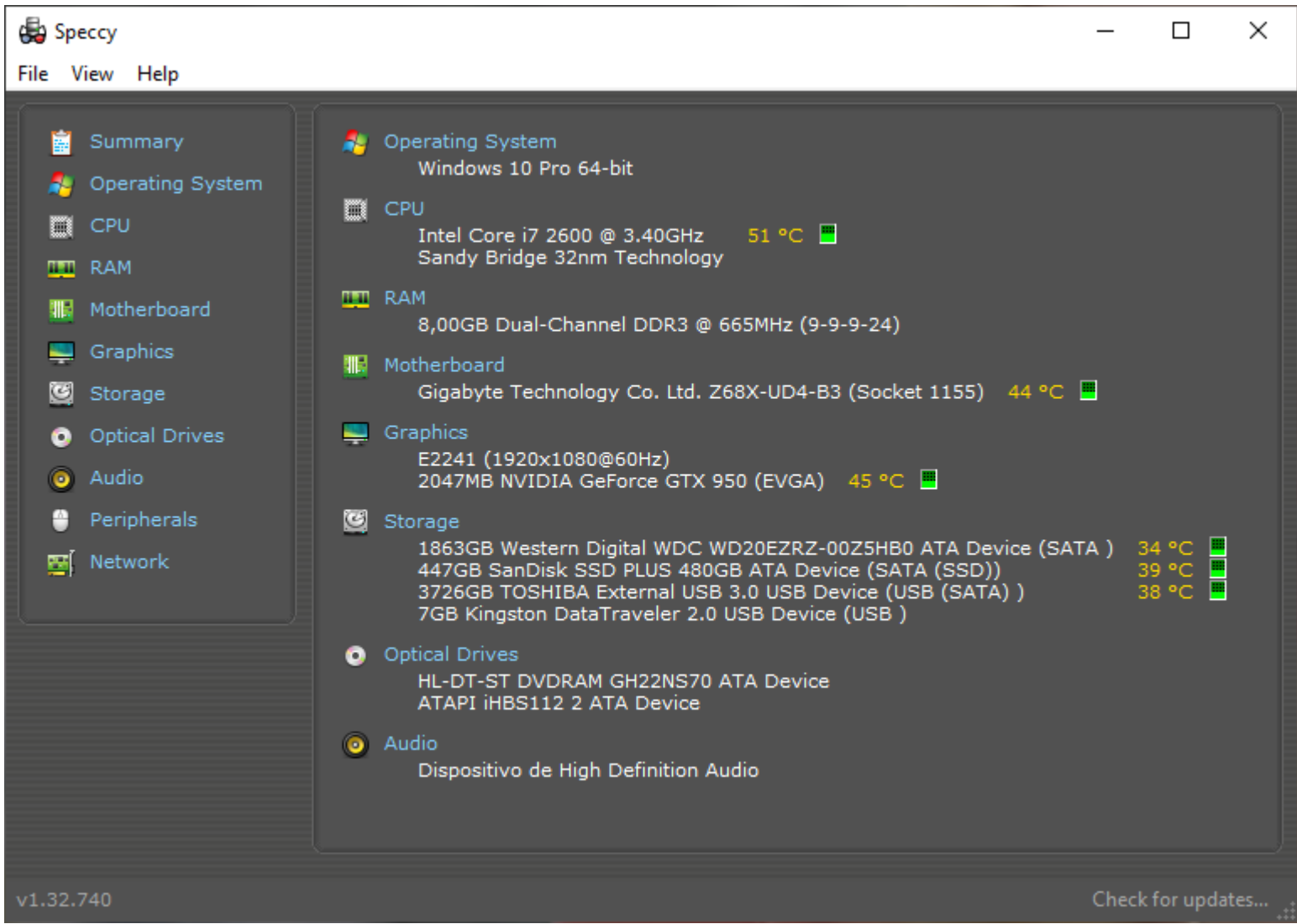
```
}  
}  
}  
}  
}  
arregloestatico[j + 1] = tmp;  
}
```

Se compiló usando el parámetro "Release" en Visual Studio 2019, que elimina código de depuración.



### Ilustración 23: Compilación en modo "Release"

Se ejecutó en el siguiente PC:



*Ilustración 24: Características del PC donde se hacen las pruebas*

Y en el siguiente framework:

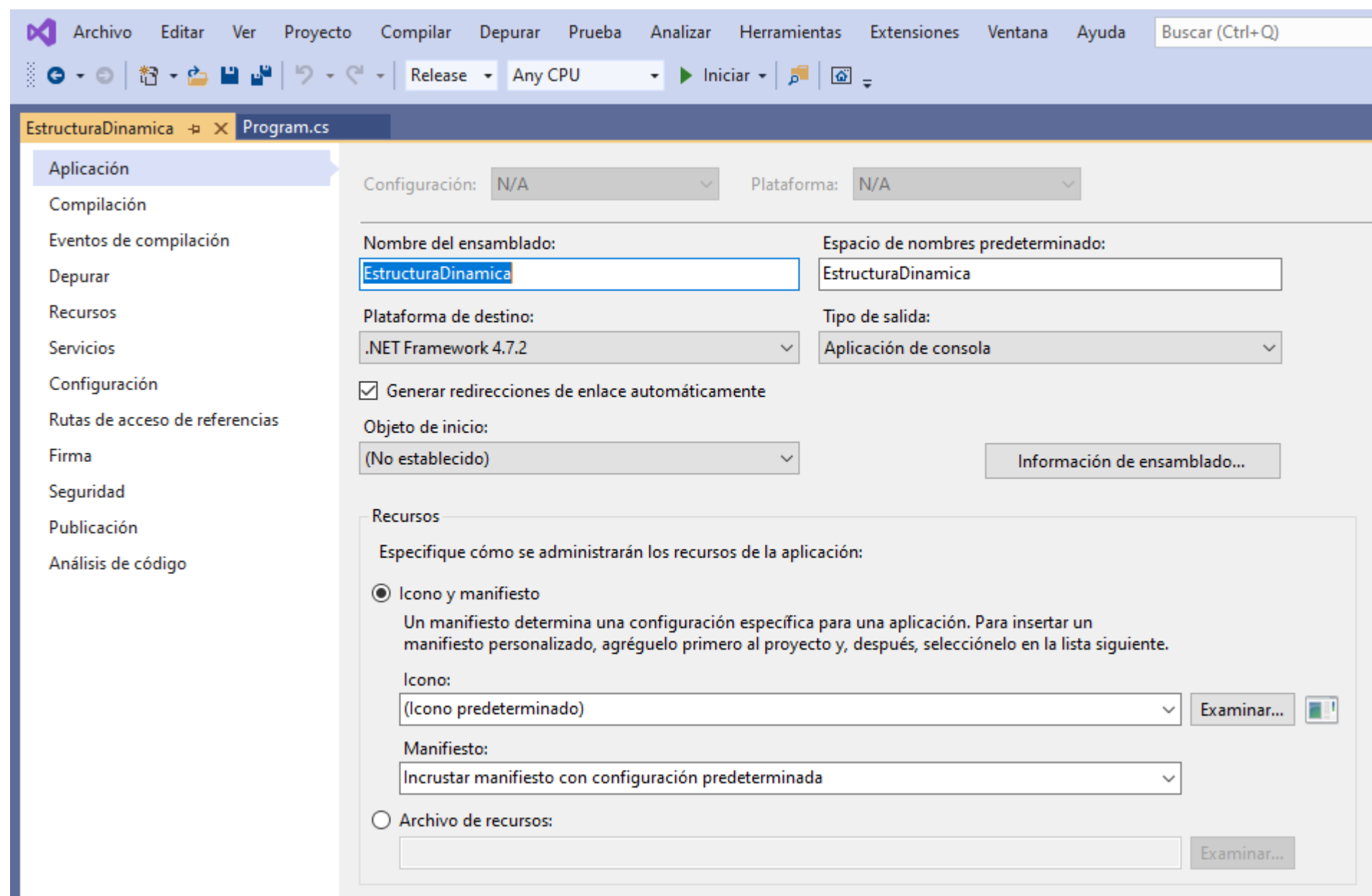


Ilustración 25: Versión del .NET Framework

Para hacer la mejor comparativa en desempeño, las siguientes condiciones se tuvieron en consideración:

1. La misma secuencia de números es ordenada por cada estructura.
2. Sólo se considera el tiempo de ordenamiento por parte del algoritmo de burbuja. No se considera el tiempo de copiar los números en cada estructura.

Y este es el resultado:

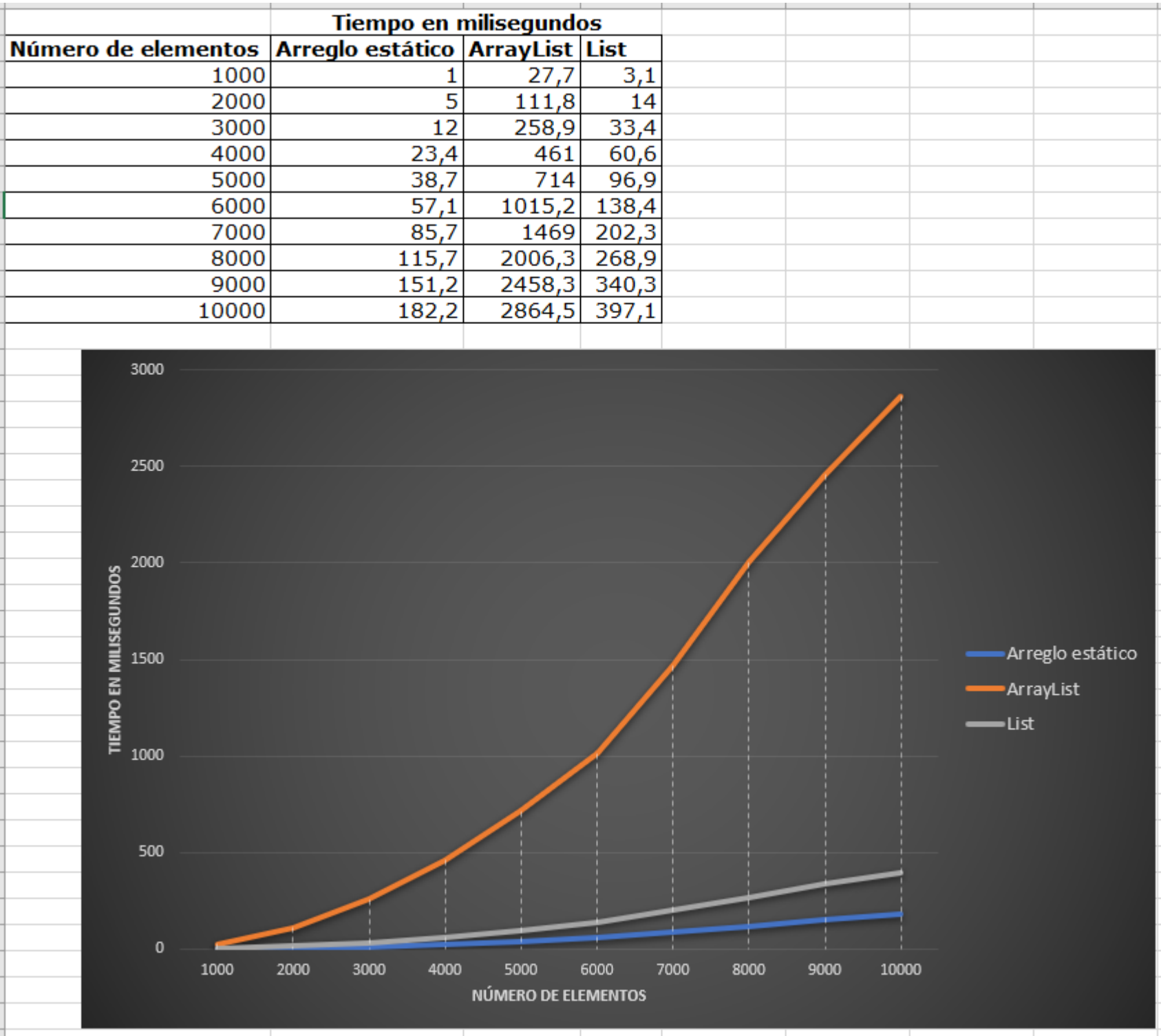


Ilustración 26: Comparativa de desempeño Arreglo estático vs ArrayList vs List ordenando series de datos de tipo entero

Se concluye que el arreglo estático es la estructura más rápida, el List tarda más del doble en hacer las mismas operaciones, y definitivamente el ArrayList es muchísimo más lento. Se nota entonces el precio a pagar por usar estructuras dinámicas (que pueden crecer o decrecer según el número de datos).

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            /* Prueba de velocidad de los diferentes tipos de estructuras:
             * arreglo estático, ArrayList, List
             * Se usará el método de ordenación de burbuja en el que
             * hace una gran cantidad de lectura y escritura sobre la estructura
             * (por eso es el más lento pero muy bueno para hacer esta comparativa)
             * */

            int minimoOrdena = 1000; //Mínimo número de elementos a ordenar
            int maximoOrdena = 10000; //Máximo número de elementos a ordenar
            int avanceOrdena = 1000; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
             * Luego el tiempo de ordenar N elementos es el promedio de esas pruebas
             * así se evita que por algún motivo los tiempos tengan picos o valles */
            int numPruebas = 10;

            //Limite es el tamaño de datos que se van a ordenar
            for (int Limite = minimoOrdena; Limite <= maximoOrdena; Limite += avanceOrdena)
                Ordenamiento(Limite, numPruebas);

            Console.WriteLine("\r\nFinal de la prueba");
            Console.ReadKey();
        }

        static void Ordenamiento(int Limite, int numPruebas) {
            Random azar = new Random();

            //Las estructuras usadas: arreglo estático, ArrayList, List
            double[] numerosA = new double[Limite];
            double[] numerosB = new double[Limite];
            ArrayList arraylist = new ArrayList();
            List<double> list = new List<double>();

            //Medidor de tiempos
            Stopwatch temporizador = new Stopwatch();

            //Almacena los tiempos de cada método de ordenación
            long TParreglo = 0, TParrraylist = 0, TPlist = 0;

            //Para evitar una optimización agresiva del compilador, se acumula el valor de diferentes
            //posiciones del arreglo ordenado.
            double valor = 0;

            //Para disminuir picos o valles en el tiempo, se hacen varias pruebas
            for (int prueba = 1; prueba <= numPruebas; prueba++) {

                //Llena con valores al azar el arreglo
                LlenaAzar(numerosA, azar);

                //Ordenación por Burbuja ArrayList
                arraylist.Clear();
                arraylist.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaArrayList(arraylist);
                TParrraylist += temporizador.ElapsedMilliseconds;
                valor += Convert.ToDouble(arraylist[0]);

                //Ordenación por Burbuja List
                list.Clear();
                list.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaList(list);
                TPlist += temporizador.ElapsedMilliseconds;
                valor += list[1];

                //Ordenación por Burbuja Arreglo estático unidimensional
```

```

        Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
        temporizador.Reset();
        temporizador.Start();
        BurbujaArreglo(numerosB);
        TParreglo += temporizador.ElapsedMilliseconds;
        valor += numerosB[2];
    }

    double Tarreglo = (double)TParreglo / numPruebas;
    double Tarraylist = (double)TParraylist / numPruebas;
    double Tlist = (double)TPList / numPruebas;

    //Console.WriteLine("=====");
    /*Console.WriteLine("\r\nValor de control (contra optimización agresiva): " +
valor.ToString());
    Console.WriteLine("Número de elementos: " + Limite.ToString());
    Console.WriteLine("Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: "
+ Tarreglo.ToString());
    Console.WriteLine("Burbuja ArrayList, tiempo promedio en milisegundos: " +
Tarraylist.ToString());
    Console.WriteLine("Burbuja List, tiempo promedio en milisegundos: " + Tlist.ToString());*/

    //Para estudiarlo en Excel
    string CSV = Limite.ToString() + ";" + Tarreglo.ToString() + ";" + Tarraylist.ToString() + ";"
+ Tlist.ToString() + ";" + valor.ToString();
    Console.WriteLine(CSV);
}

//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(double[] numerosA, Random azar) {
    for (int cont = 0; cont < numerosA.Length; cont++)
        numerosA[cont] = azar.NextDouble();
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (Convert.ToDouble(arraylist[j]) > Convert.ToDouble(arraylist[j + 1])) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando List
static void BurbujaList(List<double> list) {
    int tamano = list.Count;
    double tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (list[j] > list[j + 1]) {
                tmp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando arreglo unidimensional estático
static void BurbujaArreglo(double[] arregloestatico) {
    int tamano = arregloestatico.Length;
    double tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (arregloestatico[j] > arregloestatico[j + 1]) {
                tmp = arregloestatico[j];
                arregloestatico[j] = arregloestatico[j + 1];
                arregloestatico[j + 1] = tmp;
            }
        }
    }
}
}

```

Y este es el resultado:

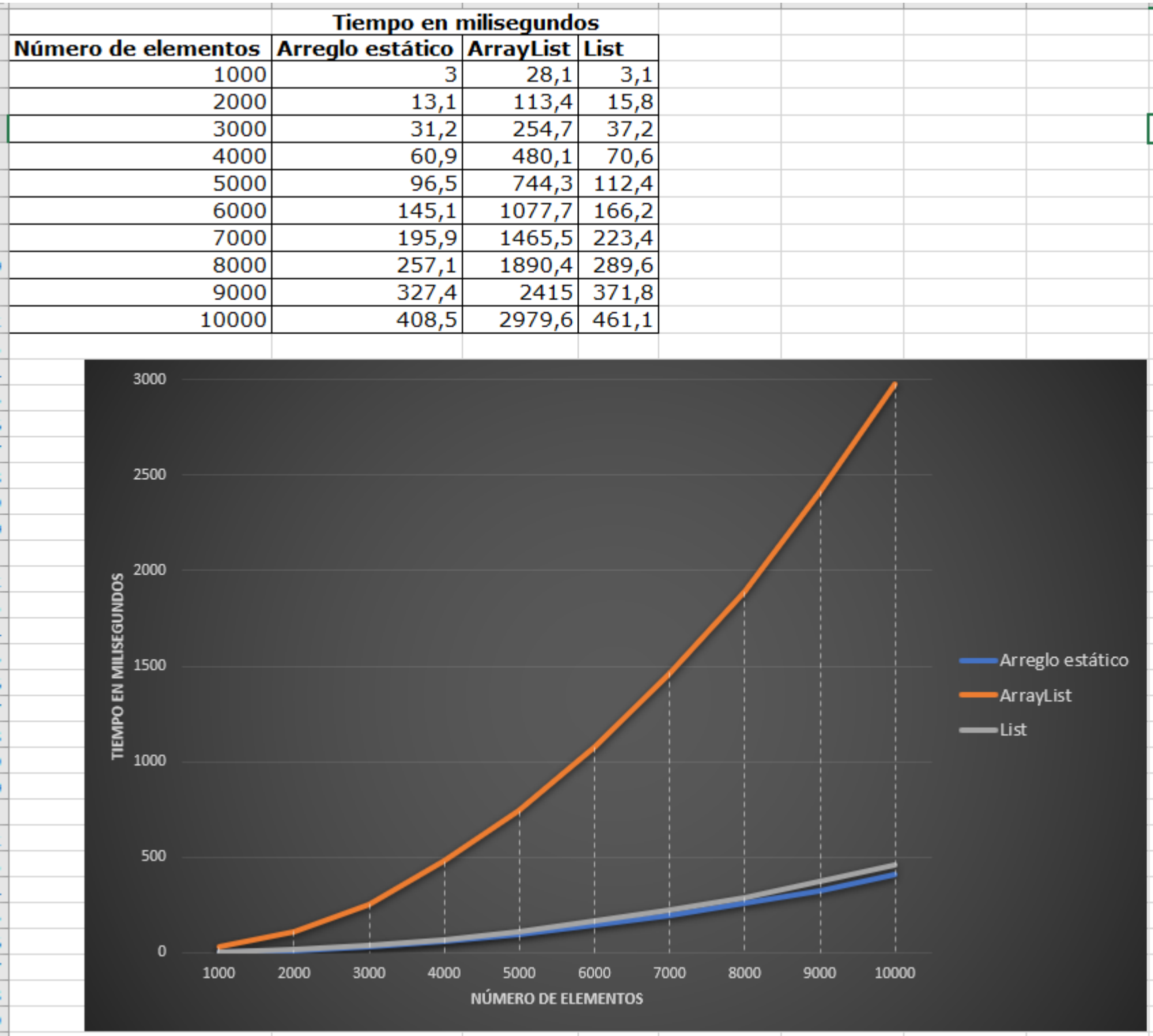


Ilustración 27: Comparativa de desempeño Arreglo estático vs ArrayList vs List ordenando series de datos de tipo double

Se concluye que también con elementos de tipo double el arreglo estático es la estructura más rápida, pero el List casi la alcanza haciendo las mismas operaciones (la diferencia es pequeña), y definitivamente el ArrayList es muchísimo más lento.



```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            /* Prueba de velocidad de los diferentes tipos de estructuras:
             * arreglo estático, ArrayList, List
             * Se usará el método de ordenación de burbuja en el que
             * hace una gran cantidad de lectura y escritura sobre la estructura
             * (por eso es el más lento pero muy bueno para hacer esta comparativa)
             * */

            int minimoOrdena = 1000; //Mínimo número de elementos a ordenar
            int maximoOrdena = 10000; //Máximo número de elementos a ordenar
            int avanceOrdena = 1000; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
             * Luego el tiempo de ordenar N elementos es el promedio de esas pruebas
             * así se evita que por algún motivo los tiempos tengan picos o valles */
            int numPruebas = 10;

            //Limite es el tamaño de datos que se van a ordenar
            for (int Limite = minimoOrdena; Limite <= maximoOrdena; Limite += avanceOrdena)
                Ordenamiento(Limite, numPruebas);

            Console.WriteLine("\r\nFinal de la prueba");
            Console.ReadKey();
        }

        static void Ordenamiento(int Limite, int numPruebas) {
            Random azar = new Random();

            //Las estructuras usadas: arreglo estático, ArrayList, List
            char[] numerosA = new char[Limite];
            char[] numerosB = new char[Limite];
            ArrayList arraylist = new ArrayList();
            List<char> list = new List<char>();

            //Medidor de tiempos
            Stopwatch temporizador = new Stopwatch();

            //Almacena los tiempos de cada método de ordenación
            long TParreglo = 0, TParrraylist = 0, TPlist = 0;

            //Para evitar una optimización agresiva del compilador, se acumula el valor de diferentes
            //posiciones del arreglo ordenado.
            string valor = "";

            //Para disminuir picos o valles en el tiempo, se hacen varias pruebas
            for (int prueba = 1; prueba <= numPruebas; prueba++) {

                //Llena con valores al azar el arreglo
                LlenaAzar(numerosA, azar);

                //Ordenación por Burbuja ArrayList
                arraylist.Clear();
                arraylist.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaArrayList(arraylist);
                TParrraylist += temporizador.ElapsedMilliseconds;
                valor += Convert.ToChar(arraylist[0]);

                //Ordenación por Burbuja List
                list.Clear();
                list.AddRange(numerosA);
                temporizador.Reset();
                temporizador.Start();
                BurbujaList(list);
                TPlist += temporizador.ElapsedMilliseconds;
                valor += list[1];
            }
        }
    }
}
```



```

        //Ordenación por Burbuja Arreglo estático unidimensional
        Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
        temporizador.Reset();
        temporizador.Start();
        BurbujaArreglo(numerosB);
        TParreglo += temporizador.ElapsedMilliseconds;
        valor += numerosB[2];
    }

    double Tarreglo = (double)TParreglo / numPruebas;
    double Tarraylist = (double)TParraylist / numPruebas;
    double Tlist = (double)TPlist / numPruebas;

    //Console.WriteLine("=====");
    /*Console.WriteLine("\r\nValor de control (contra optimización agresiva): " +
valor.ToString());
    Console.WriteLine("Número de elementos: " + Limite.ToString());
    Console.WriteLine("Burbuja arreglo estático unidimensional, tiempo promedio en milisegundos: "
+ Tarreglo.ToString());
    Console.WriteLine("Burbuja ArrayList, tiempo promedio en milisegundos: " +
Tarraylist.ToString());
    Console.WriteLine("Burbuja List, tiempo promedio en milisegundos: " + Tlist.ToString());*/

    //Para estudiarlo en Excel
    string CSV = Limite.ToString() + ";" + Tarreglo.ToString() + ";" + Tarraylist.ToString() + ";"
+ Tlist.ToString() + ";" + valor;
    Console.WriteLine(CSV);
}

//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(char[] numerosA, Random azar) {
    string Permitido = "abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
    for (int cont = 0; cont < numerosA.Length; cont++)
        numerosA[cont] = Permitido[azar.Next(Permitido.Length)];
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (Convert.ToChar(arraylist[j]) > Convert.ToChar(arraylist[j + 1])) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando List
static void BurbujaList(List<char> list) {
    int tamano = list.Count;
    char tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (list[j] > list[j + 1]) {
                tmp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando arreglo unidimensional estático
static void BurbujaArreglo(char[] arregloestatico) {
    int tamano = arregloestatico.Length;
    char tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (arregloestatico[j] > arregloestatico[j + 1]) {
                tmp = arregloestatico[j];
                arregloestatico[j] = arregloestatico[j + 1];
                arregloestatico[j + 1] = tmp;
            }
        }
    }
}
}

```

```
}  
}
```

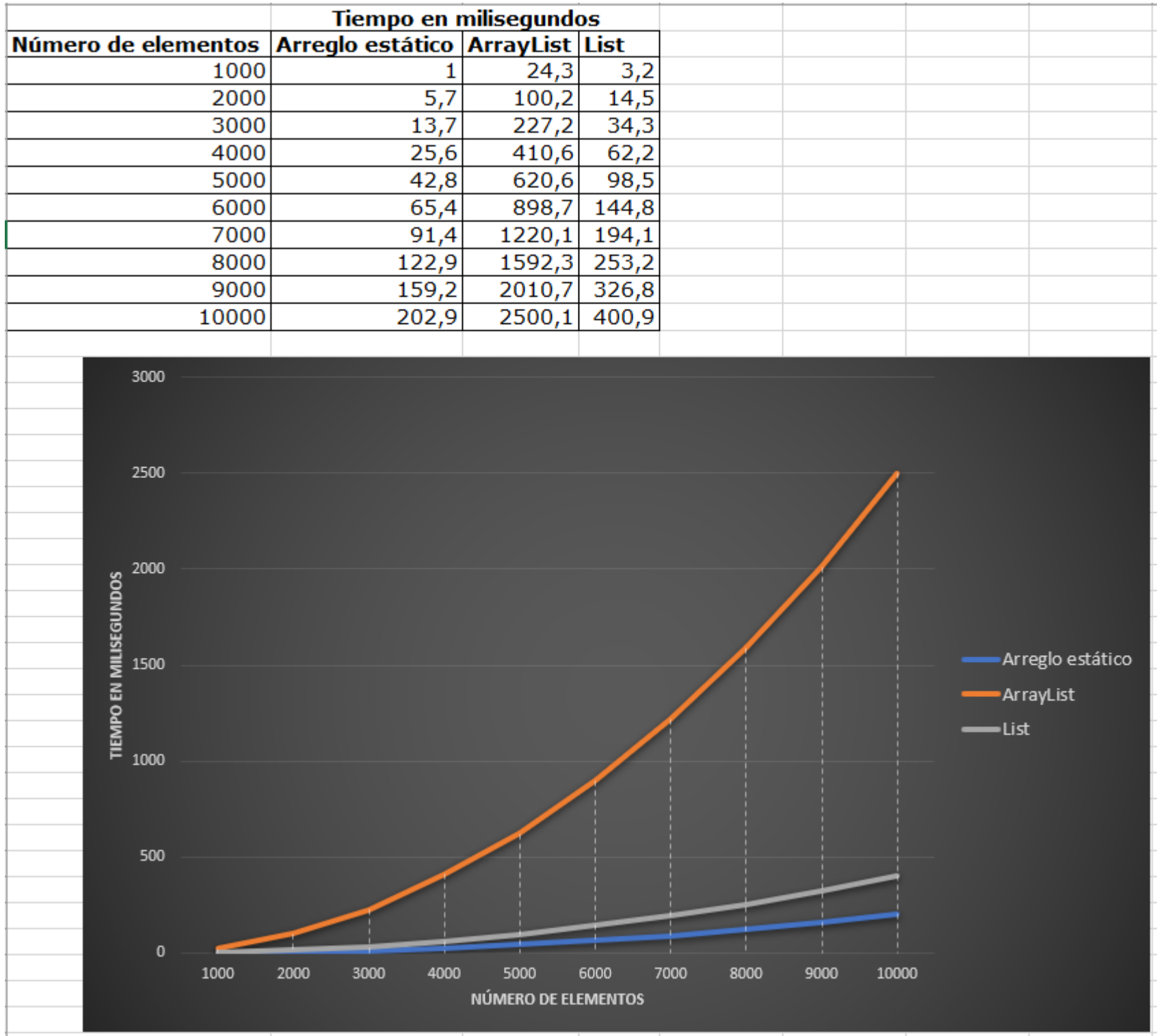


Ilustración 28: Comparativa de desempeño Arreglo estático vs ArrayList vs List ordenando series de datos de tipo char

Se concluye que también con elementos de tipo char el arreglo estático es la estructura más rápida, el List es casi tres veces más de lento, y definitivamente el ArrayList es muchísimo más lento.

Lista de objetos

Un frecuente uso de List es para tener un listado de objetos del mismo tipo, no solo tipos de datos nativo. Se requiere entonces dos clases, en una está definido el tipo de objeto a guardar (Ejemplo.cs) y en la otra se crean, adicionan, actualizan y borran del List (Program.cs)

Carpeta 021. Ejemplo.cs

```
using System;

namespace EstructuraDinamica {
    class Ejemplo {
        //Atributos variados
        public int ValorEntero { get; set; }
        public double NumeroReal { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        //Constructor
        public Ejemplo(int ValorEntero, double NumeroReal, char Caracter, string Cadena) {
            this.ValorEntero = ValorEntero;
            this.NumeroReal = NumeroReal;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }

        //Imprime los valores
        public void Imprime() {
            Console.WriteLine("\r\nEntero: " + ValorEntero.ToString());
            Console.WriteLine("Real: " + NumeroReal.ToString());
            Console.WriteLine("Caracter: " + Caracter.ToString());
            Console.WriteLine("Cadena: [" + Cadena + "]");
        }
    }
}
```

Carpeta 021. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            List<Ejemplo> listado = new List<Ejemplo>();

            //Adiciona objetos a la lista
            listado.Add(new Ejemplo(16, 83.29, 'R', "Rui señor"));
            listado.Add(new Ejemplo(29, 89.7, 'A', "Águila"));
            listado.Add(new Ejemplo(2, 80.19, 'M', "Manatí"));
            listado.Add(new Ejemplo(95, 7.21, 'P', "Puma"));

            //Llama al método de imprimir del objeto
            for (int cont = 0; cont < listado.Count; cont++) listado[cont].Imprime();

            //Inserta un objeto
            listado.Insert(1, new Ejemplo(88, 3.33, 'Z', "ZZZZZZZZ"));

            //Elimina un objeto
            listado.RemoveAt(3);

            //Llama al método de imprimir del objeto
            Console.WriteLine("\r\nDespués de modificar");
            for (int cont = 0; cont < listado.Count; cont++) listado[cont].Imprime();

            Console.WriteLine("\r\nFinal");
            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\l

Entero: 16
Real: 83,29
Caracter: R
Cadena: [Rui señor]

Entero: 29
Real: 89,7
Caracter: A
Cadena: [Águila]

Entero: 2
Real: 80,19
Caracter: M
Cadena: [Manatí]

Entero: 95
Real: 7,21
Caracter: P
Cadena: [Puma]

Después de modificar
```

Ilustración 29: List y objetos

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\F

Después de modificar

Entero: 16
Real: 83,29
Caracter: R
Cadena: [Rui señor]

Entero: 88
Real: 3,33
Caracter: Z
Cadena: [QQQQQQQQ]

Entero: 29
Real: 89,7
Caracter: A
Cadena: [Águila]

Entero: 95
Real: 7,21
Caracter: P
Cadena: [Puma]

Final
```

Ilustración 30: List y objetos

Listas en Listas

Un objeto de una lista, a su vez tiene listas. Un ejemplo con series de TV, personajes y actores:

- 1. La serie tiene un nombre y se puede ver información sobre esta en IMDB.
- 2. Los personajes son interpretados por actores y actrices.
- 3. No es nada extraño que los actores participen en diversas series interpretando algún personaje en alguna serie, sólo es ver su ficha en IMDB, ejemplo: [https://www.imdb.com/name/nm2362068/?ref=tt\\_cl\\_tl](https://www.imdb.com/name/nm2362068/?ref=tt_cl_tl)

Carpeta 022. ActorActriz.cs

```
//Datos del actor o actriz
namespace EstructuraDinamica {
    class ActorActriz {
        public string Nombre { get; set; }
        public string URLIMDB { get; set; }

        //Constructor
        public ActorActriz(string Nombre, string URLIMDB) {
            this.Nombre = Nombre;
            this.URLIMDB = URLIMDB;
        }
    }
}
```

Carpeta 022. Serie.cs

```
//Datos de la serie de televisión
using System.Collections.Generic;

namespace EstructuraDinamica {
    class Serie {
        public string Nombre { get; set; }
        public string URLIMDB { get; set; }

        //Listado de actores y actrices que actúan en la serie
        public List<ActorActriz> Protagonistas = new List<ActorActriz>();

        //Constructor
        public Serie(string Nombre, string URLIMDB) {
            this.Nombre = Nombre;
            this.URLIMDB = URLIMDB;
        }
    }
}
```

Carpeta 022. Persistencia.cs

```
using System.Collections.Generic;

//La parte que simula la capa de persistencia
namespace EstructuraDinamica {
    class Persistencia {
        List<ActorActriz> Actores;
        List<Serie> Series;

        //Carga datos de prueba
        public Persistencia() {
            Actores = new List<ActorActriz>();
            Series = new List<Serie>();

            //Un listado de actores y actrices
            Actores.Add(new ActorActriz("Paulina Andreeva", "https://www.imdb.com/name/nm5475514/"));
            Actores.Add(new ActorActriz("Kirill Käro", "https://www.imdb.com/name/nm1874211/"));
            Actores.Add(new ActorActriz("Aleksandr Ustyugov", "https://www.imdb.com/name/nm1784957/"));
            Actores.Add(new ActorActriz("Emily Berrington", "https://www.imdb.com/name/nm4970834/"));
            Actores.Add(new ActorActriz("Gemma Chan", "https://www.imdb.com/name/nm2110418/"));
            Actores.Add(new ActorActriz("Lucy Carless", "https://www.imdb.com/name/nm6845331/"));

            //Un listado de series
            Series.Add(new Serie("Better Than US", "https://www.imdb.com/title/tt8285216/"));
            Series.Add(new Serie("Humans", "https://www.imdb.com/title/tt4122068/"));
            Series.Add(new Serie("Westworld", "https://www.imdb.com/title/tt0475784/"));
        }
    }
}
```

```

Series.Add(new Serie("Real Humans", "https://www.imdb.com/title/tt2180271/"));
Series.Add(new Serie("Almost Human", "https://www.imdb.com/title/tt2654580/"));
Series.Add(new Serie("Battlestar Galactica", "https://www.imdb.com/title/tt0407362/"));
Series.Add(new Serie("Metod", "https://www.imdb.com/title/tt5135336/"));

//Añado actores y actrices a la serie "Better Than US"
Series[0].Protagonistas.Add(Actores[1]);
Series[0].Protagonistas.Add(Actores[2]);

//Observe que un mismo actor o actriz puede estar en dos series distintas
Series[0].Protagonistas.Add(Actores[0]);
Series[6].Protagonistas.Add(Actores[0]);
}

//Trae datos de la serie
public string SerieNombre(int pos) { return Series[pos].Nombre; }
public string SerieIMDB(int pos) { return Series[pos].URLIMDB; }

//Trae datos del actor
public string ActorNombre(int pos) { return Actores[pos].Nombre; }
public string ActorURL(int pos) { return Actores[pos].URLIMDB; }

//Total de registros
public int ActorTotal() { return Actores.Count; }
public int SerieTotal() { return Series.Count; }

//Adicionar actor
public void ActorAdiciona(string Nombre, string URL) {
    Actores.Add(new ActorActriz(Nombre, URL));
}

//Editar actor
public void ActorEdita(int codigo, string Nombre, string URL) {
    Actores[codigo].Nombre = Nombre;
    Actores[codigo].URLIMDB = URL;
}

//Borrar actor
public void ActorBorra(int codigo) {
    Actores.RemoveAt(codigo);
}

//Retorna una lista de series donde el actor trabaja
public List<string> ActorTrabaja(int codigo) {
    List<string> ListaSeries = new List<string>();
    for (int cont = 0; cont < Series.Count; cont++) {
        for (int num = 0; num < Series[cont].Protagonistas.Count; num++) {
            if (Actores[codigo] == Series[cont].Protagonistas[num])
                ListaSeries.Add(Series[cont].Nombre);
        }
    }
    return ListaSeries;
}

//Adicionar serie
public void SerieAdiciona(string Nombre, string URL) {
    Series.Add(new Serie(Nombre, URL));
}

//Editar serie
public void SerieEdita(int codigo, string Nombre, string URL) {
    Series[codigo].Nombre = Nombre;
    Series[codigo].URLIMDB = URL;
}

//Borrar serie
public void SerieBorra(int codigo) {
    Series.RemoveAt(codigo);
}

//Retornar los actores que trabajan en la serie
public List<string> SerieActores(int codigo) {
    List<ActorActriz> actores = Series[codigo].Protagonistas;
    List<string> Nombres = new List<string>();
    for (int cont = 0; cont < actores.Count; cont++)
        Nombres.Add(actores[cont].Nombre);
    return Nombres;
}

//Añade un actor a una serie

```

```

        public void SerieAsocia(int serie, int actor) {
            Series[serie].Protagonistas.Add(Actores[actor]);
        }

        //Quita un actor de una serie
        public void SerieDisocia(int serie, int actor) {
            Series[serie].Protagonistas.RemoveAt(actor);
        }
    }
}

```

Carpeta 022. Visual.cs

```

using System;
using System.Collections.Generic;

//La parte visual del programa
namespace EstructuraDinamica {
    class Visual {
        public Persistencia Datos;

        //Conecta con la capa de persistencia
        public Visual(Persistencia objDatos) {
            Datos = objDatos;
        }

        //Menú principal
        public void Menu() {
            int opcion;
            do {
                Console.Clear();
                Console.WriteLine("\n===== Software TV Show 1.3 (Diciembre de 2020)
=====");

                Console.WriteLine("1. CRUD de actores y actrices");
                Console.WriteLine("2. CRUD de series");
                Console.WriteLine("3. Salir");
                Console.Write("¿Opción? ");
                opcion = Convert.ToInt32(Console.ReadLine());
                switch (opcion) {
                    case 1: CRUDactores(); break;
                    case 2: CRUDseries(); break;
                }
            } while (opcion != 3);
        }

        //Menú de actores y actrices
        public void CRUDactores() {
            int opcion;
            do {
                Console.Clear();
                Console.WriteLine("\n===== Software TV Show. Actores/Actrices =====");
                for (int cont = 0; cont < Datos.ActorTotal(); cont++) {
                    Console.Write "[" + cont.ToString() + "] ");
                    Console.Write(Datos.ActorNombre(cont));
                    Console.WriteLine(" Nace: " + Datos.ActorURL(cont).ToString());
                }
                Console.WriteLine(" \n1. Adicionar");
                Console.WriteLine("2. Editar");
                Console.WriteLine("3. Borrar");
                Console.WriteLine("4. ¿En cuáles series trabaja?");
                Console.WriteLine("5. Volver a menú principal");
                Console.Write("¿Opción? ");
                opcion = Convert.ToInt32(Console.ReadLine());
                switch (opcion) {
                    case 1: ActorAdiciona(); break;
                    case 2: ActorEdita(); break;
                    case 3: ActorBorra(); break;
                    case 4: ActorTrabaja(); break;
                }
            } while (opcion != 5);
        }

        //Menú de series de TV
        public void CRUDseries() {
            int opcion;
            do {
                Console.Clear();
                Console.WriteLine("\n===== Software TV Show. Series =====");
            }
        }
    }
}

```



```

        for (int cont = 0; cont < Datos.SerieTotal(); cont++) {
            Console.WriteLine("[ " + cont.ToString() + " ] ");
            Console.WriteLine(Datos.SerieNombre(cont));
            Console.WriteLine(" URL: " + Datos.SerieIMDB(cont));
        }
        Console.WriteLine("\n1. Adicionar");
        Console.WriteLine("2. Editar");
        Console.WriteLine("3. Borrar");
        Console.WriteLine("4. Detalles de la serie");
        Console.WriteLine("5. Asociar actor/actriz a serie");
        Console.WriteLine("6. Disociar actor/actriz a serie");
        Console.WriteLine("7. Volver a menú principal");
        Console.Write("¿Opción? ");
        opcion = Convert.ToInt32(Console.ReadLine());
        switch (opcion) {
            case 1: SerieAdiciona(); break;
            case 2: SerieEdita(); break;
            case 3: SerieBorra(); break;
            case 4: SerieDetalle(); break;
            case 5: SerieAsocia(); break;
            case 6: SerieDisocia(); break;
        }
    } while (opcion != 7);
}

//Pantalla para adicionar actores
public void ActorAdiciona() {
    Console.WriteLine("\tAdicionar actor o actriz al listado");
    Console.Write("¿Nombre? ");
    string nombre = Console.ReadLine();
    Console.Write("¿URL de IMDB? ");
    string URL = Console.ReadLine();
    Datos.ActorAdiciona(nombre, URL);
    Console.WriteLine("\nActor/actriz adicionado. Presione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para editar actores
public void ActorEdita() {
    Console.WriteLine("\tEditar actor o actriz");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());
    Console.Write("¿Nombre? ");
    string nombre = Console.ReadLine();
    Console.Write("¿URL de IMDB? ");
    string URL = Console.ReadLine();
    Datos.ActorEdita(codigo, nombre, URL);
    Console.WriteLine("\nActor/actriz editado. Presione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para borrar actores
public void ActorBorra() {
    Console.WriteLine("\tBorrar actor o actriz");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());
    Datos.ActorBorra(codigo);
    Console.WriteLine("\nActor/actriz borrado. Presione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para mostrar en que series trabaja el actor
public void ActorTrabaja() {
    List<string> ListaSeries;
    Console.WriteLine("\tListar las series donde trabaja el actor/actriz");
    Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
    int codigo = Convert.ToInt32(Console.ReadLine());
    ListaSeries = Datos.ActorTrabaja(codigo);
    for (int cont = 0; cont < ListaSeries.Count; cont++) Console.WriteLine(ListaSeries[cont]);
    Console.WriteLine("\nPresione ENTER para continuar");
    Console.ReadKey();
}

//Pantalla para adicionar series
public void SerieAdiciona() {
    Console.WriteLine("\tAdicionar serie al listado");
    Console.Write("¿Nombre? ");
    string nombre = Console.ReadLine();
    Console.Write("¿URL en IMDB? ");
    string url = Console.ReadLine();

```



```

        Datos.SerieAdiciona(nombre, url);
        Console.WriteLine("\nSerie adicionada. Presione ENTER para continuar");
        Console.ReadKey();
    }

    //Pantalla para editar series
    public void SerieEdita() {
        Console.WriteLine("\tEditar serie");
        Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());
        Console.Write("¿Nombre? ");
        string nombre = Console.ReadLine();
        Console.Write("¿URL en IMDB? ");
        string url = Console.ReadLine();
        Datos.SerieEdita(codigo, nombre, url);
        Console.WriteLine("\nSerie editada. Presione ENTER para continuar");
        Console.ReadKey();
    }

    //Pantalla para borrar series
    public void SerieBorra() {
        Console.WriteLine("\tBorrar serie");
        Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());
        Datos.SerieBorra(codigo);
        Console.WriteLine("\nSerie borrada. Presione ENTER para continuar");
        Console.ReadKey();
    }

    //Pantalla para ver el detalle de la serie
    public void SerieDetalle() {
        List<string> ListaActores;

        Console.WriteLine("\t === Detalle de una serie ===");
        Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Nombre: " + Datos.SerieNombre(codigo));
        Console.WriteLine("URL: " + Datos.SerieIMDB(codigo));
        Console.WriteLine("Actores");
        ListaActores = Datos.SerieActores(codigo);
        for (int cont = 0; cont < ListaActores.Count; cont++) Console.WriteLine("\t" +
ListaActores[cont]);

        Console.WriteLine("\nPresione ENTER para continuar");
        Console.ReadKey();
    }

    //Asociar actor o actriz a una serie
    public void SerieAsocia() {
        Console.WriteLine("\tAsocia un actor o actriz a una serie");
        Console.Write("¿Cuál serie? Escriba el número que está entre [ ]: ");
        int serie = Convert.ToInt32(Console.ReadLine());
        for (int cont = 0; cont < Datos.ActorTotal(); cont++) {
            Console.Write "[" + cont.ToString() + "] ";
            Console.Write(Datos.ActorNombre(cont));
            Console.WriteLine(" URL: " + Datos.ActorURL(cont).ToString());
        }
        Console.Write("¿Cuál actor/actriz? Escriba el número que está entre [ ]: ");
        int actor = Convert.ToInt32(Console.ReadLine());
        Datos.SerieAsocia(serie, actor);
        Console.WriteLine("\nActor/actriz asociado a la serie. Presione ENTER para continuar");
        Console.ReadKey();
    }

    //Pantalla para disociar actor de alguna serie
    public void SerieDisocia() {
        List<string> ListaActores;

        Console.WriteLine("\t === Disociar actor de la serie ===");
        Console.Write("¿Cuál serie? Escriba el número que está entre [ ]: ");
        int serie = Convert.ToInt32(Console.ReadLine());

        ListaActores = Datos.SerieActores(serie);
        for (int cont = 0; cont < ListaActores.Count; cont++)
            Console.WriteLine "[" + cont.ToString() + "] " + ListaActores[cont]);

        Console.Write("¿Cuál actor/actriz quiere quitar? Escriba el número que está entre [ ]: ");
        int actor = Convert.ToInt32(Console.ReadLine());
    }

```

```

        Datos.SerieDisocia(serie, actor);
        Console.WriteLine("\nActor/actriz retirado de la serie. Presione ENTER para continuar");
        Console.ReadKey();
    }
}
}

```

Carpeta 022. Program.cs

```

//Inicia el programa
namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se debe llamar primero la capa de persistencia (carga datos de ejemplo)
            Persistencia objDatos = new Persistencia();

            //Luego se llama la capa visual
            Visual objVisual = new Visual(objDatos);
            objVisual.Menu();
        }
    }
}

```

¿Cómo ejecuta?

Este es el menú principal un CRUD (Create, Read, Update, Delete) de dos listas: actores/actrices y series de televisión

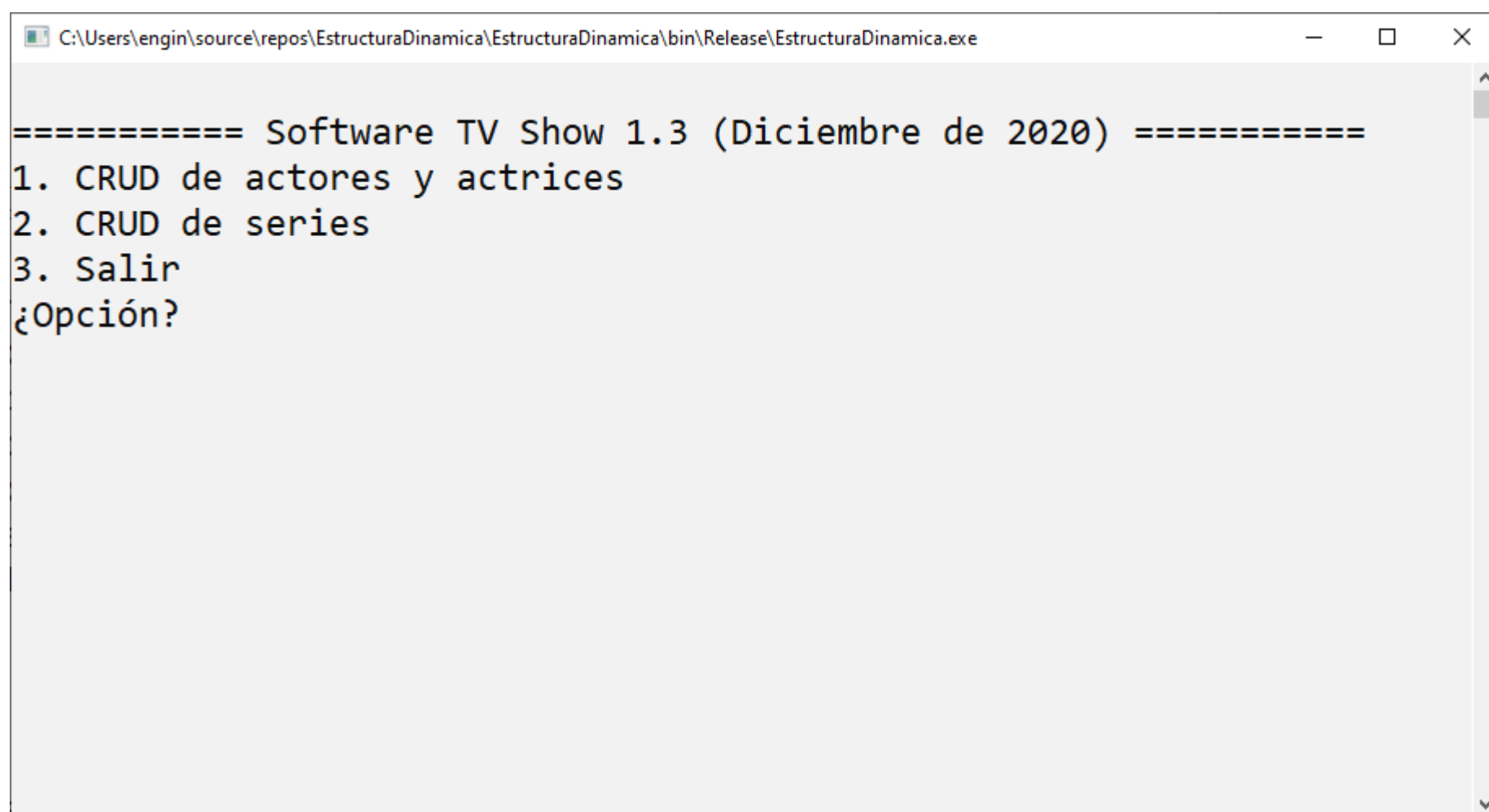


Ilustración 31: Menú inicial

Muestra los actores y actrices almacenados, las opciones CRUD y la opción de ver en que serie de televisión trabaja un actor/actriz seleccionado.

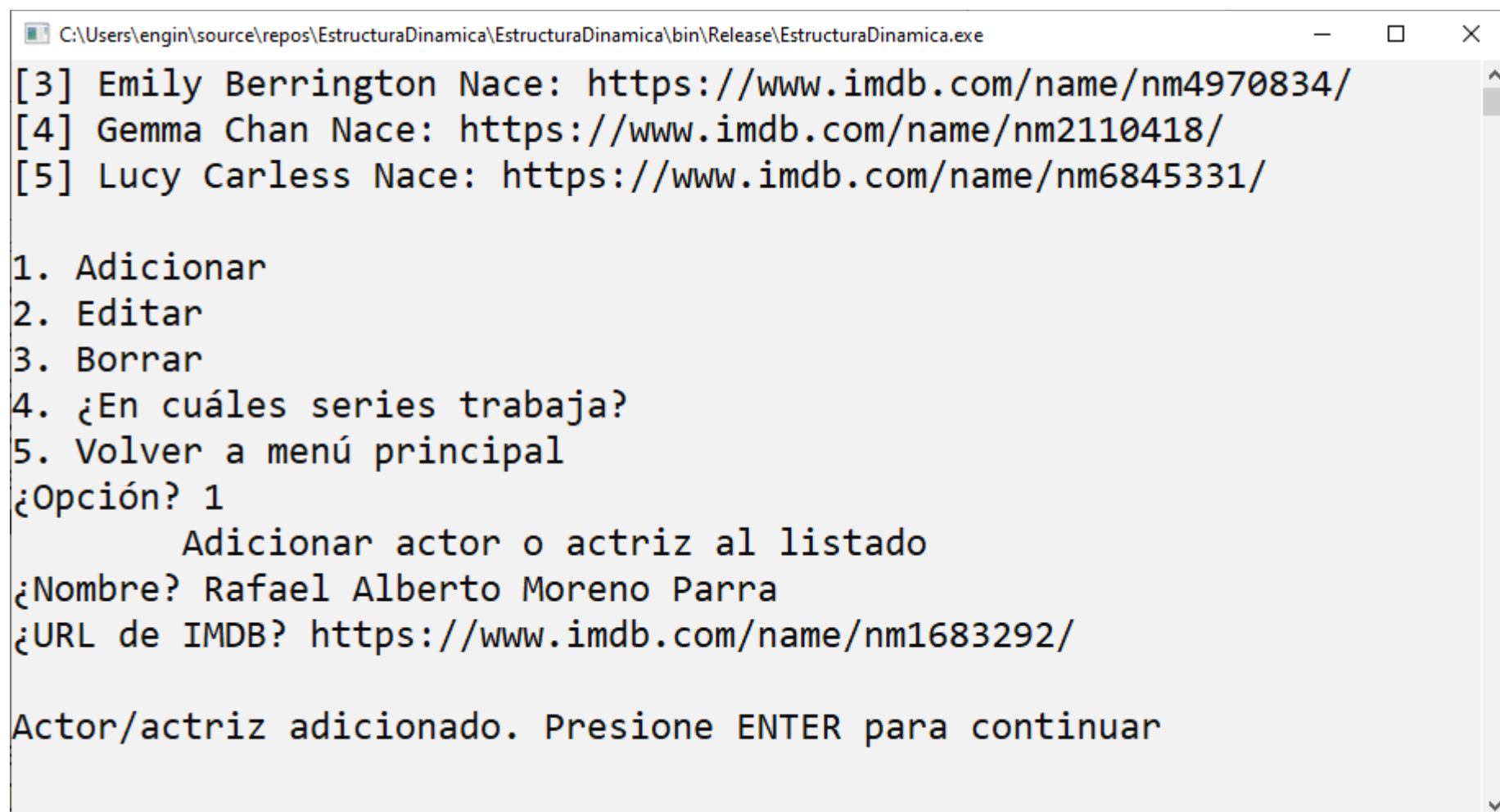


```
==== Software TV Show. Actores/Actrices =====
[0] Paulina Andreeva Nace: https://www.imdb.com/name/nm5475514/
[1] Kirill Käro Nace: https://www.imdb.com/name/nm1874211/
[2] Aleksandr Ustyugov Nace: https://www.imdb.com/name/nm1784957/
[3] Emily Berrington Nace: https://www.imdb.com/name/nm4970834/
[4] Gemma Chan Nace: https://www.imdb.com/name/nm2110418/
[5] Lucy Carless Nace: https://www.imdb.com/name/nm6845331/

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal
¿Opción?
```

Ilustración 32: Menú de actores y actrices

#### Adición de actores

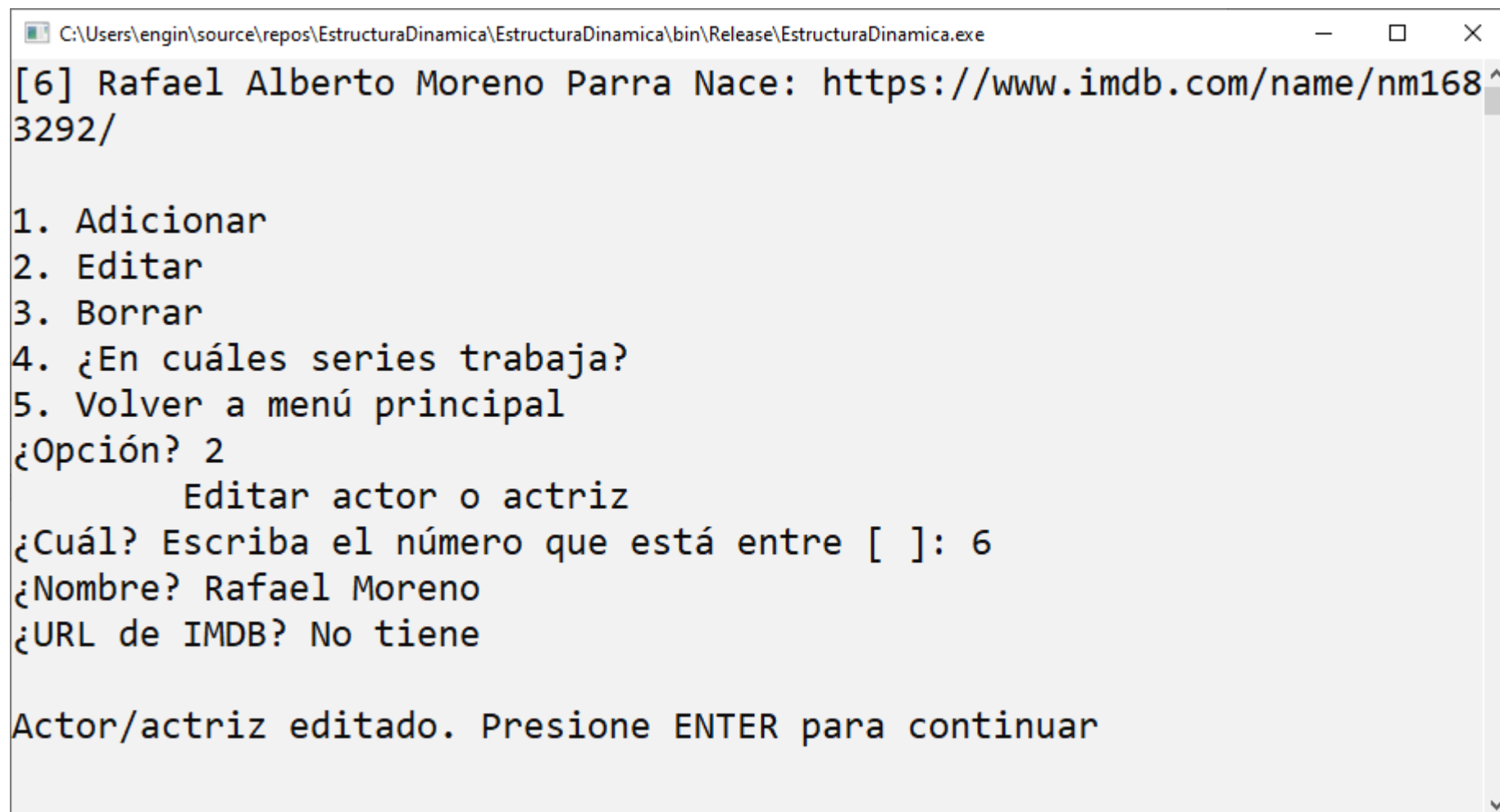


```
[3] Emily Berrington Nace: https://www.imdb.com/name/nm4970834/
[4] Gemma Chan Nace: https://www.imdb.com/name/nm2110418/
[5] Lucy Carless Nace: https://www.imdb.com/name/nm6845331/

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal
¿Opción? 1
    Adicionar actor o actriz al listado
¿Nombre? Rafael Alberto Moreno Parra
¿URL de IMDB? https://www.imdb.com/name/nm1683292/

Actor/actriz adicionado. Presione ENTER para continuar
```

Ilustración 33: Adiciona un actor o actriz



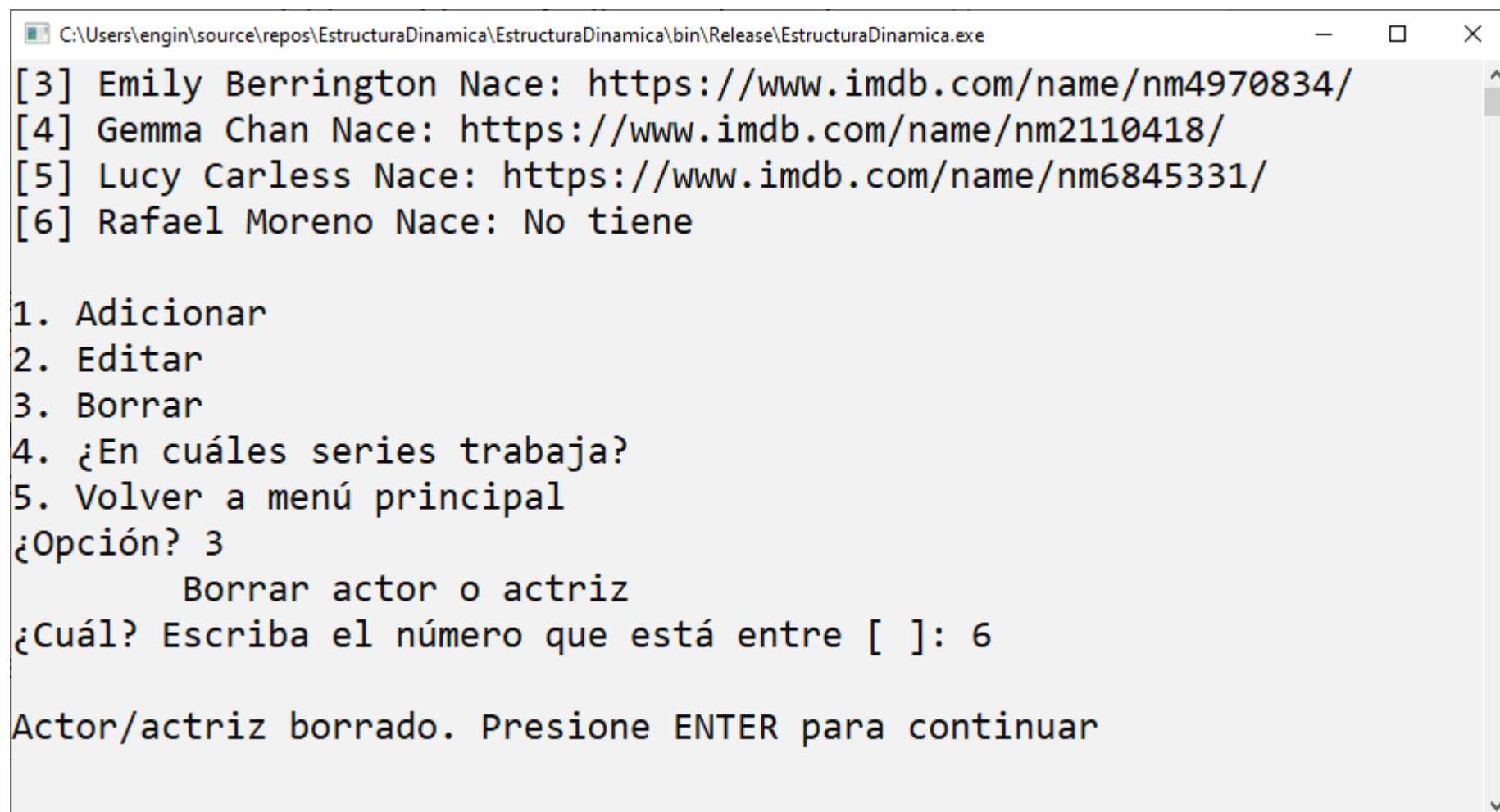
```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

[6] Rafael Alberto Moreno Parra Nace: https://www.imdb.com/name/nm1683292/

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal
¿Opción? 2
    Editar actor o actriz
¿Cuál? Escriba el número que está entre [ ]: 6
¿Nombre? Rafael Moreno
¿URL de IMDB? No tiene

Actor/actriz editado. Presione ENTER para continuar
```

Ilustración 34: Edita el actor o actriz



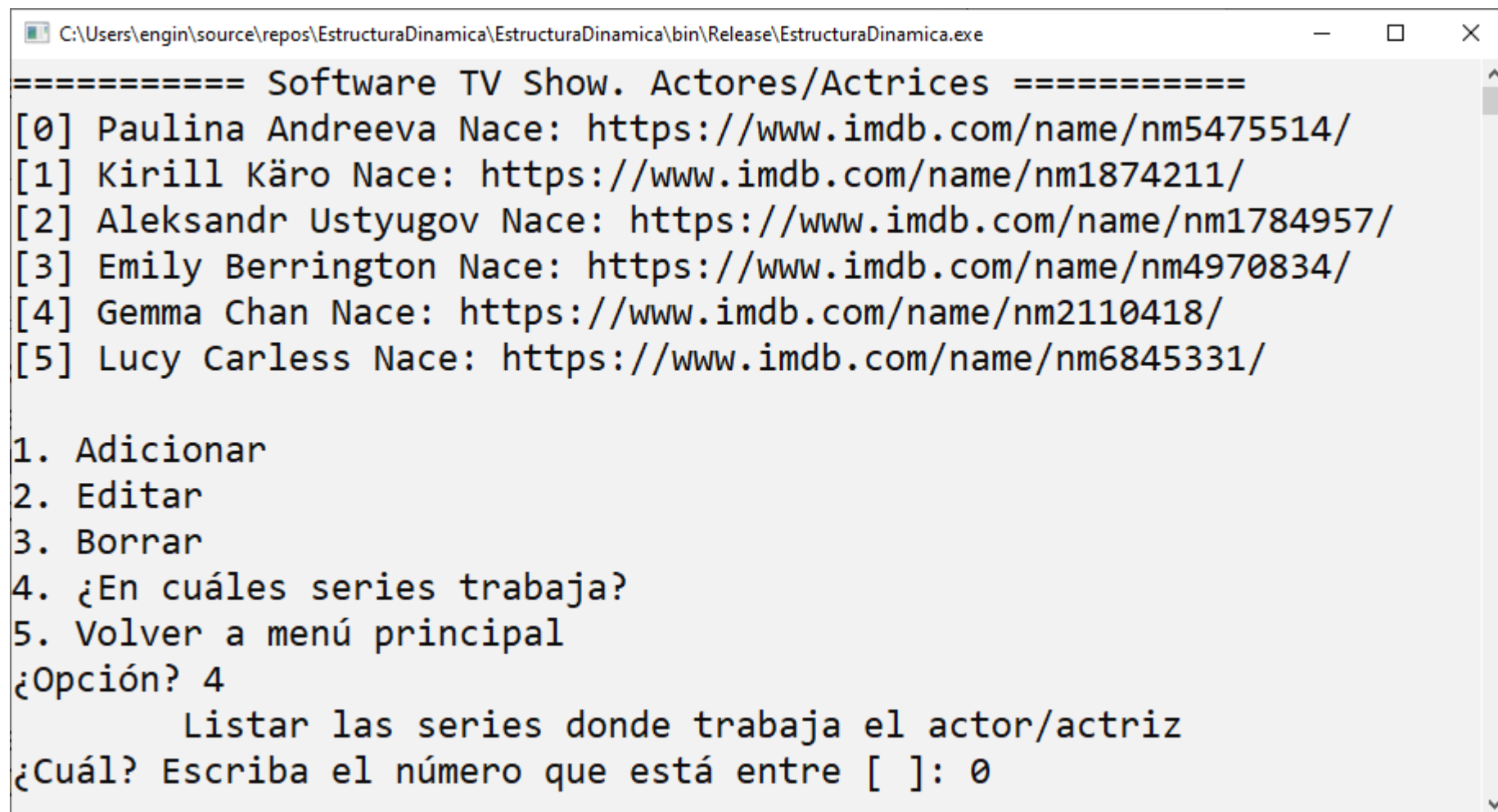
```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

[3] Emily Berrington Nace: https://www.imdb.com/name/nm4970834/
[4] Gemma Chan Nace: https://www.imdb.com/name/nm2110418/
[5] Lucy Carless Nace: https://www.imdb.com/name/nm6845331/
[6] Rafael Moreno Nace: No tiene

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal
¿Opción? 3
    Borrar actor o actriz
¿Cuál? Escriba el número que está entre [ ]: 6

Actor/actriz borrado. Presione ENTER para continuar
```

Ilustración 35: Borra actor o actriz

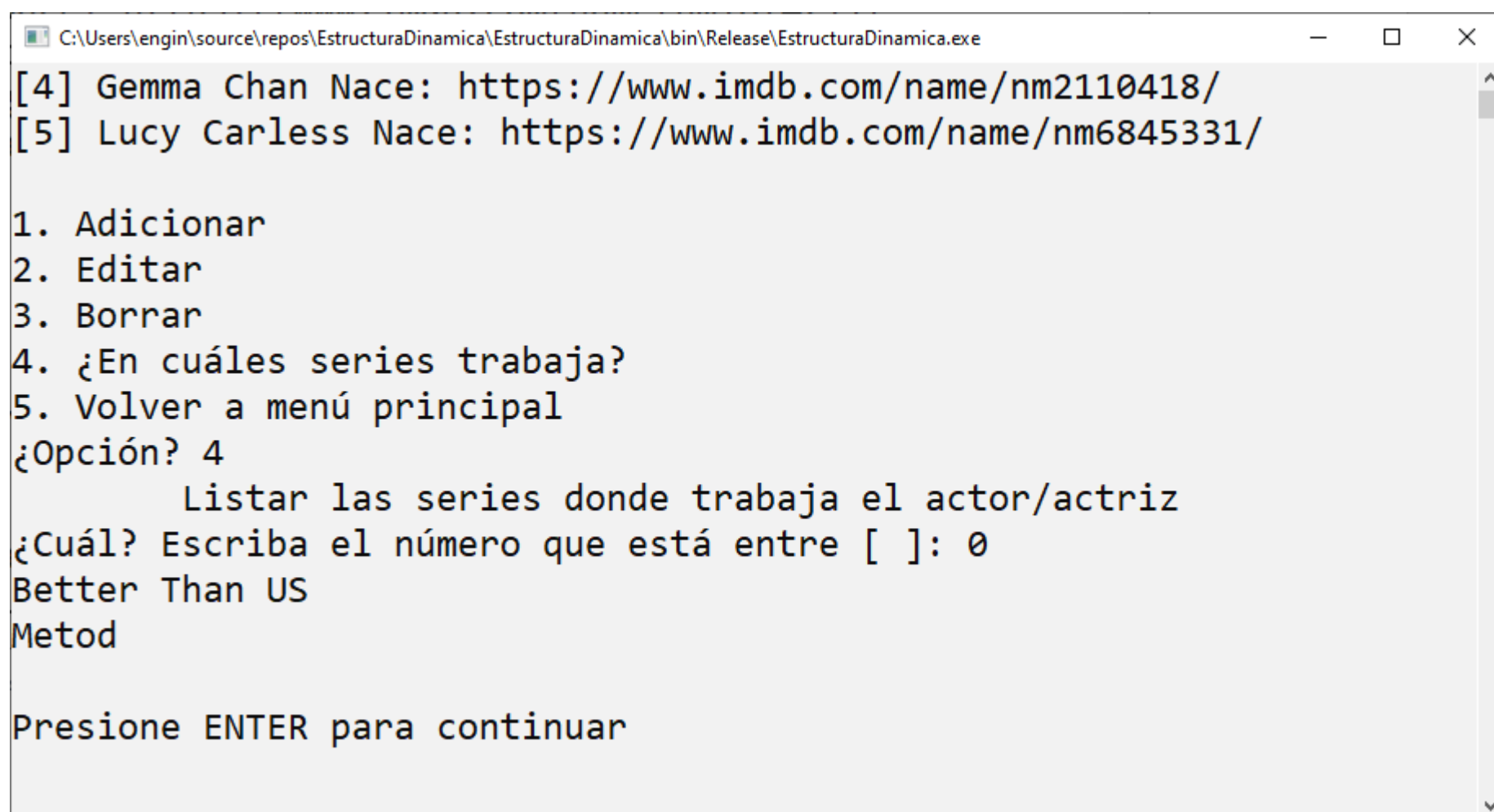


```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

===== Software TV Show. Actores/Actrices =====
[0] Paulina Andreeva Nace: https://www.imdb.com/name/nm5475514/
[1] Kirill Käro Nace: https://www.imdb.com/name/nm1874211/
[2] Aleksandr Ustyugov Nace: https://www.imdb.com/name/nm1784957/
[3] Emily Berrington Nace: https://www.imdb.com/name/nm4970834/
[4] Gemma Chan Nace: https://www.imdb.com/name/nm2110418/
[5] Lucy Carless Nace: https://www.imdb.com/name/nm6845331/

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal
¿Opción? 4
    Listar las series donde trabaja el actor/actriz
¿Cuál? Escriba el número que está entre [ ]: 0
```

Ilustración 36: Mostrar en que series trabaja el actor o actriz



```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

[4] Gemma Chan Nace: https://www.imdb.com/name/nm2110418/
[5] Lucy Carless Nace: https://www.imdb.com/name/nm6845331/

1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal
¿Opción? 4
    Listar las series donde trabaja el actor/actriz
¿Cuál? Escriba el número que está entre [ ]: 0
Better Than US
Metod

Presione ENTER para continuar
```

Ilustración 37: Listado de series donde trabaja el actor o actriz





```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

===== Software TV Show. Series =====
[0] Better Than US URL: https://www.imdb.com/title/tt8285216/
[1] Humans URL: https://www.imdb.com/title/tt4122068/
[2] Westworld URL: https://www.imdb.com/title/tt0475784/
[3] Real Humans URL: https://www.imdb.com/title/tt2180271/
[4] Almost Human URL: https://www.imdb.com/title/tt2654580/
[5] Battlestar Galactica URL: https://www.imdb.com/title/tt0407362/
[6] Metod URL: https://www.imdb.com/title/tt5135336/

1. Adicionar
2. Editar
3. Borrar
4. Detalles de la serie
5. Asociar actor/actriz a serie
6. Disociar actor/actriz a serie
7. Volver a menú principal
¿Opción? 1
```

Ilustración 38: Menú de series de televisión

Adicionar una serie. El CRUD es similar al de actores.



```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

[3] Real Humans URL: https://www.imdb.com/title/tt2180271/
[4] Almost Human URL: https://www.imdb.com/title/tt2654580/
[5] Battlestar Galactica URL: https://www.imdb.com/title/tt0407362/
[6] Metod URL: https://www.imdb.com/title/tt5135336/

1. Adicionar
2. Editar
3. Borrar
4. Detalles de la serie
5. Asociar actor/actriz a serie
6. Disociar actor/actriz a serie
7. Volver a menú principal
¿Opción? 1
    Adicionar serie al listado
¿Nombre? Sally Suini Capuchina Grisú Vikingo Milú
¿URL en IMDB? https://www.imdb.com/title/ttmsctsgts

Serie adicionada. Presione ENTER para continuar
```

Ilustración 39: Adiciona una serie de televisión

Se asocia un actor/actriz a una serie de televisión en particular



```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

[1] Humans URL: https://www.imdb.com/title/tt4122068/
[2] Westworld URL: https://www.imdb.com/title/tt0475784/
[3] Real Humans URL: https://www.imdb.com/title/tt2180271/
[4] Almost Human URL: https://www.imdb.com/title/tt2654580/
[5] Battlestar Galactica URL: https://www.imdb.com/title/tt0407362/
[6] Metod URL: https://www.imdb.com/title/tt5135336/
[7] Sally Suini Capuchina Grisú Vikingo Milú URL: https://www.imdb.com/title/ttmsctsgts

1. Adicionar
2. Editar
3. Borrar
4. Detalles de la serie
5. Asociar actor/actriz a serie
6. Disociar actor/actriz a serie
7. Volver a menú principal
¿Opción? 5
    Asocia un actor o actriz a una serie
¿Cuál serie? Escriba el número que está entre [ ]: 7
```

Ilustración 40: Adiciona un actor/actriz existente a una serie



```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

===== Software TV Show. Series =====
[0] Better Than US URL: https://www.imdb.com/title/tt8285216/
[1] Humans URL: https://www.imdb.com/title/tt4122068/
[2] Westworld URL: https://www.imdb.com/title/tt0475784/
[3] Real Humans URL: https://www.imdb.com/title/tt2180271/
[4] Almost Human URL: https://www.imdb.com/title/tt2654580/
[5] Battlestar Galactica URL: https://www.imdb.com/title/tt0407362/

[6] Metod URL: https://www.imdb.com/title/tt5135336/
[7] Sally Suini Capuchina Grisú Vikingo Milú URL: https://www.imdb.com/title/ttmsctsgts

1. Adicionar
2. Editar
3. Borrar
4. Detalles de la serie
5. Asociar actor/actriz a serie
6. Disociar actor/actriz a serie
7. Volver a menú principal
¿Opción? 5
    Asocia un actor o actriz a una serie
¿Cuál serie? Escriba el número que está entre [ ]: 7
[0] Paulina Andreeva URL: https://www.imdb.com/name/nm5475514/
[1] Kirill Käro URL: https://www.imdb.com/name/nm1874211/
[2] Aleksandr Ustyugov URL: https://www.imdb.com/name/nm1784957/
[3] Emily Berrington URL: https://www.imdb.com/name/nm4970834/
[4] Gemma Chan URL: https://www.imdb.com/name/nm2110418/
[5] Lucy Carless URL: https://www.imdb.com/name/nm6845331/
¿Cuál actor/actriz? Escriba el número que está entre [ ]: 3

Actor/actriz asociado a la serie. Presione ENTER para continuar
```

Ilustración 41: El actor/actriz entonces es asignado a una serie de televisión

El resultado se puede apreciar al mostrar los detalles de una serie:



```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

===== Software TV Show. Series =====
[0] Better Than US URL: https://www.imdb.com/title/tt8285216/
[1] Humans URL: https://www.imdb.com/title/tt4122068/
[2] Westworld URL: https://www.imdb.com/title/tt0475784/
[3] Real Humans URL: https://www.imdb.com/title/tt2180271/
[4] Almost Human URL: https://www.imdb.com/title/tt2654580/
[5] Battlestar Galactica URL: https://www.imdb.com/title/tt0407362/

[6] Metod URL: https://www.imdb.com/title/tt5135336/
[7] Sally Suini Capuchina Grisú Vikingo Milú URL: https://www.imdb.
com/title//ttmsctsgts

1. Adicionar
2. Editar
3. Borrar
4. Detalles de la serie
5. Asociar actor/actriz a serie
6. Disociar actor/actriz a serie
7. Volver a menú principal
¿Opción? 4
    === Detalle de una serie ===
¿Cuál? Escriba el número que está entre [ ]: 7
Nombre: Sally Suini Capuchina Grisú Vikingo Milú
URL: https://www.imdb.com/title//ttmsctsgts
Actores
    Emily Berrington

Presione ENTER para continuar
```

Ilustración 42: El detalle de una serie muestra los actores que trabajan allí.

También se puede quitar un actor/actriz de la serie de televisión.



```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

===== Software TV Show. Series =====
[0] Better Than US URL: https://www.imdb.com/title/tt8285216/
[1] Humans URL: https://www.imdb.com/title/tt4122068/
[2] Westworld URL: https://www.imdb.com/title/tt0475784/
[3] Real Humans URL: https://www.imdb.com/title/tt2180271/
[4] Almost Human URL: https://www.imdb.com/title/tt2654580/
[5] Battlestar Galactica URL: https://www.imdb.com/title/tt0407362/

[6] Metod URL: https://www.imdb.com/title/tt5135336/
[7] Sally Suini Capuchina Grisú Vikingo Milú URL: https://www.imdb.
com/title//ttmsctsgts

1. Adicionar
2. Editar
3. Borrar
4. Detalles de la serie
5. Asociar actor/actriz a serie
6. Disociar actor/actriz a serie
7. Volver a menú principal
¿Opción? 6
      === Disociar actor de la serie ===
¿Cuál serie? Escriba el número que está entre [ ]: 7
[0] Emily Berrington
¿Cuál actor/actriz quiere quitar? Escriba el número que está entre
[ ]: 0

Actor/actriz retirado de la serie. Presione ENTER para continuar
```

Ilustración 43: Retira un actor o actriz de una serie

# Dictionary

## Uso de llaves

En una estructura diccionario [4], hay una llave y un valor (entero, cadena, objeto). Puede llegar a ese valor usando la llave. Con la instrucción: NombreDiccionario[Llave]. Ejemplo:

Carpeta 024. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            Random Azar = new Random();

            //Se define un diccionario: llave, cadena
            //En este caso la llave es un número entero
            Dictionary<int, string> Animales = new Dictionary<int, string> {
                {11, "Ballena"},
                {12, "Tortuga marina"},
                {13, "Tiburón"},
                {14, "Estrella de mar"},
                {15, "Hipocampo"},
                {16, "Serpiente marina"},
                {17, "Delfín"},
                {18, "Pulpo"},
                {19, "Caballito de mar"},
                {20, "Coral"},
                {21, "Pingüinos"},
                {22, "Calamar"},
                {23, "Gaviota"},
                {24, "Foca"},
                {25, "Manaties"},
                {26, "Ballena con barba"},
                {27, "Peces Guppy"},
                {28, "Orca"},
                {29, "Medusas"},
                {30, "Mejillones"},
                {31, "Caracoles"}
            };

            for (int cont = 1; cont <= 10; cont++) {
                int Llave = Azar.Next(11, Animales.Count + 11);
                Console.WriteLine("Llave: " + Llave + " cadena: " + Animales[Llave]);
            }

            Console.ReadKey();
        }
    }
}
```

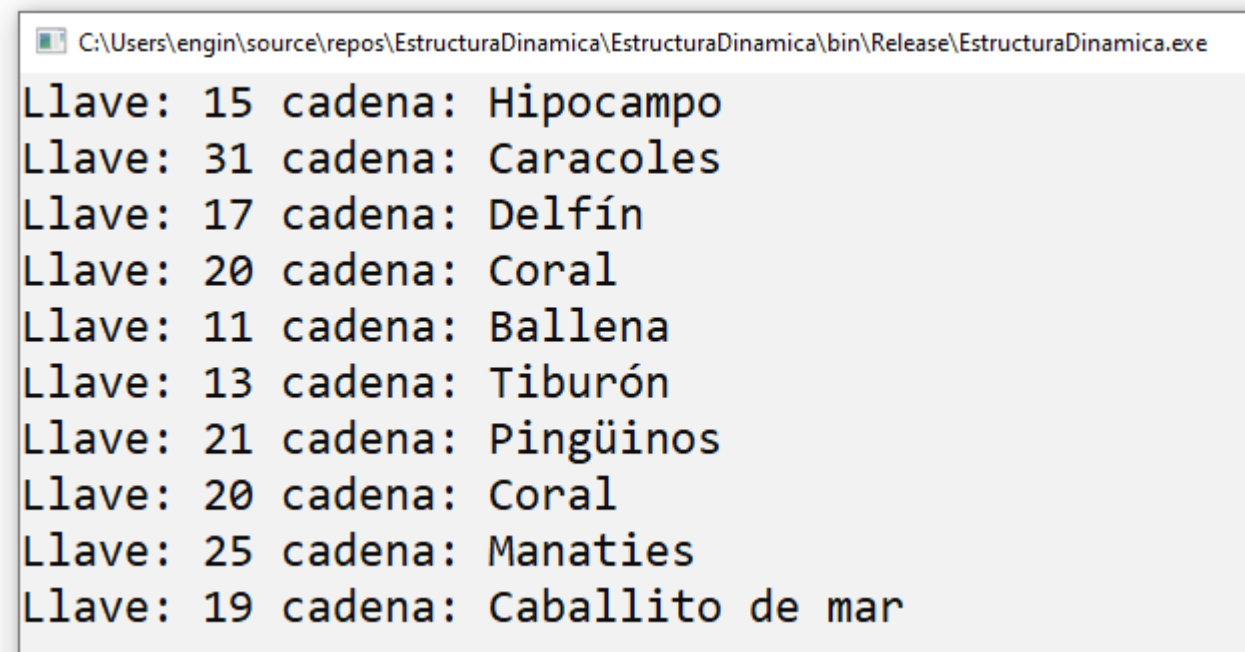


Ilustración 44: Busca llaves al azar y trae el valor

## Llaves tipo string

También puede usar una llave de tipo cadena. El diccionario también tiene instrucciones de adicionar y borrar.

Carpeta 025. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define un diccionario: llave, cadena
            //En este caso la llave es una cadena
            Dictionary<string, string> Extensiones = new Dictionary<string, string> {
                {"exe", "Ejecutable"},
                {"com", "Ejecutable DOS"},
                {"vb", "Visual Basic .NET"},
                {"cs", "C#"},
                {"js", "JavaScript"},
                {"xlsx", "Excel"},
                {"docx", "Word"},
                {"html", "HTML 5"}
            };

            Extensiones.Add("pptx", "PowerPoint"); //Otra forma de adicionar

            //Trae un elemento dada una llave
            string Llave = "cs";
            Console.WriteLine("Llave: " + Llave + " valor es: " + Extensiones[Llave]);

            //Tamaño del diccionario
            Console.WriteLine("Tamaño:" + Extensiones.Count);

            //Elimina un elemento
            Extensiones.Remove("docx");

            //Tamaño del diccionario
            Console.WriteLine("Después de eliminar. Tamaño: " + Extensiones.Count);

            Console.ReadKey();
        }
    }
}
```

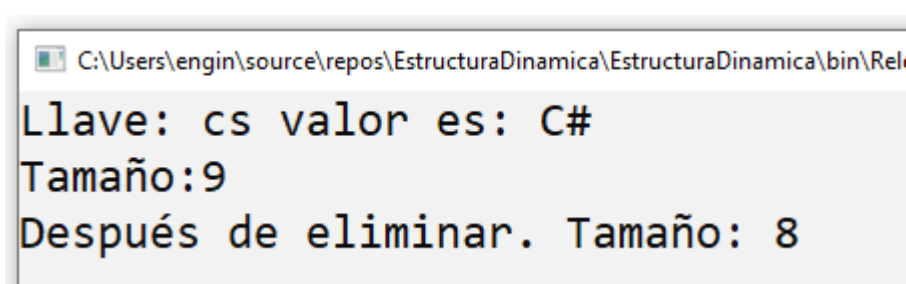


Ilustración 45: Llave de tipo string y eliminar elemento usando llave



# Manejo de objetos en un Dictionary

Un “Dictionary” puede albergar objetos. Además, tiene una serie de métodos (adicionar, consultar, listar llaves, verificar si existe llave) que se ven a continuación:

Carpeta 026. Ejemplo.cs

```
namespace EstructuraDinamica {
    //Una clase con varios atributos
    class Ejemplo {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public Ejemplo(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }
}
```

Carpeta 026. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define un diccionario: llave, objeto
            //En este caso la llave es una cadena
            var Objetos = new Dictionary<string, Ejemplo> {
                {"uno", new Ejemplo(1, 0.2, 'r', "Leafar") },
                {"dos", new Ejemplo(8, -7.1, 'a', "Otrebla")},
                {"tres", new Ejemplo(23, -13.6, 'm', "Onerom")},
                {"cuatro", new Ejemplo(49, 16.83, 'p', "Arrap")}
            };

            //Trae los datos del objeto guardado en el diccionario
            string Llave = "tres";
            Console.WriteLine("Llave: " + Llave + " atributo es: " + Objetos[Llave].Cadena);
            Console.WriteLine("Llave: " + Llave + " atributo es: " + Objetos[Llave].Numero);
            Console.WriteLine("Llave: " + Llave + " atributo es: " + Objetos[Llave].Valor);

            //Guarda las llaves en una lista
            Console.WriteLine("\r\nLista de Llaves:");
            var ListaLlaves = new List<string>(Objetos.Keys);
            foreach (string Llaves in ListaLlaves) {
                Console.WriteLine("Llave: " + Llaves);
            }

            //Verifica si existe una llave
            Console.WriteLine("\r\nVerifica si existe una llave:");
            if (Objetos.ContainsKey("cuatro")) {
                Console.WriteLine(Objetos["cuatro"].Cadena);
            }
            else {
                Console.WriteLine("No existe esa llave");
            }

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\
Llave: tres atributo es: Onerom
Llave: tres atributo es: 23
Llave: tres atributo es: -13,6

Lista de Llaves:
Llave: uno
Llave: dos
Llave: tres
Llave: cuatro

Verifica si existe una llave:
Arrap
```

Ilustración 46: Dictionary maneja objetos y tiene varios métodos para adicionar, consultar, listar llaves, verificar si existe llave



## Queue (Cola)

Una cola [5] se parece a un ArrayList, la diferencia es que NO se puede acceder a los elementos por un índice, se respeta el orden de llegada, primero en entrar es primero en salir.

Carpeta 027. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una cola: Queue
            Queue Cola = new Queue();

            //Se agregan elementos a la cola
            Cola.Enqueue("aaa");
            Cola.Enqueue("bbb");
            Cola.Enqueue("ccc");
            Cola.Enqueue("ddd");
            Cola.Enqueue("eee");
            Cola.Enqueue("fff");

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach(object elemento in Cola) Console.Write(elemento + ", ");

            //Quitar elemento de la cola
            Cola.Dequeue(); //Primero en llegar, primero en salir, luego quitaría a "aaa"
            Console.WriteLine("\r\nAl quitar un elemento de la cola: ");
            foreach (object elemento in Cola) Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la cola
            string Buscar = "ddd";
            if (Cola.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa cola contiene el elemento: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa cola NO contiene el elemento: " + Buscar);

            //Obtener el primer elemento de la cola sin borrar ese elemento
            string PrimerElemento = Convert.ToString(Cola.Peek());
            Console.WriteLine("\r\nPrimer elemento de la cola: " + PrimerElemento);

            //Leer y borrar la cola
            Console.WriteLine("\r\nLee y borra la cola: ");
            while (Cola.Count > 0)
                Console.Write(Cola.Dequeue() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Relea
Número de elementos: 6

Elementos:
aaa, bbb, ccc, ddd, eee, fff,
Al quitar un elemento de la cola:
bbb, ccc, ddd, eee, fff,

La cola contiene el elemento: ddd

Primer elemento de la cola: bbb

Lee y borra la cola:
bbb; ccc; ddd; eee; fff;
Número de elementos: 0
```

*Ilustración 47: Trabajo con colas*

## Dato definido en la cola

Los elementos de la cola pueden ser de tipo definido. Se modifica el programa anterior para que trabaje con el tipo string, obteniendo el mismo resultado.

Carpeta 028. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una cola de tipo string: Queue
            Queue<string> Cola = new Queue<string>();

            //Se agregan elementos a la cola
            Cola.Enqueue("aaa");
            Cola.Enqueue("bbb");
            Cola.Enqueue("ccc");
            Cola.Enqueue("ddd");
            Cola.Enqueue("eee");
            Cola.Enqueue("fff");

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach(object elemento in Cola) Console.Write(elemento + ", ");

            //Quitar elemento de la cola
            Cola.Dequeue(); //Primero en llegar, primero en salir, luego quitaría a "aaa"
            Console.WriteLine("\r\nAl quitar un elemento de la cola: ");
            foreach (object elemento in Cola) Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la cola
            string Buscar = "ddd";
            if (Cola.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa cola contiene el elemento: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa cola NO contiene el elemento: " + Buscar);

            //Obtener el primer elemento de la cola sin borrar ese elemento
            string PrimerElemento = Cola.Peek();
            Console.WriteLine("\r\nPrimer elemento de la cola: " + PrimerElemento);

            //Leer y borrar la cola
            Console.WriteLine("\r\nLee y borra la cola: ");
            while (Cola.Count > 0)
                Console.Write(Cola.Dequeue() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);

            Console.ReadKey();
        }
    }
}
```

## Objetos en la cola

Una cola puede tener objetos personalizados.

Carpeta 029. Ejemplo.cs

```
namespace EstructuraDinamica {
    //Una clase con varios atributos
    class Ejemplo {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public Ejemplo(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }
}
```

Carpeta 029. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una cola de tipo objeto personalizado
            Queue<Ejemplo> Cola = new Queue<Ejemplo>();

            //Se agregan elementos a la cola
            Cola.Enqueue(new Ejemplo(1, 0.2, 'r', "Leafar"));
            Cola.Enqueue(new Ejemplo(8, -7.1, 'a', "Otrebla"));
            Cola.Enqueue(new Ejemplo(23, -13.6, 'm', "Onerom"));
            Cola.Enqueue(new Ejemplo(49, 16.83, 'p', "Arrap"));

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach (Ejemplo elemento in Cola) Console.Write(elemento.Cadena + ", ");

            //Quitar elemento de la cola
            Cola.Dequeue(); //Primero en llegar, primero en salir
            Console.WriteLine("\r\nAl quitar un elemento de la cola: ");
            foreach (Ejemplo elemento in Cola) Console.Write(elemento.Cadena + ", ");

            //Obtener el primer elemento de la cola sin borrar ese elemento
            Ejemplo PrimerElemento = Cola.Peek();
            Console.WriteLine("\r\n\r\nPrimer elemento de la cola: " + PrimerElemento.Cadena);

            //Leer y borrar la cola
            Console.WriteLine("\r\nLee y borra la cola: ");
            while (Cola.Count > 0)
                Console.Write(Cola.Dequeue().Cadena + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);

            //Agrega elementos a la cola y luego la borra
            Cola.Enqueue(new Ejemplo(7, 6.5, 'z', "qwerty"));
            Cola.Enqueue(new Ejemplo(4, -3.2, 'y', "asdfg"));
            Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);
            Cola.Clear();
            Console.WriteLine("Después de borrar. Número de elementos: " + Cola.Count);

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinam
Número de elementos: 4

Elementos:
Leafar, Otrebla, Onerom, Arrap,
Al quitar un elemento de la cola:
Otrebla, Onerom, Arrap,

Primer elemento de la cola: Otrebla

Lee y borra la cola:
Otrebla; Onerom; Arrap;
Número de elementos: 0

Número de elementos: 2
Después de borrar. Número de elementos: 0
```

*Ilustración 48; Objetos personalizados en la cola*

## Stack (Pila)

La pila [6] es una estructura que análogo a una pila de platos, cuando se adicionan elementos, estos van quedando encima, por lo que el último en entrar es el primero en salir. Es muy similar a la cola, sólo cambian algunos métodos como el Push (poner) y Pop (retirar).

Carpeta 030. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una pila: Queue
            Stack Pila = new Stack();

            //Se agregan elementos a la pila
            Pila.Push("aaa");
            Pila.Push("bbb");
            Pila.Push("ccc");
            Pila.Push("ddd");
            Pila.Push("eee");
            Pila.Push("fff");

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (object elemento in Pila) Console.Write(elemento + ", ");

            //Quitar elemento de la pila
            Pila.Pop(); //Último en llegar, primero en salir, luego quitaría a "fff"
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (object elemento in Pila) Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la pila
            string Buscar = "ddd";
            if (Pila.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa pila contiene el elemento: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa pila NO contiene el elemento: " + Buscar);

            //Obtener el primer elemento de la pila sin borrar ese elemento
            string PrimerElemento = Convert.ToString(Pila.Peek());
            Console.WriteLine("\r\nPrimer elemento de la pila: " + PrimerElemento);

            //Leer y borrar la pila
            Console.WriteLine("\r\nLee y borra la pila: ");
            while (Pila.Count > 0)
                Console.Write(Pila.Pop() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Releas
Número de elementos: 6

Elementos:
fff, eee, ddd, ccc, bbb, aaa,
Al quitar un elemento de la pila:
eee, ddd, ccc, bbb, aaa,

La pila contiene el elemento: ddd

Primer elemento de la pila: eee

Lee y borra la pila:
eee; ddd; ccc; bbb; aaa;
Número de elementos: 0
```

Ilustración 49: Pilas



## Dato definido en la pila

Los elementos de la pila pueden ser de tipo definido. Se modifica el programa anterior para que trabaje con el tipo string, obteniendo el mismo resultado.

Carpeta 031. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una pila: Queue
            Stack<string> Pila = new Stack<string>();

            //Se agregan elementos a la pila
            Pila.Push("aaa");
            Pila.Push("bbb");
            Pila.Push("ccc");
            Pila.Push("ddd");
            Pila.Push("eee");
            Pila.Push("fff");

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (object elemento in Pila) Console.Write(elemento + ", ");

            //Quitar elemento de la pila
            Pila.Pop(); //Último en llegar, primero en salir, luego quitaría a "fff"
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (string elemento in Pila) Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la pila
            string Buscar = "ddd";
            if (Pila.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa pila contiene el elemento: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa pila NO contiene el elemento: " + Buscar);

            //Obtener el primer elemento de la pila sin borrar ese elemento
            string PrimerElemento = Pila.Peek();
            Console.WriteLine("\r\nPrimer elemento de la pila: " + PrimerElemento);

            //Leer y borrar la pila
            Console.WriteLine("\r\nLee y borra la pila: ");
            while (Pila.Count > 0)
                Console.Write(Pila.Pop() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);

            Console.ReadKey();
        }
    }
}
```

```
namespace EstructuraDinamica {
    //Una clase con varios atributos
    class Ejemplo {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Character { get; set; }
        public string Cadena { get; set; }

        public Ejemplo(int Numero, double Valor, char Character, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Character = Character;
            this.Cadena = Cadena;
        }
    }
}
```

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una pila de tipo objeto personalizado
            Stack<Ejemplo> Pila = new Stack<Ejemplo>();

            //Se agregan elementos a la pila
            Pila.Push(new Ejemplo(1, 0.2, 'r', "Leafar"));
            Pila.Push(new Ejemplo(8, -7.1, 'a', "Otrebla"));
            Pila.Push(new Ejemplo(23, -13.6, 'm', "Onerom"));
            Pila.Push(new Ejemplo(49, 16.83, 'p', "Arrap"));

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (Ejemplo elemento in Pila) Console.Write(elemento.Cadena + ", ");

            //Quitar elemento de la pila
            Pila.Pop(); //Último en llegar, primero en salir
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (Ejemplo elemento in Pila) Console.Write(elemento.Cadena + ", ");

            //Obtener el primer elemento de la pila sin borrar ese elemento
            Ejemplo PrimerElemento = Pila.Peek();
            Console.WriteLine("\r\n\r\nElemento más arriba de la pila: " + PrimerElemento.Cadena);

            //Leer y borrar la pila
            Console.WriteLine("\r\nLee y borra la pila: ");
            while (Pila.Count > 0)
                Console.Write(Pila.Pop().Cadena + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);

            //Agrega elementos a la pila y luego la borra
            Pila.Push(new Ejemplo(7, 6.5, 'z', "qwerty"));
            Pila.Push(new Ejemplo(4, -3.2, 'y', "asdfg"));
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);
            Pila.Clear();
            Console.WriteLine("Después de borrar. Número de elementos: " + Pila.Count);

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe
Número de elementos: 4

Elementos:
Arrap, Onerom, Otrebla, Leafar,
Al quitar un elemento de la pila:
Onerom, Otrebla, Leafar,

Elemento más arriba de la pila: Onerom

Lee y borra la pila:
Onerom; Otrebla; Leafar;
Número de elementos: 0

Número de elementos: 2
Después de borrar. Número de elementos: 0
```

Ilustración 50: Uso de pilas (Stack)

# Hashtable

Hashtable [7] funciona similar a Dictionary. Estas son sus diferencias:

Hashtable	Dictionary
Es seguro ser accedido por múltiples hilos ("thread safe").	Sólo miembros públicos estáticos son seguros para ser accedidos por hilos.
Retorna "null" si se intenta acceder a un dato por una llave inexistente.	Genera un error si intenta acceder por una llave inexistente. Requiere usar try catch.
La recuperación de datos es más lenta.	La recuperación de datos es más rápida.
No requiere definir el tipo de dato de la llave y el valor.	Requiere definir el tipo de datos de la llave y el valor.

Carpeta 033. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            Random Azar = new Random();

            //Se define un Hashtable
            //En este caso la llave es un número entero
            Hashtable Animales = new Hashtable();
            Animales.Add(11, "Ballena");
            Animales.Add(12, "Tortuga marina");
            Animales.Add(13, "Tiburón");
            Animales.Add(14, "Estrella de mar");
            Animales.Add(15, "Hipocampo");
            Animales.Add(16, "Serpiente marina");
            Animales.Add(17, "Delfín");
            Animales.Add(18, "Pulpo");
            Animales.Add(19, "Caballito de mar");
            Animales.Add(20, "Coral");
            Animales.Add(21, "Pingüinos");
            Animales.Add(22, "Calamar");
            Animales.Add(23, "Gaviota");
            Animales.Add(24, "Foca");
            Animales.Add(25, "Manaties");
            Animales.Add(26, "Ballena con barba");
            Animales.Add(27, "Peces Guppy");
            Animales.Add(28, "Orca");
            Animales.Add(29, "Medusas");
            Animales.Add(30, "Mejillones");
            Animales.Add(31, "Caracoles");

            for (int cont = 1; cont <= 10; cont++) {
                int Llave = Azar.Next(11, Animales.Count + 11);
                Console.WriteLine("Llave: " + Llave + " cadena: " + Animales[Llave]);
            }

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe
Llave: 30 cadena: Mejillones
Llave: 26 cadena: Ballena con barba
Llave: 24 cadena: Foca
Llave: 26 cadena: Ballena con barba
Llave: 24 cadena: Foca
Llave: 13 cadena: Tiburón
Llave: 22 cadena: Calamar
Llave: 13 cadena: Tiburón
Llave: 31 cadena: Caracoles
Llave: 21 cadena: Pingüinos
```

*Ilustración 51: Uso de Hashtable*

Manejo de objetos en un Hashtable

Cabe recordar que hay que hacer la conversión para acceder a los atributos del objeto almacenado así:

```
(objeto as clase).atributo
```

Carpeta 034. Ejemplo.cs

```
namespace EstructuraDinamica {
    //Una clase con varios atributos
    class Ejemplo {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public Ejemplo(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }
}
```

Carpeta 034. Program.cs

```
using System;
using System.Collections;

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define un Hashtable
            Hashtable Objetos = new Hashtable();

            Objetos.Add("uno", new Ejemplo(1, 0.2, 'r', "Leafar"));
            Objetos.Add("dos", new Ejemplo(8, -7.1, 'a', "Otrebla"));
            Objetos.Add("tres", new Ejemplo(23, -13.6, 'm', "Onerom"));
            Objetos.Add("cuatro", new Ejemplo(49, 16.83, 'p', "Arrap"));

            //Trae los datos del objeto guardado en el diccionario
            string Llave = "tres";
            Console.WriteLine("Llave: " + Llave + " atributo es: " + (Objetos[Llave] as Ejemplo).Cadena);
            Console.WriteLine("Llave: " + Llave + " atributo es: " + (Objetos[Llave] as Ejemplo).Numero);
            Console.WriteLine("Llave: " + Llave + " atributo es: " + (Objetos[Llave] as Ejemplo).Valor);

            //Guarda las llaves en una variable de colección
            Console.WriteLine("\r\nLista de Llaves:");
            var ListaLlaves = Objetos.Keys;
            foreach (string Llaves in ListaLlaves) {
                Console.WriteLine("Llave: " + Llaves);
            }

            //Verifica si existe una llave
            Console.WriteLine("\r\nVerifica si existe una llave:");
            if (Objetos.ContainsKey("cuatro")) {
                Console.WriteLine((Objetos["cuatro"] as Ejemplo).Cadena);
            }
            else {
                Console.WriteLine("No existe esa llave");
            }

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\l
Llave: tres atributo es: Onerom
Llave: tres atributo es: 23
Llave: tres atributo es: -13,6

Lista de Llaves:
Llave: dos
Llave: tres
Llave: cuatro
Llave: uno

Verifica si existe una llave:
Arrap
```

Ilustración 52: Objetos y Hashtable



# SortedList

SortedList [8] es muy similar a Dictionary, en este caso la lista es ordenada automáticamente por las llaves. Eso es visible al imprimirla.

Carpeta 035. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una lista ordenada: llave, cadena
            //En este caso la llave es una cadena
            SortedList<string, string> Extensiones = new SortedList<string, string> {
                {"exe", "Ejecutable"},
                {"com", "Ejecutable DOS"},
                {"vb", "Visual Basic .NET"},
                {"cs", "C#"},
                {"js", "JavaScript"},
                {"xlsx", "Excel"},
                {"docx", "Word"},
                {"html", "HTML 5"}
            };

            Extensiones.Add("pptx", "PowerPoint"); //Otra forma de adicionar

            //Imprime la lista ordenada
            foreach (object elemento in Extensiones) Console.WriteLine(elemento);

            //Imprime llave y valor
            var ListaLlaves = Extensiones.Keys;
            Console.WriteLine("\r\nImprime llave y valor en separado");
            foreach (string Llave in ListaLlaves) {
                Console.WriteLine("Llave: " + Llave + " Valor: " + Extensiones[Llave]);
            }
            Console.ReadKey();
        }
    }
}
```

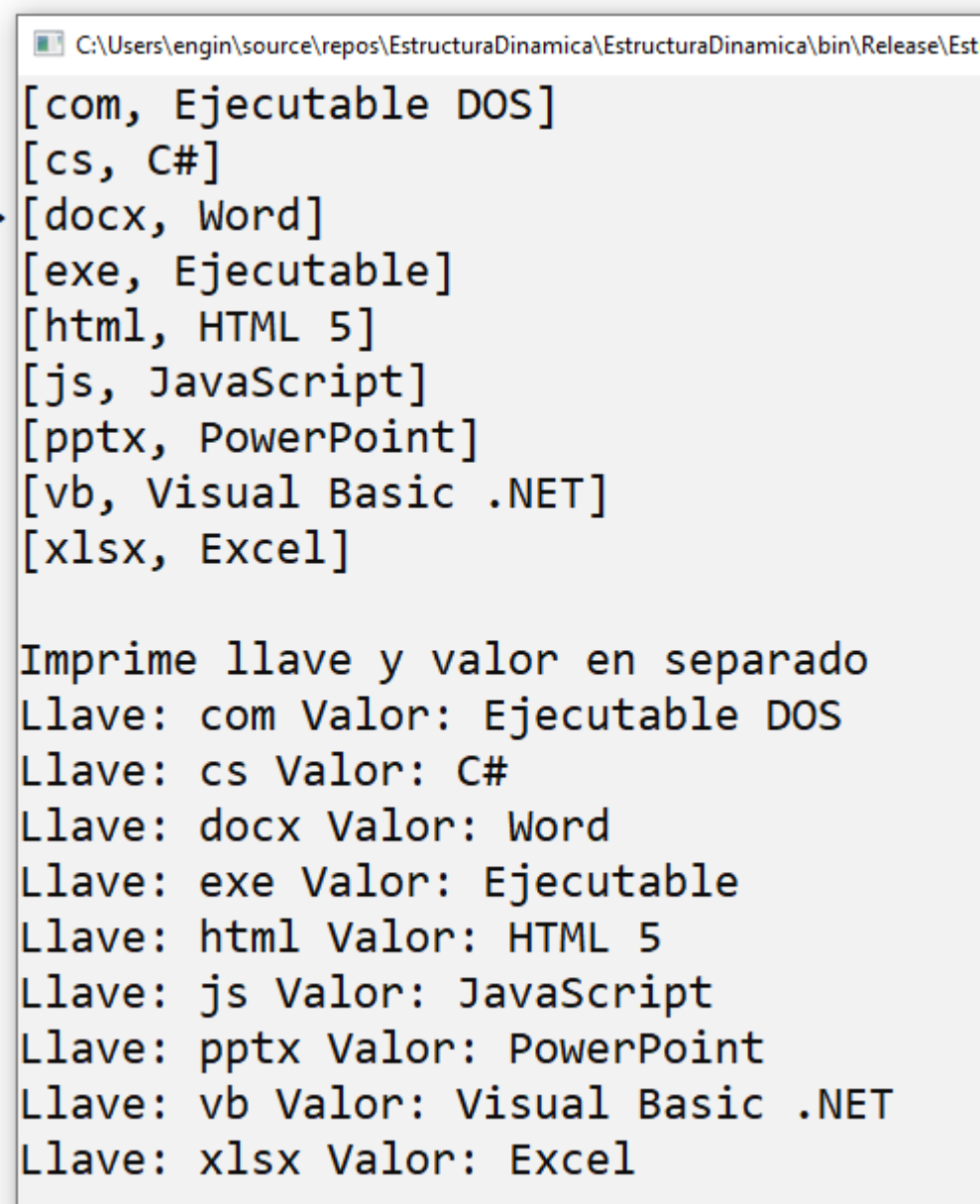


Ilustración 53: SortedList

# LinkedList

Lista enlazada [9]. no se accede directamente por un índice.

Carpeta 036. Program.cs

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

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una lista enlazada
            LinkedList<string> Lenguajes = new LinkedList<string>();

            //Agrega al final
            Lenguajes.AddLast("Visual Basic .NET");
            Lenguajes.AddLast("F#");
            Lenguajes.AddLast("C#");
            Lenguajes.AddLast("TypeScript");

            //Imprime esa lista
            Console.WriteLine("Agregando con AddLast");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Agrega al inicio
            Lenguajes.AddFirst("C++");
            Lenguajes.AddFirst("C");

            //Imprime esa lista
            Console.WriteLine("\r\n\r\nAgregando con AddFirst");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Agrega al final
            Lenguajes.AddLast("Python");
            Console.WriteLine("\r\n\r\nAgregando con AddLast");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Cantidad
            Console.WriteLine("\r\n\r\nCantidad es: " + Lenguajes.Count);

            //Elimina primer elemento
            Lenguajes.RemoveFirst();
            Console.WriteLine("\r\n\r\nEliminado el primer elemento");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Elimina último elemento
            Lenguajes.RemoveLast();
            Console.WriteLine("\r\n\r\nEliminado el último elemento");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Elimina determinado elemento
            Lenguajes.Remove("F#");
            Console.WriteLine("\r\n\r\n\r\nEliminado F#");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Adiciona antes de C#
            LinkedListNode<string> nodoPosiciona = Lenguajes.Find("C#"); //Busca el nodo que tiene C#
            Lenguajes.AddBefore(nodoPosiciona, "Assembler");
            Console.WriteLine("\r\n\r\n\r\nAdiciona antes de C#");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            //Adiciona después de C#
            Lenguajes.AddAfter(nodoPosiciona, "Ada");
            Console.WriteLine("\r\n\r\n\r\nAdiciona después de C#");
            foreach (string elemento in Lenguajes) Console.Write(elemento + "; ");

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

Agregando con AddLast
Visual Basic .NET; F#; C#; TypeScript;

Agregando con AddFirst
C; C++; Visual Basic .NET; F#; C#; TypeScript;

Agregando con AddLast
C; C++; Visual Basic .NET; F#; C#; TypeScript; Python;

Cantidad es: 7

Eliminado el primer elemento
C++; Visual Basic .NET; F#; C#; TypeScript; Python;

Eliminado el último elemento
C++; Visual Basic .NET; F#; C#; TypeScript;

Eliminado F#
C++; Visual Basic .NET; C#; TypeScript;

Adiciona antes de C#
C++; Visual Basic .NET; Assembler; C#; TypeScript;

Adiciona después de C#
C++; Visual Basic .NET; Assembler; C#; Ada; TypeScript;
```

Ilustración 54: Uso de LinkedList

```

namespace EstructuraDinamica {
    //Una clase con varios atributos
    class Ejemplo {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Caracter { get; set; }
        public string Cadena { get; set; }

        public Ejemplo(int Numero, double Valor, char Caracter, string Cadena) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Caracter = Caracter;
            this.Cadena = Cadena;
        }
    }
}

```

```

using System;
using System.Collections.Generic;

namespace EstructuraDinamica {
    class Program {
        static void Main() {
            //Se define una lista enlazada
            LinkedList<Ejemplo> Lenguajes = new LinkedList<Ejemplo>();

            //Agrega al final
            Lenguajes.AddLast(new Ejemplo(16, 83.29, 'R', "Lenguaje R"));
            Lenguajes.AddLast(new Ejemplo(29, 89.7, 'A', "ADA"));
            Lenguajes.AddLast(new Ejemplo(2, 80.19, 'M', "Máquina"));
            Lenguajes.AddLast(new Ejemplo(95, 7.21, 'P', "PHP"));

            //Imprime esa lista
            Console.WriteLine("Agregando con AddLast");
            foreach (Ejemplo elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

            //Agrega al inicio
            Lenguajes.AddFirst(new Ejemplo(78, 12.32, 'C', "C#"));
            Lenguajes.AddFirst(new Ejemplo(55, -3.21, 'V', "Visual Basic .NET"));

            //Imprime esa lista
            Console.WriteLine("\r\n\r\nAgregando con AddFirst");
            foreach (Ejemplo elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

            //Agrega al final
            Lenguajes.AddLast(new Ejemplo(16, 83.29, 'T', "TypeScript"));
            Console.WriteLine("\r\n\r\nAgregando con AddLast");
            foreach (Ejemplo elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

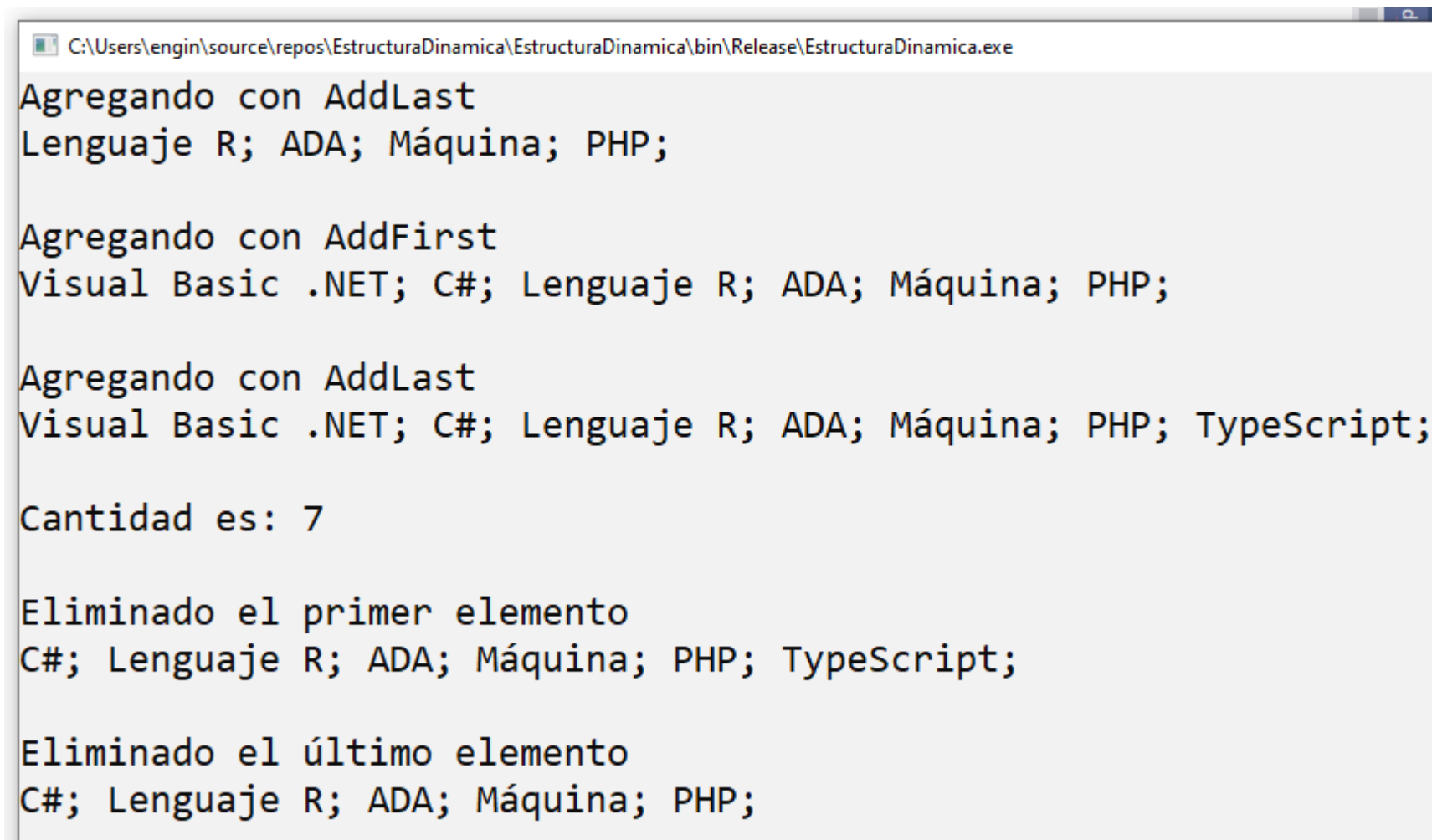
            //Cantidad
            Console.WriteLine("\r\n\r\nCantidad es: " + Lenguajes.Count);

            //Elimina primer elemento
            Lenguajes.RemoveFirst();
            Console.WriteLine("\r\nEliminado el primer elemento");
            foreach (Ejemplo elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

            //Elimina último elemento
            Lenguajes.RemoveLast();
            Console.WriteLine("\r\n\r\nEliminado el último elemento");
            foreach (Ejemplo elemento in Lenguajes) Console.Write(elemento.Cadena + "; ");

            Console.ReadKey();
        }
    }
}

```



```
C:\Users\engin\source\repos\EstructuraDinamica\EstructuraDinamica\bin\Release\EstructuraDinamica.exe

Agregando con AddLast
Lenguaje R; ADA; Máquina; PHP;

Agregando con AddFirst
Visual Basic .NET; C#; Lenguaje R; ADA; Máquina; PHP;

Agregando con AddLast
Visual Basic .NET; C#; Lenguaje R; ADA; Máquina; PHP; TypeScript;

Cantidad es: 7

Eliminado el primer elemento
C#; Lenguaje R; ADA; Máquina; PHP; TypeScript;

Eliminado el último elemento
C#; Lenguaje R; ADA; Máquina; PHP;
```

Ilustración 55: Objetos en LinkedList

## Bibliografía

- [1] Wikipedia, «GNU Lesser General Public License,» 2017. [En línea]. Available: [https://es.wikipedia.org/wiki/GNU\\_Lesser\\_General\\_Public\\_License](https://es.wikipedia.org/wiki/GNU_Lesser_General_Public_License). [Último acceso: mayo 2020].
- [2] GeeksforGeeks, «C# | ArrayList Class,» 03 abril 2019. [En línea]. Available: <https://www.geeksforgeeks.org/c-sharp-arraylist-class/>. [Último acceso: 07 enero 2021].
- [3] TutorialsTeacher, «C# - List<T>,» 2020. [En línea]. Available: <https://www.tutorialsteacher.com/csharp/csharp-list>. [Último acceso: 07 enero 2021].
- [4] Microsoft, «Dictionary<TKey,TValue> Clase,» 2021. [En línea]. Available: <https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.dictionary-2?view=net-5.0>. [Último acceso: 07 enero 2021].
- [5] Guru 99, «C# Queue with Examples,» 2021. [En línea]. Available: <https://www.guru99.com/c-sharp-queue.html>. [Último acceso: 07 enero 2021].
- [6] csharp.com.es, «Tutorial de C#,» 2021. [En línea]. Available: <https://csharp.com.es/la-pila-en-c-lifo-la-cola-lilo/>. [Último acceso: 07 enero 2021].
- [7] TutorialsPoint, «C# - Hashtable Class,» 2021. [En línea]. Available: [https://www.tutorialspoint.com/csharp/csharp\\_hashtable.htm](https://www.tutorialspoint.com/csharp/csharp_hashtable.htm). [Último acceso: 07 enero 2021].
- [8] dot net perls, «C# SortedList,» 2021. [En línea]. Available: <https://www.dotnetperls.com/sortedlist>. [Último acceso: 07 enero 2021].
- [9] A. Sharma, «Implementing Linked List In C#,» 22 septiembre 2019. [En línea]. Available: <https://www.c-sharpcorner.com/article/linked-list-implementation-in-c-sharp/>. [Último acceso: 07 enero 2021].