C# Y .NET 10
Parte 5. Estructuras de datos dinámicas

2026-01

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones

Acerca del autor

Rafael Alberto Moreno Parra

ramsoftware@gmail.com o enginelife@hotmail.com

Sitio Web: http://darwin.50webs.com (dedicado a la investigación de algoritmos evolutivos y vida artificial).

Github: https://github.com/ramsoftware

Youtube: https://www.youtube.com/@RafaelMorenoP

Licencia de este libro

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 2026 ® Enlace: https://visualstudio.microsoft.com/es/vs/

Dedicatoria

En recuerdo de Michu

Introducción

En este libro se estudiarán las estructuras de datos implementadas en .NET 10 y su uso en C#. Algunas estructuras no son documentadas como ArrayList o Hashtable porque ya han sido reemplazadas por mejores como List o Dictionary.

List

Adicionar, tamaño, buscar e imprimir

En C# está el List, es un contenedor de objetos de un tipo definido

E/001.cs

```
namespace Ejemplo;

internal class Program {
   static void Main(string[] args) {
      //Declara la lista que almacenará cadenas
      List<string> ListaAnimales =
      [
         //Inicia con elementos
         "Ballena",
         "Tortuga marina",
         "Tiburón",
         "Estrella de mar",
         "Hipocampo",
         "Serpiente marina",
         "Delfín",
         "Pulpo",
         "Pingüinos",
         "Calamar",
      ];

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

      //Traer un determinado elemento de la lista
      int posicion = 5;
      string texto = ListaAnimales[posicion].ToString();
      Console.WriteLine("\r\nEn posición: " + posicion + " es: " + texto);

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

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

      //Agrega elementos a la lista
```


Autor: Rafael Alberto Moreno Parra

```
        ListaAnimales.Add("León Marino");
        ListaAnimales.Add("Foca");
        ListaAnimales.Add("Langostino");

        //Imprime la lista
        Console.WriteLine("\r\nLista completa es:");
        for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
           Console.Write(ListaAnimales[Cont] + ";");

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

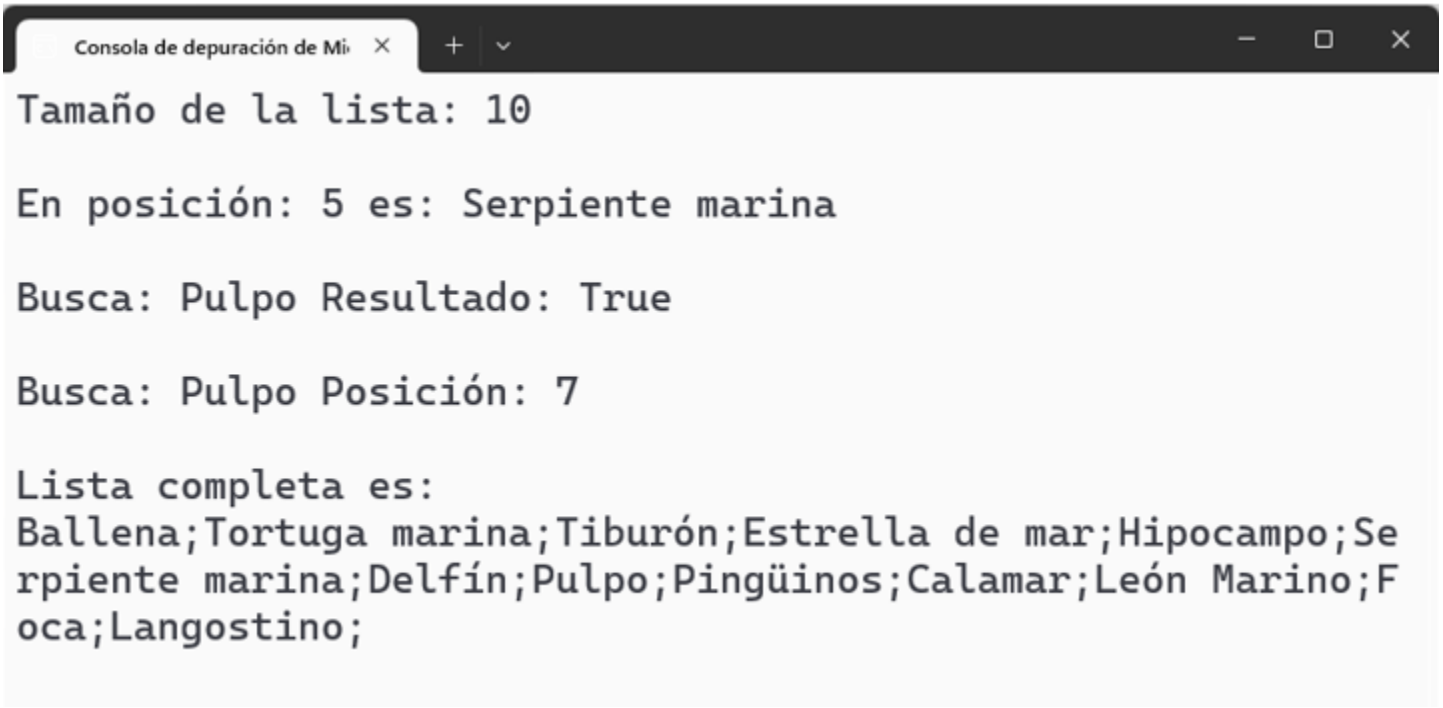


Ilustración 1: List Adicionar, tamaño, buscar e imprimir

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

Borrar elemento

E/002.cs

```
namespace Ejemplo;

internal class Program {
   static void Main(string[] args) {
      //Declara la lista que almacenará cadenas
      List<string> ListaAnimales =
      [
         //Inicia con elementos
         "Ballena",
         "Tortuga marina",
         "Tiburón",
         "Estrella de mar",
         "Hipocampo",
         "Serpiente marina",
         "Delfín",
         "Pulpo",
         "Pingüinos",
         "Calamar",
      ];

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

      //Traer un determinado elemento de la lista
      int posicion = 5;
      string texto = ListaAnimales[posicion].ToString();
      Console.WriteLine("\r\nEn posición: " + posicion + " es: " + texto);

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

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

      //Agrega elementos a la lista
      ListaAnimales.Add("León Marino");
      ListaAnimales.Add("Foca");
      ListaAnimales.Add("Langostino");

      //Imprime la lista
```

```
      Console.WriteLine("\r\nLista completa es:");
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");

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

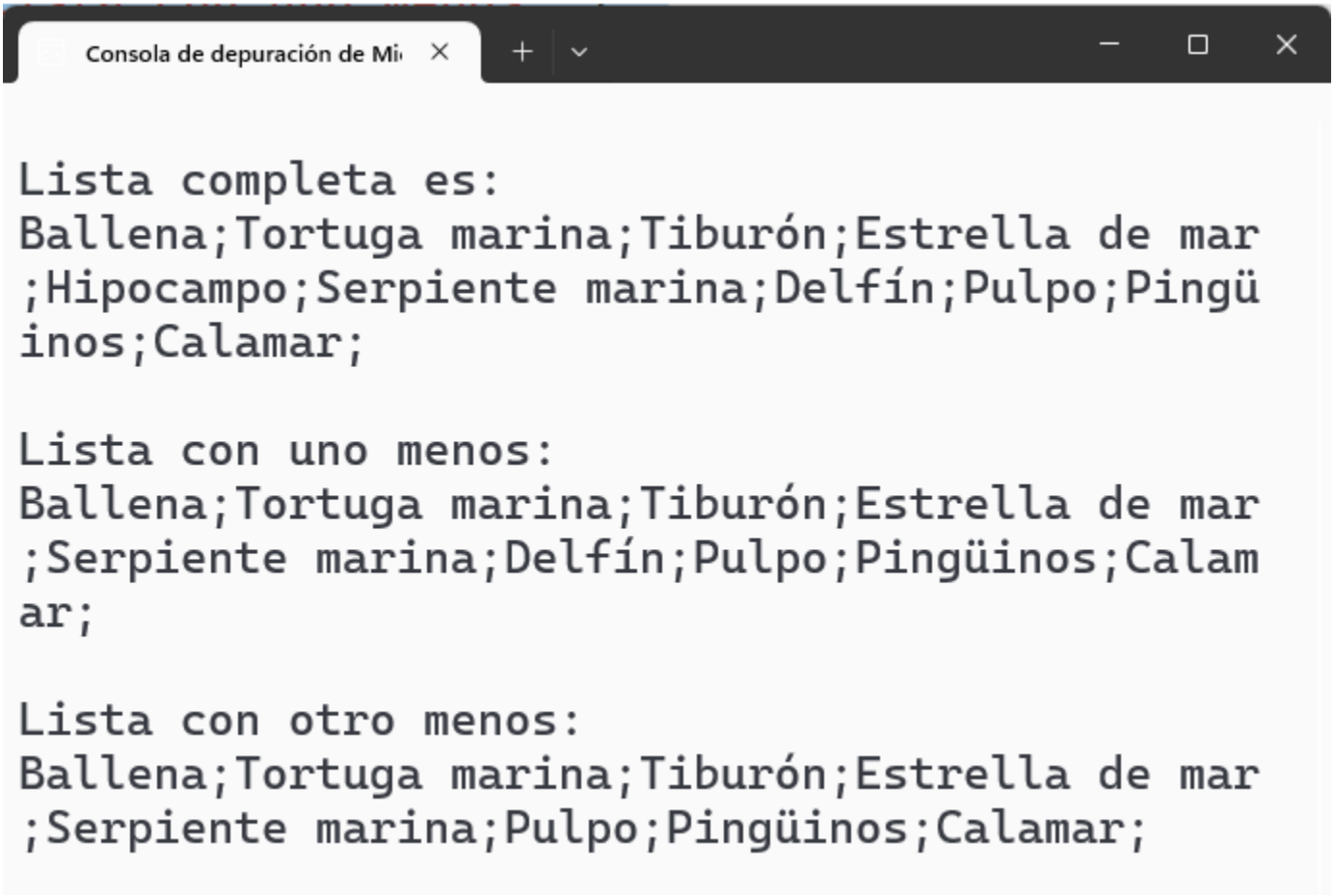


Ilustración 2: List, borrar elemento

Dos técnicas para eliminar elementos de un List, 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.

E/003.cs

```
namespace Ejemplo;

internal class Program {
   static void Main(string[] args) {
      //Declara la lista que almacenará cadenas
      List<string> ListaAnimales =
      [
         //Inicia con elementos
         "Ballena",
         "Tortuga marina",
         "Tiburón",
         "Estrella de mar",
         "Hipocampo",
         "Serpiente marina",
         "Delfín",
         "Pulpo",
         "Pingüinos",
         "Calamar",
      ];

      //Imprime la lista
      Console.WriteLine("\r\nLista completa es:");
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");

      //Cambia una cadena en la lista
      ListaAnimales[3] = "CACATÚA";

      //Imprime la lista
      Console.WriteLine("\r\n\r\nLista con uno editado:");
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");

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

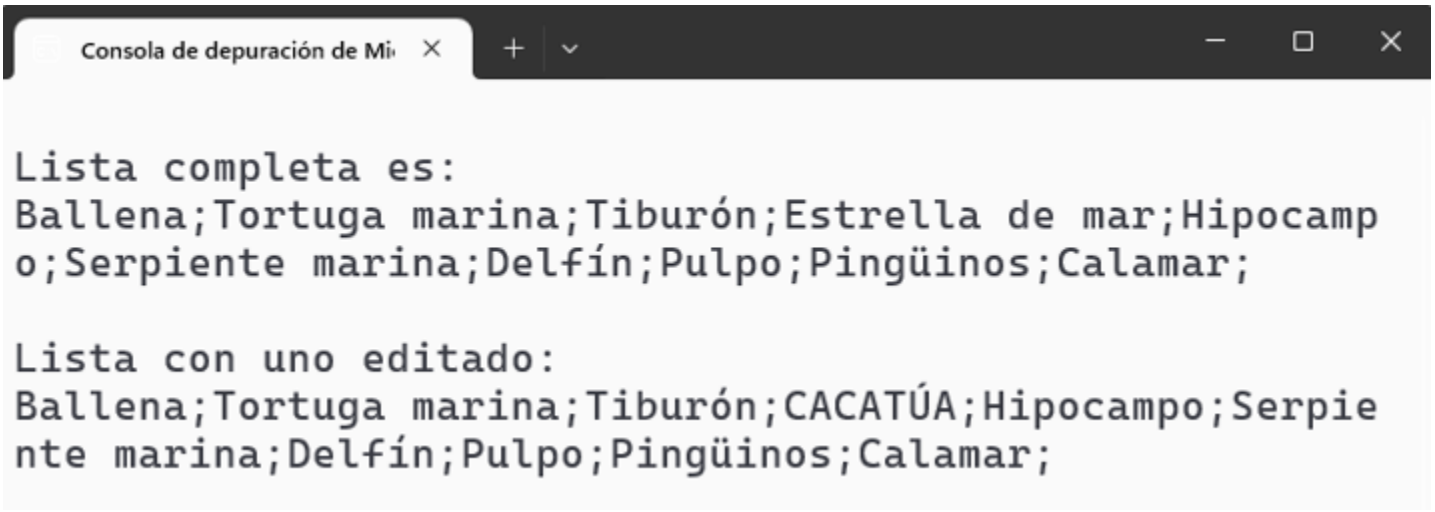


Ilustración 3: List, cambiar elemento

Insertar Elemento

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

E/004.cs

```
namespace Ejemplo;

internal class Program {
   static void Main(string[] args) {
      //Declara la lista que almacenará cadenas
      List<string> ListaAnimales =
      [
         //Inicia con elementos
         "Ballena",
         "Tortuga marina",
         "Tiburón",
         "Estrella de mar",
         "Hipocampo",
         "Serpiente marina",
         "Delfín",
         "Pulpo",
         "Pingüinos",
         "Calamar",
      ];

      //Imprime la lista
      Console.WriteLine("\r\nLista completa es:");
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");

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

      //Imprime la lista
      Console.WriteLine("\r\n\r\nLista con uno editado:");
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");

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

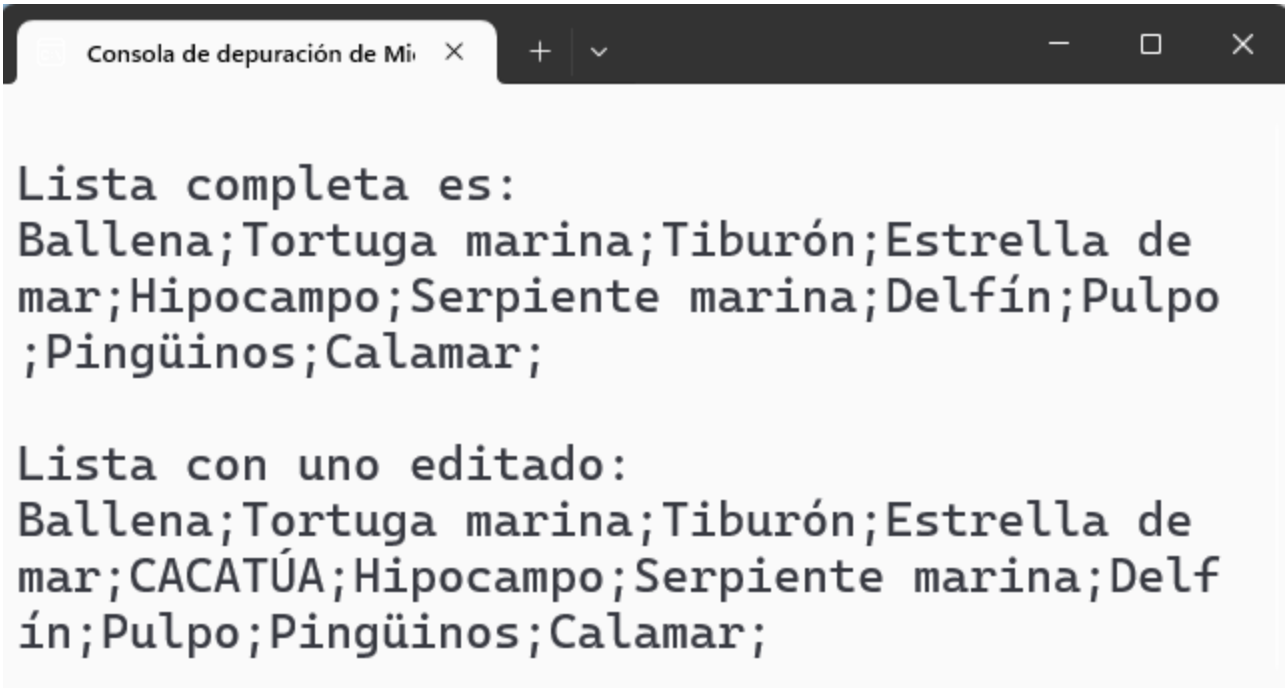


Ilustración 4: List, insertar elemento

SubLista

List<T>.GetRange crea una nueva lista (una copia de ese segmento si son tipos de datos nativos como string, int, char, double pero no hace copia si es de objetos).

E/005.cs

```
namespace Ejemplo;

internal class Program {
   static void Main(string[] args) {
      //Declara la lista que almacenará cadenas
      List<string> ListaAnimales =
      [
         //Inicia con elementos
         "Ballena",
         "Tortuga marina",
         "Tiburón",
         "Estrella de mar",
         "Hipocampo",
         "Serpiente marina",
         "Delfín",
         "Pulpo",
         "Pingüinos",
         "Calamar",
      ];

      //Imprime de nuevo la lista original
      Console.WriteLine("\r\n\r\nLista original");
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");

      //Genera nueva lista
      int posIni = 5;
      int cantidad = 3;
      List<string> nuevaLista = ListaAnimales.GetRange(posIni, cantidad);

      //Imprime valores de esa nueva lista
      Console.WriteLine("\r\n\r\nNueva lista");
      for (int Cont = 0; Cont < nuevaLista.Count; Cont++)
         Console.Write(nuevaLista[Cont] + ";");

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

      //Imprime la lista nueva con el valor alterado
      Console.WriteLine("\r\n\r\nNueva lista, primer valor editado:");
      for (int Cont = 0; Cont < nuevaLista.Count; Cont++)
```

```
            Console.Write(nuevaLista[Cont] + ";");

        //Imprime de nuevo la lista original
        Console.WriteLine("\r\n\r\nLista original");
        for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
            Console.Write(ListaAnimales[Cont] + ";");

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

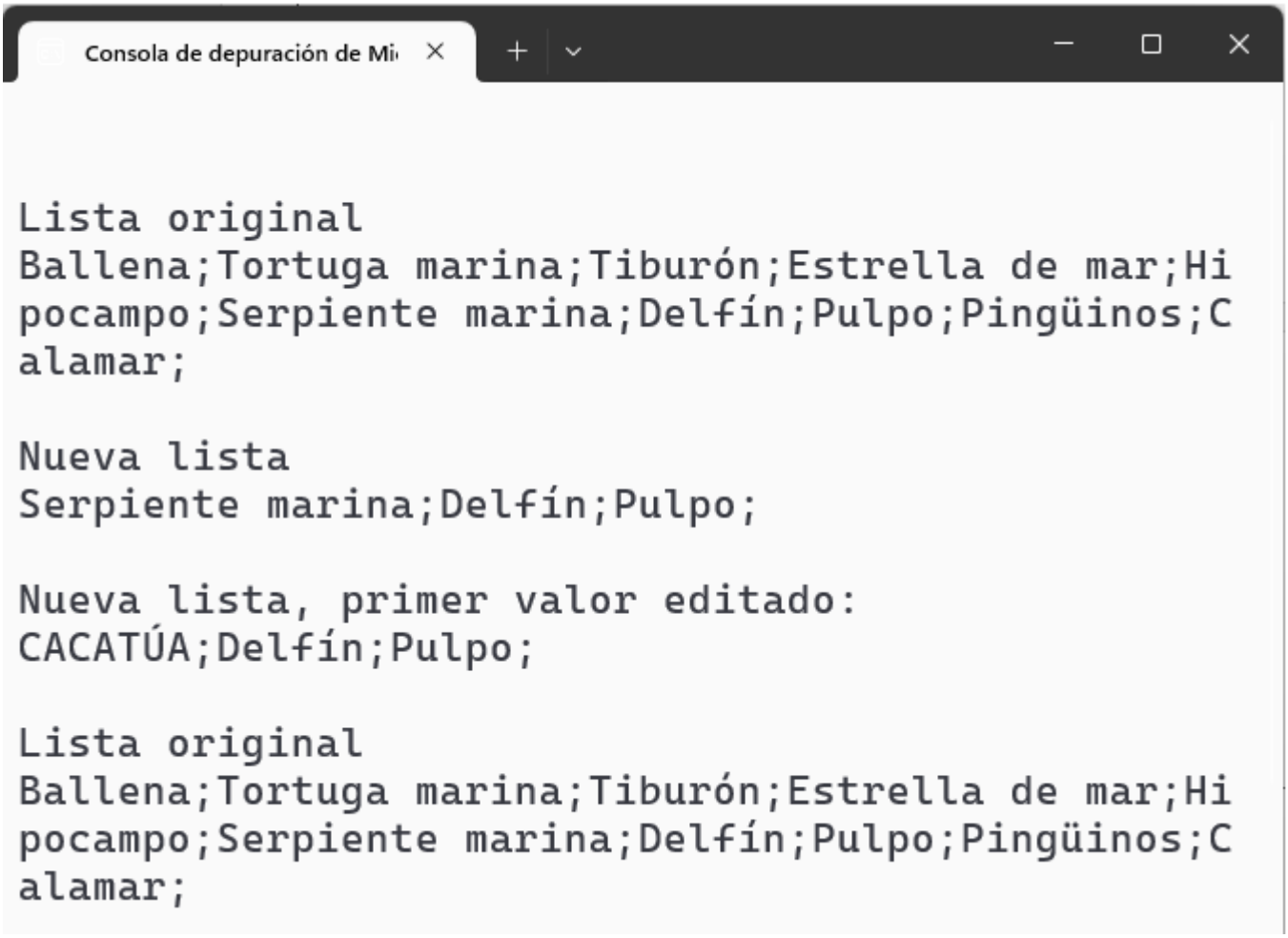


Ilustración 5: List, referenciar con una variable, un rango de la lista

Tres formas de recorrer un List

E/006.cs

```
namespace Ejemplo;

internal class Program {
   static void Main(string[] args) {
      //Declara la lista que almacenará cadenas
      List<string> ListaAnimales =
      [
         //Inicia con elementos
         "Ballena",
         "Tortuga marina",
         "Tiburón",
         "Estrella de mar",
         "Hipocampo",
         "Serpiente marina",
         "Delfín",
         "Pulpo",
         "Pingüinos",
         "Calamar",
      ];

      //Recorrido con foreach
      Console.WriteLine("Recorrido con foreach");
      foreach (Object objeto in ListaAnimales)
         Console.Write(objeto + ";");

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

      //Recorrido con un IEnumerator
      Console.WriteLine("\r\n\r\nRecorrido con un IEnumerator");
      List<string>.Enumerator elemento = ListaAnimales.GetEnumerator();
      while (elemento.MoveNext())
         Console.Write(elemento.Current + ";");

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


Autor: Rafael Alberto Moreno Parra

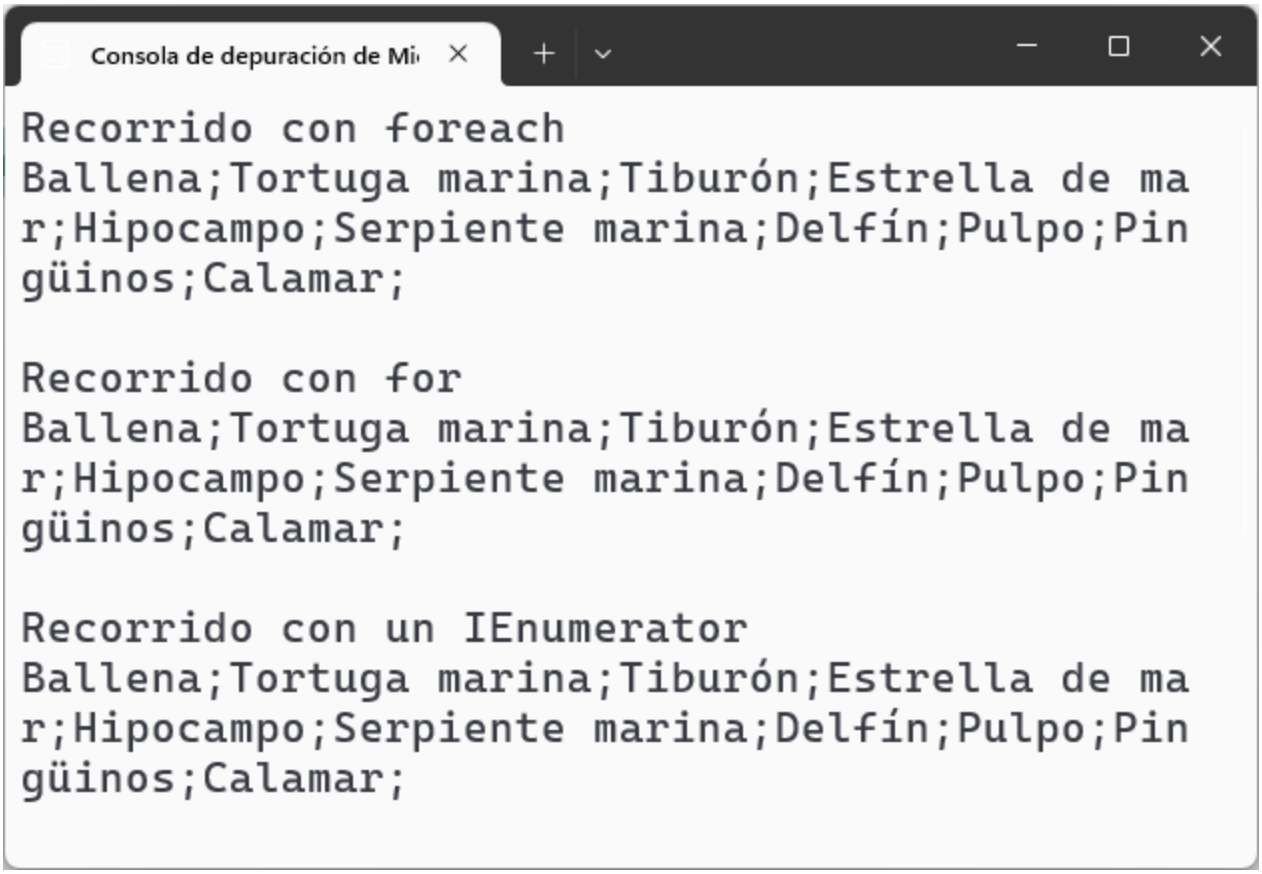


Ilustración 6: List, tres formas de recorrerlo

Borrar completamente un List

Se utiliza el método Clear()

E/007.cs

```
namespace Ejemplo;

internal class Program {
   static void Main(string[] args) {
      //Declara la lista que almacenará cadenas
      List<string> ListaAnimales =
      [
         //Inicia con elementos
         "Ballena",
         "Tortuga marina",
         "Tiburón",
         "Estrella de mar",
         "Hipocampo",
         "Serpiente marina",
         "Delfín",
         "Pulpo",
         "Pingüinos",
         "Calamar",
      ];

      //Limpia el List
      Console.WriteLine("(Antes) Elementos: " + ListaAnimales.Count);
      ListaAnimales.Clear();
      Console.WriteLine("(Después) Elementos: " + ListaAnimales.Count);

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

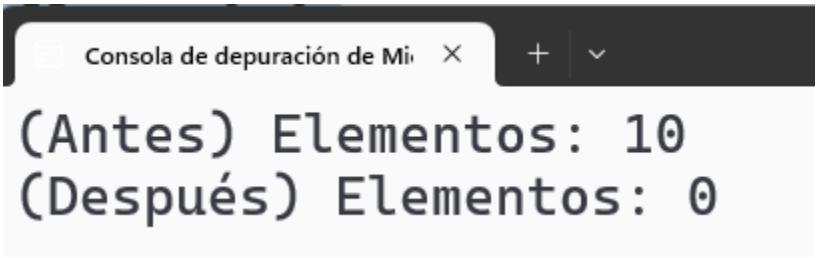


Ilustración 7: Borrar completamente un List

Borrar un rango en un List

Se utiliza el método RemoveRange()

E/008.cs

```
namespace Ejemplo;

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

      //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");

      //Elimina un rango de elementos del List
      Console.WriteLine("Antes");
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");
      Console.WriteLine(" ");

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

      Console.WriteLine("\r\nDespués");
      //Imprime valores
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");
      Console.WriteLine(" ");
   }
}
```

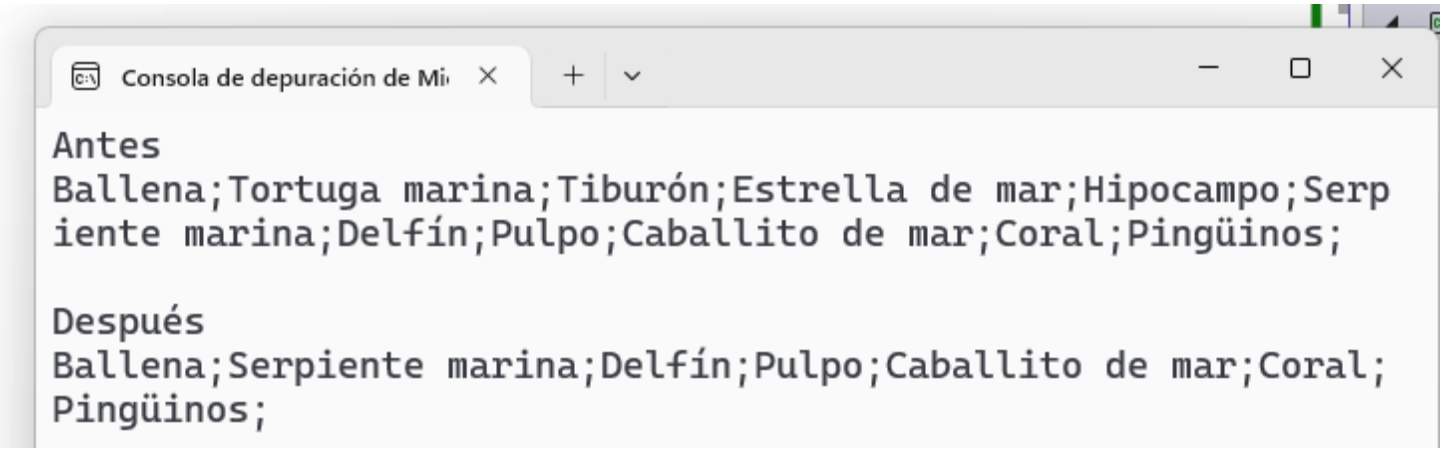


Ilustración 8: Borrar un rango en un List

Guardar el List en un arreglo estático

Todos los elementos del List pasan a un arreglo estático, sea de tipo object() o de string.

E/009.cs

```
namespace Ejemplo;

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

      //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");

      //Guarda el List en un arreglo estático
      //de tipo objeto
      Console.WriteLine("Arreglo estático de tipo objeto");
      object[] arregloEstatico = [.. ListaAnimales];
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");
      Console.WriteLine(" ");

      //Guarda el List en un arreglo estático de tipo string
      Console.WriteLine("Arreglo estático de tipo cadena");
      string[] Cadenas = [.. ListaAnimales];
      for (int cont = 0; cont < Cadenas.Length; cont++)
         Console.Write(Cadenas[cont] + ";");
   }
}
```


Autor: Rafael Alberto Moreno Parra

Agregar un arreglo estático a un List

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

E/010.cs

```
namespace Ejemplo;

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

      //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");

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

      //Adiciona el arreglo estático al List
      ListaAnimales.AddRange(Cadenas);

      //Imprime el List
      for (int cont = 0; cont < ListaAnimales.Count; cont++)
         Console.Write(ListaAnimales[cont] + "; ");
   }
}
```


Autor: Rafael Alberto Moreno Parra

Inserta un arreglo estático en una determinada posición del List

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

E/011.cs

```
namespace Ejemplo;

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

      //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");

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

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

      //Imprime el List
      for (int cont = 0; cont < ListaAnimales.Count; cont++)
         Console.Write(ListaAnimales[cont] + "; ");
   }
}
```


Autor: Rafael Alberto Moreno Parra

Insertar varios List dentro de un List

El contenido de varios List es copiado dentro de otro List. Como es una copia, no importa si se modifica el List fuente.

E/012.cs

```
namespace Ejemplo;

class Program {
   public static void Main() {
      //Declara tres List
      List<string> ListaA = [];
      List<string> ListaB = [];
      List<string> ListaC = [];

      //Adiciona elementos a esos List
      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 List en el tercero
      int posicionInserta = 1;
      ListaC.InsertRange(posicionInserta, ListaA);

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

      //Imprime el List
      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] = "CACATÚA";
      for (int cont = 0; cont < ListaC.Count; cont++)
         Console.Write(ListaC[cont] + "; ");
   }
}
```

Invertir un List

E/013.cs

```
namespace Ejemplo;

class Program {
   public static void Main() {
      //Declara la lista que almacenará cadenas
      List<string> Listado = [];

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

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

      //Aplica Reverse()
      Listado.Reverse();

      //Imprime de nuevo el List
      for (int cont = 0; cont < Listado.Count; cont++)
         Console.Write(Listado[cont] + "; ");
   }
}
```

Invertir un rango de datos en el List

E/014.cs

```
namespace Ejemplo;

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

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

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

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

      //Imprime de nuevo el List
      for (int cont = 0; cont < Listado.Count; cont++)
         Console.Write(Listado[cont] + "; ");
   }
}
```


Autor: Rafael Alberto Moreno Parra

Ordena un List

E/015.cs

```
namespace Ejemplo;

class Program {
   public static void Main() {
      //Declara la lista que almacenará cadenas
      List<string> Listado = [];

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

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

      //Ordena el List
      Listado.Sort();

      //Imprime de nuevo el List
      for (int cont = 0; cont < Listado.Count; cont++)
         Console.Write(Listado[cont] + "; ");
   }
}
```

Búsqueda Binaria

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

E/016.cs

```
namespace Ejemplo;

class Program {
   public static void Main() {
      //Declara la lista que almacenará cadenas
      List<string> Listado = [];

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

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

      //Ordena el List
      Listado.Sort();

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

      //Busca en forma binaria en el List
      string Buscar = "KL";
      int pos = Listado.BinarySearch(Buscar);
      Console.WriteLine("Buscando: " + Buscar);
      Console.WriteLine("Encontrado en: " + pos);
   }
}
```

Capacidad del List

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

E/017.cs

```
namespace Ejemplo;

class Program {
   public static void Main() {
      //Declara la lista que almacenará cadenas
      List<double> Listado = new();

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

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

```
Iteración: 1
Tamaño: 0
Capacidad: 0

Iteración: 2
Tamaño: 30
Capacidad: 32

Iteración: 3
Tamaño: 60
Capacidad: 64

Iteración: 4
Tamaño: 90
Capacidad: 128

Iteración: 5
Tamaño: 120
Capacidad: 128

Iteración: 6
Tamaño: 150
Capacidad: 256
```

Ilustración 9: Capacidad del List

Métricas: Comparativa de desempeño de List vs Arreglo estático vs ArrayList

La estructura ArrayList es antigua y obsoleta que permite guardar todo tipo de dato, fue reemplazada por estructuras modernas como List. Una de las razones del no uso de ArrayList es su lentitud. Para demostrarlo se hace una comparativa de desempeño, se utilizó el algoritmo de ordenación de burbuja, el cual 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:

E/018.cs

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

namespace Ejemplo;

class Program {
   static void Main() {
#if DEBUG
      Console.WriteLine("Modo DEBUG detectado. Las pruebas se deben hacer
en RELEASE");
      Environment.Exit(0);
#endif

      /* 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 minOrden = 500; //Mínimo número de elementos a ordenar
      int maxOrden = 9000; //Máximo número de elementos a ordenar
      int avanceOrden = 500; //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 = 40;

      //Limite es el tamaño de datos que se van a ordenar
      Console.WriteLine(".NET Versión: " + Environment.Version);
      Console.WriteLine("Ordenación. Tiempo promedio en milisegundos");
```

```
      Console.WriteLine("Elementos;Arreglo;ArrayList;List");
      for (int Lim = minOrden; Lim <= maxOrden; Lim += avanceOrden)
         Ordenamiento(Lim, numPruebas);

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

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

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

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

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

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

         //Ordenación por Burbuja Arreglo estático
         Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
         temporizador.Reset();
```

```
            temporizador.Start();
            BurbujaArreglo(numerosB);
            TParreglo += temporizador.ElapsedMilliseconds;

            //Compara las listas ordenadas
            for (int cont = 0; cont < numerosB.Length; cont++) {
               if (numerosB[cont] != list[cont] ||
                  list[cont] != Convert.ToInt32(arraylist[cont]))
                  Console.WriteLine("Error en la ordenación");
            }
         }

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

         Console.Write(Limite + ";" + Tarreglo);
         Console.Write(";" + Tarraylist);
         Console.WriteLine(";" + Tlist);
      }

      //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++) {
               int iA = Convert.ToInt32(arraylist[j]);
               int iB = Convert.ToInt32(arraylist[j + 1]);
               if (iA > iB) {
                  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;
            }
         }
      }
   }
}
```

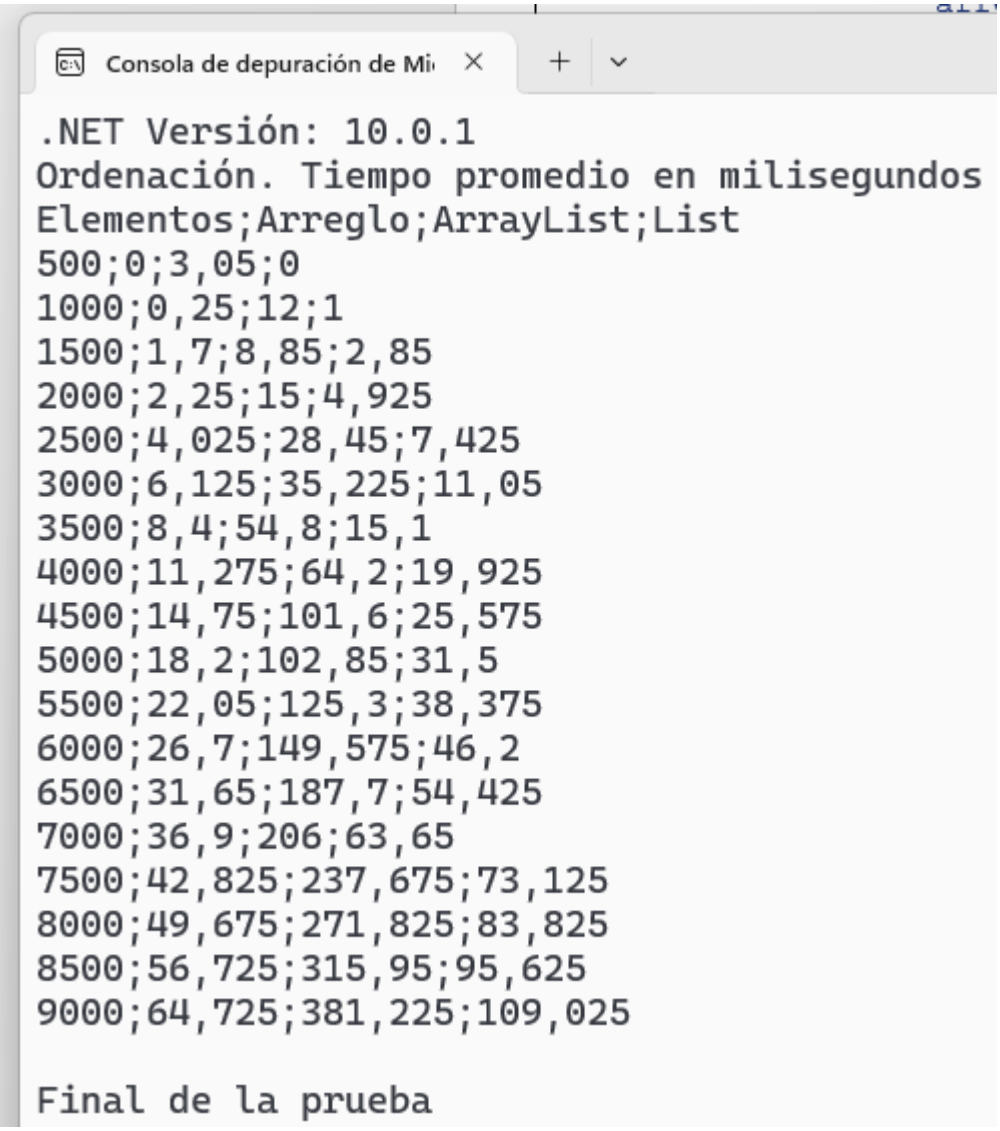
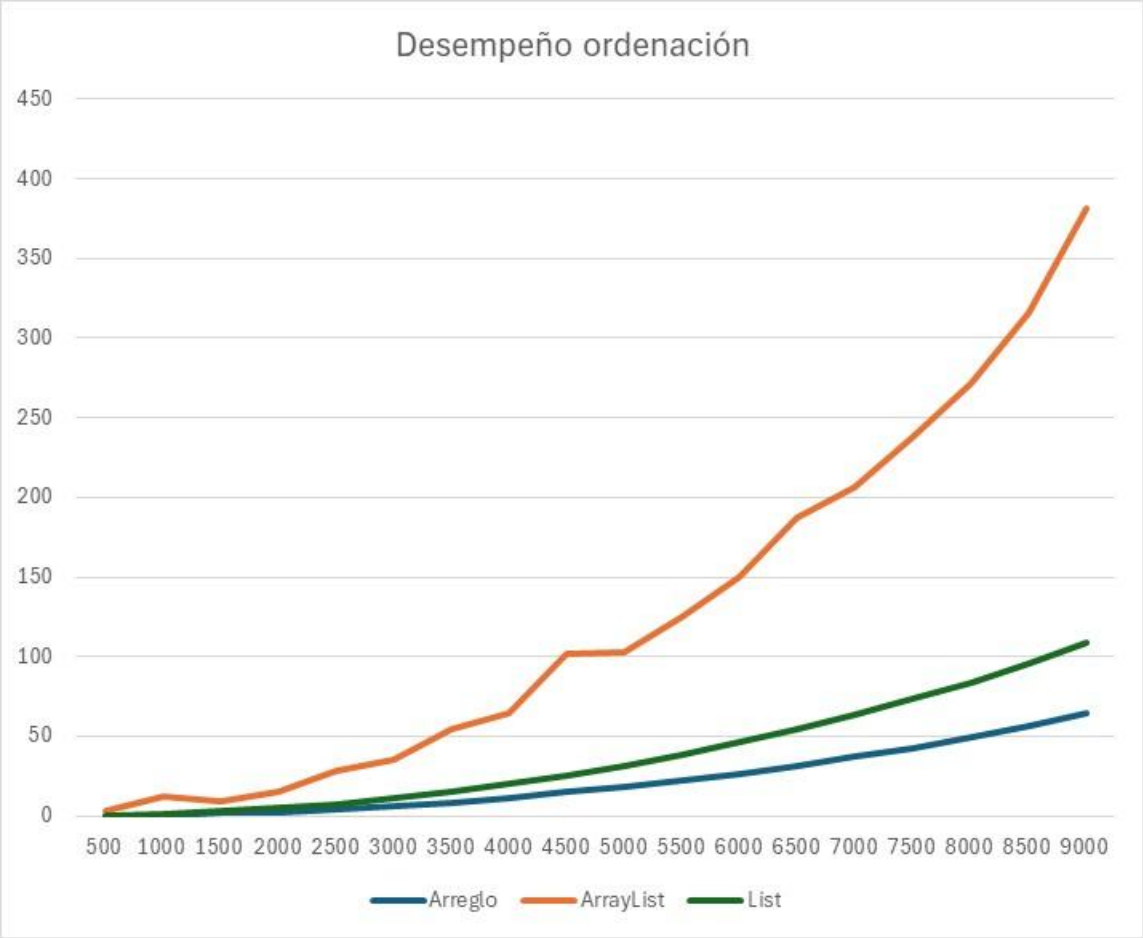


Ilustración 10: Comparativa



Los arreglos unidimensionales estáticos mostraron mejores tiempos. La estructura dinámica List es ligeramente más lenta y ArrayList es notablemente más lenta.

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 y en la otra se crean, adicionan, actualizan y borran del List.

E/019.cs

```
namespace Ejemplo;

class MiClase {
   //Atributos variados
   public int Entero { get; set; }
   public double Num { get; set; }
   public char Car { get; set; }
   public string Cad { get; set; }

   //Constructor
   public MiClase(int Entero, double Num, char Car, string Cad) {
      this.Entero = Entero;
      this.Num = Num;
      this.Car = Car;
      this.Cad = Cad;
   }

   //Imprime los valores
   public void Imprime() {
      Console.WriteLine("\r\nEntero: " + Entero);
      Console.WriteLine("Real: " + Num);
      Console.WriteLine("Caracter: " + Car);
      Console.WriteLine("Cadena: [" + Cad + "]");
   }
}

class Program {
   static void Main() {
      List<MiClase> Listado = [];

      //Adiciona objetos a la lista
      Listado.Add(new MiClase(16, 83.29, 'R', "Ruiseñor"));
      Listado.Add(new MiClase(29, 89.7, 'A', "Águila"));
      Listado.Add(new MiClase(2, 80.19, 'M', "Manatí"));
      Listado.Add(new MiClase(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 MiClase(88, 3.33, 'Z', "QQQQQ"));

      //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");
   }
}
```

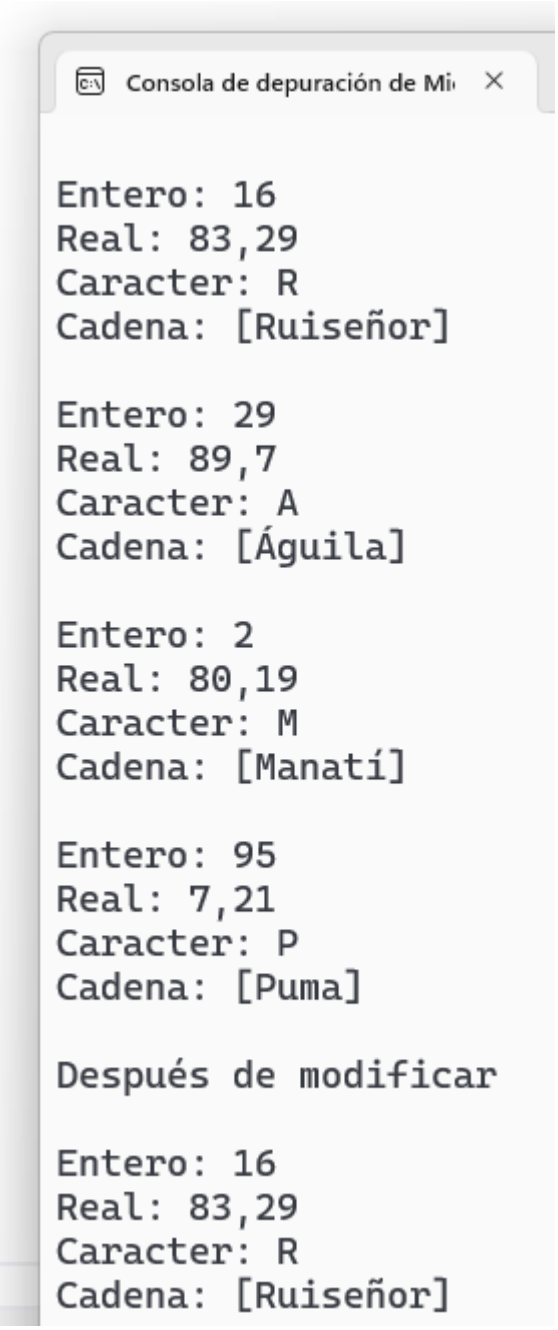


Ilustración 11: Lista de objetos

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.

E/020.cs

```
namespace Ejemplo;


//Datos del actor o actriz
class ActorActriz {
   public int Codigo { get; set; }
   public string Nombre { get; set; }
   public string URLIMDB { get; set; }

   //Constructor
   public ActorActriz(int Codigo, string Nombre, string URLIMDB) {
      this.Codigo = Codigo;
      this.Nombre = Nombre;
      this.URLIMDB = "https://www.imdb.com/name/" + URLIMDB;
   }
}

//Datos de la serie de televisión
class Serie {
   public int Codigo { get; set; }
   public string Nombre { get; set; }
   public string URLIMDB { get; set; }

   //Listado de códigos de actores que actúan en la serie
   public List<int> Actor = [];

   //Constructor
   public Serie(int Codigo, string Nombre, string URLIMDB) {
      this.Codigo = Codigo;
      this.Nombre = Nombre;
      this.URLIMDB = "https://www.imdb.com/title/" + URLIMDB;
   }
}

//La parte que simula la capa de persistencia
class Persistencia {
```

```
public List<ActorActriz> Actores;
public List<Serie> Series;

//Carga datos de prueba
public Persistencia() {
   Actores = [];
   Series = [];

   //Un listado de actores y actrices
   ActorAdiciona(78, "Ana María Orozco", "nm0650450");
   ActorAdiciona(81, "Laura Londoño", "nm2256810");
   ActorAdiciona(84, "Carolina Ramírez", "nm1329835");
   ActorAdiciona(93, "Catherine Siachoque", "nm0796171");
   ActorAdiciona(98, "Carmenza González", "nm1863990");
   ActorAdiciona(99, "Andrés Londoño", "nm2150265");

   //Un listado de series
   Series.Add(new Serie(16, "Yo soy Betty, la fea", "tt0233127"));
   Series.Add(new Serie(43, "La reina del flow", "tt8560918"));
   Series.Add(new Serie(60, "Café con Aroma de Mujer", "tt14471346"));
   Series.Add(new Serie(62, "Los Briceño", "tt10348478"));
   Series.Add(new Serie(70, "Distrito Salvaje", "tt8105958"));
   Series.Add(new Serie(72, "Mil Colmillos", "tt9701670"));
   Series.Add(new Serie(83, "Perdida", "tt10064124"));

   //Obsérvese que un mismo actor o actriz puede
   //estar en dos series distintas
   Series[0].Actor.Add(78);
   Series[0].Actor.Add(93);
   Series[0].Actor.Add(98);
   Series[6].Actor.Add(78);
   Series[4].Actor.Add(78);
}

//Adicionar actor
public bool ActorAdiciona(int Codigo, string Nombre, string URL) {
   for (int Cont = 0; Cont < Actores.Count; Cont++) {
      if (Actores[Cont].Codigo == Codigo)
         return false;
   }
   Actores.Add(new ActorActriz(Codigo, Nombre, URL));
   return true;
}

//Editar actor
public bool ActorEdita(int CodigoActor, string Nombre, string URL) {
   for (int Cont = 0; Cont < Actores.Count; Cont++) {
      if (Actores[Cont].Codigo == CodigoActor) {
```

```
            Actores[Cont].Nombre = Nombre;
            Actores[Cont].URLIMDB = URL;
            return true;
         }
      }
      return false;
   }

   //Chequea si el actor está trabajando en alguna serie
   public bool ActorEnSerie(int CodigoActor) {
      for (int Cont = 0; Cont < Series.Count; Cont++)
         for (int Num = 0; Num < Series[Cont].Actor.Count; Num++)
            if (Series[Cont].Actor[Num] == CodigoActor)
               return true;
      return false;
   }

   //Borrar actor, si y solo si no está trabajando en alguna serie
   public bool ActorBorra(int CodigoActor) {
      if (ActorEnSerie(CodigoActor) == false) {
         for (int Cont = 0; Cont < Actores.Count; Cont++)
            if (Actores[Cont].Codigo == CodigoActor) {
               Actores.RemoveAt(Cont);
               return true;
            }
      }
      return false;
   }

   //Dado el código, retorna el nombre del actor/actriz
   public string NombreActor(int CodigoActor) {
      for (int cont = 0; cont < Actores.Count; cont++) {
         if (Actores[cont].Codigo == CodigoActor)
            return Actores[cont].Nombre;
      }
      return "N/A";
   }

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

```
    }

    //Adicionar serie
    public bool SerieAdiciona(int CodigoSerie, string Nombre, string URL) {
        for (int Cont = 0; Cont < Series.Count; Cont++) {
            if (Series[Cont].Codigo == CodigoSerie)
                return false;
        }
        Series.Add(new Serie(CodigoSerie, Nombre, URL));
        return true;
    }

    //Editar serie
    public bool SerieEdita(int CodigoSerie, string Nombre, string URL) {
        for (int Cont = 0; Cont < Series.Count; Cont++) {
            if (Series[Cont].Codigo == CodigoSerie) {
                Series[Cont].Nombre = Nombre;
                Series[Cont].URLIMDB = URL;
                return true;
            }
        }
        return false;
    }

    //Borrar serie
    public bool SerieBorra(int CodigoSerie) {
        for (int Cont = 0; Cont < Series.Count; Cont++)
            if (Series[Cont].Codigo == CodigoSerie) {
                Series.RemoveAt(Cont);
                return true;
            }
        return false;
    }


    //Retornar los actores que trabajan en la serie
    public List<string> SerieActores(int CodigoSerie) {
        int Pos = PosSerie(CodigoSerie);
        List<string> Nombres = [];
        for (int Cont = 0; Cont < Series[Pos].Actor.Count; Cont++)
            Nombres.Add("[" + Series[Pos].Actor[Cont] + "] " +
NombreActor(Series[Pos].Actor[Cont]));
        return Nombres;
    }

    //Añade un actor a una serie
    public bool SerieAsocia(int CodigoSerie, int CodigoActor) {
        int PosSerial = PosSerie(CodigoSerie);
```

```
      if (PosSerial >= 0) {
         if (Series[PosSerial].Actor.Contains(CodigoActor) == false) {
            Series[PosSerial].Actor.Add(CodigoActor);
            return true;
         }
         else
            return false;
      }
      return false;
   }

   //Quita un actor de una serie
   public bool SerieDisocia(int CodigoSerie, int CodigoActor) {
      int PosSerial = PosSerie(CodigoSerie);
      if (PosSerial >= 0) {
         if (Series[PosSerial].Actor.Contains(CodigoActor) == true) {
            Series[PosSerial].Actor.Remove(CodigoActor);
            return true;
         }
         else
            return false;
      }
      return false;
   }

   //Dado el código de la serie, retorna su posición
   public int PosSerie(int CodigoSerie) {
      for (int Cont = 0; Cont < Series.Count; Cont++)
         if (Series[Cont].Codigo == CodigoSerie) {
            return Cont;
         }
      return -1;
   }
}

//La parte visual del programa
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("\nSoftware TV Show 1.7 (Marzo de 2025)");
            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("\nSoftware TV Show. Actores/Actrices");
            for (int Cont = 0; Cont < Datos.Actores.Count; Cont++) {
               Console.Write("[" + Datos.Actores[Cont].Codigo + "] ");
               Console.Write(Datos.Actores[Cont].Nombre);
               Console.WriteLine(" URL: " + Datos.Actores[Cont].URLIMDB);
            }
            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("\nSoftware TV Show. Series");
            for (int Cont = 0; Cont < Datos.Series.Count; Cont++) {
```

```
            Console.Write("[" + Datos.Series[Cont].Codigo + "] ");
            Console.Write(Datos.Series[Cont].Nombre);
            Console.WriteLine(" URL: " + Datos.Series[Cont].URLIMDB);
         }
         Console.WriteLine("\n1. Adicionar");
         Console.WriteLine("2. Editar");
         Console.WriteLine("3. Borrar");
         Console.WriteLine("4. Detalles de la serie");
         Console.WriteLine("5. Asociar actor a serie");
         Console.WriteLine("6. Disociar actor 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 al listado");
      Console.Write("¿Código? ");
      int CodigoActor = Convert.ToInt32(Console.ReadLine());
      Console.Write("¿Nombre? ");
      string Nombre = Console.ReadLine();
      Console.Write("¿URL de IMDB? ");
      string URL = Console.ReadLine();
      if (Datos.ActorAdiciona(CodigoActor, Nombre, URL))
         Console.WriteLine("\nActor adicionado.");
      else
         Console.WriteLine("\nError al adicionar el actor. El código ya
existe.");
      Console.ReadKey();
   }

   //Pantalla para editar actores
   public void ActorEdita() {
      Console.WriteLine("\tEditar actor");
      Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
      int CodigoActor = Convert.ToInt32(Console.ReadLine());
      Console.Write("¿Nombre? ");
      string Nombre = Console.ReadLine();
```

```
      Console.Write("¿URL de IMDB? ");
      string URL = Console.ReadLine();
      if (Datos.ActorEdita(CodigoActor, Nombre, URL))
         Console.WriteLine("\nActor editado.");
      else
         Console.WriteLine("\nError al editar el actor");
      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 CodigoActor = Convert.ToInt32(Console.ReadLine());
      if (Datos.ActorBorra(CodigoActor))
         Console.WriteLine("\nActor borrado.");
      else
         Console.WriteLine("\nError al borrar el actor. Código erróneo o el
actor trabaja en series.");
      Console.ReadKey();
   }

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

   //Pantalla para adicionar series
   public void SerieAdiciona() {
      Console.WriteLine("\tAdicionar serie al listado");
      Console.Write("¿Código? ");
      int codigo = Convert.ToInt32(Console.ReadLine());
      Console.Write("¿Nombre? ");
      string nombre = Console.ReadLine();
      Console.Write("¿URL en IMDB? ");
      string url = Console.ReadLine();
      if (Datos.SerieAdiciona(codigo, nombre, url))
         Console.WriteLine("\nSerie adicionada.");
      else
         Console.WriteLine("\nError al adicionar la serie");
```

```
      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();
      if (Datos.SerieEdita(codigo, nombre, url))
         Console.WriteLine("\nSerie editada.");
      else
         Console.WriteLine("\nError al editar la serie");
      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());
      if (Datos.SerieBorra(codigo))
         Console.WriteLine("\nSerie borrada.");
      else
         Console.WriteLine("\nError al borrar la serie.");
      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? Número[ ]: ");
      int CodigoSerie = Convert.ToInt32(Console.ReadLine());
      int Pos = Datos.PosSerie(CodigoSerie);
      if (Pos >= 0) {
         Console.WriteLine("Nombre: " + Datos.Series[Pos].Nombre);
         Console.WriteLine("URL: " + Datos.Series[Pos].URLIMDB);
         Console.WriteLine("Actores");
         ListaActores = Datos.SerieActores(CodigoSerie);
         for (int cont = 0; cont < ListaActores.Count; cont++)
            Console.WriteLine("\t" + ListaActores[cont]);
      }
      else
```

```
            Console.WriteLine("Error en código de la serie");
         Console.WriteLine("\nENTER 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? Número[ ]: ");
         int CodigoSerie = Convert.ToInt32(Console.ReadLine());
         for (int cont = 0; cont < Datos.Actores.Count; cont++) {
            Console.Write("[" + Datos.Actores[cont].Codigo + "] ");
            Console.Write(Datos.Actores[cont].Nombre);
            Console.WriteLine(" URL: " + Datos.Actores[cont].URLIMDB);
         }
         Console.Write("¿Cuál Actor? Número[ ]: ");
         int CodigoActor = Convert.ToInt32(Console.ReadLine());
         if (Datos.SerieAsocia(CodigoSerie, CodigoActor))
            Console.WriteLine("\nActor asociado a la serie.");
         else
            Console.WriteLine("\nError código del actor o ya estaba asociado a
la serie");
         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? Número[ ]: ");
         int CodigoSerie = Convert.ToInt32(Console.ReadLine());
         int Pos = Datos.PosSerie(CodigoSerie);
         if (Pos >= 0) {
            ListaActores = Datos.SerieActores(CodigoSerie);
            for (int cont = 0; cont < ListaActores.Count; cont++)
               Console.WriteLine(ListaActores[cont]);

            Console.Write("¿Cuál actor quiere quitar? Número[ ]: ");
            int CodigoActor = Convert.ToInt32(Console.ReadLine());

            if (Datos.SerieDisocia(CodigoSerie, CodigoActor) == true)
               Console.WriteLine("\nActor retirado de la serie.");
            else
               Console.WriteLine("\nError retirando actor de la serie");
         }
         else
            Console.WriteLine("Error en el código de la serie");
```

```
      Console.ReadKey();
   }

}

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

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

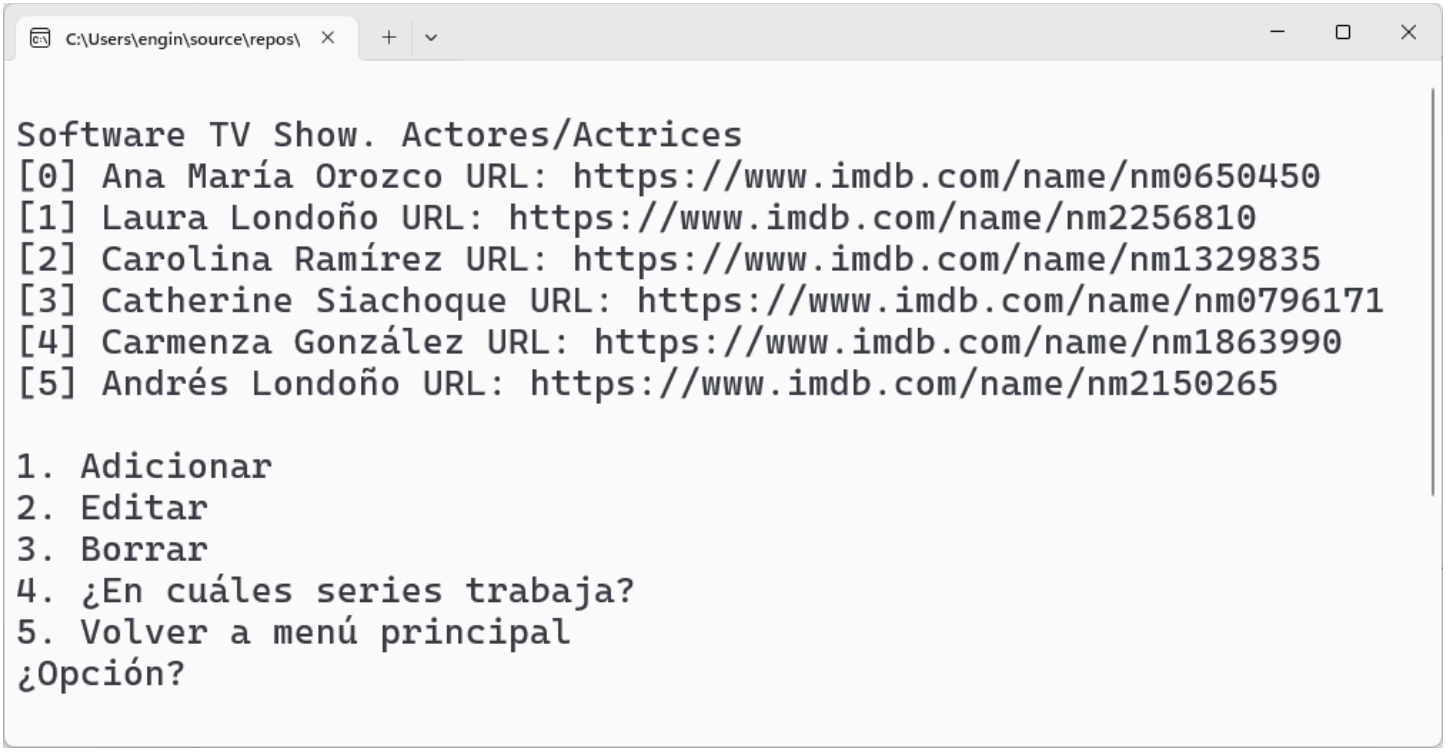


Ilustración 12: Uso de listas para simular un sistema de información

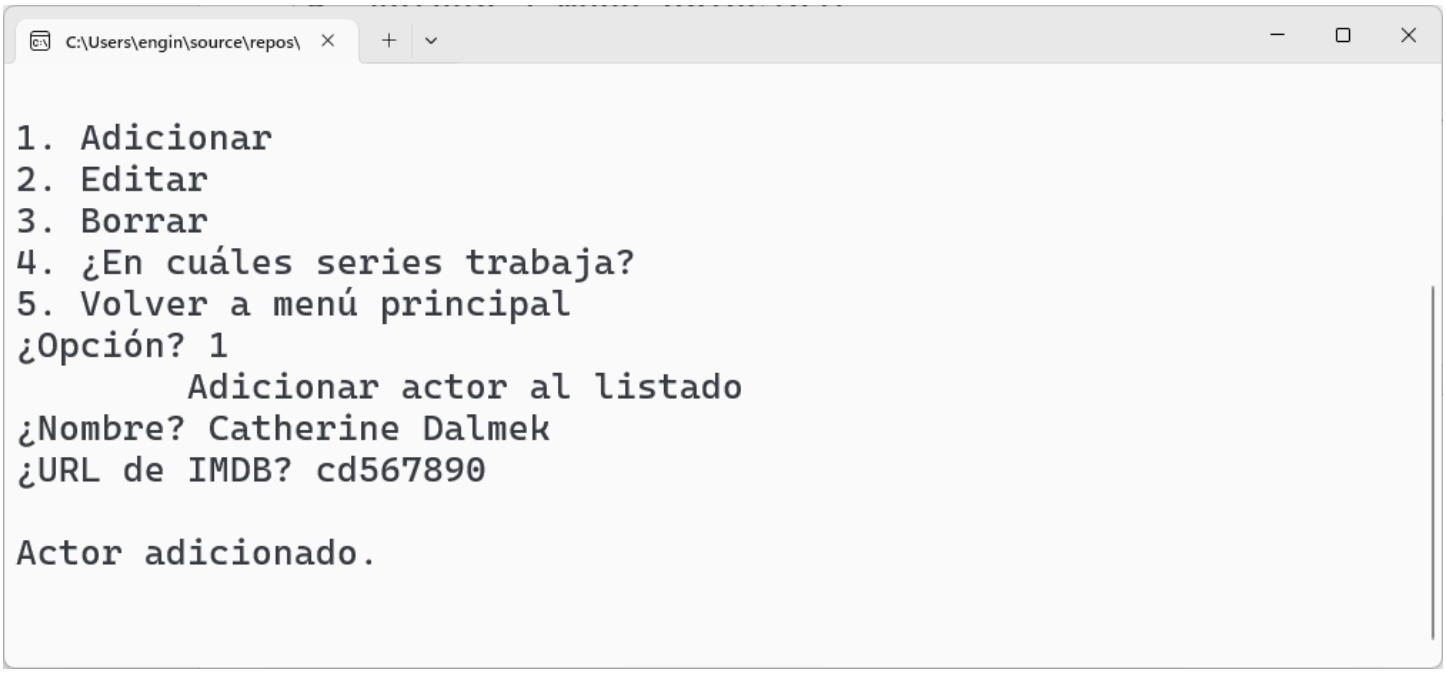


Ilustración 13: Uso de listas para simular un sistema de información

Dictionary

Uso de llaves

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

E/021.cs

```
namespace Ejemplo;

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

      //Se define un diccionario: llave, cadena
      //En este caso la llave es un número entero
      Dictionary<int, string> Animales = new() {
         {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.Write("Llave: " + Llave);
         Console.WriteLine(" cadena: " + Animales[Llave]);
      }
```

```
    }
}
```

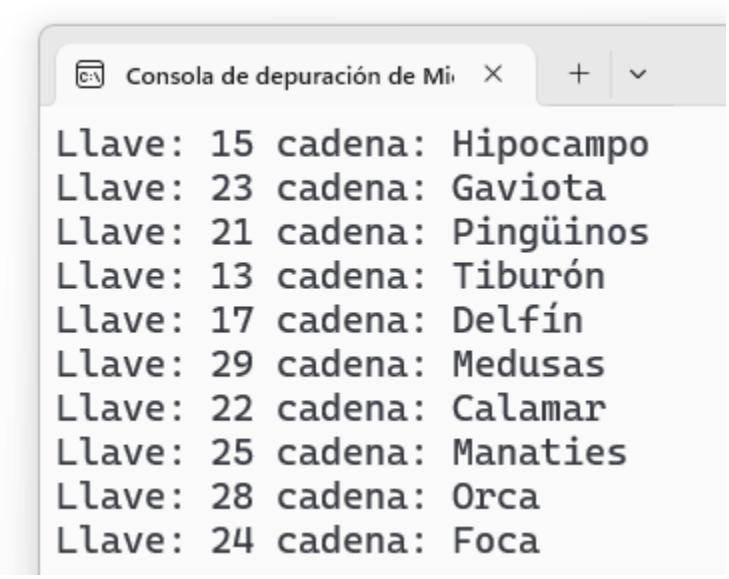


Ilustración 14: Dictionary

Llaves tipo string

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

E/022.cs

```
namespace Ejemplo;

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

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

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

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

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

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

```
Llave: cs valor es: C#
Tamaño: 9
Después de eliminar: 8
```

Ilustración 15: Dictionary

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:

E/023.cs

```
namespace Ejemplo;

//Una clase con varios atributos
class MiClase {
   public int Numero { get; set; }
   public double Valor { get; set; }
   public char Car { get; set; }
   public string Cad { get; set; }

   public MiClase(int Numero, double Valor, char Car, string Cad) {
      this.Numero = Numero;
      this.Valor = Valor;
      this.Car = Car;
      this.Cad = Cad;
   }
}

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

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

      Console.Write("Llave: " + Llave);
      Console.WriteLine(" atributo es: " + Objetos[Llave].Numero);

      Console.Write("Llave: " + Llave);
      Console.WriteLine(" 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"].Cad);
        }
        else {
            Console.WriteLine("No existe esa llave");
        }
    }
}
```

```
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 16: Dictionary, manejo de objetos

Queue (Cola)

Una cola se parece a un List, 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.

E/024.cs

```
namespace Ejemplo;

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

      //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 elmentos 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.
      //Primero en llegar, primero en salir, luego quitaría a "aaa"
      Cola.Dequeue();
      Console.WriteLine("\r\nQuitar 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: " + Buscar);
      }
      else
         Console.WriteLine("\r\n\r\nLa cola NO contiene: " + Buscar);

      //Obtener el primer elemento de la cola
```

```
      //sin borrar ese elemento
      string PrimerElemento = Convert.ToString(Cola.Peek());
      Console.WriteLine("\r\nPrimer elemento: " + 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);
   }
}
```

```
Número de elementos: 6

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

La cola contiene: ddd

Primer elemento: bbb

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

Ilustración 17: Dato definido en la cola

Objetos en la cola

Una cola puede tener objetos personalizados.

E/025.cs

```
namespace Ejemplo;

//Una clase con varios atributos
class MiClase {
   public int Numero { get; set; }
   public double Valor { get; set; }
   public char Car { get; set; }
   public string Cad { get; set; }

   public MiClase(int Numero, double Valor, char Car, string Cad) {
      this.Numero = Numero;
      this.Valor = Valor;
      this.Car = Car;
      this.Cad = Cad;
   }
}

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

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

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

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

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

```
            Console.Write(elemento.Cad + ", ");

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

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

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

```
Número de elementos: 4

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

Primer: Otrebla

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

Elementos: 2
Después de borrar: 0
```

Ilustración 18: Objetos en la cola

Stack (Pila)

La pila 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).

E/026.cs

```
namespace Ejemplo;

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

      //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 elmentos 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
      //Último en llegar, primero en salir,
      //luego quitaría a "fff"
      Pila.Pop();
      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: " + Buscar);
      }
      else
         Console.WriteLine("\r\n\r\nLa pila NO contiene: " + Buscar);
```

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

```
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: ddd

Primer elemento: eee

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

Ilustración 19: Dato definido en la pila

Objetos en la pila

E/027.cs

```
namespace Ejemplo;

//Una clase con varios atributos
class MiClase {
   public int Numero { get; set; }
   public double Valor { get; set; }
   public char Car { get; set; }
   public string Cad { get; set; }

   public MiClase(int Numero, double Valor, char Car, string Cad) {
      this.Numero = Numero;
      this.Valor = Valor;
      this.Car = Car;
      this.Cad = Cad;
   }
}

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

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

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

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

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

      //Obtener el primer elemento de la pila
      //sin borrar ese elemento
      MiClase Primer = Pila.Peek();
```

```
      Console.WriteLine("\r\n\r\nElemento más arriba: " + Primer.Cad);

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

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

```
Número de elementos: 4

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

Elemento más arriba: Onerom

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

Wlementos: 2
Después de borrar: 0
```

Ilustración 20: Objetos en la pila

SortedList

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

E/028.cs

```
namespace Ejemplo;

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() {
         { "exe", "Ejecutable" },
         { "com", "Ejecutable DOS" },
         { "vb", "Visual Basic .NET" },
         { "cs", "C#" },
         { "js", "JavaScript" },
         { "xlsx", "Excel" },
         { "docx", "Word" },
         { "pptx", "PowerPoint" }
      };

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

      //Otra forma de adicionar
      Extensiones.Add("html", "HTML 5");

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

```
[com, Ejecutable DOS]
[cs, C#]
[docx, Word]
[exe, Ejecutable]
[html, HTML 5]
[js, JavaScript]
[pptx, PowerPoint]
[vb, Visual Basic .NET]
[xlsx, Excel]

Imprime llave y valor en separado
Llave: com Valor: Ejecutable DOS
Llave: cs Valor: C#
Llave: docx Valor: Word
Llave: exe Valor: Ejecutable
Llave: html Valor: HTML 5
Llave: js Valor: JavaScript
Llave: pptx Valor: PowerPoint
Llave: vb Valor: Visual Basic .NET
Llave: xlsx Valor: Excel
```

Ilustración 21: SortedList

LinkedList

Lista enlazada, no se accede directamente por un índice.

E/029.cs

```
namespace Ejemplo;

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

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

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

      //Agrega al inicio
      Console.WriteLine("\r\n\r\nAgregando con AddFirst");
      Lenguajes.AddFirst("C++");
      Lenguajes.AddFirst("C");

      //Imprime esa lista
      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\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\nEliminado F#");
        foreach (string elemento in Lenguajes)
           Console.Write(elemento + "; ");

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

```
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 22: LinkedList

E/030.cs

```
namespace Ejemplo;


//Una clase con varios atributos
class MiClase {
   public int Numero { get; set; }
   public double Valor { get; set; }
   public char Car { get; set; }
   public string Cad { get; set; }

   public MiClase(int Numero, double Valor, char Car, string Cad) {
      this.Numero = Numero;
      this.Valor = Valor;
      this.Car = Car;
      this.Cad = Cad;
   }
}

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

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

      //Imprime esa lista
      foreach (MiClase elemento in Lenguajes)
         Console.Write(elemento.Cad + "; ");

      //Agrega al inicio
      Lenguajes.AddFirst(new MiClase(78, 12.32, 'S', "C#"));
      Lenguajes.AddFirst(new MiClase(5, -3.1, 'V', "J#"));

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

      //Agrega al final
```

```
        Lenguajes.AddLast(new MiClase(16, 83.29, 'C', "C++"));
        Console.WriteLine("\r\n\r\nAgregando con AddLast");
        foreach (MiClase elemento in Lenguajes)
            Console.Write(elemento.Cad + "; ");

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

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

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

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

Agregando con AddFirst
J#; C#; Lenguaje R; ADA; Máquina; PHP;

Agregando con AddLast
J#; C#; Lenguaje R; ADA; Máquina; PHP; C++;

Cantidad es: 7

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

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

Ilustración 23: Objetos en LinkedList

SortedSet

Ordenar los elementos y eliminar los repetidos.

E/031.cs

```
namespace Ejemplo;

class Program {
   static void Main() {

      //Ordena y elimina duplicados (enteros)
      SortedSet<int> numeros = [1, 6, 8, 3, 2, 9, 2, 9];

      foreach (var n in numeros)
         Console.Write(n + ", ");

      Console.WriteLine(" ");

      //Ordena y elimina duplicados (char)
      SortedSet<char> letras = ['e', 's', 't', 'a', 'e', 's', 'u', 'n',
'a', 'p', 'r', 'u', 'e', 'b', 'a'];

      foreach (var l in letras)
         Console.Write(l + ", ");
   }
}
```

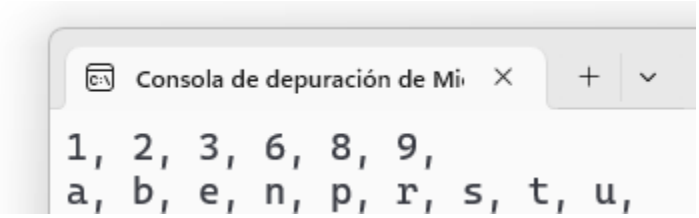


Ilustración 24: SortedSet

PriorityQueue

Cola con prioridad. El segundo argumento es la prioridad.

E/032.cs

```
namespace Ejemplo;

class Program {
   static void Main() {

      // Crea una PriorityQueue con elementos de cadena y prioridades tipo
DateTime
      var pq = new PriorityQueue<string, DateTime>();

      // Pone las tareas
      pq.Enqueue("Tarea 1", new DateTime(2026, 03, 25));
      pq.Enqueue("Tarea 2", new DateTime(2026, 01, 15));
      pq.Enqueue("Tarea 3", new DateTime(2026, 01, 24));
      pq.Enqueue("Tarea 4", new DateTime(2026, 04, 19));
      pq.Enqueue("Tarea 5", new DateTime(2026, 02, 06));
      pq.Enqueue("Tarea 6", new DateTime(2026, 02, 18));

      Console.WriteLine("Ordena las tareas según fecha");
      while (pq.Count > 0) {
         pq.TryDequeue(out string Tarea, out DateTime FechaTermina);
         Console.WriteLine($"{Tarea} - Fecha:
{FechaTermina.ToShortDateString()}");
      }
   }
}
```

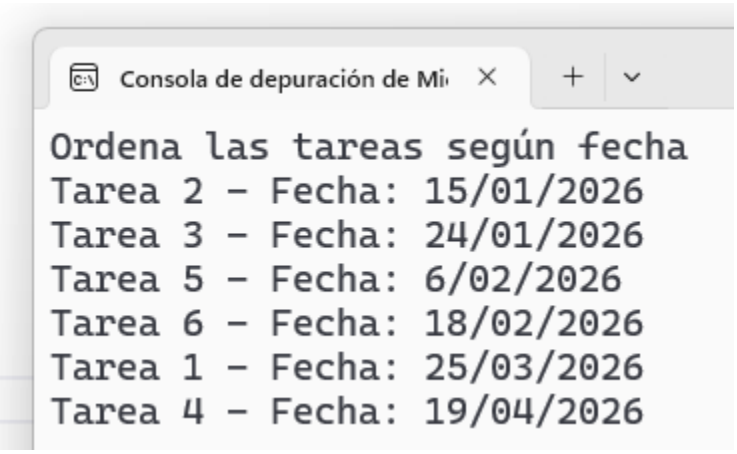


Ilustración 25: Cola con prioridad

SortedDictionary

Estructura que ordena según el valor del primer parámetro.

E/033.cs

```
namespace Ejemplo;

class Program {
   static void Main() {

      SortedDictionary<int, string> DiccionarioOrdenado = [];

      //Agrega elementos al SortedDictionary
      DiccionarioOrdenado.Add(3, "Koala");
      DiccionarioOrdenado.Add(1, "Tiburón");
      DiccionarioOrdenado.Add(7, "Calamar");
      DiccionarioOrdenado.Add(2, "Arenque");
      DiccionarioOrdenado.Add(5, "Cangrejo");
      DiccionarioOrdenado.Add(4, "Estrella de mar");

      foreach (var Dato in DiccionarioOrdenado) {
         Console.WriteLine($"Key: {Dato.Key}, Value: {Dato.Value}");
      }
   }
}
```

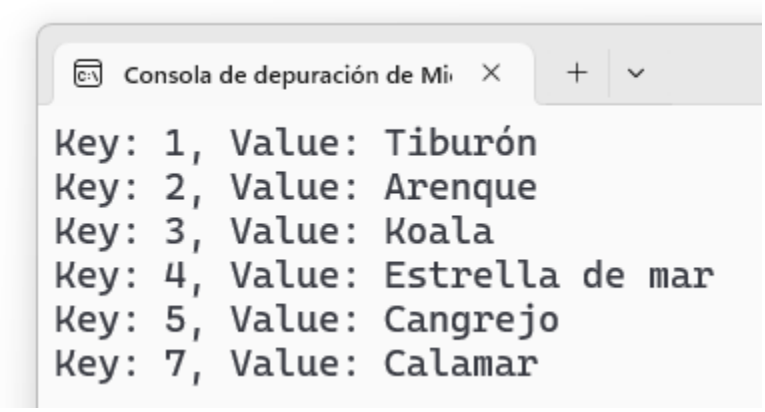


Ilustración 26: SortedDictionary

Persistencia

Almacenar los registros de un List en un archivo y volverlos a recuperar.

E/034.cs

```
namespace Ejemplo;

class Program {
   static void Main() {
      //Declara la lista
      List<string> ListaAnimales = [];

      //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
      Console.WriteLine("LISTA ORIGINAL");
      for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
         Console.Write(ListaAnimales[Cont] + ";");
      Console.WriteLine("\r\n\r\n");

      //Guarda en medio persistente
      using (StreamWriter Escritor = new("MisDatos.txt")) {
         foreach (object item in ListaAnimales) {
            Escritor.WriteLine(item.ToString());
         }
      }

      //Lee de ese medio persistente y lo guarda
      //en un nuevo List
      List<string> NuevaLista = [];
      using (StreamReader Lector = new("MisDatos.txt")) {
         string Linea;
         while ((Linea = Lector.ReadLine()) != null) {
            NuevaLista.Add(Linea);
         }
      }

      //Imprime la lista leída
```


Autor: Rafael Alberto Moreno Parra

```
      Console.WriteLine("LISTA LEIDA");
      for (int Cont = 0; Cont < NuevaLista.Count; Cont++)
         Console.Write(NuevaLista[Cont] + ";");
      Console.WriteLine("\r\n\r\n");
   }
}
```

E/035.cs

```
namespace Ejemplo;

class Program {
   static void Main() {
      List<double> datos = [3.14, -2.71, 1.618, -4.31, 7.89];

      // Guardar como archivo texto plano
      File.WriteAllText("datos.txt", string.Join(";", datos));

      // Leer desde archivo texto plano
      string contenido = File.ReadAllText("datos.txt");
      List<double> datosLeidos = [];
      foreach (var s in contenido.Split(';')) {
         datosLeidos.Add(double.Parse(s));
      }

      //Imprime los datos leídos
      for (int Cont = 0; Cont < datosLeidos.Count; Cont++) {
         Console.WriteLine(datosLeidos[Cont]);
      }
   }
}
```

E/036.cs

```
using System.Text.Json;

namespace Ejemplo;

class Program {
   static void Main() {
      List<double> datos = [3.14, -2.71, 1.618, -4.31, 7.89];

      // Guardar como JSON
      string json = JsonSerializer.Serialize(datos);
      File.WriteAllText("datos.json", json);

      // Leer desde JSON
      string jsonLeido = File.ReadAllText("datos.json");
      List<double> datosLeidos =
JsonSerializer.Deserialize<List<double>>(jsonLeido);

      //Imprime los datos leídos
      for (int Cont = 0; Cont < datosLeidos.Count; Cont++) {
         Console.WriteLine(datosLeidos[Cont]);
      }
   }
}
```

E/037.cs

```
using System.Text.Json;

namespace Ejemplo;

class MiClase {
   public int Entero { get; set; }
   public double Real { get; set; }
   public char Caracter { get; set; }
   public bool Booleano { get; set; }
   public string Cadena { get; set; }
}

class Program {
   static void Main() {
      // Crear lista de objetos
      List<MiClase> lista =
      [
      new MiClase { Entero = -1, Real = 3.1416, Caracter = 'A', Booleano =
true, Cadena = "Kakapu" },
      new MiClase { Entero = 38, Real = -2.7164, Caracter = 'K', Booleano =
false, Cadena = "Ballena" },
      new MiClase { Entero = -16, Real = 1.67, Caracter = 'M', Booleano =
false, Cadena = "Ciervo" },
      new MiClase { Entero = 49, Real = -6.112, Caracter = 'R', Booleano =
false, Cadena = "Cóndor" },
      new MiClase { Entero = -83, Real = 9.4676, Caracter = 'T', Booleano =
true, Cadena = "Águila" },
      new MiClase { Entero = 6, Real = -2.6774, Caracter = 'D', Booleano =
false, Cadena = "Gato" },
      new MiClase { Entero = -29, Real = 7.255, Caracter = 'X', Booleano =
true, Cadena = "Perro" },
      new MiClase { Entero = 72, Real = -5.23765, Caracter = 'L', Booleano
= false, Cadena = "Leopardo" },
      ];

      // Serializar a JSON
      string json = JsonSerializer.Serialize(lista, new
JsonSerializerOptions { WriteIndented = true });

      // Guardar en archivo
      File.WriteAllText("datos.json", json);
      Console.WriteLine("Archivo JSON guardado correctamente.");
```

```
      // Leer el archivo JSON
      string jsonLeido = File.ReadAllText("datos.json");

      // Deserializar a lista de objetos
      List<MiClase> listaRecuperada =
JsonSerializer.Deserialize<List<MiClase>>(jsonLeido);

      // Mostrar los datos
      foreach (var obj in listaRecuperada) {
         Console.WriteLine($"Entero: {obj.Entero}, Real: {obj.Real},
Caracter: {obj.Caracter}, Booleano: {obj.Booleano}, Cadena:
{obj.Cadena}");
      }
   }
}
```

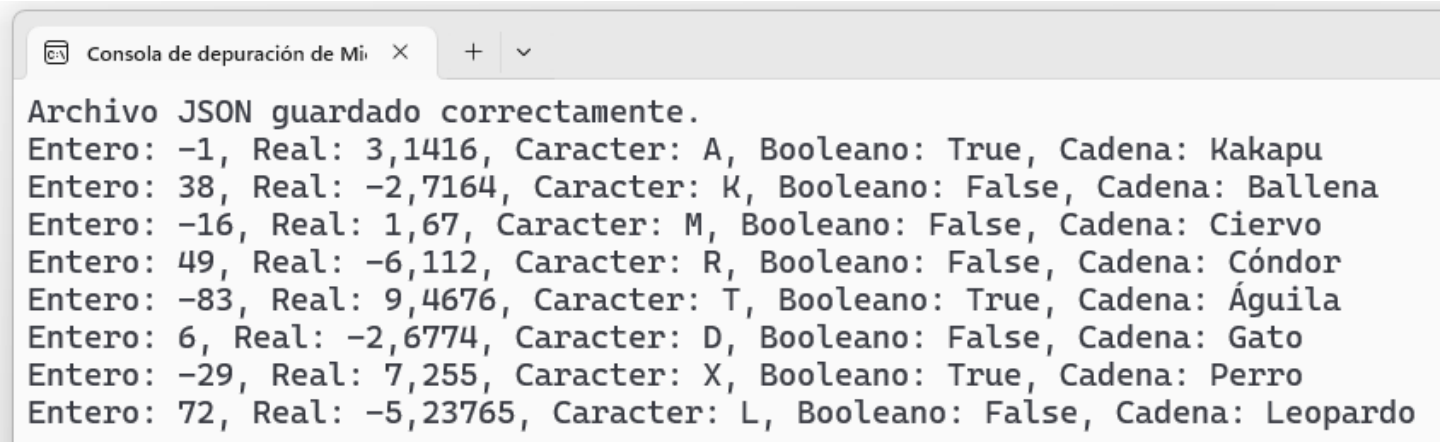


Ilustración 27: Guardando objetos

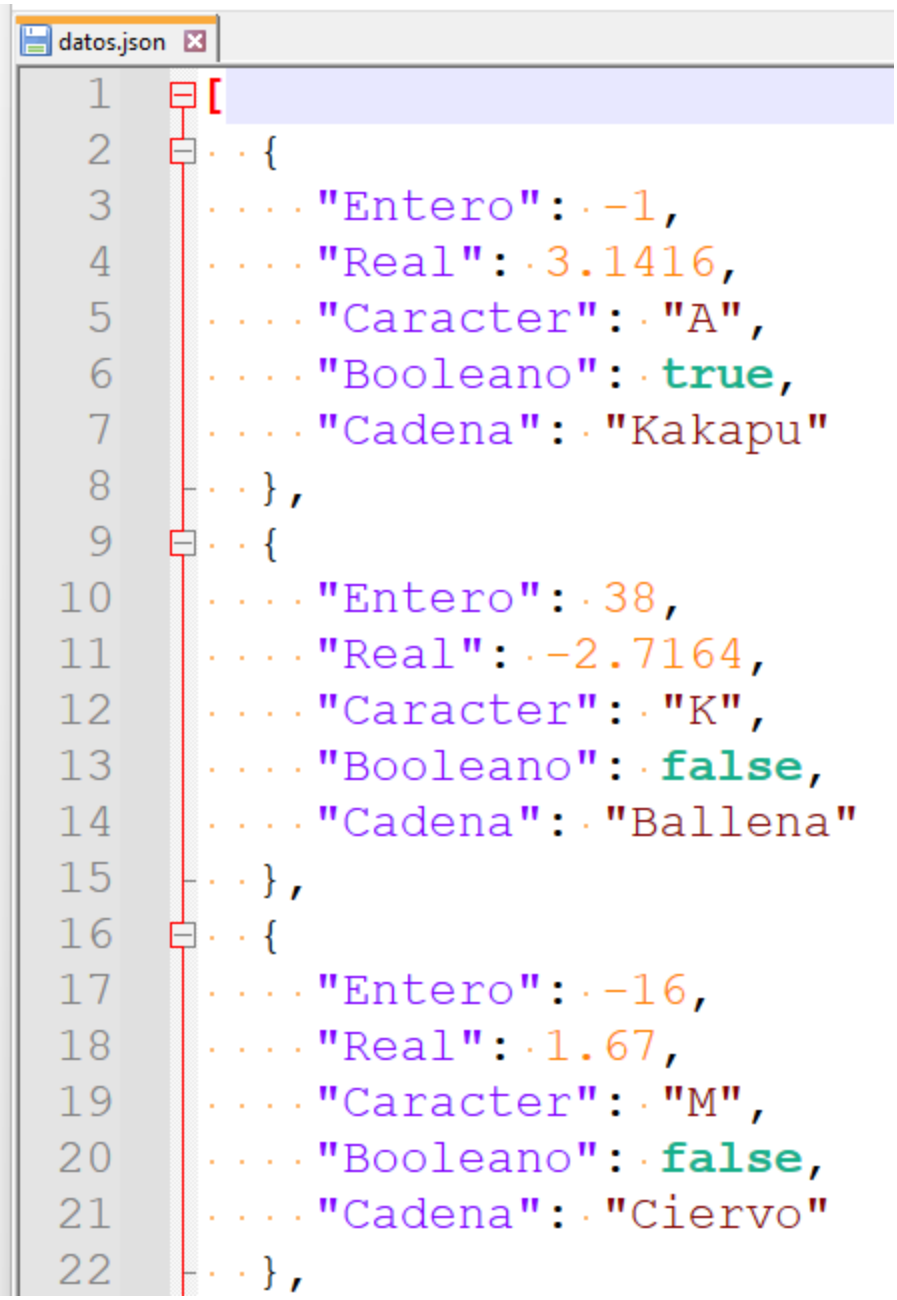


Ilustración 28: Datos JSON