Programmazione C#

Lezione 4

Collezioni e Generici

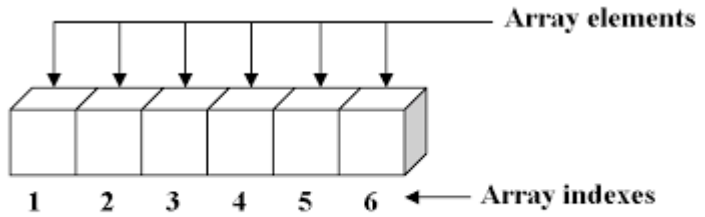
e-Linking online Systems

Sommario della Lezione

1. Collezioni in C#
2. Generici

Collezioni

- In ogni software che si rispetti abbiamo bisogno di «insiemi» di elementi
- Gli Array, che abbiamo già visto, sono un esempio di Collection in C#
- Il Framework .NET fornisce diverse possibilità per la creazione e gestione delle Collection: ArrayList, List, Hashtable ecc…



One-dimensional array with six elements

Gli Array

```
int[] myIntArray = { 1, 5, 2, 3, 6 };
int[] anotherArray = new int[5];
anotherArray[3] = 2;
var length = myIntArray.Length;  // Recupera il numero di elementi
var index =
  Array.IndexOf(myIntArray, 2);  // Indice di un elemento
var item = myIntArray[3];        // Ritorna il quarto elemento
Array.Sort(myIntArray);          // Ordina gli elementi
```

ArrayList

```
var sample = new ArrayList();          // Creazione di un arraylist
sample.Add(2);                         // Aggiunta di elementi

object value = sample[1];              // Ritorna il secondo elemento

int count = sample.Count;              // Recupera il numero di elementi

sample.Remove(2);                      // Rimuove l'intero 2
sample.Clear();                        // Rimuove tutti gli elementi
```

ArrayList

Metodo/Proprietà	Descrizione	Esempio
`Add(item)`	Aggiunge un elemento alla lista	`myList.Add("A string");`
`AddRange(items)`	Aggiunge un insieme di elementi alla lista	`myList.AddRange(new[] {1, 5, 9});`
`Clear()`	Svuota la lista	`myList.Clear();`
`Contains(item)`	Verifica se un elemento appartiene alla lista	`if (myList.Contains("item"))…`
`Count`	Ritorna il numero di elementi	`int items = myList.Count;`
`IndexOf(item)`	Ritorna l'indice della prima occorrenza di un elemento, o -1 nel caso in cui questo non sia presente	`int index = myList.IndexOf("item");`
`Item(index)`	Ritorna l'elemento corrispondente all'indice fornito. La proprietà Item può essere eventualmente omessa.	`object item = myList.Item[2];` oppure `object item = myList[2];`
`LastIndexOf(item)`	Ritorna l'indice dell'ultima occorrenza di un elemento, o -1 nel caso in cui questo non sia presente	`int index = myList.LastIndexOf(5);`
`Insert(index, item)`	Inserisce un nuovo elemento alla posizione specificata da Index	`myList.Insert(0, "First Element");`
`InsertRange(index, items)`	Inserisce un insieme di elementi a partire dalla posizione specificata da Index	`myList.InsertRange(0, new[] {1,2,3});`
`Remove(item)`	Rimuove la prima occorrenza di un determinato elemento	`myList.Remove("A string");`
`RemoveAt(index)`	Rimuove l'elemento il cui indice è pari a quello fornito	`myList.RemoveAt(0);`

Hashtable

```
var myDictionary = new                  // Creazione dell'HashTable
Hashtable();

myDictionary.Add("someIntValue",        // Aggiunta di elementi
5);
myDictionary.Add(
    "someClass",                new
    StringWriter());

myDictionary.Add(
    DateTime.Today,       "Today's
    string");


                                        // Recupera elemento dalla chiave
object value =
    myDictionary["someClass"];


myDictionary.Remove("someClass");       // Rimuove un elemento

int count = myDictionary.Count;         // Recupera il numero di elementi
myDictionary.Clear();                   // Rimuove tutti gli elementi
```

Hashtable

La classe `HashTable` *è così chiamata perché tutte le chiavi sono memorizzate internamente in una struttura basata su hash numerici; ogni oggetto .NET è infatti in grado di generare un proprio hash tramite il metodo* `GetHashCode`*. Tutto ciò rende l'operazione di ricerca di un elemento estremamente veloce ma, allo stesso tempo, richiede un certo overhead per la generazione dell'hash e per la manutenzione della tabella interna.*

Per queste ragioni, nel caso di dizionari con un numero limitato di elementi (inferiore a dieci) è preferibile utilizzare la classe `ListDictionary`*, che implementa le medesime funzionalità, ma con una strategia differente. Una terza classe, chiamata* `HybridDictionary`*, ha la capacità di configurarsi internamente prima come un* `ListDictionary`*, per poi convertirsi a* `HashTable` *nel momento in cui questa soglia viene superata. Entrambe queste classi appartengono al namespace* `System.Collection.Specialized`*.*

```
// Enumerazione di tutte le coppie chiave-valore
foreach (DictionaryEntry item in myDictionary)
{
  Console.WriteLine(item.Key + " " + item.Value);
}
// Enumerazione di tutte le chiavi
foreach (object key in myDictionary.Keys)
{
  Console.WriteLine(key);
}
// Enumerazione di tutti i valori
foreach (object value in myDictionary.Values)
{
```

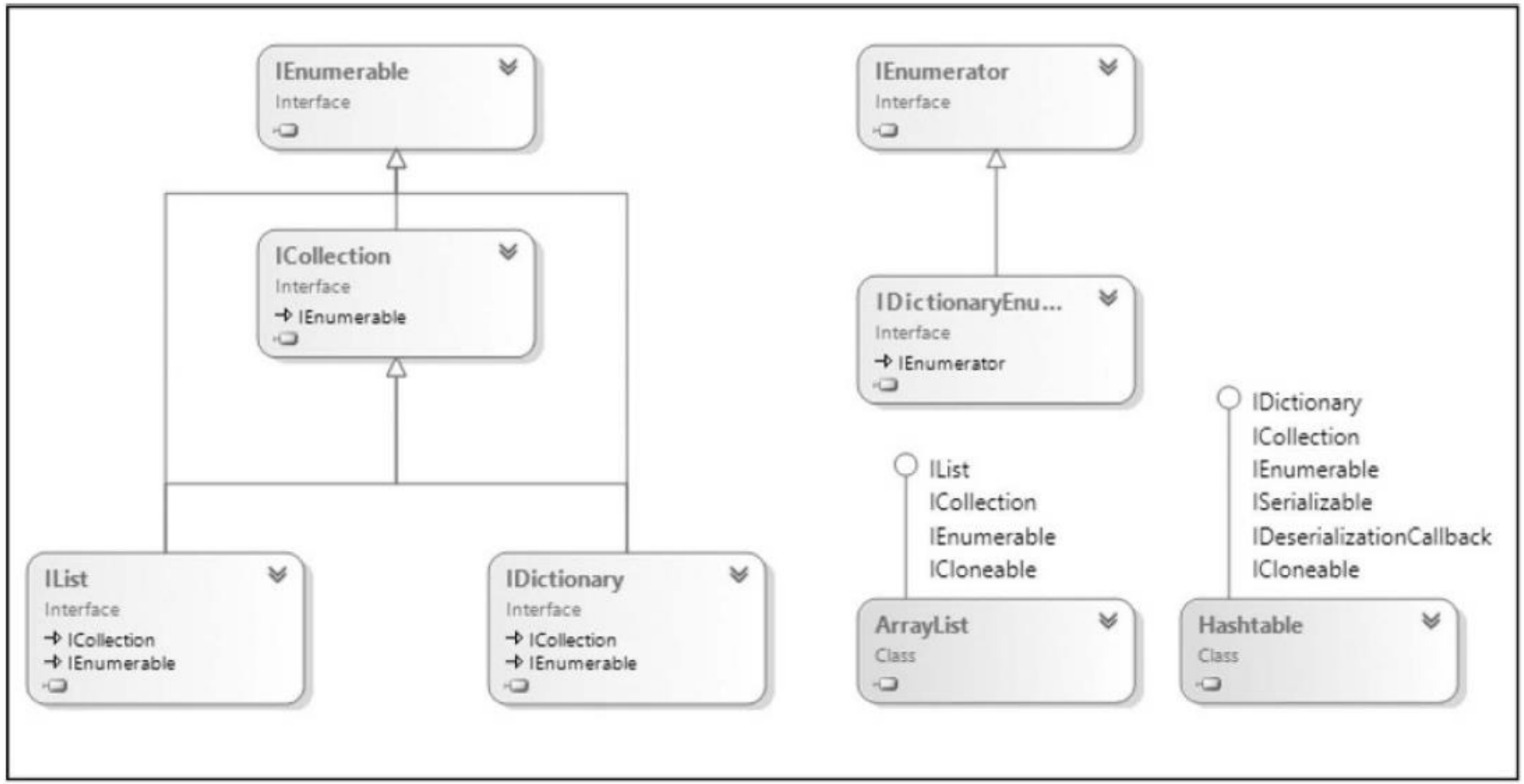
Hashtable

Metodo/Proprietà	Descrizione	Esempio
`Add(key, value)`	Aggiunge un elemento alla HashTable	`myDict.Add("someKey", 5);`
`Clear()`	Svuota il contenuto della HashTable	`myDict.Clear();`
`Contains(key)`	Ritorna true nel caso in cui la chiave specificata sia presente nel dizionario	`if (myDict.Contains("someKey")) …`
`ContainsValue(value)`	Ricerca un determinato valore nel dizionario e restituisce true nel caso in cui questo sia presente	`if (myDict.ContainsValue(5))…`

`Item(key)`	Recupera un valore tramite la corrispondente chiave. La proprietà Item può eventualmente essere omessa	`object value = myDict.Item["somekey"];`
		oppure
		`object value = myDict["someKey"]; Keys`

`Keys`	Restituisce una collection di tutte le chiavi presenti	`foreach (var key in myDict.Keys)…`
`Remove(key)`	Elimina un elemento, data la corrispondente chiave	`myDict.Remove("SomeKey");`
`Values`	Restituisce una collection di tutti i valori presenti	`foreach (var val in myDict.Values)…`

Interfacce System.Collections

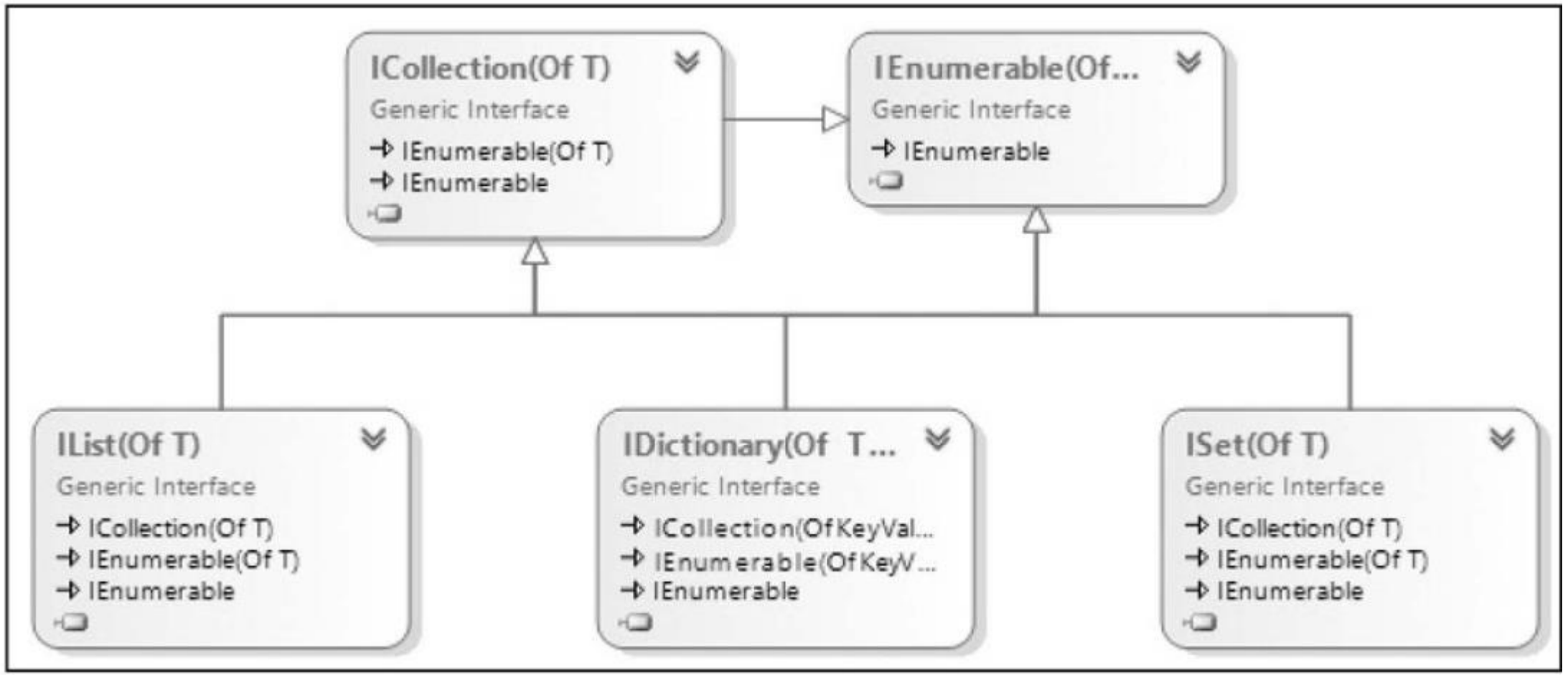


Generici

- I «Generici» vengono utilizzati per «tipizzare» una Collection

```
var strings = new List<string>();     // Inizializzazione di List<T>
strings.Add("Marco De Sanctis");    // Possiamo aggiungere solo string
strings.Add("C# 6");
strings.Insert(0, "Primo elemento");
var index = strings.IndexOf(
  "Marco De Sanctis");              // Ritorna 1
var mySubstring =
  strings[0].Substring(5);                  // Già string: cast non
  necessario
```

Interfacce Generics



Dizionari e Generici

```
var holidays = new Dictionary<string, DateTime>();
holidays.Add("Natale", new DateTime(2015, 12, 25));
holidays.Add("Capodanno", new DateTime(2015, 1, 1));
holidays.Add("Compleanno", new DateTime(2015, 7, 10));
DateTime vigiliaCompleanno = holidays["Compleanno"].AddDays(-1);
```

Creazione di Tipi Generici

```
public class ListNode<T>
{
  public T Value { get; set; }
  public ListNode<T> NextNode { get; set; }
}
```

```
public class Utils
{
  public static void Swap<T>(ref T first, ref T second)
  {
    T temp = first;
    first = second;
    second = temp;
  }
}
```

```
public static T Max<T>(IEnumerable<T> list) where T : IComparable
{
  bool isFirst = true;
  T res = default(T);
  foreach (T item in list)
  {
    if (isFirst)
    {
      res = item;
      isFirst = false;
    }
    if (res.CompareTo(item) < 0)
       res = item;
  }
  return res;
```

`where T : class`	Il tipo T deve essere una classe (tipo riferimento)
`where T : struct`	Il tipo T deve essere una struct (tipo valore)
`where T : new()`	Il tipo T deve avere un costruttore senza parametri
`where T : Interface`	Il tipo T deve implementare l'interfaccia specificata
`where T : BaseClass`	Il tipo T deve ereditare dalla classe base specificata

Fine Lezione 4

DOMANDE?