



Empowering Digital Skills For The Jobs Of The Future



by



Academy .NET

Dizionari Annidati

Sommario

- **Multi-Dictionary**
Un dizionario contenente un elenco di valori
- **Dizionari annidati:**
Un dizionario che contiene un altro dizionario
- **Sets**
HashSet<T>
SortedSet<T>

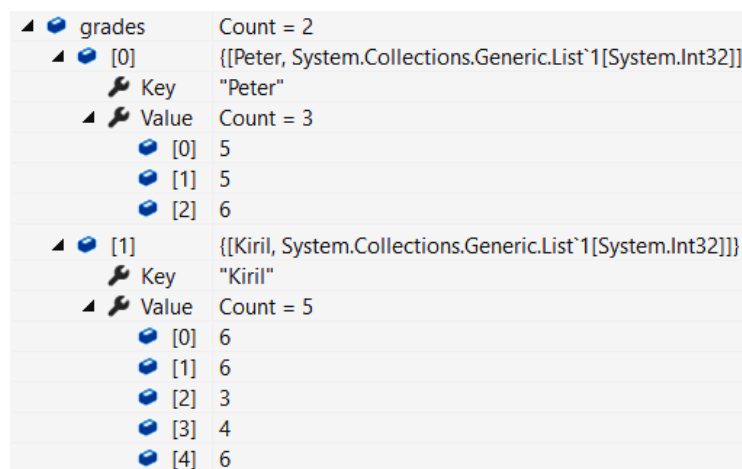
Multi-Dictionary

- Un dizionario può contenere un insieme di valori per chiave data

- Esempio: gli studenti possono avere più voti:

Peter -> [5, 5, 6]

Kiril -> [6, 6, 3, 4, 6]



The screenshot shows a tree view of a variable named 'grades'. It is a Dictionary with two entries. The first entry has a key 'Peter' and a value of type List<int> containing [5, 5, 6]. The second entry has a key 'Kiril' and a value of type List<int> containing [6, 6, 3, 4, 6].

Node	Count	Value
grades	2	
[0]		{[Peter, System.Collections.Generic.List`1[System.Int32]]}
Key		"Peter"
Value	3	
[0]		5
[1]		5
[2]		6
[1]		{[Kiril, System.Collections.Generic.List`1[System.Int32]]}
Key		"Kiril"
Value	5	
[0]		6
[1]		6
[2]		3
[3]		4
[4]		6

- **Dictionary <string, List<int>> voti;**

Esempio: voti degli studenti

- Ogni studente ha un elenco di voti
- Possiamo accedere ai voti dello studente per nome:

```
var grades = new Dictionary<string, List<int>>();  
  
grades["Peter"] = new List<int>();  
grades["Peter"].Add(5);  
grades["Peter"].Add(5);  
grades["Peter"].Add(6);  
  
var kirilGrades = new List<int>() { 6, 6, 3, 4, 6 };  
grades["Kiril"] = kirilGrades;  
  
Console.WriteLine(string.Join(" ", grades["Kiril"])); // 6 6 3 4 6
```

Diagram illustrating the code execution with annotations:

- Valore: List<int>**: Points to the `List<int>` type in the dictionary declaration.
- Inizializza la lista**: Points to the `new List<int>()` initialization for Peter.
- Accesso alla lista per chiave(key)**: Points to the `grades["Peter"]` access in the `Add` calls.
- Assegnazione lista alla chiave**: Points to the `grades["Kiril"] = kirilGrades;` assignment.

Problema: voto medio degli studenti

- Scrivi un programma per leggere i nomi degli studenti + i voti
Stampa i voti + voto medio per ogni studente come mostrato di seguito:

```
7  
Ivancho 5.20  
Mariika 5.50  
Ivancho 3.20  
Mariika 2.50  
Stamat 2.00  
Mariika 3.46  
Stamat 3.00
```



```
Ivancho -> 5.20 3.20 (avg: 4.20)  
Mariika -> 5.50 2.50 3.46 (avg: 3.82)  
Stamat -> 2.00 3.00 (avg: 2.50)
```

Soluzione: voto medio degli studenti (1)

```
var grades = new Dictionary<string, List<double>>();
```

```
var n = int.Parse(Console.ReadLine());
```

```
for (int i = 0; i < n; i++)
```

```
{
```

```
    var tokens = Console.ReadLine().Split();
```

```
    var name = tokens[0];
```

```
    var grade = double.Parse(tokens[1]);
```

```
    if (!grades.ContainsKey(name))
```

```
        grades[name] = new List<double>();
```

Lista
inizializzata

```
    grades[name].Add(grade);
```

Aggiunta
grado in lista

```
// continued on next slide ...
```


Soluzione: voti medi degli studenti (2)

```
foreach (var pair in grades)
{
    var name = pair.Key;
    var studentGrades = pair.Value;

    var average = studentGrades.Average();

    Console.WriteLine($"{name} -> ");

    foreach (var grade in studentGrades)
        Console.WriteLine($"{grade:f2} ");

    Console.WriteLine($"(avg: {average:f2})");
}
```

Key: string

Value:
List<string>

Valore medio
della lista

Dizionari Annidati

I dizionari possono contenere un altro **Dictionary<K,V>** come valore

- Esempio: popolazione per paese e città



Problema: città per continente e paese

- Scrivi un programma per leggere continenti, paesi e le loro città, inseriscili in un dizionario annidato e stampali.

9

Europe Bulgaria Sofia

Asia China Beijing

Asia Japan Tokyo

Europe Poland Warsaw

Europe Germany Berlin

Europe Poland Poznan

Europe Bulgaria Plovdiv

Africa Nigeria Abuja

Asia China Shanghai



Europe:

Bulgaria -> Sofia, Plovdiv

Poland -> Warsaw, Poznan

Germany -> Berlin

Asia:


China -> Beijing, Shanghai

Africa:

Nigeria -> Abuja

Soluzione: città per continente e paese (1)

```
var continentsData =  
    new Dictionary<string, Dictionary<string, List<string>>>>();  
var n = int.Parse(Console.ReadLine());  
for (int i = 0; i < n; i++)  
{  
    var tokens = Console.ReadLine().Split();  
    var continent = tokens[0];  
    var country = tokens[1];  
    var city = tokens[2];  
  
    // continued on next slide  
}
```



Inizializzazione del
dizionario

Soluzione: città per continente e paese (2)

```
if (!continentsData.ContainsKey(continent))  
    continentsData[continent] =  
        new Dictionary<string, List<string>>();
```

Creazione di
Continet

```
if (!continentsData[continent].ContainsKey(country))  
    continentsData[continent][country] = new List<string>();
```

Country non esiste

```
continentsData[continent][country].Add(city);  
}
```

Inizializzazione
città

```
// continued on next slide...
```

Aggiunta città a
Paese

Soluzione: città per continente e paese (3)

```
foreach (var continentCountries in continentsData)
```

```
{
```

```
    var continentName = continentCountries.Key;
```

```
    var countries = continentCountries.Value;
```

```
    Console.WriteLine($"{continentName}:");
```

```
    foreach (var countryCities in countries)
```

```
    {
```

```
        var countryName = countryCities.Key;
```

```
        var cities = countryCities.Value;
```

```
        Console.WriteLine("    {0} -> {1}",  
            countryName, string.Join(", ", cities));
```

```
    }
```

```
}
```

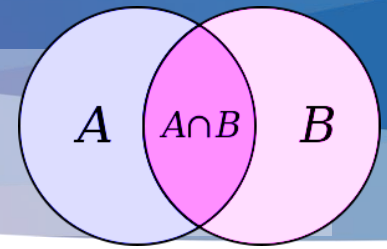
Nome Continente

Paesi nel Continente

Nome Paese

Città nel Paese

I Set



Un **Set** conserva elementi unici:

Consente di aggiungere / rimuovere / cercare elementi

Prestazioni molto rapide

HashSet<T>

Mantiene un insieme di elementi in una tabella hash

Gli elementi non sono in un ordine particolare

Simile a **List<T>**, ma un'implementazione diversa

SortedSet<T>

Gli elementi sono ordinati in modo incrementale

List<T> vs HashSet<T>

List<T>

"add" rapido, lenti "search" e "remove"
(passa attraverso ogni elemento)

Sono consentiti i **duplicati**

L'ordine di inserimento è garantito

HashSet<T>

"add", "search" e "remove" rapidi

Non consente duplicati

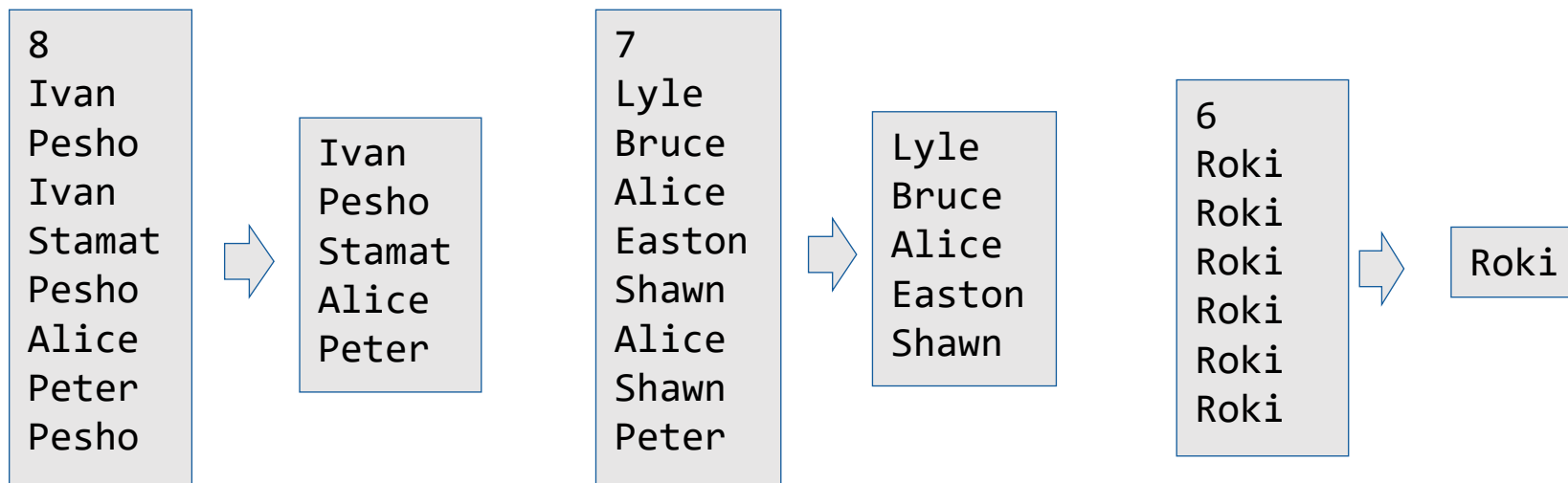
Non garantisce l'ordine di inserimento

HashSet<T> - Esempio

```
HashSet<string> set = new HashSet<string>();  
set.Add("Pesho");  
set.Add("Pesho"); // Not added again  
set.Add("Gosho");  
set.Add("Alice");  
Console.WriteLine(string.Join(", ", set));  
// Pesho, Gosho, Alice  
Console.WriteLine(set.Contains("Georgi")); // false  
Console.WriteLine(set.Contains("Pesho")); // true  
set.Remove("Pesho");  
Console.WriteLine(set.Count); // 2
```

Problema: registra nomi univoci

- Leggi una sequenza di nomi e stampa solo quelli univoci



Soluzione: registra nomi univoci

```
var names = new HashSet<string>();  
var n = int.Parse(Console.ReadLine());  
for (int i = 0; i < n; i++)  
{  
    var name = Console.ReadLine();  
    names.Add(name);  
}  
foreach (var name in names)  
    Console.WriteLine(name);
```

HashSet accetta
solo valori univoci

Aggiunge solamente
nomi non presenti

SortedSet <T> - Esempio

```
SortedSet<string> set = new SortedSet<string>();  
set.Add("Pesho");  
set.Add("Pesho");  
set.Add("Pesho");  
set.Add("Gosho");  
set.Add("Maria");  
set.Add("Alice");  
Console.WriteLine(string.Join(", ", set));  
// Alice, Gosho, Maria, Pesho
```

Ordine alfabetico

Domande & approfondimenti

Academy .NET