

Dictionaries, Hash Tables and Sets

Dictionaries, Hash Tables, Hashing, Collisions, Sets

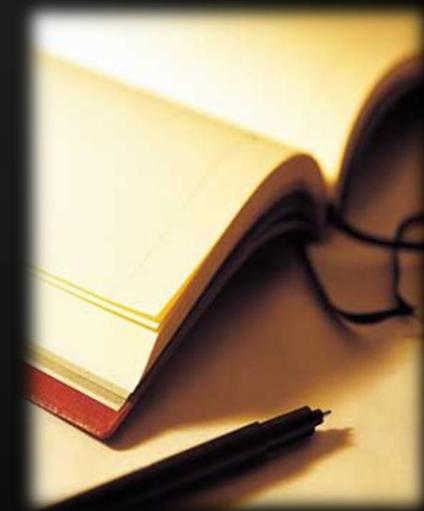
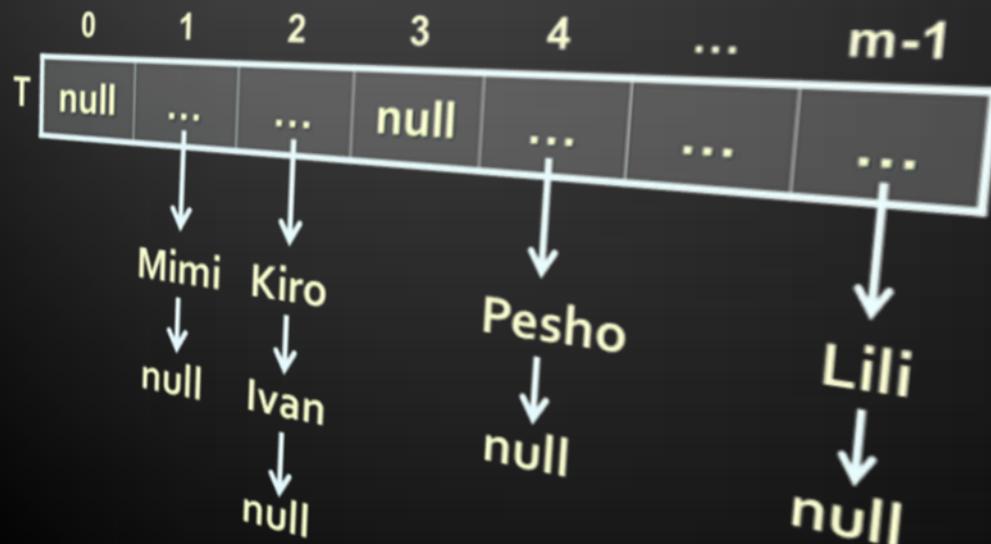
Data Structures and Algorithms

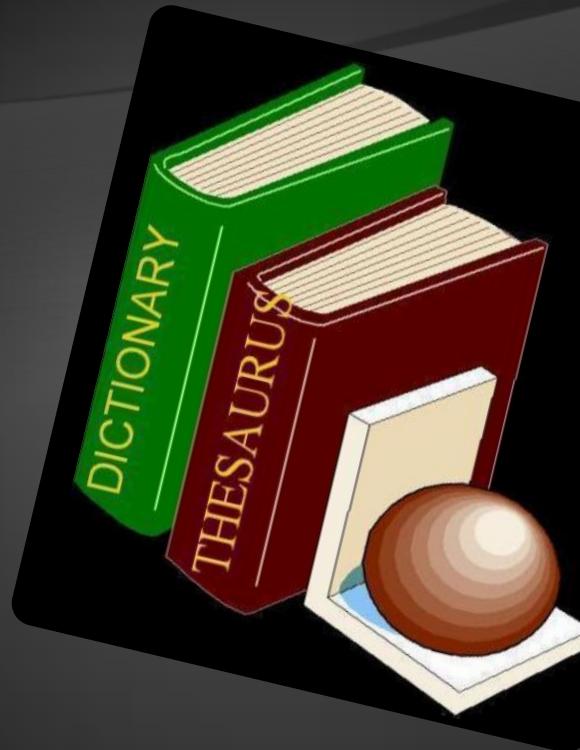
Telerik Software Academy

<http://academy.telerik.com>



1. Dictionaries
2. Hash Tables
3. Dictionary< TKey, TValue> Class
4. Sets: HashSet< T > and SortedSet< T >





Dictionaries

Data Structures that Map Keys to Values

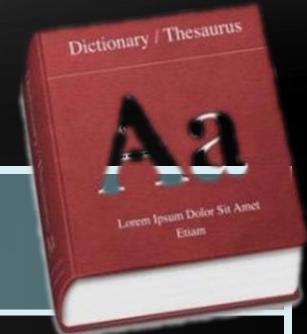
The Dictionary (Map) ADT

- ◆ The abstract data type (ADT) "dictionary" maps key to values
 - ◆ Also known as "map" or "associative array"
 - ◆ Contains a set of (key, value) pairs
- ◆ Dictionary ADT operations:
 - ◆ Add(key, value)
 - ◆ FindByKey(key) → value
 - ◆ Delete(key)
- ◆ Can be implemented in several ways
 - ◆ List, array, hash table, balanced tree, ...

key	value
firstName	Bugs
lastName	Bunny
location	Earth

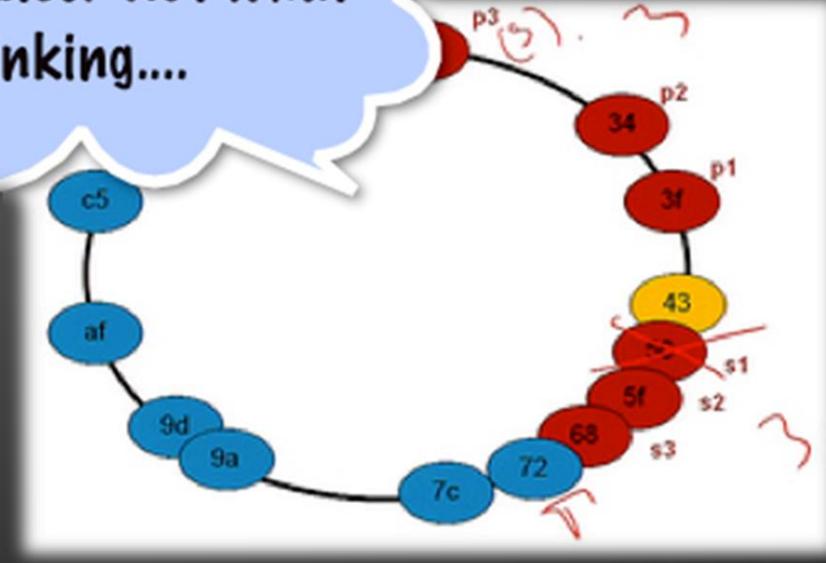
ADT Dictionary – Example

- ◆ Example dictionary:



Key	Value
C#	Modern object-oriented programming language for the Microsoft .NET platform
CLR	Common Language Runtime – execution engine for .NET assemblies, integral part of .NET Framework
compiler	Software that transforms a computer program to executable machine code
...	...

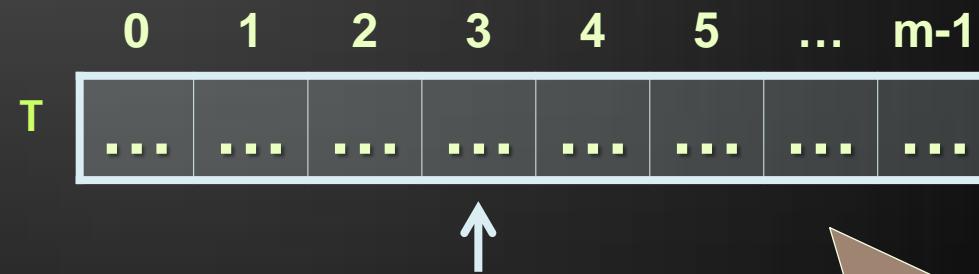
Hash Tables! Not what
you're thinking....



Hash Tables

What is Hash Table? How it Works?

- ◆ A hash table is an array that holds a set of (key, value) pairs
- ◆ The process of mapping a key to a position in a table is called hashing



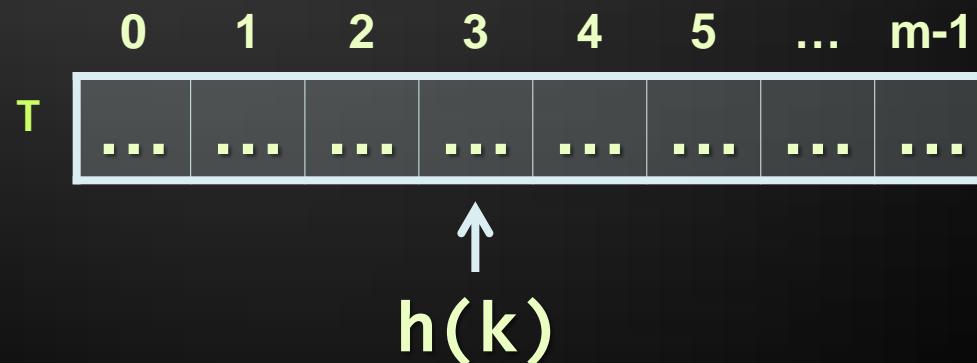
Hash function
 $h: k \rightarrow 0 \dots m-1$

$h(k)$

Hash table
of size m

Hash Functions and Hashing

- ◆ A hash table has m slots, indexed from 0 to $m-1$
- ◆ A hash function $h(k)$ maps keys to positions:
 - ◆ $h: k \rightarrow 0 \dots m-1$
- ◆ For any value k in the key range and some hash function h we have $h(k) = p$ and $0 \leq p < m$



- ◆ Perfect hashing function (PHF)
 - ◆ $h(k)$: one-to-one mapping of each key k to an integer in the range $[0, m-1]$
 - ◆ The PHF maps each key to a distinct integer within some manageable range
- ◆ Finding a perfect hashing function is in most cases impossible
- ◆ More realistically
 - ◆ Hash function $h(k)$ that maps most of the keys onto unique integers, but not all

Collisions in a Hash Table

- ◆ A collision is the situation when different keys have the same hash value

$$h(k_1) = h(k_2) \text{ for } k_1 \neq k_2$$

- ◆ When the number of collisions is sufficiently small, the hash tables work quite well (fast)
- ◆ Several collisions resolution strategies exist
 - Chaining in a list
 - Using the neighboring slots (linear probing)
 - Re-hashing (second hash function)
 - ...



Collision Resolution: Chaining

$h("Pesho") = 4$

$h("Kiro") = 2$

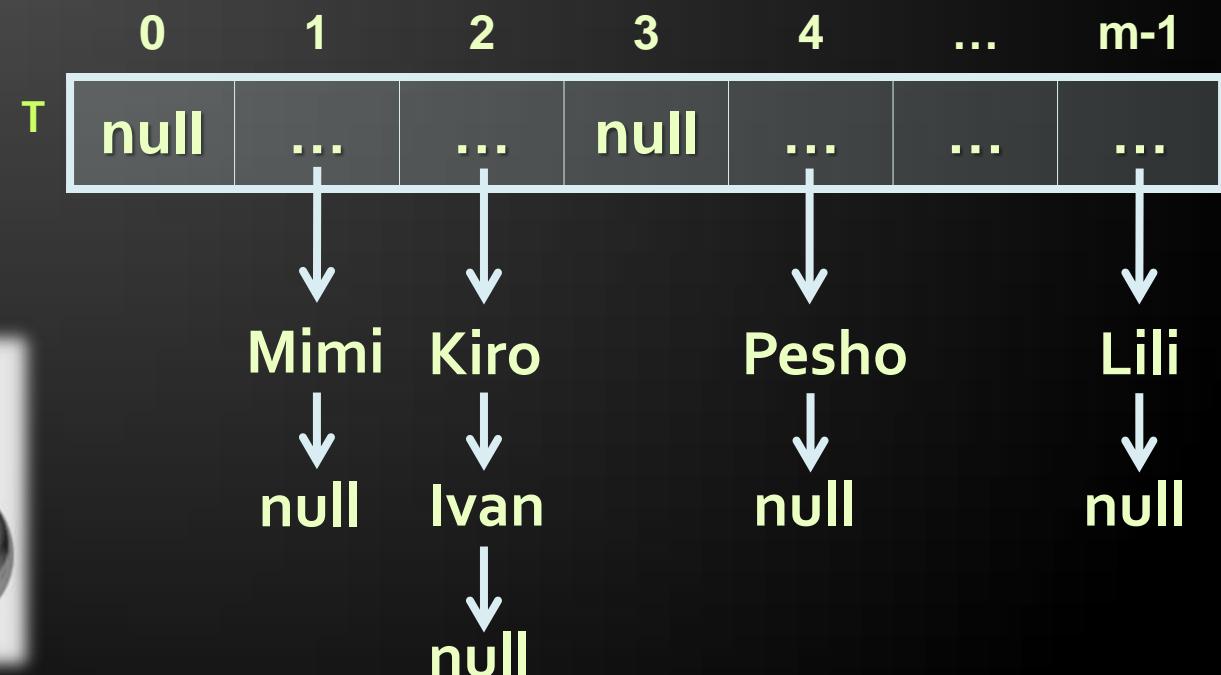
$h("Mimi") = 1$

$h("Ivan") = 2$

$h("Lili") = m-1$

collision

Chaining
elements in
case of collision



Hash Tables and Efficiency

- ◆ Hash tables are the most efficient implementation of ADT "dictionary"
- ◆ Add / Find / Delete take just few primitive operations
 - ◆ Speed does not depend on the size of the hash-table (constant time)
 - ◆ Amortized complexity $O(1)$
 - ◆ Example: finding an element in a hash-table with 1 000 000 elements, takes just few steps
 - ◆ Finding an element in array of 1 000 000 elements takes average 500 000 steps

Dictionaries: .NET Interfaces and Implementations

IDictionary< TKey, TValue >

Generic Interface

- ICollection<KeyValuePair< TKey, TValue >>
- IEnumerable<KeyValuePair< TKey, TValue ...>>
- IEnumerable

Dictionary< TKey, TValue >

Generic Class

SortedDictionary< TKey, TValue >

Generic Class





Hash Tables in C#

The Dictionary<TKey, TValue> Class

Dictionary< TKey , TValue >

- ◆ Implements the ADT dictionary as hash table
 - The size is dynamically increased as needed
 - Contains a collection of key-value pairs
 - Collisions are resolved by chaining
 - Elements have almost random order
 - Ordered by the hash code of the key
- ◆ Dictionary< TKey , TValue > relies on
 - Object.Equals() – for comparing the keys
 - Object.GetHashCode() – for calculating the hash codes of the keys

Dictionary< TKey , TValue > (2)

- ◆ Major operations:

- **Add(TKey, TValue)** – adds an element with the specified key and value
- **Remove(TKey)** – removes the element by key
- **this[]** – get / add / replace of element by key
- **Clear()** – removes all elements
- **Count** – returns the number of elements
- **Keys** – returns a collection of the keys
- **Values** – returns a collection of the values

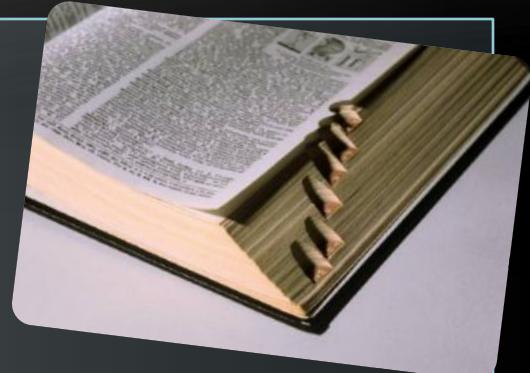
Dictionary< TKey , TValue > (3)

◆ Major operations:

- **ContainsKey(TKey)** – checks whether the dictionary contains given key
- **ContainsValue(TValue)** – checks whether the dictionary contains given value
 - Warning: slow operation – O(n)
- **TryGetValue(TKey, out TValue)**
 - If the key is found, returns it in the TValue
 - Otherwise returns false

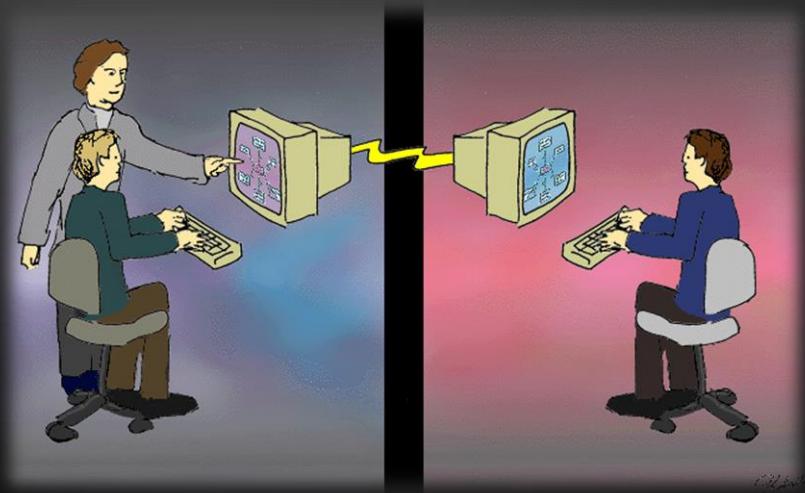
Dictionary< TKey, TValue > – Example

```
Dictionary<string, int> studentsMarks =  
    new Dictionary<string, int>();  
studentsMarks.Add("Ivan", 4);  
studentsMarks.Add("Peter", 6);  
studentsMarks.Add("Maria", 6);  
studentsMarks.Add("George", 5);  
  
int peterMark = studentsMarks["Peter"];  
Console.WriteLine("Peter's mark: {0}", peterMark);  
Console.WriteLine("Is Peter in the hash table: {0}",  
    studentsMarks.ContainsKey("Peter"));  
  
Console.WriteLine("Students and grades:");  
foreach (var pair in studentsMarks)  
{  
    Console.WriteLine("{0} --> {1}", pair.Key, pair.Value);  
}
```



Dictionary< TKey, TValue >

Live Demo



inter- /ɪn'tər-/ *prefix*
sides in a conflict group.
- ORIGIN C17: from L. *inter-* 'between' + *necare* 'to kill'.
internee /'ɪntə:nɪ:/ **n.** a person held in confinement by an occupying power.
internegative **n.** a negative photograph made from the original positive.
Internet **n.** an international network of computers, accessible via a single point of entry.
- ORIGIN C20: from **INTER-** + **net**.
interneuron /'ɪntə'njʊən/ **n.** Physiology an interneuron.
- DERIVATIVES **internetwork** **v.** to connect computers or computer networks.
internist **n.** Medicine a specialist in internal medicine.

Counting the Words in a Text

```
string text = "a text, some text, just some text";
IDictionary<string, int> wordsCount =
    new Dictionary<string, int>();
string[] words = text.Split(' ', ',', '.', ',');
foreach (string word in words)
{
    int count = 1;
    if (wordsCount.ContainsKey(word))
        count = wordsCount[word] + 1;
    wordsCount[word] = count;
}
foreach(var pair in wordsCount)
{
    Console.WriteLine("{0} -> {1}", pair.Key, pair.Value);
}
```





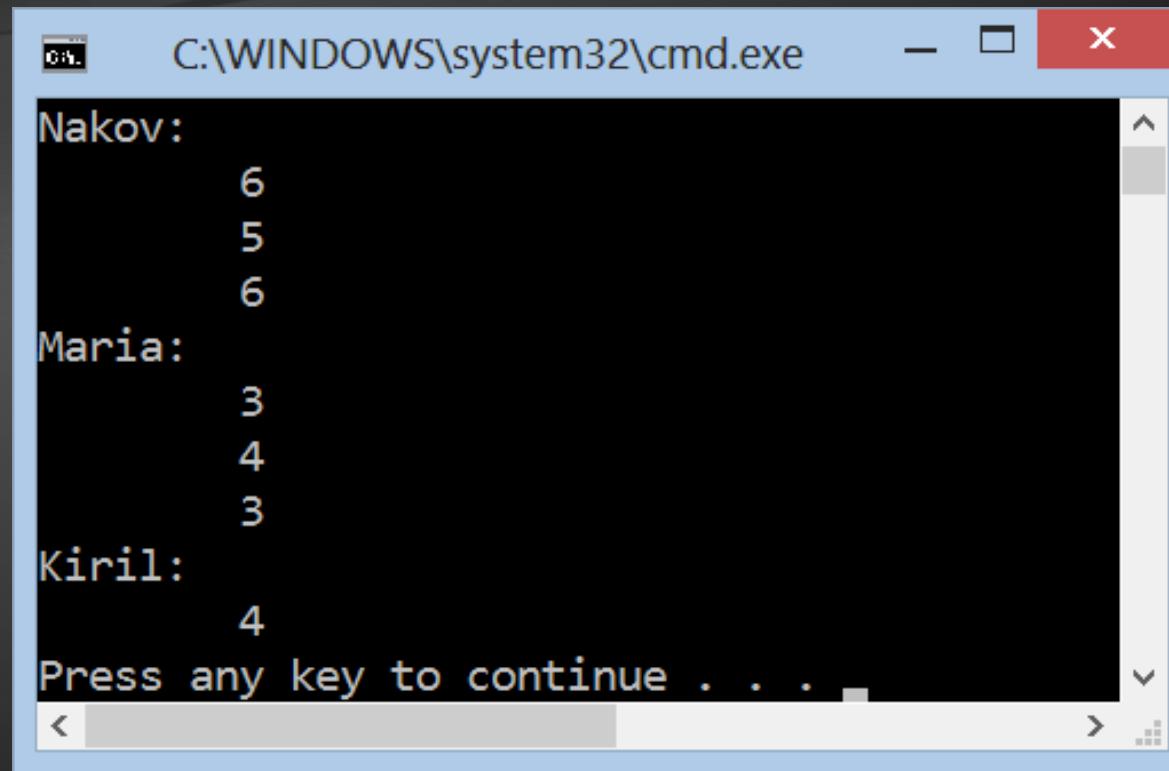
Counting the Words in a Text

Live Demo

Nested Data Structures

- ◆ Data structures can be nested, e.g. dictionary of lists: `Dictionary<string, List<int>>`

```
static Dictionary<string, List<int>> studentGrades =  
    new Dictionary<string, List<int>>();  
  
private static void AddGrade(string name, int grade)  
{  
    if (! studentGrades.ContainsKey(name))  
    {  
        studentGrades[name] = new List<int>();  
    }  
    studentGrades[name].Add(grade);  
}
```



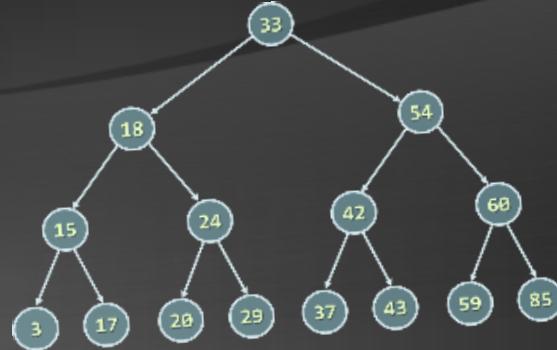
A screenshot of a Windows Command Prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays a dictionary where names are keys and lists of numbers are values. The entries are:

- Nakov: 6, 5, 6
- Maria: 3, 4, 3
- Kiril: 4

The text "Press any key to continue . . ." is visible at the bottom of the window.

Dictionary of Lists

Live Demo



Balanced Tree Dictionaries

The `SortedDictionary<TKey, TValue>` Class



SortedDictionary < TKey, TValue >

- **SortedDictionary< TKey , TValue >** implements the ADT "dictionary" as self-balancing search tree
 - Elements are arranged in the tree ordered by key
 - Traversing the tree returns the elements in increasing order
 - Add / Find / Delete perform $\log_2(n)$ operations
 - Use **SortedDictionary< TKey , TValue >** when you need the elements sorted by key
 - Otherwise use **Dictionary< TKey , TValue >** – it has better performance

Counting Words (Again)

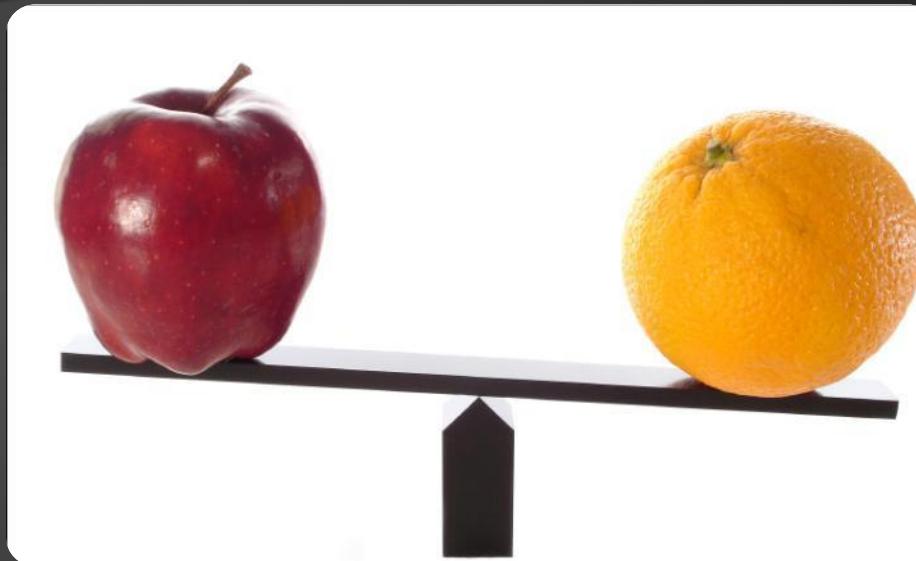
```
string text = "a text, some text, just some text";
IDictionary<string, int> wordsCount =
    new SortedDictionary<string, int>();
string[] words = text.Split(' ', ',', '.', ',');
foreach (string word in words)
{
    int count = 1;
    if (wordsCount.ContainsKey(word))
        count = wordsCount[word] + 1;
    wordsCount[word] = count;
}
foreach(var pair in wordsCount)
{
    Console.WriteLine("{0} -> {1}", pair.Key, pair.Value);
}
```





Counting the Words in a Text

Live Demo



Comparing Dictionary Keys

Using custom key classes in `Dictionary<TKey, TValue>` and `SortedDictionary<TKey, TValue>`

- ◆ **Dictionary< TKey , TValue >** relies on
 - `Object.Equals()` – for comparing the keys
 - `Object.GetHashCode()` – for calculating the hash codes of the keys
- ◆ **SortedDictionary< TKey , TValue >** relies on **IComparable<T>** for ordering the keys
- ◆ Built-in types like `int`, `long`, `float`, `string` and `DateTime` already implement `Equals()`, `GetHashCode()` and **IComparable<T>**
 - Other types used when used as dictionary keys should provide custom implementations

Implementing Equals() and GetHashCode()

```
public struct Point
{
    public int X { get; set; }
    public int Y { get; set; }

    public override bool Equals(Object obj)
    {
        if (!(obj is Point) || (obj == null)) return false;
        Point p = (Point)obj;
        return (X == p.X) && (Y == p.Y);
    }

    public override int GetHashCode()
    {
        return (X << 16 | X >> 16) ^ Y;
    }
}
```

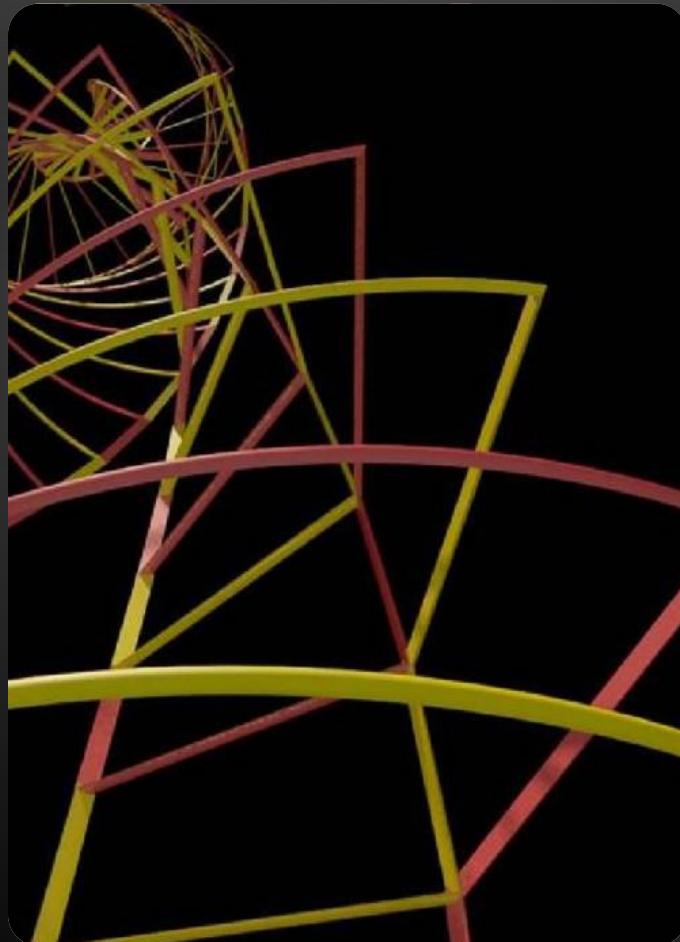


Implementing IComparable<T>

```
public struct Point : IComparable<Point>
{
    public int X { get; set; }
    public int Y { get; set; }

    public int CompareTo(Point otherPoint)
    {
        if (X != otherPoint.X)
        {
            return this.X.CompareTo(otherPoint.X);
        }
        else
        {
            return this.Y.CompareTo(otherPoint.Y);
        }
    }
}
```



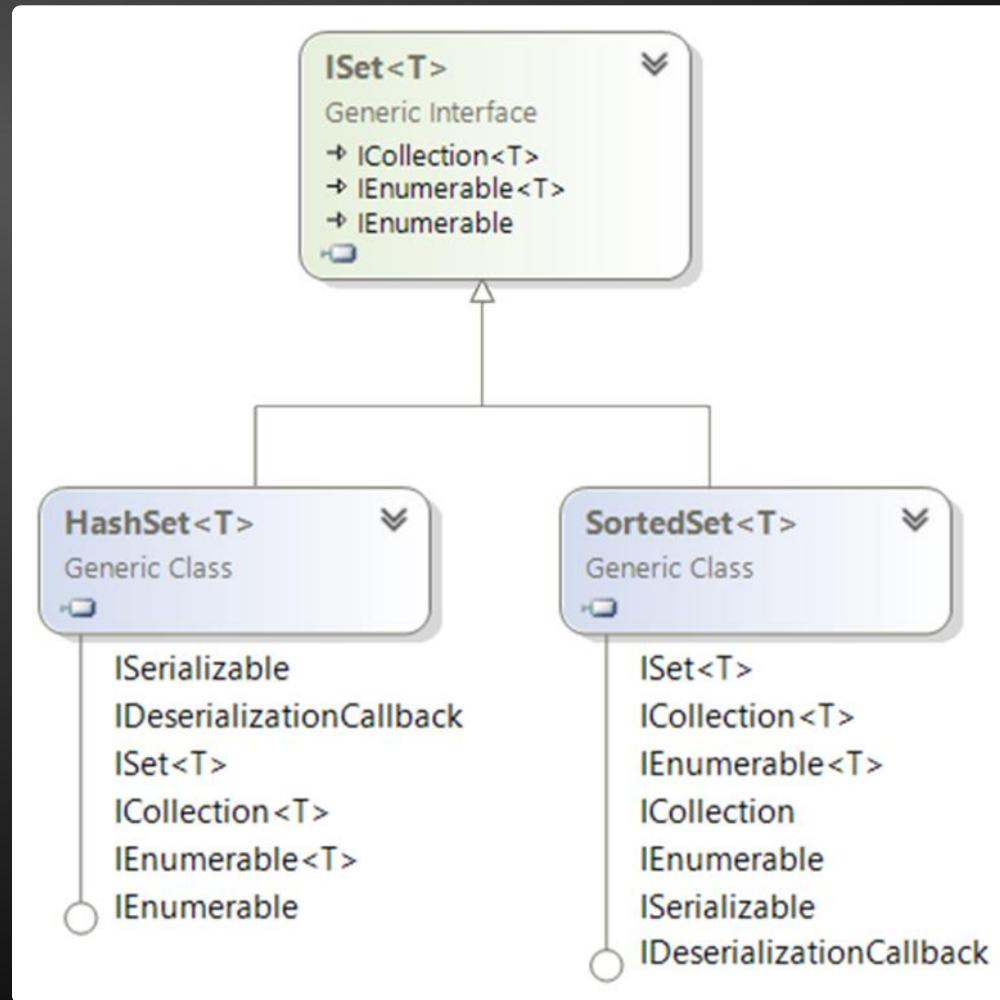


Sets

Sets of Elements

- ◆ The abstract data type (ADT) "set" keeps a set of elements with no duplicates
- ◆ Sets with duplicates are also known as ADT "bag"
- ◆ Set operations:
 - ◆ Add(element)
 - ◆ Contains(element) → true / false
 - ◆ Delete(element)
 - ◆ Union(set) / Intersect(set)
- ◆ Sets can be implemented in several ways
 - ◆ List, array, hash table, balanced tree, ...

Sets: .NET Interfaces and Implementations



- ◆ HashSet<T> implements ADT set by hash table
 - ◆ Elements are in no particular order
- ◆ All major operations are fast:
 - ◆ Add(element) – appends an element to the set
 - ◆ Does nothing if the element already exists
 - ◆ Remove(element) – removes given element
 - ◆ Count – returns the number of elements
 - ◆ UnionWith(set) / IntersectWith(set) – performs union / intersection with another set

HashSet<T> – Example

```
ISet<string> firstSet = new HashSet<string>(
    new string[] { "SQL", "Java", "C#", "PHP"  });
ISet<string> secondSet = new HashSet<string>(
    new string[] { "Oracle", "SQL", "MySQL"  });

ISet<string> union = new HashSet<string>(firstSet);
union.UnionWith(secondSet);
PrintSet(union); // SQL Java C# PHP Oracle MySQL

private static void PrintSet<T>(ISet<T> set)
{
    foreach (var element in set)
    {
        Console.Write("{0} ", element);
    }
    Console.WriteLine();
}
```



- ◆ **SortedSet<T> implements ADT set by balanced search tree (red-black tree)**
 - ◆ Elements are sorted in increasing order
- ◆ **Example:**

```
ISet<string> firstSet = new SortedSet<string>(  
    new string[] { "SQL", "Java", "C#", "PHP" });  
ISet<string> secondSet = new SortedSet<string>(  
    new string[] { "Oracle", "SQL", "MySQL" });  
ISet<string> union = new  
HashSet<string>(firstSet);  
union.UnionWith(secondSet);  
PrintSet(union); // C# Java PHP SQL MySQL Oracle
```

HashSet<T> and SortedSet<T>

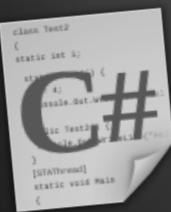
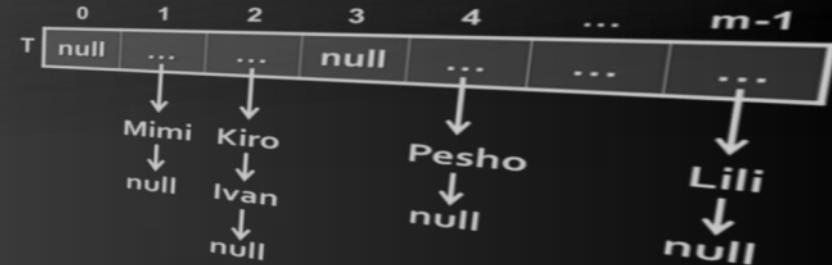
Live Demo



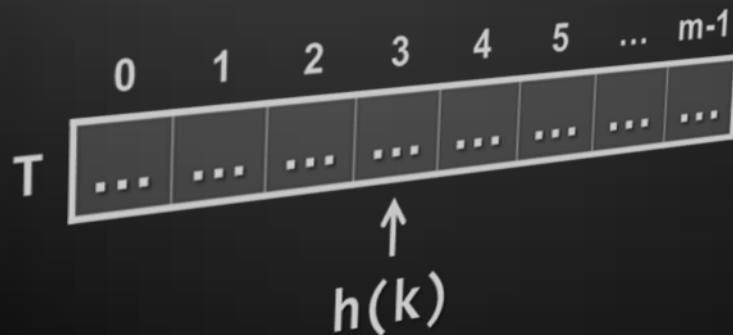
- ◆ Dictionaries map key to value
 - ◆ Can be implemented as hash table or balanced search tree
- ◆ Hash-tables map keys to values
 - ◆ Rely on hash-functions to distribute the keys in the table
 - ◆ Collisions needs resolution algorithm (e.g. chaining)
 - ◆ Very fast add / find / delete – O(1)
- ◆ Sets hold a group of elements
 - ◆ Hash-table or balanced tree implementations



Dictionaries, Hash Tables and Sets



Questions?



1. Write a program that counts in a given array of double values the number of occurrences of each value. Use Dictionary< TKey , TValue >.

Example: array = {3, 4, 4, -2.5, 3, 3, 4, 3, -2.5}

-2.5 → 2 times

3 → 4 times

4 → 3 times

2. Write a program that extracts from a given sequence of strings all elements that present in it odd number of times. Example:

{C#, SQL, PHP, PHP, SQL, SQL } → {C#, SQL}

3. Write a program that counts how many times each word from given text file `words.txt` appears in it. The character casing differences should be ignored. The result words should be ordered by their number of occurrences in the text. Example:

This is the TEXT. Text, text, text - THIS TEXT!
Is this the text?

is → 2

the → 2

this → 3

text → 6

4. Implement the data structure "hash table" in a class `HashTable<K,T>`. Keep the data in array of lists of key-value pairs (`LinkedList<KeyValuePair<K,T>>[]`) with initial capacity of 16. When the hash table load runs over 75%, perform resizing to 2 times larger capacity. Implement the following methods and properties: `Add(key, value)`, `Find(key) → value`, `Remove(key)`, `Count`, `Clear()`, `this[]`, `Keys`. Try to make the hash table to support iterating over its elements with `foreach`.
5. Implement the data structure "set" in a class `HashSet<T>` using your class `HashTable<K,T>` to hold the elements. Implement all standard set operations like `Add(T)`, `Find(T)`, `Remove(T)`, `Count`, `Clear()`, union and intersect.

6. * A text file `phones.txt` holds information about people, their town and phone number:

Mimi Shmatkata	Plovdiv	0888 12 34 56
Kireto	Varna	052 23 45 67
Daniela Ivanova Petrova	Karnobat	0899 999 888
Bat Gancho	Sofia	02 946 946 946

Duplicates can occur in people names, towns and phone numbers. Write a program to read the phones file and execute a sequence of commands given in the file `commands.txt`:

- `find(name)` – display all matching records by given name (first, middle, last or nickname)
- `find(name, town)` – display all matching records by given name and town

Free Trainings @ Telerik Academy

- ◆ C# Programming @ Telerik Academy

- ◆ csharpfundamentals.telerik.com



- ◆ Telerik Software Academy

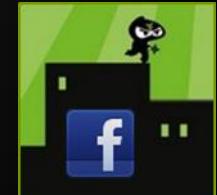
- ◆ academy.telerik.com

Telerik Academy

A large green rectangular graphic with a graduation cap icon at the top right. The word "Telerik" is written in white, and "Academy" is written below it in a slightly smaller font.

- ◆ Telerik Academy @ Facebook

- ◆ facebook.com/TelerikAcademy



- ◆ Telerik Software Academy Forums

- ◆ forums.academy.telerik.com

