Programming II

COMP 123

CENTENNIAL COLLEGE

Aderson Oliveira

Collections

CENTENNIAL COLLEGE

Aderson Oliveira

Objectives

- To Use **modern collection** types
  - `List<T>`
  - `Dictionary<TKey, TValue>`
- To **Search** collections data

Objectives

- To **<u>Modify</u>** collection's data
  - Add, Remove, Change

Arrays

- What is the limitation of Arrays?

List<T>

- List of Generics
- "T" for Type
- `System.Collections.Generic`

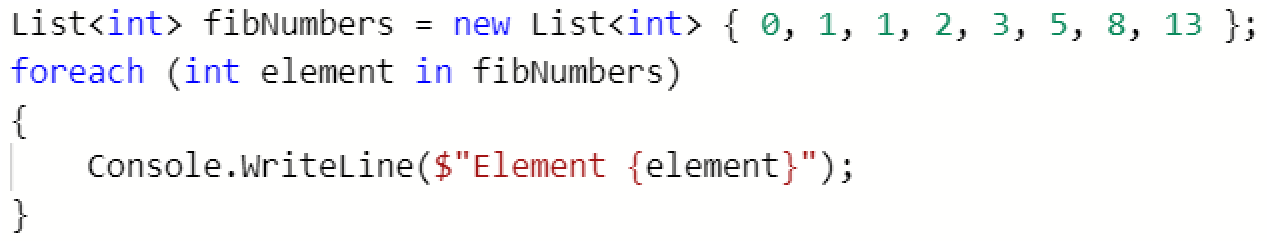
List<T>

- Inherits from
  - `IEnumerable`
  - `ICollection`
  - `IList`

CENTENNIAL COLLEGE

IEnumerable

- System.Collections
- Supports **foreach** statement



```
List<int> fibNumbers = new List<int> { 0, 1, 1, 2, 3, 5, 8, 13 };
foreach (int element in fibNumbers)
{
    Console.WriteLine($"Element {element}");
}
```

ICollection

- Inherits from **`IEnumerable`**
- Ensures that every collection supports a common way of getting the items in a collection

ICollection

- Members:
  - **CopyTo** – A way to copy the collection to an Array object
  - **Count** – Gets the number of items currently in the collection

IList

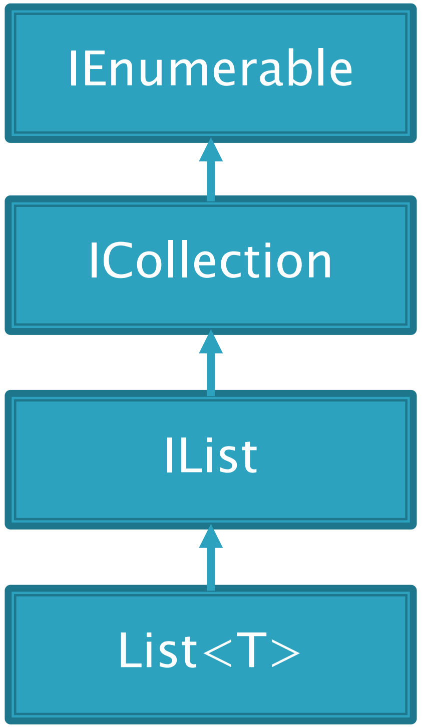
- Inherits from **`ICollection`**
- For simple list collections, the .NET Framework supports **`IList`** interface that is used to expose lists of items

IList

- Properties
  - IsFixedSize Gets an indicator of whether this collection can be resized
  - IsReadOnly Gets an indicator of whether a collection can be changed
  - Item Gets or sets the item at a specific index in the collection
- Methods
  - Add Adds an item to the collection
  - Clear Clears the collections of all items
  - Contains Tests whether a specific item is contained in the collection
  - IndexOf Finds an item in the collection, and returns the index of the item
  - Insert Adds an item at a specific index in the collection
  - Remove Removes the first occurrence of the specified object in the collection
  - RemoveAt Removes an item at a specific index in the collection

CENTENNIAL COLLEGE

List<T> Hierarchy



List<T> – Example

```
// create a new List of strings
            List<string> items = new List<string>();

            items.Add("red");
            items.Insert(0, "yellow");

            string colors = "";

            foreach (string item in items)
                colors += item + ",";
```

ListWindowsApp

List<T> – Properties and Methods

Method / Property	Usage
Add	Adds an element at the end of a List<T>.
AddRange	Adds elements of the specified collection at the end of a List<T>.
Capacity	Property that gets or sets the number of elements a List can store without resizing
Clear	Removes all the elements from a List<T>.
Contains	Checks whether the specified element exists or not in a List<T>.
Count	Property that returns the number of elements stored in the List
IndexOf	Returns the index of the first occurrence of the specified value in the list.
Insert	Inserts an element at the specified index in a List<T>.
Remove	Removes the first occurence of the specified element.
RemoveAt	Removes the element at the specified index.
RemoveRange	Removes all the elements that match with the supplied predicate function.
Sort	Sorts all the elements.
TrimExcess	Sets the capacity to the actual number of elements.

List<T> – Auto Resizes

- It doubles its capacity
- Performance tip
  - `TrimExcess()`

List<T> – When to use?

- A list can have **duplication**;
- It can **grow automatically**;
- Very **commonly used**;
- Better alternative to Array;

CENTENNIAL
COLLEGE

List<T> – When NOT to use?

- When you need **search efficiency**
- Use **`Dictionary`** instead

CENTENNIAL COLLEGE

YOUR TURN

Dictionary<TKey, TValue>

- List of Key/Value pairs
- Great performance when searching by Key
- Cannot have duplicated Keys
- `System.Collections.Generic`

Dictionary<TKey, TValue>

- Inherits from
  - `IEnumerable`
  - `ICollection`
  - `IDictionary`

IDictionary

- Inherits from **`ICollection`**
- Similar to the **`IList`** interface, but it does not allow access to items by index, only by key
- Gives access to the list of keys and values directly as collections of objects

IDictionary

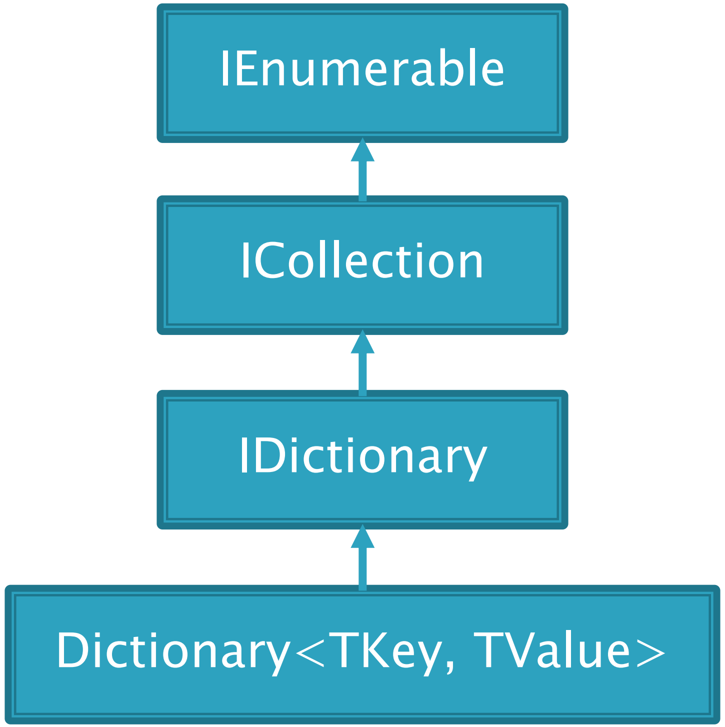
- Members
  - Properties

IsFixedSize	Gets an indicator of whether this collection can be resized
IsReadOnly	Gets an indicator of whether a collection can be changed
Item	Gets or sets the item at a specific element in the collection
Keys	Gets an ICollection object containing a list of the keys in the collection
Values	Gets an ICollection object containing a list of the values in the collection

  - Methods

Add	Adds a key/value pair to the collection.
Clear	Removes all items in the collections.
Contains	Tests whether a specific key is contained in the collection.
GetEnumerator	Returns an IDictionaryEnumerator object for the collection. This method is different than the IEnumerable interface that returns an IEnumerator interface.
Remove	Removes the item in the collection that corresponds to key.

Dictionary<TKey, TValue> Hierarchy



Dictionary – Example

```
Dictionary<int, string> groceryCollection = new Dictionary<int, string>();

 groceryCollection.Add(3, "Milk");
 groceryCollection.Add(6, "Eggs");
 groceryCollection.Add(4, "Coffe");
 groceryCollection.Add(5, "Juice");

// Display the keys and values.
foreach (int index in groceryCollection.Keys)
{
    // index is the Key
    // groceryCollection[index] is the Value is the key
 }
```

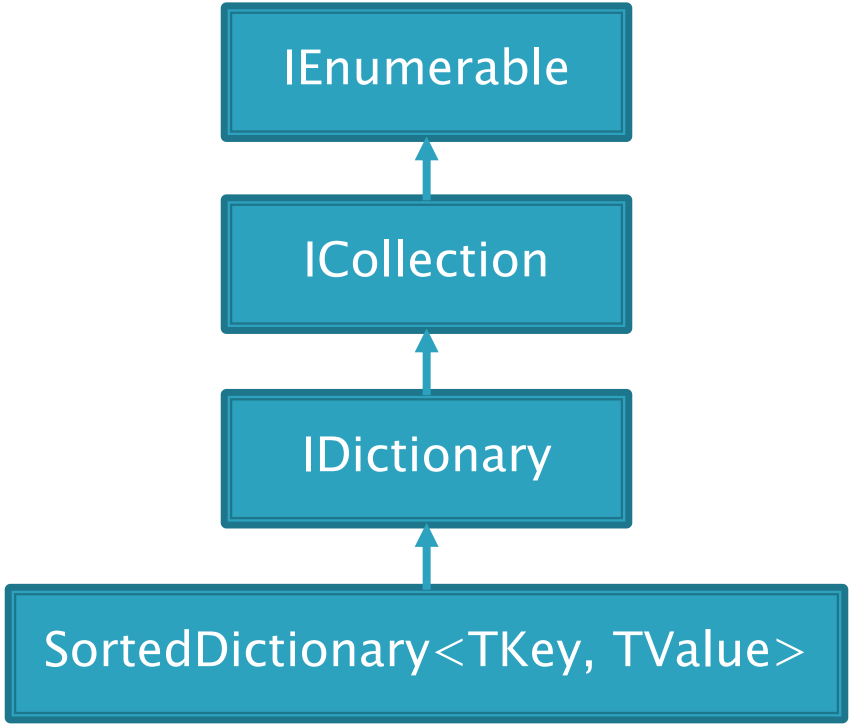
DictionaryWinApp

CENTENNIAL COLLEGE

SortedDictionary<TKey, TValue>

- Similar to Dictionary
- But Items are ordered by Key

SortedDictionary<TKey, TValue> Hierarchy



SortedDictionary – Example

```
SortedDictionary<int, string> groceryCollection =
                                  new SortedDictionary<int, string>();

 groceryCollection.Add(3, "Milk");
 groceryCollection.Add(6, "Eggs");
 groceryCollection.Add(4, "Coffe");
 groceryCollection.Add(5, "Juice");

// Display the keys and values.
foreach (int index in groceryCollection.Keys)
{
    // index is the Key
    // groceryCollection[index] is the Value is the key
 }
```

DictionaryWinApp

CENTENNIAL COLLEGE

SortedDictionary – Properties

Property	Usage
Count	Gets the number of key/value pairs contained in the SortedDictionary.
Item[TKey]	Gets or sets the value associated with the specified key.
Keys	Gets a collection containing the keys in the SortedDictionary.
Values	Gets a collection containing the values in the SortedDictionary.

SortedDictionary – Methods

Method	Usage
Add(TKey, TValue)	Adds an element with the specified key and value into the SortedDictionary.
Clear()	Removes all elements from the SortedDictionary.
ContainsKey(TKey)	Determines whether the SortedDictionary contains an element with the specified key.
ContainsValue(TValue)	Determines whether the SortedDictionary contains an element with the specified value.
Equals(Object)	Determines whether the specified object is equal to the current object.
GetEnumerator()	Returns an enumerator that iterates through the SortedDictionary.
GetHashCode()	Serves as the default hash function.
GetType()	Gets the Type of the current instance.
Remove(TKey)	Removes the element with the specified key from the SortedDictionary.
ToString()	Returns a string that represents the current object.
TryGetValue(TKey, TValue)	Gets the value associated with the specified key.

YOUR TURN

SortedDictionary
WinApp

CENTENNIAL COLLEGE

Other types of Collections

- `ArrayList`
  - Legacy collection list – Use **`List`** instead

Other types of Collections

- Stack
  - A LIFO (last in, first out) list where you push/pop records on top of each other.
- Queue
  - A FIFO (first in, first out) list where you push records on top and pop them off the bottom.

Summary

- There are **many ways** to store collection of data in C#
- **`List`**: More flexible way to collect data than **`Array`**

Summary

- **Dictionary**: Store a collection of Key/Value pairs for searching purposes
- **SortedDictionary**: Same as Dictionary, but the data is ordered by Key