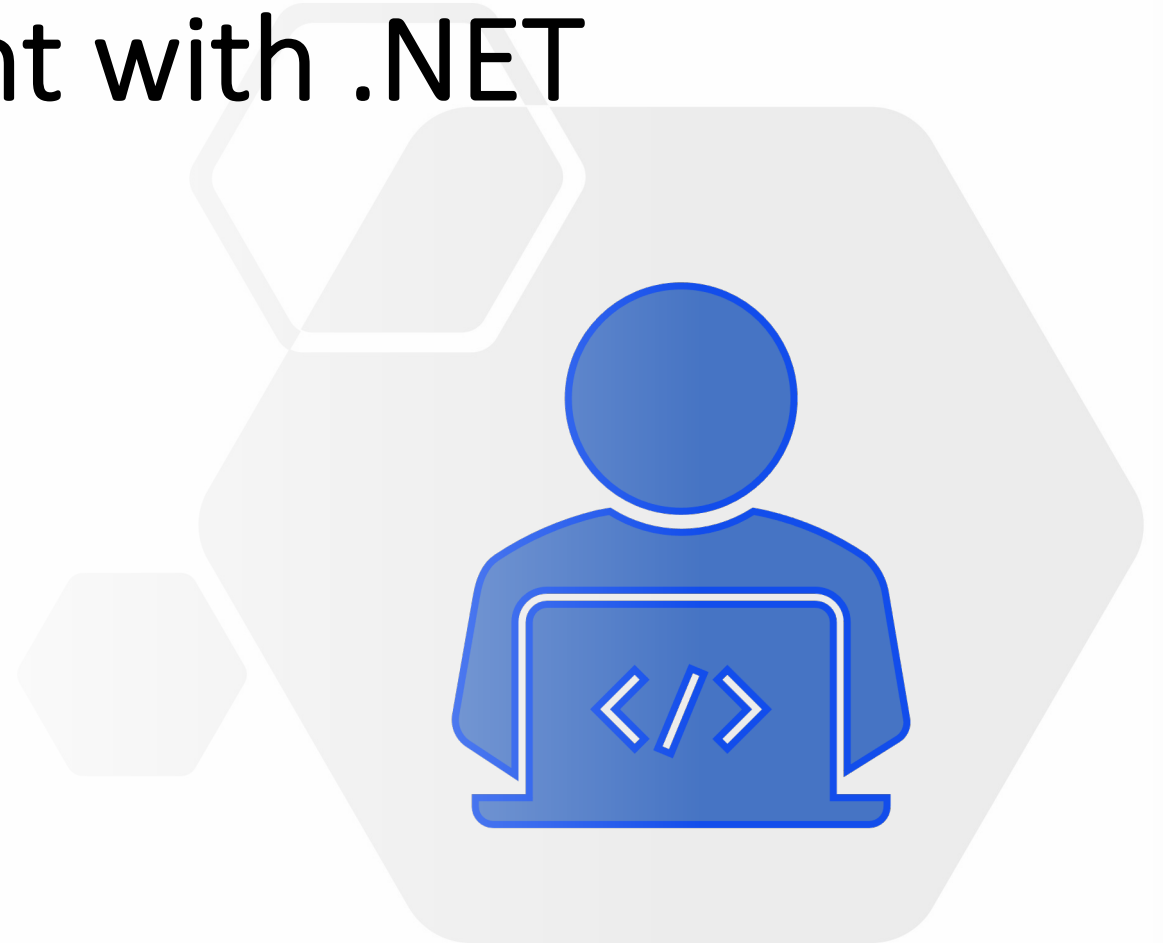


31927 32998: Application Development with .NET

Week-9 Lecture

Collections and Generics



Outline

- RichTextBox
- Collection Introduction
- Non-Generic Collections
- Generic Collections

RichTextbox Control

- RichTextBox control enables to display or edit text contents including paragraphs, images, tables, etc.
- Both TextBox and RichTextBox allows user to edit text, but RichTextBox is preferred when text formatting is required along with images, tables, etc.

RichTextbox Control

Useful Properties of RichTextBox:

- [SelectionFont](#): Gets or sets the font of the current text selection or insertion point. Useful for font related manipulations.
- [Font](#): Gets or sets the font of the text displayed by the control.
- [Text](#): Gets or sets the current text in the rich text box.

Useful methods of RichTextBox:

- [Clear\(\)](#): Clears all text from the text box control.
- [Copy\(\)](#): Copies the current selection in the text box to the **Clipboard**.
- [Cut\(\)](#): Moves the current selection in the text box to the **Clipboard**.
- [Paste\(\)](#): Replaces the current selection in the text box with the contents of the **Clipboard**.

Collections Introduction

- Collections are ways of storing retrieving data related data/objects.
- Collections are pre-packages data-structure classes- stores collections of data!
- Two ways:
 - Create array of Objects : Fixed Size
 - Create collections of Objects : Dynamically grow and shrink the group of objects
- The collection classes provide support for stacks, queues, lists, and hash tables.

Collections Introduction

- Different collections will have different performance properties.
- The C# language has two main type of collections:
 - **Non Generic (Standard) collections:** Found under [System.Collections](#) namespace
 - Class ArrayList
 - Class Stack
 - Class Hashtable
 - Class Queue
 - Class SortedList
 - **Generic collections:** Found under [System.Collections.Generic](#) namespace (preferred), added in the .Net Framework 2.0
 - List<T>
 - Queue<T>
 - Stack<T>
 - Dictionary<Tkey, Tvalue>
 - SortedList<Tkey, Tvalue>

Collections Introduction

Common Features :

1. All collections provide methods for adding, removing or finding items in collections
2. All collections that directly or indirectly implement `ICollection` interface or the `ICollection<T>` interface has:
 1. Ability to enumerate the collections
 2. Ability to copy the collection contents in an array using the `CopyTo` method
 3. Capacity (Number of elements it can contain) and Count (Actual number of elements) properties.
 4. Consistent lower bound: index of its first element.

Collections Introduction

- All collection classes implement some combination of the collection interfaces given below:

Interface	Description
IEnumerable	Has GetEnumerator method which returns IEnumerable object.
ICollection	Contains a Count property and a CopyTo method to copy collection contents to an array. Implements IEnumerable
ICollection	Provides indexer for accessing specific elements in the collection. Also has Add, Remove, Contains and IndexOf methods. Inherits ICollection
IDictionary	Defines a collection of key/value pairs. Provides an indexer, methods for modifying (e.g. Add, Remove). Keys and Values properties. Inherits ICollection

How to Choose a Collection

Generic Collection	Non-Generic Collection	When to use
Dictionary<Tkey, Tvalue>	HashTable	Store items as key/value pairs for a quick look-up by key
List<T>	Array ArrayList	Access items by index
Stack<T>	Stack	Use item in Last-in First Out (LIFO) order
Queue<T>	Queue	Use item in first-in-first-out (FIFO) order
LinkedList<T>		Accessing items sequentially

Non-generic Collections

- These collections are found in `System.Collections`
- The collections store `objects`. Since all classes and structs are derived from type `object`, using polymorphism, it can store any class and struct.
- A major disadvantage of these collections is that they are not type safe.
- The `System.Collections` namespace contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hash tables and dictionaries.

Non-generic Collections

- The following table lists some of the different types of collections in `System.Collections`

Collection Class	Implements	Description
ArrayList	ICollection	Like an array but shrinks or grows as data is deleted or added
HashTable	IDictionary	An unordered collection of key/value pairs
Queue	ICollection	First in, first out collection
Stack	ICollection	Last in, first out collection
SortedList	IDictionary	A sorted list of key/value pairs

Non-generic Collections

- The following table lists some of the different types of collections interfaces in `System.Collections`

Collection Interfaces	Description
ICollection	Defines size, enumerators, and synchronization methods for all non-generic collections.
ICollection	Defines size, enumerators, and synchronization methods for all non-generic collections.
IList	Represents a non-generic collection of objects that can be individually accessed by index.
IDictionary	Represents a non-generic collection of key/value pairs
IEnumerable	Exposes an enumerator, which supports a simple iteration over a non-generic collection.
IEnumerator	Supports a simple iteration over a non-generic collection.

Non-generic Collections: HashTable Class

- HashTable in C# represents a collection of Key/Value pairs.
- Maps Keys → Values
- Any Non Null object can be used as a Key
- Items in a HashTable are retrieved by Keys
- Syntax:

```
Hashtable <identifier> = new Hashtable();
```

Example:

```
Hashtable weekDays = new Hashtable();
```

Non-generic Collections: Hashtable Class

- Common Properties and Methods:

Property	Description
Count	Gets the number of key-and-value pairs contained in the Hashtable.
Item	Gets or sets the value associated with the specified key.
Keys	Gets an ICollection containing the keys in the Hashtable.
Values	Gets an ICollection containing the values in the Hashtable.
IsFixed	Gets a value indicating whether the Hashtable has a fixed size.

Method	Description
public virtual void Add(object key, object value)	Gets the number of key-and-value pairs contained in the Hashtable.
public virtual void Clear()	Removes all elements from the Hashtable.
public virtual void Remove(object key)	Removes the element with the specified key from the Hashtable.
public virtual bool ContainsValue(object value)	Determines whether the Hashtable contains a specific value.
public virtual bool ContainsKey(object key)	Determines whether the Hashtable contains a specific key.

Non-generic Collections: Hashtable Class

Example:

Output:

```
Friday is the last weekday!, Happy Weekend!

The Weekdays are:
Key: 1, Value: Monday
Key: 2, Value: Tuesday
Key: 3, Value: Wednesday
Key: 4, Value: Thursday
Key: 5, Value: Friday
```

```
namespace Week9ClassProgram
{
    0 references
    class HashtableDemo
    {
        0 references
        static void Main(string[] args)
        {
            // Creating a Hashtable
            Hashtable weekDays = new Hashtable();

            // Adding values as Key value pair
            weekDays.Add("1", "Monday");
            weekDays.Add("2", "Tuesday");
            weekDays.Add("3", "Wednesday");
            weekDays.Add("4", "Thursday");
            weekDays.Add("5", "Friday");
            weekDays.Add("6", "Saturday");
            weekDays.Add("7", "Sunday");

            // Check is a specific value is present
            if (weekDays.ContainsValue("Friday"))
            {
                Console.WriteLine("Friday is the last weekday!, Happy Weekend!");
                weekDays.Remove("6");
                weekDays.Remove("7");
            }

            // Retrieving the values and displaying
            // Get the collection of keys
            ICollection keys = weekDays.Keys;
            Console.WriteLine("\nThe Weekdays are:");
            foreach(string key in keys)
            {
                Console.WriteLine("Key: {0}, Value: {1}", key, weekDays[key]);
            }

            Console.ReadKey();
        }
    }
}
```

Non-generic Collections: Queue Class

Queue in C# represents a collection values arranged in First-in First-out order

Syntax:

```
Queue <identifier> = new Queue();
```

Example:

```
Queue ticketNumber = new Queue();
```


Non-generic Collections: Queue Class

- Common Properties and Methods:

Property	Description
Count	Gets the number of elements in the queue.

Method	Description
public virtual void Enqueue(object obj)	Adds an object to the end of the Queue.
public virtual void Clear()	Removes all elements from the Queue.
public virtual object Dequeue()	Removes and returns the object at the beginning of the Queue..
public virtual bool Contains (object value)	Determines whether an element is in the Queue.
public virtual object[] ToArray()	Copies the Queue to a new array

Non-generic Collections: Queue Class

Example:

```
namespace Week9ClassProgram
{
    0 references
    class QueueDemo
    {
        0 references
        static void Main(string[] args)
        {
            // Create a ticket queue for banks
            Queue ticketQueue = new Queue();

            // Add some ticket into the queue
            ticketQueue.Enqueue("120");
            ticketQueue.Enqueue("125");
            ticketQueue.Enqueue("129");
            ticketQueue.Enqueue("A");
            ticketQueue.Enqueue("B");
            ticketQueue.Enqueue("AB");

            // Display the current queue status
            foreach(string ticketno in ticketQueue)
            {
                Console.Write(ticketno + " ");
            }

            // Display the currently processed ticket
            Console.WriteLine("\n\nThe currently processed ticket is :{0}", ticketQueue.Dequeue().ToString());

            Console.ReadKey();
        }
    }
}
```

Output:

```
120 125 129 A B AB
```

```
The currently processed ticket is :120
```

Generic Collections

- Much better and type safe option for store and retrieve data
- Generic collection are available in the [System.Collections.Generic](#) namespace
- The [System.Collections.Generic](#) contains interfaces and classes that define generic collections.
- The performance is better than the non-generic collections.

Generic Collections

- The following table lists some of the commonly used types of Generic collections in `System.Collections.Generic`

Collection Class	Description
<code>LinkedList<T></code>	Represents a doubly LinkedList
<code>List<T></code>	Represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.
<code>Queue<T></code>	First in, first out generic collection
<code>Stack<T></code>	Last in, first out generic collection
<code>Dictionary<TKey,TValue></code>	Represents a collection of keys and values

Generic Collections

- The following table lists some of the commonly used types of Generic interfaces in `System.Collections.Generic`

Interface	Description
<code>ICollection<T></code>	Defines method to manipulate generic collections
<code>IComparer<T></code>	Defines a method that a type implements to compare two objects.
<code>IDictionary<K, V></code>	Represents a generic collection of key/value pairs.
<code>IEnumerable<T></code>	Exposes the enumerator, which supports a simple iteration over a collection of a specified type.
<code>IEqualityComparer<T></code>	Defines methods to support the comparison of objects for equality.
<code>IList<T></code>	Represents a collection of objects that can be individually accessed by index.

Generic Collections: Dictionary class

- Dictionary<Tkey, Tvalue> in C# represents a collection of Key/Value pairs.

Syntax:

1. Using the class:

```
Dictionary <int, int> <identifier> = new Dictionary<int, int>();
```

2. Using the interface

```
IDictionary <int, int> <identifier> = new IDictionary<int, int> ();
```

Example:

```
IDictionary <int, string> weekdays= new IDictionary<int, string> ();
```

Generic Collections: Dictionary class

- Common Properties and Methods:

Property	Description
Count	Gets the total number of elements exists in the Dictionary<TKey, TValue>.
Item	Gets or sets the element with the specified key in the Dictionary<TKey, TValue>.
Keys	Returns collection of keys of Dictionary<TKey, TValue>.
Values	Returns collection of values in Dictionary<TKey, TValue>

Method	Description
Add(TKey, TValue)	Adds the specified key and value to the dictionary.
Clear()	Removes all keys and values from the Dictionary<TKey,TValue>
Remove(TKey)	Removes the value with the specified key from the Dictionary<TKey,TValue>.
ContainsKey(TKey)	Determines whether the Dictionary<TKey,TValue> contains the specified key.
ContainsValue(TValue)	Determines whether the Dictionary<TKey,TValue> contains a specific value.

Generic Collections: Dictionary class

Output:

```
There are 7 items in the dictionary currently
Friday is the last week day!
Key: 1, Value: Monday
Key: 2, Value: Tuesday
Key: 3, Value: Wednesday
Key: 4, Value: Thursday
Key: 5, Value: Friday
The First day is Monday
```

```
namespace GenericDictionaryDemo
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Dictionary<int, string> weekdays = new Dictionary<int, string>();

            // Add values
            weekdays.Add(1, "Monday");
            weekdays.Add(2, "Tuesday");
            weekdays.Add(3, "Wednesday");
            weekdays.Add(4, "Thursday");
            weekdays.Add(5, "Friday");
            weekdays.Add(6, "Saturday");
            weekdays.Add(7, "Sunday");

            // Total entries in the dictionary
            Console.WriteLine("There are {0} items in the dictionary currently", weekdays.Count);

            // Check of an particular value
            if (weekdays.ContainsValue("Friday"))
            {
                Console.WriteLine("\nFriday is the last week day!");
                weekdays.Remove(6);
                weekdays.Remove(7);
            }

            // Display the dictionary entries
            foreach (KeyValuePair<int, string> items in weekdays)
            {
                Console.WriteLine("Key: {0}, Value: {1}", items.Key, items.Value);
            }

            // Accessing individual elements
            Console.WriteLine("\nThe First day is " + weekdays[1]);

            Console.ReadKey();
        }
    }
}
```


Generic Collections: LinkedList class

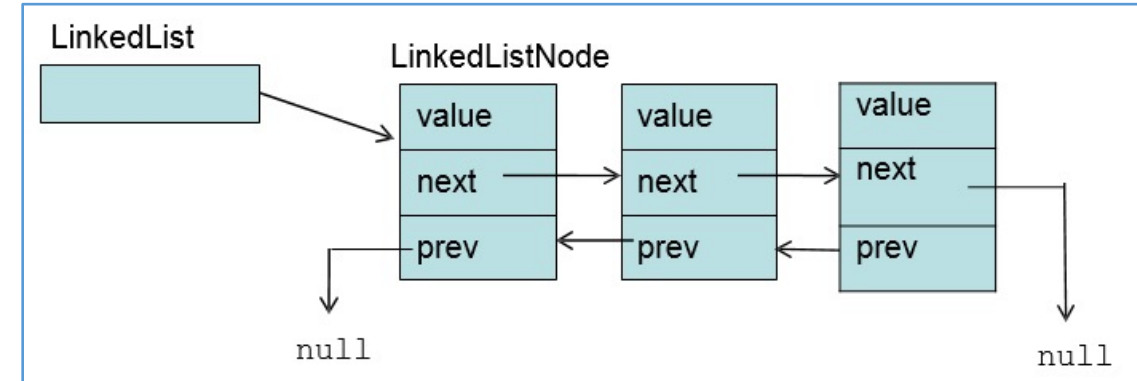
LinkedList<T> in C# represents doubly linked list.

Syntax:

```
LinkedList<T> <identifier> = new LinkedList<T>();
```

Example:

```
LinkedList<string> weekdays= new LinkedList<string> ();
```



Generic Collections: LinkedList class

- Common Properties and Methods:

Property	Description
Count	Gets the number of nodes actually contained in the <code>LinkedList<T></code> .
First	Gets the first node of the LinkedList<T>
Last	Gets the last node of the LinkedList<T>

Method	Description
AddAfter(LinkedListNode<T>, T)	Adds a new node containing the specified value after the specified existing node in the LinkedList<T>
AddBefore(LinkedListNode<T>, T)	Adds a new node containing the specified value before the specified existing node in the <code>LinkedList<T></code> .
AddFirst(LinkedListNode<T>)	Adds the specified new node at the start of the LinkedList<T>
AddLast(LinkedListNode<T>)	Adds the specified new node at the end of the LinkedList<T>
Clear()	Removes all nodes from the LinkedList<T> .
Find(T)	Finds the first node that contains the specified value.

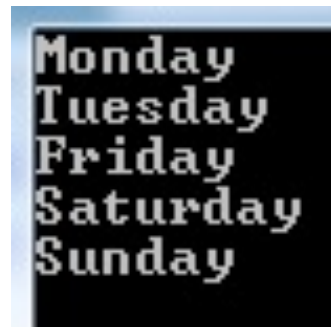
Generic Collections: LinkedList class

- Common Properties and Methods:

Method	Description
LinkedListNode<T> Findlast(T v)	Find last occurrence of v
bool Remove(T v)	Remove the first occurrence of v. Return bool on success
void Remove(LinkedListNode<t> n)	Remove n from list. Throws exception if not found
void removeFirst()	Removes first node in list
void RemoveLast()	Removes last node in list

Generic Collections: LinkedList class

Output:



Monday
Tuesday
Friday
Saturday
Sunday

```
namespace Week9CallProgram
{
    0 references
    class LinkedListDemo
    {
        0 references
        static void Main(string[] args)
        {
            // Create a LinkedList
            LinkedList<string> weekdays = new LinkedList<string>();

            // Add data into the list
            weekdays.AddFirst("Monday");
            weekdays.AddLast("Saturday");

            // Add "Tuesday" after "Monday"
            LinkedListNode<string> current = weekdays.Find("Monday");
            weekdays.AddAfter(current, "Tuesday");

            current = weekdays.Find("Saturday");
            weekdays.AddAfter(current, "Sunday");
            weekdays.AddBefore(current, "Friday");

            // Display items in LinkedList
            foreach (string items in weekdays)
            {
                Console.WriteLine(items);
            }

            Console.ReadKey();
        }
    }
}
```