List<T> Class

Represents a strongly typed list of objects that can be accessed by index. Provides methods to search, sort, and manipulate lists.

Queue<T> Class

Represents a first-in, first-out collection of objects.

# Stack<T> Class

Represents a variable size last-in-first-out (LIFO) collection of instances of the same specified type.

An interface contains only the signatures of [methods](https://msdn.microsoft.com/en-us/library/ms173114.aspx), [properties](https://msdn.microsoft.com/en-us/library/x9fsa0sw.aspx), [events](https://msdn.microsoft.com/en-us/library/awbftdfh.aspx) or [indexers](https://msdn.microsoft.com/en-us/library/6x16t2tx.aspx). A class or struct that implements the interface must implement the members of the interface that are specified in the interface definition.

interface ISampleInterface

{

void SampleMethod();

}

class ImplementationClass : ISampleInterface

{

// Explicit interface member implementation:

void ISampleInterface.SampleMethod()

{

// Method implementation.

}

static void Main()

{

// Declare an interface instance.

ISampleInterface obj = new ImplementationClass();

// Call the member.

obj.SampleMethod();

}

}

Abstract classes are closely related to interfaces. They are classes that cannot be instantiated, and are frequently either partially implemented, or not at all implemented. One key difference between abstract classes and interfaces is that a class may implement an unlimited number of interfaces, but may inherit from only one abstract class

class MyWashingMachine : WashingMachine

{

public MyWashingMachine()

{

// Initialization code goes here.

}

override public void Wash()

{

// Wash code goes here.

}

override public void Rinse(int loadSize)

{

// Rinse code goes here.

}

override public long Spin(int speed)

{

// Spin code goes here.

}

}

abstract class WashingMachine

{

public WashingMachine()

{

// Code to initialize the class goes here.

}

abstract public void Wash();

abstract public void Rinse(int loadSize);

abstract public long Spin(int speed);

}

// Declare the English units interface:

static void Main()

{

// Declare a class instance box1:

Box box1 = new Box(30.0f, 20.0f);

// Declare an instance of the English units interface:

IEnglishDimensions eDimensions = (IEnglishDimensions)box1;

// Declare an instance of the metric units interface:

IMetricDimensions mDimensions = (IMetricDimensions)box1;

// Print dimensions in English units:

System.Console.WriteLine("Length(in): {0}", eDimensions.Length());

System.Console.WriteLine("Width (in): {0}", eDimensions.Width());

// Print dimensions in metric units:

System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());

System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());

}

}

/\* Output:

Length(in): 30

Width (in): 20

Length(cm): 76.2

Width (cm): 50.8

\*/

interface IEnglishDimensions

{

float Length();

float Width();

}

// Declare the metric units interface:

interface IMetricDimensions

{

float Length();

float Width();

}

// Declare the Box class that implements the two interfaces:

// IEnglishDimensions and IMetricDimensions:

class Box : IEnglishDimensions, IMetricDimensions

{

float lengthInches;

float widthInches;

public Box(float length, float width)

{

lengthInches = length;

widthInches = width;

}

// Explicitly implement the members of IEnglishDimensions:

float IEnglishDimensions.Length()

{

return lengthInches;

}

float IEnglishDimensions.Width()

{

return widthInches;

}

// Explicitly implement the members of IMetricDimensions:

float IMetricDimensions.Length()

{

return lengthInches \* 2.54f;

}

float IMetricDimensions.Width()

{

return widthInches \* 2.54f;

}

namespace test

{

interface IStorable

{

void Read( );

void Write(object obj);

int Status { get; set; }

}

interface ICompressible

{

void Compress( );

void Decompress( );

}

public class Note : IStorable

{

private string myString;

public Note(string theString)

{

myString = theString;

}

public override string ToString( )

{

return myString;

}

#region IStorable

public void Read( )

{

Console.WriteLine("Executing Note's Read Method for IStorable");

}

public void Write(object o)

{

Console.WriteLine("Executing Note's Write Method for IStorable");

}

public int Status { get; set; }

#endregion // IStorable

}

public class Document : Note, ICompressible

{

private int documentID;

public int ID

{

get { return this.documentID; }

}

public Document(string docString, int documentID) : base(docString)

{

this.documentID = documentID;

}

#region ICompressible

public void Compress( )

{

Console.WriteLine("Executing Document's Compress Method for ICompressible");

}

public void Decompress( )

{

Console.WriteLine("Executing Document's Decompress Method for ICompressible");

}

#endregion // ICompressible

} // end Document class

class Tester

{

public void Run( )

{

string testString = "String ";

Note[] myNoteArray = new Note[3];

for (int i = 0; i < 3; i++)

{

string docText = testString + i.ToString( );

if (i % 2 == 0)

{

Document myDocument = new Document(docText, (i + 5) \* 10);

myNoteArray[i] = myDocument;

}

else

{

Note myNote = new Note(docText);

myNoteArray[i] = myNote;

}

}

foreach (Note theNote in myNoteArray)

{

Console.WriteLine("\nTesting {0} with IS", theNote);

theNote.Read( ); // all notes can do this

if (theNote is ICompressible)

{

ICompressible myCompressible = theNote as ICompressible;

myCompressible.Compress( );

}

else

{

Console.WriteLine("This storable object is not compressible.");

}

if (theNote is Document)

{

Document myDoc = theNote as Document;

// clean cast

myDoc = theNote as Document;

Console.WriteLine("my documentID is {0}", myDoc.ID);

}

}

foreach (Note theNote in myNoteArray)

{

Console.WriteLine("\nTesting {0} with AS", theNote);

ICompressible myCompressible = theNote as ICompressible;

if (myCompressible != null)

{

myCompressible.Compress( );

}

else

{

Console.WriteLine("This storable object is not compressible.");

} // end else

Document theDoc = theNote as Document;

if (theDoc != null)

{

Console.WriteLine("My documentID is {0}",

((Document)theNote).ID);

}

else

{

Console.WriteLine("Not a document.");

}

}

}

static void Main( )

{

Tester t = new Tester( );

t.Run( );

}

}

}

public class Example

{

public static void Main()

{

List<string> dinosaurs = new List<string>();

Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);

dinosaurs.Add("Tyrannosaurus");

dinosaurs.Add("Amargasaurus");

dinosaurs.Add("Mamenchisaurus");

dinosaurs.Add("Deinonychus");

dinosaurs.Add("Compsognathus");

dinosaurs.Add("Compsognathus2");

dinosaurs.Add("Compsognathus3");

dinosaurs.Add("Compsognathus4");

dinosaurs.Add("Compsognathus5");

Console.WriteLine();

foreach (string dinosaur in dinosaurs)

{

Console.WriteLine(dinosaur);

}

Console.WriteLine("\nCapacity: {0}", dinosaurs.Capacity);

Console.WriteLine("Count: {0}", dinosaurs.Count);

Console.WriteLine("\nContains(\"Deinonychus\"): {0}",

dinosaurs.Contains("Deinonychus"));

Console.WriteLine("\nInsert(2, \"Compsognathus\")");

dinosaurs.Insert(2, "Compsognathus");

Console.WriteLine();

foreach (string dinosaur in dinosaurs)

{

Console.WriteLine(dinosaur);

}

// Shows accessing the list using the Item property.

Console.WriteLine("\ndinosaurs[3]: {0}", dinosaurs[3]);

Console.WriteLine("\nRemove(\"Compsognathus\")");

dinosaurs.Remove("Compsognathus");

Console.WriteLine();

foreach (string dinosaur in dinosaurs)

{

Console.WriteLine(dinosaur);

}

dinosaurs.TrimExcess();

Console.WriteLine("\nTrimExcess()");

Console.WriteLine("Capacity: {0}", dinosaurs.Capacity);

Console.WriteLine("Count: {0}", dinosaurs.Count);

dinosaurs.Clear();

Console.WriteLine("\nClear()");

Console.WriteLine("Capacity: {0}", dinosaurs.Capacity);

Console.WriteLine("Count: {0}", dinosaurs.Count);

}

}

class Example

{

public static void Main()

{

Queue<string> numbers = new Queue<string>();

numbers.Enqueue("one");

numbers.Enqueue("two");

numbers.Enqueue("three");

numbers.Enqueue("four");

numbers.Enqueue("five");

// A queue can be enumerated without disturbing its contents.

foreach (string number in numbers)

{

Console.WriteLine(number);

}

Console.WriteLine("\nDequeuing '{0}'", numbers.Dequeue());

Console.WriteLine("Peek at next item to dequeue: {0}", numbers.Peek());

Console.WriteLine("Dequeuing '{0}'", numbers.Dequeue());

Console.WriteLine("Dequeuing '{0}'", numbers.Dequeue());

// Create a copy of the queue, using the ToArray method and the

// constructor that accepts an IEnumerable<T>.

Queue<string> queueCopy = new Queue<string>(numbers.ToArray());

Console.WriteLine("\nContents of the first copy:");

foreach (string number in queueCopy)

{

Console.WriteLine(number);

}

// Create an array twice the size of the queue and copy the

// elements of the queue, starting at the middle of the

// array.

string[] array2 = new string[numbers.Count \* 2];

numbers.CopyTo(array2, numbers.Count);

// Create a second queue, using the constructor that accepts an

// IEnumerable(Of T).

Queue<string> queueCopy2 = new Queue<string>(array2);

Console.WriteLine("\nContents of the second copy, with duplicates and nulls:");

foreach (string number in queueCopy2)

{

Console.WriteLine(number);

}

Console.WriteLine("\nqueueCopy.Contains(\"four\") = {0}",

queueCopy.Contains("four"));

Console.WriteLine("\nqueueCopy.Clear()");

queueCopy.Clear();

Console.WriteLine("\nqueueCopy.Count = {0}", queueCopy.Count);

}

}

class Example

{

public static void Main()

{

Stack<string> numbers = new Stack<string>();

numbers.Push("one");

numbers.Push("two");

numbers.Push("three");

numbers.Push("four");

numbers.Push("five");

// A stack can be enumerated without disturbing its contents.

foreach (string number in numbers)

{

Console.WriteLine(number);

}

Console.WriteLine("\nPopping '{0}'", numbers.Pop());

Console.WriteLine("Peek at next item to destack: {0}",

numbers.Peek());

Console.WriteLine("Popping '{0}'", numbers.Pop());

// Create a copy of the stack, using the ToArray method and the

// constructor that accepts an IEnumerable<T>.

Stack<string> stack2 = new Stack<string>(numbers.ToArray());

Console.WriteLine("\nContents of the first copy:");

foreach (string number in stack2)

{

Console.WriteLine(number);

}

// Create an array twice the size of the stack and copy the

// elements of the stack, starting at the middle of the

// array.

string[] array2 = new string[numbers.Count \* 2];

numbers.CopyTo(array2, numbers.Count);

// Create a second stack, using the constructor that accepts an

// IEnumerable(Of T).

Stack<string> stack3 = new Stack<string>(array2);

Console.WriteLine("\nContents of the second copy, with duplicates and nulls:");

foreach (string number in stack3)

{

Console.WriteLine(number);

}

Console.WriteLine("\nstack2.Contains(\"four\") = {0}",

stack2.Contains("four"));

Console.WriteLine("\nstack2.Clear()");

stack2.Clear();

Console.WriteLine("\nstack2.Count = {0}", stack2.Count);

}

}

class Example

{

public static void Main()

{

MediaStorage myMediaStorage = new MediaStorage();

// instantiate the two media players

AudioPlayer myAudioPlayer = new AudioPlayer();

VideoPlayer myVideoPlayer = new VideoPlayer();

// instantiate the delegates

MediaStorage.PlayMedia audioPlayerDelegate = new MediaStorage.PlayMedia(myAudioPlayer.PlayAudioFile);

MediaStorage.PlayMedia videoPlayerDelegate = new MediaStorage.PlayMedia(myVideoPlayer.PlayVideoFile);

// call the delegates

myMediaStorage.ReportResult(audioPlayerDelegate);

myMediaStorage.ReportResult(videoPlayerDelegate);

}

public class MediaStorage

{

public delegate int PlayMedia();

public void ReportResult(PlayMedia playerDelegate)

{

if (playerDelegate() == 0)

Console.WriteLine("Media played successfully.");

else

Console.WriteLine("Media did not play successfully.");

}

}

public class AudioPlayer

{

private int audioPlayerStatus;

public int PlayAudioFile()

{

Console.WriteLine("Simulating playing an audio file here.");

audioPlayerStatus = 0;

return audioPlayerStatus;

}

}

public class VideoPlayer

{

private int videoPlayerStatus;

public int PlayVideoFile()

{

Console.WriteLine("Simulating a failed video file here.");

videoPlayerStatus = -1;

return videoPlayerStatus;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace test

{

class Example

{

public static void Main()

{

// create a new clock

Clock theClock = new Clock();

// create the display and tell it to subscribe to the clock just created

DisplayClock dc = new DisplayClock();

dc.Subscribe(theClock);

// create a Log object and tell it to subscribe to the clock

LogCurrentTime lct = new LogCurrentTime();

lct.Subscribe(theClock);

// Get the clock started

theClock.Run();

}

// a class to hold the information about the event

// in this case it will hold only information

// available in the clock class, but could hold

// additional state information

public class TimeInfoEventArgs : EventArgs

{

public int hour;

public int minute;

public int second;

public TimeInfoEventArgs(int hour, int minute, int second)

{

this.hour = hour;

this.minute = minute;

this.second = second;

}

}

// The publisher: the class that other classes

// will observe. This class publishes one delegate:

// SecondChangeHandler.

public class Clock

{

private int hour;

private int minute;

private int second;

// the delegate the subscribers must implement

public delegate void SecondChangeHandler(Object clock, TimeInfoEventArgs timeInformation);

// an instance of the delegate

public SecondChangeHandler SecondChanged;

// set the clock running

// it will raise an event for each new second

public void Run()

{

for (; ; )

{

Thread.Sleep(100); // sleep 100 milliseconds

System.DateTime dt = System.DateTime.Now; // get the current time

if (dt.Second != second) // if the second has changed notify the subscribers

{

// create the TimeInfoEventArgs object to pass to the subscriber

TimeInfoEventArgs timeInformation =

new TimeInfoEventArgs(dt.Hour, dt.Minute, dt.Second);

if (SecondChanged != null) // if anyone has subscribed, notify them

SecondChanged(this, timeInformation);

}

this.second = dt.Second; // update the state

this.minute = dt.Minute;

this.hour = dt.Hour;

}

}

}

// A subscriber: DisplayClock subscribes to the

// clock's events. The job of DisplayClock is

// to display the current time

public class DisplayClock

{

// given a clock, subscribe to its SecondChangeHandler event

public void Subscribe(Clock theClock)

{

theClock.SecondChanged += new Clock.SecondChangeHandler(TimeHasChanged);

}

// the method that implements the delegated functionality

public void TimeHasChanged(Object theClock, TimeInfoEventArgs ti)

{

Console.WriteLine("Current Time: {0}:{1}:{2}", ti.hour, ti.minute, ti.second);

}

}

// a second subscriber whose job is to write to a file

public class LogCurrentTime

{

public void Subscribe(Clock theClock)

{

theClock.SecondChanged += new Clock.SecondChangeHandler(WriteLogEntry);

}

// this method should write to a file

// we write to the console to see the effect this object keeps no state

public void WriteLogEntry(Object theClock, TimeInfoEventArgs ti)

{

Console.WriteLine("Logging to file: {0}:{1}:{2}", ti.hour, ti.minute, ti.second);

}

}

}

}

// create the data connection

string connectionString =

"server=.\\sqlexpress;Trusted\_Connection=yes;database=Northwind";

// create the string to hold the SQL command to get records from the Customers table

string commandString = "Select CompanyName, ContactName from Customers";

// create the data adapter with the connection string and command

SqlDataAdapter myDataAdapter = new SqlDataAdapter(commandString, connectionString);

// Create and fill the DataSet object

DataSet myDataSet = new DataSet();

myDataAdapter.Fill(myDataSet);

// Retrieve the Customers table

DataTable myDataTable = myDataSet.Tables[0];

// iterate over the rows collection and output the fields

foreach (DataRow dataRow in myDataTable.Rows)

{

Console.WriteLine("CompanyName: {0}. Contact: {1}", dataRow["CompanyName"],

dataRow["ContactName"]);

}

public class Book

{

public string Title { get; set; }

public string Author { get; set; }

public string Publisher { get; set; }

public int PublicationYear { get; set; }

}

List<Book> bookList = new List<Book>

{

new Book { Title = "Learning C# 3.0", Author = "Jesse Liberty",

Publisher = "O'Reilly", PublicationYear = 2008 },

new Book { Title = "Programming C# 3.0", Author = "Jesse Liberty",

Publisher = "O'Reilly", PublicationYear = 2008 },

new Book { Title = "C# 3.0 Cookbook", Author = "Jay Hilyard",

Publisher = "O'Reilly", PublicationYear = 2007 },

new Book { Title = "C# 3.0 in a Nutshell", Author = "Ben Albahari",

Publisher = "O'Reilly", PublicationYear = 2007 },

};

IEnumerable<Book> resultsAuthor =

from testBook in bookList

where testBook.Author == "Jesse Liberty"

select testBook;

foreach (Book testBook in resultsAuthor)

{

Console.WriteLine("{0}, by {1}", testBook.Title, testBook.Author);

}

var resultsAuthor =

from testBook in bookList

where testBook.Author == "Jesse Liberty"

select new { testBook.Title, testBook.Author };

foreach (var testBook in resultsAuthor)

{

Console.WriteLine("{0}, by {1}", testBook.Title, testBook.Author);

}

var resultsAuthor =

bookList.Where(bookEval => bookEval.Author == "Jesse Liberty");

var resultList = //or as a lambda expression

from myBook in bookList var resultList =

orderby myBook.Author bookList.OrderBy(bookEval => bookEval.Author);

select myBook;

var resultList =

from myBook in bookList

join myOrder in orderList on myBook.Title equals myOrder.Title

where myOrder.Quantity >= 50

select new { myBook.Title, myBook.Author, myOrder.Quantity };

var resultList =

bookList.Join(orderList,

book => book.Title,

order => order.Title,

(book , order) => new {book.Title, book.Author, order.Quantity})

.Where(order => order.Quantity >= 50);

private void PMPhotoListBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

ListBox lb = sender as ListBox;

if (lb != null)

{

if (lb.SelectedItem != null)

{

string chosenName = (lb.SelectedItem as ImageURL).Name.ToString();

Title = chosenName;

}

}

else

{

throw new ArgumentException("Expected ListBox to call selection changed in "+

"PresPhotoListBox\_SelectionChanged");

}

}

void PrintText(object sender, SelectionChangedEventArgs args)

{

ListBoxItem lbi = ((sender as ListBox).SelectedItem as ListBoxItem);

tb.Text = " You selected " + lbi.Content.ToString() + ".";

}

<Window x:Class="WpfTutorialSamples.ListBox\_control.ListBoxDataBindingSample"  
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
        Title="ListBoxDataBindingSample" Height="150" Width="300">  
    <Grid Margin="10">  
                <ListBox Name="lbTodoList" HorizontalContentAlignment="Stretch">  
                        <ListBox.ItemTemplate>  
                                <DataTemplate>  
                                        <Grid Margin="0,2">  
                                                <Grid.ColumnDefinitions>  
                                                        <ColumnDefinition Width="\*" />  
                                                        <ColumnDefinition Width="100" />  
                                                </Grid.ColumnDefinitions>  
                                                <TextBlock Text="{Binding Title}" />  
                                                <ProgressBar Grid.Column="1" Minimum="0" Maximum="100" Value="{Binding Completion}" />  
                                        </Grid>  
                                </DataTemplate>  
                        </ListBox.ItemTemplate>  
                </ListBox>  
        </Grid>  
</Window>

namespace WpfTutorialSamples.ListBox\_control  
{  
        public partial class ListBoxDataBindingSample : Window  
        {  
                public ListBoxDataBindingSample()  
                {  
                        InitializeComponent();  
                        List<TodoItem> items = new List<TodoItem>();  
                        items.Add(new TodoItem() { Title = "Complete this WPF tutorial", Completion = 45 });  
                        items.Add(new TodoItem() { Title = "Learn C#", Completion = 80 });  
                        items.Add(new TodoItem() { Title = "Wash the car", Completion = 0 });  
                        lbTodoList.ItemsSource = items;  
                }  
        }  
        public class TodoItem  
        {  
                public string Title { get; set; }  
                public int Completion { get; set; }  
        }  
}