Long Le Information Technology

Task 10 Object-Oriented Programming

14/3/2018

1. **Theory Part:**

Collection types are the common variations of data collections, such as hash tables, queues, stacks, bags, dictionaries, and lists.

Collections are based on the [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection) interface, the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist) interface, the [IDictionary](https://docs.microsoft.com/en-us/dotnet/api/system.collections.idictionary) interface, or their generic counterparts. The [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist) interface and the [IDictionary](https://docs.microsoft.com/en-us/dotnet/api/system.collections.idictionary) interface are both derived from the [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection) interface; therefore, all collections are based on the [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection) interface either directly or indirectly. In collections based on the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist) interface (such as [Array](https://docs.microsoft.com/en-us/dotnet/api/system.array), [ArrayList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.arraylist), or [List<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1)) or directly on the [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection) interface (such as [Queue](https://docs.microsoft.com/en-us/dotnet/api/system.collections.queue), [ConcurrentQueue<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentqueue-1), [Stack](https://docs.microsoft.com/en-us/dotnet/api/system.collections.stack), [ConcurrentStack<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentstack-1) or [LinkedList<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.linkedlist-1)), every element contains only a value. In collections based on the [IDictionary](https://docs.microsoft.com/en-us/dotnet/api/system.collections.idictionary) interface (such as the [Hashtable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.hashtable) and [SortedList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.sortedlist) classes, the [Dictionary<TKey,TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2) and [SortedList<TKey,TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.sortedlist-2) generic classes), or the [ConcurrentDictionary<TKey,TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentdictionary-2) classes, every element contains both a key and a value. The [KeyedCollection<TKey,TItem>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.keyedcollection-2) class is unique because it is a list of values with keys embedded within the values and, therefore, it behaves like a list and like a dictionary.

Generic collections are the best solution to strong typing. However, if your language does not support generics, the [System.Collections](https://docs.microsoft.com/en-us/dotnet/api/system.collections) namespace includes base collections, such as [CollectionBase](https://docs.microsoft.com/en-us/dotnet/api/system.collections.collectionbase), [ReadOnlyCollectionBase](https://docs.microsoft.com/en-us/dotnet/api/system.collections.readonlycollectionbase), and [DictionaryBase](https://docs.microsoft.com/en-us/dotnet/api/system.collections.dictionarybase), which are abstract base classes that can be extended to create collection classes that are strongly typed. When efficient multi-threaded collection access is required, use the generic collections in the [System.Collections.Concurrent](https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent) namespace.1

Collections can vary, depending on how the elements are stored, how they are sorted, how searches are performed, and how comparisons are made. The [Queue](https://docs.microsoft.com/en-us/dotnet/api/system.collections.queue) class and the [Queue<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.queue-1) generic class provide first-in-first-out lists, while the [Stack](https://docs.microsoft.com/en-us/dotnet/api/system.collections.stack) class and the [Stack<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.stack-1) generic class provide last-in-first-out lists. The [SortedList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.sortedlist)class and the [SortedList<TKey,TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.sortedlist-2) generic class provide sorted versions of the [Hashtable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.hashtable) class and the [Dictionary<TKey,TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2) generic class. The elements of a [Hashtable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.hashtable) or a [Dictionary<TKey,TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2) are accessible only by the key of the element, but the elements of a [SortedList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.sortedlist) or a [KeyedCollection<TKey,TItem>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.keyedcollection-2) are accessible either by the key or by the index of the element. The indexes in all collections are zero-based, except [Array](https://docs.microsoft.com/en-us/dotnet/api/system.array), which allows arrays that are not zero-based.

The LINQ to Objects feature allows you to use LINQ queries to access in-memory objects as long as the object type implements [IEnumerable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable) or [IEnumerable<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.ienumerable-1). LINQ queries provide a common pattern for accessing data; are typically more concise and readable than standard foreach loops; and provide filtering, ordering and grouping capabilities. LINQ queries can also improve performance. For more information, see [LINQ to Objects](http://msdn.microsoft.com/library/73cafe73-37cf-46e7-bfa7-97c7eea7ced9)and [Parallel LINQ (PLINQ)](https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/parallel-linq-plinq).

1. **Practical Part**

**Source code:**

1. **Xaml.cs:**

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

List<SalesPerson> salespersons = new List<SalesPerson>()

{

{new SalesPerson("Stan Still", new DateTime(1976, 03, 23), 123)},

{new SalesPerson("Joe Bloggs", new DateTime(1989, 11, 27), 156)},

{new SalesPerson("Luke Warm", new DateTime(1991, 07, 9), 147)},

{new SalesPerson("Norman Knight", new DateTime(1959, 10, 16), 139)},

{new SalesPerson("Susan Wynns", new DateTime(1963, 11, 26), 239)},

{new SalesPerson("Clarence Sales", new DateTime(1977, 05, 01), 149)},

{new SalesPerson("Lynn Seed", new DateTime(1982, 06, 23), 141)},

{new SalesPerson("Barry More", new DateTime(1991, 12, 24), 135)},

{new SalesPerson("Cookie Baker", new DateTime(1969, 04, 12), 155)},

{new SalesPerson("Dawn Bright", new DateTime(1978, 01, 18), 144)}

};

List<Manager> managers = new List<Manager>()

{

{new Manager("Rocky Rhoades", new DateTime(1988, 06, 25), 1230)},

{new Manager("Bud Light", new DateTime(1969, 08, 01), 1450)},

{new Manager("Dick Durns", new DateTime(1974, 12, 13), 1510)},

{new Manager("Bob Paisley", new DateTime(1979, 09, 15), 1095)},

{new Manager("Frank Furter", new DateTime(1958, 09, 14), 1190)},

{new Manager("Wanda Round", new DateTime(1975, 08, 11), 1220)},

{new Manager("Sunny Day", new DateTime(1980, 11, 25), 1040)},

{new Manager("Joy Daley", new DateTime(1992, 02, 28), 1235)},

{new Manager("Hooker Crooke", new DateTime(1995, 11, 11), 1245)},

{new Manager("June McBride", new DateTime(1994, 05, 04), 1075)}

};

public MainWindow()

{

InitializeComponent();

}

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

{

Display.ItemsSource = null;

Display.Columns[2].Header = string.Empty;

if (DisplayChoice.SelectedIndex == 1)

{

Display.ItemsSource = salespersons;

Display.Columns[2].Header = "Sales num";

((DataGridTextColumn)Display.Columns[2]).Binding = new Binding("SalesNumber");

}

if (DisplayChoice.SelectedIndex == 2)

{

Display.ItemsSource = managers;

Display.Columns[2].Header = "Stock opt";

((DataGridTextColumn)Display.Columns[2]).Binding = new Binding("StockOption");

}

}

private void AddNew\_Click(object sender, RoutedEventArgs e)

{

if (NameInformation.Text.Trim() == "" || DOBInformation.Text.Trim() == "" || (SalesPersonRB.IsChecked == false && ManagerRB.IsChecked == false)

|| (BonusInformation.Text.Trim() == "" && StockInformation.Text.Trim() == ""))

{

MessageBox.Show("Not enough info");

}

else

{

DateTime nextDOB;

int nextBonus = 0, nextOption = 0;

string errorMes = "";

if (!DateTime.TryParse(DOBInformation.Text.Trim(), out nextDOB))

{

errorMes += "Not valid";

}

if (SalesPersonRB.IsChecked == true && BonusInformation.Text.Trim() == "" && !Int32.TryParse(BonusInformation.Text.Trim(), out nextBonus))

{

errorMes += "NOT valid";

}

if (ManagerRB.IsChecked == true && StockInformation.Text.Trim() == "" && !Int32.TryParse(StockInformation.Text.Trim(), out nextOption))

{

errorMes += "Not VALID";

}

if (errorMes != "")

{

MessageBox.Show(errorMes);

}

else

{

if (SalesPersonRB.IsChecked == true)

{

salespersons.Add(new SalesPerson(NameInformation.Text.Trim(), nextDOB, nextBonus));

MessageBox.Show("Salesperson added!");

}

else

{

managers.Add(new Manager(NameInformation.Text.Trim(), nextDOB, nextOption));

MessageBox.Show("Manager added!");

}

NameInformation.Text = DOBInformation.Text = BonusInformation.Text = StockInformation.Text = string.Empty;

if (SalesPersonRB.IsChecked == true)

SalesPersonRB.IsChecked = false;

if (ManagerRB.IsChecked == true)

ManagerRB.IsChecked = false;

}

}

}

}

}

1. **Xaml**
2. <Window x:Class="OOPLong8A.MainWindow"
3. xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4. xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5. xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
6. xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
7. xmlns:local="clr-namespace:OOPLong8A"
8. mc:Ignorable="d"
9. Title="MainWindow" Height="350" Width="525">
10. <Grid>
11. <Grid.RowDefinitions>
12. <RowDefinition Height="30"/>
13. <RowDefinition/>
14. <RowDefinition Height="30"/>
15. </Grid.RowDefinitions>
16. <Grid.ColumnDefinitions>
17. <ColumnDefinition/>
18. <ColumnDefinition/>
19. </Grid.ColumnDefinitions>
20. <Border Grid.Row="0" Grid.Column="0" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
21. <Label FontSize="15" FontWeight="Bold" HorizontalAlignment="Center">ADD A NEW STAFF MEMBER</Label>
22. </Border>
23. <Border Grid.Row="0" Grid.Column="1" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
24. <Label FontSize="15" FontWeight="Bold" HorizontalAlignment="Center">CHOOSE WHAT TO DISPLAY</Label>
25. </Border>
26. <Grid Grid.Row="1" Grid.Column="0">
27. <Grid.RowDefinitions>
28. <RowDefinition/>
29. <RowDefinition/>
30. <RowDefinition/>
31. <RowDefinition/>
32. <RowDefinition/>
33. </Grid.RowDefinitions>
34. <Grid.ColumnDefinitions>
35. <ColumnDefinition/>
36. <ColumnDefinition/>
37. </Grid.ColumnDefinitions>
38. <Border Grid.Row="0" Grid.Column="0" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
39. <Label FontSize="15">Name:</Label>
40. </Border>
41. <Border Grid.Row="1" Grid.Column="0" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
42. <Label FontSize="15">Date of Birth:</Label>
43. </Border>
44. <Border Grid.Row="2" Grid.Column="0" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
45. <RadioButton Name="SalesPersonRB" FontSize="15" Height="30" GroupName="EmployeeGroup" VerticalContentAlignment="Center">
46. Salesperson
47. </RadioButton>
48. </Border>
49. <Border Grid.Row="3" Grid.Column="0" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
50. <Label FontSize="15">Bonuses:</Label>
51. </Border>
52. <Border Grid.Row="4" Grid.Column="0" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
53. <Label FontSize="15">Stock Options:</Label>
54. </Border>
55. <Border Grid.Row="0" Grid.Column="1" Background="GhostWhite" BorderBrush="Silver" BorderThickness="3" CornerRadius="8,8,8,8">
56. <TextBox Name="NameInformation" FontSize="20" Margin="7" Width="109"/>
57. </Border>
58. <Border Grid.Row="1" Grid.Column="1" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
59. <TextBox Name="DOBInformation" FontSize="20" Margin="9" Width="109"/>
60. </Border>
61. <Border Grid.Row="2" Grid.Column="1" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
62. <RadioButton Name="ManagerRB" FontSize="15" Height="30" GroupName="EmployeeGroup" VerticalContentAlignment="Center">
63. Manager
64. </RadioButton>
65. </Border>
66. <Border Grid.Row="3" Grid.Column="1" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
67. <TextBox Name="BonusInformation" Grid.Row="3" Grid.Column="1" FontSize="20" Margin="9" IsEnabled="{Binding Path=IsChecked, ElementName=SalesPersonRB, Mode=OneWay}"/>
68. </Border>
69. <Border Grid.Row="4" Grid.Column="1" Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,8,8">
70. <TextBox Name="StockInformation" Grid.Row="4" Grid.Column="1" FontSize="20" Margin="9" IsEnabled="{Binding Path=IsChecked, ElementName=ManagerRB, Mode=OneWay}"/>
71. </Border>
72. </Grid>
73. <DataGrid Name="Display" Grid.Row="1" Grid.Column="1" AutoGenerateColumns="False">
74. <DataGrid.Columns>
75. <DataGridTextColumn Header="Name" Width="90" Binding="{Binding Name}"/>
76. <DataGridTextColumn Header="Date of Birth" Width="90" Binding="{Binding DateOfBirth, StringFormat={}{0:yyyy.MM.dd}}"/>
77. <DataGridTextColumn Width="60"/>
78. </DataGrid.Columns>
79. </DataGrid>
80. <Button Name="AddNew" Grid.Row="2" Grid.Column="0" FontSize="15" FontWeight="Bold" Click="AddNew\_Click">ADD NEW ENTRY</Button>
81. <ComboBox Name="DisplayChoice" Grid.Row="2" Grid.Column="1" FontSize="15" FontWeight="Bold" VerticalAlignment="Center" SelectedIndex="0" SelectionChanged="DisplayChoice\_SelectionChanged">
82. <ComboBoxItem>None</ComboBoxItem>
83. <ComboBoxItem>Salespersons</ComboBoxItem>
84. <ComboBoxItem>Managers</ComboBoxItem>
85. </ComboBox>
86. </Grid>
87. </Window>

**3. The result:**

