Collections and Generics

Collections

In .NET Framework, there is a group of objects that provides a standard set of types for storing and managing collections of objects. This group of objects is called **collections**, which contains lists, linked lists, dictionaries, and arrays to manage collections of objects.

Two (2) Types of Collections

- **Standard collections** – These are found in the `System.Collections` namespace.
- **Generic collections** – These are found in the `System.Collections.Generic` namespace.

Classes of the `System.Collections` *Namespace*

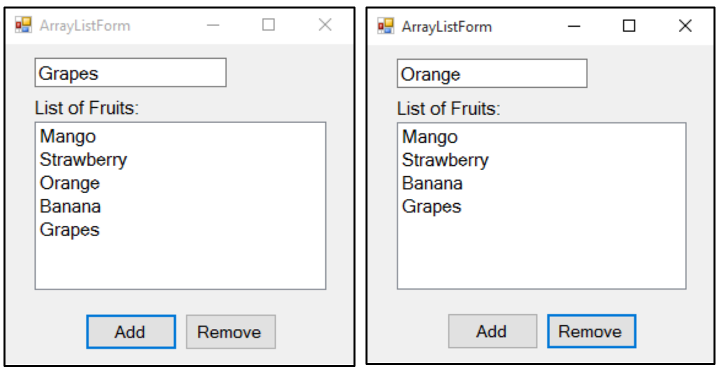
`ArrayList`

- This is a collections **class** that is known for an ordered collection of objects. This ArrayList is not the same as an array that is static in size. It is known as dynamic since items can be added and removed from the specified index location. See *Figures 1* and *2* for the sample program and output.

ArrayList

```
private ArrayList arrList;
public ArrayListForm()
{
   InitializeComponent();
   arrList = new ArrayList();
}
public void DisplayValues(IEnumerable getList)
{
   lvDisplay.Clear();
   foreach (Object obj in getList) {
      lvDisplay.Items.Add(obj.ToString());
   }
}
private void btnAdd_Click(object sender, EventArgs e)
{
   arrList.Add(txtBoxInput.Text.ToString());
   DisplayValues(arrList);
}
private void btnRemove_Click(object sender, EventArgs e)
{
   arrList.Remove(txtBoxInput.Text.ToString());
   DisplayValues(arrList);
}
```

Figure 1. ArrayList (Code)



```
ArrayListForm
Grapes
List of Fruits:
Mango
Strawberry
Orange
Banana
Grapes
```

```
ArrayListForm
Orange
List of Fruits:
Mango
Strawberry
Banana
Grapes
```

Figure 2. ArrayList (Output)

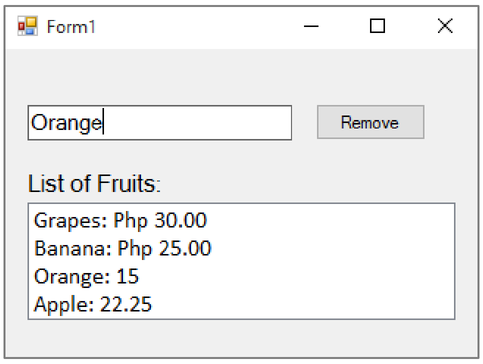
`Hashtable`

This class stores key or value pairs where the key represents the value in the collection. When accessing the values in Hashtable, the key must be specified. See *Figure 3* for the syntax and *Figure 4* for the output.

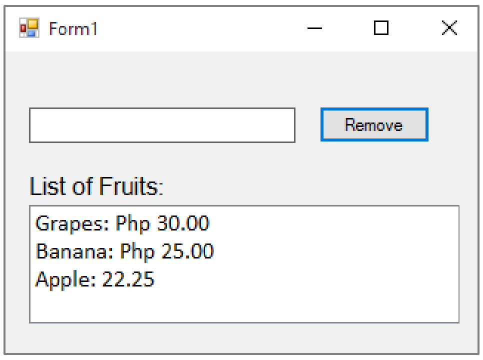
Hashtable

```
public partial class Form1 : Form{
   ICollection fruitCollection;
   Hashtable fruitsHashTable;

   public Form1(){
      InitializeComponent();
      fruitsHashTable = new Hashtable();
      hashTableData();
   }
   public void hashTableData(){
      fruitsHashTable.Add("Apple", 22.25);
      fruitsHashTable.Add("Banana", "Php 25.00");
      fruitsHashTable.Add("Grapes", "Php 30.00");
      fruitsHashTable.Add("Orange", 15);
      setList();
   }
   private void setList(){
      fruitCollection = fruitsHashTable.Keys;
      foreach(string key in fruitCollection){
         listBox1.Items.Add(key + ": " + fruitsHashTable[key].ToString());
      }
   }
   private void btnRemove_Click(){
      string getSearch = txtSearch.Text;
      if(fruitsHashTable.ContainsKey(getSearch)){
         fruitsHashTable.Remove(getSearch);
         listBox1.Items.Clear();

         setList();
      }else{
         MessageBox.Show("Not Available");
      }
   }
}
```

Figure 3. Hashtable (Code)



```
Grapes: Php 30.00
Banana: Php 25.00
Orange: 15
Apple: 22.25
```



```
Grapes: Php 30.00
Banana: Php 25.00
Apple: 22.25
```

Figure 4. Hashtable Output

SortedList

This class is a combination of **ArrayList** and **Hashtable**. It stores key or value pairs where the key values sort values. To access the values in **SortedList class**, the key or index value must be specified. See *Figure 5* for the example program of SortedList.

In this class, there are some **properties** that can be used:

- **Capacity** – It gets or sets the capacity of the SortedList.
- **Count** – It gets the count of the number of elements in the SortedList.
- **Item** – It gets and sets the value associated with a specific key in the SortedList.
- **Keys** – These carry the keys in the SortedList.
- **Values** – These carry the values in the SortedList.

There are few methods that are used in the SortedList:

- **`void Add(object key, object value)`** – It adds an item with the specified key and value into the `SortedList`.
- **`void Clear()`** – It is used to remove all the items in the `SortedList`.
- **`bool ContainsKey(object key)`** – If the `SortedList` contains the specified key, then it will return the Boolean value **`true`**.
- **`bool ContainsValue(object value)`** – If the `SortedList` contains the specified value, then it will return the Boolean value **`true`**.
- **`object GetByIndex(int index)`** – This method returns the value of the specified index.
- **`object GetKey(int index)`** – This method returns the key of the specified index.
- **`void Remove(object key)`** – In the `SortedList`, a key that is specified will remove its element.
- **`void RemoveAt(int index)`** – It is an index that is specified will remove its element in the `SortedList`.

SortedList

```
SortedList fruitsSL = new SortedList();
fruitsSL.Add("Grapes", 15);
fruitsSL.Add("Strawberry", 2);
fruitsSL.Add("Orange", 6);
fruitsSL.Add("Mango", 5);
Console.WriteLine("Price of Mango is " + fruitsSL["Mango"]);

//Displaying all the elements
for (int i = 0; i < fruitsSL.Count; i++)
{
   Console.WriteLine("Price of " + fruitsSL.GetKey(i) + " is " +
                     fruitsSL.GetByIndex(i));
}
```

Figure 5. SortedList Using Console

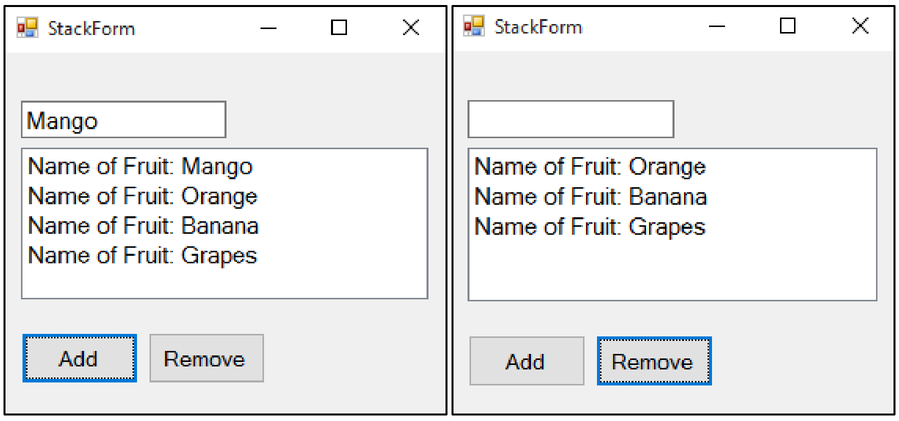
Stack

This represents a Last In, First Out (LIFO) collection of objects. When inserting and removing objects from the `stack`, it uses push and pop operations. See *Figures 6* and *7* for the program and output.

Stack

```
Stack fruitsStack;
public StackForm()
{
    InitializeComponent();
    fruitsStack = new Stack();
}
private void btnAdd_Click(object sender, EventArgs e)
{
   fruitsStack.Push(txtInput.Text);
   DisplayStack(fruitsStack);
}
private void btnRemove_Click(object sender, EventArgs e)
{
   fruitsStack.Pop();
   DisplayStack(fruitsStack);
}
public void DisplayStack(IEnumerable listOfFruit)
{
    lvFruit.Clear();
    foreach (Object obj in listOfFruit)
    {
        lvFruit.Items.Add("Name of Fruit: " + obj);
    }
}
```

Figure 6. Stack (Code)


StackForm

```
Mango
Name of Fruit: Mango
Name of Fruit: Orange
Name of Fruit: Banana
Name of Fruit: Grapes
```

StackForm

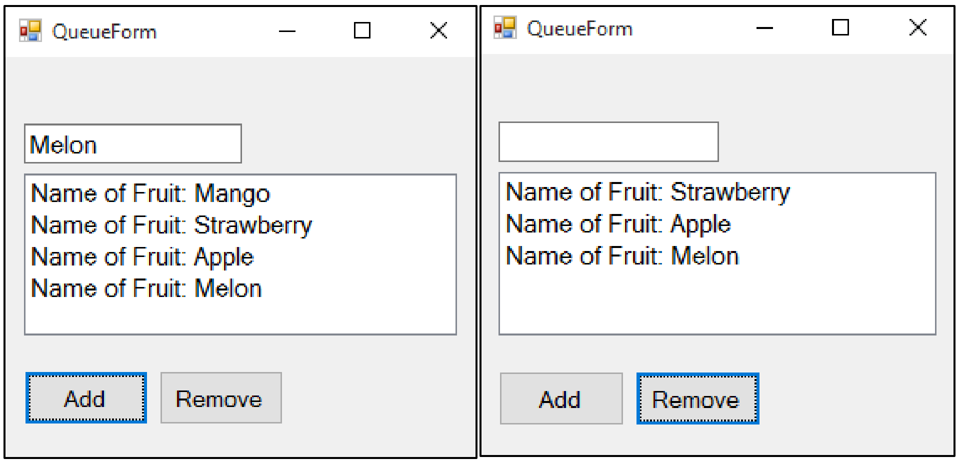
```
Name of Fruit: Orange
Name of Fruit: Banana
Name of Fruit: Grapes
```

Figure 7. Stack (Output)

Queue
This class represents a First In, First Out (FIFO) collection of objects. Adding and removing objects from the queue class uses an enqueue and dequeue operations. See *Figures 8* and *9* for the program and output.

Queue
```
Queue fruitsQueue;
public QueueForm()
{
   InitializeComponent();
   fruitsQueue = new Queue();
}
private void btnAdd_Click(object sender, EventArgs e)
{
   fruitsQueue.Enqueue(txtInput.Text);
   DisplayQueue(fruitsQueue);
}
private void btnRemove_Click(object sender, EventArgs e)
{
   fruitsQueue.Dequeue();
   DisplayQueue(fruitsQueue);
}
public void DisplayQueue(IEnumerable listOfFruit)
{
   lvFruit.Clear();
   foreach (Object obj in listOfFruit)
   {
      lvFruit.Items.Add("Name of Fruit: " + obj);
   }
}
```

Figure 8. Queue (Code)


QueueForm

```
Melon
Name of Fruit: Mango
Name of Fruit: Strawberry
Name of Fruit: Apple
Name of Fruit: Melon
```

QueueForm

```
Name of Fruit: Strawberry
Name of Fruit: Apple
Name of Fruit: Melon
```

Figure 9. Queue (Output)

Interfaces in the `System.Collections` *Namespace*
`IEnumerable` – It is an interface that allows you to loop through elements in a collection.

`ICollection` – It is an interface that allows you to determine the number of elements in a collections and copy them in a simple array type.

`IDictionary` – It is an interface that provides a list of elements that are accessible via a key or value rather than an index.

Generics

This makes the code reusable across different types by creating a template that contains placeholder types. Generics allows specifying the data types of the programming elements when they are actually used in the program. The **`System.Collections.Generic`** namespace contains several generic collection classes. When using generics, it allows creating collections like **linked lists**, **hashtables, stacks**, and **queues** that operate on an item of any data type.

Generic Collections

It avoids creation of custom collections for each type in the application. The generic collections are the generic equivalents for the `System.Collections` classes found in the `System.Collections.Generic` namespace. Below are the generic collections found in `System.Collections.Generic` namespace:

`List<T>`

- It is a generic collection that provides an efficient and dynamically allocated array, which is commonly used to store a list of duplicate objects. The **`List<T>`** class can grow and shrink with the number of objects.
- The syntax below is the format for creating a **`List<T>`** collection. See *Figure 10* for sample program and *Figure 11* for the output.

```
List<T> variableName = new List<T>();
```

- There are few methods that be used in List Collection, see *Table 1* for the descriptions of each method.

Method	Description

`Add()`	This method is used to add items and is placed to the end of a list. The elements can be accessed by calling its index value.
	
	Example:
	`//Adds a string "Rose"`
	`nameOfStud.Add("Rose");`
	
	`//Display the element by index number`
	`Console.WriteLine(nameOfStud[2]);`

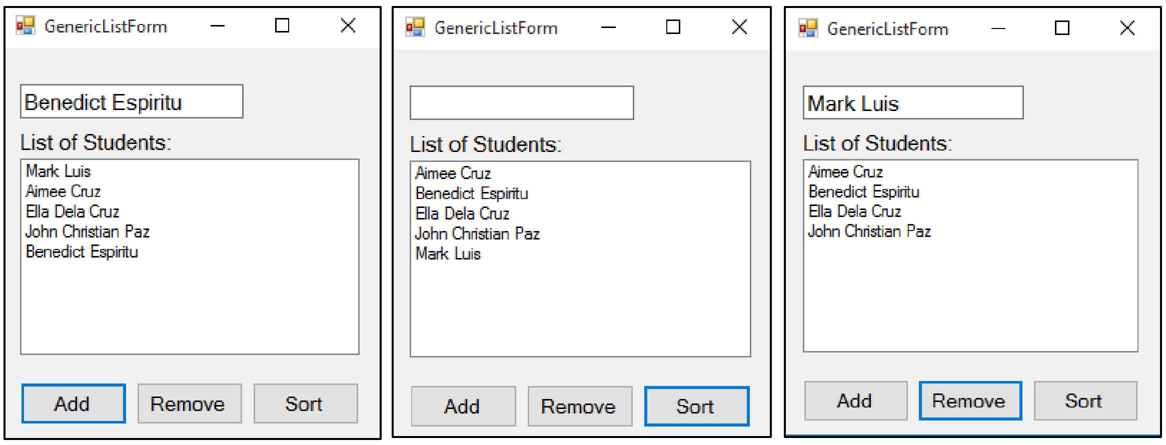
`Remove()`	This method removes the specified item from the list of object. If in case the list contains duplicate data, the method removes the first matching instance. **Example:** `nameOfStud.Remove("Mike");`
`IndexOf()`	This method is used to check the specified element in the specified list object. It will return its index value if it is found; if not, it will return the value of -1. **Example:** `Console.WriteLine(nameOfStud.IndexOf("Rose"));`
`Sort()`	This method is used to sort the element in the list object. **Example:** `nameOfStud.Sort();`

Table 1. List collections methods
Source: Learning Object-Oriented Programming in C# 5.0, 2015, p. 389.

Generic List<T>

```
private List<string> nameOfStudents;
public GenericListForm()
{
   InitializeComponent();
   nameOfStudents = new List<string>();
}
private void btnAdd_Click(object sender, EventArgs e)
{
   nameOfStudents.Add(txtBoxInput.Text.ToString());
   displayValue();
}
private void btnRemove_Click(object sender, EventArgs e)
{
   nameOfStudents.Remove(txtBoxInput.Text.ToString());
   displayValue();
}
private void btnSort_Click(object sender, EventArgs e)
{
   nameOfStudents.Sort();
   displayValue();
}
public void displayValue()
{
   lvDisplay.Clear();
   foreach (string listOfStudents in nameOfStudents) {
      lvDisplay.Items.Add(listOfStudents);
   }
}
```

Figure 10. Generic list (Code)



```
Benedict Espiritu
List of Students:
Mark Luis
Aimee Cruz
Ella Dela Cruz
John Christian Paz
Benedict Espiritu
```

```
List of Students:
Aimee Cruz
Benedict Espiritu
Ella Dela Cruz
John Christian Paz
Mark Luis
```

```
Mark Luis
List of Students:
Aimee Cruz
Benedict Espiritu
Ella Dela Cruz
John Christian Paz
```

Figure 11. Generic list (Output)

Queue<T>

- It is a generic collection that represents a First In, First Out (FIFO) collection of objects.
- There are a few methods that can be used in **Queue<T>** generic collection. See *Table 2* for the descriptions of each method.
- The syntax below defines a generic queue. See *Figure 12* for sample program and *Figure 13* for the output:

```
Queue<int> ageQueue = new Queue<int>();
```

Method	Description
Enqueue()	This method adds an element to the end of queue. **Example:** ageQueue.Enqueue(18);

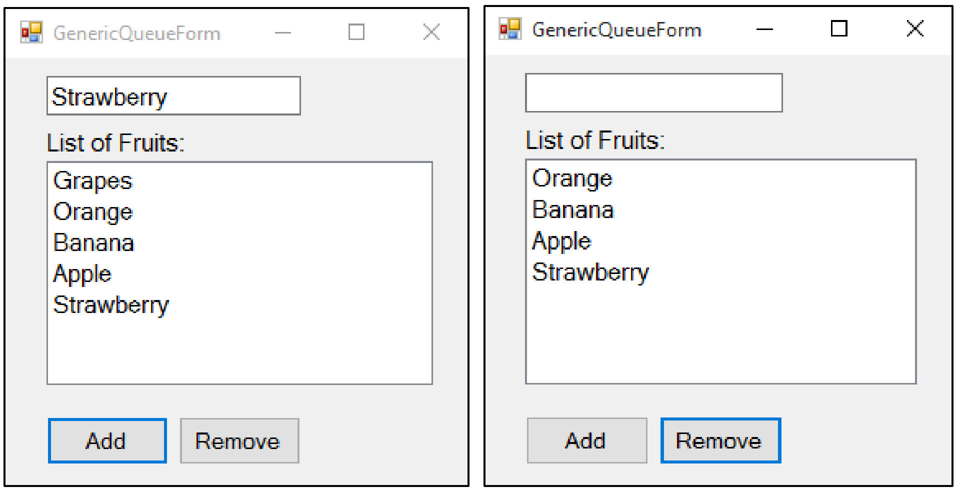
Method	Description
Peek()	This will return the element at the beginning of the queue without removing it. **Example:** `ageQueue.Peek();`
Dequeue()	This method removes and returns the value at the beginning of the queue. **Example:** `int getValue = ageQueue.Dequeue();`

Table 2. Queue collections methods
Source: Learning Object-Oriented Programming in C# 5.0, 2015, p.395.

Generic Queue<T>

```
private Queue<string> fruits;
public GenericQueueForm()
{
   InitializeComponent();
   fruits = new Queue<string>();
}
public void displayListOfFruits()
{
   lvDisplay.Clear();
   foreach (string getList in fruits) {
      lvDisplay.Items.Add(getList);
   }
}
private void btnAdd_Click(object sender, EventArgs e)
{
   fruits.Enqueue(txtBoxInput.Text.ToString());
   displayListOfFruits();
}
private void btnRemove_Click(object sender, EventArgs e)
{
   fruits.Dequeue();
   displayListOfFruits();
}
```

Figure 12. Generic Queue (Code)



```
Strawberry

Grapes
Orange
Banana
Apple
Strawberry
```

```
Orange
Banana
Apple
Strawberry
```

Figure 13. Generic Queue (Output)

Stack<T>

- This generic collection represents the Last In, First Out (LIFO) collection of instances.
- There are methods that can be used with **Stack<T>** generic collection. See *Table 3* for the descriptions of each method.
- The syntax below defines the generic stack. See *Figure 14* for sample program and *Figure 15* for the output:

`Stack<int> ageStack = new Stack<int>(20);`

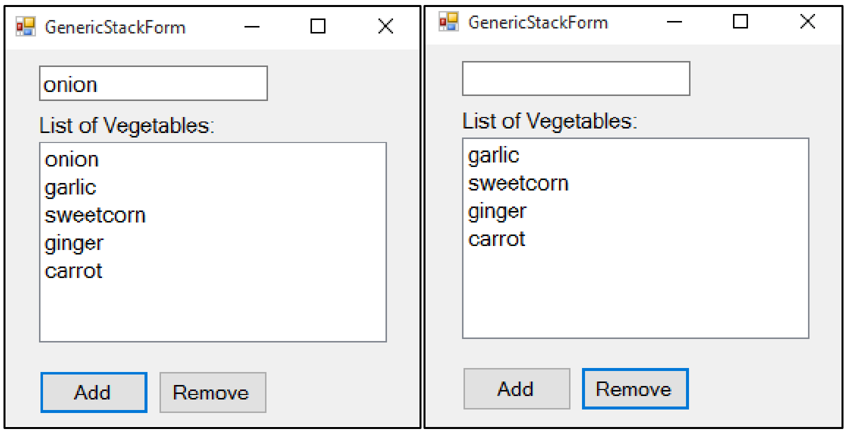
Method	Description
`Push()`	This method adds an element at the top in the stack. **Example:** `ageStack.Push(18);`
`Peek()`	This will return the element at the top of the stack without removing it. **Example:** `int getValue = ageStack.Peek();`
`Pop()`	This method removes and returns the value at the top of the stack. **Example:** `int getValue = ageStack.Pop();`

Table 3. Stack Collections Methods
Source: **Learning Object-Oriented Programming in C# 5.0, 2015, p. 396.**

Generic Stack<T>

```
private Stack<string> vegetables;
public GenericStackForm()
{
   InitializeComponent();
   vegetables = new Stack<string>();
}
public void displayListOfVegetable() {
   lvDisplay.Clear();
   foreach(string getList in vegetables) {
      lvDisplay.Items.Add(getList);
   }
}
private void btnAdd_Click(object sender, EventArgs e)
{
   vegetables.Push(txtBoxInput.Text.ToString());
   displayListOfVegetable();
}
private void btnRemove_Click(object sender, EventArgs e)
{
   vegetables.Pop();
   displayListOfVegetable();
}
```

Figure 14. Generic Stack (Code)


GenericStackForm

onion

List of Vegetables:

onion
garlic
sweetcorn
ginger
carrot

Add Remove

GenericStackForm

List of Vegetables:

garlic
sweetcorn
ginger
carrot

Add Remove

Figure 15. Generic Stack (Output)

REFERENCES:
Deitel, P. and Deitel, H. (2015). *Visual C# 2012 how to program* (5th ed.). USA: Pearson Education, Inc.
Gaddis, T. (2016). *Starting out with Visual C#* (4th ed.). USA: Pearson Education, Inc.
Harwani, B. (2015). *Learning object-oriented programming in C# 5.0.* USA: Cengage Learning PTR.
Miles, R. (2016). *Begin to code with C#.* Redmond Washington: Microsoft Press.
Doyle, B. (2015). *C# programming: from problem analysis to program design* (5th ed.). Boston, MA: Cengage Learning.