CMPE 1666
Intermediate Programming

Lecture 11- Collections

Collection Classes

- A collection is a group of individual objects stored as one entity.
- An array is an example of a collection, but we have a number of collections that can grow and Shrink dynamically.
- There are a number of collection classes available in C#
- In this course, we are interested in **generic** collections.
- These are collections to which we pass the data type at declaration time.

Collection Classes

- An example of a generic collection class that we have seen so far is the List.
- We have seen the declaration and use of List of strings and structs, for example.
- E.g. List<string> myList = new List<String>();
- List<Student> studList= new List<Student>()

Collection Classes

- There are many generic collection classes available for us to use in .NET Framework
- Not all collections contain the same functionalities.
- Also some collection classes perform better than others on certain tasks.
- It is important therefore to select the most suitable collection class for the task

Collection Classes

- The following collection classes exist in C#. We'll discuss some of them in this course.
  - Dictionary (a hash table of key/value pairs)
  - HashSet (a hash table of unique values)
  - LinkedList (a doubly linked list of values)
  - List (a vector of values)
  - Queue (a FIFO collection of values)
  - SortedDictionary (a Dictionary that is sorted by key)
  - SortedList (a binary search tree of key/value pairs)
  - Stack (a LIFO collection of values)
- Generic collections, are able to hold virtually any type of object, including value and reference types

Introduction To The Queue Collection

- Like the List, the Queue is also a generic collection.
- However, unlike the List, where we can access any element directly, the Queue has restricted access.
- It is a First In First Out (FIFO) Structure.
- The Queue makes use of **Enqueue**, **Dequeue**, and **Peek** methods.
  - **Enqueue()** – Adds an object to the end of the queue
  - **Dequeue()** –removes an object from the front of the queue and returns it
  - **Peek()** – returns the first item in the queue, without removing it.
- The Queue being a generic type we have to specify the data type when declaring and creating a Queue
  - Queue<int> Q1= new Queue<int>();

Example of Using a queue

- The following code creates a list and a queue (both of type int), adds a number of random numbers to the list, displays the list, then adds all the values to the queue.

```
List<int> aList = new List<int>();
Queue<int> q = new Queue<int>();
Random _rnd = new Random();
// add 10 random numbers between 0 and 9 inclusive
for (int i = 0; i < 10; ++i)
    aList.Add(_rnd.Next(0, 10));
// show the contents of the list
foreach (int i in aList)
    Console.Write($"{i}, ");
Console.WriteLine(" ");

//Adding the values to the queue
foreach (int i in aList)
{
    Console.WriteLine($"Enqueue : {i}");
    q.Enqueue(i);
}
```

Example of Using a queue (contd)

- The code below displays the queue (created in the previous slide) and then removes each of the elements form the queue.

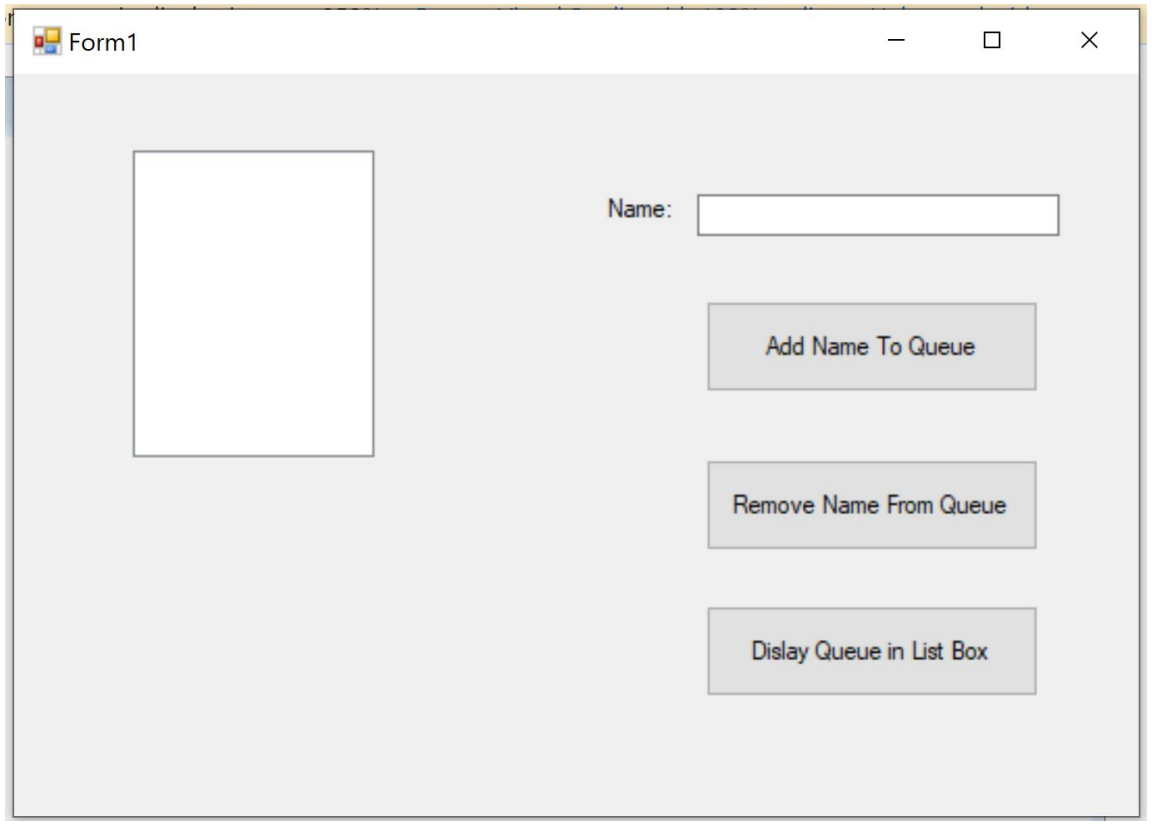
```
// show the contents of the queue
foreach (int i in q)
    Console.Write($"{i}, ");
Console.WriteLine(" ");

//Removing all elements from the queue
while (q.Count > 0)
    Console.WriteLine($"Dequeue : {q.Dequeue()}");
```

Lecture 11-Demo 1

- You have been provided with a starter project containing the UI for Lecture 11 Demo1. The UI is as shown on the next slide. Add the code as required to provide your program with functionalities as specified below:
- The program must create a Queue of Strings. When the “Add Name To Queue” button is clicked, the program adds the name to the queue. It must also clear the textbox. When the “Remove from Queue” button is clicked, the program removes the first name from the queue. It displays the name in a Message box. When the user clicks on the “Display Queue in List Box” button, the program clears the listbox and redisplays the queue in the listbox
- Run your program. Add 10 names to the queue. Get the queue displayed. Then each time remove one name, get the queue displayed again. Continue removing a name and redisplaying the queue until the queue is empty.

UI for Lecture 11 Demo1



The Stack

- The Stack is a collection that operates as a Last In First Out (LIFO) structure.
- The Stack type is enumerable (you can use foreach on it).
- Values are added to the top of the stack with the **Push()** method.
- Values are removed from the top of the stack with **Pop()** method.
- **Peek()** will allow you to look at the top of the stack without modifying the stack.
- Stacks are enumerated from the top of the stack to the bottom.
- Note: we don't use Add, Remove, Insert etc. on a stack

The Stack

```
Stack<int> StkA = new Stack<int>();

for (int i = 0; i < 10; i++)
{
    int temp = _rnd.Next(0, 10);
    Console.WriteLine($"Pushing : {temp}");
    StkA.Push(temp);
}

// show the contents of the stack
foreach (int t in StkA)
    Console.Write($"{t}, ");
Console.WriteLine(" ");

while (StkA.Count > 0)
    Console.WriteLine($"Popping : {StkA.Pop()}");
```

Lecture 11- Demo 2

- Write a program that creates a struct type **MyPoint** where each **MyPoint** instance has an X and Y value.
- The struct must contain a constructor that has a parameter for each data member and it assigns the values of the parameters to the data members.
- It must have a ToString() method that returns a string in the form (X,Y).
- Your Main function must create a stack of Point instances then continuously display a menu that allows the user to choose to:
  1. Input the coordinates for a new **MyPoint** struct. The struct is created and pushed onto the stack.
  2. Display the stack (Make use of the ToString() method)
  3. Pop() out an element from the stack if the stack has at least one element.
  4. Display the point at the top of the stack
  5. Exit