Day 19

Delegates

- C#, A delegate is a strongly typed function pointer.
- A delegate is a type that represents references to methods with a particular parameter list and return type.
- The reference can be changed at runtime.
- Delegates are especially used for implementing events and the call-back methods.
- All delegates are implicitly derived from the **System.Delegate** class.
- Delegates are used to pass methods as arguments to other methods.

use of Delegates

- if you have multiple methods with same signature (return type & number of parameters) and want to call all the methods with single object then we can go for delegates.

  Syntax for delegate declaration is:

  ```
  delegate <return type> <delegate-name> <parameter list>
  ```

- Delegates are two types
  - Single Cast Delegates
  - Multi Cast Delegates

Multicast Delegates

- Multi cast delegate is used to hold address of multiple methods in single delegate.
- To hold multiple addresses with delegate we will use overloaded += operator.
- if you want remove addresses from delegate we need to use overloaded operator -=
- Multicast delegates will work only for the methods which have return type only void.

Events

- Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications.

- Applications need to respond to events when they occur.
- For example, interrupts. Events are used for inter-process communication.

Declaring Events:

first a delegate type for the event must be declared.

```
public delegate void Handler();

//Defining event based on the above delegate

public event  Handler BoilerEventLog;
```

Generics

- Generics allow you to delay the specification of the data type of programming elements in a class or a method, until it is actually used in the program.
- In other words, generics allow you to write a class or method that can work with any data type.

Features of Generics:

- It helps you to maximize code reuse, type safety, and performance.
- You can create generic collection classes.
- classes in the **System.Collections.Generic** namespace
- You can create your own generic interfaces, classes, methods, events, and delegates.

Advantage with Generics

- Cleaner Code with Generics

  Since compiler enforces type safety with Generics. Hence fetching data from Generics doesn't required type casting which means your code is clean and easier to write and maintain.

- Better Performance with Generics

  At the time of fetching the data from the ArrayList collection we need to do type casting which cause performance degrades. But at the time of fetching the data from the generic List we don't required to do type casting. In this way Generics provide better performance than collections.

Collection Classes

- Collection classes are specialized classes for data storage and retrieval.
- These classes provide support for array list,stacks, queues, lists, and hash tables.
- Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc.

Various collection classes

- Array list
- List
- Hash table
- Stack
- queue

Array list

- It represents an ordered collection of an object that can be indexed individually.
- It is basically an alternative to an array.
- unlike array you can add and remove items from a list at a specified position using an index and the array resizes itself automatically.
- It also allows dynamic memory allocation, adding, searching and sorting items in the list.
- Namespace **system.collections**.

Methods in array list

- Add(object value);
- Clear();
- Contains(object item);
- Insert(int index, object value);
- Remove(object obj);
- Sort();
- Reverse();

Hashtable class

- Hashtable class represents a collection of key-and-value pairs that are organized based on the hash code of the key.
- It uses the key to access the elements in the collection.
- A hash table is used when you need to access elements by using key, and you can identify a useful key value.
- Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

Hash table methods

- Add(object key, object value);
- Clear();
- ContainsKey(object key);
- ContainsValue(object value);
- Remove(object key);

```
// Get a collection of the keys.
    ICollection key = ht.Keys;
    foreach (string k in key)
        {
            Console.WriteLine(k + ": " + ht[k]);
        }
```

Stack Class

- It represents a last-in, first out collection of object.
- It is used when you need a last-in, first-out access of items.
- When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.

Methods in stack

- Clear();
- Contains(object obj);
- Peek(); Returns the object at the top of the Stack without removing it
- Pop();
  Removes and returns the object at the top of the Stack.
- Push(object obj);
  Inserts an object at the top of the Stack..

Queue Class

- It represents a first-in, first out collection of object.

- It is used when you need a first-in, first-out access of items.

- When you add an item in the list, it is called enqueue, and when you remove an item, it is called deque

Methods in queue

- Clear();
- Contains(object obj);
- Dequeue();
  Removes and returns the object at the beginning of the Queue.

- Enqueue(object obj);
  Adds an object to the end of the Queue.

List

- List class is a collection and defined in the **System.Collections.Generic** namespace
- it provides the methods and properties like other Collection classes such as add, insert, remove, search etc.
- List < T > class represents a strongly typed list of objects that can be accessed by index and it supports storing values of a specific type without casting to or from object.

Creating Generic List<T> Collections

```
List<int> myInts = new List<int>();

    myInts.Add(1);
    myInts.Add(2);
    myInts.Add(3);

    for (int i = 0; i < myInts.Count; i++)
    {
        Console.WriteLine("MyInts: {0}", myInts[i]);
    }
```

Bit Array

- It represents an array of the binary representation using the values 1 and 0.
- It is used when you need to store the bits but do not know the number of bits in advance. You can access items from the BitArray collection by using an integer index, which starts from zero.

Differences between Array list and Hash table

- **Array**
- Array only value is store.
- To access value, you need to pass index number.
- Array you can store only similar type of data.

- **Hash table**
- Hash table store data as name, value pair.
- To access value from hash table, you need to pass name.
- Hash table can store different type of data in hash table, say int, string etc.

Private Assembly

- A private assembly is an assembly that is available to particular application where they are kept
- a Private Assembly cannot be references outside the scope of the folder where they are kept.

Shared Assembly

- shared Assembly is a public assembly that is shared by multiple applications.
- Shared Assembly is one that can be referenced by more than one application.
- Shared assembly is stored in GAC (Global Assembly Cache)

END...