Session: 12

Events, Delegates, and Collections

Objectives

- Explain delegates
- Explain events
- Define and describe collections

Delegates

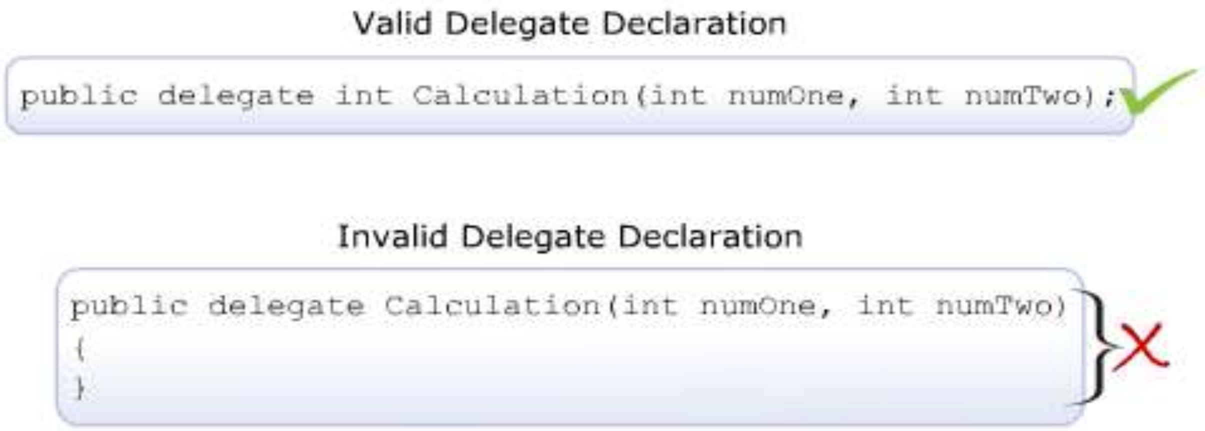
Delegates are objects that contain references to methods that need to be invoked.

Using delegates, you can call any method, which is identified only at run-time.

To associate a delegate with a particular method, the method must have the same return type and parameter type as that of the delegate.

Declaring Delegates

- Declaring a delegate is quite similar to declaring a method except that there is no implementation.
- Example :


Valid Delegate Declaration

```
public delegate int Calculation(int numOne, int numTwo);
```

Invalid Delegate Declaration

```
public delegate Calculation(int numOne, int numTwo)
{
}
```

Syntax

```
<acc_modifier> delegate <ret_type> DelegateName([parameters]);
```

Snippet

```
public delegate int Calculation(int numOne, int numTwo);
```

Instantiating Delegates

Snippet

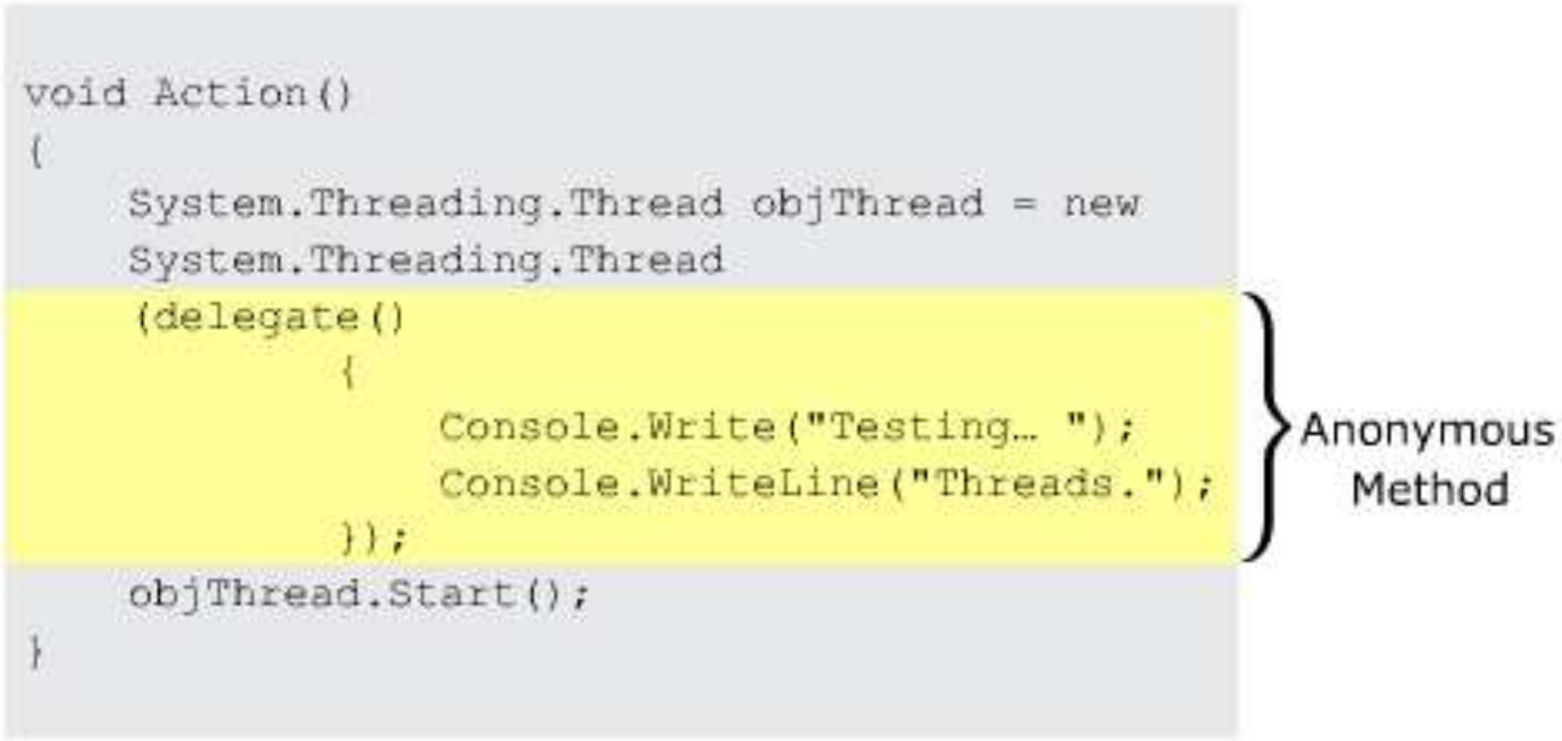
```
public delegate int Calculation (int numOne, int numTwo);

class Mathematics  {
   static int Add(int numOne, int numTwo) {
      return (numOne + numTwo);
   }
   int Subtract(int numOne, int numTwo) {
      return (numOne - numTwo);
   }

   static void Main(string[] args) {
      int n1 = 5;
      int n2 = 23;
      Calculation oCalc = new Calculation(Add);
      Console.WriteLine("{0} + {1} = {2}",n1,n2,oCalc (n1, n2) );
   }
}
```

Output

```
5 + 23= 28
```

Delegates & Anonymous Methods

- An anonymous method is an inline block of code that can be passed as a delegate parameter that helps to avoid creating named methods.
- The following figure displays an example of using anonymous methods:



```
void Action()
{
    System.Threading.Thread objThread = new
    System.Threading.Thread
    (delegate()
          {
                Console.Write("Testing… ");
                Console.WriteLine("Threads.");
          });
    objThread.Start();
}
```

Multicast Delegates

Snippet

```
public delegate void Maths (int valOne, int valTwo);

class MathsDemo {
    static void Add (int x, int y) {
        Console.WriteLine("Addition: {0} + {1} = {2}",x , y, x+y);
    }
    static void Subtract (int x, int y) {
        Console.WriteLine("Subtraction: {0} - {1} = {2}",x , y, x-y);
    }
    static void Divide(int x, int y) {
        Console.WriteLine("Division: {0} / {1} = {2}",x , y, x/y);
    }
    static void Main(string[] args) {
        Maths oMaths = new Maths(Add);
        oMaths += new Maths(Subtract);
        oMaths += new Maths(Divide);
        if (oMaths != null) {   oMaths(20, 10); }
    }
}
```

`System.Delegate` Class

- The **`Delegate`** class is a built-in class defined to create delegates in C#.
- All delegates in C# implicitly inherit from the **`Delegate`** class.
- Constructors :

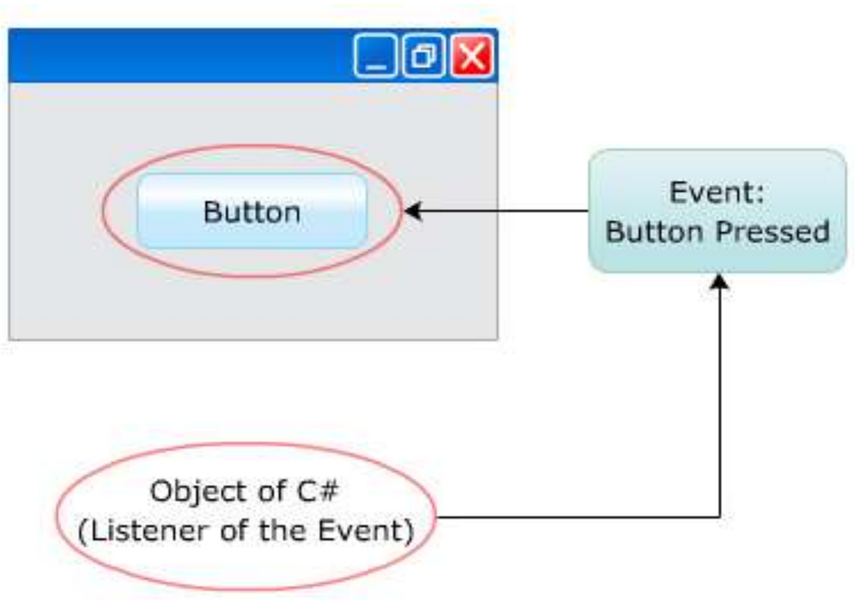
Constructor	Description
`Delegate(object, string)`	Calls a method referenced by the object of the class given as the parameter
`Delegate(type, string)`	Calls a static method of the class given as the parameter

- Properties :

Property	Description
`Method`	Retrieves the referenced method
`Target`	Retrieves the object of the class in which the delegate invokes the referenced method

Events

- An event is a user-generated or system-generated action.
- In C#, events allow an object (source of the event) be able to notify other objects (subscribers) about the appeared event (a change having occurred).
- The following figure depicts the concept of events:



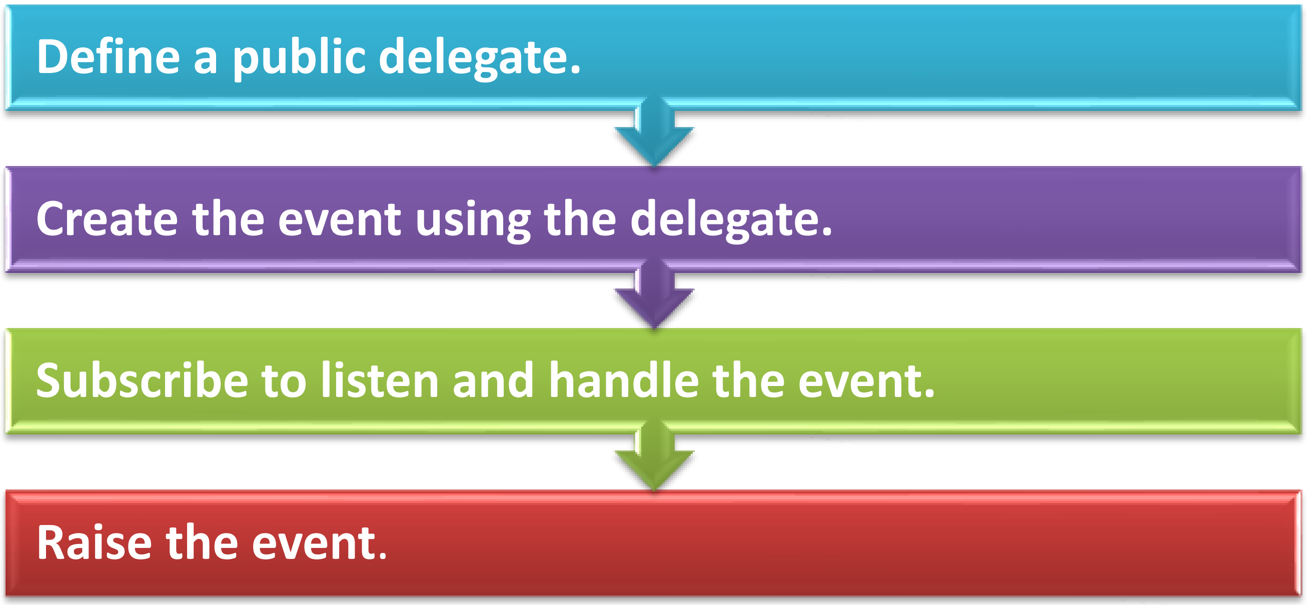
Features of Event

- can be declared in classes and interfaces.
- can be declared as abstract, virtual or sealed.
- implemented using delegates.

- Events can be used to perform customized actions that are not already supported by C#.
- Events are widely used in creating GUI based applications, where events such as, selecting an item from a list and closing a window are tracked.

Creating and Using Events

- Following are the four steps for implementing events in C#:



Define a public delegate.

Create the event using the delegate.

Subscribe to listen and handle the event.

Raise the event.

- Events use delegates to call methods in objects that have subscribed to the event.
- When an event containing a number of subscribers is raised, many delegates will be invoked.

Example

```
public delegate void PrintDetails();

class TestEvent {
    event PrintDetails Print;
    void Show()
    {
        Console.WriteLine(“how to subscribe objects to an event”);
    }

    static void Main(string[] args)
    {
        TestEvent o = new TestEvent();
        o.Print += new PrintDetails(o.Show);
        o.Print();
    }
}
```

Collections

- A collection is a set of related data that may not necessarily belong to the same data type that can be set or modified dynamically at run-time.
- Accessing collections is similar to accessing arrays, where elements are accessed by their index numbers.
- However, there are differences between arrays and collections in C#:

Array	Collection
Cannot be resized at run-time.	**Can be resized at run-time.**
The individual elements are of the same data type.	**The individual elements can be of different data types.**
Do not contain any methods for operations on elements.	**Contain methods for operations on elements.**

System.Collections Namespace

Class/Interface	Description
ArrayList Class	similar to an array except that the items can be dynamically added and retrieved from the list and it can contain values of different types
Stack Class	follows the Last-In-First-Out (LIFO) principle, which means the last item inserted in the collection, will be removed first
Hashtable Class	Provides a collection of key and value pairs that are arranged, based on the hash code of the key
SortedList Class	Provides a collection of key and value pairs where the items are sorted, based on the keys
IDictionary Interface	Represents a collection consisting of key/value pairs
IDictionaryEnumerator Interface	Lists the dictionary elements
Ienumerable Interface	Defines an enumerator to perform iteration over a collection
ICollection Interface	Specifies the size and synchronization methods for all collections
IEnumerator Interface	Supports iteration over the elements of the collection
IList Interface	Represents a collection of items that can be accessed by their index number

ArrayList Class

- Features of **ArrayList**:

is a variable-length array.

can store elements of different data types.

can accept null values.

can also include duplicate elements.

allows to specify the size and the capacity of the collection.

default capacity is 16.

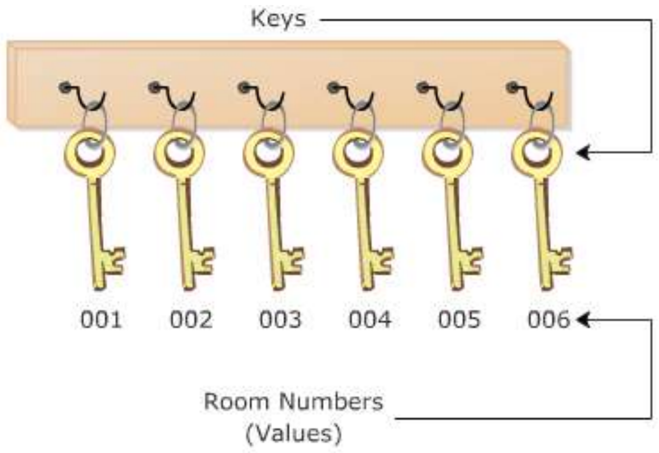
If the number of elements in the list reaches the specified capacity, the capacity of the list gets doubled automatically..

consists of different methods and properties that are used to add, modify, and delete any element in the list even at run-time.

can be accessed the elements by using the index position.

Hashtable Class

- Consider the reception area of a hotel where you find the keyholder storing a bunch of keys.
- Each key in the keyholder uniquely identifies a room and thus, each room is uniquely identified by its key.
- Similar to the keyholder, the **`Hashtable`** class in C# allows you to create collections in the form of keys and values that associates keys with their corresponding values.
- The **`Hashtable`** class uses the hashtable to retrieve values associated with their unique key.



`SortedList` Class

- represents a collection of key and value pairs where elements are sorted according to the key.
- By default, the **`SortedList`** sorts the elements in ascending order, however, this can be changed if an **`IComparable`** object is passed to the constructor of the **`SortedList`** class.
- These elements are either accessed using the corresponding keys or the index numbers.
- If you access elements using their keys, the **`SortedList`** behaves like a hashtable, whereas if you access elements based on their index number, it behaves like an array.

DictionaryGeneric Class

- consists of a generic collection of elements organized in key and value pairs and maps the keys to their corresponding values.
- The following syntax declares a **Dictionary** generic class:

```
Dictionary<TKey, TValue>
```

Summary

- A delegate in C# is used to refer to a method in a safe manner.
- An event is a data member that enables an object to provide notifications to other objects about a particular action.
- The **`System.Collections.Generic`** namespace consists of generic collections that allow reusability of code and provide better type-safety.
- The **`ArrayList`** class allows you to increase or decrease the size of the collection during program **`executionHashtable`** class.
- The **`Hashtable`** class stores elements as key and value pairs where the data is organized based on the hash code. Each value in the `hashtable` is uniquely identified by its key.
- The **`SortedList`** class allows you to store elements as key and value pairs where the data is sorted based on the key.
- The **`Dictionary`** generic class represents a collection of elements organized in key and value pairs.