DELEGATES AND EVENTS

DELEGATES

- A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.
- If you were a C or C++ programmer, this would sound familiar because a *delegate* is basically a function pointer.
- Why do I need a reference to a method?" ->maximum flexibility to implement any functionality you want at runtime.
- Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the **System.Delegate** class.

DECLARING DELEGATES

- Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.
- For example, consider a delegate:

```
public delegate int MyDelegate (string s);
```

- This can be used to reference any method that has a single *string* parameter and returns an *int* type variable.
- Syntax for delegate declaration is:

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

INSTANTIATING DELEGATES

- Once a delegate type is declared, a delegate object must be created with the **new** keyword and be associated with a particular method.
- When creating a delegate, the argument passed to the **new** expression is written similar to a method call, but without the arguments to the method. For example:

```
public delegate void printString(string s);
...
printString ps1 = new printString(WriteToScreen);
printString ps2 = new printString(WriteToFile);
```

EXAMPLE

- Following example demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

```
using System;

delegate int NumberChanger(int n);
namespace DelegateAppl
{
   class TestDelegate
   {
      static int num = 10;
      public static int AddNum(int p)
      {
         num += p;
         return num;
      }

      public static int MultNum(int q)
      {
         num *= q;
         return num;
      }
      public static int getNum()
      {
         return num;
      }

      static void Main(string[] args)
      {
         //create delegate instances
         NumberChanger nc1 = new NumberChanger(AddNum);
         NumberChanger nc2 = new NumberChanger(MultNum);

         //calling the methods using the delegate objects
         nc1(25);
         Console.WriteLine("Value of Num: {0}", getNum());
         nc2(5);
         Console.WriteLine("Value of Num: {0}", getNum());
         Console.ReadKey();
      }
   }
}
```

```
Value of Num: 35

Value of Num: 175
```

USING DELEGATES

- The following example demonstrates the use of delegate. The delegate *printString* can be used to reference method that takes a string as input and returns nothing.
- We use this delegate to call two methods, the first prints the string to the console, and the second one prints it to a file:

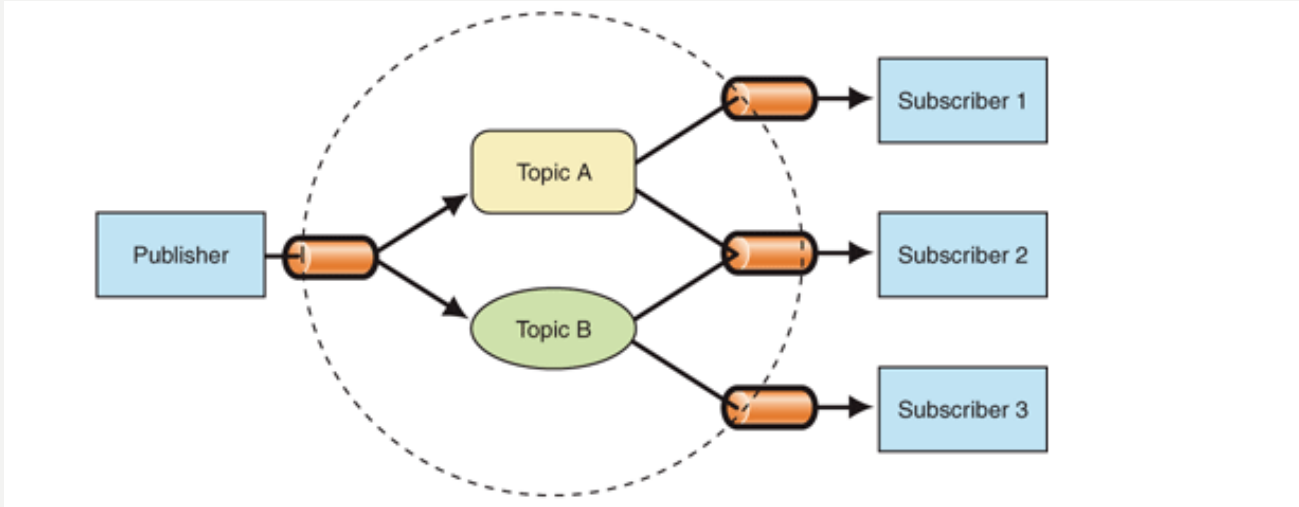
```
namespace DelegateAppl
{
   class PrintString
   {
      static FileStream fs;
      static StreamWriter sw;

      // delegate declaration
      public delegate void printString(string s);

      // this method prints to the console
      public static void WriteToScreen(string str)
      {
         Console.WriteLine("The String is: {0}", str);
      }

      //this method prints to a file
      public static void WriteToFile(string s)
      {
         fs = new FileStream("c:\\message.txt",
         FileMode.Append, FileAccess.Write);
         sw = new StreamWriter(fs);
         sw.WriteLine(s);
         sw.Flush();
         sw.Close();
         fs.Close();
      }

      // this method takes the delegate as parameter and uses it to
      // call the methods as required
      public static void sendString(printString ps)
      {
         ps("Hello World");
      }
      static void Main(string[] args)
      {
         printString ps1 = new printString(WriteToScreen);
         printString ps2 = new printString(WriteToFile);
         sendString(ps1);
         sendString(ps2);
         Console.ReadKey();
      }
   }
}
```

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.

USING DELEGATES WITH EVENTS

- The **publisher-subscriber** model :
- The sender (also called a *publisher*) uses a topic-based approach to publish messages to topic A and to topic B.
- Three receivers (also called *subscribers*) subscribe to these topics; one receiver subscribes to topic A, one receiver subscribes to topic B, and one receiver subscribes to both topic A and to topic B.
- The arrows show messages flowing from the publisher to each subscriber according to these subscriptions.



USING DELEGATES WITH EVENTS

- A **publisher** is an object that contains the definition of the event and the delegate. The event-delegate association is also defined in this object. A publisher class object invokes the event and it is notified to other objects.
- A **subscriber** is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

DECLARING EVENTS

- To declare an event inside a class, first a delegate type for the event must be declared. For example,

```
public delegate string MyDel(string str);
```

- Next, the event itself is declared, using the **event** keyword:

```
event MyDel MyEvent;
```

EXAMPLE

- The preceding code defines a delegate named *BoilerLogHandler* and an event named *BoilerEventLog*, which invokes the delegate when it is raised.

```
namespace SampleApp {
   public delegate string MyDel(string str);

   class EventProgram {
      event MyDel MyEvent;

      public EventProgram() {
         this.MyEvent += new MyDel(this.WelcomeUser);
      }

      public string WelcomeUser(string username) {
         return "Welcome " + username;
      }

      static void Main(string[] args) {
         EventProgram obj1 = new EventProgram();
         string result = obj1.MyEvent("Tutorials Point");
         Console.WriteLine(result);
      }

   }
}
```