**Multicasting with Delegates:**

* One of the most exciting features of a delegate is its support for multicasting.
* In simple terms, multicasting is the ability to create an invocation list, or chain, of methods that will be automatically called when a delegate is invoked.
* Such a chain is very easy to create. Simply instantiate a delegate, and then use the + or += operator to add methods to the chain.
* To remove a method, use – or – =.
* If the delegate returns a value, then the value returned by the last method in the list becomes the return value of the entire delegate invocation.
* Thus, a delegate that makes use of multicasting will often have a void return type.
* Here is an example of multicasting. Notice that it reworks the preceding examples by changing the string manipulation method’s return type to void and using a ref parameter to return the altered string to the caller. This makes the methods more appropriate for multicasting.

// Demonstrate multicasting.

**using System;**

// Declare a delegate type.

**delegate void StrMod(ref string str);**

**class MultiCastDemo**

{

// Replaces spaces with hyphens.

**static void ReplaceSpaces(ref string s)**

{

Console.WriteLine("Replacing spaces with hyphens.");

s = s.Replace(' ', '-');

}

// Remove spaces.

**static void RemoveSpaces(ref string s)**

{

string temp = "";

int i;

Console.WriteLine("Removing spaces.");

for(i=0; i<s.Length; i++)

if(s[i] != ' ') temp += s[i];

s = temp;

}

// Reverse a string.

**static void Reverse(ref string s)**

{

string temp = "";

int i, j;

Console.WriteLine("Reversing string.");

for(j=0, i=s.Length-1; i>= 0; i--, j++)

temp += s[i];

s = temp;

}

}

**Class y**

{

**public static void Main(String[] args)**

{

// Construct delegates.

**StrMod strOp;**

**StrMod** replaceSp = **ReplaceSpaces;**

**StrMod** removSp = **RemoveSpaces;**

**StrMod** reverseStr = **Reverse;**

string str = "This is a test";

// Set up multicast.

**strOp** = replaceSp;

**strOp** += reverseStr;

// Call multicast.

**strOp(ref str);**

Console.WriteLine("Resulting string: " + str);

Console.WriteLine();

// Remove replace and add remove.

**strOp** -= replaceSp;

**strOp** += removeSp;

str = "This is a test."; // reset string

// Call multicast.

**strOp(ref str);**

Console.WriteLine("Resulting string: " + str);

Console.WriteLine();

}

}

**Here is the output:**

Replacing spaces with hyphens.

Reversing string.

Resulting string: tset-a-si-sihT

Reversing string.

Removing spaces.

Resulting string: .tsetasisihT

* In Main( ), four delegate instances are created. One, strOp, is null. The other three refer to specific string modification methods. Next, a multicast is created that calls RemoveSpaces( ) and Reverse( ).
* This is accomplished via the following lines:

strOp = replaceSp;

strOp += reverseStr;

* First, strOp is assigned replaceSp. Next, using +=, reverseStr is added. When strOp is invoked, both methods are invoked, replacing spaces with hyphens and reversing the string, as the output illustrates.
* Next, replaceSp is removed from the chain, using this line:

strOp -= replaceSp;

and removeSP is added using this line:

strOp += removeSp;

* Then, strOp is again invoked. This time, spaces are removed and the string is reversed.
* Delegate chains are a powerful mechanism because they allow you to define a set of methods that can be executed as a unit.
* This can increase the structure of some types of code. Also, as you will soon see, delegate chains have a special value to events.