Delegates

Delegate: a person sent or authorized to represent others. A delegate in C# programming represents a method that can be invoked remotely or can be passes as a parameter.

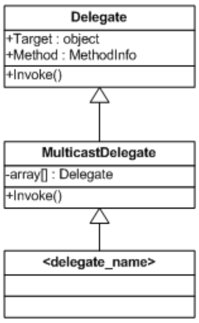
- **Reference type, abstraction of method**
- **Represents any method with a compatible signature – smart method**
- **.NET equivalent of a functor, or function object**
- **Inherits from** MulticastDelegate**, which inherits from** Delegate
- **Provides asynchronous event handling**
- **Provides callback functionality – method with a parameter a function pointer to another function that will then call back (via the passed pointer):**

Callback method

- **Asynchronous processing – the client continues processing without being blocked on a potentially lengthy synchronous call**
  - **the code calls a method, passing to it the callback method**
  - **the calling method starts a thread and returns immediately**
  - **the thread does the work, calling the callback function as needed**
- **Injecting custom code into a class's code path – the client specifies a method that will be called to perform custom processing**

1. Defining a delegate – the standard naming convention is to append the word Callback

```
[<attribute>] [<access_modifier>]
delegate <return_type> <delegate_name> ([<parameters>]);
```



2. Defining a callback method that takes as a parameter the delegate and executes the delegate (invokes the method it represents)

Delegate as a parameter

```
[<attribute>] [<access_modifier>]  <return_type> <callback_method>
                    ([<parameters>], <delegate_name> <delegate_instance>)
{
    <return_type> <variable> = <delegate_instance>([<parameters>]);
    //<return_type> <variable>=<delegate_instance>.Invoke([<parameters>]);
     return <variable>;
}
```

Executing the delegate

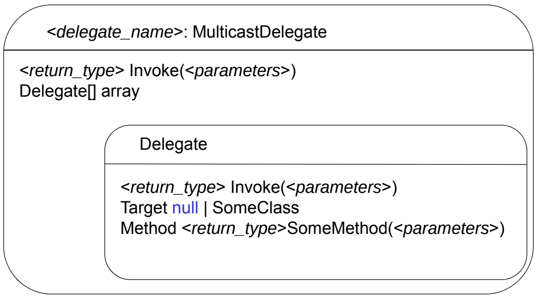
3. Defining a client method that has the same signature as the delegate

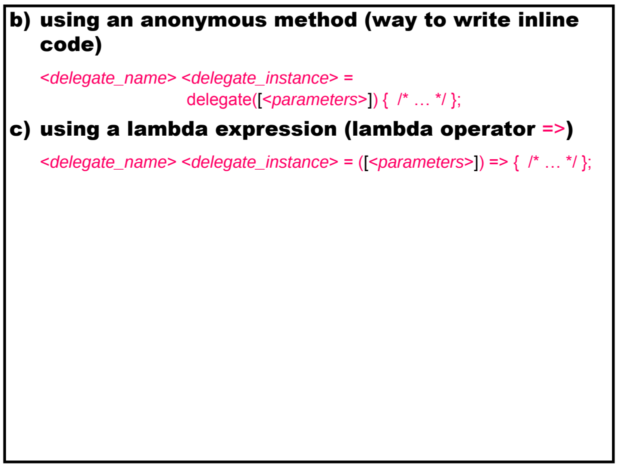
```
[<attribute>] [<access_modifier>]
<return_type> <client_method>(<[parameters>])
```

4. Instantiating the delegate

a) using the new **operator, passing it the name of the method**

```
<delegate_name> <delegate_instance> =
                                   new <delegate_name>(<method_name>);
```


<delegate_name>: MulticastDelegate
<return_type> Invoke(<parameters>)
Delegate[] array
Delegate
<return_type> Invoke(<parameters>)
Target null | SomeClass
Method <return_type>SomeMethod(<parameters>)

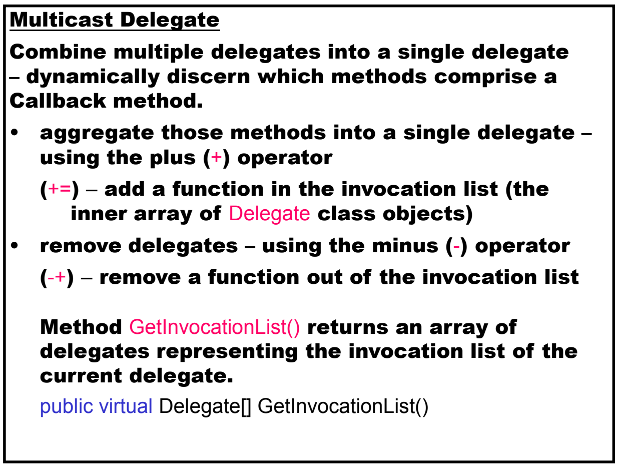


b) using an anonymous method (way to write inline code)

```
<delegate_name> <delegate_instance> =
                        delegate([<parameters>]) {  /* … */ };
```

c) using a lambda expression (lambda operator =>)

```
<delegate_name> <delegate_instance> = ([<parameters>]) => {  /* … */ };
```



Multicast Delegate

Combine multiple delegates into a single delegate – dynamically discern which methods comprise a Callback method.

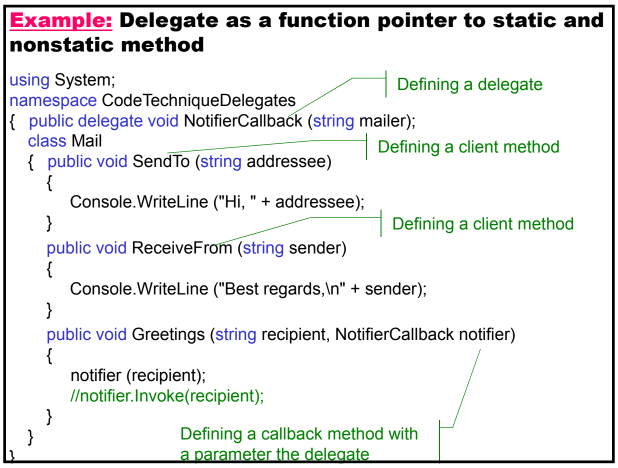
- **aggregate those methods into a single delegate – using the plus (+) operator**

  (+=) – add a function in the invocation list (the inner array of Delegate class objects)
- **remove delegates – using the minus (-) operator**

  (-+) – remove a function out of the invocation list

  Method GetInvocationList() returns an array of delegates representing the invocation list of the current delegate.

  `public virtual Delegate[] GetInvocationList()`



Example: Delegate as a function pointer to static and nonstatic method

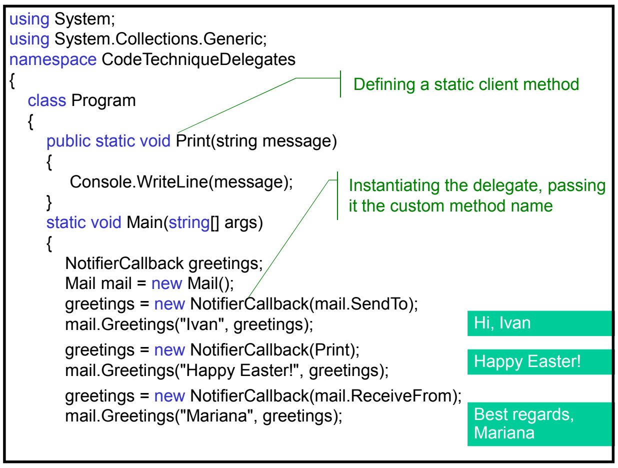
```
using System;
namespace CodeTechniqueDelegates
{   public delegate void NotifierCallback (string mailer);
    class Mail
    {   public void SendTo (string addressee)
        {
            Console.WriteLine ("Hi, " + addressee);
        }
        public void ReceiveFrom (string sender)
        {
            Console.WriteLine ("Best regards,\n" + sender);
        }
        public void Greetings (string recipient, NotifierCallback notifier)
        {
            notifier (recipient);
            //notifier.Invoke(recipient);
        }
    }
}
```

Defining a delegate

Defining a client method

Defining a client method

Defining a callback method with a parameter the delegate



```
using System;
using System.Collections.Generic;
namespace CodeTechniqueDelegates
{
    class Program
    {
        public static void Print(string message)
        {
            Console.WriteLine(message);
        }
        static void Main(string[] args)
        {
            NotifierCallback greetings;
            Mail mail = new Mail();
            greetings = new NotifierCallback(mail.SendTo);
            mail.Greetings("Ivan", greetings);
            greetings = new NotifierCallback(Print);
            mail.Greetings("Happy Easter!", greetings);
            greetings = new NotifierCallback(mail.ReceiveFrom);
            mail.Greetings("Mariana", greetings);
```

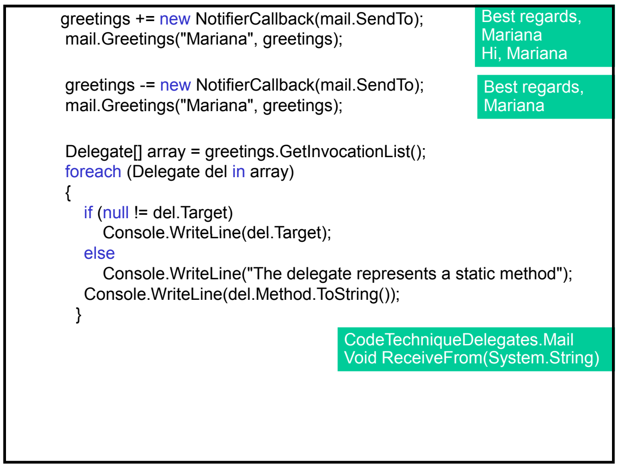
Defining a static client method

Instantiating the delegate, passing it the custom method name

```
Hi, Ivan
```

```
Happy Easter!
```

```
Best regards,
Mariana
```

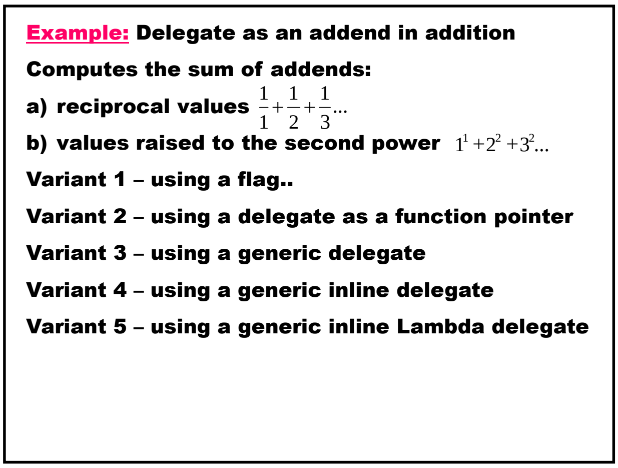


```
        greetings += new NotifierCallback(mail.SendTo);
         mail.Greetings("Mariana", greetings);

         greetings -= new NotifierCallback(mail.SendTo);
         mail.Greetings("Mariana", greetings);

         Delegate[] array = greetings.GetInvocationList();
         foreach (Delegate del in array)
         {
            if (null != del.Target)
                 Console.WriteLine(del.Target);
            else
                 Console.WriteLine("The delegate represents a static method");
            Console.WriteLine(del.Method.ToString());
          }
```

```
Best regards,
Mariana
Hi, Mariana
```

```
Best regards,
Mariana
```

```
CodeTechniqueDelegates.Mail
Void ReceiveFrom(System.String)
```



Example: Delegate as an addend in addition

Computes the sum of addends:

a) reciprocal values $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}...$

b) values raised to the second power $1^1+2^2+3^2...$

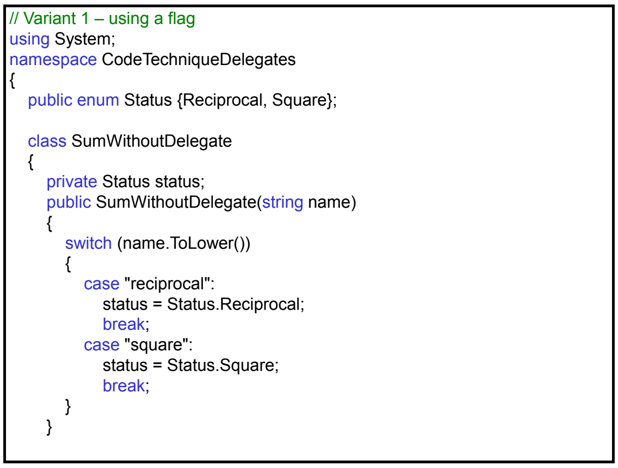
Variant 1 – using a flag..

Variant 2 – using a delegate as a function pointer

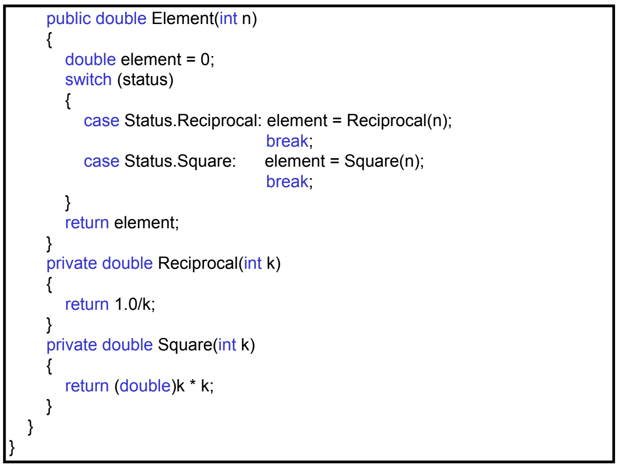
Variant 3 – using a generic delegate

Variant 4 – using a generic inline delegate

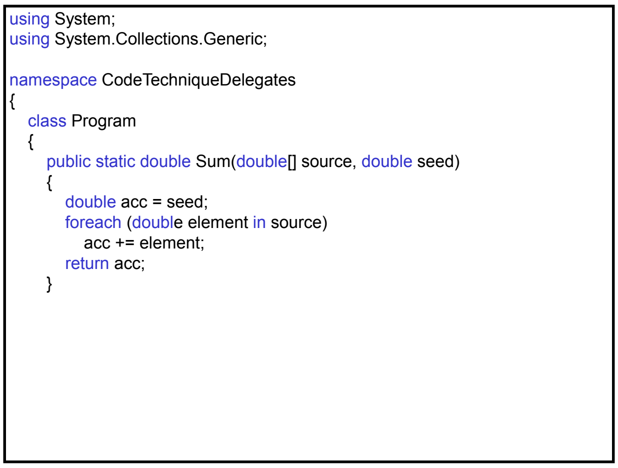
Variant 5 – using a generic inline Lambda delegate



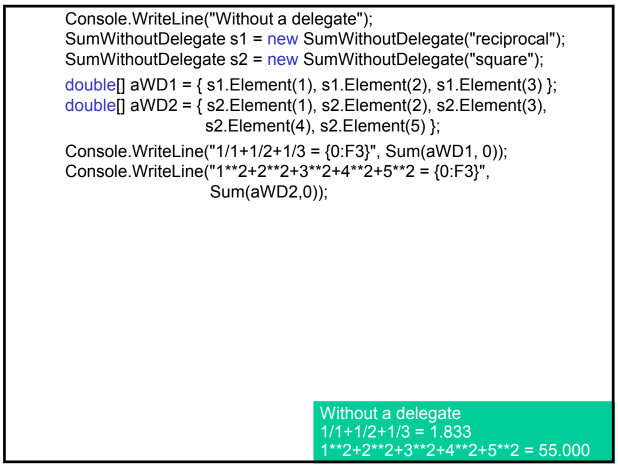
```
// Variant 1 – using a flag
using System;
namespace CodeTechniqueDelegates
{
   public enum Status {Reciprocal, Square};

   class SumWithoutDelegate
   {
      private Status status;
      public SumWithoutDelegate(string name)
      {
         switch (name.ToLower())
         {
            case "reciprocal":
               status = Status.Reciprocal;
               break;
            case "square":
               status = Status.Square;
               break;
         }
      }
```



```
      public double Element(int n)
      {
         double element = 0;
         switch (status)
         {
            case Status.Reciprocal: element = Reciprocal(n);
                                    break;
            case Status.Square:     element = Square(n);
                                    break;
         }
         return element;
      }
      private double Reciprocal(int k)
      {
         return 1.0/k;
      }
      private double Square(int k)
      {
         return (double)k * k;
      }
   }
}
```

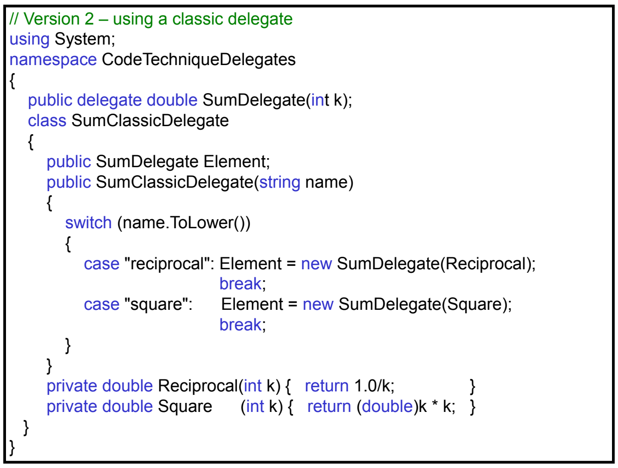


```
using System;
using System.Collections.Generic;

namespace CodeTechniqueDelegates
{
   class Program
   {
      public static double Sum(double[] source, double seed)
      {
         double acc = seed;
         foreach (double element in source)
            acc += element;
         return acc;
      }
```

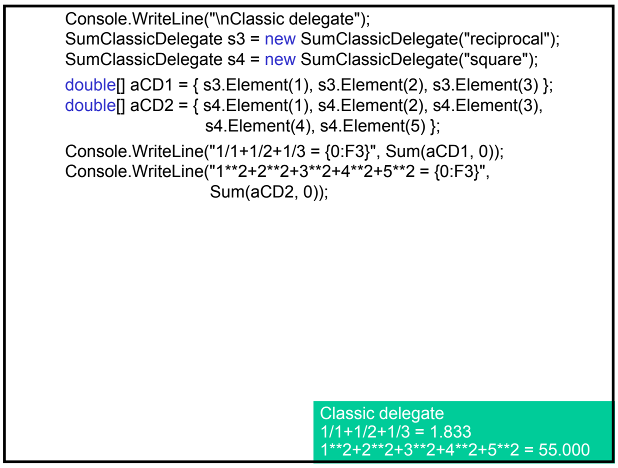


```
         Console.WriteLine("Without a delegate");
         SumWithoutDelegate s1 = new SumWithoutDelegate("reciprocal");
         SumWithoutDelegate s2 = new SumWithoutDelegate("square");
         double[] aWD1 = { s1.Element(1), s1.Element(2), s1.Element(3) };
         double[] aWD2 = { s2.Element(1), s2.Element(2), s2.Element(3),
                           s2.Element(4), s2.Element(5) };
         Console.WriteLine("1/1+1/2+1/3 = {0:F3}", Sum(aWD1, 0));
         Console.WriteLine("1**2+2**2+3**2+4**2+5**2 = {0:F3}",
                            Sum(aWD2,0));
```

```
Without a delegate
1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000
```



```
// Version 2 – using a classic delegate
using System;
namespace CodeTechniqueDelegates
{
   public delegate double SumDelegate(int k);
   class SumClassicDelegate
   {
      public SumDelegate Element;
      public SumClassicDelegate(string name)
      {
         switch (name.ToLower())
         {
            case "reciprocal": Element = new SumDelegate(Reciprocal);
                               break;
            case "square":     Element = new SumDelegate(Square);
                               break;
         }
      }
      private double Reciprocal(int k) {   return 1.0/k;           }
      private double Square     (int k) {   return (double)k * k;   }
   }
}
```



```
         Console.WriteLine("\nClassic delegate");
         SumClassicDelegate s3 = new SumClassicDelegate("reciprocal");
         SumClassicDelegate s4 = new SumClassicDelegate("square");
         double[] aCD1 = { s3.Element(1), s3.Element(2), s3.Element(3) };
         double[] aCD2 = { s4.Element(1), s4.Element(2), s4.Element(3),
                           s4.Element(4), s4.Element(5) };
         Console.WriteLine("1/1+1/2+1/3 = {0:F3}", Sum(aCD1, 0));
         Console.WriteLine("1**2+2**2+3**2+4**2+5**2 = {0:F3}",
                            Sum(aCD2, 0));
```

```
Classic delegate
1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000
```

Build-in delegates in .NET Framework

.NET offeres embedded Generic data type to many of build-in delegates.

Func<T, TResult> Encapsulates a method that has one parameter of type T and returns a value of the type specified by the TResult parameter.

Converter<TInput, TOutput> Represents a method that converts an object from one type TInput to another type TOutput.

```
// Version 3 – using a generic delegate
using System;
using System.Collections.Generic;
namespace CodeTechniqueDelegates
{
   class SumGenericDelegate
   {
      public Func<int, double> Element;
      public SumGenericDelegate(string name)
      {
         switch (name.ToLower())
         {
            case "reciprocal": Element = Reciprocal;
                               break;
            case "square":     Element = Square;
                               break;
         }
      }
      private double Reciprocal(int k) {   return 1.0/k;              }
      private double Square     (int k) {   return (double)k * k;   }
   }
}
```

```
Console.WriteLine("\nGeneric delegate");
SumGenericDelegate s5 = new SumGenericDelegate("reciprocal");
SumGenericDelegate s6 = new SumGenericDelegate("square");
double[] aGD1 = { s5.Element(1), s5.Element(2), s5.Element(3) };
double[] aGD2 = { s6.Element(1), s6.Element(2), s6.Element(3),
                  s6.Element(4), s6.Element(5) };
Console.WriteLine("1/1+1/2+1/3 = {0:F3}", Sum(aGD1, 0));
Console.WriteLine("1**2+2**2+3**2+4**2+5**2 = {0:F3}",
                   Sum(aGD2, 0));
```

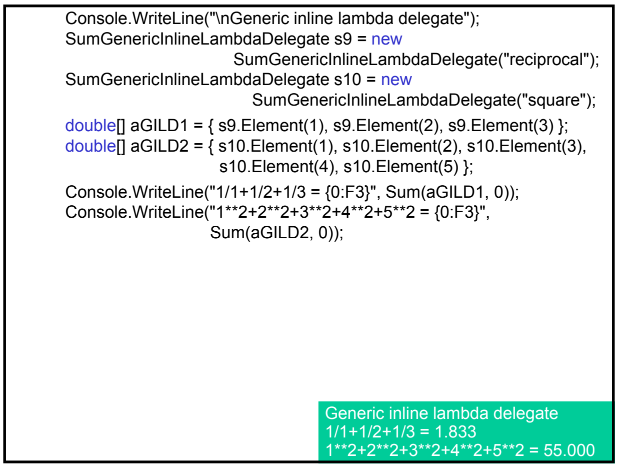
```
Generic delegate
1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000
```

```
// Variant 4 – using a generic inline delegate
using System;
using System.Collections.Generic;
namespace CodeTechniqueDelegates
{
   class SumGenericInlineDelegate
   {
      public Func<int, double> Element;
      public SumGenericInlineDelegate(string name)
      {
         switch (name.ToLower())
         {
            case "reciprocal": Element = delegate(int k) {   return 1.0 / k;            };
                               break;
            case "square":     Element = delegate(int k) {   return (double)k * k;   };
                               break;
         }
      }
   }
}
```

```
Console.WriteLine("\nGeneric inline delegate");
SumGenericInlineDelegate s7 = new SumGenericInlineDelegate("reciprocal");
SumGenericInlineDelegate s8 = new SumGenericInlineDelegate("square");
double[] aGID1 = { s7.Element(1), s7.Element(2), s7.Element(3) };
double[] aGID2 = { s8.Element(1), s8.Element(2), s8.Element(3),
                   s8.Element(4), s8.Element(5) };
 Console.WriteLine("1/1+1/2+1/3 = {0:F3}", Sum(aGID1, 0));
 Console.WriteLine("1**2+2**2+3**2+4**2+5**2 = {0:F3}", Sum(aGID2, 0));
```

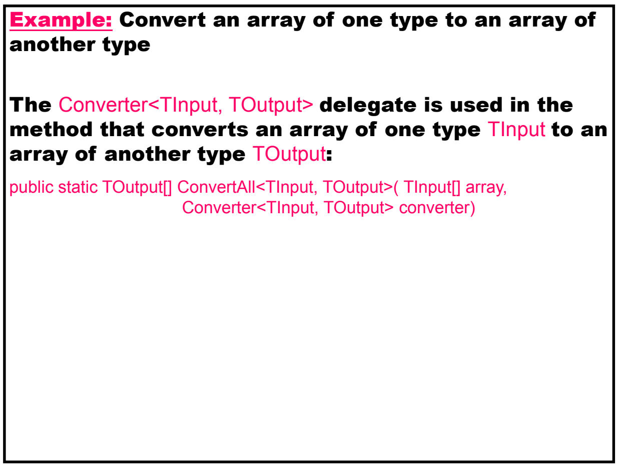
```
Generic inline delegate
1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000
```

```
// Variant 5 – using a generic lambda delegate
using System;
using System.Collections.Generic;
namespace CodeTechniqueDelegates
{
   class SumGenericInlineLambdaDelegate
   {
      public Func<int, double> Element;
      public SumGenericInlineLambdaDelegate(string name)
      {
         switch (name.ToLower())
         {
            case "reciprocal": Element = (k) => {   return 1.0 / k;            };
                               break;
            case "square":     Element = (k) => {   return (double)k * k;  };
                               break;
         }
      }
   }
}
```



```
        Console.WriteLine("\nGeneric inline lambda delegate");
        SumGenericInlineLambdaDelegate s9 = new
                           SumGenericInlineLambdaDelegate("reciprocal");
        SumGenericInlineLambdaDelegate s10 = new
                             SumGenericInlineLambdaDelegate("square");
        double[] aGILD1 = { s9.Element(1), s9.Element(2), s9.Element(3) };
        double[] aGILD2 = { s10.Element(1), s10.Element(2), s10.Element(3),
                         s10.Element(4), s10.Element(5) };
        Console.WriteLine("1/1+1/2+1/3 = {0:F3}", Sum(aGILD1, 0));
        Console.WriteLine("1**2+2**2+3**2+4**2+5**2 = {0:F3}",
                         Sum(aGILD2, 0));
```

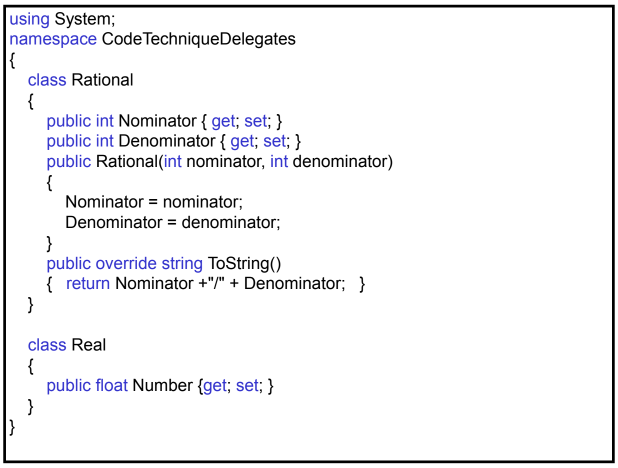
```
Generic inline lambda delegate
1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000
```



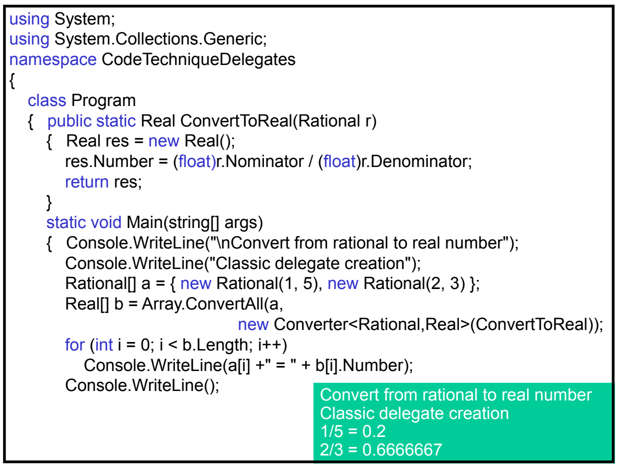
Example: Convert an array of one type to an array of another type

The Converter<TInput, TOutput> delegate is used in the method that converts an array of one type TInput to an array of another type TOutput:

```
public static TOutput[] ConvertAll<TInput, TOutput>( TInput[] array,
                         Converter<TInput, TOutput> converter)
```

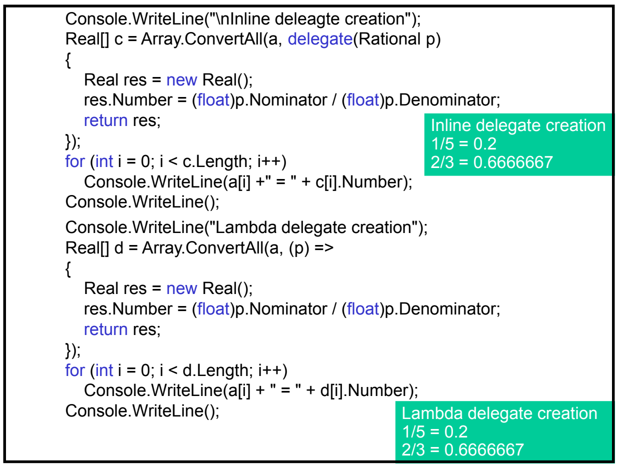


```
using System;
namespace CodeTechniqueDelegates
{
   class Rational
   {
      public int Nominator { get; set; }
      public int Denominator { get; set; }
      public Rational(int nominator, int denominator)
      {
         Nominator = nominator;
         Denominator = denominator;
      }
      public override string ToString()
      {   return Nominator +"/" + Denominator;   }
   }

   class Real
   {
      public float Number {get; set; }
   }
}
```



```
using System;
using System.Collections.Generic;
namespace CodeTechniqueDelegates
{
   class Program
   {  public static Real ConvertToReal(Rational r)
      {  Real res = new Real();
         res.Number = (float)r.Nominator / (float)r.Denominator;
         return res;
      }
      static void Main(string[] args)
      {  Console.WriteLine("\nConvert from rational to real number");
         Console.WriteLine("Classic delegate creation");
         Rational[] a = { new Rational(1, 5), new Rational(2, 3) };
         Real[] b = Array.ConvertAll(a,
                           new Converter<Rational,Real>(ConvertToReal));
         for (int i = 0; i < b.Length; i++)
            Console.WriteLine(a[i] +" = " + b[i].Number);
         Console.WriteLine();
```

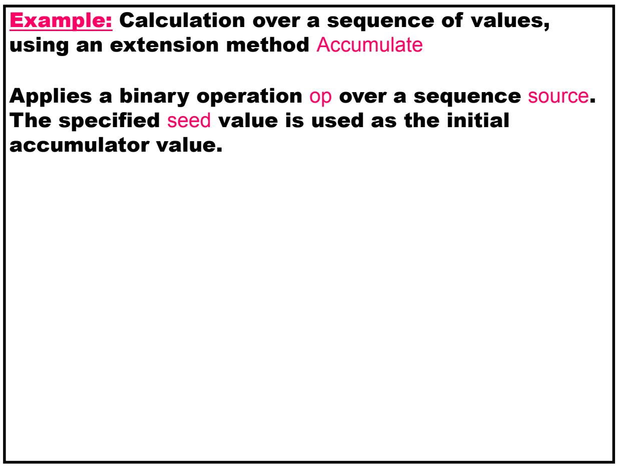
```
Convert from rational to real number
Classic delegate creation
1/5 = 0.2
2/3 = 0.6666667
```



```
         Console.WriteLine("\nInline deleagte creation");
         Real[] c = Array.ConvertAll(a, delegate(Rational p)
         {
            Real res = new Real();
            res.Number = (float)p.Nominator / (float)p.Denominator;
            return res;
         });
         for (int i = 0; i < c.Length; i++)
            Console.WriteLine(a[i] +" = " + c[i].Number);
         Console.WriteLine();
         Console.WriteLine("Lambda delegate creation");
         Real[] d = Array.ConvertAll(a, (p) =>
         {
            Real res = new Real();
            res.Number = (float)p.Nominator / (float)p.Denominator;
            return res;
         });
         for (int i = 0; i < d.Length; i++)
            Console.WriteLine(a[i] + " = " + d[i].Number);
         Console.WriteLine();
```

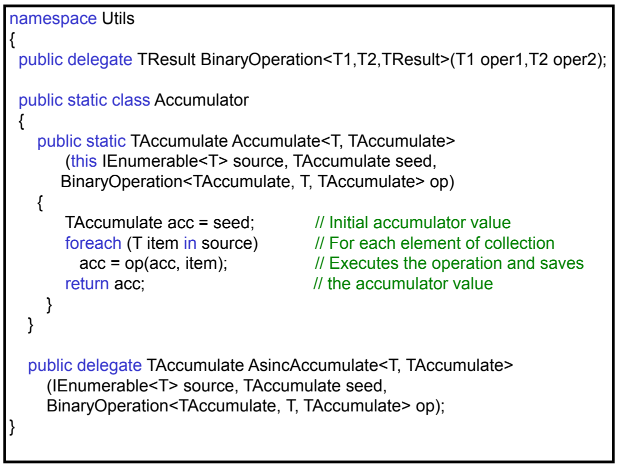
```
Inline delegate creation
1/5 = 0.2
2/3 = 0.6666667
```

```
Lambda delegate creation
1/5 = 0.2
2/3 = 0.6666667
```

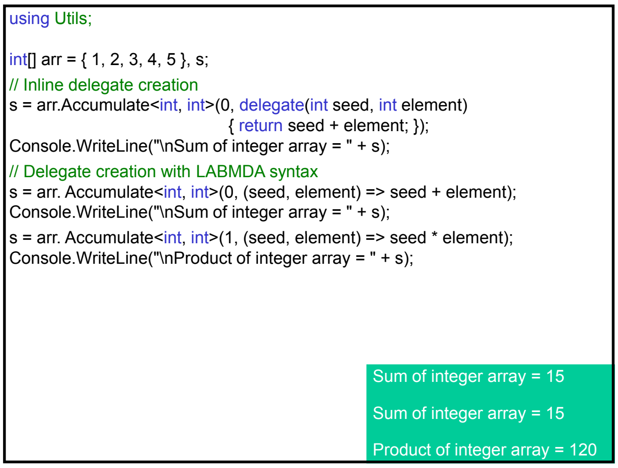


Example: Calculation over a sequence of values, using an extension method Accumulate

Applies a binary operation op over a sequence source. The specified seed value is used as the initial accumulator value.

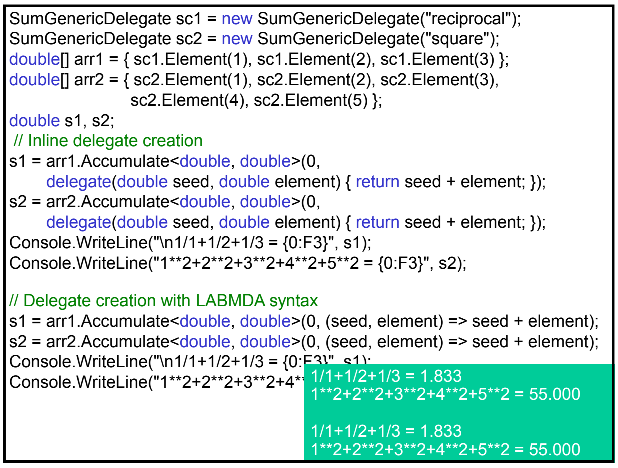


```
namespace Utils
{
  public delegate TResult BinaryOperation<T1,T2,TResult>(T1 oper1,T2 oper2);

  public static class Accumulator
  {
     public static TAccumulate Accumulate<T, TAccumulate>
           (this IEnumerable<T> source, TAccumulate seed,
          BinaryOperation<TAccumulate, T, TAccumulate> op)
     {
           TAccumulate acc = seed;          // Initial accumulator value
           foreach (T item in source)       // For each element of collection
              acc = op(acc, item);          // Executes the operation and saves
           return acc;                      // the accumulator value
       }
   }

   public delegate TAccumulate AsincAccumulate<T, TAccumulate>
       (IEnumerable<T> source, TAccumulate seed,
       BinaryOperation<TAccumulate, T, TAccumulate> op);
}
```



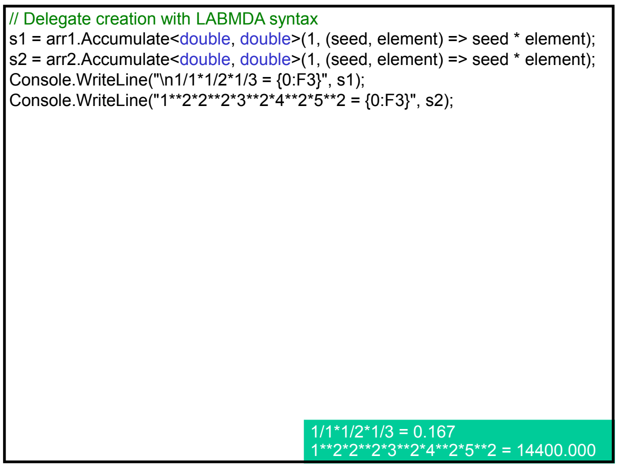
```
using Utils;

int[] arr = { 1, 2, 3, 4, 5 }, s;
// Inline delegate creation
s = arr.Accumulate<int, int>(0, delegate(int seed, int element)
                                  { return seed + element; });
Console.WriteLine("\nSum of integer array = " + s);
// Delegate creation with LABMDA syntax
s = arr. Accumulate<int, int>(0, (seed, element) => seed + element);
Console.WriteLine("\nSum of integer array = " + s);
s = arr. Accumulate<int, int>(1, (seed, element) => seed * element);
Console.WriteLine("\nProduct of integer array = " + s);
```

```
Sum of integer array = 15

Sum of integer array = 15

Product of integer array = 120
```



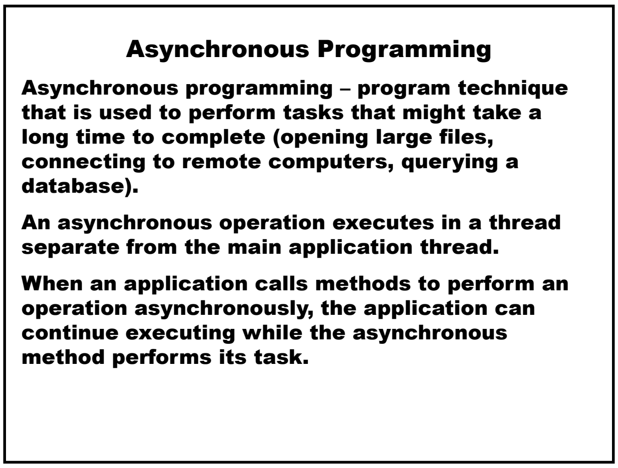
```
SumGenericDelegate sc1 = new SumGenericDelegate("reciprocal");
SumGenericDelegate sc2 = new SumGenericDelegate("square");
double[] arr1 = { sc1.Element(1), sc1.Element(2), sc1.Element(3) };
double[] arr2 = { sc2.Element(1), sc2.Element(2), sc2.Element(3),
                  sc2.Element(4), sc2.Element(5) };
double s1, s2;
 // Inline delegate creation
s1 = arr1.Accumulate<double, double>(0,
       delegate(double seed, double element) { return seed + element; });
s2 = arr2.Accumulate<double, double>(0,
       delegate(double seed, double element) { return seed + element; });
Console.WriteLine("\n1/1+1/2+1/3 = {0:F3}", s1);
Console.WriteLine("1**2+2**2+3**2+4**2+5**2 = {0:F3}", s2);

// Delegate creation with LABMDA syntax
s1 = arr1.Accumulate<double, double>(0, (seed, element) => seed + element);
s2 = arr2.Accumulate<double, double>(0, (seed, element) => seed + element);
Console.WriteLine("\n1/1+1/2+1/3 = {0:F3}", s1);
Console.WriteLine("1**2+2**2+3**2+4**
```

```
1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000

1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000
```



```
// Delegate creation with LABMDA syntax
s1 = arr1.Accumulate<double, double>(1, (seed, element) => seed * element);
s2 = arr2.Accumulate<double, double>(1, (seed, element) => seed * element);
Console.WriteLine("\n1/1*1/2*1/3 = {0:F3}", s1);
Console.WriteLine("1**2*2**2*3**2*4**2*5**2 = {0:F3}", s2);
```

```
1/1*1/2*1/3 = 0.167
1**2*2**2*3**2*4**2*5**2 = 14400.000
```

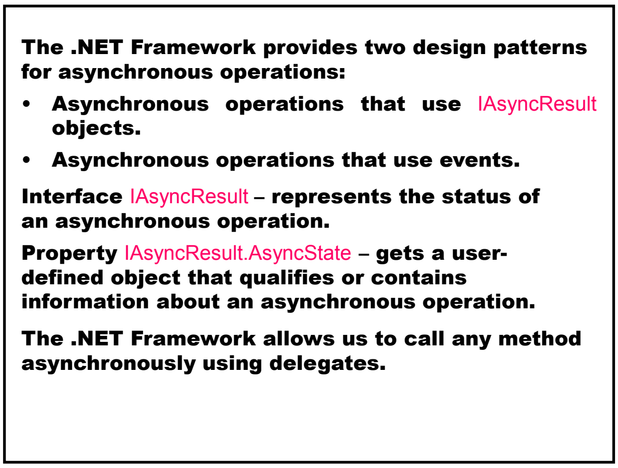


Asynchronous Programming

Asynchronous programming – program technique that is used to perform tasks that might take a long time to complete (opening large files, connecting to remote computers, querying a database).

An asynchronous operation executes in a thread separate from the main application thread.

When an application calls methods to perform an operation asynchronously, the application can continue executing while the asynchronous method performs its task.



The .NET Framework provides two design patterns for asynchronous operations:

- **Asynchronous operations that use** IAsyncResult **objects.**
- **Asynchronous operations that use events.**

Interface IAsyncResult **– represents the status of an asynchronous operation.**

Property IAsyncResult.AsyncState **– gets a user-defined object that qualifies or contains information about an asynchronous operation.**

The .NET Framework allows us to call any method asynchronously using delegates.

When defining a delegate the runtime system automatically defines the methods:

Invoke – **initiate a synchronous operation – the target method will called directly on the current thread.**

BeginInvoke – **initiate an asynchronous operation – the target method will be called on a thread from the thread pool.**

BeginInvoke **includes the following parameters:**

- **all input,** out, ref **and referential parameters**
- AsyncCallback **delegate that references a method to be called when the asynchronous call completes the callback function**

  public delegate void AsyncCallback (IAsyncResult ar);
- **user-defined object that passes information into the callback method**

BeginInvoke **returns immediately and does not wait for the asynchronous call to complete.**

BeginInvoke **returns an** IAsyncResult, **which can be used to monitor the progress of the asynchronous call.**

EndInvoke – **retrieves the results of the asynchronous call, blocks the calling thread until it completes.**

EndInvoke **includes the parameters:**

- out, ref **and referential parameters**
- IAsincResult **returned by** BeginInvoke

EndInvoke **returns the original method type.**

Example: Using a delegate to asynchronous method call

Asynchronous calculation of a sum of reciprocal values or of squares.

```
using System;
using System.Runtime.Remoting.Messaging;
using System.Threading;
using Utils;
namespace CodeTechniqueDelegates
{
   class Program
   {
      // Callback method – is called when the asynchronous operation ends
      public static void DoneCallback<T, TAccumulate>(IAsyncResult iar)
      {
         AsyncResult result = (AsyncResult)iar;
         AsincAccumulate<T, TAccumulate> caller =
                   (AsincAccumulate<T, TAccumulate>)result.AsyncDelegate;
         string formatString = (string)iar.AsyncState;
         TAccumulate sum = caller.EndInvoke(iar);
         Console.WriteLine(formatString, sum);
      }
```

Gets the delegate object on which the asynchronous call was invoked

Gets the last parameter of calling of BeginInvoke

```
      public static void DoSomething()
      {
         Console.WriteLine("\nDoSomething starts\n");
         long start = DateTime.Now.Ticks;
         //do something
         Thread.Sleep(3000);
         long end = DateTime.Now.Ticks;
         Console.WriteLine("\nDoSomething total time {0} seconds",
                           (end - start) / TimeSpan.TicksPerSecond);
      }
```

Represents the number of ticks in 1 second (10,000,000)

```
      static void Main(string[] args)
      {  …
         long start = DateTime.Now.Ticks;          // mark start time
         AsincAccumulate<double, double> sum1 =
               new AsincAccumulate<double,double>
                     (Accumulator.Accumulate<double, double>);
         IAsyncResult result = sum1.BeginInvoke(arr1, 0,
               delegate(double seed, double element) { return seed + element; },
               new AsyncCallback(DoneCallback<double, double>),
               "1/1*1/2*1/3 = {0:F3}");
         DoSomething();
         result = sum1.BeginInvoke(arr2, 0,
              delegate(double seed, double element) { return seed + element; },
              new AsyncCallback(DoneCallback<double, double>),
              "1**2*2**2*3**2*4**2*5**2 = {0:F3}");
         DoSomething();
         long end = DateTime.Now.Ticks;            // mark end time
         Console.WriteLine("\nTotal time {0} seconds", (end - start) / 10000000);
         Console.WriteLine("The main thread ends.");
      }
   }
}
```

```
DoSomething starts

1/1+1/2+1/3 = 1.833

DoSomething total time 3 seconds

DoSomething starts

1**2+2**2+3**2+4**2+5**2 = 55.000

DoSomething total time 3 seconds

Total time 6 seconds
The main thread ends.
```

```
// Synchronous call
s1 = sum1.Invoke(arr1, 0,
       delegate(double seed, double element) { return seed + element; });
s2 = sum1.Invoke(arr2, 0,
       delegate(double seed, double element) { return seed + element; });
Console.WriteLine("\n1/1*1/2*1/3 = {0:F3}", s1);
Console.WriteLine("1**2*2**2*3**2*4**2*5**2 = {0:F3}", s2);
```

```
1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000
```

Events

Event – the notification that can be generated by the class when something of interest happens.

Examples: mouse button click, keyboard key click, graphic button click.

Asynchronous event-handling:

- **multicast delegates**
- **keyword** event

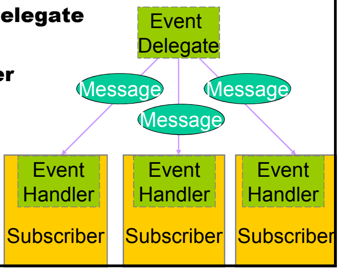
Publish/Subscribe design pattern:

- **Publisher – class publishes the event**
- **Subscriber – number of classes subscribe the event**

The runtime notifies each subscriber that the event has occurred and calls a method (event handler) defined by a delegate.

C# Event Model

- **Publisher/Subscriber design pattern**
  - **Event message**
  - **Event handler delegate**
  - **Event publisher**
  - **Event subscriber**



Class-Publisher

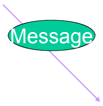
1. **Define a delegate with two arguments:**
   - **the object that raised the event (the publisher)**
   - **an event information object that inherits the** EventArgs **class**
2. **Define the event**

   [*<attribute>*] [*<access_modifier>*]
   event *<delegate_type> <event_name>*;
3. **Define a method that raises the event:**
   - **publishes the event**
   - **raises the event for all subscribers**

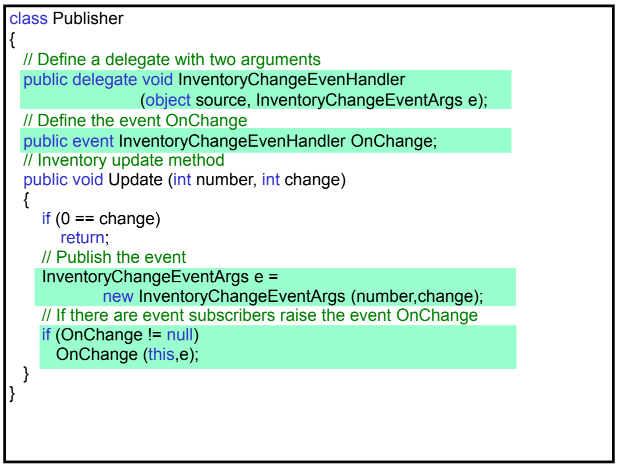
Class-Subscriber

1. **Add itself as a subscriber:**
   - **instantiate a new delegate**
   - **add to the list of subscribers using the (+=) compound assignment operator to avoid erasing any previous subscribers**
2. **Implement the event handler**

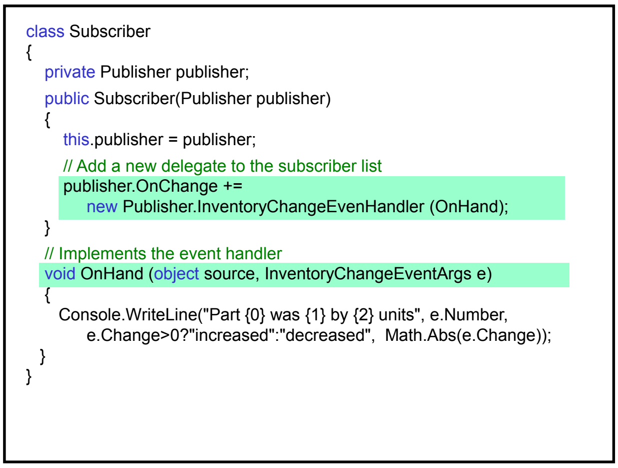
Example: Update inventory



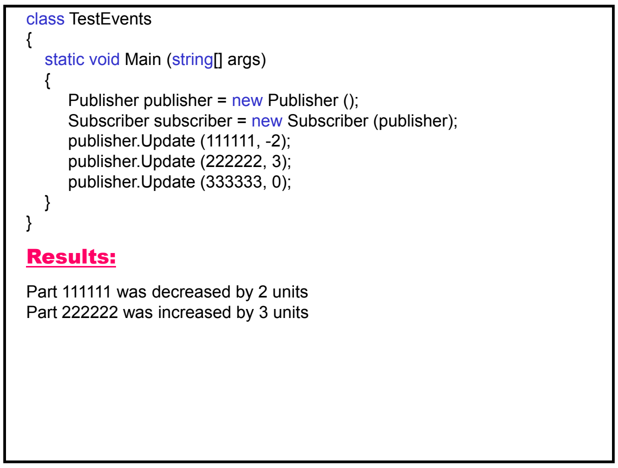
```
using System;
// Event information class (event message)
class InventoryChangeEventArgs: EventArgs
{
   private int number;
   public int Number
   {
      get   {   return number;   }
   }
   private int change;
   public int Change
   {
      get   {   return change;   }
   }
   public InventoryChangeEventArgs (int number, int change)
   {   this.number = number;
       this.change = change;
   }
}
```



```
class Publisher
{
  // Define a delegate with two arguments
  public delegate void InventoryChangeEvenHandler
                    (object source, InventoryChangeEventArgs e);
  // Define the event OnChange
  public event InventoryChangeEvenHandler OnChange;
  // Inventory update method
  public void Update (int number, int change)
  {
     if (0 == change)
        return;
     // Publish the event
     InventoryChangeEventArgs e =
              new InventoryChangeEventArgs (number,change);
     // If there are event subscribers raise the event OnChange
     if (OnChange != null)
       OnChange (this,e);
  }
}
```



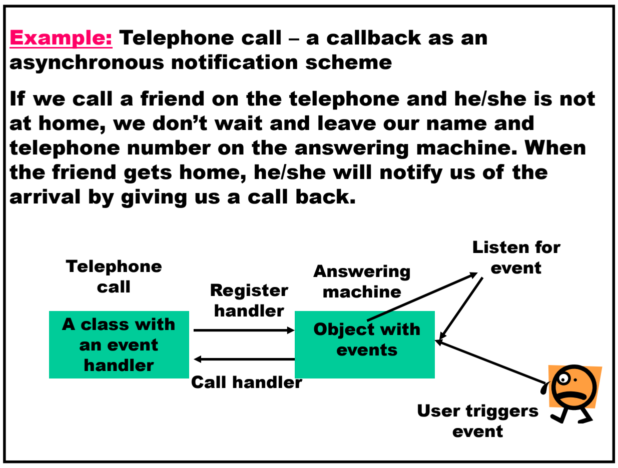
```
class Subscriber
{
   private Publisher publisher;
   public Subscriber(Publisher publisher)
   {
      this.publisher = publisher;
      // Add a new delegate to the subscriber list
      publisher.OnChange +=
         new Publisher.InventoryChangeEvenHandler (OnHand);
   }
   // Implements the event handler
   void OnHand (object source, InventoryChangeEventArgs e)
   {
     Console.WriteLine("Part {0} was {1} by {2} units", e.Number,
         e.Change>0?"increased":"decreased",  Math.Abs(e.Change));
   }
}
```



```
class TestEvents
{
   static void Main (string[] args)
   {
       Publisher publisher = new Publisher ();
       Subscriber subscriber = new Subscriber (publisher);
       publisher.Update (111111, -2);
       publisher.Update (222222, 3);
       publisher.Update (333333, 0);
   }
}
```

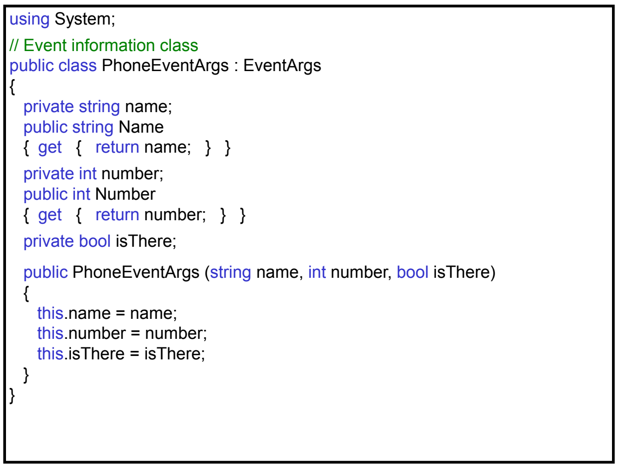
Results:

```
Part 111111 was decreased by 2 units
Part 222222 was increased by 3 units
```

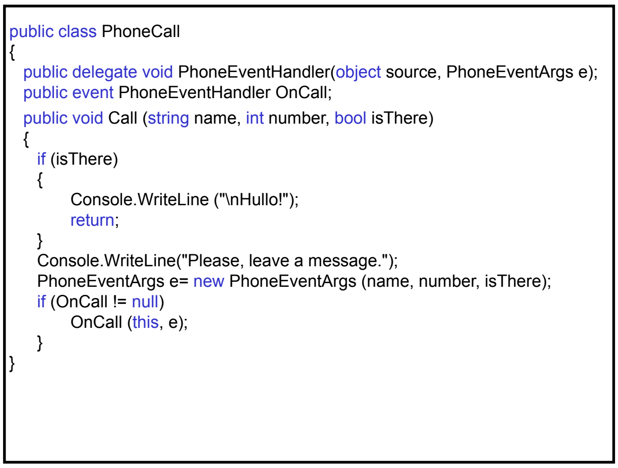


Example: Telephone call – a callback as an asynchronous notification scheme

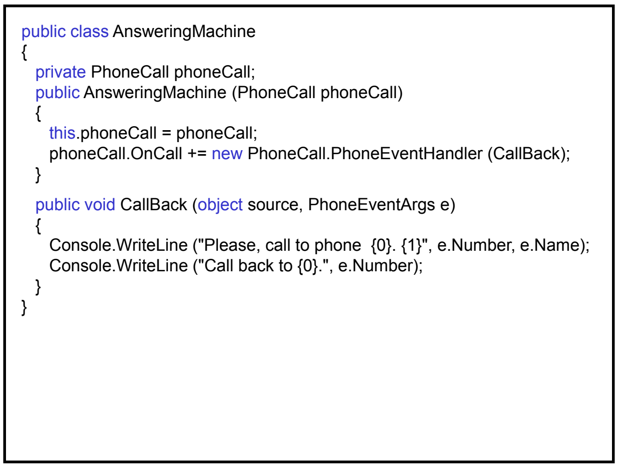
If we call a friend on the telephone and he/she is not at home, we don’t wait and leave our name and telephone number on the answering machine. When the friend gets home, he/she will notify us of the arrival by giving us a call back.



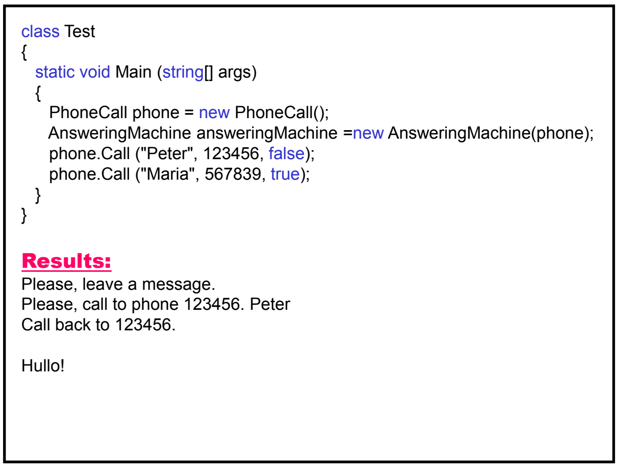
```
using System;
// Event information class
public class PhoneEventArgs : EventArgs
{
  private string name;
  public string Name
  {  get   {   return name;   }  }
  private int number;
  public int Number
  {  get   {   return number;   }  }
  private bool isThere;
  public PhoneEventArgs (string name, int number, bool isThere)
  {
    this.name = name;
    this.number = number;
    this.isThere = isThere;
  }
}
```



```
public class PhoneCall
{
  public delegate void PhoneEventHandler(object source, PhoneEventArgs e);
  public event PhoneEventHandler OnCall;
  public void Call (string name, int number, bool isThere)
  {
    if (isThere)
    {
          Console.WriteLine ("\nHullo!");
          return;
    }
    Console.WriteLine("Please, leave a message.");
    PhoneEventArgs e= new PhoneEventArgs (name, number, isThere);
    if (OnCall != null)
          OnCall (this, e);
    }
}
```



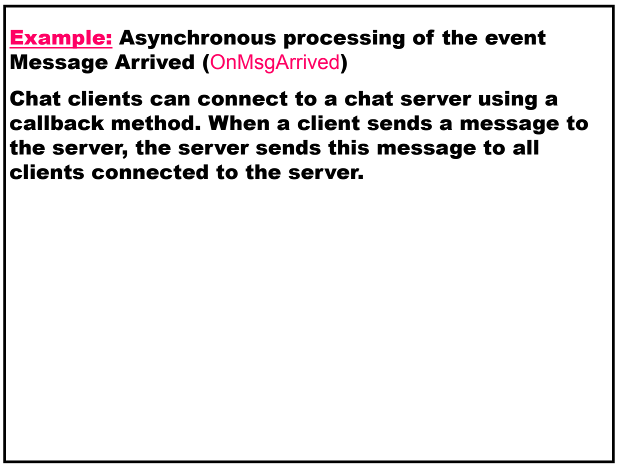
```
public class AnsweringMachine
{
   private PhoneCall phoneCall;
   public AnsweringMachine (PhoneCall phoneCall)
   {
      this.phoneCall = phoneCall;
      phoneCall.OnCall += new PhoneCall.PhoneEventHandler (CallBack);
   }
   public void CallBack (object source, PhoneEventArgs e)
   {
      Console.WriteLine ("Please, call to phone  {0}. {1}", e.Number, e.Name);
      Console.WriteLine ("Call back to {0}.", e.Number);
   }
}
```



```
class Test
{
   static void Main (string[] args)
   {
      PhoneCall phone = new PhoneCall();
      AnsweringMachine answeringMachine =new AnsweringMachine(phone);
      phone.Call ("Peter", 123456, false);
      phone.Call ("Maria", 567839, true);
   }
}
```

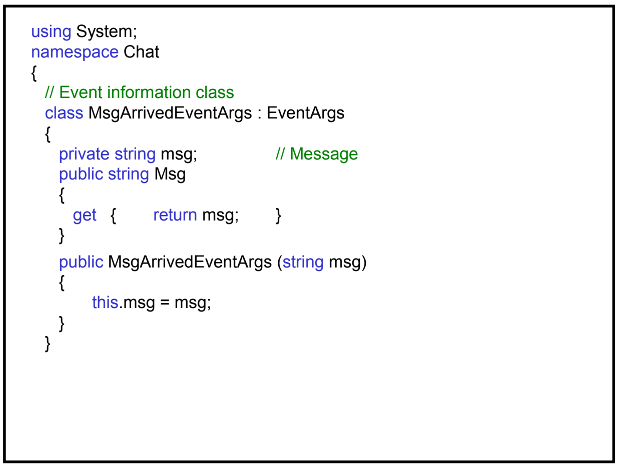
Results:

```
Please, leave a message.
Please, call to phone 123456. Peter
Call back to 123456.

Hullo!
```

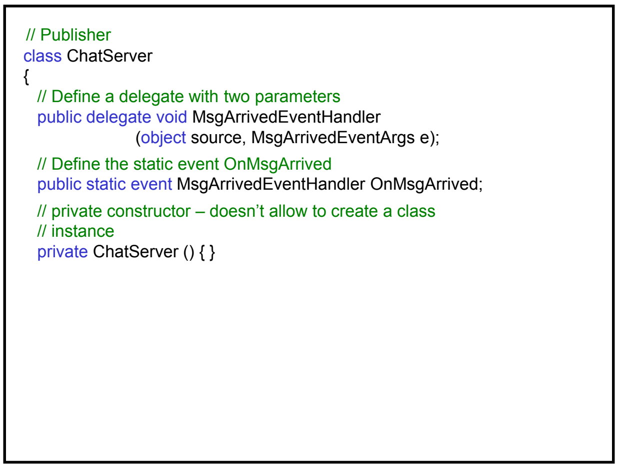


Example: Asynchronous processing of the event Message Arrived (OnMsgArrived)

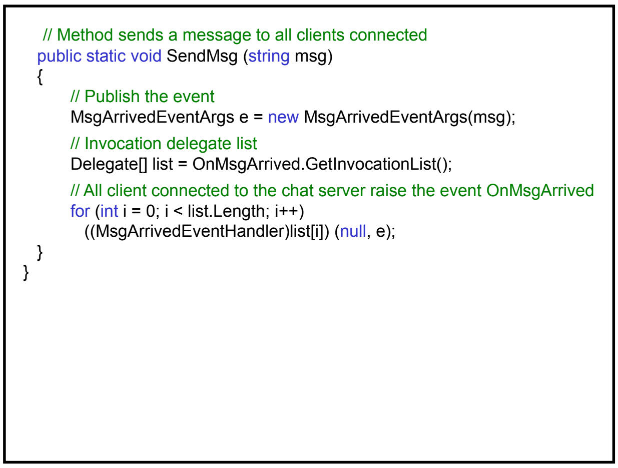
Chat clients can connect to a chat server using a callback method. When a client sends a message to the server, the server sends this message to all clients connected to the server.



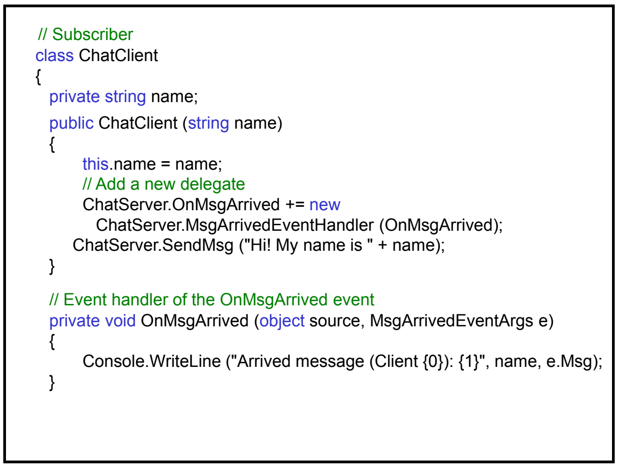
```
using System;
namespace Chat
{
   // Event information class
   class MsgArrivedEventArgs : EventArgs
   {
      private string msg;              // Message
      public string Msg
      {
         get  {       return msg;       }
      }
      public MsgArrivedEventArgs (string msg)
      {
            this.msg = msg;
      }
   }
```



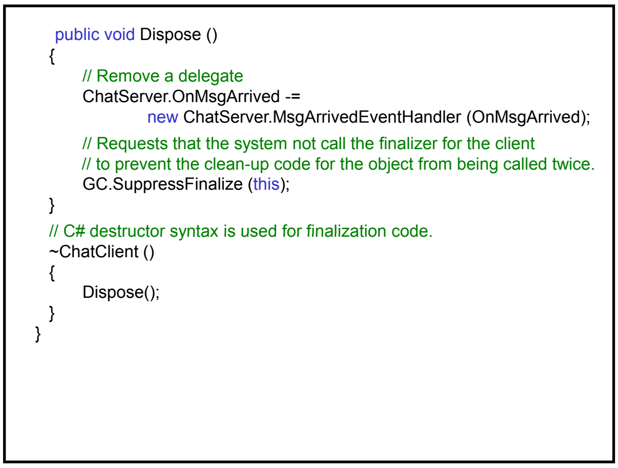
```
 // Publisher
class ChatServer
{
   // Define a delegate with two parameters
   public delegate void MsgArrivedEventHandler
                     (object source, MsgArrivedEventArgs e);
   // Define the static event OnMsgArrived
   public static event MsgArrivedEventHandler OnMsgArrived;
   // private constructor – doesn’t allow to create a class
   // instance
   private ChatServer () { }
```



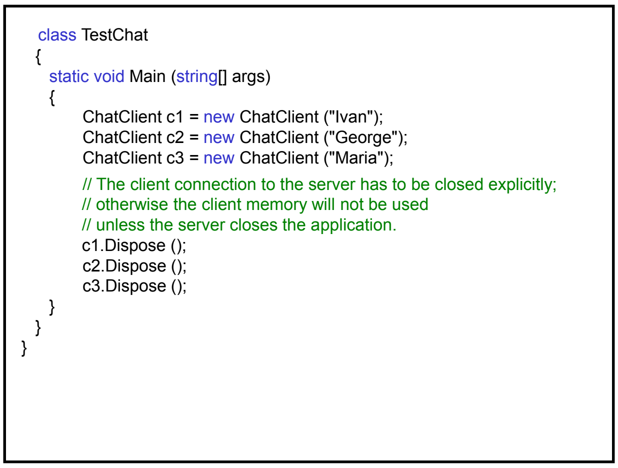
```
  // Method sends a message to all clients connected
  public static void SendMsg (string msg)
  {
        // Publish the event
        MsgArrivedEventArgs e = new MsgArrivedEventArgs(msg);
        // Invocation delegate list
        Delegate[] list = OnMsgArrived.GetInvocationList();
        // All client connected to the chat server raise the event OnMsgArrived
        for (int i = 0; i < list.Length; i++)
           ((MsgArrivedEventHandler)list[i]) (null, e);
  }
}
```



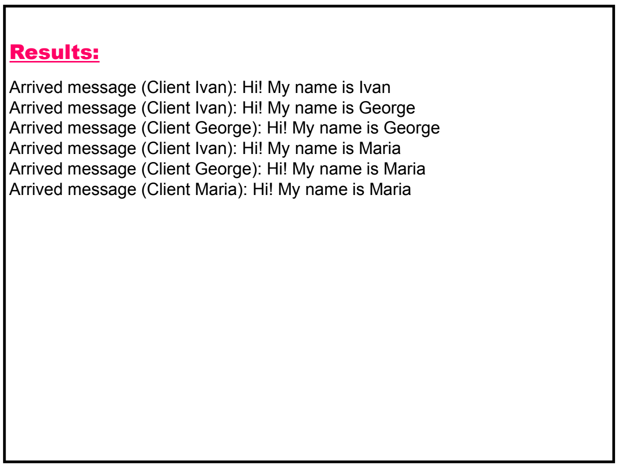
```
 // Subscriber
class ChatClient
{
  private string name;
  public ChatClient (string name)
  {
       this.name = name;
       // Add a new delegate
       ChatServer.OnMsgArrived += new
         ChatServer.MsgArrivedEventHandler (OnMsgArrived);
     ChatServer.SendMsg ("Hi! My name is " + name);
  }

  // Event handler of the OnMsgArrived event
  private void OnMsgArrived (object source, MsgArrivedEventArgs e)
  {
       Console.WriteLine ("Arrived message (Client {0}): {1}", name, e.Msg);
  }
```



```
  public void Dispose ()
  {
       // Remove a delegate
       ChatServer.OnMsgArrived -=
                 new ChatServer.MsgArrivedEventHandler (OnMsgArrived);
       // Requests that the system not call the finalizer for the client
       // to prevent the clean-up code for the object from being called twice.
       GC.SuppressFinalize (this);
  }
  // C# destructor syntax is used for finalization code.
  ~ChatClient ()
  {
       Dispose();
  }
}
```



```
  class TestChat
  {
    static void Main (string[] args)
    {
         ChatClient c1 = new ChatClient ("Ivan");
         ChatClient c2 = new ChatClient ("George");
         ChatClient c3 = new ChatClient ("Maria");
         // The client connection to the server has to be closed explicitly;
         // otherwise the client memory will not be used
         // unless the server closes the application.
         c1.Dispose ();
         c2.Dispose ();
         c3.Dispose ();
    }
  }
}
```



Results:

```
Arrived message (Client Ivan): Hi! My name is Ivan
Arrived message (Client Ivan): Hi! My name is George
Arrived message (Client George): Hi! My name is George
Arrived message (Client Ivan): Hi! My name is Maria
Arrived message (Client George): Hi! My name is Maria
Arrived message (Client Maria): Hi! My name is Maria
```