Делегати

Делегат: човек, изпратен или оторизиран да представлява други хора. Делегатът в езика C# представя метод, който може да бъде извикан отдалечено или да бъде предаден като параметър.

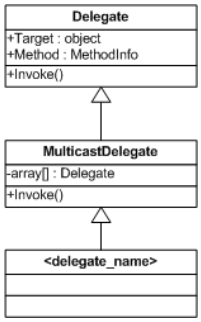
- **референтен тип, абстракция на метод;**
- **разглежда се като елегантен метод – представя метод с дадена сигнатура;**
- **еквивалент на обект на функция;**
- **производен клас на типа System.MulticastDelegate, наследник на System.Delegate;**
- **осигурява асинхронна обработка на събития;**
- **използва се за т.н. callback функционалност – методи с параметър указател към функция, която се извиква чрез този указател:**

Callback метод

- **Асинхронна обработка – клиентът продължава обработката, без да бъде блокиран от потенциално дълго синхронно извикване**
  - **кодът извиква метод, предавайки му callback метода**
  - **извикващият метод стартира нишка и прекъсва веднага**
  - **нишката извършва работата, извиквайки callback метода при необходимост**
- **Инжектиране на потребителски код в йерархията на класовете – клиентът определя метод, който ще бъде извикан, за да се извърши потребителска обработка**

1. Дефиниране на делегат – стандартна конвенция е името му да завършва със Callback

```
[<атрибут>] [<модификатор_за_достъп>]
delegate <върнат_тип> <име_на_делегат> ([<параметри>]);
```



Свойство Target – връща инстанцията на класа, за която текущият делегат извиква метода на инстанцията;
Свойство Method – връща метода, представен чрез делегата;
Метод Invoke – синхронно извикване на делегат.

Delegate array[] **– списък от извикани делегати.**

2. Дефиниране на callback метод, който има за параметър делегата и изпълнява делегата (извиква метода, който той представя)

```
[<атрибут>] [<модификатор_за_достъп>] <върнат_тип>
<callback_метод>
([<параметри>], <име_на_делегат> <инстанция_на_делегат)
{
  <върнат_тип> <променлива> =
                        <инстанция_на_делегата>([<параметри>]);
  //<върнат_тип> <променлива> =
  //                    <инстанция_на_делегата>.Invoke([<параметри>]);
  return <променлива>;
}
```

Делегат като параметър

Изпълнение на делегата

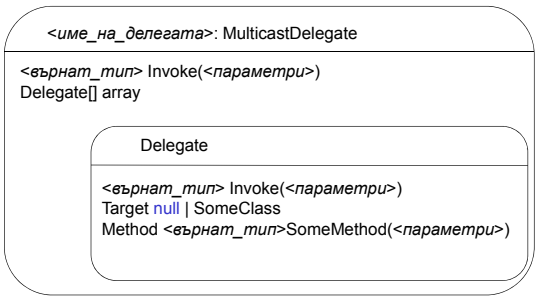
3. Дефиниране на клиентски метод, който има същата сигнатура като тази на делегата

```
[<атрибут>] [<модификатор_на_достъп>]
<върнат_тип> <клиентски_метод>(<[параметри>])
```

4. Създаване на инстанция на делегата

- **използвайки оператор new, като му предава името на метода**

```
<име_на _делегат> <инстанция_на_делегат> =
                        new <име_на_делегат>(<име_на_метод>);
```



- **използвайки анонимен метод (начин за писане на inline код)**

  <име_на_делегат> <инстанция_на_делегат> =
  delegate([<параметри>]) { / … */ };*

- **използвайки ламбда израз (ламбда оператор =>)**

  <име_на_делегат> <инстанция_на_делегат> =
  ([<параметри>]) => { / … */ };*

Комбиниран делегат (Multicast Delegate)

Комбинира много делегати в единствен делегат – динамично разпознава кои методи съдържат callback метод.

- **агрегиране на тези методи в единствен делегат – използвайки оператора плюс (+)**

  (+=) – добавя функция в списъка на извикванията (вътрешния масив от обекти от класа Delegate)

- **премахване на делегати – използвайки оператора минус (-)**

  (-+) – премахва функция от списъка на извикванията

Методът GetInvocationList() връща масив от делегати, представящ списъка на извикванията на текущия делегат.

public virtual Delegate[] GetInvocationList()

Пример: Делегат като указател към функция към статичен и нестатичен метод

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

Дефиниране на делегат

Дефиниране на клиентски метод

Дефиниране на клиентски метод

Дефиниране на callback метод с параметър делегата

```
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("Иван", greetings);
            greetings = new NotifierCallback(Print);
            mail.Greetings("Честита пролет!", greetings);
            greetings = new NotifierCallback(mail.ReceiveFrom);
            mail.Greetings("Мариана", greetings);
```

Дефиниране на статичен клиентски метод

Създаване на инстанция на делегата, предавайки му името на потребителски метод

Здравей, Иван

Честита пролет!

Много поздрави, Мариана

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

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

        Delegate[] array = greetings.GetInvocationList();
        foreach (Delegate del in array)
        {
            if (null != del.Target)
                Console.WriteLine(del.Target);
            else
                Console.WriteLine("Делегатът представя статичен метод");
            Console.WriteLine(del.Method.ToString());
        }
}
```

Много поздрави, Мариана
Здравей, Мариана

Много поздрави, Мариана

CodeTechniqueDelegates.Mail
Void ReceiveFrom(System.String)

Пример: Делегат като събираемо в събиране

Изчислява сумата от събираеми:

- **реципрочни стойности** $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}\ldots$
- **стойности, повдигнати на квадрат** $1^1+2^2+3^2\ldots$

Вариант 1 – използвайки флаг.

Вариант 2 – използвайки делегат като указател към функция

Вариант 3 – използвайки параметризиран делегат

Вариант 4 – използвайки параметризиран inline делегат

Вариант 5 – използвайки параметризиран inline ламбда делегат

```
// Вариант 1 – използвайки флаг
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("Без делегат");
      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));
```

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

```
// Версия 2 – използване на класически делегат
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("\nКласически делегат");
      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));
```

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

Вградени делегати в .NET Framework

.NET предлага вградени типизирани делегати.

Func<T, TResult> Капсулира метод, който има един параметър от тип T и връща стойност от типа, определен от параметъра TResult.

Converter<TInput, TOutput> Представя метод, който конвертира обект от един тип TInput в друг тип TOutput.

```
// Версия 3 – използване на параметризиран делегат
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("\nПараметризиран делегат");
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));
```

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

```
// Вариант 4 – използвайки параметризиран inline делегат
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("\nПараметризиран inline делегат");
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));
```

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

```
// Вариант 5 – използвайки параметризиран ламбда делегат
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("\nПараметризиран inline ламбда делегат");
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));
```

```
Параметризиран inline ламбда делегат
1/1+1/2+1/3 = 1.833
1**2+2**2+3**2+4**2+5**2 = 55.000
```

Пример: Преобразува масив от един тип в масив от друг тип

Делегатът Converter<TInput, TOutput> **се използва в метода** ConvertAll**, който преобразува масив от един тип** TInput **в масив от друг тип** 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("\nПреобразува от рационално в реално число");
         Console.WriteLine("Създаване на класически делегат");
         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();
```

```
Преобразува от рационално в реално число
Създаване на класически делегат
1/5 = 0.2
2/3 = 0.6666667
```

```
Console.WriteLine("\nСъздаване на inline делегат");
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("Създаване на ламбда делегат");
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();
```

```
Създаване на inline делегат
1/5 = 0.2
2/3 = 0.6666667
```

```
Създаване на ламбда делегат
1/5 = 0.2
2/3 = 0.6666667
```

Пример: Изчислява последователност от стойности, използвайки разширен метод Accumulate

Прилага операция с два операнда op **за последователност** source**. Стойността** seed **се използва като начална стойност на акумулатора.**

```
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;          // Начална стойност на акумулатора
         foreach (T item in source)       // За всеки елемент от колекцията
           acc = op(acc, item);           // изпълнява операцията и запазва
         return acc;                      // стойността на акумулатора
     }
  }

   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;
// Създаване на inline делегат
s = arr.Accumulate<int, int>(0, delegate(int seed, int element)
                                { return seed + element; });
Console.WriteLine("\nСума от целочислен масив = " + s);
// Създаване на ламбда делегат
s = arr. Accumulate<int, int>(0, (seed, element) => seed + element);
Console.WriteLine("\nСума от целочислен масив = " + s);
s = arr. Accumulate<int, int>(1, (seed, element) => seed * element);
Console.WriteLine("\nПроизведение от целочислен масив = " + s);
```

```
Сума от целочислен масив = 15

Сума от целочислен масив = 15

Произведение от целочислен масив = 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;
 // Създаване на inline делегат
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);

// Създаване на ламбда делегат
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
```

```
// Създаване на ламбда делегат
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
```

Асинхронно програмиране

Асинхронно програмиране – програмна техника, която се използва за задачи, изискващи дълго време за изпълнение (отваряне на големи файлове, свързване към отдалечени компютри, заявки към бази от данни).

Асинхронната операция се изпълнява в нишка, отделно от нишката на главното приложение.

Когато едно приложение извиква метод, за да изпълни операция асинхронно, приложението може да продължи да се изпълнява, докато асинхронният метод изпълнява своята задача.

.NET Framework осигурява два шаблона за асинхронни операции:

- **асинхронни операции, които използват IAsyncResult обекти;**
- **асинхронни операции, които използват събития.**

Интерфейс IAsyncResult – представя статуса на асинхронна операция.

Свойство IAsyncResult.AsyncState – дава обект, дефиниран от потребителя, който съдържа информация за асинхронната операция.

.NET Framework позволява всеки метод да бъде извикан асинхронно, използвайки делегати.

Когато дефинираме делегат, изпълнителната система автоматично дефинира методите:

Invoke – **стартира синхронна операция – методът се извиква директно в текущата нишка.**

BeginInvoke – **стартира асинхронна операция – методът се извиква в нишка от пула с нишки.**

BeginInvoke **включва следните параметри:**

- **всички входни,** out**,** ref **и референтни параметри;**
- AsyncCallback **делегат, който се отнася към метод, който се извиква, когато асинхронното извикване на callback функцията завърши;**

  public delegate void AsyncCallback (IAsyncResult ar);
- **обект, дефиниран от потребителя, който предава информация към callback метода.**

BeginInvoke **завършва незабавно и не чака завършването на асинхронното извикване.**

BeginInvoke **връща** IAsyncResult**, който може да се използва, за да се наблюдава напредъка на асинхронното извикване.**

EndInvoke – **връща резултатите от асинхронното извикване, блокира извикващата нишка, докато завърши.**

EndInvoke **включва параметрите:**

- out**,** ref **и референтни параметри;**
- IAsincResult**, върнат от** BeginInvoke**.**

EndInvoke **връща типа на оригиналния метод.**

Пример: Използване на делегат за асинхронно извикване на метод

Асинхронно изчисляване на сума от реципрочни стойности или от стойности, повдигнати на квадрат.

```
using System;
using System.Runtime.Remoting.Messaging;
using System.Threading;
using Utils;
namespace CodeTechniqueDelegates
{
  class Program
  {
    // Callback метод – извиква се, когато завърши асинхронната операция
    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);
    }
```

Дава обекта на делегат, за който е асинхронното извикване

Дава последния параметър от извикването на BeginInvoke

```
    public static void DoSomething()
    {
      Console.WriteLine("\nDoSomething стартира\n");
      long start = DateTime.Now.Ticks;
      // прави нещо
      Thread.Sleep(3000);
      long end = DateTime.Now.Ticks;
      Console.WriteLine("\nDoSomething общо време {0} секунди",
                        (end - start) / TimeSpan.TicksPerSecond);
    }
```

Представя броя на тактовете в 1 секунда (10,000,000)

```
    static void Main(string[] args)
    {  …
      long start = DateTime.Now.Ticks;          // маркира началното време
      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;            // маркира крайното време
      Console.WriteLine("\nОбщо време {0} секунди",(end-start)/10000000);
      Console.WriteLine("Главната нишка завършва.");
    }
  }
}
```

```
DoSomething стартира
1/1+1/2+1/3 = 1.833
DoSomething общо време 3 секунди
DoSomething стартира
1**2+2**2+3**2+4**2+5**2 = 55.000
DoSomething общо време 3 секунди
Общо време 6 секунди
Главната нишка завършва.
```

```
// Синхронно извикване
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
```

Събития

Събитие – интересна случка за потребителя.

Примери: натискане бутон на мишката, клавиш от клавиатурата, графичен бутон или плъзгач.

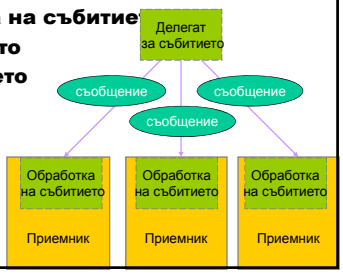
Асинхронна обработка на събития:
- **комбинирани делегати;**
- **ключова дума event.**

Шаблон източник/приемник:
- **източник – публикува събитието;**
- **приемник – улавя събитието.**

Изпълнителната система уведомява всички абонати при възникване на дадено събитие и извиква метод, дефиниран чрез делегат.

C# Модел на събития

Шаблон източник/приемник
- **съобщение на събитието**
- **делегат за обработка на събитие**
- **източник на събитието**
- **приемник на събитието**



Клас-източник

1. **Дефинира делегат с два параметъра:**
   - **обект-източник, пораждащ събитието;**
   - **обект с информация за събитието – наследник на класа EventArgs;**
2. **Дефинира събитието.**

   [<атрибут>] [<модификатор_за_достъп>] event <тип_на_делегат> <име_на_събитие>;
3. **Дефинира метод, който вдига събитието:**
   - **публикува събитието;**
   - **вдига събитието за всички абонати.**

Клас-приемник

1. **Добавя се като приемник:**
   - **създава нов делегат;**
   - **добавя се чрез комбинирания оператор за събиране (+=), за да не се изтрие предишният приемник;**
2. **Реализира манипулатор на събитието.**

Пример: Обновяване на склад


```
using System;
// Клас с информация за събитието
class InventoryChangeEventArgs: EventArgs
{
  private int number;                          // Номер
  public int Number
  {
    get  {  return number;  }
  }
  private int change;                          // Промяна
  public int Change
  {
    get  {  return change;  }
  }
  public InventoryChangeEventArgs (int number, int change)
  {
     this.number = number;
     this.change = change;
  }
}
```

```
class Publisher                                  // Клас-източник
{
  // Дефиниране на делегат с два параметъра
  public delegate void InventoryChangeEvenHandler
                                  (object source, InventoryChangeEventArgs e);
  // Дефиниране на събитието ОБНОВЯВАНЕ на склад
  public event InventoryChangeEvenHandler OnChange;
  // Метод за обновяване на склада
  public void Update (int number, int change)
  {
     if (0 == change)
        return;
     // Публикува събитието ОБНОВЯВАНЕ
     InventoryChangeEventArgs e =
                              new InventoryChangeEventArgs (number,change);
     // Ако има приемници на събитието, вдига събитието ОБНОВЯВАНЕ
     if (OnChange != null)
        OnChange (this,e);
  }
}
```

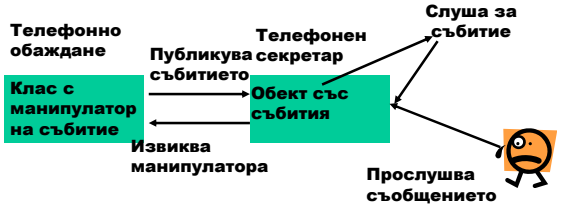
```
class Subscriber                                 // Клас-приемник
{
   private Publisher publisher;
   public Subscriber(Publisher publisher)
   {
      this.publisher = publisher;
      // Добавя нов делегат
      publisher.OnChange +=
         new Publisher.InventoryChangeEvenHandler (OnHand);
   }
   // Манипулатор на събитието ОБНОВЯВАНЕ
   void OnHand (object source, InventoryChangeEventArgs e)
   {
      Console.WriteLine("Артикул {0} е {1} с {2} броя",e.Number,
      e.Change>0?"увеличен":"намален",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);
   }
}
```

Резултати:

```
Артикул 111111 е намален с 2 броя
Артикул 222222 е увеличен с 3 броя
```

Пример: Асинхронна обработка – обаждане по телефон на приятел.
Приятелят не е в къщи и вместо да чакаме неговото завръщане, оставяме името и телефонния си номер на телефонния секретар. Когато приятелят се върне в къщи, прослушва съобщението и ни съобщава за завръщането си чрез обратно позвъняване по телефона.




```
using System;
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 ("\nАло!");
        return;
    }
    Console.WriteLine("Моля, оставете съобщение.");
    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 ("Моля, обади се на {0}. {1}", e.Number, e.Name);
    Console.WriteLine ("Обратно позвъняване на {0}.", e.Number);
  }
}
```

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

Резултати:

```
Моля, оставете съобщение.
Моля, обади се на 123456. Петър
Обратно позвъняване на 123456.

Ало!
```

Пример: Асинхронна обработка на събитие ПРИСТИГАНЕ НА СЪОБЩЕНИЕ.

Към „чат“ сървър могат да се свързват много клиенти чрез callback метод. Когато един клиент изпрати съобщение към сървъра, сървърът препраща съобщението към всички свързали се клиенти.

```
using System;
namespace Chat
{
  // Клас с информация за събитието ПРИСТИГАНЕ НА СЪОБЩЕНИЕ
  class MsgArrivedEventArgs : EventArgs
  {
    private string msg;              // Съобщение
    public string Msg
    {
      get  {      return msg;      }
    }
    public MsgArrivedEventArgs (string msg)
    {
        this.msg = msg;
    }
  }
```

```
  // Клас-източник
  class ChatServer
  {
    // Дефиниране на делегат с два параметъра
    public delegate void MsgArrivedEventHandler
                              (object source, MsgArrivedEventArgs e);
    // Дефиниране на събитието ПРИСТИГАНЕ НА СЪОБЩЕНИЕ
    public static event MsgArrivedEventHandler OnMsgArrived;
    // private конструктор – не позволява да се създава екземпляр на
    // класа
    private ChatServer () { }
```

```
    // Метод за изпращане на съобщение към всички
    // свързали се клиенти
    public static void SendMsg (string msg)
    {
        // Публикува събитието ПРИСТИГАНЕ НА СЪОБЩЕНИЕ
        MsgArrivedEventArgs e = new MsgArrivedEventArgs (msg);
        // Списък от извиканите делегати
        Delegate[] list = OnMsgArrived.GetInvocationList();
        // Всички клиенти, свързали се към сървъра, вдигат
        // събитието ПРИСТИГАНЕ НА СЪОБЩЕНИЕ
        for (int i = 0; i < list.Length; i++)
          ((MsgArrivedEventHandler)list[i]) (null, e);
    }
  }
```

```
// Клас-приемник
class ChatClient
{
   private string name;            // Име на клиент
   public ChatClient (string name)
   {
        this.name = name;
        // Добавя нов делегат
        ChatServer.OnMsgArrived +=
          new ChatServer.MsgArrivedEventHandler (OnMsgArrived);
        ChatServer.SendMsg ("Здравейте! Аз съм " + name);
   }
   // Манипулатор на събитието ПРИСТИГАНЕ НА СЪОБЩЕНИЕ
   private void OnMsgArrived (object source, MsgArrivedEventArgs e)
   {
        Console.WriteLine ("Пристигнало съобщение "+
                           " (Клиент {0}): {1}", name, e.Msg);
   }
```

```
public void Dispose ()
{
     // Изтрива делегат
     ChatServer.OnMsgArrived -=
       new ChatServer.MsgArrivedEventHandler (OnMsgArrived);
     // Не позволява на системата да извика финализиращия метод за
     // клиента, за да предотварти изчистващият код за обекта да се
     // извика два пъти
     GC.SuppressFinalize (this);
}
~ChatClient ()
{
     Dispose();
}
}
```

```
class TestChat
{  static void Main (string[] args)
   {    ChatClient c1 = new ChatClient ("Иван");
        ChatClient c2 = new ChatClient ("Георги");
        ChatClient c3 = new ChatClient ("Мария");
        // Връзката на клиентите към сървъра трябва да се прекъсне
        // изрично. В противен случай паметта за клиентите не може да
        // се използва, докато сървърът не прекъсне приложението.
        c1.Dispose ();
        c2.Dispose ();
        c3.Dispose ();
   }
}
}
```

Резултати:

```
Пристигнало съобщение (Клиент Иван): Здравейте! Аз съм Иван
Пристигнало съобщение (Клиент Иван): Здравейте! Аз съм Георги
Пристигнало съобщение (Клиент Георги): Здравейте! Аз съм Георги
Пристигнало съобщение (Клиент Иван): Здравейте! Аз съм Мария
Пристигнало съобщение (Клиент Георги): Здравейте! Аз съм Мария
Пристигнало съобщение (Клиент Мария): Здравейте! Аз съм Мария
```