Делегати

Делегат

- **референтен тип, абстракция на метод;**
- **разглежда се като елегантен метод;**
- **еквивалент на обект на функция;**
- **производен клас на типа** System.MulticastDelegate**;**
- **използва се за т.н. callback функционалност – методи с параметър указател към функция, която се извиква чрез този указател:**
  - **а) асинхронна обработка на събития;**
  - **б) инжектиране на потребителски код в йерархията на класовете.**

1. **Дефиниране на делегат – стандартна конвенция е името му да завършва със** Callback**.**
   [<атрибут>] [<модификатор_за_достъп>] delegate <върнат_тип> <име_на_делегат> ([параметри]);
2. **Дефиниране на callback метод с параметър-делегат.**
3. **Дефиниране на потребителски метод със същите параметри като тези на делегата.**
4. **Създаване на екземпляр на делегата чрез** new **и предаване името на потребителския метод .**

Пример: Делегат като указател към функция

Вариант 1

Изчисляване на сума от реципрочни стойности или от стойности, повдигнати на квадрат – чрез използване на флаг.

```
using System;
namespace SumWithoutDelegate
{
  public enum Status {Reciprocal, Square};
  class Test
  {
    static double Reciprocal (int k)
    {
       return 1.0 / k;
    }
    static double Square (int k)
    {
       return (double)k * k;
    }
```

```
    static double Sum (int n, Status status)
    {   double s = 0.0;
        for (int i = 1; i <= n; i++)
          if (status == Status.Reciprocal)
            s += Reciprocal (i);
          else if (status == Status.Square)
            s += Square (i);
        return s;
    }
    static void Main (string[] args)
    {
        Console.WriteLine ("1+1/2+1/3 = {0}",
                           Sum (3, Status.Reciprocal));
        Console.WriteLine ("1**2+2**2+3**2+4**2+5**2 = {0}",
                            Sum (5, Status.Square));
    }
  }
}
```

Пример: Делегат като указател към функция

Вариант 2

Изчисляване на сума от реципрочни стойности или от стойности, повдигнати на квадрат – чрез използване на делегат.

```
using System;
namespace SumWithDelegate
{
  public delegate double SumCallback (int k);
  class Test
  {
    static double Reciprocal (int k)
    {
        return 1.0 / k;
    }
    static double Square (int k)
    {
        return (double)k * k;
    }
```

```
    static double Sum (int n, SumCallback callback)
    {
        double s = 0.0;
        for (int i = 1; i <= n; i++)
          s += callback (i);
        return s;
    }
    static void Main (string[] args)
    {
        SumCallback c1 = new SumCallback (Reciprocal);
        Console.WriteLine ("1+1/2+1/3 = {0}", Sum (3, c1));
        SumCallback c2 = new SumCallback (Square);
        Console.WriteLine ("1**2+2**2+3**2+4**2+5**2 = {0}",
                           Sum (5, c2));
    }
  }
}
```

Пример: Делегат като указател към статичен и нестатичен метод

Дефинира делегат с параметър СТУДЕНТ и метод с параметър този делегат.

Потребителят дефинира два метода с параметър СТУДЕНТ: статичен метод за отпечатване името на студент и нестатичен метод, който изчислява общия брой студенти в групата и общия успех.

```
using System;
using System.Collections;
// Описва студент
public class Student
{  protected string name;                    // Име
   public string Name
   {  get  {  return name;  }
   }
   protected float grade;                    // Успех
   public float Grade
   {  get  {  return grade;  }
   }
   public Student (string name, float grade)
   {  this.name = name;
      this.grade = grade;
   }
}
```

```
// Дефиниране на тип делегат за обработка на студент
public delegate void ProcessStudentCallback(Student student);
// Обработка на студентска група
class StudentGroup
{  // Списък на всички студенти в групата
   protected ArrayList group = new ArrayList();
   // Добавяне на студент в групата
   public void AddStudent (string name, float grade)
   {
      group.Add (new Student (name, grade));
   }
   // Дефиниране на метод за обработка на студент от
   // групата с параметър делегат
   public void ProcessStudent (ProcessStudentCallback
                                        processStudent)
   {  foreach (Student s in group)
         processStudent (s);              // Извиква делегата
   }
}
```

```
// Клас за обща и средна оценка на студентска група
class GradeTotaller
{  int countStudents = 0;
   float gradeStudents = 0.0f;
   // Дефиниране на потребителски метод със същите
   // параметри, като тези на делегата – изчислява общия
   // брой студенти в групата и общия успех
   public void AddStudentToTotal (Student student)
   {  countStudents++;
      gradeStudents += student.Grade;
   }
   // Среден успех на студентската група
   public float AverageGrade()
   {
        return gradeStudents / countStudents;
   }
}
```

```
// Клас за тестване на студентската група
class Test
{
   // Дефиниране на статичен потребителски метод със
   // същите параметри като тези на делегата – отпечатва
   // име на студент
   static void PrintName (Student s)
   {
        Console.WriteLine("   {0}", s.Name);
   }
   // Инициализира студентската група с няколко студенти
   static void AddStudents (StudentGroup studentGroup)
   {
     studentGroup.AddStudent("Иван Петров Георгиев",5.60f);
     studentGroup.AddStudent("Анна Иванова Стоянова",5.20f);
     studentGroup.AddStudent("Георги Миланов Петров",5.40f);
   }
```

```
   // Начало на изпълнението
   static void Main()
   {
      StudentGroup studentGroup = new StudentGroup();

      // Инициализиране на групата със студенти
      AddStudents (studentGroup);

      // Отпечатване имената на студентите
      Console.WriteLine ("Студенти:");
```

```
        // Създава нов обект на делегата, свързан със
        // статичния метод Test.PrintName
        studentGroup.ProcessStudent
                        (new ProcessStudentCallback (PrintName));
        // Изчислява средния успех на студентите чрез
        // използване на обект GradeTotaller
        GradeTotaller totaller = new GradeTotaller();
        // Създава нов обект на делегата, свързан с
        // нестатичния метод AddStudentToTotal на обекта
        // totaller
        studentGroup.ProcessStudent (new
            ProcessStudentCallback (totaller.AddStudentToTotal));
        Console.WriteLine("Среден успех на групата: {0:#.#0}",
                        totaller.AverageGrade());
    }
}
```

Резултати:

```
Студенти:
  Иван Петров Георгиев
  Анна Иванова Стоянова
  Георги Миланов Петров
Среден успех на групата: 5,40
```

Комбиниран делегат (multicast)

Комбинира много делегати в един – динамично решава кои методи да съответстват на callback метода.

а) агрегиране на делегати в един делегат – чрез оператор плюс (+);

б) премахване – чрез оператор минус (-).

Пример: Дефинира комбиниран делегат и по време на изпълнение се решава дали да се извика методът за изчисляване на реципрочна стойност или методът за повдигане на квадрат.

```
using System;
// Комбиниран делегат
delegate double MultipleCallback (int k);
class MyClass
{
   // Реципрочна стойност
   public static double Reciprocal (int k)
   {
      Console.WriteLine (1.0 / k);
      return 1.0 / k;
   }
   // Повдигане на квадрат
   public static double Square (int k)
   {
      Console.WriteLine ((double)k * k);
      return (double)k * k;
   }
```

```
public static void Main()
{
   MultipleCallback a, b, c, d;
   // Създава обекта a на делегат, който се обръща към
   // метода Reciprocal
   a = new MultipleCallback (Reciprocal);
   // Създава обекта b на делегат, който се обръща към
   // метода Square
   b = new MultipleCallback (Square);
   // Двата делегата a и b образуват комбинирания
   // делегат c,  който извиква последователно двата
   // метода
   c = a + b;
  // Премахва a от комбинирания делегат и остава b,
  // който извиква само метода Square
  d = c - a;
```

```
   double aa = a(2);                  // 1./2=0.5
   double bb = b(3);                  // 3*3=9
   double cc = c(4);                  // 1./4=0.25
                                      // 4*4=16
   double dd = d(5);                  // 5*5=25
  }
}
```

Резултати:

```
0,5
9
0,25
16
25
```

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.

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);
   }
}
```

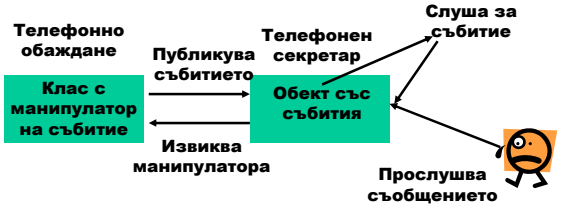
```
class Subscriber                                 // Клас-приемник
{
   private Publisher publisher;
   public Subscriber(Publisher publisher)
   {
      this.publisher = publisher;
      // Добавя нов делегат
      publisher.OnChange +=
         new Publisher.InventoryChangeEvenHandler (OnHand);
   }
   // 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;
      // Доабвя 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
        // предотвърти 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 ();
    }
  }
}
```

```
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
```

Пример: Събитие АЛАРМА на будилник.

```
using System;
// Клас с данни за събитието АЛАРМА.
// Наследник е на класа System.EventArgs.
public class AlarmEventArgs : EventArgs
{
  private int nrings;
  // Свойството NumRings връща броя на позвъняванията на
  // будилника, когато се генерира събитието АЛАРМА
  public int NumRings
  {
    get
    {
      return nrings;
    }
  }
```

```
  private bool snoozePressed ;
  // Свойството SnoozePressed показва дали е натиснат
  // бутонът дрямка на алармата, когато се генерира
  // събитието АЛАРМА
  public bool SnoozePressed
  {  get {  return snoozePressed;}
  }
  // Свойството AlarmText съдържа съобщението за
  // събуждане
  public string AlarmText
  {
    get
    {  if (snoozePressed)
         return ("Ставай!!! Времето за дрямка мина.");
       else
         return ("Ставай!");
    }
  }
```

```
  public AlarmEventArgs (bool snoozePressed, int nrings)
  {
    this.snoozePressed = snoozePressed;
    this.nrings = nrings;
  }
}
```

```
// Клас Будилник – вдига събитието АЛАРМА
public class AlarmClock                    // Източник
{
  private bool snoozePressed = false;
  private int nrings = 0;
  private bool stop = false;
  // Свойството Stop показва дали алармата е
  // изключена
  public bool Stop
  {
    get  {  return stop;  }
    set  {  stop = value;  }
  }
```

```
  // Свойството SnoozePressed показва дали бутонът
  // дрямка е натиснат, когато се генерира събитието
  // АЛАРМА
  public bool SnoozePressed
  {
    get  {  return snoozePressed;  }
    set  {  snoozePressed = value;  }
  }
  // Деклариране на делегат
  public delegate void AlarmEventHandler (object sender,
                                          AlarmEventArgs e);
  // Дефиниране на събитието АЛАРМА от тип
  // AlarmEventHandler
  public event AlarmEventHandler Alarm;
```

```
  // Аларменият механизъм се симулира чрез цикъл, който
  // вдига събитието АЛАРМА при всяка итерация с време
  // закъснение от 300 милисекунди, ако бутонът дрямка не
  // е натиснат, и съответно 1000 милисекунди – при
  // натиснат бутон дрямка.
  public void Start()
  {
    for (; ;)
    {  nrings++;
       if (stop)
         break;
       else if (snoozePressed)
       {  System.Threading.Thread.Sleep(1000);
          AlarmEventArgs e =
                  new AlarmEventArgs (snoozePressed, nrings);
          if (Alarm != null)
            Alarm (this, e);          // Извиква делегата
       }
```

```
        else
        {
          System.Threading.Thread.Sleep(300);
          AlarmEventArgs e =
                  new AlarmEventArgs (snoozePressed, nrings);
          if (Alarm != null)
            Alarm (this, e);                    // Извиква делегата
        }
      }
    }
}
```

```
// Клас WakeMeUp с манипулатор AlarmRang на събитието
// АЛАРМА
public class WakeMeUp                                  // Приемник
{
  private AlarmClock  clock;

  public WakeMeUp (AlarmClock  clock)
  {
    this.clock = clock;
    clock.Alarm +=
              new AlarmClock.AlarmEventHandler (AlarmRang);
  }
```

```
  public void AlarmRang(object sender, AlarmEventArgs e)
  {
    Console.WriteLine(e.AlarmText + "\n");
    if (!(e.SnoozePressed))
    {
      if (e.NumRings % 10 == 0)
      {
        Console.WriteLine
                (" Да звънне ли часовникът? Въведете Y");
        Console.WriteLine
                 (" Натиснат ли е дрямка? Въведете N");
        Console.WriteLine
                (" Да спре ли алармата? Въведете Q");
        String input = Console.ReadLine();
        if (input.Equals("Y") || input.Equals("y"))
          return;
```

```
        else if (input.Equals("N") || input.Equals("n"))
        {
          ((AlarmClock)sender).SnoozePressed = true;
          return;
        }
        else
        {
          ((AlarmClock)sender).Stop = true;
          return;
        }
      }
    }
```

```
      else
      {
        Console.WriteLine
                (" Да звънне ли часовникът? Въведете Y");
        Console.WriteLine
                (" Да спре ли алармата? Въведете Q");
        String input = Console.ReadLine();
        if (input.Equals("Y") || input.Equals("y"))
          return;
        else
        {
          ((AlarmClock)sender).Stop = true;
          return;
        }
      }
    }
}
```

```
// Драйверен клас, който прикача манипулатора AlarmRang
// на класа WakeMeUp към събитието АЛАРМА на обект от
// тип Alarm, използвайки делегат.
public class AlarmDriver
{
  public static void Main ()
  {
    // Създава екземпляр на източника на събитието
    AlarmClock clock = new AlarmClock();
    // Създава екземпляр на приемника на събитието
    WakeMeUp w= new WakeMeUp (clock);
    clock.Start ();
  }
}
```

Резултати:

```
Ставай!
Ставай!
Ставай!
Ставай!
Ставай!
Ставай!
Ставай!
Ставай!
Ставай!
Ставай!
 Да звънне ли часовникът? Въведете Y
 Натиснат ли е дрямка? Въведете N
 Да спре ли алармата? Въведете Q
N
Ставай!!! Времето за дрямка мина.
 Да звънне ли часовникът? Въведете Y
 Да спре ли алармата? Въведете Q
Q
```

Асинхронно програмиране

Асинхронно програмиране – програмна техника, която позволява на програмата да дава усещането, че компютърът извършва едновременно няколко неща.

Чрез тази техника се създава отговарящ потребителски интерфейс, докато програмата изпълнява дълга операция.

Всеки метод в .NET може да се извика асинхронно чрез използване на делегати.

При дефиниране на делегат изпълнителната система автоматично дефинира методите:

Invoke – **инициира синхронна операция.**

BeginInvoke – **инициира асинхронна операция; включва следните параметри:**
– **всички входни,** out, ref **и референтни параметри;**
– **делегат от тип** AsyncCallback **за callback функция;**
– **делегата, даващ състоянието от тип** AsyncState **(чрез свойството** AsyncState **на интерфейса** IAsyncResult**);**
методът завършва веднага, като връща обект, реализиращ интерфейса IAsyncResult**.**

След завършване на асинхронната операция се изпълнява callback функцията, която извиква метода EndInvoke**, за да се получи резултатът.**

EndInvoke – **включва следните параметри:**
– out, ref **и референтни параметри;**
– IAsincResult **като последен параметър;**
връща оригиналния тип на оригиналния метод.

Използва се вграденият асинхронен делегат:

```
public delegate void AsyncCallback (IAsyncResult ar);
```

Пример: Използване на делегат за асинхронно извикване на методи

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

```
using System;
namespace SumAsyncCallback
{
  public delegate double SumCallback (int k);
  class Test
  {
    static double Reciprocal (int k)
    {
        return 1.0 / k;
    }
    static double Square (int k)
    {
        return (double)k * k;
    }
```

```
// Дефиниране на делегат
public delegate void SumAsyncCallback
            (int n, SumCallback callback, out double s);

// Дълга операция
public static void Sum
            (int n, SumCallback callback, out double s)
{
  s =0.0;
  for (int i = 1; i <= n; i++)
    s += callback (i);
}
```

```
// Callback метод – извиква се, когато асинхронната
// операция завърши
public static void DoneCallback (IAsyncResult iar)
{
    SumAsyncCallback add =
                    (SumAsyncCallback)iar.AsyncState;
    double sum;
    add.EndInvoke (out sum, iar);
    Console.WriteLine ("Сума = {0}", sum);
}
```

Дава последния параметър от извикването на BeginInvoke

```
static void Main (string[] args)
{
    SumCallback c1 = new SumCallback (Reciprocal);
    SumCallback c2 = new SumCallback (Square);
    SumAsyncCallback add = new SumAsyncCallback (Sum);
    double sum;
    add.BeginInvoke (3, c1, out sum,
                     new AsyncCallback (DoneCallback), add);
    for (int i=1; i<= 15; i++)
      Console.WriteLine(".");

    add.BeginInvoke (5, c2, out sum,
                     new AsyncCallback (DoneCallback), add);
    for (int i=1; i<= 15; i++)
      Console.WriteLine(".");
  }
 }
}
```

Наименовани пространства

Наименовани пространства – групиране на класове и типове (пакети, библиотеки, приложни програмни интерфейси API).

Предимство: многократно използване на класовете.

1. Дефиниране

```
namespace <име_на_наименовано_пространство>
{
    <тяло_на_наименованото_пространство>
}
```

2. Използване – чрез директивата using

```
using <име_на_наименовано_пространство>
```

или

```
using <псевдоним> = <име_на_клас>
```

Изграждане и изпълнение на .NET приложения

Предимства на .NET:

1. **Еднаква функционалност на всички .NET езици;**
2. **Комбиниране на код, написан на различни езици.**

Особености:

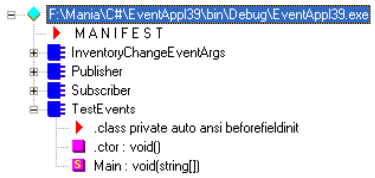
Създаване на Междинен код MSIL (Microsoft Intermediate Language) от компилатора.

Компилиране на MSIL кода от JIT (just-in-time) компилатора – преобразува междинния код в инструкции за процесора.

Деасемблиране на изпълним файл – приложение ILDASM

Start ⇒ Programs ⇒
Microsoft Visual Studio .NET 2003 ⇒
Visual Studio .NET Tools ⇒
Visual Studio .NET Command prompt ⇒
команден прозорец : ildasm

прозорец на ILDASM :
File ⇒ Open ⇒
драйв:\...\директория_на_приложение\bin\Debug\приложение.exe



Асемблии – фундаментална част в .NET;
създава се от компилатора;
съдържа код, изпълняващ се от системата.

– колекция от модули, експортирани типове и ресурси, която има име и версия (от гледна точка на клиента);

– начин за пакетиране на модули, типове и ресурси, използвани от клиента (от гледна точка на създателя).

Манифест – съдържа информация за мета данните, описваща асемблито.

– запис .assembly за обръщение към външно асемблии;

– запис .assembly с информация за асемблито;

– запис .module, съдържащ името на физическия файл и друга външна информация.

Предимства на асемблито:

1. **Пакетира много модули в един физически файл; зарежда само необходимите модули при многофайлово асемблии – редуцира работната среда за приложението.**
2. **Развива класа – едно приложение може да се изгради от много асемблита.**
3. **Асемблито има версия, съхраняваща се в манифеста – приложението ще функционира независимо от новите версии на споделените DLL.**

Изграждане на асемблии
– ключ /t:exe при изграждане на EXE;
– ключ /t:library при изграждане на DLL;
– ключ /r за свързване на DLL.

Изграждане на модул
– ключ /t:module;
– ключ /addmodule или инструмент Assembly Generation – добавяне на модул към асемблии.

C# компилатор

csc.exe

Установяване на пътя до изпълнимия файл csc.exe

VCVARS32.bat

(напр. C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\vcvars32.bat**)**

Пример: Създаване на асемблии чрез ключа /t:library – компилаторът създава библиотека (DLL), която се свързва чрез ключа /r.

```
// DllTestServer.cs
// Изгражда се чрез csc /t:library DllTestServer.cs
// Създава се асемблито DllTestServer.dll
public class DllTestServer
{
  public static void Foo()
  {
    System.Console.WriteLine
                    ("DllTestServer.Foo (DllTestServer.DLL)");
  }
}
```

```
// DllTestClient.cs
// Изгражда се чрез csc DllTestClint.cs /r:DllTestServer.dll
// Създава се асемблито DllTestClient.exe
using System;
using System.Diagnostics;
using System.Reflection;
class DllTestClientApp
{
  public static void Main()
  {  Assembly DllAssembly =
                  Assembly.GetAssembly(typeof(DllTestServer));
     Console.WriteLine
                  ("\nDllTestServer.dll Информация за асемблии");
     Console.WriteLine("\t"+DllAssembly);
     Process p=Process.GetCurrentProcess();
     string AssemblyName=p.ProcessName+".exe";
     Assembly ThisAssembly =
                        Assembly.LoadFrom(AssemblyName);
```

```
      Console.WriteLine
                ("\nDllTestClient.exe Информация за асембли ");
      Console.WriteLine("\t"+ThisAssembly+"\n");
      Console.WriteLine("Извикване на DllTestServer.Foo...");
      DllTestServer.Foo();
   }
}
```

Изпълнение:
```
DllTestServer.dll Информация за асембли
    DllTestServer, Version=0.0.0.0, Culture=neutral,
    PublicKeyToken=null
DllTestClient.exe Информация за асембли
    DllTestClient, Version=0.0.0.0, Culture=neutral,
    PublicKeyToken=null
Извикване на DllTestServer.Foo...
DllTestServer.Foo (DllTestServer.DLL)
```

Клас System.Reflection.Assembly – **представя асемблы.**

Метод GetAssembly – **дава асемблито, в което даденият клас е дефиниран.**

Метод LoadFrom – **зарежда асемблы с дадено име.**

Клас System.Diagnostics.Process – **осъществява достъп до локални и отдалечени процеси.**

Метод GetCurrentProcess – **дава нов процес и го свързва с текущия активен процес.**

Свойство ProcessName – **връща името на процеса.**

Пример: Създаване на асемблы с много модули. Клас с модификатор за достъп public **е достъпен от всеки код (в предния пример); с модификатор** internal **е достъпен само от код в същото асемблы.**

```
// NetModuleTestServer.cs
// Изгражда се чрез csc /t:module NetModuleTestServer.cs
// Създава се модулът NetModuleTestServer.netmodule
internal class NetModuleTestServer
{
   public static void Bar()
   {
      System.Console.WriteLine("NetModuleTestServer.Bar" +
                  " (NetModuleTestServer.netmodule)");
   }
}
```

```
// NetModuleTestClient.cs
// Модулът се добавя към асемблито чрез командния ред
// csc /addmodule:NetModuleTestServer.netmodule
//       NetModuleTestClient.cs
// Създава се асемблито NetModuleTestClient.exe
using System;
using System.Diagnostics;
using System.Reflection;
class NetModuleTestClientApp
{
  public static void Main()
  {  Assembly DllAssembly =
        Assembly.GetAssembly(typeof(NetModuleTestServer));
     Console.WriteLine("\nNetModuleTestServer.netmodule " +
                                        "Информация за модул");
     Console.WriteLine("\t"+DllAssembly);
     Process p=Process.GetCurrentProcess();
     string AssemblyName=p.ProcessName+".exe";
```

```
     Assembly ThisAssembly =
                        Assembly.LoadFrom(AssemblyName);
     Console.WriteLine
       ("\nNetModuleTestClient.exe Информация за асемблы");
     Console.WriteLine("\t"+ThisAssembly+"\n");
     Console.WriteLine
                ("Извикване на NetModuleTestServer.Bar...");
     NetModuleTestServer.Bar();
   }
}
```

Изпълнение:
```
NetModuleTestServer.netmodule Информация за модул
        NetModuleTestServer, Version=0.0.0.0, Culture=neutral,
PublicKeyToken=null
NetModuleTestClient.exe Информация за асемблы
        NetModuleTestClient, Version=0.0.0.0, Culture=neutral,
PublicKeyToken=null
Извикване на NetModuleTestClient.Bar...
NetModuleTestClient.Bar (NetModuleTestServer.netmodule)
```

Споделено асемблы – използва се от много приложения.

а) създаване на силно име (чрез средството Strong Name в .NET SDK) – създава се изходен файл, който съдържа ключа;

sn –k име_на_файл.key

Получава се съобщението:

Key pair written to има_на_файл.key

б) добавяне на атрибута AssemblyKeyFile **към потребителския сорс файл.**

Пример: Споделено асембли

```
// SharedAssemblyServer.cs
// Изгражда се чрез csc /t:module SharedAssemblyServer.cs
// Създава се модулът SharedAssemblyServer.netmodule
internal class SharedAssemblyServer
{
  public static void Test()
  {
    System.Console.WriteLine("SharedAssemblyServer.Test" +
                             "(SharedAssemblyServer.netmodule)");
  }
}
```

```
// SharedAssemblyClient.cs
// Модулът се добавя към асемблито чрез командния ред
// csc /addmodule:SharedAssemblyServer.netmodule
//          SharedAssemblyClient.cs
// Създава се асемблито SharedAssemblyClient.exe
using System;
using System.Diagnostics;
using System.Reflection;
[assembly:AssemblyKeyFile("MyFile.key")]
class SharedAssemblyClientApp
{
  public static void Main()
  {
    Assembly DllAssembly =
        Assembly.GetAssembly(typeof(SharedAssemblyServer));
    Console.WriteLine("\nSharedAssemblyServer.netmodule " +
                      "Информация за модул");
   Console.WriteLine("\t"+DllAssembly);
```

```
    Process p=Process.GetCurrentProcess();
    string AssemblyName=p.ProcessName+".exe";
    Assembly ThisAssembly =
                        Assembly.LoadFrom(AssemblyName);
    Console.WriteLine
      ("\nSharedAssemblyClient.exe Информация за асемблi");
    Console.WriteLine("\t"+ThisAssembly+"\n");
    Console.WriteLine
              ("Извикване на SharedAssemblyServer.Test...");
    SharedAssemblyServer.Test();
  }
}
```

Изпълнение: NetModuleTestClient

```
SharedAssemblyServer.netmodule Информация за модул
        SharedAssemblyServer, Version=0.0.0.0, Culture=neutral,
PublicKeyToken=6355fb151d0bfbff
SharedAssemblyClient.exe Информация за асемблi
        SharedAssemblyClient, Version=0.0.0.0, Culture=neutral,
PublicKeyToken=6355fb151d0bfbff
Извикване на SharedAssemblyServer.Test...
SharedAssemblyServer.Test
(SharedAssemblyServer.netmodule)
```