Основы программирования

Делегаты, события и лямбда выражения

Делегаты

Определение

Делегат -- объект, который может ссылаться на метод.

Когда создается делегат, то в итоге получается объект, содержащий ссылку на метод.

По этой ссылке можно вызывать метод.

Иными словами, делегат позволяет вызывать метод, на который он ссылается.

Делегат - это специальный класс, объект которого предназначен для хранения ссылок на методы.

Синтаксис делегата описывает сигнатуру методов, которые могут быть вызваны с его помощью.

Определение

Это объект, указывающий на другой метод (или, возможно, список методов) приложения, который может быть вызван позднее.

В частности, объект делегата поддерживает три важных фрагмента информации:

- адрес метода, которой вызывается;
- аргументы (если есть) этого метода;
- возвращаемое значение (если есть) этого метода.

Синтаксис

```
<атрибуты> <спецификаторы> delegate <тип>
<имя>(<список параметров>);
```

где,

спецификаторы могут быть следующими - `new`, `public`, `protected`, `internal`, `private`;

тип – это тип возвращаемого значения из методов, вызываемых с помощью делегата;

список параметров – набор параметров этих методов с указанием их типов.

Замечания 1

Делегат может хранить ссылки на несколько методов, их списки параметров должны совпадать по порядку и типу параметров, методы вызываются поочерёдно.

Наследовать от делегата запрещено, объявлять делегат можно как в классе, так и непосредственно в пространстве имен (вне класса).

Для использования делегата нужно создать его экземпляр и зарегистрировать в нём методы. При вызове экземпляра делегата вызываются все методы, ссылки на которые он хранит.

Замечания 2

Делегаты позволяют определить исполняемый метод во время выполнения программы, а не на этапе компиляции, а это, в свою очередь, позволяет создавать методы, вызывающие другие методы.

Делегат предназначен для реализации механизма событий.

Пример

Сначала описываем делегат:

```
delegate void Message();
                    // 1. Описание/определение делегата
```

Для использования делегата объявляется переменная этого делегата:

```
Message mes;    // 2. Объявление делегата/
                       создание переменной делегата

mes = GoodMorning;
                    // 3. Инициализация делегата/
                    присвоение переменной адреса метода
```

Через делегат вызываем метод, на который ссылается данный делегат:

```
mes();              // 4. Вызываем метод
```

Пример

```
delegate int Del(int a, int b);
                          // Определение делегата
class Class
{
      public int Sum(int a, int b)
      {
      return a+b;
      }

      public int Sub(int  a, int b)
      {
      return a-b;
      }
}
```

Пример

```
class Program {
      static void Main()
      {
      int a=5;
      int b=2;
      Del d;                          // Объявление делегата
      Class c=new Class();
      d = new Del(c.Sum);             //Инициализация
      int e =d(a,b);                  // Выполнение делегата
      d=new Del(c.Sub);
      int f=d(a,b);
      }
}
```

Вызов делегата

Вызов делегата - тот же вызов метода.

Если в делегате хранятся ссылки на несколько методов, то они выполняются последовательно, и изменения, вносимые в параметры одним методом, влияют на последующие.

Формирование списка методов, вызываемых с помощью делегата, производится с помощью операции сложения.

Удаление из списка – с помощью операции вычитания

Либо методов `Combine()` и `Remove()` у объекта делегат.

Пример

```
class Class {

      public static int Sum(int a, int b) {
      Console.WriteLine("Сумма равна {0}",a+b);
      return a+b; }

      public static int Sub(int a, int b) {
      Console.WriteLine("Разность равна {0}",a-b);
      return a-b; }
}

delegate int Del(int a, int b);
```

Пример

```
class Program
{
      static void Main()    {
      int a = 10;      int b = 6;

      Del d = new Del(Class.Sum);
      d += new Del(Class.Sub);

      int c = d(a,b);
      Console.WriteLine("Результат {0}",c);

      d -= new Del(Class.Sub);
      int e = d(a,b);
      Console.WriteLine("Результат {0}",e);
      }
}
```

Передаваемые методы

В делегат можно передавать методы:

- обычные (по имени объекта);
- статические (по имени класса).

Сигнатура методов должна полностью соответствовать делегату.

Передаваемые параметры

Параметры могут передаваться не по значению, а по ссылке.

В этом случае изменение параметра в одном методе из списка методов влияет на параметр, передаваемый в последующие методы.

Поскольку методы в списке вызова делегата выполняются последовательно, набор параметров в методы передаётся также последовательно.

Исключения при вызове делегатов

Если вызвать на исполнение делегат, в списке которого нет методов, то будет сгенерировано исключение `NullReferenceException`.

Исключения при вызове делегатов

Если в методе из списка делегата возникло исключение, то

1. исключение, обработанное в том же методе, никак не влияет на выполнение последующих методов;
2. если исключение в методе не обработано, то последующие методы не выполняются, а производится поиск обработчика исключения в методе, вызывающем делегат.

Пример

```
class Program   {
      static void Main()    {
      Del d=new Del(Class1.Method1);
      d+=new Del(Class1.Method2);
      d+=new Del(Class1.Method3);
      d();
      }
}
```

Пример

Выполнение данного примера приведет к выбросу исключения в одном из методов в списке вызова делегата. Необходима обработка исключения.

Пример

```
class Program   {
static void Main()          {
Del d=new Del(Class1.Method1);
d+=new Del(Class1.Method2);
d+=new Del(Class1.Method3);
try   {
d();  }
catch (Exception e)         {
Console.WriteLine(e.Message);    }
      }
}
```

Пример

```
// или так:

foreach (Del d1 in d.GetInvokationList())   {
try   {
d1(); }
catch (Exception e)                         {
Console.WriteLine(e.Message);}              }
}
}
```

Делегаты в параметрах методов

Делегат – это ссылка на методы, и объект специального класса, его можно передавать в методы.

Таким образом, с использованием делегата создаются универсальные методы, которые вызывают другие методы, причём заранее неизвестно, какие, они передаются в виде делегата.

Пример

```
delegate double Function(double x);

class Class1
{
  public static void CountFunction(Function f,
  double a, double b, double dx)
            {
  for (double x=a; x<=b; x+=dx)
  {
     Console.WriteLine("x={0,5:0.##}
     y={1,5:0.##}",x,f(x));
  }

}
```

Пример

```
public static double Sinus(double x)   {
return Math.Sin(x);
}

public static double Cosinus(double x){
return Math.Cos(x);
}

public static double Const(double x)   {
return 3;
}
}
```

Пример

```
class Program            {
  static void Main()     {

  Console.WriteLine("Функция синуса");
  Function d=new Function(Class1.Sinus);
  Class1.CountFunction(d,-2,2,0.1);

  Console.WriteLine("Функция косинуса");
  d=new Function(Class1.Cosinus);
  Class1.CountFunction(d,-2,2,0.05);

  Console.WriteLine("Функция y=3");
  Class1.CountFunction(new
  Function(class1.Const),0,5,1);
  }
}
```

Пример. Комментарий

На данной технологии основан механизм обратных вызовов: описывается метод, содержащий бизнес-логику, этот метод через делегат передаётся в универсальный метод, а он вызывает делегат, то есть передаёт вызов методу с бизнес-логикой (обратный вызов).

Операции с делегатами

Делегаты можно сравнивать на равенство и неравенство.

Делегаты считаются равными, если они содержат ссылки на одни и те же методы в одном и том же порядке, либо не содержат методы.

Делегаты, имеющие один тип возвращаемого значения и одинаковые списки параметров, но различающиеся по имени класса, также можно сравнивать, на равенство и неравенство, хотя такие делегаты считаются принадлежащими разным типам.

Операции с делегатами

Делегат относится к неизменяемым типам данных, то есть при его изменении создаётся новый экземпляр, а старый теряется.

Делегаты одного типа можно складывать и вычитать.

Пример

```
…
Del d1=new Del(Class1.Sum);
Del d2=new Del(Class1.Sub);
Del d3=new Del(Class1.Sub);
d3=d1+d2;
d3+=d1;
d3+=d2;
d3-=d1;
…
```