**Лекция 17**

**Делегаты, события и лямбды**

**Часть 1. Делегаты**

**Последнее обновление: 13.10.2018**

Делегаты представляют такие объекты, которые **указывают** на методы.

То есть делегаты - это указатели на методы. С помощью делегатов мы можем вызвать эти методы.

* 1. **Определение делегатов**

**Делегат можно определить (объявить) внутри класса или вне класса - внутри пространства имен.**

Для объявления делегата используется ключевое слово **delegate**, после которого идет **возвращаемый** **тип**, **название и параметры**.

Например,

|  |  |
| --- | --- |
|  | **delegate void** Message(); |

Делегат Message в качестве возвращаемого типа имеет тип **void** (то есть он ничего не возвращает) и не принимает никаких параметров. Это значит, что такой делегат может указывать на любой метод, который не принимает никаких параметров и ничего не возвращает.

Рассмотрим применение такого делегата:

|  |  |
| --- | --- |
|  | **class** Program  {    //-----------Создаем два метода класса Program -----------------------  **private static void** GoodMorning()       {Console.WriteLine("Good Morning");}  **private static void** GoodEvening()       {Console.WriteLine("Good Evening");}  //--------------------------------------------------------------------  **delegate void** **Message**(); // 1. Объявляем делегат именем Message  **static void** Main(string[] args)      {  **Message** mes; // 2. Создаем переменную делегата  // То есть переменную ссылочного типа          if (DateTime.Now.Hour < 12)          {              mes = GoodMorning; // 3. Этой переменной присваиваем **адрес**  **метода GoodMorning класса Program**          }          else          {              mes = **GoodEvening**; // Метод **GoodEvening**;          }          mes(); // 4. Теперь вызываем метод          Console.ReadKey();      }// Конец Main  } |

Прокомментируем пример.

Сначала мы объявили делегат:

|  |  |
| --- | --- |
|  | **delegate void** Message(); // 1. Объявляем делегат |

Для использования делегата **объявляем переменную этого делегата**:

|  |  |
| --- | --- |
|  | Message mes; // 2. Создаем переменную делегата |

С помощью **свойства** DateTime.Now.**Hour** получаем текущий час.

Далее в зависимости от времени **в делегат передается адрес определенного метода**.

Важно то, что эти методы имеют то же возвращаемое значение и тот же набор параметров (в данном случае отсутствие параметров), что и делегат.

|  |  |
| --- | --- |
|  | mes = GoodMorning; // 3. Присваиваем этой переменной адрес метода |

Затем через делегат **вызываем метод, на который ссылается данный делегат**:

|  |  |
| --- | --- |
|  | mes(); // 4. Вызываем метод |

Вызов делегата производится подобно обычному вызову метода.

Рассмотрим другой пример.

|  |  |
| --- | --- |
|  | **class** Program  {  //---- Создаем два метода в классе Program ---------------------  //---- Обратить внимание, что методы с двумя параметрами и возвращают тип **int**  //---- таким же будет и делегат ---------------------------------------------  **private** **static** **int** Add(**int** x, **int** y)      {  **return** x+y;      }  **private static int** Multiply (**int** x, **int** y)      {  **return** x \* y;      }  //В этом же классе создаем делегат с двумя параметрами, который будет возвращать **int**  **delegate** **int** **Op**(**int** x, **int** y);    **static** **void** Main(**string**[] args)      {          // присваивание адреса метода **через конструктор**  **Op D = Add; // сейчас делегат указывает на метод Add**  **int** result = **D**(4,5); // задаем фактические параметры Add(4, 5)          Console.WriteLine(result);    **D = Multiply; // теперь делегат указывает на метод Multiply**          result = **D**(4, 5); // задаем фактические параметры Multiply(4, 5)          Console.WriteLine(result);            Console.Read();      }   }// Конец класса Program |

В данном случае делегат Op возвращает значение типа int и имеет два параметра типа int. Поэтому этому делегату соответствует любой метод, который возвращает значение типа int и принимает два параметра типа int.

Методы Add и Multiply именно такие. Значит мы можем присвоить переменной делегата любой из этих методов и вызывать.

Поскольку делегат принимает два параметра типа int, то при его вызове необходимо передать значения для этих параметров: D(4,5).

Делегаты необязательно могут указывать только на методы, которые определены в том же классе, где определена переменная делегата. Они могут указывать **на методы из других классов и структур.**

|  |  |
| --- | --- |
|  | **class** Math  {//----- Создаем свой класс -------------------------      public int Sum(int x, int y) { return x + y; }  }  //---------------------------------------------------  **class** Program  {  **delegate int Op (int x, int y);**        static void Main(string[] args)      {          Math math = **new** Math(); // Экземпляр класса Math  **Op** D = math.Sum;          int result = D(4, 5);     // math.Sum(4, 5)          Console.WriteLine(result);  // 9            Console.Read();      }  } |

**Присвоение ссылки на метод**

Выше переменной делегата напрямую присваивался метод.

Есть еще один способ создания делегата - через объект делегата, в конструктор которого передается нужный метод:

|  |  |
| --- | --- |
|  | **class** Program  {  **delegate** **int** Operation(**int** x, **int** y);    **static void** Main(**string**[] args)      {          Operation deleg = Add;          Operation deleg2 = **new** Operation(Add);            Console.Read();      }  **private** **static int** Add(**int** x, **int** y) { **return** x + y; }  } |

Оба способа равноценны.

**Соответствие методов делегату**

Как было написано выше, методы соответствуют делегату, если они имеют один и тот же возвращаемый тип и один и тот же набор параметров.

Однако, в параметрах методов кроме типа могут быть указаны модификаторы - **ref** или **out**. Это приводит к тому, что такие методы не соответствуют делегату.

Например, пусть у нас есть делегат:

|  |  |
| --- | --- |
|  | delegate void SomeDel(int a, double b); |

Этому делегату соответствует, например, следующий метод:

|  |  |
| --- | --- |
|  | **void** SomeMethod1(**int** g, **double** n) { } |

А следующие методы **НЕ** соответствуют:

|  |  |
| --- | --- |
|  | **int** SomeMethod2(**int** g, **double** n) { }  **void** SomeMethod3(**double** n, **int** g) { }  **void** SomeMethod4(**ref** **int** g, **double** n) { }  **void** SomeMethod5(**out int** g, **double** n) { g = 6; } |

Здесь

* метод SomeMethod2 имеет другой возвращаемый тип, отличный от типа делегата;
* SomeMethod3 имеет другой набор параметров;
* параметры SomeMethod4 и SomeMethod5 также отличаются от параметров делегата, поскольку имеют модификаторы **ref** и **out**.

**Как в делегат добавить методы**

В примерах выше переменная делегата указывала только на один метод. В реальности **один** **делегат может указывать на множество методов**, которые имеют подходящую сигнатуру, возвращаемые тип и связаны с этим делегатом.

Все методы в делегате попадают в специальный список - список вызова или **invocation list**. И при вызове делегата все методы из этого списка последовательно вызываются.

Далее можно этот список пополнять.

|  |  |
| --- | --- |
|  | class Program  {//-----Пусть имеются два метода ----------------------------      private static void Hello()      {          Console.WriteLine("Hello");      }      private static void HowAreYou()      {          Console.WriteLine("How are you?");  //-----------------------------------------------------------------------------------------------  **delegate void** **Message**();    **static void** Main(string[] args)      {  **Message** mes1 = Hello; // инициализация при создании переменной типа  //**delegate**          mes1 += HowAreYou;  // пополняем список методов делегата  // При этом создается новый делегат уже с 2-мя  // методами          mes1(); // вызываются **оба метода** - Hello и HowAreYou          Console.Read();      }      }  } |

Для *добавления делегатов* применяется операция **+=**.

В список вызова нового делегата (с тем же именем) mes1 добавлены два метода - Hello и HowAreYou. И при вызове mes1 **вызываются сразу оба этих метода**.

То есть **в реальности происходит создание нового объекта делегата**, который получает методы старой копии делегата и новый метод. Создается новый объект делегата, который присваивается переменной mes1 (ссылочный тип переменной). Такой процесс называется «добавление делегатов».

Мы можем добавить ссылку на один и тот же метод несколько раз, и в списке вызова делегата тогда будет несколько ссылок на один и то же метод. Соответственно при вызове делегата добавленный метод будет вызываться столько раз, сколько он был добавлен:

|  |  |
| --- | --- |
|  | Message mes1 = Hello;  mes1 += HowAreYou;  mes1 += Hello;  mes1 += Hello;    mes1(); // переменную |

Консольный вывод:

Hello

How are you?

Hello

Hello

Подобным образом **можно удалять методы из делегата** с помощью операции **-=**:

|  |  |
| --- | --- |
|  | static void Main(string[] args)  {      Message mes1 = Hello; // Создали переменную делегата с инициализацией      mes1 += HowAreYou; // Уже содержит 2 метода      Message mes2 = HowAreYou; // Создали вторую переменную делегата      Message mes3 = mes1 + mes2; // Два списка вызовов делегатов складываются      mes1(); // вызываются все методы из mes1 и mes2      mes1 -= HowAreYou;  // удаляем метод HowAreYou      mes1(); // вызывается только метод Hello        Console.Read();  } |

При удалении методов из делегата так же будет создаватья новый делегат, который в списке вызова методов будет содержать на один метод меньше.

При удалении следует учитывать, что если делегат содержит несколько ссылок на один и тот же метод, то операция -= начинает поиск с конца списка вызова делегата и удаляет только одно (первое найденное) вхождение. Если подобного метода в списке вызова делегата нет, то операция -= не имеет никакого эффекта.

**Объединение делегатов**

Делегаты можно объединять в другие делегаты. Например,

|  |  |
| --- | --- |
|  | class Program  { //---пусть есть два метода-----------------------------------  **private static void** Hello()      {Console.WriteLine("Hello");}  **private static void** HowAreYou()      {Console.WriteLine("How are you?");}  //-------Описываем делегат----------------------------------  **delegate void** Message();        static void Main(string[] args)      {          Message mes1 = Hello;          Message mes2 = HowAreYou;          Message mes3 = mes1 + mes2; // объединяем **однотипные** делегаты          mes3(); // вызываются все методы из mes1 и mes2            Console.Read();      }       }  } |

В данном случае объект mes3 представляет объединение делегатов mes1 и mes2. Объединение делегатов значит, что в список вызова делегата mes3 попадут все методы из делегатов mes1 и mes2. И при вызове делегата mes3 все эти методы одновременно будут вызваны.

**Вызов делегата**

В примерах выше делегат вызывался как обычный метод. Если делегат принимал параметры, то при его вызове параметрам передавались необходимые значения:

|  |  |
| --- | --- |
|  | class Program  {      delegate int Operation(int x, int y);      delegate void Message();        static void Main(string[] args)      {          Message mes = Hello;          mes();          Operation op = Add;          op(3, 4);          Console.Read();      }      private static void Hello() { Console.WriteLine("Hello"); }      private static int Add(int x, int y) { return x + y; }  } |

Другой способ вызова делегата – с помощью **метода** **Invoke()**:

|  |  |
| --- | --- |
|  | class Program  { private static void Hello() { Console.WriteLine("Hello"); }      private static int Add(int x, int y) { return x + y; }  //----- Пусть определены два делегата ---------------------------  **delegate int** **Operation**(int x, int y);  **delegate void** **Message**();   //----------------------------------------------------------------------------------      static void Main(string[] args)      {          Message mes = Hello; //Создали переменную делегата Message  **mes.Invoke()**; // Вызвали делегата          Operation op = Add; //Создали переменную делегата Operation  **op.Invoke(3, 4)**; // Вызвали делегата          Console.Read();      }      } |

Если делегат принимает параметры, то в **метод** **Invoke** передаются значения для этих параметров.

Следует учитывать, что если делегат пуст, то есть в его списке вызова нет ссылок ни на один из методов (то есть делегат равен Null), то при вызове такого делегата мы получим исключение, как, например, в следующем случае:

|  |  |
| --- | --- |
|  | Message mes = null;  //mes();        // ! Ошибка: делегат равен null    Operation op = Add;  op -= Add;      // **делегат op пуст**  op(3, 4);       // Вызываем делегат !Ошибка: делегат равен null |

Поэтому **при вызове делегата всегда лучше проверять, не равен ли он значению null.**

Либо можно использовать метод **Invoke** и оператор условного **null**:

|  |  |
| --- | --- |
|  | Message mes = null;  mes?.Invoke();        // ошибки нет, делегат просто не вызывается    Operation op = Add;  op -= Add;          // делегат op пуст  **op?.Invoke(3, 4);**   // ошибки не будет, делегат просто не будет вызываться |

**Если делегат возвращает некоторое значение, то возвращается значение последнего метода из списка вызова** (если в списке вызова несколько методов). Например,

|  |  |
| --- | --- |
|  | class Program  {      delegate int Operation(int x, int y);        static void Main(string[] args)      {          Operation op = Subtract;          op += Multiply;          op += Add;          Console.WriteLine(**op(7, 2)**);    // Add(7,2) = 9          Console.Read();      }  //------------------------------------------------------------      private static int Add(int x, int y) { return x + y; }      private static int Subtract(int x, int y) { return x - y; }      private static int Multiply(int x, int y) { return x \* y; }  } |

**Делегаты как параметры методов**

Также делегаты могут быть параметрами методов:

|  |  |
| --- | --- |
|  | class Program  {  **delegate void** GetMessage(); // Определили делегат GetMessage()    **static void** Main(string[] args)      {          if (DateTime.Now.Hour < 12)          {              Show\_Message(GoodMorning);          }          else          {               Show\_Message(GoodEvening);          }          Console.ReadLine();      }  //-------Методы класса Program -----------------------------------  **private static void** Show\_Message(GetMessage \_del)       {del?.Invoke(); }  **private static void** GoodMorning()      { Console.WriteLine("Good Morning"); }  **private static void** GoodEvening()      { Console.WriteLine("Good Evening"); }  } |

**Обобщенные делегаты**

Делегаты могут быть обобщенными, например,

|  |  |
| --- | --- |
|  | **delegate** T Operation<T, K> (K val);    class Program  {      static void Main(string[] args)      {          Operation<decimal, int> op = Square;            Console.WriteLine(Square(5));          Console.Read();      }        static decimal Square(int n)      {          return n \* n;      }  } |

Дополнительные материалы

* **https://metanit.com/sharp/tutorial/3.13.php**