Делегаты

Делегаты представляют такие объекты, которые указывают на другие методы. То есть делегаты - это указатели на методы.

С помощью делегатов мы можем вызвать определенные методы в ответ на некоторые произошедшие действия. То есть, по сути, делегаты раскрывают нам функционал функций обратного вызова.

Функция обратного вызова — это передача одного исполняемого кода в качестве параметра другому. Обратный вызов позволяет в функции исполнять код который задается в аргументах при ее вызове.

Методы, на которые ссылаются делегаты, должны иметь те же параметры и тот же тип возвращаемого значения.

```
delegate int Operation(int x, int y);
delegate void GetMessage();
```

Первый делегат ссылается на функцию, которая в качестве параметров принимает два значения типа int и возвращает некоторое число.

Второй делегат ссылается на метод без параметров, который ничего не возвращает

Чтобы использовать делегат, надо создать его объект с помощью конструктора, в который мы передаем адрес метода, вызываемого делегатом. Чтобы вызвать метод, на который указывает делегат, надо использовать его метод **Invoke**.

```
class Program
{
    delegate int Operation(int x, int y);

    static void Main(string[] args)
    {
        // присваивание адреса метода через контруктор
        Operation del = new Operation(Add); // делегат указывает на метод Add
        int result = del.Invoke(4,5);
        Console.WriteLine(result);

        del = Multiply; // теперь делегат указывает на метод Multiply
        result = del.Invoke(4, 5);
        Console.WriteLine(result);

        Console.Read();
    }
    private static int Add(int x, int y)
    {
        return x+y;
    }
    private static int Multiply (int x, int y)
    {
        return x * y;
    }
}
```

Использование делегатов в качестве параметров методов:

```
class Program
{
   delegate void GetMessage();

   static void Main(string[] args)
   {
      if (DateTime.Now.Hour < 12)
      {
         Show_Message(GoodMorning);
      }
      else
      {
          Show_Message(GoodEvening);
      }
      Console.ReadLine();
   }
   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");
   }
}
```

Поскольку делегат объявлен внутри класса Account, то чтобы к нему получить доступ, используется выражение:

Account.AccountStateHandler.

Зачем они?

Не всегда у нас есть доступ к коду классов. Например, часть классов может создаваться и компилироваться одним человеком, который не будет знать, как эти классы будут использоваться. А использовать эти классы будет другой разработчик.

Так, здесь мы выводим сообщение на консоль. Однако для класса Account не важно, как это сообщение выводится. Классу Account даже не известно, что вообще будет делаться в результате списания денег. Он просто посылает уведомление об этом через делегат.

В первом методе метод **Combine** объединяет делегаты _del и del в один, который потом присваивается переменной del. Во втором методе метод **Remove** возвращает делегат, из списка вызовов у которого удален делегат _del.

```
// Регистрируем делегат
public void RegisterHandler(AccountStateHandler _del)
{
    Delegate mainDel = System.Delegate.Combine(_del, del);
    del = mainDel as AccountStateHandler;
}
// Отмена регистрации делегата
public void UnregisterHandler(AccountStateHandler _del)
{
    Delegate mainDel = System.Delegate.Remove(del, _del);
    del = mainDel as AccountStateHandler;
}
```