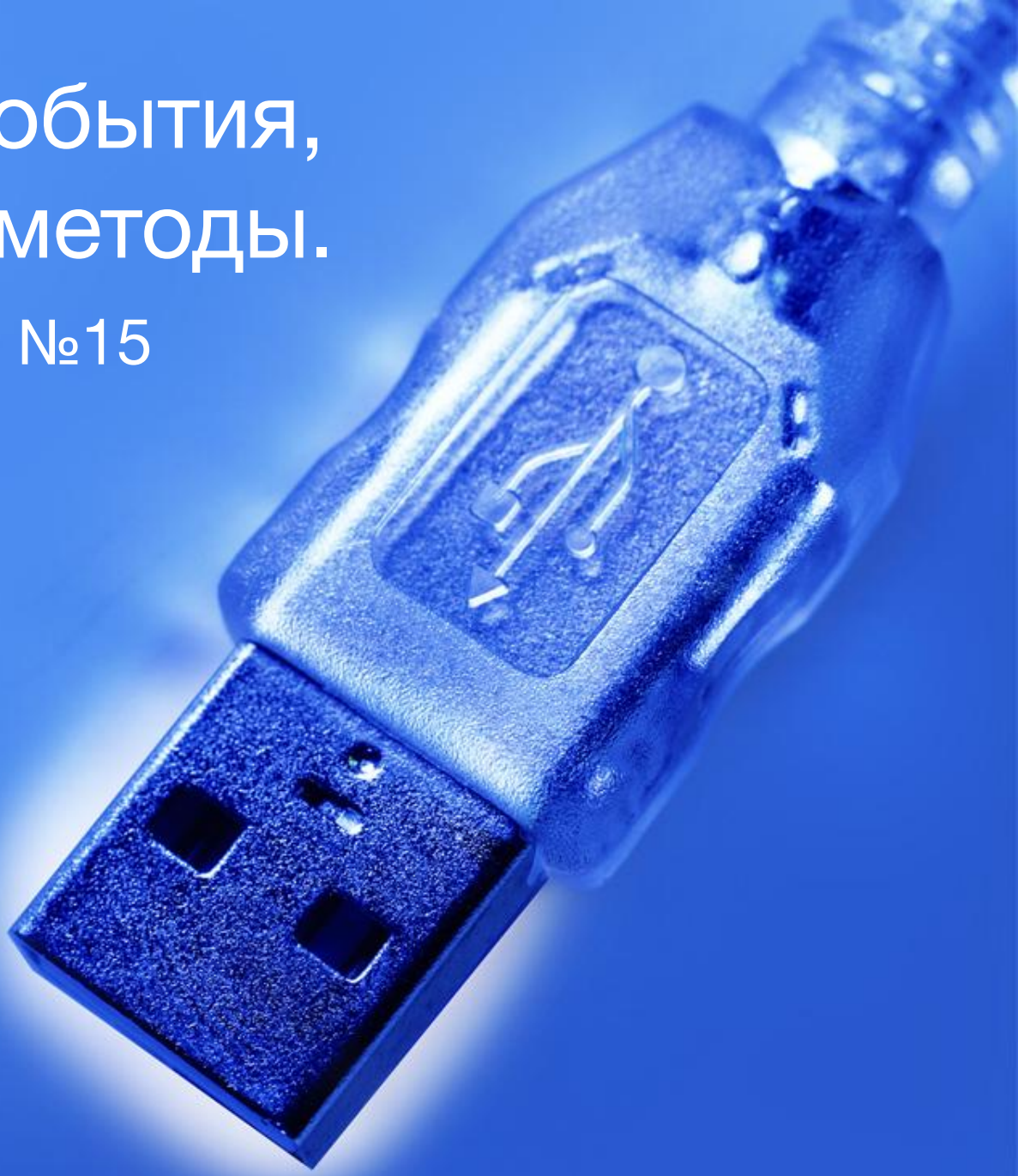
Делегаты, события, анонимные методы.

Делегаты лекция №15



Делегаты

- *Делегат — это вид класса, предназначенный для хранения ссылок на методы.*

Использование делегатов

- *Делегат,* как и любой другой класс, можно передать в качестве параметра, а затем вызвать инкапсулированный в нем метод.
- Делегаты используются для поддержки событий, а также как самостоятельная конструкция языка.

Описание делегатов

- **[атрибуты] [спецификаторы] delegate тип имяделегата ([параметры])**
- public delegate void D (int i);

Описание делегатов

- Делегат может хранить ссылки на несколько методов и вызывать их поочередно естественно, что сигнатуры всех методов должны совпадать.
- Объявление делегата можно размещать непосредственно в пространстве имен или внутри класса.

Почему делегат тип данных

- Делегат, как и всякий класс, представляет собой тип данных. Его базовым классом является класс System.Delegate.
- Наследовать от делегата нельзя.

Использование делегатов

Делегаты применяются в основном для следующих целей:

- получения возможности определять вызываемый метод не при компиляции, а динамически во время выполнения программы;
- обеспечения связи между объектами по типу ≪источник — наблюдатель≫;
- создания универсальных методов, в которые можно передавать другие методы;
- поддержки механизма обратных вызовов.

```
public static void ChetToUpper(ref string s)
{
    string temp = "" ;
    for ( int i = 0; i < s.Length; ++i )
        if ( i / 2 * 2 == i ) temp += char.ToUpper( s [ i ] );
        else temp += s [ i ] ;

     s = temp;
}
```

```
public static void NechetToUpper(ref string s)
{
    string temp = "";
    for (int i = 0; i < s.Length; ++i)
        if (i / 2 * 2 != i) temp += char.ToUpper(s[i]);
        else temp += s[i];

    s = temp;
}
```

```
static void Main(string[] args)
{
    string text = "chet ili nechet?";
    Del d;
    for(int i =0; i<2;i++)
    {
        d = ChetToUpper;
        if (i == 1) d = NechetToUpper;
        d(ref text);
        Console.WriteLine(text);
    }
}
```

```
ChEt iLi nEcHeT?
CHET ILI NECHET?
Для продолжения нажмите любую клавишу . . .
```

Вызов цепочки методов

При вызове последовательности методов с помощью делегата необходимо учитывать следующее:

- **сигнатура методов должна в точности соответствовать делегату;**
- **методы могут быть как статическими, так и обычными методами класса;**
- **каждому методу в списке передается один и тот же набор параметров;**
- **если параметр передается по ссылке, изменения параметра в одном методе отразятся на его значении при вызове следующего метода;**

Вызов цепочки методов

- если параметр передается с ключевым словом out или метод возвращает значение, результатом выполнения делегата является значение, сформированное последним из методов списка (в связи с этим рекомендуется формировать списки только из делегатов, имеющих возвращаемое значение типа voi d);
- если в процессе работы метода возникло исключение, не обработанное в том же методе, последующие методы в списке не выполняются, а происходит поиск обработчиков в объемлющих делегат блоках;
- попытка вызвать делегат, в списке которого нет ни одного метода, вызывает генерацию исключения System. Null Ref erenceExcepti on.

Паттерн «наблюдатель»

- **Для обеспечения гибкой, динамической связи между объектами во время выполнения программы применяется следующая стратегия. Объект, называемый *источником*, при изменении своего состояния, которое может представлять интерес для других объектов, посылает им уведомления. Эти объекты называются *наблюдателями*. *Получив уведомление, наблюдатель опрашивает источник, чтобы* синхронизировать с ним свое состояние.**

Код

```
namespace Observer
{
    public delegate void Del( object o ); // объявление делегата
    ссылок 2
    class Subj
    {// класс-источник
            Del dels; // объявление экземпляра делегата
            ссылок 3
            public void Register( Del d ) // регистрация делегата
            {dels += d;}
            ссылка 1
            public void OOPS()
            {// что-то произошло
                Console.WriteLine( "OOPS!" );
                if ( dels != null ) dels( this ); // оповещение наблюдателей
            }
    }
```

```
class ObsA
{// класс-наблюдатель
    ссылок 2
    public void Do( object o ) // реакция на событие источника
    {       Console.WriteLine( "Вижу, что OOPS!" );}
}
ссылка 1
static class ObsB
{// класс-наблюдатель
    ссылка 1
    public static void See(object o) // реакция на событие источ
    {
        Console.WriteLine( "Я тоже вижу, что OOPS!" );
    }
}
```

```
ссылок 0
class Program
{
    ссылок 0
    static voi
    {
        Subj s = new Subj();
        ObsA o1 = new ObsA();
        ObsA o2 = new ObsA();

        s.Register(o1.Do);
        s.Register(o2.Do);
        s.Register(ObsB.See);

        s.OOPS();
    }
}
```

```
OOPS!
Вижу, что OOPS!
Вижу, что OOPS!
Я тоже вижу, что OOPS!
Для продолжения нажмите любую клавишу . . .
```

Операции над делегатами

Делегаты можно *сравнивать на равенство и неравенство.*

Два делегата равны, если они оба не содержат ссылок на методы или если они содержат ссылки на одни и те же методы в одном и том же порядке.

Операции над делегатами

- **С делегатами одного типа можно *выполнять операции простого и сложного присваивания*, например:**
- Del **dl = new Del(o1.Do): // o1.Do**
- Del d2 = new Del(o2.Do); // o2.Do
- Del d3 = **dl + d2; // o1.Do и o2.Do**
- d3 += **d1 : // o1.Do. o2.Do и o1.Do**
- d3 -= d2; **// o1.Do и o1.Do**

Immutable delegate

- **Делегат, как и строка string, является неизменяемым типом данных, поэтому при любом изменении создается новый экземпляр, а старый впоследствии удаляется сборщиком мусора.**

Передача делегатов в методы

- Поскольку делегат является классом, его можно передавать в методы в качестве параметра. Таким образом обеспечивается *функциональная параметризация: в метод* можно передавать не только различные данные, но и различные функции их обработки.

```
public static void Table( Fun F, double x, double b )
{
    Console.WriteLine( "-----X-----Y-----" );
    while ( x <= b )
    {
        Console.WriteLine( "| {0,8:0.000} | {1,8:0.000} |", x, F(x));
        x+= 1;
    }
    Console.WriteLine( "----------------------");
}
ссылка 1
public static double Simple( double x )
{
    return 1;
}
```

```
Console.WriteLine( " Таблица функции Sin " );
Table( new Fun( Math.Sin ), -2, 2 );
Console.WriteLine( " Таблица функции Simple "
Table( new Fun( Simple ), 0, 3 );
```

```
 Таблица функции Sin
-----X-----Y-----
|   -2,000 |   -0,909 |
|   -1,000 |   -0,841 |
|    0,000 |    0,000 |
|    1,000 |    0,841 |
|    2,000 |    0,909 |
----------------------
 Таблица функции Simple
-----X-----Y-----
|    0,000 |    1,000 |
|    1,000 |    1,000 |
|    2,000 |    1,000 |
|    3,000 |    1,000 |
----------------------
```

Упрощенная форма записи Анонимный метод

```
Console.WriteLine( " Таблица функции Sin " );
Table( Math.Sin, -2, 2 );                                    // упрощение 1

Console.WriteLine( " Таблица функции Simple " );
Table( delegate (double x ){ return 1; }, 0, 3 ); // упрощение 2
```

Обратный вызов(CallBack)

Он представляет собой вызов функции, передаваемой в другую функцию в качестве параметра.

Домашнее задание

Пример:

- Fun[] masFun = new Fun[3];
- masFun[1] = Math.Cos;
- masFun[1](0,9);

Изменить лабораторную работу 8. Таким образом чтобы вызов пункта меню был через массив делегатов, инициализированных методами меню.