Делегаты и события

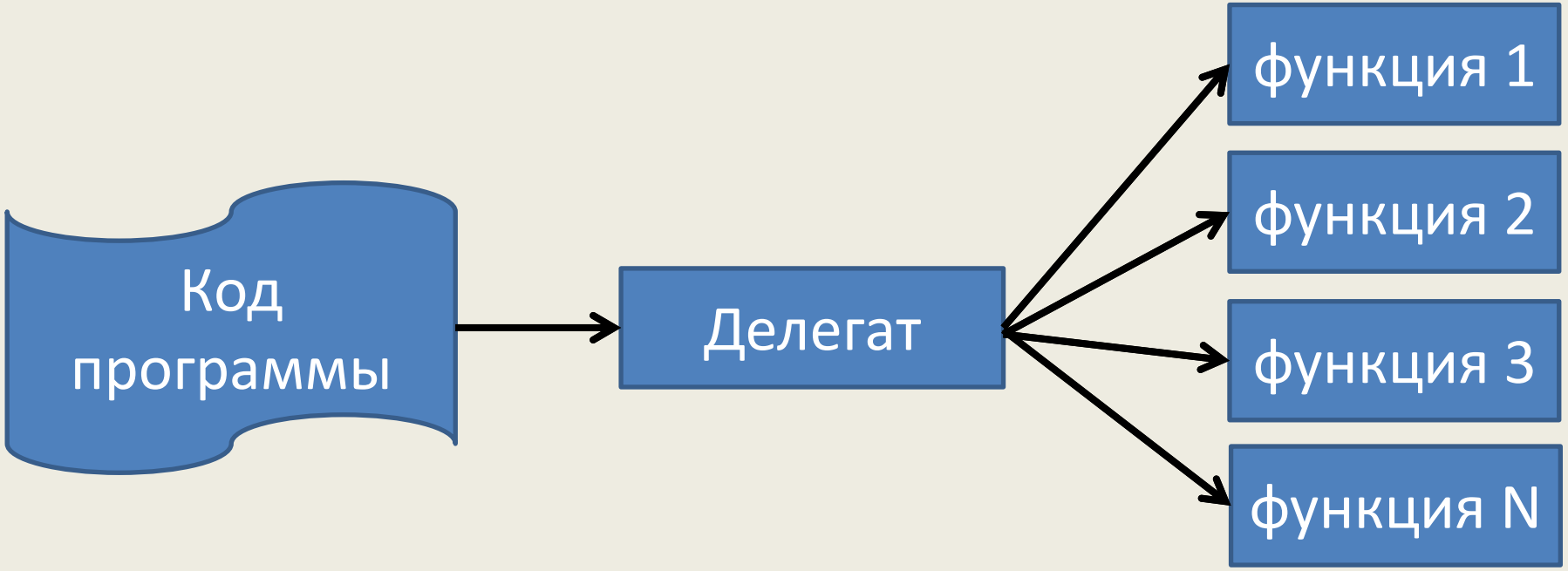
Делегаты

Делегат – вид класса, представляющий ссылки на методы.

- Делегаты инкапсулируют указатели.
- предоставляют удобные сервисы для работы с ними.
- Делегаты – immutable (неизменяемые) типы
  - не происходит изменения существующего объекта типа делегата, вместо этого создаётся новый объект (аналогия – тип string).

Делегаты представлены в виде классов-наследников Delegate и MulticastDelegate.

Делегат может ссылаться на N методов



Зачем нужны делегаты?

- Передача ссылок на методы в качестве параметров
  - поддержка механизма обратных вызовов;
- поддержка событий;
- создание анонимных методов;

Особенность наследования

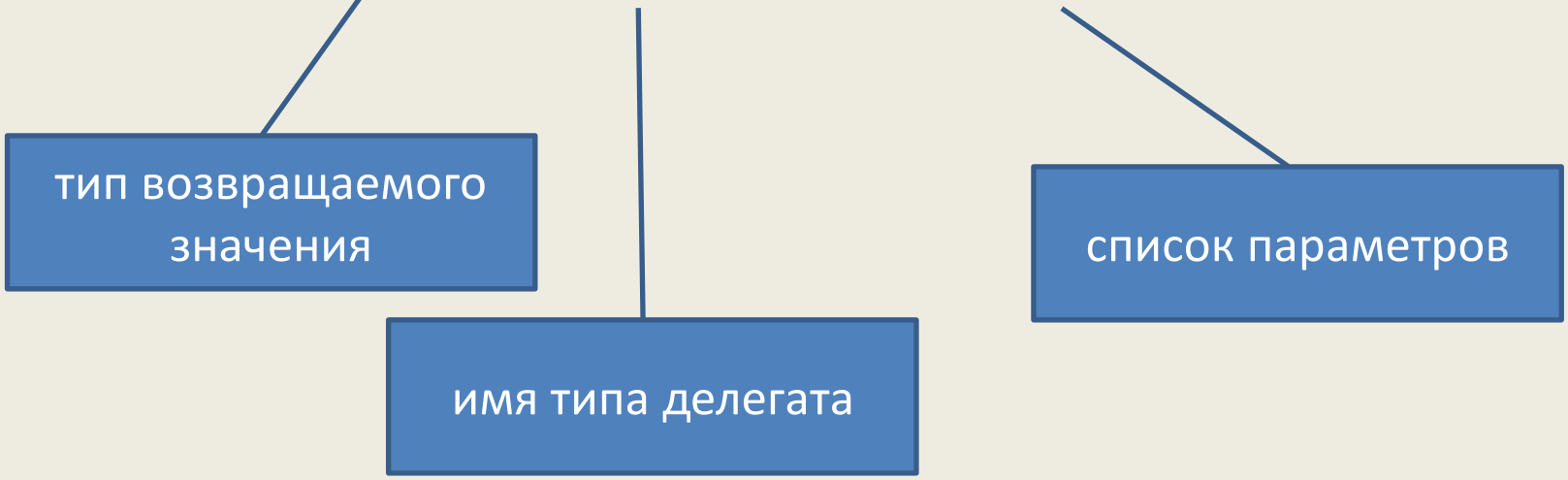
```
class Action :  MulticastDelegate   {    }
```

'CSConsoleApplication.Action' cannot derive from special class 'System.MulticastDelegate‘

В C# нельзя явно наследоваться от типов Delegate и MulticastDelegate.

Ключевое слово delegate

```
delegate void MyDelegate(string str);
```



На самом деле создаётся новый тип – MyDelegate, наследник MulticastDelegate

```
class MyDelegate :  MulticastDelegate    {
```

Использование типов-делегатов

```
delegate void MyDelegate(string str);
static void WriteSomething(string message)        {
        Console.WriteLine(message);
}
 static void Main()
{
        MyDelegate del = new MyDelegate(WriteSomething);
        del("Hello, world!");
        Console.ReadKey();
}
```

Вызов методов, на которые ссылается делегат, аналогично (синтаксически) вызову метода

Делегаты: static & instance-методы

Что нужно, чтобы вызвать static-метод:

- адрес метода
- параметры

Для вызова instance-метода требуется

- ссылка на объект, к которому привязан метод

Тип MulticastDelegate

Это базовый для делегатов в C# / .NET тип (он, в свою очередь – потомок Delegate)

Как следствие – обратим внимание на функциональность, которую он предоставляет.

Прежде всего – информация, требуемая для вызова методов, представлена в виде свойств:

- Method Возвращает метод, на который ссылается делегат
- Target Возвращает объект, к которому привязан метод, на который ссылается делегат

MuticastDelegate

Методы:

- **DynamicInvoke** – позволяет динамически обратиться к методам, связанным с делегатом.
- **GetInvocationList** – возвращает массив делегатов, привязанных к делегату, в порядке, в котором они вызываются.
- **Equality Operator** – *оператор (==),* позволяет определить равенство делегатов.
- **Inequality Operator** – *оператор (!=),* позволяет определить, различны ли делегаты.
- **Combine** – конкатенирует два (или более) делегата, создавая новый делегат, список вызовов которого включает списки объединяемых делегатов. Исходные делегаты не модифицируются.
- **Remove** – удаляет список вызовов одного делегата из списка вызовов другого. При этом создаётся новый делегат, список вызовов которого представляет собой результат удаления. Исходные делегаты не модифицируются.
- **CreateDelegate** – позволяет динамически создать делегат.

Операции над делегатами

Сравнение на равенство/неравенство:

```
public static bool operator == (Delegate d1, Delegate d2);
public static bool operator != (Delegate d1, Delegate d2);
```

- Эти операторы позволяют узнать, ссылаются ли 2 делегата на один и тот же метод
  - если делегаты ссылаются на множество методов, списоки методов должны быть идентичны
  - если делегаты не содержат ссылок на методы, они считаются эквивалентными (значение – null)
- При сравнении учитывается ссылка на объект, с которым связан метод.

Методы MulticastDelegate.Combine и MulticastDelegate.Remove

Эти методы предназначены для поддержки делегатов, которые ссылаются на несколько методов.

Метод Combine *позволяет объединить несколько делегатов в один*, в списке вызовов которого находятся ссылки на объединяемые делегаты.

Метод Remove производит *обратную Combine операцию.*

При вызове Combine или Remove создаётся новый объект!

В C# существует лаконичная форма записи вызова этих двух методов: ***+=*** *(для Combine),* ***-=*** *(для Remove), также возможно использовать просто операторы* ***+ и -.***

«Нулевые» делегаты

```
del = new MyDelegate(SomeMethod);
del -= new MyDelegate(SomeMethod);
//del == null (true)
```

Если не известно значение делегата на 100%, следует писать проверку вида:

```
if (del != null)
        del("Hello, World!\n");
```

Много примеров

- таймер
- таблица значений функции
- сортировка

События

Событие — элемент класса, позволяющий получать другим объектам (наблюдателям) уведомления *(об изменении своего состояния).*

Пример: кнопка -

Кнопка: элемент управления, нажатие на которой инициирует действие.

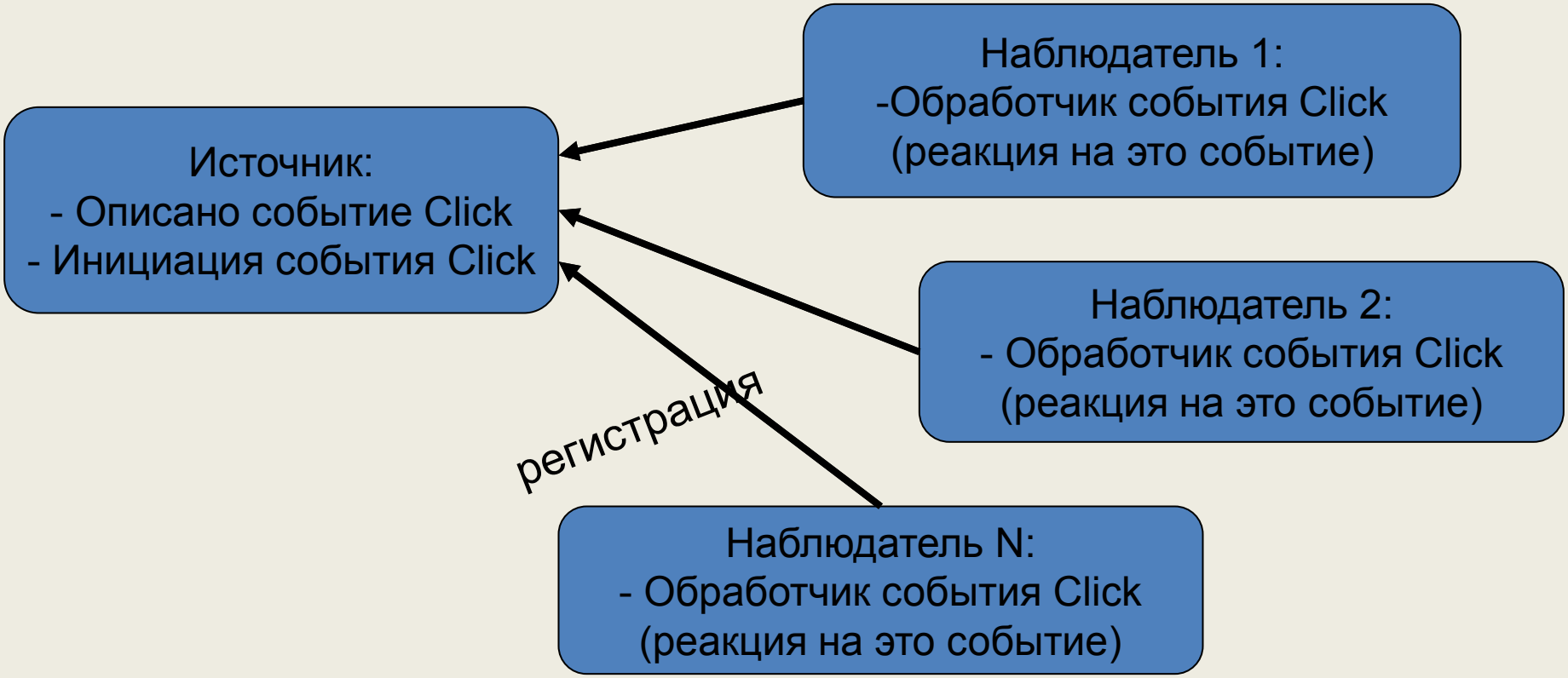
```
// Введем специальный делегат. delegate void ClickHandler();
class Button
{
        //Это общедоступное поле-делегат, к которому каждый
        //может присоединить собственный метод.
        public ClickHandler Click;
        //Идеализированная функция обработки нажатия на кнопку
        void OnMsg()
        {
                // Вот мы как бы засекли нажатие на кнопку.
                if (Click != null)
                        Click();
        }
}
```

Usage

```
static void Main(string[] args)
{
        Button button = new Button();
        button.Click += Button_ClickHandler;
        //нарушается инкапсуляция
        button.Click = null;
}
static void Button_ClickHandler()
{
        Console.WriteLine("Button pressed");

}
```

Модель взаимодействия



Больше инкапсуляции!

- .NET ориентирована на использование ООП!
- Необходимо соблюдать правила использования полей

<пример кода см. в статье по теме>

События

```
delegate void ClickHandler();
class Button
{
        public event ClickHandler Click;
        public void SimulateClick()
        {

            // Вызываем функции, связанные с событием Click,
            // предварительно проверив, зарегистрировался
            // ли кто-нибудь в данном событии.
            if (Click != null)
                Click();
        }
}
```

```
static void Main(string[] args
{
        Button btn = new Button();
        btn.Click += new ClickHandler(Btn_ClickHandler);
        btn.Click += Btn_OtherClickHandler;
        //имитируем нажатие пользователем на кнопку
        btn.SimulateClick();
}
static void Btn_ClickHandler()
{
        Console.WriteLine("Click handled!");
}
static void Btn_OtherClickHandler()
{
        Console.WriteLine("Handled twice!");
}
```

События: взгляд «изнутри»

```
public void add_Click(MyDelegate del)
{
        Click += del;
}
public void remove_Click(MyDelegate del)
{
        Click -= del;
 }
```

Контроль над событиями

```
event DelegateName SomeEvent
{
        add   { }
        remove {}
}
```

Встроенные делегаты

Делегаты, представляющие действие (соответствуют методам с параметрами типа T или без параметра, не возвращающим значений)

- System.Action
- System.Action<T>
- System.Action<T1, T2>
  - и так далее – перегрузка по generic-параметрам

Встроенные делегаты

Аналогичны Action, но позволяет методам, на которые ссылаются, возвращают значение типа TResult:

- System.Func<TResult>
- System.Func<TResult, T>
- System.Func<TResult, T1, T2>

На самом деле, generic-параметры определены с модификаторами in и out, но про это – в следующей серии.