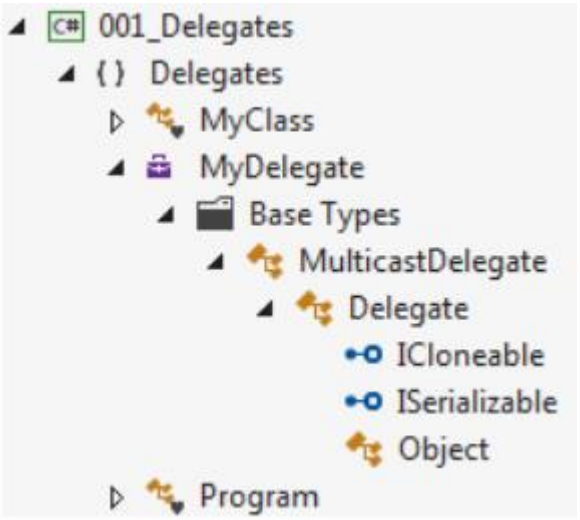
Delegates

Делегат (**delegate)**—это разновидность объектов которые содержат в себе указатели на методы. Те это безопасный указатель на метод. В классе-делегате строго указаны тип возвращаемого значения и аргументы метода (сигнатура)



> Все делегаты, являются производными от абстрактного класса `System.MulticastDelegate`, который в свою очередь наследуется от абстрактного класса `Delegate`.

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

Для объявления делегата используется ключевое слово **delegate**, после которого идет возвращаемый тип, название и параметры. 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, нам 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**. Кроме того, делегаты могут выполняться в асинхронном режиме, при этом нам не надо создавать второй поток, нам надо лишь вместо метода Invoke использовать пару методов **BeginInvoke/EndInvoke**

Сигнатура

Экземпляр делегата может ссылаться на любой статический метод или метод экземпляра –при условии, что сигнатура метода совпадает с сигнатурой делегата

```
static class MyClass
{
    public static void Method()
    {
    }
}
```

```
public delegate void MyDelegate();

class Program
{
    static void Main()
    {
        MyDelegate myDelegate = new MyDelegate(MyClass.Method);
        myDelegate.Invoke();
        myDelegate();
    }
}
```

Комбинированные делегаты

```
static void Main()
{
    MyDelegate delegate = null;
    MyDelegate delegate1 = new MyDelegate(Method1);
    MyDelegate delegate2 = new MyDelegate(Method2);
    MyDelegate delegate3 = new MyDelegate(Method3);

    delegate = delegate1 + delegate2 + delegate3;
}
```

Делегаты Action, Predicate и Func

Action

Делегат Action является обобщенным, принимает параметры и возвращает значение void:

```
public delegate void Action<T>(T obj)
```

Данный делегат имеет ряд перегруженных версий. Каждая версия принимает разное число параметров: от `Action<in T1>` до`Action<in T1, in T2,....in T16>`. Таким образом можно передать до 16 значений в метод.

Как правило, этот делегат передается в качестве параметра метода и предусматривает вывоз определенных действий в ответ на произошедшие действия

Predicate

Делегат `Predicate<T>`, как правило, используется для сравнения, сопоставления некоторого объекта T определенному условию. В качестве выходного результата возвращается значение true, если условие соблюдено, и false, если не соблюдено:

```
1   Predicate<int> isPositive = delegate (int x) { return x > 0; };
2
3   Console.WriteLine(isPositive(20));
4   Console.WriteLine(isPositive(-20));
```

В данном случае возвращается true или false в зависимости от того, больше нуля число или нет.

Func

Еще одним распространенным делегатом является **Func**. Он возвращает результат действия и может принимать параметры. Он также имеет различные формы: от `Func<out T>()`, где T - тип возвращаемого значения, до `Func<in T1, in T2,...in T16, out TResult>()`, то есть может принимать до 16 параметров.

```
1   TResult Func<out TResult>()
2   TResult Func<in T, out TResult>(T arg)
3   TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2)
4   TResult Func<in T1, in T2, in T3, out TResult>(T1 arg1, T2 arg2, T3 arg3)
5   TResult Func<in T1, in T2, in T3, in T4, out TResult>(T1 arg1, T2 arg2, T3 arg3, T
6   //.....................................
```

Данный делегат также часто используется в качестве параметра в методах

Anonymous Functions

Анонимные методы – это оператор или выражение "**inline",** которое можно использовать каждый раз, когда ожидается тип делегата. Ее можно использовать для инициализации именованного делегата или подставить вместо типа именованного делегата в качестве параметра метода

> Существует два типа анонимных методов (лямбда-методов),-это **Лямбда-операторы**и**Лямбда-выражения**

Anonymous Methods

Создание анонимных методов является, по существу, способом передачи блока кода в качестве параметра делегата

```
static void Main()
{
    MyDelegate myDelegate = delegate { Console.WriteLine("Hello world!");};
    myDelegate();
}
```

Lambda-operators

Во всех лямбда-выражениях используется лямбда-оператор =>, который читается как "переходит в". Левая часть лямбда-оператора определяет параметры ввода (если таковые имеются), а правая часть содержит выражение или блок

оператора. Лямбда-выражение **x=>x*x** читается как "**x** переходит в **x, x** раз"

Lambda expression

Лямбда - выражение—это анонимный метод, который содержит выражения и операторы и может использоваться для создания делегатов

> **Лямбда-Оператор** - это многооператорное лямбда выражение. **Лямбда-Выражение** - это однооператорный лямбда оператор

```
MyDelegate myDelegate;

myDelegate = delegate(int x) { return x * 2; };
myDelegate = (x) => { return x * 2;
myDelegate = x => x * 2;

int result = myDelegate(4);
```

Техника предположения делегатов

```
static void Main()
{
 // MyDelegate myDelegate = new MyDelegate(MyClass.Method);

    MyDelegate myDelegate = MyClass.Method;
    myDelegate();
}
```

> Для упрощения записи, можно использовать технику предположения делегатов.

Правила использования

Следующие правила применимы к области действия переменной в лямбда-выражениях

- Захваченная переменная не будет уничтожена сборщиком мусора до тех пор, пока делегат, который на нее ссылается, не выйдет за границы области.
- Переменная, введенная в лямбда-выражение, невидима во внешнем методе.
- Лямбда – выражение не может непосредственно захватывать параметры `ref` или `out` из включающего их метода.
- Лямбда – выражение не может содержать оператор `goto`, оператор `break` или оператор `continue`, для которых, метка перехода находится вне тела либо в теле содержащейся анонимной функции.