

Делегаты, лямбды в C# и UniRx

Что такое делегаты и сигнатура

- Делегат – это указатель на метод. С помощью него из любого места в коде можно вызвать указанный метод.
- Сигнатура метода – возвращаемый тип, и список типов всех аргументов этого метода.

```
void MethodName(int arg1, string arg2){ ... }
```

Например, метод выше имеет сигнатуру “void, int, string”

- Делегаты различаются между собой именно по сигнатуре методов,
на которые они могут указывать.

Использование делегатов

- Делегаты можно использовать в качестве типов переменных, если объявить их сигнатуру и имя
- Чтобы получить указатель на метод, достаточно написать его название в выражении
- Делегаты так же как и обычные переменные можно передавать в методы и хранить в классах и т.д.
- Определенный тип делегата поддерживает любые методы с такой же сигнатурой

```
1 public class TestClass
2 {
3     delegate int Operation(float a, float b); // Объявление делегата Operation с сигнатурой int, float, float
4
5     public void TestMethod()
6     {
7         Operation plus = Plus; // Мы можем записать указатель на метод в переменную с типом делегата нужной сигнатуры
8
9         // Будет = 8
10        int result1 = Evaluate(plus); // И передать например в качестве аргумента в метод
11
12        // Будет = 2
13        int result2 = Evaluate(Minus); // Или же можем напрямую передать указатель на метод, без переменной
14    }
15
16    2 usages
17    private int Evaluate(Operation operation)
18    {
19        return operation.Invoke(a: 5, b: 3); // С помощью Invoke вызываем метод по указателю в делегате
20    }
21
22    // Ниже объявлены методы с нужной сигнатурой. Чтобы получить их указатель, можешь просто написать их название
23    1 usage
24    private int Plus(float a, float b)
25    {
26        return (int) (a + b);
27    }
28
29    1 usage
30    private int Minus(float a, float b)
31    {
32        return (int) (a - b);
33    }
34 }
```

Анонимные методы и лямбды

- Методы, которые не имеют представления в классе, а сразу создают экземпляр объекта в методе называются анонимными
 - Существует краткая запись анонимных методов, которые зовутся лямбда выражениями
 - $(arg1, arg2) \Rightarrow arg1 + arg2$ – лямбда выражение, которое можно использовать вместо метода Plus из предыдущего примера
 - Лямбды можно записывать по разному:
 - $arg1 \Rightarrow arg1 * 5$
 - $() \Rightarrow CallSomeMethod()$
- ```
int result = CallSomeMethod();
return result;
}
```

# Action, Func, Predicate

- В C# существуют унифицированные делегаты, которые можно сразу использовать (записывая как типы), не объявляя их
- `Action<arg1, arg2, ...>` - возвращает `void`, может принимать сколько угодно аргументов
- `Func<arg1, arg2, ..., returnType>` - возвращает `returnType` (стоит всегда в конце), принимает сколько угодно аргументов
- `Predicate<arg>` - возвращает `bool`, принимает лишь 1 аргумент. Используется в большинстве случаев как фильтр, например в LINQ в методе `Where`, по факту является математическим предикатом

# Переделка примера под лямбды и Func

```
1 using System;|
2
3 public class TestClass
4 {
5 public void TestMethod()
6 {
7 Func<float, float, int> plus = (a, b) => (int) (a + b); // Создаем экземпляр анонимного метода
8
9 // Будет = 8
10 int result1 = Evaluate(plus);
11
12 // Будет = 2
13 int result2 = Evaluate(operation: (a :float , b :float) =>
14 {
15 return (int) (a - b);
16 }); // Можем не записывать, а сразу передавать в метод. Многострочный пример записи лямбда-выражения
17 }
18
19 private int Evaluate(Func<float, float, int> operation)
20 {
21 return operation.Invoke(5, 3);
22 }
23 }
```

# Полезные методы в UniRx

- `TimerFrame(int frames)` – Создает покадровый таймер
- `TakeUntilDestroy(GameObject/Component target)` – Уничтожает таймер, если уничтожен данный объект/компонент
- `TakeUntilDisable(GameObject/Component target)` – То же самое, но при выключении
- `TakeUntil(IObservable timer)` – Таймер будет работать, пока не истечёт таймер в аргументе этого метода
- `TakeWhile(Func<long, bool> predicate)` – Таймер остановится, когда предикат выдаст `false`



# Полезные методы в UniRx

- SkipUntil(IObservable timer) – Таймер не будет обрабатываться, пока внутренний таймер не окончится
- SkipWhile(Func<T, bool> predicate) – Таймер не будет обрабатываться, пока предикат выдает true
- Finally(Action action) – Выполняет action когда таймер кончается или вообще уничтожается
- Interval(TimeSpan period) – Повторяющийся таймер с указанным периодом
- IntervalFrame(int frameCount, FrameCountType type) – Тоже самое, но по кадрам, можно указать на какие типы тиков ориентироваться (Update/FixedUpdate/EndOfFrame)

# Примеры таймеров

## 1) Перемещение объекта к точке с заданной

```
float speed = 0.1f;
Vector3 point = new Vector3(x: 10f, y: 5f, z: 3f);

Observable.IntervalFrame(1)
 .TakeWhile(x :long => Vector3.Distance(a: point, b: transform.position) > 0)
 .Finally(() => transform.position = point) // IObservable<long>
 .Subscribe(onNext: x :long => transform.Translate(translation: (point - transform.position).normalized * speed));
```

## 2) Вызов какого-либо действия раз в секунду, когда объект вблизи точки на 2 метра, и пока он существует

```
Observable.Interval(period: 1.Seconds())
 .SkipWhile(x :long => Vector3.Distance(a: point, b: transform.position) > 2f)
 .TakeUntilDestroy(gameObject) // IObservable<long>
 .Subscribe(onNext: x :long => { DoSomeAction(); });
```