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

```
public class TestClass
{
    delegate int Operation(float a, float b); // Объявление делегата Operation с сигнатурой int, float, float

    public void TestMethod()
    {
        Operation plus = Plus; // Мы можем записать указатель на метод в переменную с типом делегата нужной сигнатуры

        // Будет = 8
        int result1 = Evaluate(plus); // И передать например в качестве аргумента в метод

        // Будет = 2
        int result2 = Evaluate(Minus); // Или же можем напрямую передать указатель на метод, без переменной
    }

    2 usages
    private int Evaluate(Operation operation)
    {
        return operation.Invoke( a: 5,  b: 3); // С помощью Invoke вызываем метод по указателю в делегате
    }

    // Ниже объявлены методы с нужной сигнатурой. Чтобы получить их указатель, можешь просто написать их название
    1 usage
    private int Plus(float a, float b)
    {
        return (int) (a + b);
    }

    1 usage
    private int Minus(float a, float b)
    {
        return (int) (a - b);
    }
}
```

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) => arg1 + arg2 – лямбда выражение, которое можно использовать вместо метода Plus из предыдущего примера
- Лямбды можно записывать по разному:
  - arg1 => arg1*5
  - () => CallSomeMethod()

```
(arg1, arg2) =>
{
    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

```
using System;

public class TestClass
{
    public void TestMethod()
    {
        Func<float, float, int> plus = (a, b) => (int) (a + b); // Создаем экземпляр анонимного метода

        // Будет = 8
        int result1 = Evaluate(plus);

        // Будет = 2
        int result2 = Evaluate( operation: (a :float , b :float ) =>
        {
            return (int) (a - b);
        }); // Можем не записывать, а сразу передавать в метод. Многострочный пример записи лямбда-выражения
    }

    2 usages
    private int Evaluate(Func<float, float, int> operation)
    {
        return operation.Invoke(5, 3);
    }
}
```

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(IObsevable 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 обрабатыватся, пока предикат выдает true
- Finally(Action action) – Выполняет action когда таймер кончается или вообще уничтожается
- Interval(TimeSpan period) – Повторяющийся таймер с указанным периудом
- IntervalFrame(int frameCount, FrameCountType type) – Тоже самое, но по кадрам, можно указать на какие типы тиков ориетировatся (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(); });
```