Лабораторная работа №7

# Разработка приложений с применением ООП

В этой лабораторной работе мы научимся создавать ботов для системы обмена сообщениями Telegram на языке C#, применять ООП подход к проектированию программы и декомпозицию предметной области, а также подключать и использовать дополнительные библиотеки из репозитория NuGet.

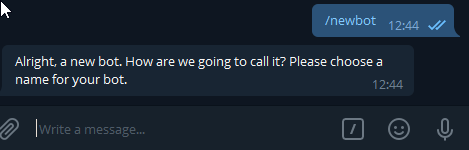
## Подготовка к выполнению лабораторной работы

Боты для телеграмм — это многофункциональный инструмент, который может использоваться в личных, образовательных и бизнес-целях. Каждый пользователь Telegram может создавать неограниченное количество ботов и каждый бот будет иметь уникальный токен.

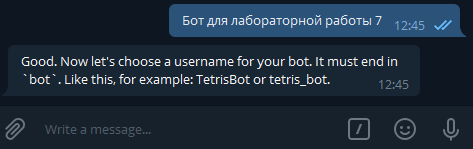
Поэтому первым шагом при создании бота является получение уникального ключа, который будет использоваться для авторизации в Telegram при запуске программы. Такие ключи принято называть Токенами.

Для того чтобы создать бота и получить токен, необходимо через клиент Telegram написать @BotFather и отправить команду для создания бота /newbot:

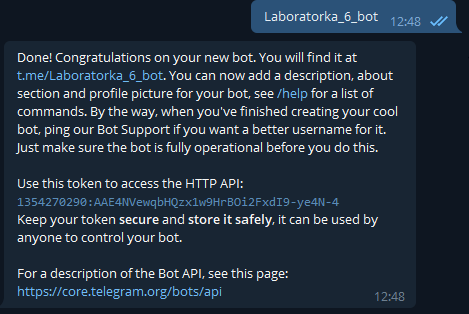




Затем сообщением отправить название бота. Оно может содержать любое количество слов на любом языке:



А после этого имя бота, оно должно быть записано латиницей и содержать только буквы, цифры, знаки подчеркивания - как названия переменных в языках программирования. Имя бота обязано заканчиваться словом **bot**.



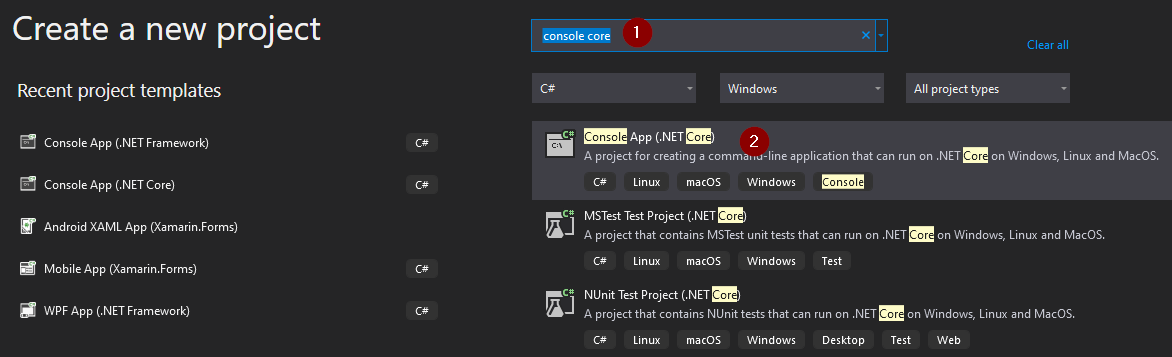
После этого **@BotFather** пришлет вам сообщение, которое будет содержать токен для авторизации из программы. На этом скриншоте токен это - **1354270290:AAE4NVewqbHQzx1w9HrBOi2FxdI9-ye4N-4**

Внимание! Созданный токен должен храниться в секрете так же, как и ваши пароли. Если вы случайно обнародовали ваш ключ (например, подготавливая скриншоты для методички), то вы должны пересоздать его, используя соответствующую команду **@BotFather**.

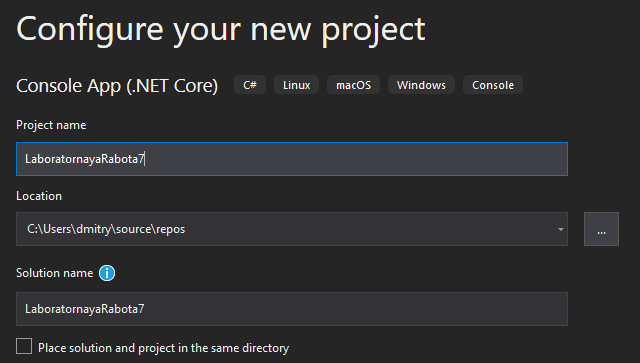
# Создание проекта

Программа на языке **C#** для бота **Telegram** может быть как консольной, так и графической WPF, в зависимости от ваших предпочтений, вы можете использовать тот или иной тип проекта. Однако, на практике лучше всего подходит тип проекта **Console Application .net Core**, так как это позволит в дальнейшем запускать бота на удаленном сервере под управлением **Linux**. Поэтому в этой лабораторной работе будем использовать именно этот тип проекта.

При создании нового проекта в поле фильтра вводим “**console core**” и выбираем нужный пункт из списка шаблонов проектов:

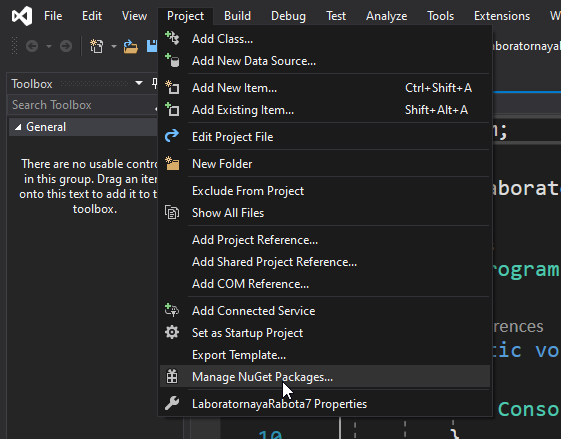


Как и предыдущих работах вводим название проекта и место его размещения на диске:

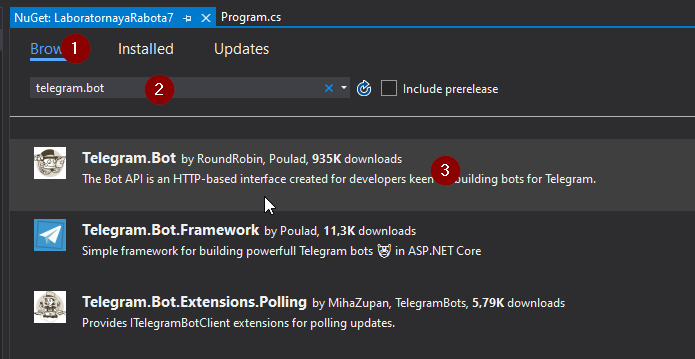


После того как проект создан, для начала разработки бота нужно подключить вспомогательные библиотеки, которые возьмут на себя все внутренние нюансы по связи с серверами **Telegram**, и позволят нам сосредоточиться на основных деталях: что бот будет делать и как себя вести с пользователем.

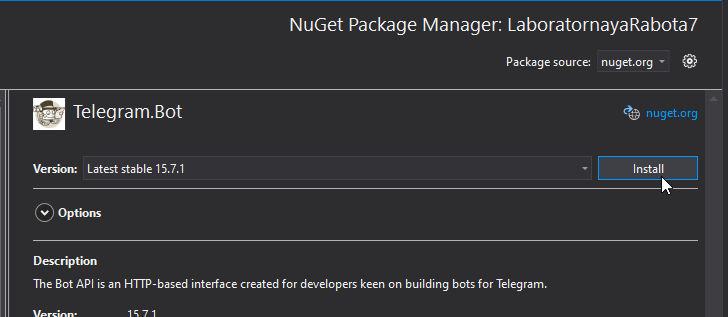
Для распространения библиотек в C# используется система управления пакетами **NuGet** и общедоступный репозиторий библиотек **NuGet.org**. Для того чтобы найти подходящие библиотеки и подключить их к проекту нужно открыть клиент для управления пакетами. Для этого выбираем пункт меню **Project\Manage NuGet Packages.**



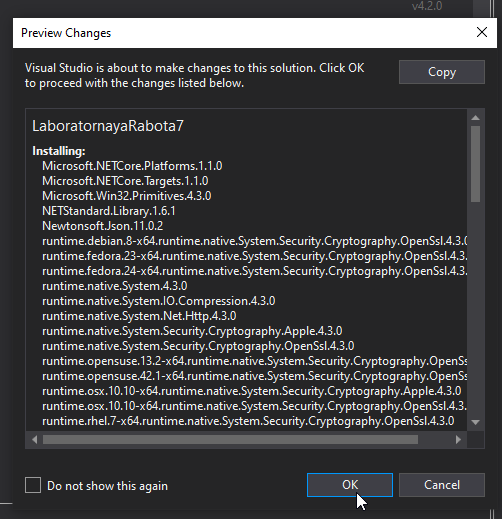
В открывшемся окне переходим на вкладку **Browse** и в поле поиска вводим “**telegram.bot**”. Вероятнее всего, нужная библиотека будет первой в списке находок, выбираем ее мышью



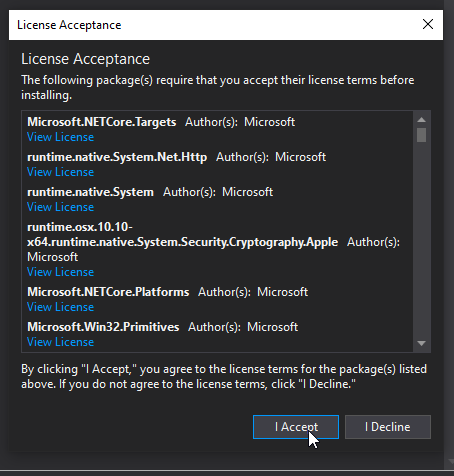
И в правой части окна нажимаем на кнопку **Install**. Это приведет к тому, что из Интернет будут скачаны все нужные файлы и добавятся в наш проект.



Установка займет какое-то время, в процессе установки нужно согласиться с предлагаемыми изменениями проекта:

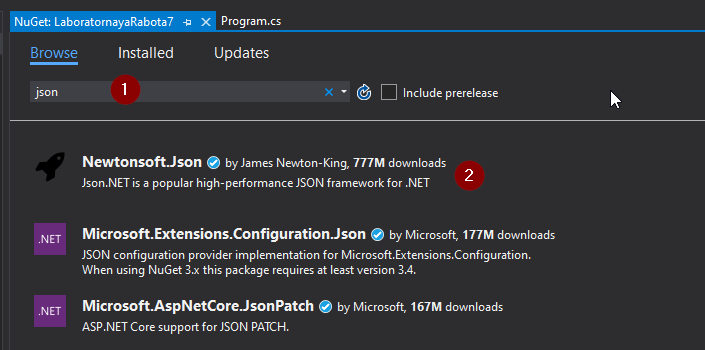


А также принять лицензионное соглашение:

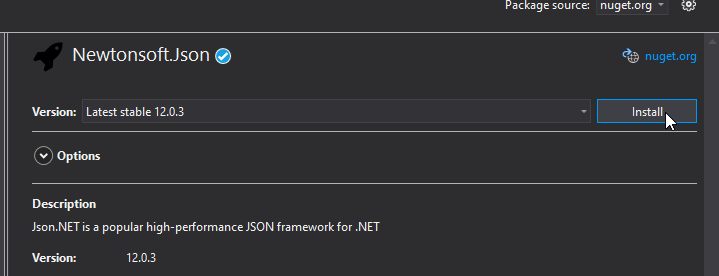


Помимо библиотеки для работы с **Telegram** нам понадобится библиотека для парсинга **Json** (читается “**джейсон**”). **Json** — это текстовый формат обмена данными, основанный на **JavaScript** и широко применяется для обмена данными с веб сервисами, чем мы и будем заниматься.

В строке поиска вводим “**json**” и находим библиотеку **Newtonsoft.Json**.

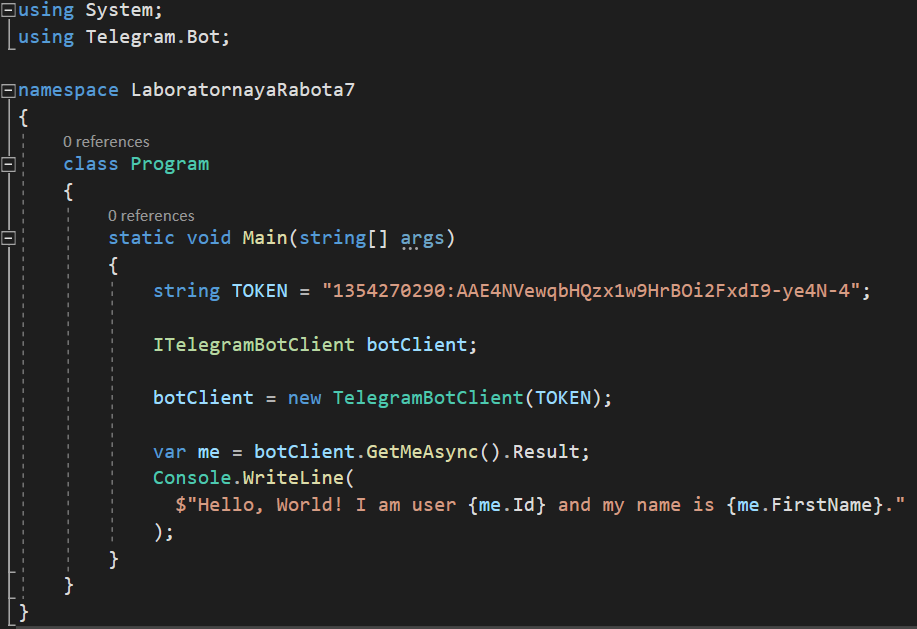


Нажимаем кнопку Install для установки. В процессе соглашаемся со всеми предложениями.



После того как все нужные библиотеки были установлены, нужно проверить, что все прошло правильно, без ошибок и попробовать запустить бота в тестовом режиме.

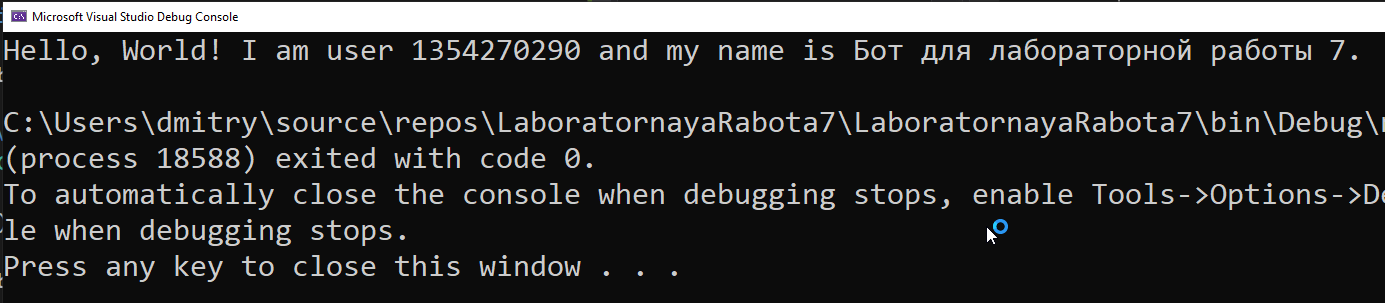
Перепечатайте код из листинга ниже, полностью заменив содержимое файла **Program.cs**. Не забудьте указать токен который был выделен для лично для вас.



В первой версии программы создается экземпляр класса **TelegramBotClient** в его конструктор передается токен для подключения к серверу.

После этого вызывается асинхронный (**Async**) метод **botClient.GetMe**, который возвращает информацию о самом боте - его имя, уникальный идентификатор и другие сведения. К сожалению, рассмотрения особенностей Асинхронных методов выходит за рамки нашего курса, тут я лишь уточню, что методы, помеченные как **async\await** выполняются параллельно основной программе и чаще всего применяются при разработке сетевых приложений (как наше).

Затем запускаем программу, и на экране должно быть отображено что-то похожее на экран ниже:



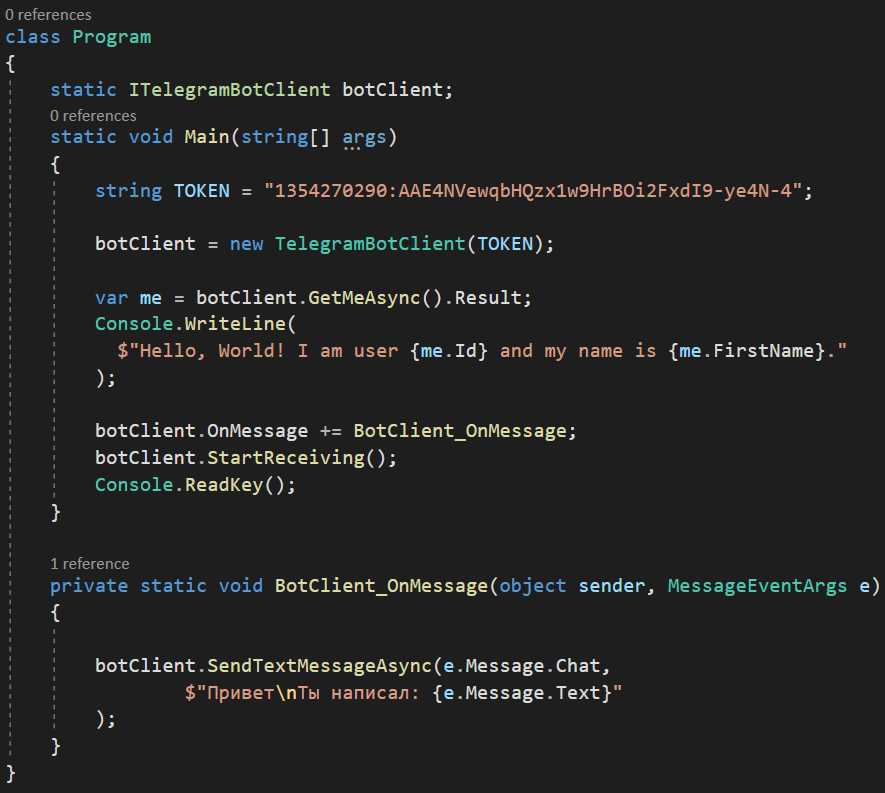
Если бот смог успешно подключиться к серверу и запросить информацию о себе, значит все подготовительные работы прошли успешно и можно начинать разработку.

# Эхо бот

Основная задача любого бота в мессенджере — это общение с пользователями. Это значит, что бот должен принимать сообщения пользователей и уметь на них отвечать.

Так же, как и с графическим интерфейсом пользователя, где для реакции на воздействия пользователя использовались события, и обработчики событий в приложении с ботом используется событий механизм.

Основное событие, которое может генерировать бот — это событие о новом сообщении пользователя. Так как в этой программе нет **XAML** и графического редактора, обработчик события нужно подключить вручную в коде. Для этого немного перепишем исходную программу



В этой версии было изменено следующее:

1. К событию **OnMessage** объекта **botClient** был подключен (+=) обработчик события **BotClient**\_**OnMessage**. Он будет вызываться каждый раз, когда от пользователя придет новое сообщение
2. Бот переведен в режим приема при помощи метода **StartReceiving**(). Только после этого бот начнет принимать входящие сообщения
3. Для того чтобы программа не закрылась сразу же, вызываем **Console.ReadKey**.
4. Описан обработчик сообщения **BotClient\_OnMessage**, в котором пользователю отправляется обратно текст, который он прислал.
5. Указатель на **botClient** был вынесен в виде статического поля класса **Program**. Это необходимо поскольку объект **botClient** должен быть доступен как в методе **Main**, так и в обработчике **BotClient\_OnMessage**

Если запустить эту версию программы, то с ботом можно будет пообщаться через клиент **Telegram**:



В качестве параметра в обработчик события **BotClient\_OnMessage** приходит аргумент **MessageEventArgs e**, который содержит всю актуальную информацию о том, кто, когда и что прислал боту. В частности **e.Message.Text** - содержит текст сообщения, **e.Message.Chat** - идентификатор чата, в который пришло сообщение (бот может одновременно общаться с тысячами пользователей, в том числе и в групповых чатах), а **e.Message.From** - информация об отправителе сообщения.

Для того, чтобы отправить ответ в тот чат, из которого пришло сообщение, необходимо вызывать метод объекта **botClient.SendMessageAsync**:

**botClient.SendTextMessageAsync(e.Message.Chat, “Hello”);**

Первым параметром указывается номер чата, в который нужно отправить сообщение, а вторым - сам текст сообщения.

# Использование Веб Сервисов

В этом году на мир свалилось множество бед и невзгод. Уровень стресса у населения повышен, и нервное напряжение нарастает. Что может помочь человечеству, если не телеграмм-бот, присылающий фотографии животных?!

Теперь, когда мы научились принимать и отправлять сообщения, попробуем добавить в бот какую-то полезную функциональность и заодно разберемся как использовать доступные и открытые веб сервисы, использующие **REST API**.

При разработке различных клиент-серверных приложений разработчикам часто приходится использовать какие-то тестовые серверы для разработки и отладки своих программ. И довольно популярны серверы, которые по запросу возвращают случайные наборы данных, например фотографии животных, в частности кошек и собак. Ими мы и воспользуемся.

Разработаем бота, который по запросу пользователя будет отправлять случайную фотографию.

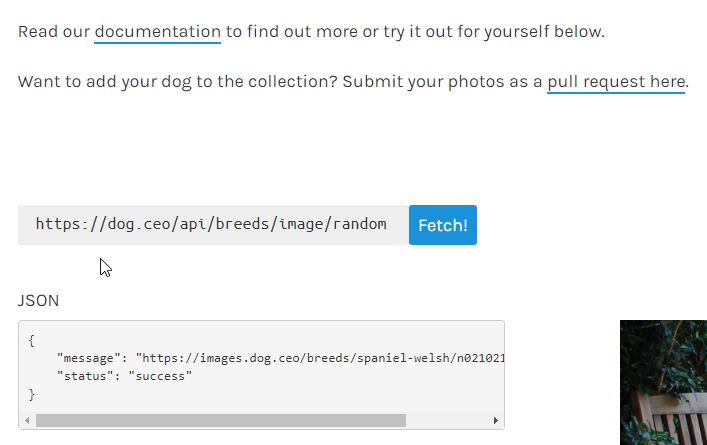
Мы могли бы остановиться только на кошках, но как показало исследование Целевой Аудитории, наши пользователи делятся на кошатников и собачников, поэтому придется поддерживать работу с несколькими типами данных, и построить в приложении соответствующую архитектуру классов.

Для получения фотографий выберем два сервиса: <https://thecatapi.com> и <http://dog.ceo>. При желании вы можете найти другие сервисы в каталоге <https://www.programmableweb.com/category/all/apis>

## Как получить данные из веб сервиса

Получение данных производится по стандартному протоколу HTTP, это то же самое что использует браузер, и браузер так же можно использовать для отладки работы.

Как правило, сервисы описывают свой программный интерфейс в документации, например, для dog.ceo он описан на странице <https://dog.ceo/dog-api/>.



Это значит, что, отправив запрос <https://dog.ceo/api/breeds/image/random> в ответ мы получим **JSON** данные, которые будут содержать ссылку на случайную фотографию.

Наша задача разбивается на три:

1. Отправить запрос и получить ответ
2. Обработать ответ и вычленить из него ссылку на фотографию
3. Отправить фотографию пользователю

### Отправка Веб-запроса

Для выполнения веб запросов в **C#** используется специальный класс **WebClient**:

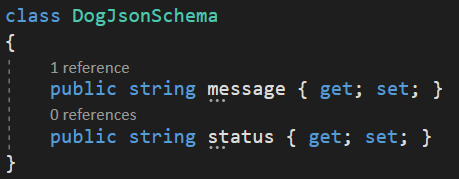


В этом кусочке кода создается объект класса **WebClient**, и выполняется запрос с веб-сервису **dog.ceo**. А результат запроса сохраняется в переменной **json**.

### Обработка ответа

Для того чтобы преобразовать **JSON** данные в объекты, которые сможет понять C# нужно создать класс, который будет содержать все те поля, которые содержатся в **JSON** (смотри скриншот выше). Такого типа классы обычно называются **Схемами Данных**, а по-английски - **Schema**.

Создадим класс **DogJsonSchema** в который будем преобразовывать результат запроса к веб-сервису **dog.ceo.** Обратите внимание, что он должен совпадать с данными описанными в документации к сервису:



Тогда, для преобразования нужно будет написать следующее:



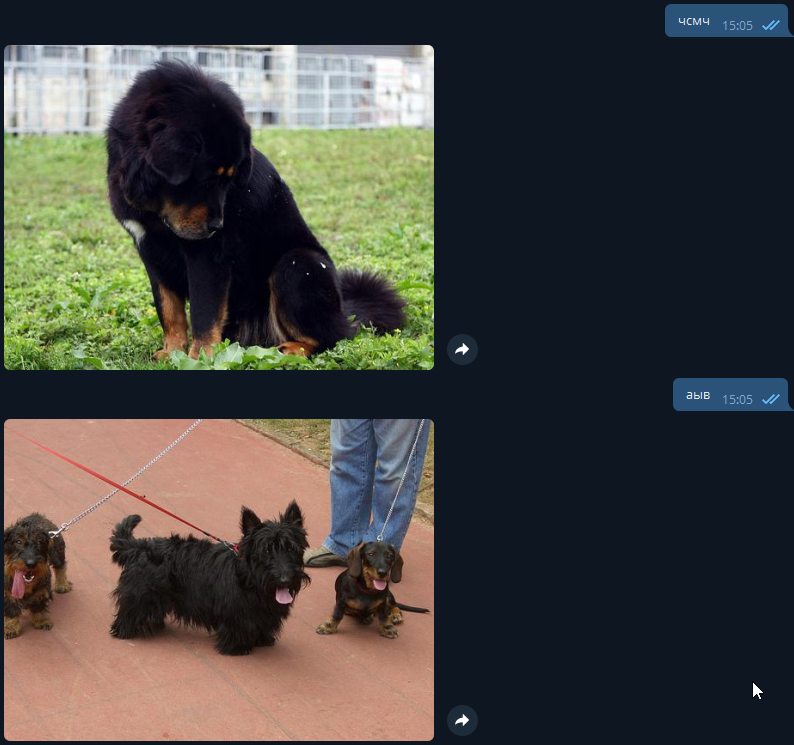
После этого объект **dog** будет иметь поле **message**, которое содержит ссылку на файл со случайной собакой.

Отправка фотографии пользователю у объекта **botClient** есть специальный метод **SendPhotoAsync** и фотография отправляется следующим образом:



И весь измененный код вместе (следите за скобками!), метод Main остался без изменений.



После того как мы внесем все изменения, запустим программу и попробуем написать боту что-нибудь в чате, мы получим в ответ случайные фотографии:

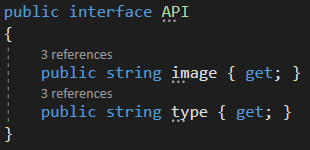
Однако, то, что мы написали хорошо подходит для отправки фотографий собак, но в нашей задаче так же есть и отправка кошек. Нужно снова преобразовать программу, и создать иерархию классов кошек и собак.

# Классовая структура

Для начала, необходимо выделить общие свойства, которые будут характерны для всех классов программы:

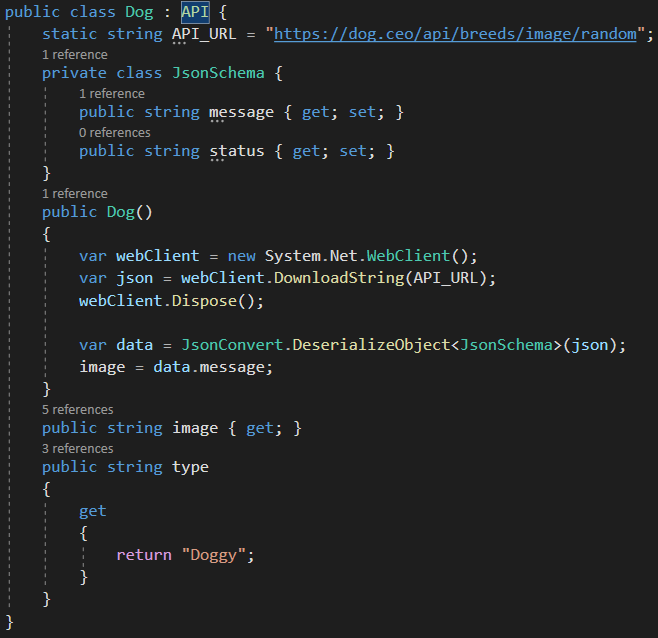
* Ссылка на фотографию
* Тип фотографии (кошка\собака)

Объявляем интерфейс, который содержит эти свойства

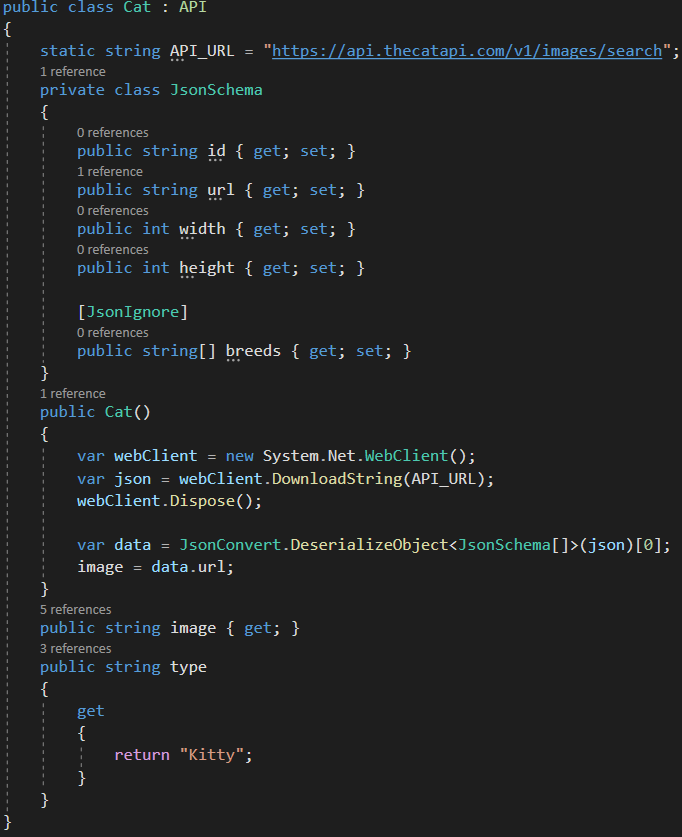


После этого необходимо создать классы имплементирующие данный интерфейс для кошек и собак. В эти же классы (а точнее в их конструкторы) поместим код получающий ссылку на картинку.

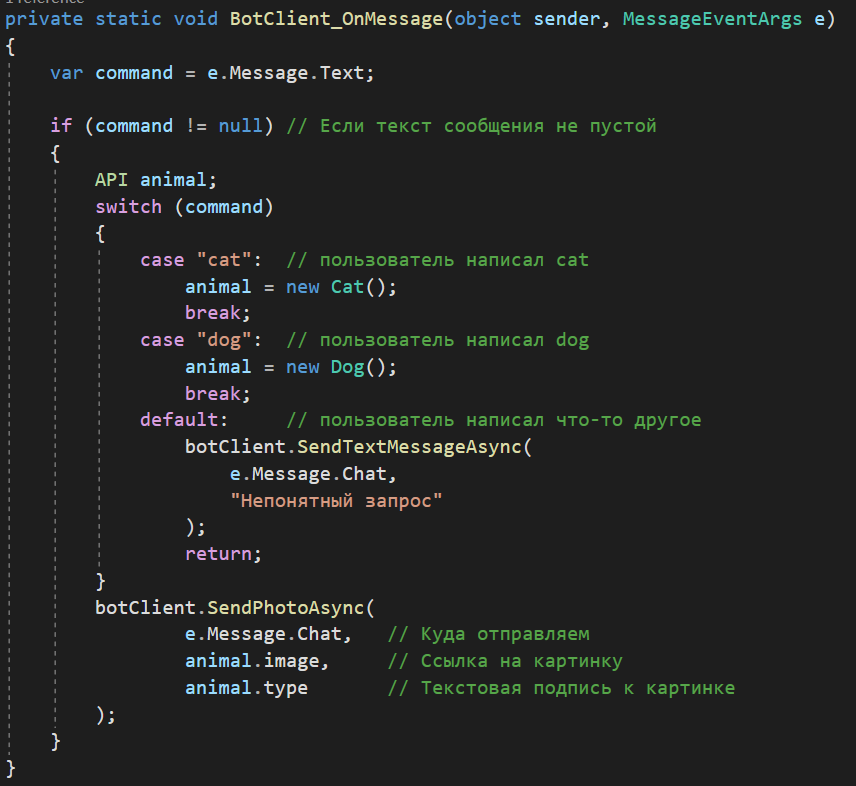
**Класс Dog:**



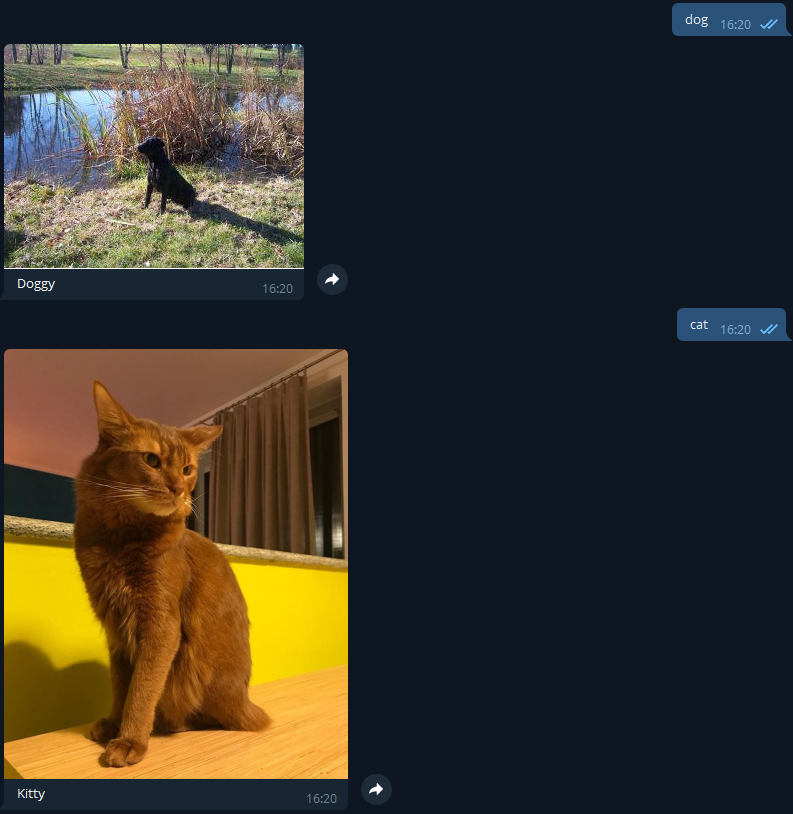
**Класс Cat:**



И в окончании работ перепишем обработчик события сообщения так, чтобы он использовал новые классы и интерфейс:



После внесения изменений снова запустим программу и попробуем отправить в бота команды dog и cat:



Все работает! (по задумке)

Задания для закрепления понимания:

* По команде Reverse - прислать пользователю перевернутый текст:
  + Reverse Привет мир!  
    !рим тевирП
* По комманде Upper - прислать пользователю текст записанный капсом:
  + Upper Привет мир!  
    ПРИВЕТ МИР!
* По запросу “Какой сегодня день” прислать пользователю название дня.
* По запросу “сколько времени в Париже” прислать пользователю текущее время в Париже.