Day 03. Async/Await

## Общие требования

* Убедитесь, что на вашем компьютере установлен [SDK для разработки на .NET 5](https://dotnet.microsoft.com/download) и вы используете именно его.
* Помните, ваш код будут читать! Обратите особое внимание на оформление вашего кода и именование переменных. Обязательно придерживайтесь общепринятых стандартов [C# Coding Conventions](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/inside-a-program/coding-conventions).
* Самостоятельно выберите удобную для себя IDE.
* Программа должна иметь возможность запуска через командную строку dotnet.
* В каждом из заданий указаны примеры ввода и вывода. Решение должно использовать их как верный формат.
* В начале каждого задания приведен список рекомендованных языковых конструкций.
* Если затрудняетесь в решении задачи, обратитесь с вопросами к другим участникам бассейна, интернету, Google, посмотрите на StackOverflow.
* Избегайте **хардкода** и “**магических чисел**”.
* Вы демонстрируете все решение, верный результат работы программы – лишь один из способов проверки ее корректной работы. Поэтому когда необходимо получить определенный вывод в результате работы ваших программ, запрещено показывать пред рассчитанный результат.
* Обращайте особое внимание на термины, выделенные **bold** шрифтом: их лучше сразу погуглить, их изучение пригодится вам как в выполнении текущего задания, так и в вашей дальнейшей карьере .NET разработчика.
* Have fun :)

## Требования к заданиям дня

* Каждому из заданий должно соответствовать отдельное решение, точкой входа в которое является консольное приложение, созданное на основе стандартного шаблона .NET SDK.
* Используйте **var**.
* Название решения (и его отдельного каталога) должно выглядеть как d{*xx*}, где *xx* - цифры текущего дня. Названия проектов указаны в задании.
* Для форматирования выходных данных используйте **культуру** en-GB: N2 для вывода денежных сумм, d для дат.

# Per aspera ad astra

Все мы немного романтики. Интересующиеся и восхищенные бескрайними просторами звездного неба, таинственной космической неизвестностью. А неизвестностью ли? Ведь иногда частички бескрайнего космоса оказываются очень-очень близко к Земле. А еще космос можно сфотографировать.

У NASA есть набор [API](https://api.nasa.gov/) (**Application Programming Interface**) с разнообразной информацией о просторах нашей Вселенной. Там во вкладке “Browse APIs*”* можно найти внушительный список вариантов использования. Так почему бы не попробовать посмотреть на космос с помощью программирования?

Поехали! ©

## Что нового

* Конфигурация приложений
* API, REST
* HTTP запросы, идемпотентность
* Async/Await, TAP
* Разделение приложения на проекты
* Ковариантность и контравариантность

## Рекомендованные языковые конструкции

* Async/Await
* Интерфейсы и абстрактные классы
* Классы и записи
* Generics
* HttpClient
* JSON

## Структура проекта

d03/

d03.Nasa/

Apod/

Models/

MediaOfToday.cs

ApodClient.cs

NeoWs/

Models/

AsteroidInfo.cs

AsteroidLookup.cs

AsteroidRequest.cs

NeoWsClient.cs

INasaClient.cs

ApiClientBase.cs

d03.Host/

Program.cs

appsettings.json

## Абстракции

В заданиях сегодняшнего дня вам нужно будет реализовать обращение к методам NASA API, получение данных об интересующих нас объектах и вывод их в необходимом формате. КажReqдое задание будет для обращения к отдельному ресурсу. Но сначала давайте поговорим об общем.

Для начала работы с **API** вы должны получить ключ доступа, **API Key**: это можно сделать на вкладке “Generate API Key*”*. Мы будем использовать его в наших методах, но хранить ключи и пароли (и прочие чувствительные данные) прямо в коде приложения не безопасно. Запомните это как истину!

Днем ранее мы уже писали собственную конфигурацию, но на деле в .NET Core для этого существуют уже готовые средства. Давайте использовать их: файл конфигурации ***appsettings.json*** и **nuget-пакеты** для управления конфигурацией из JSON файла. Установите [Microsoft.Extensions.Configuration.Json](https://www.nuget.org/packages/Microsoft.Extensions.Configuration.Json/6.0.0-preview.4.21253.7) и обратите внимание на пакеты-зависимости, которые nuget-менеджер установит для него. Пример настройки и использования конфигураций приложения можно найти в [статье](https://docs.microsoft.com/en-us/archive/msdn-magazine/2016/february/essential-net-configuration-in-net-core). Туда же можно будет вынести дополнительные настройки.

Положим ключ в файл конфигурации:

**{**

"ApiKey": "{ApiKey}"

**}**

Чтобы была возможность в будущем опубликовать в Nuget и переиспользовать ваши наработки, разделите код решения на два проекта: библиотеку классов *d03.Nasa,* отвечающую за логику получения данных и преобразования их в **DTO**, и консольное приложение *d03.Host,* которая будет использовать эти данные и заниматься их выводом.

| Проверьте себя после реализации: в проекте *d03.Nasa* не должно быть ссылок на библиотеки конфигурации приложения (**Microsoft.Extensions.Configuration** и **Microsoft.Extensions.Configuration.Json**). Проект *d03.Host* не должен использовать библиотеки **System.Net.Http** и **System.Text.Json**. Ответственность поделена. **Связность** проектов слабая. |
| --- |

В консольном приложении *d03.Host* нужны только конфигурация *appsettings.json* и точка входа - метод *Main* класса *Program.cs,* который будет подгружать конфигурацию, считывать ввод и отвечать запрошенной информацией. К примерам ввода и вывода мы вернемся в заданиях.

Библиотека *d03.Nasa* будет отвечать за связь с космосом и содержать реализации клиентов для каждого из заданий.

Пока остановимся чуть подробнее на **абстракциях** для них.

Чтобы воспользоваться **принципом инверсии зависимости** давайте выделим интерфейс *INasaClient.* Он будет представлять **контракт** для любого из клиентов и открывать доступ к методу получения данных - методу *GetAsync*. При этом входные и выходные параметры в этом методе для каждой из реализаций могут быть разными. Здесь нам помогут ключевые слова **in** и **out**, они позволят реализациям самим задать типы входных и выходных параметров, а интерфейс сделают **контравариантным** и **ковариантным** соответственно.

Получим интерфейс *INasaClient<in TIn, out TOut>* с методом *TOut GetAsync(TIn input).*

Также для избежания дублирования кода было бы полезно выделить общий функционал запросов в базовый класс *ApiClientBase.* Класс будет **абстрактным**, так как будет предоставлять наследникам только базовую реализацию и сам по себе **инстанцироваться** не будет. В него нужно будет выделить общее:

* Поле с ключом к API
* Конструктор, принимающий ключ API: *ApiClientBase(string apiKey)*
* Реализацию **HTTP GET** запроса и десериализацию ответа в нужный объект: *HttpGetAsync<T>(string url)*

Все это логично сделать **protected**, потому что понадобится только наследникам.

Обращаться к API мы будем с помощью **HTTP GET** запросов. В **REST** это тип **идемпотентных** запросов, которые отвечают за получение данных и не изменяют их (для изменения пригодились бы, например, **POST**, **PUT** или **DELETE** запросы). Для реализации HTTP-запросов вам понадобится **HttpClient.**

Если запрос был выполнен успешно и имеет [StatusCode = 200 (OK)](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes), получаем из **JSON** необходимые данные и [десериализуем](https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-customize-properties) их. Если статус ответа иной, результат запроса необходимо вернуть тому, кто метод вызвал, и в нем вывести в консоль.

Так как HTTP-запросы это **I/O операция**, для предотвращения блокировки программы на время выполнения запроса к удаленному ресурсу (серверу API), методы классов, содержащие эти запросы, должны быть асинхронными. В этом вам помогут ключевые слова **async/await**. К тому же, по принципу “[асинхронность от начала до конца](https://docs.microsoft.com/en-us/archive/msdn-magazine/2013/march/async-await-best-practices-in-asynchronous-programming)”, стоит избегать синхронного вызова асинхронного кода, так что метод *Main* (точка входа приложения) также должен быть асинхронным. Чтобы лучше понимать, что к чему, почитайте об [асинхронных завтраках](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/).

Обратите внимание, что асинхронные методы имеют **постфикс Async** в названии метода. Это является хорошим тоном при разработке библиотек.

Помните о [C# Coding Conventions](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/inside-a-program/coding-conventions)! Переменные в C# должны быть **PascalCasing** или **CamelCase**, для парсинга воспользуйтесь **аннотациями** **JsonPropertyName**.

Теперь, когда мы проделали всю подготовительную работу и поработали над абстракциями (а они, вы помните, не должны знать о реализациях), приступим к самим заданиям.

# Задание 00. Космос дня

“We are just an advanced breed of monkeys on a minor planet of a very average star. But we can understand the Universe. That makes us something very special.”

― **Stephen Hawking**

Рассмотрим “APOD(Astronomy Picture of the Day)” - ресурс, возвращающий коллекцию космических фотографий (или видео) дня, один из самых популярных ресурсов NASA. По их словам “It has the popular appeal of a Justin Bieber video”. Что же, давайте полюбуемся на них.

Найти документацию к API “APOD*”* можно перейдя на [Сайт API](https://api.nasa.gov/) -> Browse APIs -> APOD. Здесь можно прочитать о параметрах, ссылке для GET запроса и посмотреть пример возвращаемых данных.

Разберемся подробнее.

Ответственность за общение с API “APOD*”* должна быть выделена в отдельный клиент - *ApodClient*. Выше мы обсудили абстракции для него, давайтерассмотрим **наследование**.

Для *INasaClient* наш астероидный клиент должен реализовывать метод *GetAsync.* На входе нам будут нужно количество результатов (*ResultCount*), это будет наш **in** тип.

Метод будет возвращать коллекцию объектов с информацией о каждом медиа объекте *MediaOfToday*. Обратите внимание, что **async** методы возвращают **Task.**

**public class ApodClient : INasaClient<int, Task<MediaOfToday[]>>**

Наследование от *ApiClientBase* даст нам возможность переиспользовать реализации **базового конструктора** и метода *HttpGetAsync<T>*, а также обращаться к полю с ключом.

**public class ApodClient : ApiClientBase,**

**INasaClient<int, Task<MediaOfToday[]>>**

Осталось собрать все вместе и реализовать.

Структура необходимых данных:

{

**copyright**: "Ignacio Diaz Bobillo",

**date**: "2021-06-03",

**explanation**: "Globular star cluster Omega Centauri, also known as NGC 5139, is some 15,000 light-years away. The cluster is packed with about 10 million stars much older than the Sun within a volume about 150 light-years in diameter. It's the largest and brightest of 200 or so known globular clusters that roam the halo of our Milky Way galaxy. Though most star clusters consist of stars with the same age and composition, the enigmatic Omega Cen exhibits the presence of different stellar populations with a spread of ages and chemical abundances. In fact, Omega Cen may be the remnant core of a small galaxy merging with the Milky Way. Omega Centauri's red giant stars (with a yellowish hue) are easy to pick out in this sharp, color telescopic view.",

**title**: "Millions of Stars in Omega Centauri",

**url**: "<https://apod.nasa.gov/apod/image/2106/OmegaCent_LRGB_final1_1024.jpg>"

}

## Входные параметры

Количество результатов будем задавать из командной строки.

| ResultCount | int | Количество элементов для вывода |
| --- | --- | --- |

## Параметры конфигурации

Если вы делаете задание 01, конфигурация будет общей.

| ApiKey | string | Ключ к API |
| --- | --- | --- |

## Формат ответа

{mediaOfToday.Date}

‘{mediaOfToday.Title}’ by {mediaOfToday.Copyright}

{mediaOfToday.Explanation}

{mediaOfToday.Url}

{mediaOfToday.Date}

‘{mediaOfToday.Title}’

{mediaOfToday.Explanation}

{mediaOfToday.Url}

...

{mediaOfToday.Date}

‘{mediaOfToday.Title}’

{mediaOfToday.Explanation}

{mediaOfToday.Url}.

#### При вызове метода API произошла ошибка

GET “{apiUrl}” returned {StatusCode}:

{Content}

#### Примеры запуска приложения из папки проекта и вывода

**{**

**"ApiKey": "API\_KEY"**

**}**

$dotnet run apod 5

19/01/2018

'Clouds in the LMC' by Josep Drudis

An alluring sight in southern skies, the Large Magellanic Cloud (LMC) is seen in this deep and detailed telescopic mosaic. Recorded with broadband and narrowband filters, the scene spans some 5 degrees or 10 full moons. The narrowba

nd filters are designed to transmit only light emitted by hydrogen, and oxygen atoms. Ionized by energetic starlight, the atoms emit their characteristic light as electrons are recaptured and the atoms transition to a lower energy s

tate. As a result, in this image the LMC seems covered with its own clouds of ionized gas surrounding its massive, young stars. Sculpted by the strong stellar winds and ultraviolet radiation, the glowing clouds, dominated by emissio

n from hydrogen, are known as H II (ionized hydrogen) regions. Itself composed of many overlapping H II regions, the Tarantula Nebula is the large star forming region at the left. The largest satellite of our Milky Way Galaxy, the L

MC is about 15,000 light-years across and lies a mere 160,000 light-years away toward the constellation Dorado.

https://apod.nasa.gov/apod/image/1801/LMC\_RGBNB-Don-Josep5-Cc1024.jpg

04/04/2008

'Layers in Aureum Chaos'

At first glance these undulating shapes in shades of blue might look like waves on an ocean. Seen here in a false-color image from the Mars Reconnaissance Orbiter's HiRISE camera, they are actually layered rock outcrops found in Aur

eum Chaos. The larger Aureum Chaos region is a chaotic jumble of eroded terrain in the eastern part of Mars' immense canyon Valles Marineris. Distinct layers composing these outcrops could have been laid down by dust or volcanic ash

settling from the atmosphere, sand carried by martian winds, or sediments deposited on the floor of an ancient lake. This close-up view of the otherwise red planet spans about 4 kilometers, a distance you might walk over flat groun

d in less than an hour. digg\_url = 'http://apod.nasa.gov/apod/ap080404.html'; digg\_skin = 'compact';

https://apod.nasa.gov/apod/image/0804/PSP\_007006\_1765\_e800.jpg

31/10/2010

'Halloween and the Ghost Head Nebula'

Halloween's origin is ancient and astronomical. Since the fifth century BC, Halloween has been celebrated as a cross-quarter day, a day halfway between an equinox (equal day / equal night) and a solstice (minimum day / maximum nigh

t in the northern hemisphere). With a modern calendar, however, the real cross-quarter day will occur next week. Another cross-quarter day is Groundhog's Day. Halloween's modern celebration retains historic roots in dressing to sc

are away the spirits of the dead. Perhaps a fitting tribute to this ancient holiday is this view of the Ghost Head Nebula taken with the Hubble Space Telescope. Similar to the icon of a fictional ghost, NGC 2080 is actually a star

forming region in the Large Magellanic Cloud, a satellite galaxy of our own Milky Way Galaxy. The Ghost Head Nebula spans about 50 light-years and is shown in representative colors.

https://apod.nasa.gov/apod/image/1010/ngc2080\_hst.jpg

15/01/1997

'Black Holes Signature From Advective Disks

Research Credit:'

star. perhaps brighter than allowable from an ADAF onto a neutronservationsws (ADAFs).

https://apod.nasa.gov/apod/image/9701/xraybin\_heasarc.gif

13/02/2007

'Vela Supernova Remnant in Visible Light'

The explosion is over but the consequences continue. About eleven thousand years ago a star in the constellation of Vela could be seen to explode, creating a strange point of light briefly visible to humans living near the beginnin

g of recorded history. The outer layers of the star crashed into the interstellar medium, driving a shock wave that is still visible today. A roughly spherical, expanding shock wave is visible in X-rays. The above image captures m

uch of that filamentary and gigantic shock in visible light, spanning almost 100 light years and appearing twenty times the diameter of the full moon. As gas flies away from the detonated star, it decays and reacts with the interste

llar medium, producing light in many different colors and energy bands. Remaining at the center of the Vela Supernova Remnant is a pulsar, a star as dense as nuclear matter that completely rotates more than ten times in a single sec

ond.

https://apod.nasa.gov/apod/image/0702/vela\_skyfactory.jpg

**{**

**"ApiKey": ""**

**}**

$ dotnet run apod 5

GET “https://api.nasa.gov/neo/rest/v1/feed?start\_date=2015-09-07&end\_date=2015-09-08&api\_key=” returned Forbidden:

{

"error": {

"code": "API\_KEY\_MISSING",

"message": "No api\_key was supplied. Get one at https://api.nasa.gov:443"

}

}

# Задание 01. Трекер астероидов

“It is most gratifying," it said, "that your enthusiasm for our planet continues unabated, and so we would like to assure you that the guided missiles currently converging with your ship are part of a special service we extend to all of our most enthusiastic clients, and the fully armed nuclear warheads are of course merely a courtesy detail. We look forward to your custom in future lives ... thank you.”

― **Douglas Adams,** [**The Hitchhiker's Guide to the Galaxy**](https://www.goodreads.com/work/quotes/3078186)

У NASA есть “Asteroids NeoWs*”* - группа методов, которые возвращают список объектов и их приближение к Земле. Давайте реализуем собственный трекер астероидов. Будем обращаться напрямую к NASA и учитывать астероиды, которые приближались ближе всего к Земле за указанный временной период (в порядке увеличения расстояния).

Итак, в API “*Asteroids NeoWs”* ([Сайт API](https://api.nasa.gov/) -> Browse APIs -> Asteroids NeoWs) нас интересуют два метода: метод “Neo Feed”, позволяющий получить список космических объектов, приближавшихся к Земле в рамках заданных дат, и метод “Neo Lookup”, предоставляющий подробную информацию о запрошенном объекте.

План прост. Сначала делаем запрос к “Neo Feed”, указав необходимые нам даты (*StartDate* и *EndDate*). Полученный список сортируем по расстоянию от Земли и берем из него только необходимое нам количество элементов (*ResultCount*) с наименьшим расстоянием. Далее, когда список сформирован (чтобы не запрашивать лишнее), для каждого из элементов можно запросить подробные данные в “Neo Lookup”. Это и будет ответом.

Разберем подробнее.

Ответственность за общение с API “Asteroids NeoWs*”* должна быть выделена в отдельный клиент - *NeoWsClient*. Выше мы обсудили абстракции для него, давайтерассмотрим **наследование**.

Для *INasaClient* наш астероидный клиент должен реализовывать метод *GetAsync.* На входе нам будут нужны: настройки временного периода (*StartDate* и *EndDate*) и количество результатов (*ResultCount*). Объединим их в класс или запись *AsteroidRequest*, это будет наш **in** тип.

Метод будет возвращать коллекцию объектов с информацией о каждом астероиде *AsteroidLookup*. Обратите внимание, что **async** методы возвращают **Task.**

public class NeoWsClient: INasaClient**<**AsteroidRequest, Task**<**AsteroidLookup**[]>>**

Наследование от *ApiClientBase* даст нам возможность переиспользовать реализации **базового конструктора** и метода *HttpGetAsync<T>*, а также обращаться к полю с ключом.

public class NeoWsClient: ApiClientBase,

INasaClient**<**AsteroidRequest, Task**<**AsteroidLookup**[]>>**

Осталось собрать все вместе и реализовать.

Структура необходимых данных из Neo Feed:

**near\_earth\_objects:**

**{**

**2015-09-08**: [

{

**id**: "2465633",

**close\_approach\_data**: [

{

**miss\_distance**:

{

**kilometers**: "45290438.204452618"

}

}

]

},

…

{

**id**: "2465633",

**close\_approach\_data**: [

{

**miss\_distance**:

{

**kilometers**: "45290438.204452618"

}

}

]

}

],

…

**2015-09-07:** [

{

**id**: "2465633",

**close\_approach\_data**: [

{

**miss\_distance**:

{

**kilometers**: "45290438.204452618"

}

}

]

}

* ]

}

Обратите внимание, коллекцию near\_earth\_objects можно распарсить в Dictionary<DateTime, *AsteroidInfo*>. Расстояние до Земли, по которому будет необходима сортировка, определяется в close\_approach\_data->miss\_distance->kilometers.

Для фильтрации коллекций, выборки данных, выборки необходимого их количества не используйте циклы! Вам помогут расширения **LINQ**: **Select**/**SelectMany**, **Where**, **OrderBy**/**OrderByDescending**, **First**/**Last/Single**, **FirstOrDefault**/**LastOrDefault/SingleOrDefault**, **Take**/**Skip**.

Если метод Neo Feed вернул меньше объектов, чем указано в *ResultCount*, то выбираем все, что вернулось.

Структура необходимых данных из Neo Lookup:

{

**neo\_reference\_id**: "3542519",

**name**: "(2010 PK9)",

**nasa\_jpl\_url**: "<http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3542519>",

**is\_potentially\_hazardous\_asteroid**: true,

**orbital\_data**:

{

**orbit\_class**:

{

**orbit\_class\_type**: "ATE",

**orbit\_class\_description**: "Near-Earth asteroid orbits similar to that of 2062 Aten"

}

}

}

## Входные параметры

Количество результатов будем задавать из командной строки.

| ResultCount | int | Количество элементов для вывода |
| --- | --- | --- |

## Параметры конфигурации

Даты давайте вынесем в конфигурацию. Конфигурация будет общей с Заданием 00.

| StartDate | DateTime | Дата старта срока выборки |
| --- | --- | --- |
| EndDate | DateTime | Дата окончания срока выборки |
| ApiKey | string | Ключ к API |

## Формат ответа

- Asteroid {asteroid.Name}, SPK-ID: {asteroid.Id}

IS POTENTIALLY HAZARDOUS!

Classification: {asteroid.OrbitalData.OrbitClass.OrbitClassType}, {asteroid.OrbitalData.OrbitClass.OrbitClassDescription}.

Url: {asteroid.NasaUrl}.

...

- Asteroid {asteroid.Name}, SPK-ID: {asteroid.Id}

Classification: {asteroid.OrbitalData.OrbitClass.OrbitClassType}, {asteroid.OrbitalData.OrbitClass.OrbitClassDescription}.

Url: {asteroid.NasaUrl}.

#### При вызове метода API произошла ошибка

GET “{apiUrl}” returned {StatusCode}:

{Content}

#### Примеры запуска приложения из папки проекта и вывода

**{**

**"ApiKey": "API\_KEY",**

**"NeoWs": {**

**"StartDate": "2015-09-07",**

**"EndDate": "2015-09-08"**

**}**

**}**

$ dotnet run neows 5

- Asteroid (2015 RG2), SPK-ID: 3726788

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3726788.

- Asteroid (2015 RC), SPK-ID: 3726710

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3726710.

- Asteroid (2015 RO36), SPK-ID: 3727181

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727181.

- Asteroid (2015 RL35), SPK-ID: 3727036

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727036.

- Asteroid (2015 RY83), SPK-ID: 3727663

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727663.

**{**

**"ApiKey": "",**

**"NeoWs": {**

**"StartDate": "2015-09-07",**

**"EndDate": "2015-09-08"**

**}**

**}**

$ dotnet run neows 5

GET “https://api.nasa.gov/neo/rest/v1/feed?start\_date=2015-09-07&end\_date=2015-09-08&api\_key=” returned Forbidden:

{

"error": {

"code": "API\_KEY\_MISSING",

"message": "No api\_key was supplied. Get one at https://api.nasa.gov:443"

}

}

**{**

**"ApiKey": "API\_KEY",**

**"NeoWs": {**

**"StartDate": "2015-09-07",**

**"EndDate": "2015-09-08"**

**}**

**}**

$ dotnet run neows

- Asteroid (2015 RG2), SPK-ID: 3726788

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3726788.

- Asteroid (2015 RC), SPK-ID: 3726710

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3726710.

- Asteroid (2015 RO36), SPK-ID: 3727181

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727181.

- Asteroid (2015 RL35), SPK-ID: 3727036

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727036.

- Asteroid (2015 RY83), SPK-ID: 3727663

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727663.

- Asteroid (2015 TX237), SPK-ID: 3730577

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3730577.

- Asteroid (2015 RH36), SPK-ID: 3727179

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727179.

- Asteroid (2015 UG), SPK-ID: 3731587

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3731587.

- Asteroid (2016 RN41), SPK-ID: 3759690

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3759690.

- Asteroid (2015 RN83), SPK-ID: 3727639

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727639.

- Asteroid (2016 RT), SPK-ID: 3758838

Classification: ATE, Near-Earth asteroid orbits similar to that of 2062 Aten.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3758838.

- Asteroid (2016 RU33), SPK-ID: 3759353

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3759353.

- Asteroid (2008 QV11), SPK-ID: 3426410

Classification: ATE, Near-Earth asteroid orbits similar to that of 2062 Aten.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3426410.

- Asteroid (2016 EK158), SPK-ID: 3747356

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3747356.

- Asteroid (2015 RX83), SPK-ID: 3727662

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3727662.

- Asteroid 465633 (2009 JR5), SPK-ID: 2465633IS POTENTIALLY HAZARDOUS!

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2465633.

- Asteroid (2015 FC35), SPK-ID: 3713989

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3713989.

- Asteroid (2019 QK4), SPK-ID: 3843641

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3843641.

- Asteroid (2020 BY), SPK-ID: 3986741

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3986741.

- Asteroid (2018 RZ2), SPK-ID: 3827337

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3827337.

- Asteroid (2020 WZ), SPK-ID: 54088823

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=54088823.

- Asteroid (2010 XT10), SPK-ID: 3553060

Classification: APO, Near-Earth asteroid orbits which cross the Earth's orbit similar to that of 1862 Apollo.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=3553060.

- Asteroid 440012 (2002 LE27), SPK-ID: 2440012

Classification: AMO, Near-Earth asteroid orbits similar to that of 1221 Amor.

Url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2440012.