Day 02. Generics, LINQ

## 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.
* С основными возможностями языка C# можно ознакомиться в [официальной спецификации](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/language-specification/introduction).
* 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 изучение пригодится вам как в выполнении текущего задания, так и в вашей дальнейшей карьере .NET разработчика.
* Have fun :)

## Требования к заданиям дня

* Все сборки должны быть в одном решении.
* Каждому из заданий должно соответствовать отдельное консольное приложение, созданное на основе стандартного шаблона .NET SDK.
* Используйте **top-level-statements** и **var**.
* Название проекта (и его отдельного каталога) должно выглядеть как d{*xx*}\_ex{*yy*}, где *xx* - цифры текущего дня, *yy* - цифры текущего задания.
* Название решения (и его отдельного каталога) - d{*xx*}, где *xx* - цифры текущего дня.
* Для форматирования выходных данных используйте **культуру** en-GB: N2 для вывода денежных сумм, d для дат.
* Файлы с данными в заданиях считать верными и не нуждающимися в валидации. Файлы можно найти в соответствующих номеру задания папках example files.

# Задание 00. А теперь отдыхаем

## Рекомендованные языковые конструкции

* Интерфейсы
* Классы
* Коллекции и списки
* Циклы
* LINQ

## Структура проекта

d02\_ex00

Program.cs

Model

ISearchable.cs

Book.cs

Movie.cs

## Задание

Жизнь на работе и учебе начинают налаживаться, но отдыхать тоже важно. И как было бы здорово, думаете вы, если бы определиться с этим было проще. Открыл готовый список и выбрал. Здесь вам на помощь приходит архив New York Times - издание постоянно публикует списки бестселлеров и обзоры на лучшие фильмы, почему бы не воспользоваться? Остается всего ничего - скомпилировать это в единый каталог увеселений.

Итак, у вас есть выгрузка книг-бестселлеров этой недели, взятых из разных категорий. Считаем, что файл периодически обновляется и содержит актуальную информацию. Из приведенной в списке информации нас интересуют: название книги (*Title*), ее автор (*Author*), описание (*SummaryShort*), место (*Rank*) в рейтинге и его название (*ListName*), а также ссылка на страницу в магазине (*Url*).

Для кино-настроения рядом есть выгрузка свежих ревью на кино. Считаем, что и этот файл обновляется и содержит актуальную информацию. Из приведенных в этом списке данных нам нужны: название фильма (*Title*), его рейтинг (*Rating*), является ли фильм выбором критиков (*IsCriticsPick*), саммари его ревью (*SummaryShort*) и ссылка на страницу публикации (*Url*).

Список книг должен подгружаться из файла book\_reviews.json, список фильмов - из movies.json, их нужно **десериализовать** из **JSON** в [соответствующие сущности](https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-customize-properties). При этом было бы здорово, думаете вы, реализовать поиск в одном методе - искать-то придется по заголовку и там, и там, а дублировать код - [дурной тон](https://x-team.com/blog/principles-clean-code/). И это значит, что здесь пригодится **принцип подстановки Лисков**.

Итак, вам необходимо разработать консольное приложение, которое при запуске будет собирать информацию из двух файлов **JSON** в две разных коллекции с элементами. Элементы должны реализовать один интерфейс *ISearchable*: и у книг, и у фильмов есть заголовки, по которым будет вестись поиск, а также возможность настроить верный формат вывода с помощью **перегрузки метода** ToString(). Далее по введенной строке поискового запроса нужно выводить список всех найденных элементов: книг и отзывов на фильмы.

Поиск не должен зависеть от регистра.

Непосредственно поиск должен осуществляться в одном методе, для этого важно использовать абстракции. Помните о **принципе единственной ответственности**: метод поиска должен отвечать только непосредственно за поиск элементов и фильтровать коллекцию *ISearchable.*

Результат поиска должен быть сгруппирован по типу медиа (кино/книга). Не используйте для поиска циклы, посмотрите в сторону **LINQ**! Обратите внимание на **методы расширения LINQ**, они делают код куда более читаемым и простым для восприятия.

В случае отсутствия соответствующих поиску элементов приложение должно выдавать сообщение “There are no “{search}” in media today.”

В случае пустой строки поиска приложение должно выдавать все элементы.

## Входные параметры

> Input search text:

| search | string | Строка поиска |
| --- | --- | --- |

## Формат ответа

Items found: {itemCount}

Book search result [bookCount]:

{book.Title} by {book.Author} [{book.Rank} on NYT’s {book.ListName}]

{book.SummaryShort}

{book.Url}

...

Movie search result [movieCount]:

{movie.Title} {movie.IsCriticsPick ? “[NYT critic’s pick]” : “”}

{movie.SummaryShort}

{movie.Url}

...

$ dotnet run

> Input search text:

ar

Items found: 6

Book search result [3]:

- THE MIDNIGHT LIBRARY by Matt Haig [6 on NYT's Hardcover Fiction]

Nora Seed finds a library beyond the edge of the universe that contains books with multiple possibilities of the lives one could have lived.

https://www.amazon.com/dp/0525559477?tag=NYTBSREV-20

- THE INVISIBLE LIFE OF ADDIE LARUE by V.E. Schwab [9 on NYT's Hardcover Fiction]

A Faustian bargain comes with a curse that affects the adventure Addie LaRue has across centuries.

https://www.amazon.com/dp/0765387565?tag=NYTBSREV-20

- KLARA AND THE SUN by Kazuo Ishiguro [13 on NYT's Hardcover Fiction]

An "Artificial Friend" named Klara is purchased to serve as a companion to an ailing 14-year-old girl.

https://www.amazon.com/dp/059331817X?tag=NYTBSREV-20

Movie search result [3]:

- IN OUR MOTHERS' GARDENS

The Netflix documentary sets out to show how maternal lineages have shaped generations of Black women.

https://www.nytimes.com/2021/05/06/movies/in-our-mothers-gardens-review.html

- MARIGHELLA [NYT critic's pick]

Wagner Moura's provocative feature debut chronicles the armed struggle led by Carlos Marighella against Brazil's military dictatorship in the 1960s.

https://www.nytimes.com/2021/04/29/movies/marighella-review.html

- THINGS HEARD & SEEN

Amanda Seyfried and James Norton move into a haunted house in this busy, creaky Netflix thriller.

https://www.nytimes.com/2021/04/29/movies/things-heard-and-seen-review.html

# Задание 01. Быстрее, выше, YAML

## Рекомендованные языковые конструкции

* Классы
* Наследование
* Полиморфизм
* Интерфейсы
* Коллекции и списки

## Структура проекта

d02\_ex01

Program.cs

Configuration

Configuration.cs

Sources

IConfigurationSource.cs

YamlSource.cs

JsonSource.cs

## Задание

Ваши приложения становятся все изящнее и больше. Вы понимаете, что дело подходит к распространению вашего ПО, а это значит, чувствительные данные и настройки, зависящие от окружений - строку подключения к базе данных, различные флаги, адрес сервера, константы, - хорошо бы вынести из кода и хранить в отдельной конфигурации приложения. Отличным вариантом является чтение этих параметров из файлов **JSON** или **YAML**.

Итак, у вас есть два приложенных к заданию тестовых файла: *config.json* и *config.yml*. Вам требуется написать приложение, которое будет подтягивать свои настройки из этих файлов.

Для обработки и хранения параметров вы решаете написать свой класс *Configuration*. Он будет содержать коллекцию *Params* - необходимый для работы приложения набор параметров, словарь ключ-значение. Ключи должен быть уникальными, для этого неплохо подойдут **хеш-таблицы**.

Вспомнив о **принципе единственной ответственности**, вы понимаете, что класс *Configuration* не должен ничего знать об источнике(ах) данных, JSON это, YAML или что-то еще. Правильнее будет, если для создания *Configuration* нужно будет передать список разных источников, откуда он сам соберет нужную коллекцию. За это он и будет отвечать.

Для источников данных вы решаете вынести общий интерфейс *IConfigurationSource*, реализациями для него будут: *JsonSource* для JSON и *YamlSource* для YAML. Так будет соблюден **принцип инверсии зависимостей**: класс *Configuration* будет зависеть от абстракции *IConfigurationSource*, а не от конкретных реализаций данного интерфейса. В дальнейшем, если вы решите добавить новые источники, вам не придется переписывать конфигурацию.

Также нужно помнить, что класс, который отвечает за чтение из одного источника данных, не должен знать о других источниках данных.

В чем работа источника данных, элемента *IConfigurationSource*? Он ответственен за загрузку данных из файла. Значит в нем должен быть путь к файлу и метод, который считывает текст из файла и десериализует его в коллекцию параметров. Разница здесь будет в реализации: JSON и YAML файлы имеют разный формат и разные правила парсинга.

Для превращения JSON файла в **Hashtable** подойдут встроенные средства **System.Text.Json**. Для того, чтобы получить **Hashtable** из YAML-файла встроенных средств пока нет, но это не беда. Дело в том, что любая разработка, так или иначе, командная работа, и для обмена полезными наработками существует инструмент **nuget**. Среди множества полезных пакетов нам пригодится YamlDotNet.

Здесь считаем, что структура параметров в файлах плоская, и вложенности данных нет.

Обратите внимание, что десериализация файлов с неподходящим форматом приведет к **исключению** в работе приложения. Это нормальная ситуация, если исключение обработать. В этом случае нужно вывести ошибку о некорректных данных.

Так класс *Configuration* будет создаваться, собирая данные из разных источников. Однако финальная коллекция параметров, которая будет являться актуальной для приложения, должна быть одна. Поэтому вы должны иметь возможность слияния одноименных параметров из разных источников конфигураций. А это значит, что каждый из источников должен иметь параметр *Priority*, чтобы *Configuration* мог определить в каком порядке будет загружать и мерджить данные.

Осталось всего ничего. Для проверки корректности конфигурации приложение должно вывести ее в терминал.

#### Дополнительно

Порой бывает полезно сделать еще один шаг и вспомнить методологию [The twelve-factor](https://12factor.net/config). Многие приложения позволяют читать параметры не только из файлов, но из **переменных окружения**.

В задании этого не требуется, но учтите на будущее!

## Входные параметры

$ dotnet run “{jsonPath}” {jsonPriority} “{yamlPath}” {yamlPriority}

Приоритет попадания

| jsonPath | string | Путь к JSON конфигурации |
| --- | --- | --- |
| jsonPriority | int | Приоритет загрузки параметров из файла JSON |
| yamlPath | string | Путь к YAML конфигурации |
| yamlPriority | int | Приоритет загрузки параметров из файла YAML |

## Формат ответа на выходе

Configuration

{Ключ} : {Значение}

{Ключ} : {Значение}

...

{Ключ} : {Значение}

#### Пользователь указал некорректные данные

Invalid data. Check your input and try again.

## Пример запуска приложения из папки проекта и вывода

$ dotnet run “pathToJson” 1 “pathToYaml” 2

Configuration

Application: ex01

Port: 8080

Source: YAML

CheckForUpdates: false

Domain: <http://localhost>

$ dotnet run “pathToJson” 1 “pathToYaml” 0

Application: ex01

CheckForUpdates: True

Port: 1234

Source: JSON

Domain: <http://localhost>