Day 02.1. Быстрее, выше, YAML

## 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 решения и его проекта (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.

## Что нового

* YAML, JSON
* Хеш-таблицы
* Обработка исключений
* Принцип единственной ответственности
* Принцип инверсии зависимостей
* nuget-пакеты

## Интро

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.

## Структура проекта

d03/

Program.cs

Configuration/

Configuration.cs

Sources/

IConfigurationSource.cs

YamlSource.cs

JsonSource.cs

# Задание 00. Configuration

Реализуйте класс *Configuration* для обработки и хранения параметров. Он должен содержать коллекцию *Params* - необходимый для работы приложения набор параметров, словарь ключ-значение. Ключи должны быть уникальными, для этого неплохо подойдут **хеш-таблицы**.

Помните о **принципе единственной ответственности**: класс *Configuration* не должен ничего знать об источнике(ах) данных, JSON это, YAML или что-то еще. Поэтому выделите общий интерфейс *IConfigurationSource* для реализации разных источников: именно он будет ответственен за загрузку данных из файла. *IConfigurationSource* должен иметь метод, возвращающий коллекцию параметров.

Реализуйте конструктор класса *Configuration* так, чтобы он принимал коллекцию *IConfigurationSource* и хеш-таблица *Params* заполнялась с помощью них.

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.

# Задание 01. JSON

Реализуйте класс *JsonSource,* наследник *IConfigurationSource.* Единственным конструктором класса должен быть конструктор принимающий путь к json файлу, из которого предстоит загрузить конфигурацию. Для превращения **JSON** файла в **Hashtable** используйте встроенные средства **System.Text.Json**.

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

Создайте экземпляр класса *Configuration* с помощью *JsonSource,* заполненного из файла *config.json.* Выведите в консоль конфигурацию в следующем формате:

Configuration

{Ключ}: {Значение}

{Ключ}: {Значение}

...

{Ключ}: {Значение}

## Входные параметры

$ dotnet run “{filePath}”

## Пример запуска приложения из папки проекта и вывода

Configuration

Port: 1234

CheckForUpdates: True

Domain: http://localhost

Source: JSON

# Задание 02. YAML

Реализуйте класс *YamlSource,* наследник *IConfigurationSource.* Единственным конструктором класса должен быть конструктор принимающий путь к yaml файлу, из которого предстоит загрузить конфигурацию. 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**. подключите и используйте пакет YamlDotNet.

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

Создайте экземпляр класса *Configuration* с помощью *YamlSource,* заполненного из файла *config.yml.* Выведите в консоль конфигурацию в следующем формате:

Configuration

{Ключ}: {Значение}

{Ключ}: {Значение}

...

{Ключ}: {Значение}

## Входные параметры

$ dotnet run “{filePath}”

## Пример запуска приложения из папки проекта и вывода

Configuration

CheckForUpdates: false

Port: 8080

Source: YAML

Application: ex03

# Задание 03. Исключения

**Десериализация** файлов с неподходящим форматом может привести к **исключениям** в работе приложения. Это нормальная ситуация, если исключение [обработать](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions).

Допишите вызов классов обработки конфигурации так, чтобы отлавливать исключения, возникающие при десериализации. В этом случае нужно вывести ошибку о некорректных данных:

Invalid data. Check your input and try again.

# Задание 04. Приоритеты

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.

Добавьте в интерфейс *IConfigurationSource* свойство, отображающее приоритет источника конфигурации. Свойство должно быть приватным для редактирования и заполняемым из конструктора класса.

Допишите *Configuration* так, чтобы приоритет учитывался при загрузке из разных источников конфигураций.

Создайте экземпляр класса *Configuration* с конфигурацией из двух источников: *YamlSource* и *JsonSource.* Путь к файлам и их приоритет задайте из консоли при запуске приложения.

Выведите в консоль конфигурацию в следующем формате:

Configuration

{Ключ}: {Значение}

{Ключ}: {Значение}

...

{Ключ}: {Значение}

## Входные параметры

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

## Пример запуска приложения из папки проекта и вывода

$ dotnet run “pathToJson” 1 “pathToYaml” 2

Configuration

Source: YAML

Application: ex03

CheckForUpdates: false

Domain: http://localhost

Port: 8080

$ dotnet run “pathToJson” 1 “pathToYaml” 0

Configuration

CheckForUpdates: True

Application: ex03

Port: 1234

Source: JSON

Domain: http://localhost

# 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! Попробуйте добавить в приложение новый источник EnvSource и реализовать чтение параметров из него.