Day 01. ООП

## 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. 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 разрешенных 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 изучение пригодится вам как в выполнении текущего задания, так и в вашей дальнейшей карьере .NET разработчика.
* Have fun :)

## Требования к заданиям дня

* Нельзя использовать nuget-пакеты.
* 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-US: N2 для вывода денежных сумм, d для дат.
* Файлы с данными в заданиях считать верными и не нуждающимися в валидации.

# Задание 00. Конвертер валют

## Разрешенные языковые конструкции

* Структуры
* Классы
* Списки
* Перегрузка методов
* Циклы
* Интерполяция строк
* System.IO

## Структура проекта

d01\_ex00

Program.cs

Exchanger.cs

Models

ExchangeRate.cs

ExchangeSum.cs

## Задание

Вы уже более опытный программист, и вместе с опытом растет сложность ваших задач. Вы уже руководствуетесь принципами **SOLID** в разработке и разбиваете ваш код на **классы** и **структуры**.

Вашим очередным вызовом становится реализация конвертера валют для пар EUR-RUB, USD-RUB, EUR-USD, USD-EUR, RUB-USD, RUB-EUR. На входе программа будет запрашивать сумму в одной из разрешенных валют (EUR, RUB, USD). В ответ выводить таблицу, в которой содержатся сконвертированные в другие разрешенные валюты суммы.

Для удобства определим, что у каждой валюты есть свой уникальный код-идентификатор (EUR, RUB, USD). Для хранения данных хорошо подойдут **структуры**: тогда для суммы в валютеможно определить *ExchangeSum* (сумма, идентификатор), а для информации об обмене - *ExchangeRate* (валюта “из”, валюта “в”, коэффициент обмена).

Используйте **принцип единственной ответственности**! Согласно ему структуры не должны знать о других валютах и курсах обмена. Но могут самостоятельно отвечать за парсинг данных из текста или формат данных вывода при приведении в string. Вам может помочь **перегрузка** метода ToString().

Информацию о курсах обмена предоставляет обменник/биржа. В приложенном к заданию архиве вы найдете файлы с коэффициентами для конвертации. Считаем, что формат данных в файле строго задан. Предполагается, что файлы будут обновляться раз в день, поэтому путь к папке будет одним из входных аргументов.

Функционал обменника нужно выделить в отдельный класс *Exchanger*. Он будет содержать коллекцию курсов обмена, отвечать за загрузку их из файлов и за конвертацию.

Помните, *Exchanger* не должен знать, откуда его вызывают и для чего используют результат - не нужно взваливать на него ответственность за вывод в терминал. Если метод конвертации будет возвращать список возможных сумм в валюте, его использование будет куда более гибким.

Обратите внимание на оператор **yield**, он может пригодиться.

### Входные параметры

|  |  |  |
| --- | --- | --- |
| sum | string | Сумма с указанием валюты |
| ratesDirectory | string | Путь к папке с файлами с коэффициентами для конвертации |

### Формат ответа на выходе

Сумма в исходной валюте: 100.00 RUB

Сумма в USD: 1.36 USD

Сумма в EUR: 1.11 EUR

Сумма в исходной валюте: 100.00 EUR

Сумма в USD: 81.97 USD

Сумма в RUB: 8,990.38 RUB

#### Пользователь указал некорректные данные

Ошибка ввода. Проверьте входные данные и повторите запрос.

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

$ dotnet run “100 RUB” “path/to/folder”

Сумма в исходной валюте: 100.00 RUB

Сумма в USD: 1.36 USD

Сумма в EUR: 1.11 EUR

# Задание 01

## Разрешенные языковые конструкции

* Классы
* Наследование
* Полиморфизм
* Списки
* Enum
* DateTime
* Record
* Циклы
* Приведение данных
* Переопределение методов
* Интерполяция строк

## Структура проекта

d01\_ex01

Program.cs

Tasks

Task.cs

TaskType.cs

TaskPriority.cs

TaskState.cs

Events

Event.cs

CreatedEvent.cs

TaskDoneEvent.cs

TaskWontDoEvent.cs

## Задание

Вы все больше пропадаете на работе. Семья, котик и любимый суккулент стали забывать, как вы выглядите. Чтобы как-то соблюсти баланс между работой, учебой и личной жизнью, вы решаете создать для себя персональный трекер задач.

Сначала нужно решить, что такое задача. Так мы сможем спроектировать наше приложение, основываясь на предметной области. Итак, задача - какая-либо проблемная ситуация или цель.

Чтобы знать содержание цели, вам потребуется ее описать. Чтобы найти ее среди прочих, задать заголовок. Чтобы достигать целей, необходимо ставить сроки. Итак, у задачи должны быть: *заголовок* (текст), *описание* (текст), *срок выполнения* (дата).

Помимо этого, в задачах будет куда удобнее ориентироваться, если им можно будет задать *приоритет* (*Низкий, Средний, Высокий*) и *категорию* (*Работа, Учеба, Личное*). Здесь нам помогут **enum**.

Статус задачи может меняться: она создается *Новой*, далее по завершении всех работ становится *Завершенной*, кроме того задачи, работу над которыми больше не имеет смысла продолжать, можно отметить статусом *Неактуальная* и больше к ним не возвращаться. Задача может находиться только в одном статусе и переходить между ними в определенном порядке (*Завершенная* не может стать *Неактуальной*), звучит как **конечный автомат**.

Для управления статусами задачи будем использовать подход, который называется [**Event-Sourcing**](https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing). Представим, что статус задачи определяется не отдельным свойством *Статус*, а агрегируется из событий - хранимой в каждой задаче истории изменений статусов. События **иммутабельные** - если что-то изменилось, это новое событие, - для их хранения отлично подойдет тип **Record**. Получается, для каждого изменения статуса задачи необходимо создавать и записывать в историю новое событие.

Для разных ситуаций лучше определить разные события: они могут наследоваться от одного абстрактного родителя и хранить в себе тип статуса.

Вам необходимо реализовать программу, которая позволяет составить список задач c возможностью отобразить список, добавить новую или отметить задачу выполненной или неактуальной.

Задача должна иметь историю изменений. Свойства задачи должны быть закрытыми для изменения извне. Для изменения статусов задача должна предоставлять специальные методы, которые **инкапсулируют** логику создания событий и вычисления нового статуса.

### Метод 1. Создание задачи

> add

> Введите заголовок

> {title}

> Введите описание

> {summary}

> Введите срок

> {dueDate}

> Введите тип

> {type}

> Установите приоритет

> {priority}

Обратите внимание, статус задачи не задается вводом: задача всегда должна создаваться в статусе *Новая*.

#### Входные параметры

|  |  |  |
| --- | --- | --- |
| title | string  required | Заголовок задачи |
| summary | string  not required | Описание того, что нужно сделать |
| dueDate | datetime (nullable)  not required | К какой дате задача планируется быть выполненной |
| type | enum [Work, Study, Personal]  required | Тип задачи |
| priority | enum [Low, Normal, High]  not required (default value: Normal) | Приоритет задачи |

Обратите внимание на обязательность входных аргументов!

Если параметр обязательный, нужно проверять его на входе. Если не обязательный, в классе соответствующее ему поле должно быть **nullable**. При этом string - **ссылочный тип**, и уже по умолчанию nullable. DateTime и enum - **типы значений**, и nullable им нужно явно указать.

#### Формат ответа на выходе

{task.Title}

[{task.Type}] [{task.State}]

Priority: {task.Priority}, Due till {task.DueDate}

{task.Summary}

#### Формат ответа в случае пустой даты

{task.Title}

[{task.Type}] [{task.State}]

Priority: {task.Priority}

{task.Summary}

#### Пример ответа

- Полить цветы

[Personal] [New]

Priority: High, Due till 11/21/2021

Полить цветы на кухне и в гостиной. Не забыть про любимый суккулент!

#### Пользователь указал некорректные данные

Ошибка ввода. Проверьте входные данные и повторите запрос.

### Метод 2. Список задач

> list

Для того, чтобы задать формат отображения всех задач в одном месте, можно переопределить метод ToString() задачи. Если список пуст, нужно вывести сообщение об этом.

Сортировать задачи не нужно.

#### Формат ответа на выходе

- {task.Title}

[{task.Type}] [{task.State}]

Priority: {task.Priority}, Due till {task.DueDate}

{task.Summary}

#### Пустой список

Список задач пока пуст.

#### Пример ответа

- Полить цветы

[Personal] [New]

Priority: Normal, Due till 11/5/2021

Полить цветы на кухне и в гостиной. Не забыть про любимый суккулент!

- Закончить Day 1

[Study] [Done]

Priority: High, Due till 11/21/2021

Закончить все упражнения первого дня бассейна по .NET, залить все в репозиторий.

### Метод 3. Выполнение задачи

> done

> Введите заголовок

> {title}

Здесь обращаемся к конкретной задаче по заголовку, считая его для простоты уникальным идентификатором.

#### Входные параметры

|  |  |  |
| --- | --- | --- |
| title | string, required | Заголовок необходимой к закрытию задачи |

#### Задача не существует

Ошибка ввода. Задача с таким заголовком не найдена.

#### Пользователь указал некорректные данные

Ошибка ввода. Проверьте входные данные и повторите запрос.

#### Пример ответа

Задача [Полить цветы] выполнена!

### Метод 4. Задача не актуальна

> wontdo

> Введите заголовок

> {title}

Здесь обращаемся к конкретной задаче по заголовку, считая его для простоты уникальным идентификатором.

#### Входные параметры

|  |  |  |
| --- | --- | --- |
| title | string, required | Заголовок необходимой к закрытию задачи |

#### Задача не существует

Ошибка ввода. Задача с таким заголовком не найдена.

#### Пользователь указал некорректные данные

Ошибка ввода. Проверьте входные данные и повторите запрос.

#### Пример ответа

Задача [Полить цветы] более не актуальна!

### Выход

Приложение завершает работу, если ввести

> quit

или

> q