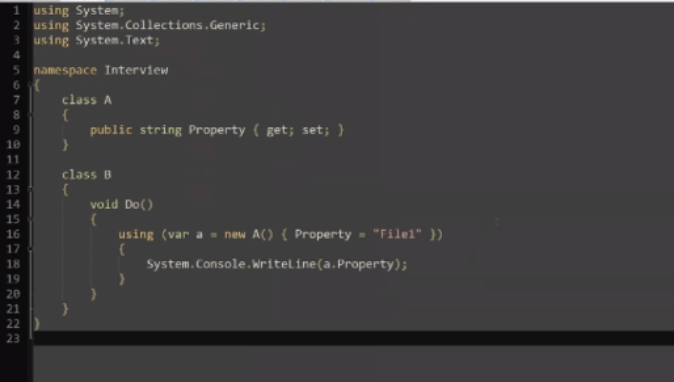
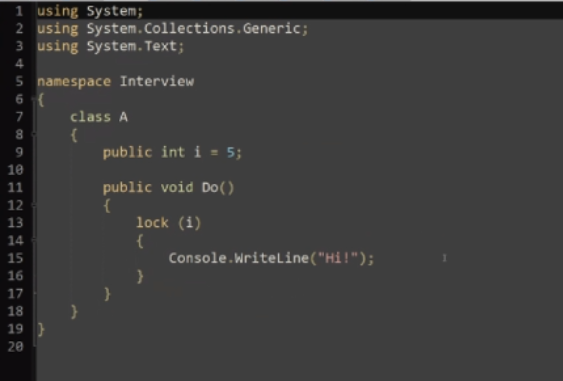
Все интервью строится вокруг задач. Задача, потом вопросы в догонку.

Задача 1) Вопрос – скомпилируется ли код.



Будет ошибка на этапе компиляции (intellisense подсветит проблему). Класс A должен реализовать интерфейс IDisposable

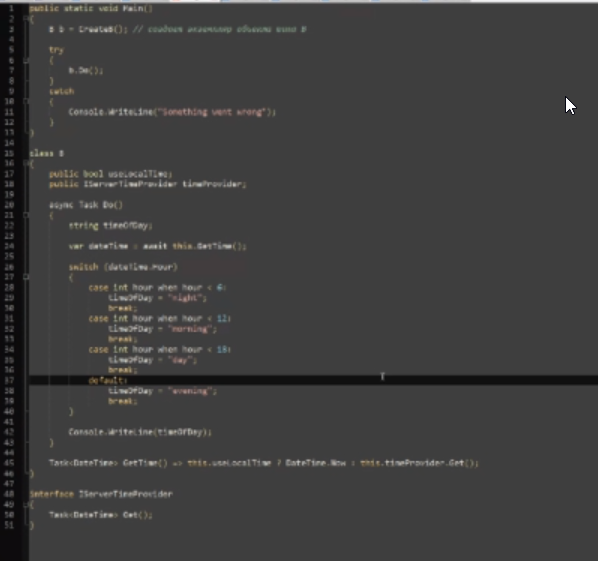
Задача 2) Скомпилируется ли код?

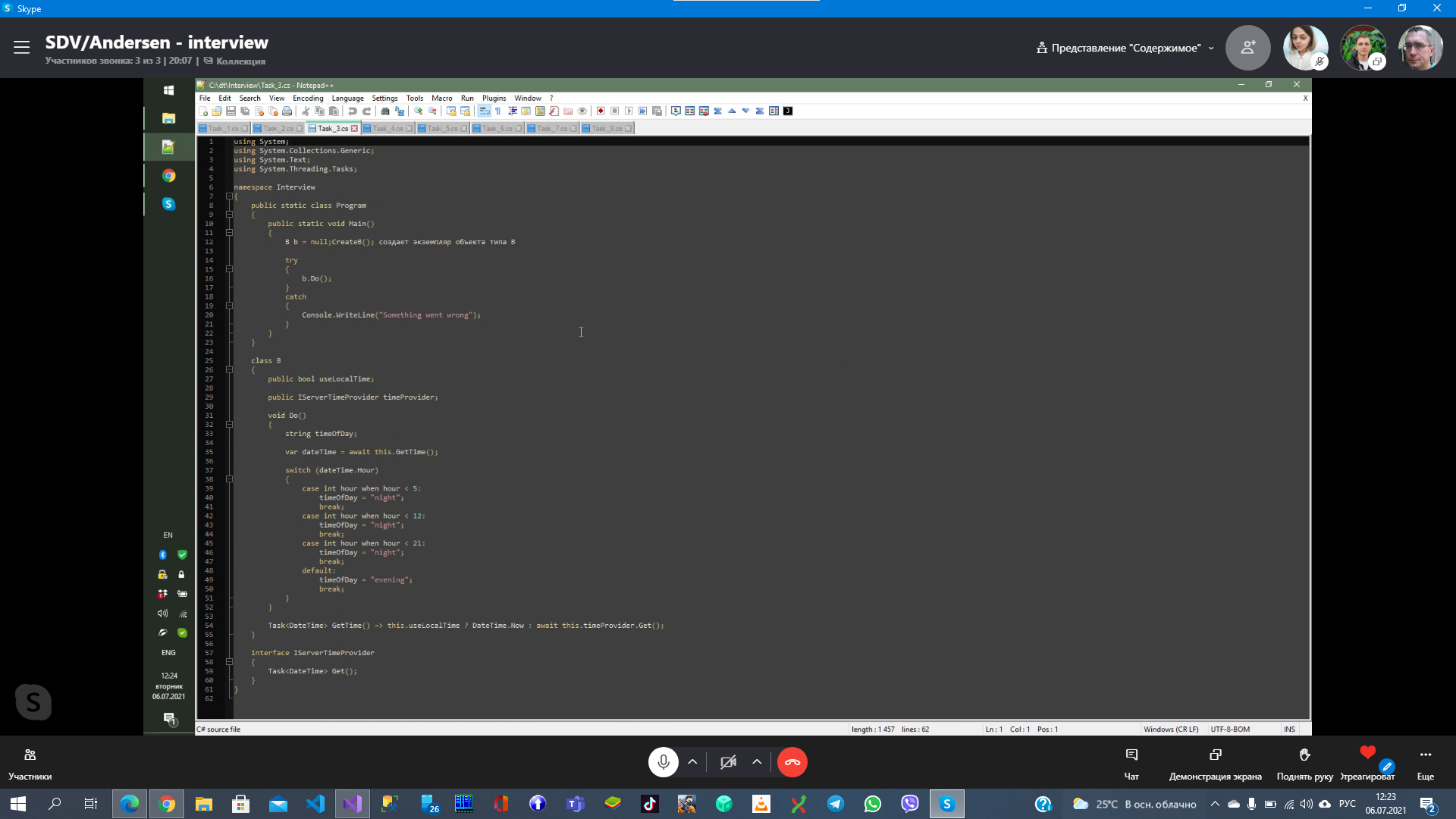


Будет ошибка на этапе компиляции (intellisense подсветит проблему). В lock нужно передавать reference type так как при передаче value type произойдет боксинг переменной каждый раз будет блокироваться новое значение соответственно разные потоки будут инициализировать (путем боксинга новое значение lock объекта и возможно заходить в критическую секцию одновременно), это не допустимо.

Задача 3) Скомпилируется ли?

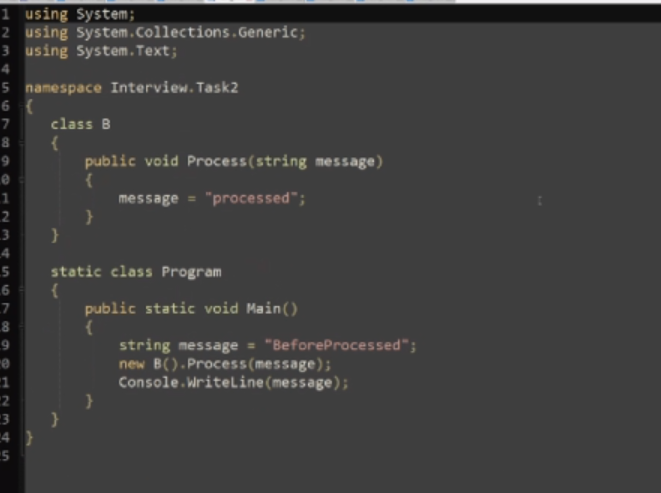
сори за плохое качество скрина





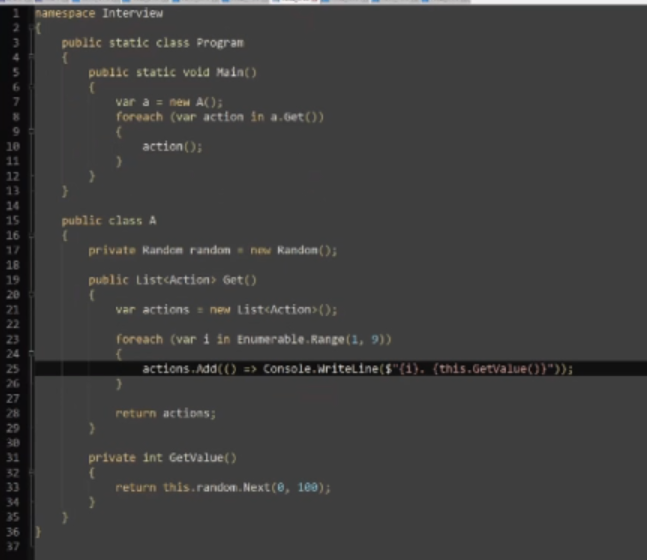
Тут суть в том что на 20 строке метод должен быть помечен async, на строке 45 после оператора ? нужно юзать Task.FromResult

Задача 4)



Выведется «BeforeProcessed» так как в Process аргумент не помечен ref. В противном случае будет «processed» так как string это reference type и внутри меняется ссылка на значение литерала в куче.

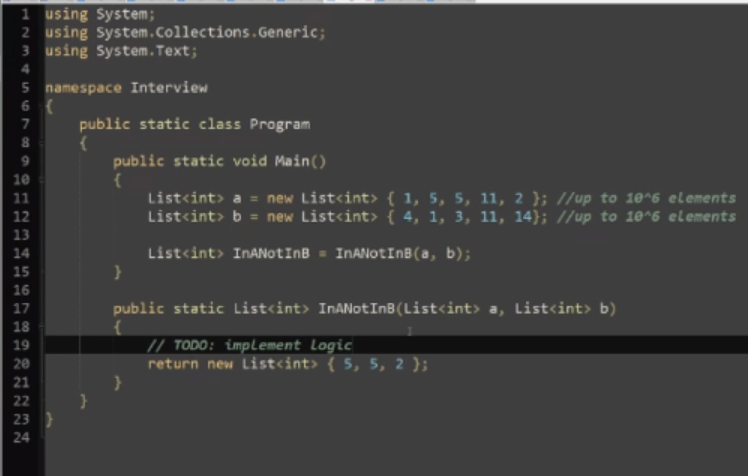
Задача 5)



На свежем фреймворке будет значения от 1 до 9. На фреймворке 4.5 ниже будет 9 ки. Эффект замыкания сработает. Как работает random.

Тут в место переменной i будет печататься одно и тоже значение 9 каждый раз. Решение перед этой строкой объявить временную переменную в нее передавать значение i и уже во WriteLine передавать эту временную переменную. Более подробнее почему так в рассмотрении задачи 6 вот здесь <https://proglib.io/p/8-csharp-questions/> . Вроде в какой то версии фреймворка или языка это поведение поменялось?

Задача 6)



Задача в том что бы метод InANotInB возвращал те значения которые есть в a, но НЕ встречаются в b. Я предложил решение сложностью O(n):

Создаем Dictionary<int, int>() – ключем будет элемента из **a** а value – кол-во элементов

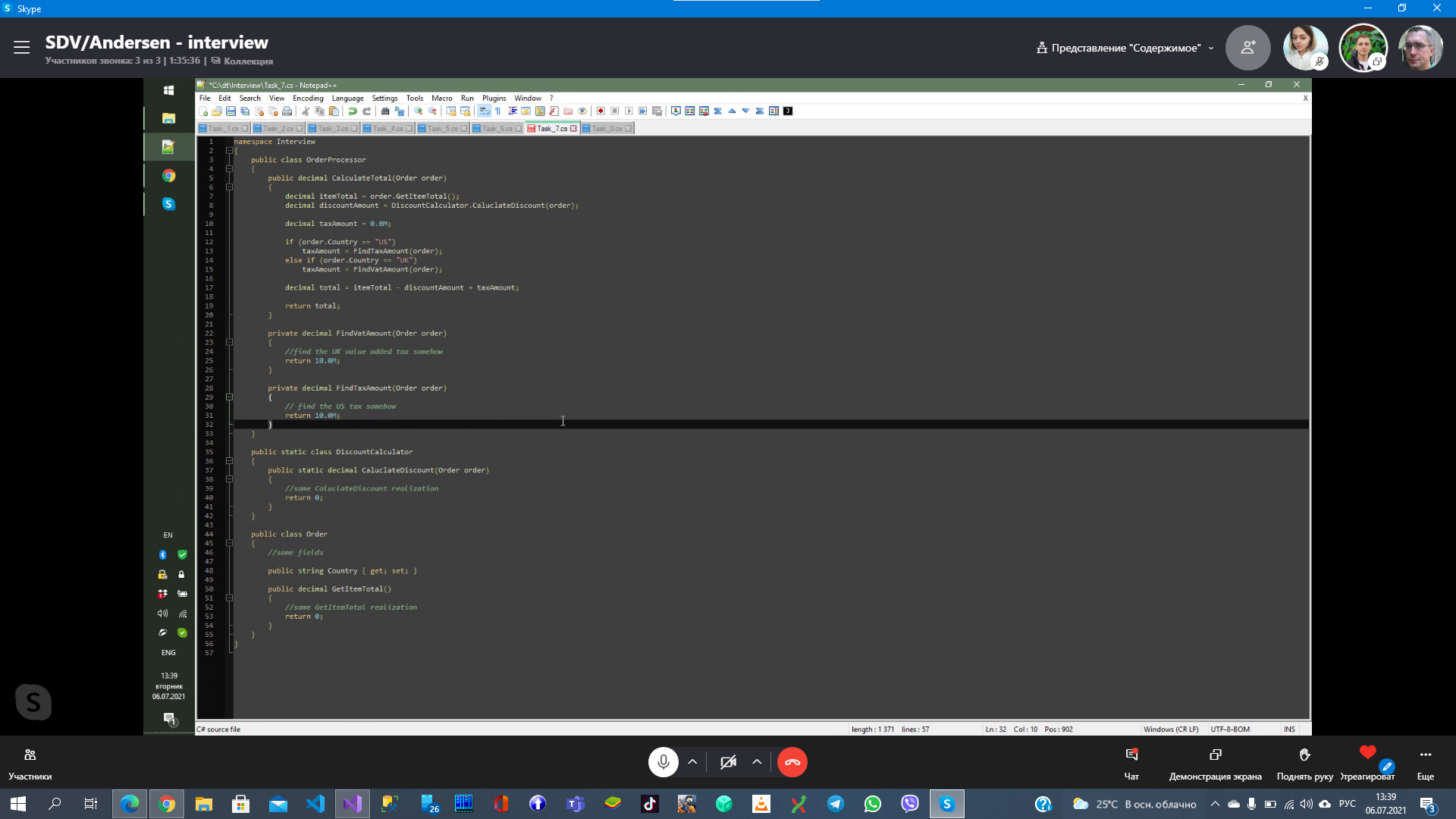
Далее прошелся бы по а и заолнял бы массив таким образом – если ключ отсутствует добавлял бы бы новое KeyValue (a[i], 1) если ключ уже есть то инкриментил бы значение в словаре (dictionary[a[i]]++)

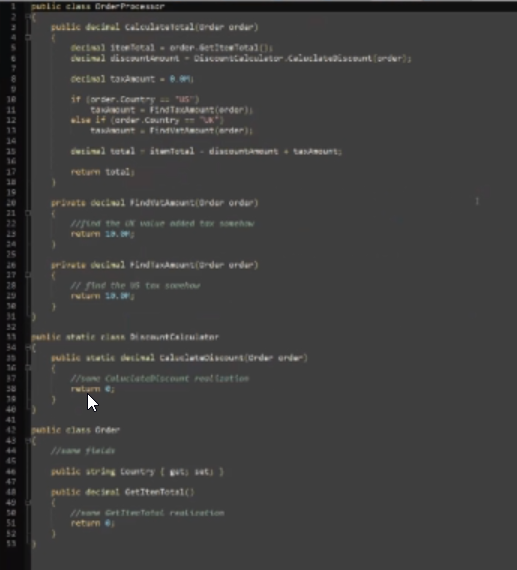
Далее проитерировался бы по b и если в словаре есть вхожение ключа со b[i] то удалил бы его из массива

Далее прошелся бы по словарю и распечатал ключи столько раз какое значение стоит в value.

Второй способ бинарный поиск, сортируем по возрастанию коллекцию и ищем.

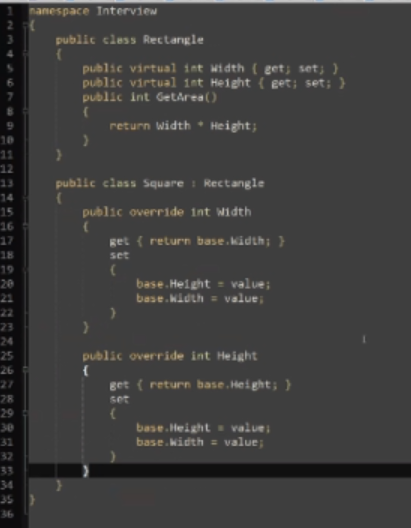
Задача 7)





Тут нужно обратить внимание на строки 10-13 их нужно вынести в отдельный сервисный интерфейс который будет иметь разные имплементации и его можно инжектить через контейнер зависимостей. Статический класс на сроке 33 тоже можно сделать как часть сервиса (статика может привести к утечке памяти – ее нужно избегать).

Задача 8)



Тут нарушен принцип solid LSP (принцип подстановки Лисков) нужно пояснить почему это проблема и КАК это решить

Проблема <https://youtu.be/NqvwYcjrwdw?t=505>

Решение примерно такое – объявляем новый АБСТРАКТНЫЙ класс FourSidedShape в нем объявляем необходимые методы и свойства. Квадрат и прямоугольник наследуем от FourSidedShape. После этого у нас не будет «непредсказуемого дочернего поведения»

Будет задача на кеширование:

Есть база выборка из которой стоит миллиард долларов и выборка остается актуальной в течении часа. Выборка нужна часто вопрос как минимизировать затраты?

ИМХО ошибочно или не оптимально: Я предположил что нужно юзать Redis и объект синхронизации, который содержит поле статуса и ключ от редиса который укажет на закешированный результат в Redis. Логика такая: Каждый запрос смотрит статус в объекта синхронизации если статус «processing» то ожидаем и пробуем чуть позже, если статус отличен то в блоке try выставляем статус «processing» и выполняем запрос, если норм то выставляем статус в «success» и помещаем результат в редис запоминаем в объекте ключ редиса (при помещении в редис выставляем TTL час) в блоке catch статус ставим в «failure».

Если статус «success» пробуем получить данные из Redis (если истек TTL то данных не будет).

ВАЖНО понять как этот код будет вести себя с параллельными запросами и какие секции тут будет критичными – тут у меня возникли трудности.

СКОРЕЕ ВСЕГО есть более элегантное решение

