Домашнее задание #4

Проверка форм

Содержание

Проверка форм

Java-программисты одной компании постоянно занимаются одним и тем же: они переписывают код *проверки форм*, заполняемых клиентами.

Одной из таких форм является форма бронирования (BookingForm.java):

BookingForm.java

```
public class BookingForm {
    private List<GuestForm> guests;
    private List<String> amenities;
    private String propertyType;
    // геттеры / сеттеры опущены для краткости
}
```

GuestForm.java

```
public class GuestForm {
    private String firstName;
    private String lastName;
    private int age;
    // геттеры / сеттеры опущены для краткости
}
```

Типичный код, взаимодействующий с ней, выглядит примерно так:

BookingService.java

```
public class BookingService {
    public void bookRoom(BookingForm bookingForm) {
        if (bookingForm != null && bookingForm.getAmenities() != null
                && List.of("TV", "Kitchen").containsAll(bookingForm.getAmenities())
                && bookingForm.getPropertyType() != null
                && List.of("House", "Hostel").contains(bookingForm.getPropertyType())
                && bookingForm.getGuests() != null
                && bookingForm.getGuests()
                .stream()
                .allMatch(it -> it != null && it.getAge() > 0
                        && it.getFirstName() != null && !it.getFirstName().isBlank()
                        && it.getLastName() != null && !it.getLastName().isBlank())
        ) {
            doBookRoom(bookingForm);
            return;
        }
        throw new RuntimeException("Форма заполнена неверно");
    }

    private void doBookRoom(BookingForm bookingForm) {
        // Бронируем комнату, реализация опущена
    }
}
```

Требования к ограничениям на вводимые данные часто меняются, из-за чего программисты постоянно совершают ошибки в проверках.

После очередного шквала звонков недовольных клиентов с детьми до одного года руководство компании задумалось и провело беседу с разработчиками.

Пристально изучив возможности Java, программисты решили описывать требования к формам **декларативно**, используя **аннотации**, которые будут обработаны *отдельной библиотекой*.

Это позволит им со спокойной душой удалить тот самый if, заменив его аннотациями в классах форм и вызовом библиотеки.

Измененная версия класса BookingForm могла бы выглядеть примерно так:

```
@Constrained
public class BookingForm {
    @NotNull
    @Size(min = 1, max = 5)
    private List<@NotNull GuestForm> guests;
    @NotNull
    private List<@AnyOf({"TV", "Kitchen"}) String> amenities;
    @NotNull
    @AnyOf({"House", "Hostel"})
    private String propertyType;
}
```

Программисты решили поручить разработку данной библиотеки Вам, а сами ушли исправлять ошибки в коде.

Задание

Необходимо разработать библиотеку, основной возможностью которой станет **проверка произвольных объектов в соответствии с выставленными аннотациями**.

Входные и выходные данные данной библиотеки описаны в терминах Java интерфейсов ниже.

Интерфейс библиотеки

ValidationError.java

```
public interface ValidationError {

    String getMessage();

    String getPath();

    Object getFailedValue();
}
```

Validator.java

```
public interface Validator {
    Set<ValidationError> validate(Object object);
}
```

interface ValidationError

Интерфейс ValidationError представляет собой *ошибку валидации*.

Метод getMessage()

возвращает описание ошибки. Его:

- можно показать пользователю, заполнявшему форму, в **пользовательском интерфейсе**
- его должен мочь понять **не** программист

Метод getPath()

Метод getPath() возвращает путь, по которому можно найти значение, не прошедшее проверку.

Он конструируется при обходе графа объектов (возможно, *рекурсивном*), следующим образом:

- Корень — пустая строка
- Каждый последующий уровень добавляет имя поля к строке
- Вложенные свойства разделяются точкой: .
- Элементы списка — как массивы: в квадратных скобках — индекс элемента ([0])
- Примеры:

- guests[0].firstName - в объекте, переданном в метод validate есть поле типа List. Первый элемент этого списка содержит поле firstName, проверка которого не прошла.
- amenities - корневой объект содержит поле amenities, не прошедшее проверку (возможно, оно было null)

Метод getFailedValue()

- возвращает Object, который не прошёл проверку

interface Validator

Интерфейс Validator имеет один единственный метод validate, он:

- принимает Object
  - Вывов данного метода с объектом, тип которого не имеет аннотации @Constrained - ошибочная ситуация
- возвращает Set обнаруженных ValidationError 'ов

Поддерживаемые аннотации

Аннотации, которые должны быть поддержаны библиотекой приведены в таблице

Таблица 1. Список поддерживаемых аннотаций

Аннотация	**Описание**	**Типы, у которых аннотация может находиться**	@Target	**Элементы аннотации**
@NotNull	Значение не должно быть null	Любой reference -тип	TYPE_USE	-
@Positive	Значение положительно	byte, short, int, long и их обёртки	TYPE_USE	-
@Negative	Значение отрицательно	byte, short, int, long и их обёртки	TYPE_USE	-
@NotBlank	См. String.isBlank	String	TYPE_USE	-
@NotEmpty	Значение не должно быть пустым	List<T>, Set<T>, Map<K, V>, String	TYPE_USE	-
@Size	Размер аннотированного элемента должен находиться в диапазоне [min, max]	List<T>, Set<T>, Map<K, V>, String	TYPE_USE	int min, int max
@InRange	Значение лежит в диапазоне [min; max]	byte, short, int, long и их обёртки	TYPE_USE	long min, long max

Аннотация	Описание	Типы, у которых аннотация может находиться	@Target	Элементы аннотации
@AnyOf	Значение находится в массиве, указанном в аннотации	String	TYPE_USE	String[] value
@Constrained	Тип подвергается проверке	Любой reference-тип	TYPE	-

String.isEmpty() и String.isBlank() - разные методы

- @Retention для всех аннотаций - RetentionPolicy.RUNTIME
- @NotNull - особый случай. Если значение - null и стоит другая аннотация, то её действие отменяется, а значение считается валидным. Пример:
  - @AnyOf("a") String s - значение null - валидно
  - @Negative Integer x - значение null - валидно
  - List<@NotBlank String> x - null 'ы, находящиеся внутри списка — валидные значения.
- Аннотации могут находиться внутри параметров типов
  - Например, List<@NotBlank String>
  - Но только внутри списков
- Допустимы списки списков, например: List<@NotEmpty List<@NotNull @NotBlank String>>
  - Это - список *непустых* списков (null 'ы допустимы) не null строк, вызов метода isBlank на которых должен возвращать false
  - Другие варианты *коллекций коллекций* недопустимы, но списки могут быть **любой вложенности**
- Проверяемые элементы - **поля** (методы просматривать не нужно)
- Доступ к private полям осуществляется напрямую, с использованием setAccessible(true)
- Для доступа к аннотациям над полями, у которых единственный @Target - TYPE_USE используйте Field#getAnnotatedType и методы AnnotatedType
- Способ обработки противоречивого набора аннотаций над полем (например, @Positive @Negative int x) остается на Ваше усмотрение

Пример использования

В данном разделе — небольшой пример потенциального использования библиотеки.

GuestForm.java - *форма гостя*

```
@Constrained
public class GuestForm {
    @NotNull
    @NotBlank
    private String firstName;
    @NotBlank
    @NotNull
    private String lastName;
    @InRange(min = 0, max = 200)
    private int age;

    public GuestForm(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }
}
```

Unrelated.java - *класс без аннотации* @Constrained

```
// Нет аннотации @Constrained, поэтому проверке не подвергается
public class Unrelated {
    @Positive
    private int x;

    public Unrelated(int x) {
        this.x = x;
    }
}
```

BookingForm.java - *форма бронирования*

```
@Constrained
public class BookingForm {
    @NotNull
    @Size(min = 1, max = 5)
    private List<@NotNull GuestForm> guests;
    @NotNull
    private List<@AnyOf({"TV", "Kitchen"}) String> amenities;
    @NotNull
    @AnyOf({"House", "Hostel"})
    private String propertyType;
    @NotNull
    private Unrelated unrelated;

    public BookingForm(List<GuestForm> guests, List<String> amenities, String
propertyType, Unrelated unrelated) {
        this.guests = guests;
        this.amenities = amenities;
        this.unrelated = unrelated;
    }
}
```

- Объявлены 3 класса
- 2 из них имеют аннотацию @Constrained, 1 - нет
- Форма бронирования содержит список *форм гостей*

И сам вызов библиотеки:

```
public class Main {
    public static void test(Validator validator) {
        List<GuestForm> guests = List.of(
                new GuestForm(/*firstName*/ null, /*lastName*/ "Def", /*age*/ 21),
                new GuestForm(/*firstName*/ "", /*lastName*/ "Ijk", /*age*/ -3)
        );
        Unrelated unrelated = new Unrelated(-1);
        BookingForm bookingForm = new BookingForm(
                guests,
                /*amenities*/ List.of("TV", "Piano"),
                /*propertyType*/ "Apartment",
                unrelated
        );
        Set<ValidationError> validationErrors = validator.validate(bookingForm);
    }
}
```

В результате validationErrors должен содержать следующие ошибки:

Таблица 2. validationErrors, *полученные в результате вызова выше*

Путь	Пример сообщения	failedValue	Причина
guests[0].firstName	must not be null	null	firstName первого гостя - null
guests[1].age	must be in range between 0 and 200	-3	age *второго* гостя — отрицательный
guests[1].firstName	must not be blank	""	Вызов isBlank на guests[1].firstName == true
amenities[1]	must be one of 'TV', 'Kitchen'	Piano	Piano не входит в TV, Kitchen
propertyType	must be one of 'House', 'Hostel'	Apartment	Apartment не входит в House, Hostel

- Ошибки unrelated.x нет, т.к. Unrelated не помечен @Constrained
  - Однако, если бы unrelated был == null, то ошибка должна была бы попасть в результат

Результат работы

Структура проекта

Проект должен иметь 2 дерева исходных файлов:

- **основное**, которое содержит:
  - реализацию библиотеки
  - интерфейсы, аннотации
- **для тестов**. В нём:
  - JUnit 5 @Test ' ы
  - **Тестовые примеры форм**

Тесты

- Тесты должны **покрывать** код библиотеки *как минимум* на 70%

Архив

Имя архива составляется по формуле HW4_zzz_Фамилия_Имя.**zip**, где:

- zzz - номер группы
- Фамилия - Ваша фамилия латиницей
- Имя - Ваше имя латиницей

Единственный допустимый формат архива - .zip.

Дедлайн

23:00 MSK 21.02.2020