**POJO** – это plain old Java object, простой Java-объект, не ограниченный какими-либо запретами, специфичными для того или иного фреймворка (за исключением спецификации самой Java, разумеется) и пригодный для использования в любой среде.

POJO используются для универсальной и наглядной сериализации и десериализации данных, так что основное их назначение в тестировании с RestAssured – это обработка Json в теле запросов и ответов.

Они позволяют достаточно гибко настраивать содержимое Json-объектов, отправляемых в теле запросов – например можно использовать один POJO-класс для составления Json-объектов с разным набором значений, избежав при этом многократного повторения одного и того же кода, или легко создавать Json-объекты с многоуровневыми вложениями, сохраняя при этом простую и наглядную структуру.

При обработке входящих ответов POJO позволяют извлекать любые необходимые значения для дальнейшего использования, а также дают (как уже было сказано выше – простой и наглядный) доступ к вложенным значениям.

**Свойства класса POJO**

Поскольку POJO имеют неограниченный доступ и использование, эти классы всегда должны быть общедоступными.

Классы POJO должны иметь общедоступный конструктор по умолчанию.

Конструкторы аргументов необязательны для классов POJO.

Как и модификатор доступа класса, все методы получения и установки класса POJO также должны быть общедоступными, чтобы значения, полученные методами получения, которые были установлены методами установки, также могли быть доступны другим классам.

Объекты в классе POJO могут иметь любой из следующих модификаторов доступа, таких как private, public или protected. Однако для обеспечения большей безопасности все переменные экземпляра должны быть определены как закрытые.

Предопределенные классы не должны расширяться классом POJO.

Классы POJO не должны реализовывать заранее заданные интерфейсы.

Классы POJO также не должны иметь заранее заданных аннотаций.

**Как работать с классом POJO.**

**POJO** - это публичный класс, все переменные в котором закрыты, а взаимодействие с ними происходит через сеттеры и геттеры. Также у POJO должен быть дефолтный конструктор, определяющий необходимый набор параметров для любого экземпляра этого класса. Соответственно, в самом базовом варианте POJO будет иметь следующий вид:

public class Book {

private String title;

private String author;

public Book() {

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;

}

}

Конструктор, как и любой другой метод, можно перегружать. Таким образом можно определить несколько допустимых наборов параметров при создании экземпляра класса. Например, следующие конструкторы позволят создать только книгу без названия и автора или книгу с названием и автором, но не с чем-то одним:

public Book() {

}

public Book(String title, String author) {

this.title = title;

this.author = author;

}

Впрочем, ничто не мешает нам создать книгу пустым конструктором, а затем назначить ей только один параметр с помощью сеттера.

Создание экземпляра класса Book происходит следующим образом:

Book book1 = new Book();

или

Book book2 = new Book("War and Peace", "Leo Tolstoy");

Использование pojo

Для демонстрации работы POJO будет использован API Restful Booker.

И в исходящих запросах, и во входящих ответах в этом API используется json-объект следующего вида:

{

"firstname" : "Jim",

"lastname" : "Brown",

"totalprice" : 111,

"depositpaid" : true,

"bookingdates" : {

"checkin" : "2018-01-01",

"checkout" : "2019-01-01"

},

"additionalneeds" : "Breakfast"

}

Для того чтобы описать этот json, нам нужно создать класс Booking, в котором мы объявим все необходимые переменные, таким же образом, как в приведенном выше примере.

Вы можете заметить, что поле bookingdates содержит в себе два вложенных значения, и у нас нет готового типа данных, который можно было бы присвоить соответствующей ему переменной. Это проблема решается крайне просто: нам всего лишь необходимо создать класс BookingDates:

public class BookingDates {

private String checkin;

private String checkout;

public BookingDates(String checkin, String checkout) {

this.checkin = checkin;

this.checkout = checkout;

}

}

Теперь у нас есть тип данных, подходящий для поля bookingdates, так что у нас есть все необходимое, чтобы создать класс Booking.

public class Booking {

private String firstname;

private String lastname;

private Integer totalprice;

private Boolean depositpaid;

private BookingDates bookingdates;

private String additionalneeds;

public Booking(String firstname, String lastname, Integer totalprice, Boolean depositpaid, BookingDates bookingdates, String additionalneeds) {

this.firstname = firstname;

this.lastname = lastname;

this.totalprice = totalprice;

this.depositpaid = depositpaid;

this.bookingdates = bookingdates;

this.additionalneeds = additionalneeds;

}

}

Точно таким же образом можно описывать и более сложные многоуровневые вложения.

Так как все переменные в обоих классах приватные, а нам понадобится доступ к хранящимся в них значениям, напишем для каждой из них Getter:

public String getFirstname() {

return firstname;

}

public String getLastname() {

return lastname;

}

(и так далее)

На этом подготовка POJO заканчивается, и мы можем переходить непосредственно к сериализации и десериализации.

**Сериализация** — процесс перевода структуры данных в последовательность байтов

В качестве примера сериализации с помощью POJO мы сформируем json-объект, который необходимо отправить в POST-запросе на эндпойнт /booking для создания новой брони.

Для этого создадим экземпляры классов Booking и BookingDates со всеми необходимыми для формирования Json-объекта данными

BookingDates bookingDates = new BookingDates("2021-08-31", "2021-09-10");

Booking booking = new Guest("John", "Doe", 100, true, bookingDates, "Breakfast");

Rest Assured прекрасно умеет преобразовывать POJO в Json, так что никаких дополнительных манипуляций над созданным объектом нам не потребуется. Все, что нужно сделать после этого – поместить созданный объект booking в тело нашего запроса:

given()

.contentType("application/json")

.accept("application/json")

.body(booking)

.post("https://restful-booker.herokuapp.com/booking")

.then()

.statusCode(200);

Таким образом мы отправляем запрос со следующим содержанием:

{

"firstname" : "John",

"lastname" : "Doe",

"totalprice" : 100,

"depositpaid" : true,

"bookingdates" : {

"checkin" : "2021-08-31",

"checkout" : "2021-09-10"

},

"additionalneeds" : "Breakfast"

}

**десериализации** — создание структуры данных из битовой последовательности

Для изучения работы десериализации с помощью POJO мы создадим такую же бронь, как до этого, десериализуем ответ от API, извлечем значение bookingid, а затем запросим данные о брони с этим bookingid и сравним из с отправленным json-объектом.

Для начала посмотрим на то, как выглядит ответ на отправленный нами запрос о создании новой брони:

{

"bookingid": 36,

"booking": {

"firstname": "John",

"lastname": "Doe",

"totalprice": 100,

"depositpaid": true,

"bookingdates": {

"checkin": "2021-08-31",

"checkout": "2021-09-10"

},

"additionalneeds": "Breakfast"

}

}

Как мы видим, структура json стала чуть сложнее и к ней добавился еще один уровень. По аналогии с предыдущими классами, создадим класс BookingInfo:

public class BookingInfo {

private int bookingid;

private Booking booking;

public BookingInfo() {

}

public int getBookingid() {

return bookingid;

}

public Booking getBooking() {

return booking;

}

}

Также добавляем в уже существующие классы пустые конструкторы:

public BookingDates() {

}

и

public Booking() {

}

Теперь мы можем сохранить ответ на POST-запрос в переменную, одновременно с этим десериализовав его:

BookingInfo bookingInfo = given()

.contentType("application/json")

.accept("application/json")

.body(body)

.post(POST\_URI)

.then()

.extract().body().as(BookingInfo.class);

Как видите, процесс десериализации практически идентичен сериализации – Rest Assured можно преобразовать Json в POJO без каких-либо дополнительных действий и библиотек.

Теперь у нас есть экземпляр класса BookingInfo, который содержит в себе всю полученную информацию. Чтобы проверить, что данные действительно сохранились, используем полученный bookingid, чтобы получить информацию о созданной нами брони:

Booking checkBooking = given()

.get("https://restful-booker.herokuapp.com/booking/" + bookingInfo.getBookingid())

.then()

.extract().body().as(Booking.class);

В ответе мы получили верные данные, но чтобы не полагаться на визуальную проверку, сравним полученный Json с тем, который мы создали при формировании POST-запроса.

Самый простой способ это сделать – это привести POJO к строке. Для этого перепишем метод toString() в Booking и BookingDates следующим образом:

@Override

public String toString() {

return "firstname: " + this.firstname + "; lastname: " + this.lastname + "; totalprice: " + this.totalprice + "; depositpaid: " + this.depositpaid + "; bookingdates: " + this.bookingdates + "; additionalneeds: " + this.additionalneeds;

}

и

@Override

public String toString() {

return "checkin: " + this.checkin + "; checkout: " + this.checkout;

}

А затем сравним полученные строки:

Assert.assertEquals(checkBooking.toString(), bookingInfo.getBooking().toString());