Домашнее задание #4

Mapper

Содержание

Идея . . . 1
- Пример использования . . . 1

Формат . . . 3
- Примеры . . . 3

Интерфейс . . . 5

Сохранение . . . 9

Восстановление . . . 10

Как сохранять и восстанавливать объекты . . . 11
- Ограничения . . . 11
- Примеры . . . 11

Поддержка `record`-классов . . . 12

retainIdentity . . . 13
- Пример . . . 13

Поддерживаемые типы . . . 15

Набор поддерживаемых аннотаций . . . 16
- `@Exported` . . . 16
- `@PropertyName` . . . 17
- `@Ignored` . . . 19
- `@DateFormat` . . . 19

Формат сдачи . . . 21
- Проект . . . 21
- Архив . . . 21

Тесты . . . 22

Оценка . . . 23
- Максимальный балл . . . 23
- Распределение баллов . . . 23

Идея

Необходимо разработать библиотеку сохранения и восстановления Java-объектов по аннотациям. Пользователи библиотеки смогут сохранять Java-объекты в файл, `OutputStream` или `String` и восстанавливать объекты из `String`, `InputStream` или файла.

Сперва покажем небольшой пример, а затем дадим формальное описание требований

Пример использования

ReviewComment — пример сохраняемого и восстанавливаемого класса

```
@Exported ①
public class ReviewComment {
    private String comment;
    @Ignored ②
    private String author;
    private boolean resolved;
    // toString, геттеры, сеттеры опущены для краткости
}
```

① `@Exported` классы — классы, с которыми работает `Mapper`. `Mapper` — интерфейс библиотеки, полная спецификация — в пункте «Интерфейс»

② `@Ignored` поля `Mapper` игнорирует

Пример сохранения

WriteExample.java

```
public class WriteExample {
    public static void main(String[] args) throws IOException {
        ReviewComment reviewComment = new ReviewComment(); ①
        reviewComment.setComment("Одни static'и в работе");
        reviewComment.setResolved(false);
        reviewComment.setAuthor("Проверяющий #1");

        Mapper mapper = ...; ②
        String string = mapper.writeToString(reviewComment); ③
        System.out.println(string); ④
    }
}
```

① Создаём экземпляр `ReviewComment`

② Создаём экземпляр класса, реализующего `Mapper`

③ Сохраняем `reviewComment`

④ Выводим строковое представление объекта

Пример вывода

```
{"comment":"Одни static'и в работе","resolved":false}
```

В выводе нет автора, потому что мы пометили поле `author` в классе `ReviewComment` аннотацией `@Ignored`

Пример восстановления

ReadExample.java

```
public class ReadExample {
    public static void main(String[] args) {
        Mapper mapper = new DefaultMapper();
        ReviewComment reviewComment = mapper.readFromString(ReviewComment.class, """
                {"comment":"Одни static'и в работе","resolved":false}""");
        System.out.println(reviewComment);
    }
}
```

Пример вывода

```
ReviewComment(comment=Одни static'и в работе, author=null, resolved=false)
```

Формат

1. Разработайте собственный формат представления объектов или воспользуйтесь существующим. Примеры — ниже
2. Опишите формат в текстовом файле в корне проекта, даже если Вы решили взять существующий формат

Обязательные требования к формату:

1. Человекочитаемость
2. Формат хранит *ключи*:
   a. Если поле помечено аннотацией `@PropertyName`, то ключ — `value` аннотации `@PropertyName`.
   b. Если над полем нет аннотации `@PropertyName`, то ключ — имя поля.
   c. Если реализация поддерживает `record`'ы, то для `record`-классов:
      i. Если компонента `record`'а помечена аннотацией `@PropertyName`, то ключ — `value` аннотации `@PropertyName`
      ii. Иначе — имя `RecordComponent`

Посмотрите «Пример сохранения». В нём строковое представление объекта содержит *ключи* — подстроки `comment` и `resolved`

Примеры

Человекочитаемые форматы

JSON

```
{"comment":"Одни static'и в работе","resolved":false}
```

.properties

```
comment=Одни static'и в работе
resolved=false
author.name=Josh
```

YAML

```
comment: Одни static'и в работе
resolved: false
author:
  name: Josh
```

Нечитаемый человеком формат

```
98080ZNHasduy091283nsfdahjsdfoioiwqerXX
```

1. *Можно ли использовать формат, похожий на Json?*

   Да, можно

Не используйте готовые средства записи и чтения форматов Json, XML, properties и т.п. Соответствующие средства должны быть реализованы самостоятельно. Мы поставим 0 за работу, если это будет не так.

Интерфейс

Интерфейс `Mapper`, реализуйте все его методы в решении

`Mapper.java`

```
public interface Mapper {
    /**
     * Читает сохранённый экземпляр класса {@code clazz} из строки {@code input}
     * и возвращает восстановленный экземпляр класса {@code clazz}.
     * <p>
     * Пример вызова:
     *
     * <pre>
     *         String input = """
     *                 {"comment":"Хорошая работа","resolved":false}""";
     *         ReviewComment reviewComment =
mapper.readFromString(ReviewComment.class, input);
     *         System.out.println(reviewComment);
     * </pre>
     *
     * @param clazz класс, сохранённый экземпляр которого находится в {@code input}
     * @param input строковое представление сохранённого экземпляра класса {@code
clazz}
     * @param <T> возвращаемый тип метода
     * @return восстановленный экземпляр {@code clazz}
     */
    <T> T readFromString(Class<T> clazz, String input);

    /**
     * Читает объект класса {@code clazz} из {@code InputStream}'a
     * и возвращает восстановленный экземпляр класса {@code clazz}.
     * <p>
     * Данный метод закрывает {@code inputStream}.
     * <p>
     * Пример вызова:
     *
     * <pre>
     *         String input = """
     *                 {"comment":"Хорошая работа","resolved":false}""";
     *         ReviewComment reviewComment = mapper.read(ReviewComment.class,
     *                 new
ByteArrayInputStream(input.getBytes(StandardCharsets.UTF_8)));
     *         System.out.println(reviewComment);
     * </pre>
     *
     * @param clazz класс, сохранённый экземпляр которого находится в {@code
inputStream}
     * @param inputStream поток ввода, содержащий строку в {@link
java.nio.charset.StandardCharsets#UTF_8} кодировке
```

```
     * @param <T> возвращаемый тип метода
     * @return восстановленный экземпляр класса {@code clazz}
     * @throws IOException в случае ошибки ввода-вывода
     */
    <T> T read(Class<T> clazz, InputStream inputStream) throws IOException;

    /**
     * Читает сохранённое представление экземпляра класса {@code clazz} из {@code
File}'а
     * и возвращает восстановленный экземпляр класса {@code clazz}.
     * <p>
     * Пример вызова:
     *
     * <pre>
     *         ReviewComment reviewComment = mapper.read(ReviewComment.class, new
File("/tmp/review"));
     *         System.out.println(reviewComment);
     * </pre>
     *
     * @param clazz класс, сохранённый экземпляр которого находится в файле
     * @param file файл, содержимое которого - строковое представление экземпляра
{@code clazz}
     * в {@link java.nio.charset.StandardCharsets#UTF_8} кодировке
     * @param <T> возвращаемый тип метода
     * @return восстановленный экземпляр {@code clazz}
     * @throws IOException в случае ошибки ввода-вывода
     */
    <T> T read(Class<T> clazz, File file) throws IOException;

    /**
     * Сохраняет {@code object} в строку
     * <p>
     * Пример вызова:
     *
     * <pre>
     *         ReviewComment reviewComment = new ReviewComment();
     *         reviewComment.setComment("Хорошая работа");
     *         reviewComment.setResolved(false);
     *
     *         String string = mapper.writeToString(reviewComment);
     *         System.out.println(string);
     * </pre>
     *
     * @param object объект для сохранения
     * @return строковое представление объекта в выбранном формате
     */
    String writeToString(Object object);

    /**
     * Сохраняет {@code object} в {@link OutputStream}.
     * <p>
```

```
     * То есть после вызова этого метода в {@link OutputStream} должны оказаться
байты, соответствующие строковому
     * представлению {@code object}'а в кодировке {@link
java.nio.charset.StandardCharsets#UTF_8}
     * <p>
     * Данный метод закрывает {@code outputStream}
     * <p>
     * Пример вызова:
     *
     * <pre>
     *         ReviewComment reviewComment = new ReviewComment();
     *         reviewComment.setComment("Хорошая работа");
     *         reviewComment.setResolved(false);
     *
     *         mapper.write(reviewComment, new FileOutputStream("/tmp/review"));
     * </pre>
     *
     * @param object объект для сохранения
     * @throws IOException в случае ошибки ввода-вывода
     */
    void write(Object object, OutputStream outputStream) throws IOException;

    /**
     * Сохраняет {@code object} в {@link File}.
     * <p>
     * То есть после вызова этого метода в {@link File} должны оказаться байты,
соответствующие строковому
     * представлению {@code object}'а в кодировке {@link
java.nio.charset.StandardCharsets#UTF_8}
     * <p>
     * Данный метод закрывает {@code outputStream}
     * <p>
     * Пример вызова:
     *
     * <pre>
     *         ReviewComment reviewComment = new ReviewComment();
     *         reviewComment.setComment("Хорошая работа");
     *         reviewComment.setResolved(false);
     *
     *         mapper.write(reviewComment, new File("/tmp/review"));
     * </pre>
     *
     * @param object объект для сохранения
     * @throws IOException в случае ошибки ввода-вывода
     */
    void write(Object object, File file) throws IOException;
}
```

Вы можете поменять интерфейс:

1. Кинуть непроверяемое исключение, не указанное в Javadoc
2. Добавить новый метод

Только не забудьте дополнить Javadoc.

Сохранение

Сохранение — это когда мы преобразуем Java-объект в строку выбранного формата.

В интерфейсе `Mapper` методы, название которых начинается со слова `write`, сохраняют объекты:

1. `writeToString(Object)` сохранит объект в строку и вернёт её
2. `write(Object, OutputStream)` сохранит объект в `OutputStream`
3. `write(Object, File)` сохранит объект в `File`

Посмотрите пример вызова `writeToString` в «Примере сохранения».

Восстановление

Восстановление — это когда мы преобразуем строковое представление обратно в Java-объект.

В интерфейсе `Mapper` методы, название которых начинается со слова `read`, восстанавливают объекты:

1. `readFromString(Class<T>, String)` прочитает экземпляр класса из строки
2. `read(Class<T>, InputStream)` прочитает экземпляр класса из `InputStream`'а
3. `read(Class<T>, File)` прочитает экземпляр класса из файла

Посмотрите примеры вызова `readFromString` в «Примере восстановления» и Javadoc интерфейса.

Как сохранять и восстанавливать объекты

Реализация интерфейса `Mapper` должна:

1. *Сохранять* и *восстанавливать* объекты по полям их классов.
    a. Статические поля и поля, для которых метод `Field#isSynthetic` возвращает `true`, `Mapper` должен пропустить.
2. Обращаться к полям напрямую, чтобы восстановить или сохранить объект.
    a. При необходимости она использует метод `AccessibleObject#trySetAccessible`.

Ограничения

Классы, с которым работает `Mapper` должны:

1. Иметь `public` конструктор без параметров, за исключением `record`-классов.
2. Наследоваться непосредственно из `java.lang.Object` или из `java.lang.Record`, если Вы реализуете поддержку `record`-классов.

Эти ограничения должны контролироваться программно.

Примеры

1. Пример сохранения. В нём реализация `Mapper`:
    1. Проходит по всем полям рефлексивно
    2. Записывает значения полей в строку в соответствии с форматом
    3. Возвращает строку
2. Пример восстановления. В нём реализация `Mapper`:
    1. Читает переданную строку в определённом формате
    2. Вызывает рефлексивно конструктор без параметров класса `ReviewComment`
    3. Сопоставляет значения в строке полям класса `ReviewComment` и устанавливает поля созданному экземпляру `ReviewComment`

Поддержка `record`-классов

1 балл из 10 — за поддержку `record`-классов.

Если реализация `Mapper` не поддерживает `record`-классы, то она должна кинуть `UnsupportedOperationException` при попытке сохранения или восстановления `record`-класса.

Прочитайте статью «Record Classes» на сайте Oracle.

Работа с `record`-классами похожа на работу с обычными классами. Используйте:

1. `RecordComponent`
2. Метод `Class#getRecordComponents`. Он вернёт массив компонент `record`'а

Song.java — пример `record`'а

```
@Exported
public record Song(
        String artist,
        @PropertyName("myTitle") String title
) {
}
```

Вызовите канонический конструктор, передав все значения компонент `record`'а, чтобы восстановить экземпляр `record`-класса.

retainIdentity

3 балла из 10 — за поддержку `retainIdentity = true`

`retainIdentity` — `boolean` настройка `Mapper`'а. Она определяет, должен ли `Mapper` сохранять идентичность `@Exported` объектов.

Реализация интерфейса `Mapper` должна принимать параметр `boolean retainIdentity` в конструкторе.

Пример

IdentityRetainingExample.java

```
public class IdentityRetainingExample {
    public static void main(String[] args) throws IOException {
        Foo foo = new Foo();
        Bar bar = new Bar(); ①
        foo.setBar0(bar); ②
        foo.setBar1(bar); ③

        boolean retainIdentity = true;
        Mapper mapper = new DefaultMapper(retainIdentity);
        Foo restored = mapper.readFromString(Foo.class, mapper.writeToString(foo)); ④

        System.out.println(restored.getBar0() == restored.getBar1()); ⑤
    }
}
```

① Создали один экземпляр `Bar`

② Установили `bar` значением поля `bar0` через сеттер `setBar0`

③ Установили `bar` значением поля `bar1` через сеттер `setBar1`

④ Сохраняем и восстанавливаем экземпляр `foo`

⑤ Спрашиваем, равны ли поля `foo.bar0` и `foo.bar1`

1. Если `Mapper` настроен с `retainIdentity = true`, то сравнение должно выдать `true`, иначе — `false`.
2. Если в графе объектов есть две разные ссылки на один и тот же объект `@Exported` класса по `==`, то восстановленный граф должен тоже давать `true` при сравнении этих же ссылок, если `retainIdentity = true`.
3. Если же `retainIdentity = false`, то сравнение даёт `false` независимо от того, какой результат `==` был до вызова метода `write*`

Посмотрите на класс `IdentityHashMap`

Что делать с циклами?

Поддерживать циклы не нужно. Реализация `Mapper` должна кинуть исключение, если обнаруживает цикл

Поддерживаемые типы

Поддержите сохранение и восстановление экземпляров `@Exported` классов с полями типа `T`, где `T` — одно из:

1. Примитив или соответствующий wrapper-класс
2. `String`
3. `@Exported` класс — класс, сохраняемый и восстанавливаемый `Mapper`'ом
4. `List<T>`
5. `Set<T>`
6. `enum`
7. `LocalDate`
8. `LocalTime`
9. `LocalDateTime`

То есть, экземпляр класса `BookingForm` библиотека должна суметь сохранить и восстановить:

```
// геттеры и сеттеры опущены для краткости
@Exported
public class BookingForm {
    private List<Guest> guests;
}

@Exported
public class Guest {
    private String name;
    private Set<Preference> preferences;
}

@Exported
public class Preference {
    private String description;
}
```

Набор поддерживаемых аннотаций

@Exported

Методы `Mapper` кидают исключение при попытке сохранить или восстановить экземпляр класса без аннотации `@Exported`.

Exported.java

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Exported {
    NullHandling nullHandling() default NullHandling.EXCLUDE;

    UnknownPropertiesPolicy unknownPropertiesPolicy()
        default UnknownPropertiesPolicy.FAIL;
}
```

NullHandling

`nullHandling` настраивает обработку значений `null` для reference типов.

NullHandling.java

```
public enum NullHandling {
    EXCLUDE,
    INCLUDE,
}
```

- Значение `EXCLUDE` исключает запись `null` значения в строковое представление
- `INCLUDE` его включает

Пример с `NullHandling.EXCLUDE`

ExcludeNullsExample.java

```
@Exported(nullHandling = NullHandling.EXCLUDE) ①
public class ExcludeNullsExample {
    private final Integer integer = null; ②

    public static void main(String[] args) throws IOException {
        Mapper mapper = new DefaultMapper();
        String string = mapper.writeToString(new ExcludeNullsExample());
        System.out.println(string);
    }
}
```

① Явно указали `EXCLUDE` в `@Exported`

② Значение поля `integer` — `null`

Пример вывода:

```
{}
```

Пример с `NullHandling.INCLUDE`

IncludeNullsExample.java

```
@Exported(nullHandling = NullHandling.INCLUDE)
public class IncludeNullsExample {
    private final Integer integer = null;

    public static void main(String[] args) throws IOException {
        Mapper mapper = new DefaultMapper();
        String string = mapper.writeToString(new ExcludeNullsExample());
        System.out.println(string);
    }
}
```

Пример вывода:

```
{"integer":null}
```

UnknownPropertiesPolicy

`UnknownPropertiesPolicy` определяет, что делает `Mapper`, если при восстановлении он встречает неизвестное поле.

UnknownPropertiesPolicy.java

```
public enum UnknownPropertiesPolicy {
    IGNORE,
    FAIL,
}
```

- `IGNORE` игнорирует неизвестное поле
- `FAIL` кидает исключение

@PropertyName

PropertyName.java

```
@Target({
        ElementType.RECORD_COMPONENT,
        ElementType.FIELD,
})
@Retention(RetentionPolicy.RUNTIME)
public @interface PropertyName {
    String value();
}
```

`@PropertyName` меняет ключ, по которому `Mapper` восстанавливает или сохраняет значение поля.

Пример

ReviewComment.java

```
@Exported
public class ReviewComment {
    @PropertyName("opinion") ①
    private String comment;
    // toString опущен
}
```

① Задали `@PropertyName`, ожидаем в выводе «opinion», а не «comment»

WriteExample.java

```
public class WriteExample {
    public static void main(String[] args) throws IOException {
        ReviewComment reviewComment = new ReviewComment();
        reviewComment.setComment("Молодец, но пустые catch блоки");

        Mapper mapper = ...;
        String string = mapper.writeToString(reviewComment);
        System.out.println(string);

        ReviewComment restored = mapper.readFromString(ReviewComment.class, string);
        System.out.println(restored);
    }
}
```

Пример вывода:

```
{"opinion":"Молодец, но пустые catch блоки"} ①
ReviewComment(comment=Молодец, но пустые catch блоки) ②
```

① Сохранили `reviewComment`, в выводе — `opinion` вместо `comment`

② Восстановили `reviewComment`, установили `comment`, прочитав `opinion` из строки, потому что поле `comment` помечено аннотацией `@PropertyName("opinion")`

@Ignored

Ignored.java

```
@Target({
        ElementType.RECORD_COMPONENT,
        ElementType.FIELD,
})
@Retention(RetentionPolicy.RUNTIME)
public @interface Ignored {
}
```

1. Значение поля и ключ не попадают в строковое представление объекта
2. `Mapper` пропускает значение поля при восстановлении объекта

@DateFormat

`@DateFormat` устанавливает, по какому шаблону `Mapper` сохраняет и восстанавливает поля типов `LocalDate`, `LocalTime` и `LocalDateTime`.

DateFormat.java

```
@Target({
        ElementType.RECORD_COMPONENT,
        ElementType.FIELD,
})
@Retention(RetentionPolicy.RUNTIME)
public @interface DateFormat {
    /**
     * Шаблон для сохранения и восстановления дат
     */
    String value();
}
```

Используйте:

1. `DateTimeFormatter#ofPattern`
2. Методы `parse` и `format` классов `LocalDate`, `LocalTime` и `LocalDateTime`

Пример

LocalDateTimeWrapper.java

```
@Exported
public class LocalDateTimeWrapper {
    @DateFormat("uuuu-MMMM-dd HH:mm:ss") ①
    private final LocalDateTime myLocalDateTime = LocalDateTime.now();


    public static void main(String[] args) throws IOException {
        LocalDateTimeWrapper wrapper = new LocalDateTimeWrapper();
        Mapper mapper = new DefaultMapper();
        String string = mapper.writeToString(wrapper);
        System.out.println(string);
    }
}
```

① Шаблон для сохранения и восстановления `LocalDateTime`

Пример вывода:

```
{"myLocalDateTime":"2022-February-05 11:18:47"}
```

Формат сдачи

Проект

- Интерфейс `Mapper` и аннотации — в пакете `ru.hse.homework4`
- Реализация интерфейса — в другом пакете, имя пакета не должно начинаться с `ru.hse.homework4`
- Структура проекта соответствует стандартной структуре maven-проекта:
  - Исходный код библиотеки — в `src/main/java`
  - Ресурсы библиотеки, если они есть, — в `src/main/resources`
  - Тесты библиотеки — в `src/test/java`
  - Ресурсы для тестов, если они есть, — в `src/test/resources`

Архив

`zip` архив, его имя — `lastName_firstName_group_HW4.zip`, где:

- `lastName` — Ваша фамилия латиницей
- `firstName` — Ваше имя латиницей
- `group` — номер группы

Пример: `Ivanov_Ivan_201_HW4.zip`

Тесты

Используйте JUnit5 для тестирования.

Тесты должны:

1. Покрывать методы `Mapper`'а
2. Тестировать варианты значений элементов используемых аннотаций
3. Покрывать 70% строк кода библиотеки

Оценка

Максимальный балл

Без выполненных требований к тестам максимальная оценка — 5.

Поддержка `record`-классов	Поддержка `retainIdentity = true`	Максимальный балл
Не реализована	Не реализована	6
Реализована	Не реализована	7
Не реализована	Реализована	9
Реализована	Реализована	10

Максимальный балл — это максимальный балл, который мы сможем поставить.

Если Вы выполните требования к тестам, поддержите `record`'ы и `retainIdentity = true`, то это **не** означает, что мы поставим 10. Вы должны не просто реализовать требования *как-то*, а сделать это правильно с соответствующими JUnit5 тестами

Распределение баллов

Таблица 1. Распределение баллов. Незначительные отклонения допустимы

Критерий	Требования	Баллы
Проект	1. Не сдана директория `target`, созданная maven'ом 2. Формат сдачи учтен 3. Код отформатирован 4. Соблюдено соглашение об именовании 5. Выбран пакет не по умолчанию и не `com.company` 6. maven запускает тесты 7. Нет предупреждений Idea или они подавлены с объяснением 8. Выбранный формат описан 9. Декомпозиция, архитектура	**До 2-х баллов**

Критерий	Требования	Баллы
Сохранение	1. Реализовано сохранение `@Exported` классов 2. Поддержаны все типы 3. Поддержаны все аннотации 4. Корректная работа с потоками вывода 5. Адекватная обработка ошибок, исключений	**До 2-х баллов**
Восстановление	То же самое, что в пункте «Сохранение» и: 1. Библиотека восстанавливает объекты через `public` конструктор без параметров и устанавливает поля напрямую 2. Поддержаны все типы 3. Поддержаны все аннотации 4. Корректная работа с потоками ввода	**До 2-х баллов**
Поддержка `record`-классов	То же самое, что и в пункте «Восстановление», и библиотека поддерживает `record`'ы	**До одного балла**
Поддержка retainIdentity	То же самое, что в пункте «Восстановление», и библиотека поддерживает retainIdentity `= true`	**До 3-х баллов**