Санкт-Петербургский государственный университет

кафедра системного программирования

Свитков Сергей Андреевич

Реализация библиотеки для потоковой записи файлов формата XLSX

Курсовая работа

Научный руководитель:
доц. к.т.н. Литвинов Ю. В.

Консультант:
рук. отд. раз. ПО, "НТЦ Протей" Заведеев М. В.

Санкт-Петербург
2018

Оглавление

Введение

В современном мире большой популярностью пользуются многопользовательские веб-приложения. Приложения такого рода могут использоваться для самых разнообразных целей — от совместного редактирования документов несколькими пользователями до анализа различной статистики операторами связи.

Одну из категорий веб-приложений представляют проекты в сфере телекоммуникаций и биллинга. Такие приложения используются операторами связи для анализа различной статистики по действиям абонентов: перемещения между зонами роуминга, количество входящих/исходящих вызовов, и т.д.

Для формирования отчётов требуется формат представления данных, предоставляющий строгое структурирование. К таким можно отнести JSON [4], XML [8], XLSX [7]. Однако, следует принять во внимание, что отчетность или статистика, представленная в таком формате, может использоваться как при взаимодействии различных компонент приложения или различных приложений, так и для анализа человеком. Преимущество формата XLSX в том, что для просмотра таких файлов существуют общеизвестные решения (Microsoft Excel, OpenOffice Calc, Google Sheets).

Таким образом, возникает необходимость в библиотеке, которая позволила бы формировать документы формата XLSX. Следует отметить, что, поскольку объем данных не ограничен, а веб-приложения являются многопользовательскими, то необходимо формировать файл в потоковом режиме, то есть, сохранять в оперативной памяти только ограниченное количество данных.

Исходя из этого была поставлена задача о разработке своей библиотеки для подобных задач. Консультантом работы был предложен алгоритм для генерации файлов формата XLSX в потоковом режиме, была начата его реализация. Позже, в ходе обзора решений, было выяснено, что одна из существующих реализаций уже обладает нужной функциональностью. Несмотря на это, было принято решение продол-

жить разработку и сравнить полученную реализацию с существующими библиотеками, тем самым реализовав потенциально применимую на практике библиотеку и получив ценный опыт разработки проекта с открытым исходным кодом.

1. Постановка задачи

Целью данной работы является разработка библиотеки для потоковой записи файлов формата XLSX и сравнение её с существующими реализациями. Для достижения этой цели были поставлены следующие задачи:

- сформулировать алгоритм, который будет использоваться для генерации файлов формата XLSX в потоковм режиме;
- реализовать предложенный алгоритм в виде переиспользуемой библиотеки с открытым исходным кодом, документацией, примерами использования и артефактом в Maven;
- сравнить полученную реализацию с существующими библиотеками по потреблению RAM (оперативной памяти) и скорости работы при создании документа.

2. Обзор существующих решений

Задача формирования документов формата XLSX не является новой, имеется ряд существующих библиотек.

2.1. Apache POI

Библиотека Apache POI [1] (далее — просто POI) предоставляет средства как для формирования, так и для чтения файлов формата XLSX. До версии 3.8 в библиотеке отсутствовала поддержка потоковой обработки файлов. Несмотря на то, что, начиная с версии 3.8, появилась поддержка потоковой обработки, некоторые проблемы с использованием оперативной памяти остались. Так, некоторые операции над документами всё равно можно проводить только храня целый документ в памяти. К недостаткам библиотеки можно отнести отсутствие возможности задать условия автоматического создания новых страниц в документе, а так же отсутствие полной документации и примеров использования.

На момент начала данной работы не было известно о том, что в POI реализована поддержка потоковой обработки документов, что во многом и послужило мотивацией для постановки задачи о реализации своей библиотеки. После анализа последней версии библиотеки было решено реализовать собственный алгоритм потокового формирования документов и сравнить полученную реализацию с POI по производительности.

2.2. SJXLSX

Библиотека SJXLSX [6]— проект с открытым исходным кодом. Документации к проекту крайне мало, к тому же, последнее обновление в репозитории было сделано в 2015м году. В ходе анализа исходного кода было выяснено, что данная библиотека не предоставляет возможности генерировать файлы в потоковом режиме.

2.3. Итоги обзора

Исходя из результатов обзора было принято решение о реализации библиотеки для потоковой генерации файлов и её последующем сравнении с рассмотренными существующими решениями.

3. Анализ формата XLSX

Для реализации библиотеки потребовалось изучить структуру формата XLSX.

Формат XLSX был создан в декабре 2006 года при участии Microsoft, Ecma, ISO/IEC. К сожалению, документации по стандарту MS-XLSX крайне мало, в открытом доступе можно найти два стандарта: ISO/IEC 29500 [3] и ECMA-376 [2]. Эти документы имеют крайне большой объем (порядка 7 тысяч страниц), и на их изучение ушло бы большое количество времени.

Для формата XLSX так же существует другой стандарт, OOXML [5]. В отличие от MS-XLSX, данный стандарт довольно хорошо документирован, поэтому было принято решение формировать документы в соответствии с данным стандартом.

Формат XLSX представляет собой ZIP-архив с XML файлами. Его структура (рис. 1) представляет собой следующий набор файлов и директорий:

- `Content_Types`.xml — типы данных в архиве и пути к ним;
- `_rels` — зависимости между файлами внутри архива;
- docProps — метаданные: имя автора, дата создания, ...;
- xl — директория с основными файлами архива: workbook, страницы, стили, таблицы.

Рассмотрим более подробно основные компоненты архива: Workbook и Worksheet.

3.1. Workbook

Workbook представляет собой XML-файл, который не содержит данных файла, но содержит следующие мета-данные: ссылки на отдельные Worksheet и их свойства. В работе приведён пример файла workbook.xml см. в листинге 1. Содержимое документа содержится непосредственно в Worksheets.

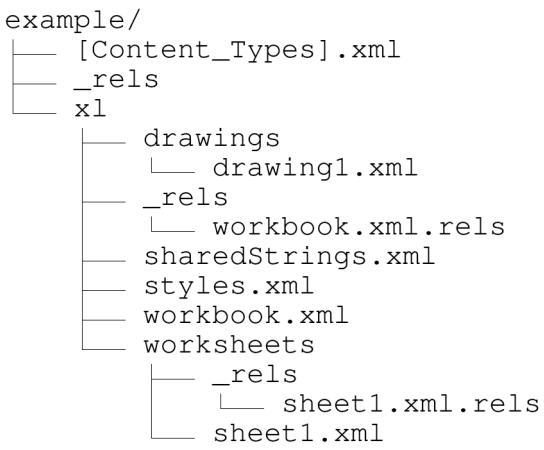

Рис. 1: Структура формата XLSX

3.2. Worksheet

Worksheet содержат данные, из которых и состоит документ. Worksheet может иметь один из следующих форматов: Grid, Chart, Dialog sheet. Наиболее популярным и хорошо задокументированным является Grid, рассмотрим его более подробно.

Grid представляет собой ”сетку” из ”клеток” (cells) с данными. Каждая клетка может содержать какой-то определенный тип данных: числа, булевские переменные, формулы, и т.д.. Для оптимизации использования памяти строковые значения хранятся не в теле самой клетки, а в отдельной части документа. Это позволяет минимизировать дупликацию строк. Пример файла worksheet.xml см. в листинге 2.

Закончив анализ формата XLSX можно приступить к реализации библиотеки.

4. Реализация

В данной секции будут описаны алгоритм работы и архитектура библиотеки. Библиотека была названа oxml-doc (поскольку файлы создаются в соответствии с стандартом OOXML) и реализована на языке Java с использованием системы сборки Maven. Исходный код реализации библиотеки опубликован на Github в репозитории организации "НТЦ Протей"[1]. Работа велась под учётной записью likeanowl[2]. Код реализации опубликован под лицензией MIT. Так же был опубликован артефакт в Maven, с groupId ru.protei.

4.1. Алгоритм

Было принято решение реализовать следующий алгоритм для генерации файлов формата XLSX:

- для каждой страницы создавать временный файл;
- хранить в RAM только один ряд (во время создания);
- после создания добавлять ряд во временный файл страницы;
- после завершения формирования документа — записывать данные из временных файлов в основной файл;
- для экономии дискового пространства сжимать временные файлы.

4.2. Архитектура

Упрощённая архитектура приложения представлена на рис. 2.

Основными элементами приложения являются классы WorkbookWriter и Worksheet. StreamConsumer хранит список временных файлов, генерируемых при создании документа, а так же

[1]https://github.com/protei-rnd/oxml-doc
[2]https://github.com/likeanowl

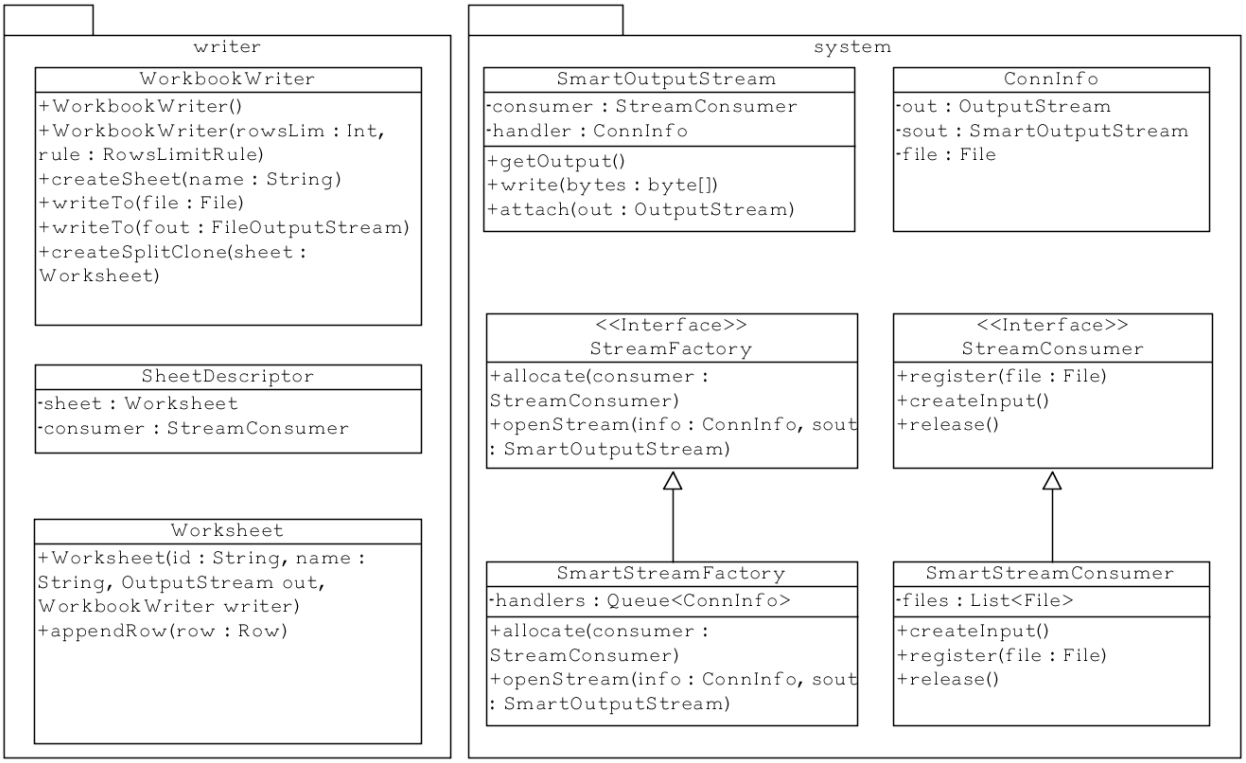

Рис. 2: Архитектура библиотеки oxml-doc, упрощённый вариант

позволяет создать InputStream из хранимых файлов. Класс ConnInfo используется для связи SmartOutputStream и StreamConsumer. SmartOutputStream позволяет создать OutputStream из временных файлов и добавлять записи в них. SheetDescriptor используется для связи Worksheet и StreamConsumer. SmartStreamFactory используется для управления открытыми SmartOutputStream и создания новых экземпляров этого класса.

Помимо этого в приложении присутствуют классы для представления ряда, клетки (Cell) и задания стилей для этих классов.

Полный цикл формирования документа:

- WorkbookWriter создаёт новую страницу, создавая для неё SheetDescriptor, StreamConsumer и SmartOutputStream. Из OutputStream, полученного из SmartOutputStream, создаётся ZipOutputStream для нового временного файла. Созданная страница помечается как готовая к записи.

- После создания страницы к ней можно добавлять ряды. Каждый

ряд обрабатывается, применяются стили, если они были заданы для ряда, пишется заголовок страницы, если он ещё не был записан. В том случае, когда задано ограничение на максимальное количество рядов для одной страницы, при его превышении происходит автоматическое разбиение документа и в WorkbookWriter создается новый экземпляр Workheet.

- После завершения формирования документа, происходит формирование архива с XML файлами. Это делается путём обхода списка SheetDescriptor, созданных для страниц документа. Из временных файлов при помощи StreamConsumer создается SequenceInputStream, данные из него пишутся в соответствующий странице файл архива.

В репозитории с исходным кодом были размещены примеры (см. листинг 3, листинг 4) использования библиотеки.

5. Эксперименты

Было принято решение сравнить полученную реализацию библиотеки с существующими решениями по следующим метрикам: количество потребляемой при генерации файла оперативной памяти и скорость работы. Исходный код инструмента, который был применен для тестирования, опубликован на Github[3]. Эксперименты проводились на машине с конфигурацией:

- CPU — Intel i7-7700
- RAM — DDR4 2400 MHz, 16 Gb
- SSD — SATA 3, скорость записи 340 MB/s

В ходе тестирования библиотек на количество используемой оперативной памяти формировались файлы размером 100 тысяч рядов. Генерация производилась добавлением порций данных по 1000 строк с небольшим интервалом между итерациями. Такой эксперимент позволил симулировать типовые условия, в которых предполагается использование библиотеки: получение некоторого количества данных из базы данных или файла другого формата на сервере, их обработка и добавление в файл.

Следует так же отметить, что непрерывная запись файла построчно не создает проблем в работе библиотеки oxml-doc (это подтверждается экспериментами, проведенными для сравнения скорости работы библиотек, см. рис. 4), но такой тип записи не представляет интереса, т.к. не соответствует тому, как предполагается использовать реализацию.

Результаты эксперимента (рис. 3) показали, что реализованная в рамках данной работы библиотека по количеству используемой RAM находится на одном уровне с Apache POI, при этом Apache POI и oxml-doc показывают более хорошие результаты, чем SJXLSX. Графики имеют довольно большие скачки, это связано с работой Java Garbage

[3]https://github.com/likeanowl/simpletestutil

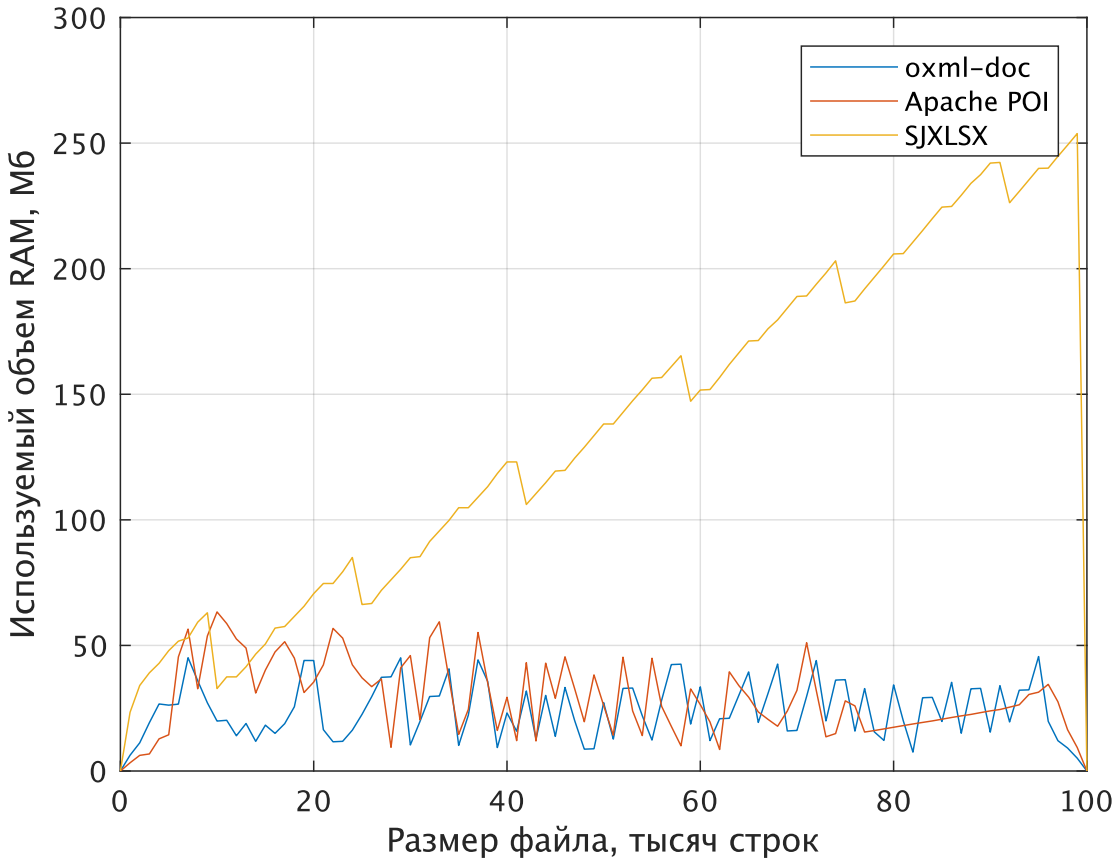

Рис. 3: Сравнение библиотек oxml-doc, Apache POI, SJXLSX по количеству потребляемой оперативной памяти при генерации файлов

Collector, который периодически удаляет объекты, созданные в ходе записи.

Результаты эксперимента по сравнению скорости работы библиотек представлены на рис. 4. В данном эксперименте запись документов производилась непрерывно.

По итогам проведенных экспериментов можно сделать выводы о том, что реализованная в рамках данной работы библиотека oxml-doc по количеству используемой RAM в ходе генерации документа находится на одном уровне с библиотекой Apache POI и существенно превосходит SJXLSX, а по скорости работы опережает Apache POI, но уступает SJXLSX на маленьких объемах данных. То, что SJXLSX формирует файлы малого размера с более высокой скоростью, можно объяснить тем, что при записи документа библиотека SJXLSX сохраняет весь до-

кумент в оперативной памяти.

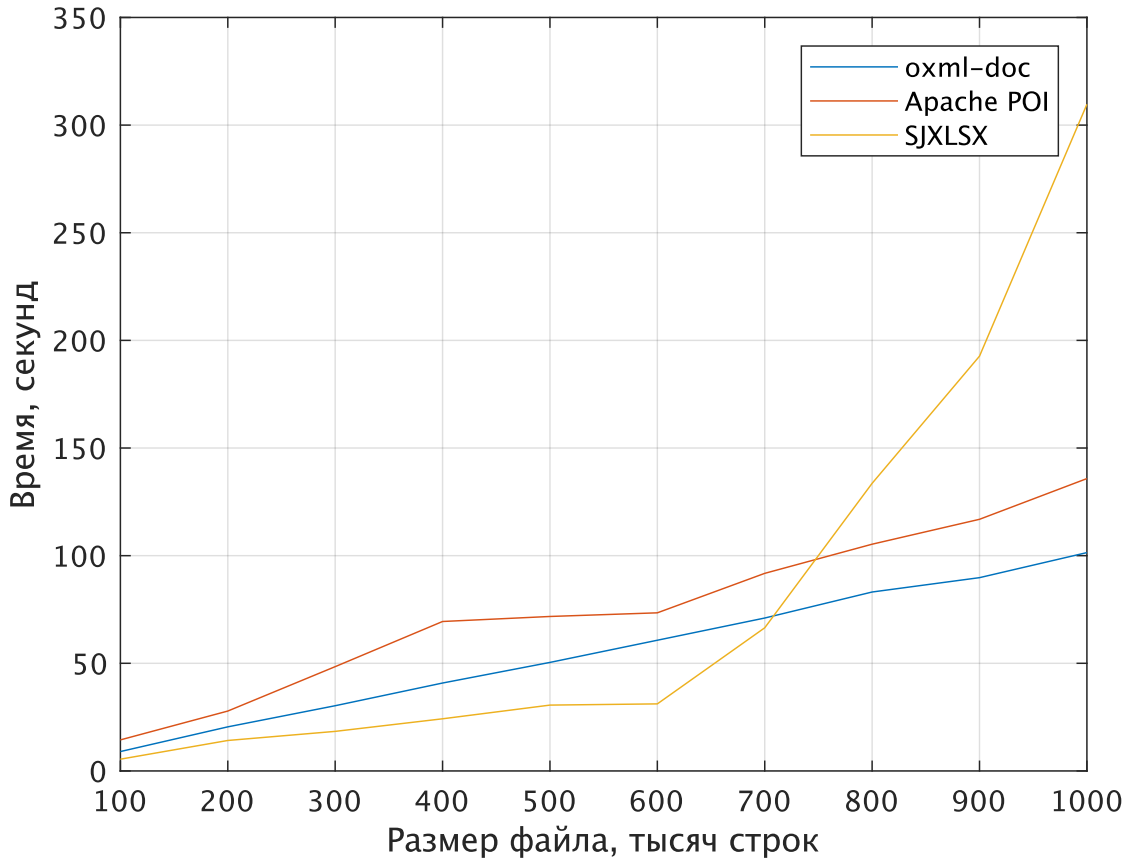

Рис. 4: Сравнение библиотек oxml-doc, Apache POI, SJXLSX по скорости работы

Заключение

В ходе данной работы были достигнуты следующие результаты:

- реализован предложенный в рамках работы алгоритм записи файлов формата XLSX;
- реализация оформлена в виде библиотеки для потоковой записи файлов формата XLSX с открытым исходным кодом и артефактов в Maven;
- проведено тестирование реализации;
- проведено сравнение реализации с существующими библиотеками по количеству используемой RAM и скорости работы при создании документа;

Листинги

```
<workbook . . .>
    . . .
    <workbookPr . . ./>
    <sheets>
        <sheet name=''sheet1'' r:id=''rId1''>
        <sheet name=''sheet2'' r:id=''rId2''>
        <sheet name=''sheet3'' r:id=''rId3''>
    </sheets>
    . . .
</workbook>
```

Листинг 1: Пример файла workbook.xml

```
<worksheet . . .>
. . .
    <cols>
        <col min=''1'' max=''1'' width=''26.140625'' customWidth=''1''/>
            . . .
        </cols>

        <sheetData>
            <row r=''1''>
            <c r=''A1'' s=''1'' t=''s''>
            <v>0</v>
            . . .
            </c>
            </row>
            . . .
        </sheetData>
        . . .

        <mergeCells count=''1''>
        <mergeCell ref=''B12:J16''/>
        </mergeCells>

        <pageMargins . . ./>
        <pageSetup . . ./>

        <tableParts ccount=''1''>
        <tableParts count=''1''>
    </tablePart r:id=''rId2''/>
</worksheet>
```

Листинг 2: Пример файла worksheet.xml

```
try (WorkBookWriter writer = new WorkBookWriter(1000, RowLimitRules.SPLIT)) {
    final WorkSheet sheet = writer.createNewSheet(''sheet'');
    final Row row = new Row();
    for (int i = 0; i < 1000; i++) {
        for (int j = 0; j < 55; j++)
            row.append(''test'');
        sheet = sheet.appendRow(row);
        row.clear();
    }
    writer.writeTo(new File(''Test.xlsx''));
} catch (Throwable e) {
    e.printStackTrace();
}
```

Листинг 3: Пример создания простого документа XLSX с помощью библиотеки oxml-doc

```
try (WorkBookWriter writer = new WorkBookWriter()) {
    // base text font
    final Font textFont = writer.createFont();
    textFont.setFontSize(''11'');
    textFont.setName(''Calibri'');
    textFont.setColor(''FF2020FF'');

    // headers text font
    final Font headerFont = writer.createFont();
    headerFont.setFontSize(''11'');
    headerFont.setName(''Calibri'');
    headerFont.setColor(''FF2020F0'');
    headerFont.setBold(true);

    // special font for the best sold items
    final Font bestSoldFont = writer.createFont();
    bestSoldFont.setFontSize(''11'');
    bestSoldFont.setName(''Calibri'');
    bestSoldFont.setColor(''FFF02020'');

    // headers fill
    final Fill emptyFill = writer.createFill();

    final Fill headerFill = writer.createFill();
    headerFill.setPattern(FillPattern.SOLID);
    headerFill.setBackgroundColor(new Color(0xFFD0D0D0));
    headerFill.setFrontColor(new Color(0xFFD0D0D0));

    // cpecial fill for price column
```

```
30      final Fill priceFill = writer.createFill();
31      priceFill.setPattern(FillPattern.SOLID);
32      priceFill.setBackgroundColor(new Color(0xFFFFFF00));
33      priceFill.setFrontColor(new Color(0xFFFFFF00));
34
35
36      // base style
37      final CellStyle baseStyle = writer.createCellStyle();
38      baseStyle.setFont(textFont);
39
40      // header style
41      final CellStyle headerStyle = writer.createCellStyle();
42      headerStyle.setFont(headerFont);
43      headerStyle.setFill(headerFill);
44      headerStyle.setAligment(new Aligment(HorizontalAligment.CENTER, false,
            VerticalAligment.CENTER));
45
46      // header row style
47      final RowStyle headerRowStyle = new RowStyle(30, headerStyle, true, true);
48
49      // price column style
50      final CellStyle priceColumnStyle = writer.createCellStyle();
51      priceColumnStyle.setFont(headerFont);
52      priceColumnStyle.setFill(priceFill);
53      priceColumnStyle.setNumberFormat(writer.createNumFormat().asNumberFmt(2));
54
55      final CellStyle bestSoldStyle = writer.createCellStyle();
56      bestSoldStyle.setFont(bestSoldFont);
57      bestSoldStyle.setFill(emptyFill);
58
59      final RowStyle bestSoldRowStyle = new RowStyle(20, bestSoldStyle, true, true);
60
61      //Lets create a sheet
62      writer.createNewSheet(''Market place'');
63
64      //Column rules
65      writer.addColumnRule(new ColumnRule(0, 50));
66      writer.addColumnRule(new ColumnRule(1, 25));
67      writer.addColumnRule(new ColumnRule(2, 40));
68      writer.addColumnRule(new ColumnRule(3, 25, priceColumnStyle));
69      writer.addColumnRule(new ColumnRule(4, 20, priceColumnStyle));
70
71      writer.appendRow(new Row(headerRowStyle).append(''Item'', headerStyle).append(''Count'',
            headerStyle)
72              .append(''Last changes'', headerStyle)
73              .append(''Price'', headerStyle));
```

```
74
75      writer.appendRow(new Row().append(''Oranges'').append(''2 kg'').append(new Date()).append
            (10).append(''10.0''));
76      writer.appendRow(new Row().append(''Apples'').append(''1 kg'').append(new Date()).append
            (20).append(''20.0''));
77      writer.appendRow(new Row().append(''Mangos'').append(''50 kg.'').append(new Date()).
            append(30).append(''30.0''));
78
79      writer.appendRow(new Row(bestSoldRowStyle).append(''Bananas'', bestSoldStyle)
80              .append(''sold out'', bestSoldStyle)
81              .append(new Date(), bestSoldStyle)
82              .append(25.3).append(''25.3''));
83
84      final File testFile = new File(''test_base.xlsx'');
85      try (FileOutputStream fout = new FileOutputStream(testFile)) {
86          writer.writeTo(fout);
87      } catch (Throwable e) {
88          e.printStackTrace();
89      }
90      Assert.assertNotEquals(testFile.length(), 0);
91
92      //cleaning up
93      testFile.delete();
94  } catch (Throwable e) {
95      e.printStackTrace();
96  }
```

Листинг 4: Пример создания документа XLSX с форматированием с помощью библиотеки oxml-doc

Список литературы

[1] Apache. Apache POI // Apache official page. — URL: https://poi.apache.org/ (online; accessed: 18.05.2018).

[2] Ecma. ECMA-376 // web page. — URL: https://www.ecma-international.org/publications/standards/Ecma-376.htm (online; accessed: 18.05.2018).

[3] ISO/IEC. ISO/IEC 29500 // web page. — URL: https://www.iso.org/standard/71691.html (online; accessed: 18.05.2018).

[4] JSON. JSON // JSON official page. — URL: https://www.json.org/ (online; accessed: 18.05.2018).

[5] OOXML. OOXML // web page. — URL: http://officeopenxml.com/SScontentOverview.php (online; accessed: 18.05.2018).

[6] Pelfree David. SJXLSX // Github repository. — URL: https://github.com/davidpelfree/sjxlsx (online; accessed: 18.05.2018).

[7] XLSX. XLSX // Microsoft XLSX format description page. — URL: https://msdn.microsoft.com/en-us/library/dd922181(v=office.12).aspx (online; accessed: 18.05.2018).

[8] XML. XML // XML specification page. — URL: https://www.w3.org/TR/xml/ (online; accessed: 18.05.2018).