Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им.В.И.Ульянова (Ленина)»

Кафедра МОЭВМ

ОТЧЕТ
по лабораторно-практической работе № 10
«Протоколирование работы приложения»
по дисциплине: «Объектно - ориентированное программирование на языке Java»

Выполнил: Локтионов Т. И.

Факультет КТИ

Группа № 3311

Подпись преподавателя _____

Санкт-Петербург

2024 г

Цель работы:
знакомство с методами протоколирования работы приложения с использованием библиотеки Log4j.

Описание задания:
1. Проанализируйте методы в различных потоках приложения и определите основные действия, которые необходимо контролировать. На основе этого анализа опишите конфигурационный файл.

2. Подключите библиотеку Log4j и настройте вывод в лог-файл.

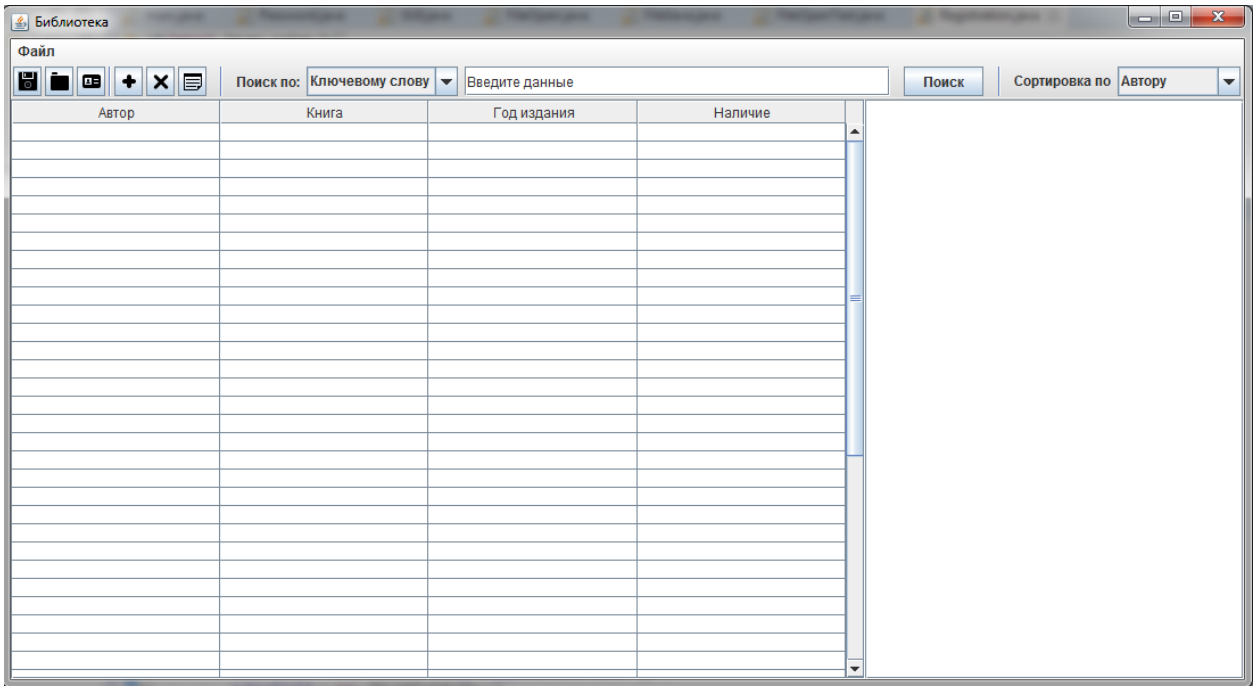
3. Организуйте вывод в лог-файл сообщений типа WARN, INFO и DEBUG. В код классов должны быть вставлены комментарии документации, поясняющие смысл выводимой информации.

4. Запустите приложение в различных режимах протоколирования.

Описание экранной формы:

Экранная форма предназначена для отображения списка больных и врачей для администратора регистратуры поликлиники, она может менять свой размер на экране (начальный размер 800x600). Форма должна реализовывать следующие функции: загрузку списка пациентов, болезней, врачей, дат приема и состояния приема из файла и выгрузку этой информации в файл; редактирование списка, включая: добавление, удаление, корректировку информации; удобный поиск, по ключевым словам, и/или другими методами (имя пациента, дата и т.д.)

Макет (взят из приложенных к методичке файлов):



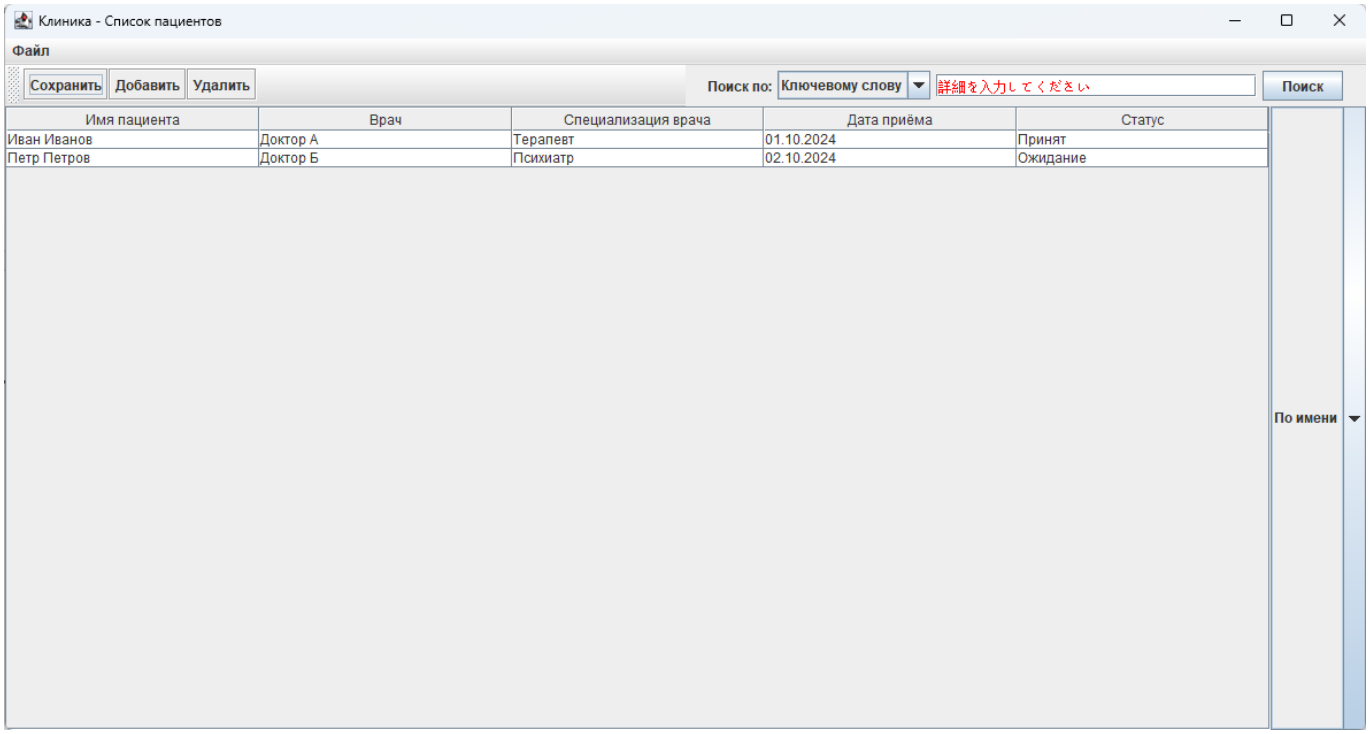
Перечень возможных исключений:

- Пустые поля ввода, которые недопустимы при добавлении пациента в таблицу.

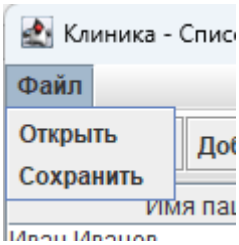
- Некорректные данные, которые не соответствуют установленным требованиям (например, неверный формат).
- Дублирование записей, когда пользователь пытается добавить существующего пациента.
- Неожиданные ошибки, возникающие в ходе выполнения программы, которые могут быть перехвачены и обработаны.

Работоспособность приложения:

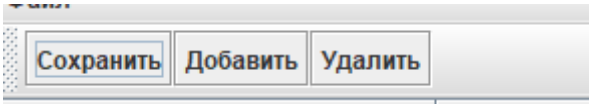
JFrame:



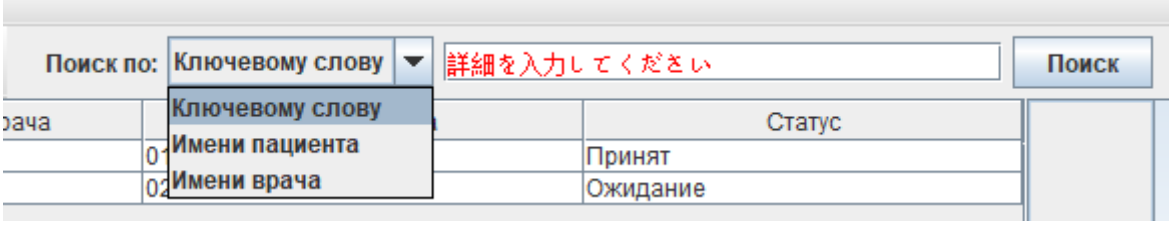
JMenuBar:



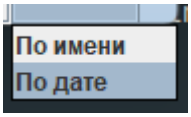
JToolBar:



JPanel && JTextField:

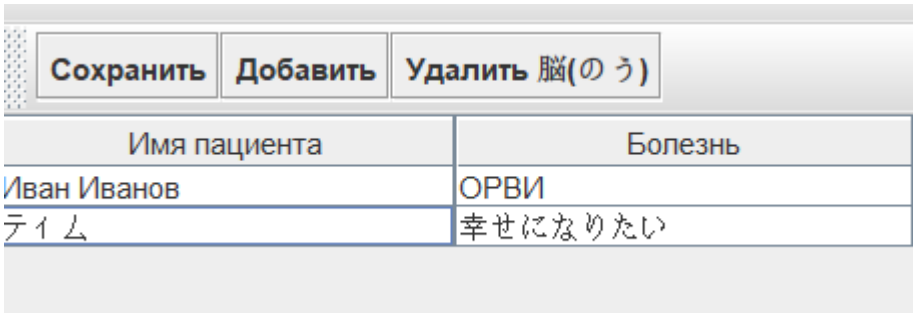


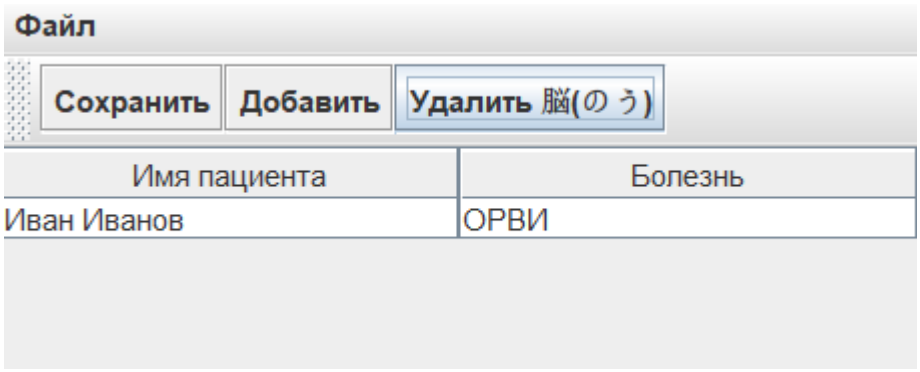
JComboBox:



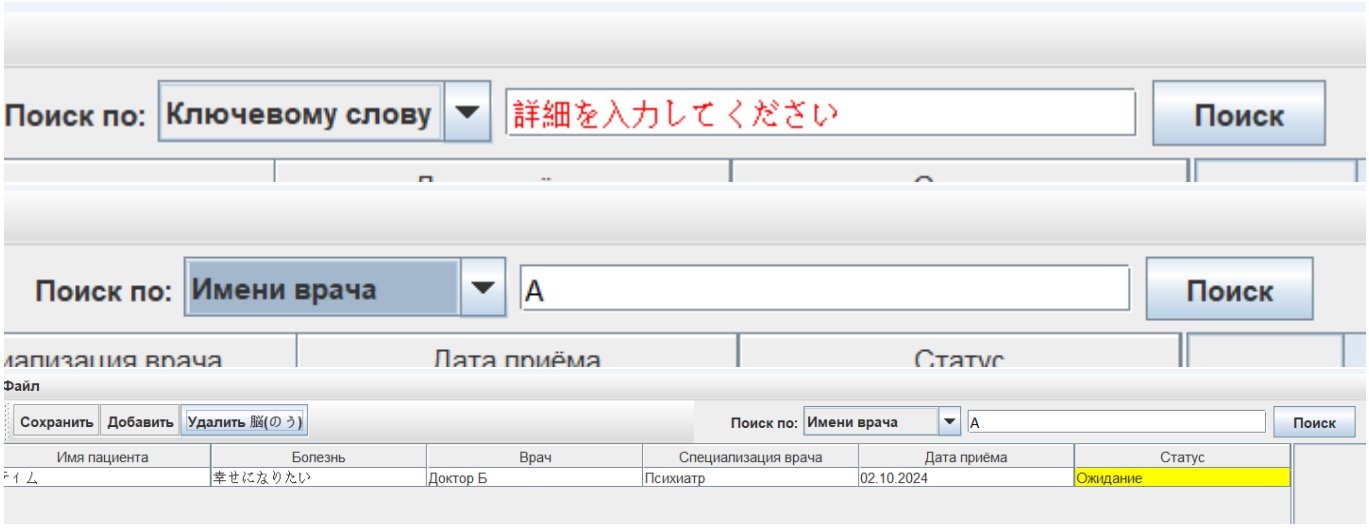
Работоспособность слушателей:

deleteButton:



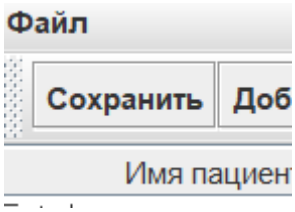


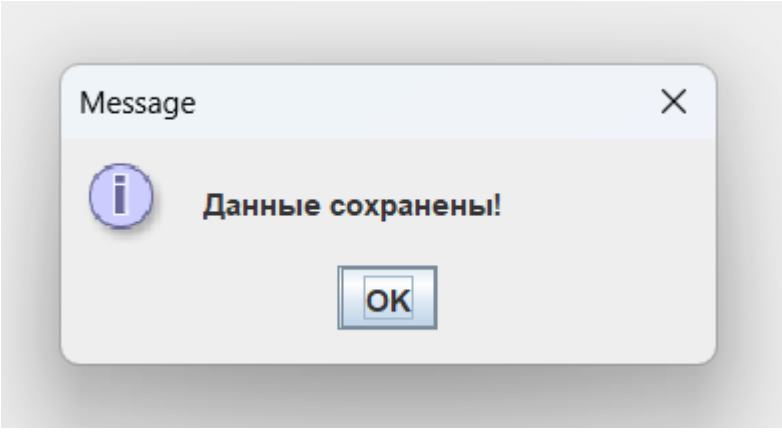
searchField && searchButton:



// работает как поиск по ключевому слову, так и изменение поля поиска при начале ввода

saveButton:

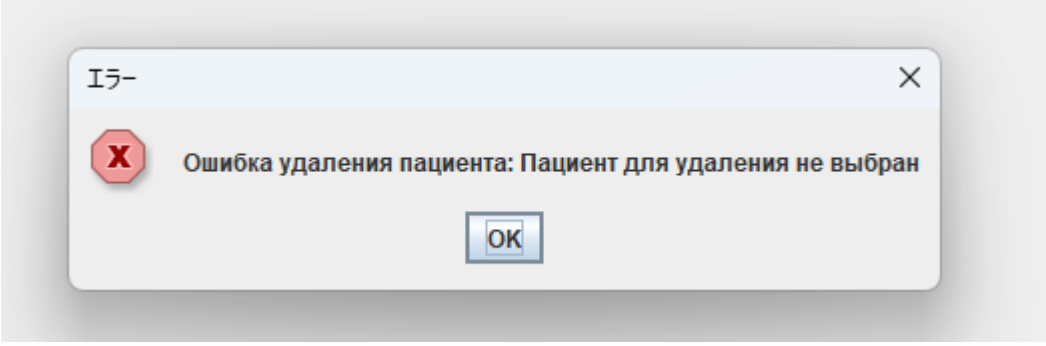




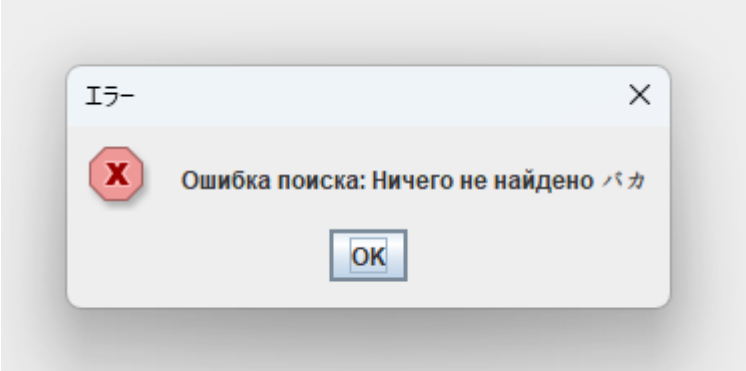
// только вывод сообщения

Работоспособность исключений:

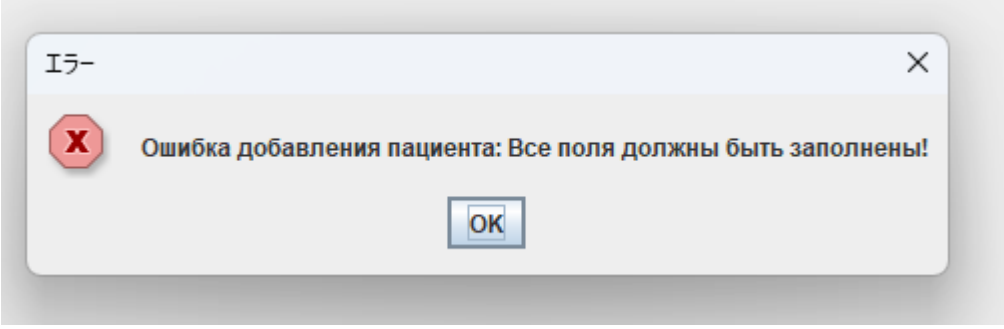
// deleteButton



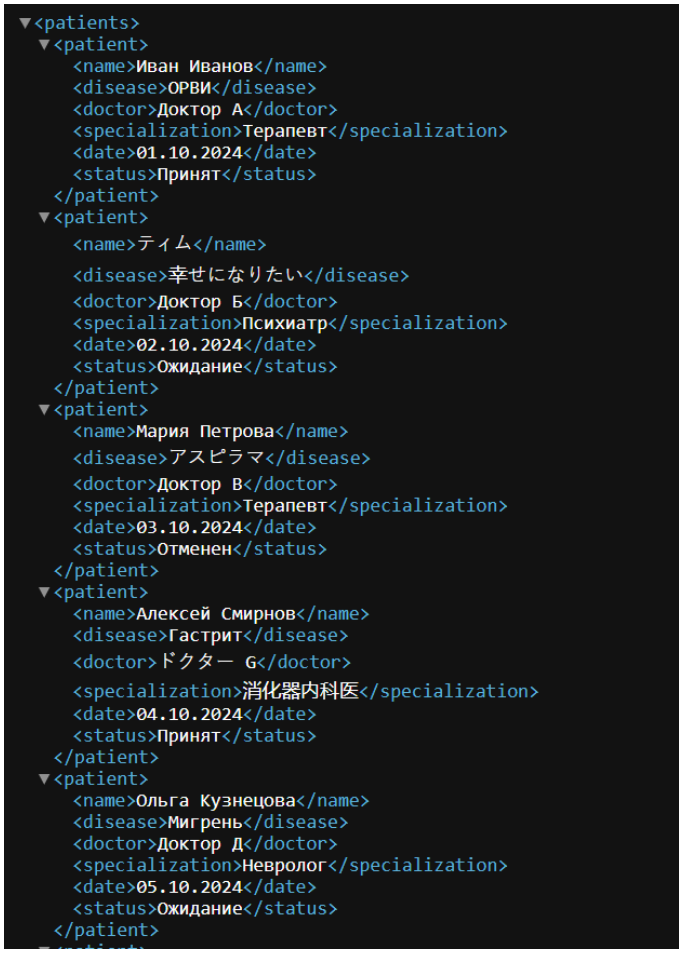
// searchButton

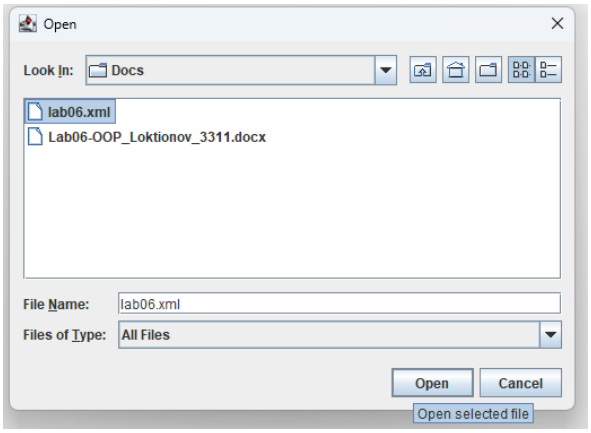


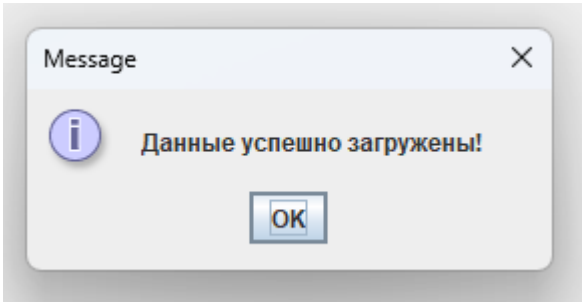
// addButton

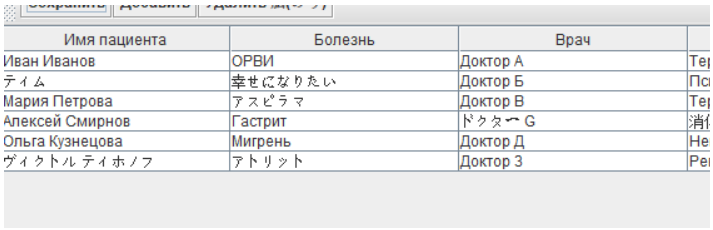


Работоспособность загрузки/выгрузки данных из XML файла:

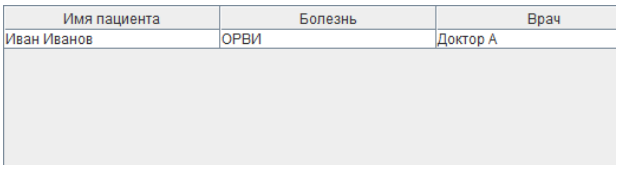




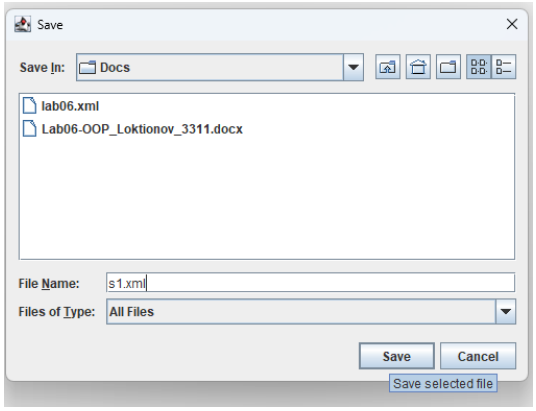


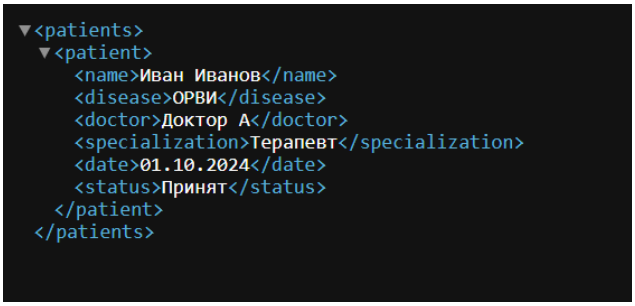


Имя пациента	Болезнь	Врач	
Иван Иванов	ОРВИ	Доктор А	Тер
ティム	幸せになりたい	Доктор Б	Пси
Мария Петрова	アスピラマ	Доктор В	Тер
Алексей Смирнов	Гастрит	ドクター G	消化
Ольга Кузнецова	Мигрень	Доктор Д	Не
ヴィクトル ティホノフ	アトリット	Доктор З	Ре

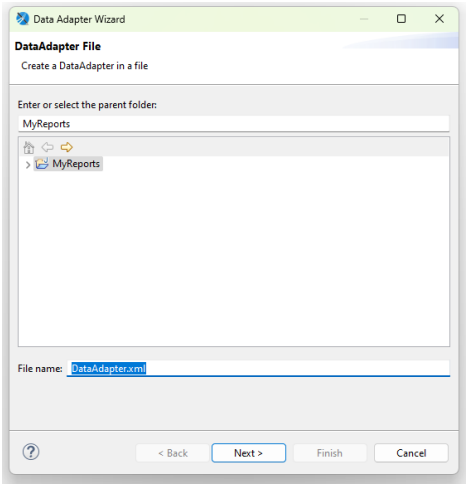


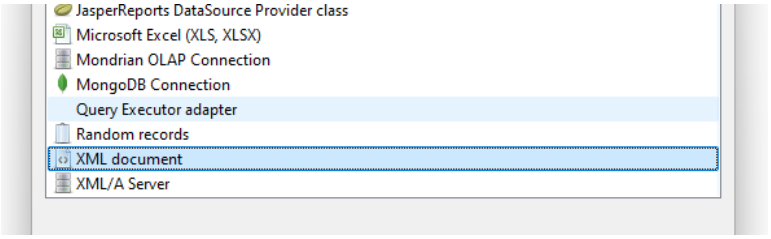
Имя пациента	Болезнь	Врач
Иван Иванов	ОРВИ	Доктор А

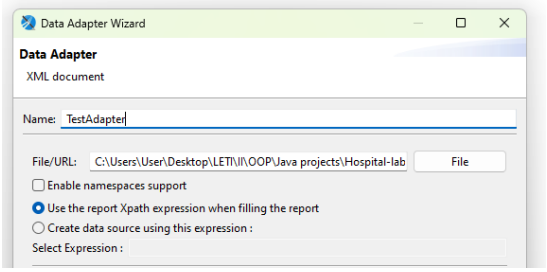


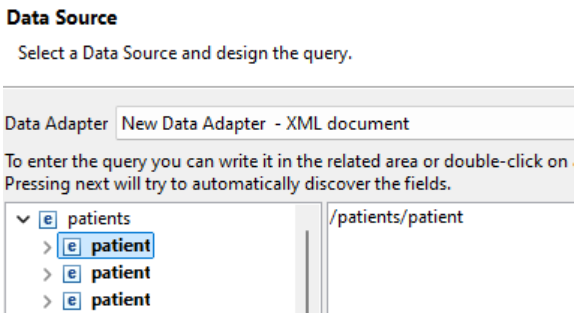


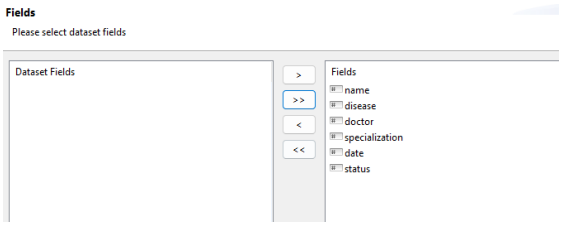
Создание .jrxml файла (шаблона)

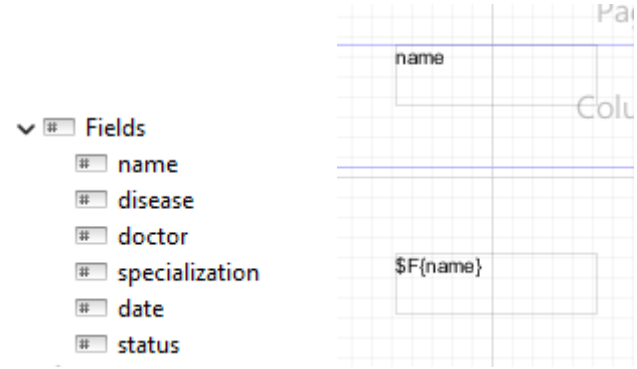












Сгенерированные шаблоны:

ClinicAPP PDF format ^_^

name	disease	doctor	specialization	date	status
Ivan Ivanov	Acute Respiratory Viral Infection	Doctor A	Therapist	01.10.2024	Accepted
Tim	Depression	Doctor B	Psychiatrist	02.10.2024	Waiting
Maria Petrova	Asthma	Doctor C	Therapist	03.10.2024	Canceled
Aleksei Smirnov	Gastritis	Doctor D	Gastroenterologist	04.10.2024	Accepted
Olga Kuznetsova	Migraine	Doctor E	Neurologist	05.10.2024	Waiting
Victor Tikhonov	Arthritis	Doctor F	Rheumatologist	08.10.2024	Accepted
Anna Sidorova	Pneumonia	Doctor G	Pulmonologist	09.10.2024	Waiting
Maksim Nikiforov	Hypertension	Doctor H	Cardiologist	10.10.2024	Accepted
Lidia Krylova	Diabetes	Doctor I	Endocrinologist	11.10.2024	Canceled
Sergey Pavlov	Lung Cancer	Doctor J	Oncologist	12.10.2024	Accepted
Ekaterina Morozova	Osteochondrosis	Doctor K	Orthopedist	13.10.2024	Waiting
Nikolay Fomin	Pancreatitis	Doctor L	Gastroenterologist	14.10.2024	Accepted
Vera Lebedeva	Insomnia	Doctor M	Neurologist	15.10.2024	Waiting
Oleg Sokolov	Allergy	Doctor N	Allergist	16.10.2024	Accepted
Nadezhda Saveleva	Tonsillitis	Doctor O	Otolaryngologist	17.10.2024	Canceled
Yuriy Grishin	Kidney Disease	Doctor P	Nephrologist	18.10.2024	Accepted
Marina Sokolova	Cholecystitis	Doctor Q	Surgeon	19.10.2024	Waiting
Ilya Borisov	Glaucoma	Doctor R	Ophthalmologist	20.10.2024	Accepted

ClinicApp HTML

name	disease	doctor	specialization	date	status
Ivan Ivanov	Acute Respiratory Viral Infection	Doctor A	Therapist	01.10.2024	Accepted
Tim	Depression	Doctor B	Psychiatrist	02.10.2024	Waiting
Maria Petrova	Asthma	Doctor C	Therapist	03.10.2024	Canceled
Aleksei Smirnov	Gastritis	Doctor D	Gastroenterologist	04.10.2024	Accepted
Olga Kuznetsova	Migraine	Doctor E	Neurologist	05.10.2024	Waiting
Victor Tikhonov	Arthritis	Doctor F	Rheumatologist	08.10.2024	Accepted
Anna Sidorova	Pneumonia	Doctor G	Pulmonologist	09.10.2024	Waiting
Maksim Nikiforov	Hypertension	Doctor H	Cardiologist	10.10.2024	Accepted
Lidia Krylova	Diabetes	Doctor I	Endocrinologist	11.10.2024	Canceled
Sergey Pavlov	Lung Cancer	Doctor J	Oncologist	12.10.2024	Accepted
Ekaterina Morozova	Osteochondrosis	Doctor K	Orthopedist	13.10.2024	Waiting
Nikolay Fomin	Pancreatitis	Doctor L	Gastroenterologist	14.10.2024	Accepted
Vera Lebedeva	Insomnia	Doctor M	Neurologist	15.10.2024	Waiting
Oleg Sokolov	Allergy	Doctor N	Allergist	16.10.2024	Accepted
Nadezhda Saveleva	Tonsillitis	Doctor O	Otolaryngologist	17.10.2024	Canceled
Yuriy Grishin	Kidney Disease	Doctor P	Nephrologist	18.10.2024	Accepted
Marina Sokolova	Cholecystitis	Doctor Q	Surgeon	19.10.2024	Waiting
Ilya Borisov	Glaucoma	Doctor R	Ophthalmologist	20.10.2024	Accepted

Для лабораторной №8:

Исходный .xml и .html файл:


mainReport.xml


report.html

Итоговый .xml .html файл:


mainReport.xml


report.html

Если открыт .pdf:

Исходный и итоговый .xml файл:

```
<patients>
<patient>
<name>Ivan Ivanov</name>
<disease>Acute Respiratory Viral Infection</disease>
<doctor>Doctor A</doctor>
<specialization>Therapist</specialization>
<date>01.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Tim</name>
<disease>Depression</disease>
<doctor>Doctor B</doctor>
<specialization>Psychiatrist</specialization>
<date>02.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Maria Petrova</name>
<disease>Asthma</disease>
<doctor>Doctor C</doctor>
<specialization>Therapist</specialization>
<date>03.10.2024</date>
<status>Canceled</status>
</patient>
<patient>
<name>Aleksei Smirnov</name>
<disease>Gastritis</disease>
```

```
<patients>
<patient>
<name>Ivan Ivanov</name>
<disease>Acute Respiratory Viral Infection</disease>
<doctor>Doctor A</doctor>
<specialization>Therapist</specialization>
<date>01.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Tim</name>
<disease>Depression</disease>
<doctor>Doctor B</doctor>
<specialization>Psychiatrist</specialization>
<date>02.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Maria Petrova</name>
<disease>Asthma</disease>
<doctor>Doctor C</doctor>
<specialization>Therapist</specialization>
<date>03.10.2024</date>
<status>Canceled</status>
</patient>
<patient>
<name>Aleksei Smirnov</name>
<disease>Gastritis</disease>
```

```
<doctor>Doctor D</doctor>
<specialization>Gastroenterologist</specialization>
<date>04.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Olga Kuznetsova</name>
<disease>Migraine</disease>
<doctor>Doctor E</doctor>
<specialization>Neurologist</specialization>
<date>05.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Victor Tikhonov</name>
<disease>Arthritis</disease>
<doctor>Doctor F</doctor>
<specialization>Rheumatologist</specialization>
<date>08.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Anna Sidorova</name>
<disease>Pneumonia</disease>
<doctor>Doctor G</doctor>
<specialization>Pulmonologist</specialization>
<date>09.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Maksim Nikiforov</name>
<disease>Hypertension</disease>
<doctor>Doctor H</doctor>
<specialization>Cardiologist</specialization>
<date>10.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Lidia Krylova</name>
<disease>Diabetes</disease>
```


```
<doctor>Doctor D</doctor>
<specialization>Gastroenterologist</specialization>
<date>04.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Olga Kuznetsova</name>
<disease>Migraine</disease>
<doctor>Doctor E</doctor>
<specialization>Neurologist</specialization>
<date>05.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Victor Tikhonov</name>
<disease>Arthritis</disease>
<doctor>Doctor F</doctor>
<specialization>Rheumatologist</specialization>
<date>08.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Anna Sidorova</name>
<disease>Pneumonia</disease>
<doctor>Doctor G</doctor>
<specialization>Pulmonologist</specialization>
<date>09.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Maksim Nikiforov</name>
<disease>Hypertension</disease>
<doctor>Doctor H</doctor>
<specialization>Cardiologist</specialization>
<date>10.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Lidia Krylova</name>
<disease>Diabetes</disease>
```

```
<doctor>Doctor I</doctor>
<specialization>Endocrinologist</specialization>
<date>11.10.2024</date>
<status>Canceled</status>
</patient>
<patient>
<name>Sergey Pavlov</name>
<disease>Lung Cancer</disease>
<doctor>Doctor J</doctor>
<specialization>Oncologist</specialization>
<date>12.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Ekaterina Morozova</name>
<disease>Osteochondrosis</disease>
<doctor>Doctor K</doctor>
<specialization>Orthopedist</specialization>
<date>13.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Nikolay Fomin</name>
<disease>Pancreatitis</disease>
<doctor>Doctor L</doctor>
<specialization>Gastroenterologist</specialization
>
<date>14.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Vera Lebedeva</name>
<disease>Insomnia</disease>
<doctor>Doctor M</doctor>
<specialization>Neurologist</specialization>
<date>15.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Oleg Sokolov</name>
<disease>Allergy</disease>
```


```
<doctor>Doctor I</doctor>
<specialization>Endocrinologist</specialization>
<date>11.10.2024</date>
<status>Canceled</status>
</patient>
<patient>
<name>Sergey Pavlov</name>
<disease>Lung Cancer</disease>
<doctor>Doctor J</doctor>
<specialization>Oncologist</specialization>
<date>12.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Ekaterina Morozova</name>
<disease>Osteochondrosis</disease>
<doctor>Doctor K</doctor>
<specialization>Orthopedist</specialization>
<date>13.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Nikolay Fomin</name>
<disease>Pancreatitis</disease>
<doctor>Doctor L</doctor>
<specialization>Gastroenterologist</specialization
>
<date>14.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Vera Lebedeva</name>
<disease>Insomnia</disease>
<doctor>Doctor M</doctor>
<specialization>Neurologist</specialization>
<date>15.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Oleg Sokolov</name>
<disease>Allergy</disease>
```

```
<doctor>Doctor N</doctor>
<specialization>Allergist</specialization>
<date>16.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Nadezhda Saveleva</name>
<disease>Tonsillitis</disease>
<doctor>Doctor O</doctor>
<specialization>Otolaryngologist</specialization>
<date>17.10.2024</date>
<status>Canceled</status>
</patient>
<patient>
<name>Yuriy Grishin</name>
<disease>Kidney Disease</disease>
<doctor>Doctor P</doctor>
<specialization>Nephrologist</specialization>
<date>18.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Marina Sokolova</name>
<disease>Cholecystitis</disease>
<doctor>Doctor Q</doctor>
<specialization>Surgeon</specialization>
<date>19.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Ilya Borisov</name>
<disease>Glaucoma</disease>
<doctor>Doctor R</doctor>
<specialization>Ophthalmologist</specialization>
<date>20.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Alyona Gavrilova</name>
<disease>Obesity</disease>
```

```
<doctor>Doctor N</doctor>
<specialization>Allergist</specialization>
<date>16.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Nadezhda Saveleva</name>
<disease>Tonsillitis</disease>
<doctor>Doctor O</doctor>
<specialization>Otolaryngologist</specialization>
<date>17.10.2024</date>
<status>Canceled</status>
</patient>
<patient>
<name>Yuriy Grishin</name>
<disease>Kidney Disease</disease>
<doctor>Doctor P</doctor>
<specialization>Nephrologist</specialization>
<date>18.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Marina Sokolova</name>
<disease>Cholecystitis</disease>
<doctor>Doctor Q</doctor>
<specialization>Surgeon</specialization>
<date>19.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Ilya Borisov</name>
<disease>Glaucoma</disease>
<doctor>Doctor R</doctor>
<specialization>Ophthalmologist</specialization>
<date>20.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Alyona Gavrilova</name>
<disease>Obesity</disease>
```

```
<doctor>Doctor S</doctor>
<specialization>Dietitian</specialization>
<date>21.10.2024</date>
<status>Canceled</status>
</patient>
<patient>
<name>Stanislav Popov</name>
<disease>Epilepsy</disease>
<doctor>Doctor T</doctor>
<specialization>Neurologist</specialization>
<date>22.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Elena Yakovleva</name>
<disease>Anemia</disease>
<doctor>Doctor U</doctor>
<specialization>Hematologist</specialization>
<date>23.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Dmitry Kravtsov</name>
<disease>Chronic Stress</disease>
<doctor>Doctor V</doctor>
<specialization>Psychologist</specialization>
<date>24.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Oksana Chernyshova</name>
<disease>Varicose Veins</disease>
<doctor>Doctor W</doctor>
<specialization>Surgeon</specialization>
<date>25.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Andrey Zuev</name>
<disease>Stomach Cancer</disease>
```


```
<doctor>Doctor S</doctor>
<specialization>Dietitian</specialization>
<date>21.10.2024</date>
<status>Canceled</status>
</patient>
<patient>
<name>Stanislav Popov</name>
<disease>Epilepsy</disease>
<doctor>Doctor T</doctor>
<specialization>Neurologist</specialization>
<date>22.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Elena Yakovleva</name>
<disease>Anemia</disease>
<doctor>Doctor U</doctor>
<specialization>Hematologist</specialization>
<date>23.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Dmitry Kravtsov</name>
<disease>Chronic Stress</disease>
<doctor>Doctor V</doctor>
<specialization>Psychologist</specialization>
<date>24.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>Oksana Chernyshova</name>
<disease>Varicose Veins</disease>
<doctor>Doctor W</doctor>
<specialization>Surgeon</specialization>
<date>25.10.2024</date>
<status>Waiting</status>
</patient>
<patient>
<name>Andrey Zuev</name>
<disease>Stomach Cancer</disease>
```

```
<doctor>Doctor X</doctor>
<specialization>Oncologist</specialization>
<date>26.10.2024</date>
<status>Accepted</status>
</patient>
<patient>
<name>1</name>
<disease>2</disease>
<doctor>3</doctor>
<specialization>4</specialization>
<date>5</date>
<status>6</status>
</patient>
</patients>
```

```
<doctor>Doctor X</doctor>
<specialization>Oncologist</specialization>
<date>26.10.2024</date>
<status>Accepted</status>
</patient>
</patients>
```

Изменения в .html файл:

ClinicApp HTML

name	disease	doctor	specialization	date	status
1	2	3	4	5	6

ClinicApp HTML

name	disease	doctor	specialization	date	status

Перечень используемых типов сообщений, которые выводятся в логфайл:
>> DEBUG, INFO, WARN

Конфигурационный файл log4j.properties.

```
# Уровень логирования для корневого логгера
log4j.rootLogger=DEBUG, console, file
# log4j.rootLogger=INFO, console, file
# log4j.rootLogger=WARN, console, file

# Настройка уровня логирования для JasperReports
log4j.logger.net.sf.jasperreports=INFO
log4j.logger.org.apache.commons=INFO

# Консольный аппендер
# указывает, что логи будут выводиться в консоль
log4j.appender.console=org.apache.log4j.ConsoleAppender
# задаёт формат сообщений в логах
# (PatternLayout позволяет задавать формат вывода логов с помощью шаблонов
(как %d, %t, %m и т.д.).
log4j.appender.console.layout=org.apache.log4j.PatternLayout
# определяет, как именно будут выглядеть логи. Здесь шаблон:
# %d{ISO8601} — дата и время в формате ISO 8601.
# [%t] — имя потока (thread), в котором было создано сообщение.
# %-5p — уровень сообщения (DEBUG, INFO и т. д.), с минимальной шириной 5
символов.
# %c — полное имя класса, откуда вызвано логирование.
# %m — само сообщение.
# %n — перенос строки.
log4j.appender.console.layout.ConversionPattern=%d{ISO8601} [%t] %-5p %c -
%m%n

# Файловый аппендер
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=logs/application.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ISO8601} [%t] %-5p %c - %m%n
```

Режим DEBUG:

```
2024-11-20 01:06:37,949 [Thread-0] INFO  GUI - Первый поток: Начата загрузка данных.
2024-11-20 01:06:37,950 [Thread-0] INFO  GUI - Первый поток: Данные успешно загружены.
2024-11-20 01:06:37,950 [Thread-2] DEBUG GUI - Третий поток: Ожидание завершения формирования
XML.
2024-11-20 01:06:37,950 [Thread-1] DEBUG GUI - Второй поток: Ожидание завершения загрузки
данных.
2024-11-20 01:06:37,950 [Thread-1] INFO  GUI - Второй поток: Начато формирование XML.
2024-11-20 01:06:37,983 [Thread-1] INFO  GUI - Второй поток: XML успешно сохранён.
2024-11-20 01:06:37,985 [Thread-2] INFO  GUI - Третий поток: Начата генерация HTML-отчёта.
2024-11-20 01:06:39,112 [Thread-2] INFO  GUI - Третий поток: HTML-отчёт успешно создан.
```

Режим INFO:

```
2024-11-20 01:07:45,797 [Thread-0] INFO  GUI - Первый поток: Начата загрузка данных.
2024-11-20 01:07:45,797 [Thread-0] INFO  GUI - Первый поток: Данные успешно загружены.
2024-11-20 01:07:45,797 [Thread-1] INFO  GUI - Второй поток: Начато формирование XML.
2024-11-20 01:07:45,826 [Thread-1] INFO  GUI - Второй поток: XML успешно сохранён.
2024-11-20 01:07:45,826 [Thread-2] INFO  GUI - Третий поток: Начата генерация HTML-отчёта.
2024-11-20 01:07:46,987 [Thread-2] INFO  GUI - Третий поток: HTML-отчёт успешно создан.
```

Режим WARN:

```
java.lang.RuntimeException: Файл XML данных не найден: src/docs/mainRepor.xml
    at ReportGenerator.generateReport(ReportGenerator.java:51)
    at ReportGenerator.generateHtmlReport(ReportGenerator.java:30)
    at GUI.lambda$startMultithreading$3(GUI.java:237)
    at java.base/java.lang.Thread.run(Thread.java:1570)
```

Ссылки:

>> репозиторий:

https://github.com/iconlti/ltprojects/tree/master/oop/java%20projects/hospital-lab010_mav

>> видео отчет:
Google Disk:

https://drive.google.com/drive/folders/1yrkoxpkxttlnzto8e_p50spojcbnft9a?usp=sharing

Текст программы:

// ClinicApp.java

```java
import javax.swing.SwingUtilities;
import org.apache.log4j.PropertyConfigurator;

/**
 * Основной класс приложения, содержащий точку входа.
 * @author Tim Loktionov 3311
 * @version 1.00
 */
public class ClinicApp {
    /**
     * Главный метод запуска программы.
     * Вызывает метод создания и отображения интерфейса.
     *
     * @param args аргументы командной строки (не используются).
     */
    public static void main(String[] args) {
        PropertyConfigurator.configure("src/log4j.properties"); // просто явно указываем, иначе не
понимает
        SwingUtilities.invokeLater(() -> new GUI().buildAndShowGUI());
    }
}
```

// GUI.java

```java
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.TreeMap;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellRenderer;
import org.apache.log4j.Logger;

/**
 * Основной класс, отвечающий за построение интерфейса приложения "Клиника".
 * Приложение предназначено для управления списком пациентов.
 * Включает добавление, удаление пациентов, сохранение данных и поиск по имени.
 */
public class GUI {
    // логирование
    private static final Logger log = Logger.getLogger(GUI.class);
    // Объявления компонентов для потока
    private static final Object monitor = new Object(); // будет использоваться для синхронизации потоков
    private static boolean isXmlGenerated; // флаг, показывающий завершил ли свою работу второй поток
    private static boolean isDataLoaded;

    // Объявление компонентов
    JFrame frame;
    JMenuBar menuBar;
    JMenu fileMenu;
    JMenuItem openItem, saveItem, exportPdfItem, exportHtmlItem; // Добавлен exportItem
    JToolBar toolBar;
    JButton saveButton, addButton, deleteButton;
    JButton searchButton;
    JButton startTreadsButton;
    JComboBox<String> searchType;
    JComboBox<String> sortType;
    JTextField searchField;
    JTable dataTable;
    private DefaultTableModel tableModel;
    static File openedFile;


    /**
     * Метод для построения и отображения графического интерфейса.
     * Создает основное окно приложения, меню, панель инструментов,
     * элементы для поиска и таблицу данных.
     */
    public void buildAndShowGUI() {

        // открытие файла
        String xmlFilePath = "src/docs/mainReport.xml";
        openedFile = new File(xmlFilePath);

        frame = new JFrame("Клиника - Список пациентов");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(1200, 640);

        // Создание меню
        menuBar = new JMenuBar();
        fileMenu = new JMenu("Файл");
        openItem = new JMenuItem("Открыть");
        saveItem = new JMenuItem("Сохранить");
        exportPdfItem = new JMenuItem("Экспорт отчета в PDF"); // Добавлен пункт "Экспорт отчета"
        exportHtmlItem = new JMenuItem("Экспорт отчета в HTML");
        fileMenu.add(openItem);
        fileMenu.add(saveItem);
        fileMenu.add(exportPdfItem); // Добавляем в меню "Файл"
        fileMenu.add(exportHtmlItem);
        menuBar.add(fileMenu);
        frame.setJMenuBar(menuBar);

        // Панель инструментов
        toolBar = new JToolBar();
        saveButton = new JButton("Сохранить");
        addButton = new JButton("Добавить");
        deleteButton = new JButton("Удалить 脳(のう)");
        toolBar.add(saveButton);
        toolBar.add(addButton);
        toolBar.add(deleteButton);

        // Панель для потока
```

```
        startTreadsButton = new JButton("Поток");
        // Отдельно для потока
        startTreadsButton.addActionListener(e -> startMultithreading(tableModel));
        JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        buttonPanel.add(startTreadsButton);
        frame.add(buttonPanel, BorderLayout.SOUTH);

        // Панель поиска
        JPanel searchPanel = new JPanel();
        searchType = new JComboBox<>(new String[]{"Ключевому слову", "Имени пациента", "Имени врача"});
        searchField = new JTextField(25);
        searchButton = new JButton("Поиск");

        // Текст подсказка
        String placeholder = "詳細を入力してください";
        searchField.setText(placeholder);
        searchField.setForeground(Color.RED); // цвет текста
        searchField.addFocusListener(Listeners.getSearchFieldFocusListener(searchField, placeholder));

        searchPanel.add(new JLabel("Поиск по:"));
        searchPanel.add(searchType);
        searchPanel.add(searchField);
        searchPanel.add(searchButton);

        // Контейнер для обеих частей
        JPanel topPanel = new JPanel(new GridLayout(1, 2)); // Одна строка, два столбца
        topPanel.add(toolBar);
        topPanel.add(searchPanel);
        frame.add(topPanel, BorderLayout.NORTH);

        // Таблица с данными
        String[] columns = {"Имя пациента", "Болезнь", "Врач", "Специализация врача", "Дата приёма",
"Статус"};
        tableModel = new DefaultTableModel(new Object[][]{}, columns);

        dataTable = new JTable(tableModel) {
            @Override
            public Component prepareRenderer(TableCellRenderer renderer, int row, int column) {
                Component cell = super.prepareRenderer(renderer, row, column);
                // Проверяем, что это колонка "Статус" (индекс 5)
                if (column == 5) {
                    String status = (String) getValueAt(row, column);
                    switch (status) {
                        case "Accepted":
                            cell.setBackground(Color.GREEN);
                            break;
                        case "Waiting":
                            cell.setBackground(Color.YELLOW);
                            break;
                        case "Canceled":
                            cell.setBackground(Color.RED);
                            break;
                        default:
                            cell.setBackground(Color.WHITE); // Фон для остальных статусов
                            break;
                    }
                } else {
                    cell.setBackground(Color.WHITE); // Для остальных колонок устанавливаем белый фон
                }
                return cell;
            }
        };
        JScrollPane tableScrollPane = new JScrollPane(dataTable);
        frame.add(tableScrollPane, BorderLayout.CENTER);

        // Сортировка
        sortType = new JComboBox<>(new String[]{"По имени", "По дате"});
        frame.add(sortType, BorderLayout.EAST);

        // Автозаполнение таблицы
        if (openedFile.exists()) {
            XMLfile.loadFromXML(tableModel, openedFile);
        } else {
            JOptionPane.showMessageDialog(null, "Файл данных не найден: " + xmlFilePath, "Ошибка",
JOptionPane.ERROR_MESSAGE);
        }

    // Слушатели (Action)

        // Поиск
```

```
        searchButton.addActionListener(Listeners.getSearchListener(dataTable, searchField, searchType,
frame));
        // Кнопки
        saveButton.addActionListener(Listeners.getSaveDataListener(frame, tableModel));
        addButton.addActionListener(Listeners.getAddPatientListener(tableModel));
        deleteButton.addActionListener(Listeners.getDeletePatientListener(tableModel, dataTable, frame));
        // Сортировка
        sortType.addActionListener(Listeners.getSortTypeActionListener(sortType, frame));
        // Слушатели для меню
        openItem.addActionListener(Listeners.getLoadDataListener(tableModel, frame));
        saveItem.addActionListener(Listeners.getSaveToPathDataListener(frame, tableModel));

    //

        // Подключение слушателя экспорта PDF-отчёта
        exportPdfItem.addActionListener(
                Listeners.getExportPdfReportListener(
                        frame,
                        "src/docs/ClinicPDF.jrxml",
                        "src/docs/report.pdf"
                )
        );
        // Подключение слушателя экспорта HTML-отчёта
        exportHtmlItem.addActionListener(
                Listeners.getExportHtmlReportListener(
                        frame,
                        "src/docs/ClinicHTML.jrxml",
                        "src/docs/report.html"
                )
        );


        // Визуализация
        frame.setVisible(true);
    }

    private void startMultithreading(DefaultTableModel tableModel) {
        // вспомогательный логичекий флаг для более корректной синхронизации
        isXmlGenerated = false;
        isDataLoaded = false;

        // первый поток
        Thread dataLoader = new Thread(() -> { // -> -- лямбда выажение для Runnable
            // блокирует (но "не заставляет их ждать") объект monitor, чтобы другие потоки не могли его
использовать
            synchronized (monitor) { // по сути это именно "захват", а не сама блокировка
                log.info("Первый поток: Начата загрузка данных.");
                isDataLoaded = true;
                log.info("Первый поток: Данные успешно загружены.");
                monitor.notifyAll(); // уведомляет другие потоки, что теперь можно использовать этот поток
            }
        });

        // второй поток
        Thread xmlEditor = new Thread(() -> {
            synchronized (monitor) {
                log.debug("Второй поток: Ожидание завершения загрузки данных.");
                try {
                    while (!isDataLoaded) {
        // останавливает выполнение потока (как раз таки "заставляет их ждать"), пока первый не вызовет
notifyAll
                        monitor.wait();
                    }
                    log.info("Второй поток: Начато формирование XML.");
                    XMLfile.saveToXML(tableModel, new File("src/docs/mainReport.xml"));
                    log.info("Второй поток: XML успешно сохранён.");
                    isXmlGenerated = true;
                    monitor.notifyAll();
                } catch (Exception ex) {
                    log.warn("Второй поток: Ошибка при формировании XML.", ex);
                }
            }
        });

        // третий поток
        Thread reportGenerator = new Thread(() -> {
            synchronized (monitor) {
                log.debug("Третий поток: Ожидание завершения формирования XML.");
                try {
                    while (!isXmlGenerated) {
```

```
                        monitor.wait();
                    }
                    log.info("Третий поток: Начата генерация HTML-отчёта.");
                    ReportGenerator generator = new ReportGenerator();
                    generator.generateHtmlReport(
                            "src/docs/ClinicHTML.jrxml",
                            "src/docs/mainReport.xml",
                            "src/docs/report.html"
                    );
                    log.info("Третий поток: HTML-отчёт успешно создан.");
                } catch (Exception ex) {
                    log.warn("Третий поток: Ошибка при генерации HTML-отчёта.", ex);
                }
            }
        });

        dataLoader.start();
        xmlEditor.start();
        reportGenerator.start();

    }
}
```

```
// Listeners.java
```

```java
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

/**
 * Класс, содержащий слушатели для различных действий в приложении.
 */
public class Listeners {

    /**
     * Создает слушатель для добавления нового пациента.
     *
     * @param tableModel модель таблицы, в которую будет добавлен пациент
     * @return ActionListener для добавления нового пациента
     */
    public static ActionListener getAddPatientListener(DefaultTableModel tableModel) {
        return e -> {
            try {
                String name = JOptionPane.showInputDialog("Введите имя пациента:");
                String disease = JOptionPane.showInputDialog("Введите название болезни:");
                String doctor = JOptionPane.showInputDialog("Введите имя врача:");
                String specialization = JOptionPane.showInputDialog("Введите специализацию врача:");
                String date = JOptionPane.showInputDialog("Введите дату приёма:");
                String status = JOptionPane.showInputDialog("Введите статус:");

                // Проверка, что поля не пустые и не равны null, так как сама программа добавляет пустые
строки
                if (name != null && !name.trim().isEmpty() &&
                        disease != null && !disease.trim().isEmpty() &&
                        doctor != null && !doctor.trim().isEmpty() &&
                        specialization != null && !specialization.trim().isEmpty() &&
                        date != null && !date.trim().isEmpty() &&
                        status != null && !status.trim().isEmpty()) {

                    tableModel.addRow(new Object[]{name, disease, doctor, specialization, date, status});
                } else {
                    throw new IllegalArgumentException("Все поля должны быть заполнены!");
                }
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Ошибка добавления пациента: "
                        + ex.getMessage(), "   エラー", JOptionPane.ERROR_MESSAGE);
            }
        };
    }



    /**
     * Создает слушатель для удаления пациента.
     *
     * @param tableModel модель таблицы, из которой будет удален пациент
     * @param dataTable  таблица, отображающая пациентов
     * @param frame      окно, в котором отображаются сообщения
     * @return ActionListener для удаления пациента
     */
    public static ActionListener getDeletePatientListener(DefaultTableModel tableModel, JTable dataTable,
JFrame frame) {
        return e -> {
            try {
                int selectedRow = dataTable.getSelectedRow();
                if (selectedRow != -1) {
                    tableModel.removeRow(selectedRow);
                } else {
                    throw new IllegalArgumentException("Пациент для удаления не выбран");
                }
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(frame, "Ошибка удаления пациента: " + ex.getMessage(),
                        "   エラー", JOptionPane.ERROR_MESSAGE);
            }
        };
    }



    /**
```

```
     * Создает слушатель для поиска пациента по имени или врачу.
     *
     * @param dataTable   таблица, в которой производится поиск
     * @param searchField поле ввода для текста поиска
     * @param searchType  комбобокс для выбора типа поиска
     * @param frame       окно, в котором отображаются сообщения
     * @return ActionListener для поиска пациента
     */
    public static ActionListener getSearchListener(JTable dataTable, JTextField searchField,
                                                   JComboBox<String> searchType, JFrame frame) {
        return e -> {
            try {
                String searchText = searchField.getText().toLowerCase();
                int searchColumn = searchType.getSelectedIndex() == 1 ? 0 : 2; // 0 - имя пациента, 1 -
имя врача

                boolean found = false;
                for (int i = 0; i < dataTable.getRowCount(); i++) {
                    String value = dataTable.getValueAt(i, searchColumn).toString().toLowerCase();
                    if (value.contains(searchText)) {
                        dataTable.setRowSelectionInterval(i, i);
                        found = true;
                        break;
                    }
                }
                if (!found) {
                    throw new IllegalArgumentException("Ничего не найдено バカ");
                }
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(frame, "Ошибка поиска: " + ex.getMessage(),
                        "   エラー", JOptionPane.ERROR_MESSAGE);
            }

        };
    }

    /**
     * Создает слушатель для сортировки пациентов.
     *
     * @param sortType комбобокс для выбора типа сортировки
     * @param frame    окно, в котором отображаются сообщения
     * @return ActionListener для сортировки пациентов
     */
    public static ActionListener getSortTypeActionListener(JComboBox<String> sortType, JFrame frame) {
        return e -> {
            String selectedSort = (String) sortType.getSelectedItem();
            if ("По имени".equals(selectedSort)) {
                JOptionPane.showMessageDialog(frame, "Сортировка по имени");
            } else if ("По дате".equals(selectedSort)) {
                JOptionPane.showMessageDialog(frame, "Сортировка по дате");
            }
        };
    }

    /**
     * Создает слушатель для управления поведением поля поиска.
     *
     * @param searchField поле ввода для текста поиска
     * @param placeholder текст-заполнитель для поля поиска
     * @return FocusAdapter для управления поведением поля поиска
     */
    public static FocusAdapter getSearchFieldFocusListener(JTextField searchField, String placeholder) {
        return new FocusAdapter() {
            @Override
            public void focusGained(FocusEvent e) {
                if (searchField.getText().equals(placeholder)) {
                    searchField.setText("");
                    searchField.setForeground(Color.BLACK);
                }
            }

            @Override
            public void focusLost(FocusEvent e) {
                if (searchField.getText().isEmpty()) {
                    searchField.setForeground(Color.RED);
                    searchField.setText(placeholder);
                }
            }
        };
    }
```

```
    /**
     * Создает слушатель для выгрузки данных из файла
     *
     * @param tableModel ячейки таблицы
     * @param frame      окно, в котором отображаются сообщения
     */
    public static ActionListener getLoadDataListener(DefaultTableModel tableModel, JFrame frame) {
        return e -> {
            JFileChooser fileChooser = new JFileChooser(); // Окно для выбора файла
            int result = fileChooser.showOpenDialog(frame); // Открытие диалогового окна для выбора файла
            if (result == JFileChooser.APPROVE_OPTION) {
                GUI.openedFile = fileChooser.getSelectedFile(); // Получаем выбранный файл
                XMLfile.loadFromXML(tableModel, GUI.openedFile); // Загружаем данные из файла с помощью
XMLReader
                JOptionPane.showMessageDialog(frame, "Данные успешно загружены!"); // Показываем сообщение
об успехе
            }
        };
    }

    /**
     * Создает слушатель для сохранения данных в формате "сохранить как"
     *
     * @param tableModel ячейки таблицы
     * @param frame      окно, в котором отображаются сообщения
     */
    public static ActionListener getSaveToPathDataListener(JFrame frame, DefaultTableModel tableModel) {
        return e -> {
            JFileChooser fileChooser = new JFileChooser(); // Окно для выбора пути сохранения
            int result = fileChooser.showSaveDialog(frame); // Открытие диалогового окна для сохранения
файла
            if (result == JFileChooser.APPROVE_OPTION) {
                File file = fileChooser.getSelectedFile(); // Получаем выбранный файл
                XMLfile.saveToXML(tableModel, file); // Сохраняем данные в файл с помощью XMLWriter
                JOptionPane.showMessageDialog(frame, "Данные успешно сохранены!"); // Показываем сообщение
об успехе
            }
        };
    }


    /**
     * Создает слушатель для сохранения данных в тот же файл
     *
     * @param tableModel ячейки таблицы
     * @param frame      окно, в котором отображаются сообщения
     */
    public static ActionListener getSaveDataListener(JFrame frame, DefaultTableModel tableModel) {
        return e -> {
            if (GUI.openedFile != null) {
                try {
                    // Используем класс XMLfile для сохранения данных
                    XMLfile.saveToXML(tableModel, GUI.openedFile);
                    JOptionPane.showMessageDialog(frame, "Данные успешно сохранены в файл: " +
GUI.openedFile.getName());
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(frame, "Ошибка сохранения файла: " + ex.getMessage(),
                            "Ошибка", JOptionPane.ERROR_MESSAGE);
                }
            } else {
                JOptionPane.showMessageDialog(frame, "Файл для сохранения не загружен!",
                        "Ошибка", JOptionPane.ERROR_MESSAGE);
            }
        };
    }

    /**
     * Создает слушатель для экспорта отчета в PDF.
     *
     * @param frame          окно, в котором отображаются сообщения
     * @param reportPath     путь к файлу шаблона отчёта
     * @param outputFilePath путь, куда сохраняется сгенерированный отчет
     * @return ActionListener для экспорта отчета
     */
    public static ActionListener getExportPdfReportListener(JFrame frame, String reportPath, String
outputFilePath) {
        return e -> {
            try {
                String xmlFilePath = GUI.openedFile != null ? GUI.openedFile.getAbsolutePath() : null;
```

```
                if (xmlFilePath == null) {
                    JOptionPane.showMessageDialog(frame, "Сначала загрузите XML файл!", "Ошибка",
JOptionPane.ERROR_MESSAGE);
                    return;
                }

                // Генерация PDF отчета
                ReportGenerator generator = new ReportGenerator();
                generator.generatePdfReport(reportPath, xmlFilePath, outputFilePath);

                JOptionPane.showMessageDialog(frame, "PDF отчет успешно создан:\n" + outputFilePath,
"Успех", JOptionPane.INFORMATION_MESSAGE);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(frame, "Ошибка генерации PDF отчета:\n" + ex.getMessage(),
"Ошибка", JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            }
        };
    }

    /**
     * Создает слушатель для экспорта отчета в HTML.
     *
     * @param frame          окно, в котором отображаются сообщения
     * @param reportPath     путь к файлу шаблона отчёта
     * @param outputFilePath путь, куда сохраняется сгенерированный отчет
     * @return ActionListener для экспорта отчета
     */
    public static ActionListener getExportHtmlReportListener(JFrame frame, String reportPath, String
outputFilePath) {
        return e -> {
            try {
                String xmlFilePath = GUI.openedFile != null ? GUI.openedFile.getAbsolutePath() : null;

                if (xmlFilePath == null) {
                    JOptionPane.showMessageDialog(frame, "Сначала загрузите XML файл!", "Ошибка",
JOptionPane.ERROR_MESSAGE);
                    return;
                }

                // Генерация HTML отчета
                ReportGenerator generator = new ReportGenerator();
                generator.generateHtmlReport(reportPath, xmlFilePath, outputFilePath);

                JOptionPane.showMessageDialog(frame, "HTML отчет успешно создан:\n" + outputFilePath,
"Успех", JOptionPane.INFORMATION_MESSAGE);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(frame, "Ошибка генерации HTML отчета:\n" + ex.getMessage(),
"Ошибка", JOptionPane.ERROR_MESSAGE);
                ex.printStackTrace();
            }
        };
    }



}
```

// XMLfile.java

```java
import org.w3c.dom.*; // объектная модель документа (DOM) --> создание объектов
import javax.xml.parsers.*; // для создания парсеров
// ParserConfigurationException; -- Для обработки ошибок конфигурации парсера ^
import java.io.File; // для работы с файлами
import javax.swing.table.DefaultTableModel;
import javax.xml.transform.*; // для преобразования и записи в XML-документ
import javax.xml.transform.dom.DOMSource; // источник данных DOM для записи
import javax.xml.transform.stream.StreamResult; // класс для записи XML в файл (поток)


public class XMLfile {

    /**
     * Метод для загрузки данных из XML-файла и добавления их в таблицу.
     * @param tableModel модель таблицы, куда будут добавлены данные
     * @param file файл XML, откуда будут загружены данные
     */
    public static void loadFromXML(DefaultTableModel tableModel, File file) {
        try {
            // фабрика для создания парсеров
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

            // создаем сам парсер (builder)
            DocumentBuilder builder = factory.newDocumentBuilder();

            Document doc = builder.parse(file); // загружаем и парсим сам файл
            doc.getDocumentElement().normalize(); // нормализуем "чистим после парсинга"

            NodeList nodeList = doc.getElementsByTagName("patient"); // загружаем все узлы пациентов
(каждый 'patient')
            tableModel.setRowCount(0); // очищаем таблицу перед загрузкой данных

            // проходимся по каждому элементу patients
            for (int i = 0; i < nodeList.getLength(); i++) {
                Node node = nodeList.item(i); // проходимся по каждому узлу
                if (node.getNodeType() == Node.ELEMENT_NODE) {
                    Element element = (Element) node; // преобразуем каждый узел для возможности работать
с аттрибутами

                    // Извлекаем данные из каждого элемента <patient>
                    String name = element.getElementsByTagName("name").item(0).getTextContent();
                    /* getElementsByTagName("name") -- ищем дочерние элемнты с определенным именем
                    *  .item(0) -- берем первый элемент из списка
                    *  */
                    String disease = element.getElementsByTagName("disease").item(0).getTextContent();
                    String doctor = element.getElementsByTagName("doctor").item(0).getTextContent();
                    String specialization =
element.getElementsByTagName("specialization").item(0).getTextContent();
                    String date = element.getElementsByTagName("date").item(0).getTextContent();
                    String status = element.getElementsByTagName("status").item(0).getTextContent();

                    // Добавляем данные в модель таблицы
                    tableModel.addRow(new Object[]{name, disease, doctor, specialization, date, status});
                }
            }

        } catch (Exception ex) { // доделать, чтобы делал полноценный вывод; ??
            ex.printStackTrace();
        }
    }

    /**
     * Метод для сохранения данных из таблицы в XML-файл.
     * @param tableModel модель таблицы, из которой будут извлечены данные
     * @param file файл, куда будет записан XML
     */
    public static void saveToXML (DefaultTableModel tableModel, File file) {
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.newDocument();
            Element root = doc.createElement("patients"); // корневой элемент
            doc.appendChild(root);

            // проходим по строкам таблицы и создаем элементы <patient>
            for (int row = 0; row < tableModel.getRowCount(); row++) {
                Element patient = doc.createElement("patient");
```

```
                // создаем элементы <name>, <disease>, <doctor> и т.д. для каждого пациента
                Element name = doc.createElement("name");
                name.appendChild(doc.createTextNode(tableModel.getValueAt(row, 0).toString()));
                patient.appendChild(name);

                Element disease = doc.createElement("disease");
                disease.appendChild(doc.createTextNode(tableModel.getValueAt(row, 1).toString()));
                patient.appendChild(disease);

                Element doctor = doc.createElement("doctor");
                doctor.appendChild(doc.createTextNode(tableModel.getValueAt(row, 2).toString()));
                patient.appendChild(doctor);

                Element specialization = doc.createElement("specialization");
                specialization.appendChild(doc.createTextNode(tableModel.getValueAt(row, 3).toString()));
                patient.appendChild(specialization);

                Element date = doc.createElement("date");
                date.appendChild(doc.createTextNode(tableModel.getValueAt(row, 4).toString()));
                patient.appendChild(date);

                Element status = doc.createElement("status");
                status.appendChild(doc.createTextNode(tableModel.getValueAt(row, 5).toString()));
                patient.appendChild(status);

                // добавляем <patient> к корневому элементу <patients>
                root.appendChild(patient);
            }

            TransformerFactory transformerFactory = TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            DOMSource source = new DOMSource(doc);
            StreamResult result = new StreamResult(file);

            // Записываем XML в указанный файл
            transformer.transform(source, result);

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

```
// ReportGenerator.java
```

```java
import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.data.JRXmlDataSource;

import java.io.File;
import java.util.HashMap;

public class ReportGenerator {
    /**
     * Метод для генерации отчета на основе XML файла и сохранения в формате PDF.
     * @param reportPath путь к файлу отчета (.jrxml)
     * @param xmlFilePath путь к XML файлу с данными
     * @param outputFilePath путь для сохранения выходного PDF файла
     * @throws JRException при возникновении ошибок JasperReports
     */
    public void generatePdfReport(String reportPath, String xmlFilePath, String outputFilePath) throws
JRException {
        generateReport(reportPath, xmlFilePath, outputFilePath, ReportType.PDF);
    }

    /**
     * Метод для генерации отчета на основе XML файла и сохранения в формате HTML.
     * @param reportPath путь к файлу отчета (.jrxml)
     * @param xmlFilePath путь к XML файлу с данными
     * @param outputFilePath путь для сохранения выходного HTML файла
     * @throws JRException при возникновении ошибок JasperReports
     */
    public void generateHtmlReport(String reportPath, String xmlFilePath, String outputFilePath) throws
JRException {
        generateReport(reportPath, xmlFilePath, outputFilePath, ReportType.HTML);
    }

    /**
     * Общий метод для генерации отчетов.
     * @param reportPath путь к файлу отчета (.jrxml)
     * @param xmlFilePath путь к XML файлу с данными
     * @param outputFilePath путь для сохранения выходного файла
     * @param reportType тип отчета (PDF или HTML)
     * @throws JRException при возникновении ошибок JasperReports
     */
    private void generateReport(String reportPath, String xmlFilePath, String outputFilePath, ReportType
reportType) throws JRException {
        // Проверяем наличие файла отчета
        File reportFile = new File(reportPath);
        if (!reportFile.exists()) {
            throw new RuntimeException("Файл отчета не найден: " + reportPath);
        }

        // Проверяем наличие XML файла с данными
        File xmlFile = new File(xmlFilePath);
        if (!xmlFile.exists()) {
            throw new RuntimeException("Файл XML данных не найден: " + xmlFilePath);
        }

        // Создаем источник данных из XML файла
        JRXmlDataSource xmlDataSource = new JRXmlDataSource(xmlFile, "/patients/patient");

        // Компилируем JRXML отчет
        JasperReport jasperReport = JasperCompileManager.compileReport(reportPath);

        // Параметры (если нужны, передаются в HashMap)
        HashMap<String, Object> parameters = new HashMap<>();

        // Заполняем отчет данными
        JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, parameters, xmlDataSource);

        // Экспортируем отчет в соответствующий формат
        switch (reportType) {
            case PDF:
                JasperExportManager.exportReportToPdfFile(jasperPrint, outputFilePath);
                System.out.println("PDF отчет успешно создан: " + outputFilePath);
                break;

            case HTML:
                JasperExportManager.exportReportToHtmlFile(jasperPrint, outputFilePath);
                System.out.println("HTML отчет успешно создан: " + outputFilePath);
                break;
        }
```

```
    }

    // Перечисление для типов отчетов
    private enum ReportType {
        PDF, HTML
    }
}
```