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 ВТ

**ОТЧЕТ**

**по лабораторно-практическим работам № 7 и 8**

**«Построение отчетов в PDF- и HTML- форматах»**

**«Организация многопоточных приложений»**

**по дисциплине «Объектно - ориентированное программирование на языке Java»**

Выполнила Половникова А.С

Факультет КТИ

Группа № 3312

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2024 г

**Цель работы**

Знакомство с правилами и классами построения параллельных приложений в языке Java.

**Описание задания**

1. Создайте новый проект, который будет дублировать проект

лабораторной работы № 7.

2. В новом проекте опишите 3 параллельных потока, один из которых

будет загружать данные из XML-файла, второй  редактировать данные и

формировать XML-файл для отчета, а третий  строить отчет в HTMLформате. Второй поток не должен формировать XML-файл для отчета, пока

первый не загрузит данные в экранную форму, а третий поток не должен

формировать отчет, пока второй поток редактирует данные и записывает их в

XML-файл.

3. С помощью конструктора подготовьте шаблон для отчета.

4. Запустите приложение и убедитесь, что сформирован HTML-файл.

Просмотрите его в браузере и проверьте правильность данных и формы.

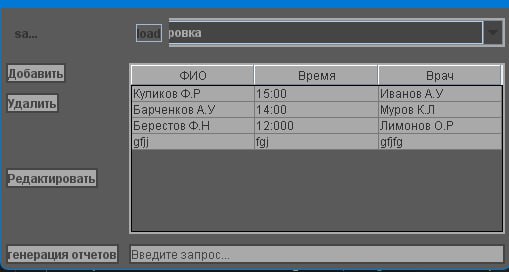
5. Сгенерируйте документацию с помощью Javadoc и просмотрите ее в

браузере.

**Описание и проверка работоспособности приложения**

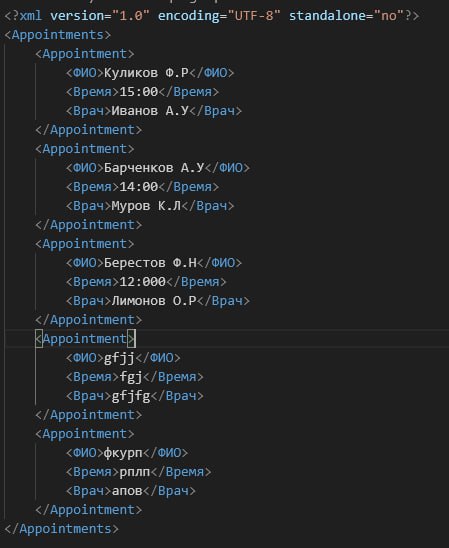
При запуске программы пользователь должен будет нажать на поле меню «генерация отчетов», после чего произойдёт следующие:

1. Начнётся загрузка данных из xml файла “appointmens” в таблицу.

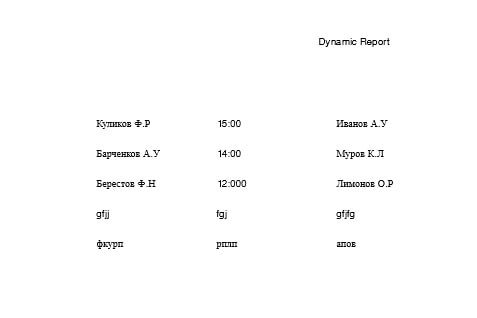


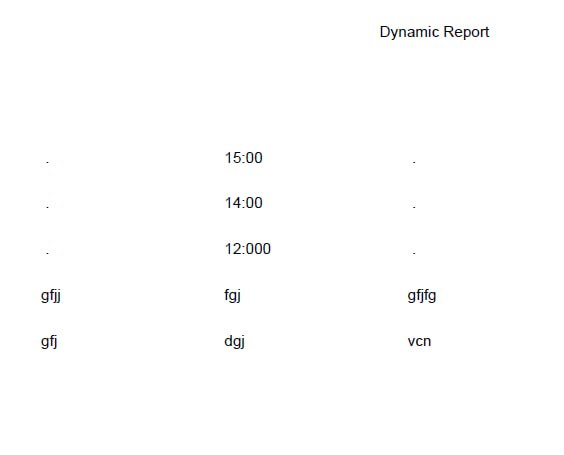
1. К данным, которые загрузились файла, будет добавлена ещё одна запись (в конце списка).



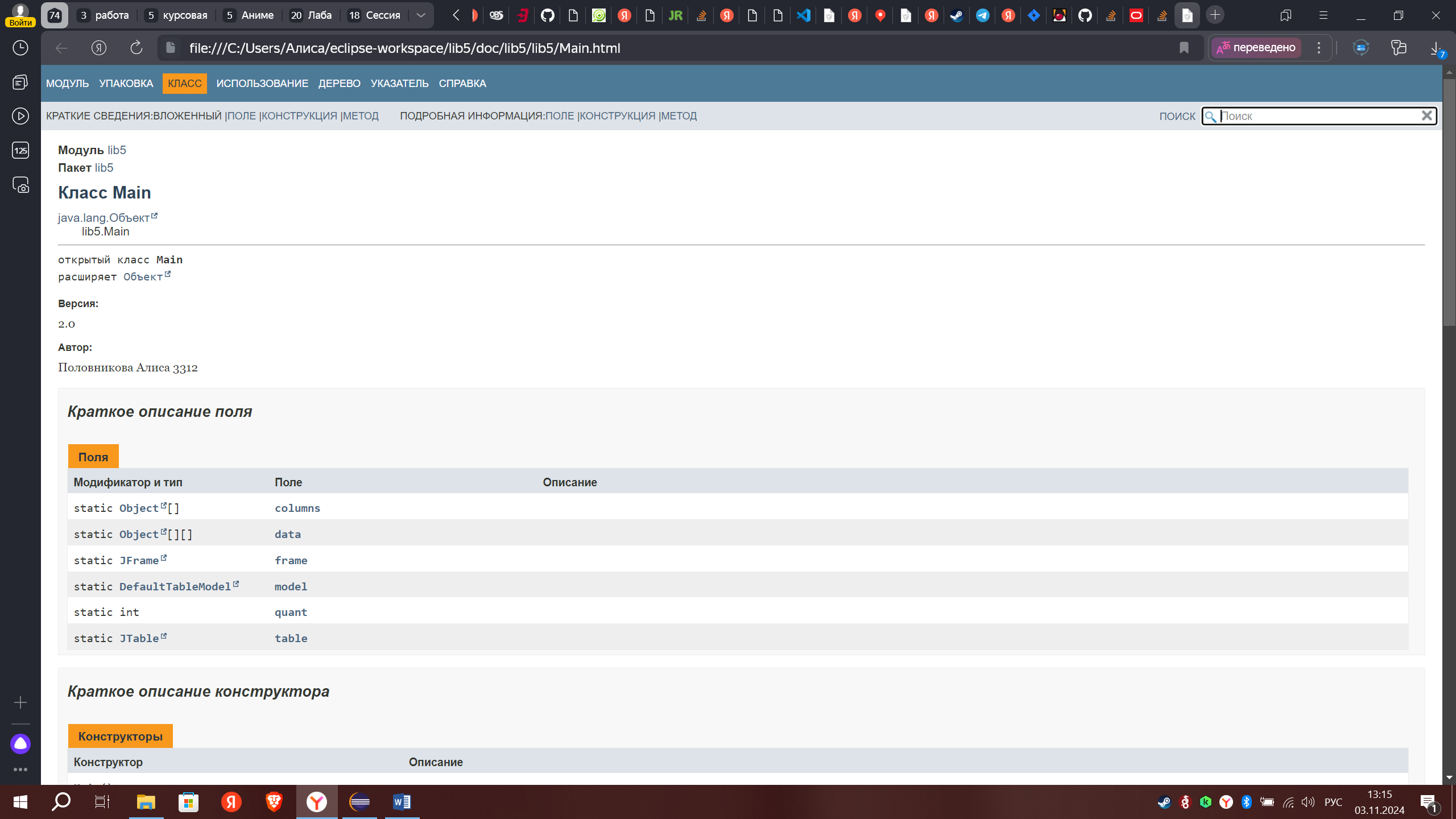
Откроем файл, чтобы убедиться, что он не содержит последнею строку.

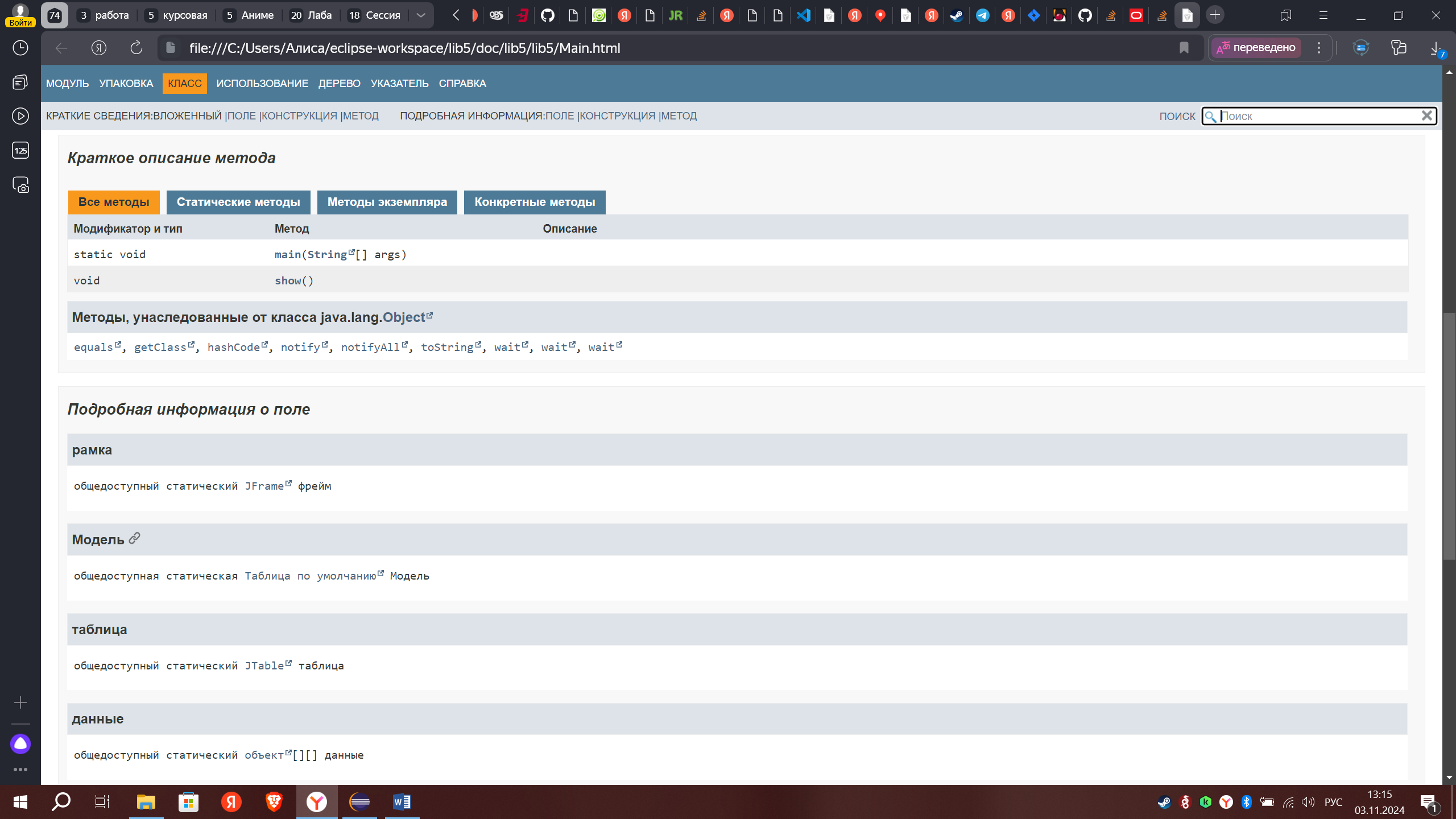
1. Вся данные из таблицы сохраняются в pdf файле и html файле





**Текст документации Javadoc**

****

****

**Фрагменты кода, отвечающие за организацию параллельной работы**

**трех потоков.**

package com.proghelp;

import net.sf.jasperreports.engine.\*;

import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.TableModel;

import net.sf.jasperreports.engine.\*;

import net.sf.jasperreports.engine.data.JRBeanCollectionDataSource;

import javax.swing.\*;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.transform.\*;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import javax.xml.parsers.ParserConfigurationException;

import java.io.File;

import java.io.FileOutputStream;

import java.util.\*;

import java.util.concurrent.\*;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

public class ReportGenerator {

public static void generateReport(JTable table)

{

ExecutorService executor = Executors.newFixedThreadPool(3);

CountDownLatch latch1 = new CountDownLatch(1);

CountDownLatch latch2 = new CountDownLatch(1);

executor.submit(() -> {

try

{

System.out.println("Loading data from XML...");

loadXMLToTable(table, "aaa.xml", Main.model);

System.out.println("Data loaded successfully.");

latch1.countDown();

}

catch (Exception e)

{

e.printStackTrace();

}

});

executor.submit(() -> {

try

{

latch1.await();

System.out.println("Editing data and saving to XML...");

saveTableToXML(table, "edited\_data.xml");

System.out.println("Data edited and saved successfully.");

latch2.countDown();

}

catch (Exception e)

{

e.printStackTrace();

}

});

executor.submit(() -> {

try

{

latch2.await();

System.out.println("Generating HTML report...");

generateHTMLReport(table);

System.out.println("HTML report generated successfully.");

}

catch (Exception e)

{

e.printStackTrace();

}

});

executor.shutdown();

}

private static void loadXMLToTable(JTable table, String filePath, DefaultTableModel model) throws Exception

{

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.parse(filePath);

doc.getDocumentElement().normalize();

NodeList nodeList = doc.getElementsByTagName("Appointment");

model.setRowCount(0);

for (int i = 0; i < nodeList.getLength(); i++)

{

Node node = nodeList.item(i);

if (node.getNodeType() == Node.ELEMENT\_NODE)

{

Element element = (Element) node;

Object[] rowData = new Object[model.getColumnCount()];

for (int j = 0; j < model.getColumnCount(); j++)

{

String columnName = model.getColumnName(j);

Node columnNode = element.getElementsByTagName(columnName).item(0);

rowData[j] = (columnNode != null) ? columnNode.getTextContent() : "";

}

model.addRow(rowData);

}

}

}

private static void saveTableToXML(JTable table, String filePath) throws Exception

{

DefaultTableModel model = (DefaultTableModel) table.getModel();

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.newDocument();

Element root = doc.createElement("Table");

doc.appendChild(root);

for (int i = 0; i < model.getRowCount(); i++)

{

Element row = doc.createElement("Row");

root.appendChild(row);

for (int j = 0; j < model.getColumnCount(); j++)

{

Element column = doc.createElement(model.getColumnName(j));

column.appendChild(doc.createTextNode(model.getValueAt(i, j).toString()));

row.appendChild(column);

}

}

TransformerFactory transformerFactory = TransformerFactory.newInstance();

Transformer transformer = transformerFactory.newTransformer();

DOMSource source = new DOMSource(doc);

StreamResult result = new StreamResult(new FileOutputStream(filePath));

transformer.transform(source, result);

}

private static void generateHTMLReport(JTable table)

{

try

{

TableModel model = table.getModel();

int rowCount = model.getRowCount();

int columnCount = model.getColumnCount();

List<Map<String, Object>> dataList = new ArrayList<>();

for (int i = 0; i < rowCount; i++)

{

Map<String, Object> rowData = new HashMap<>();

for (int j = 0; j < columnCount; j++)

rowData.put(model.getColumnName(j), model.getValueAt(i, j));

dataList.add(rowData);

}

JRBeanCollectionDataSource dataSource = new JRBeanCollectionDataSource(dataList);

String jrxmlFilePath = "dynamic\_report.jrxml";

JRXMLGenerator.generateJRXML(table, jrxmlFilePath);

JasperReport jasperReport = JasperCompileManager.compileReport(jrxmlFilePath);

Map<String, Object> parameters = new HashMap<>();

parameters.put("ReportTitle", "Generated Report");

JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, parameters, dataSource);

JasperExportManager.exportReportToHtmlFile(jasperPrint, "dynamic\_report.html");

}

catch (JRException e)

{

e.printStackTrace();

}

}

}

**Вывод**

Во время выполнения данной лабораторной работы, были получены навыки работы с технологией формирования отчетов с использованием конструктора Jaspersoft iReport Designert, библиотекой iTest и правилами построения параллельных приложений в языке Java. В пункте «Описание и проверки работоспособности приложения» ошибок выявлено не было.

**Ссылки**

https://github.com/MniAlice/lab\_oop