**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

отчет

**по лабораторной работе №8**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Логирование

|  |  |  |
| --- | --- | --- |
| Студент гр. 9308 |  | Дементьев Д.П. |
| Преподаватель |  | Гречухин М.Н. |

Санкт-Петербург

2021

# Цель работы

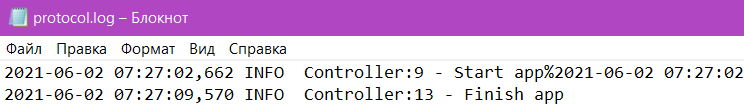
Знакомство c методами протоколирования работы приложения с использования библиотеки Log4j.

# Задача

1. Перечень используемых типов сообщений, которые выводятся в логфайл.
2. Конфигурационный файл log4j.properties.
3. Исходные тексты классов, где осуществляется протоколирование работы приложения.
4. Фрагменты кода, содержащие WARN, INFO, ERROR и DEBUG

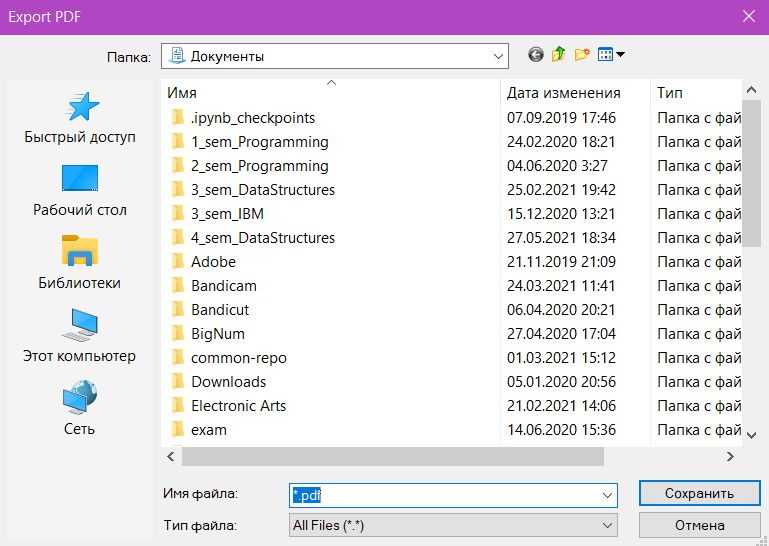
# Демонстрация результатов

При запуске программы начинается автоматическое протоколирование. Сначала в файл записывается строка «Start app» с указанием времени (это обозначает начало работы класса main). Если на этом пользователь завершает работу программы, то выводится строка «Finish app» (указывающая на завершение работы приложения). Важно уточнить, что в логфайле эти две записи разделены между собой большим количеством логов другого вида: настройка Hibernate, подключение к SQL и т.д., но для наглядности они были удалены.

**

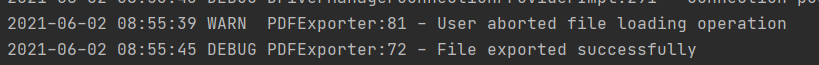
*Рисунок 1. Логирование в файле начала и завершения программы*

Другие же логи отвечают за протоколирование класса PDFExporter, ответственного за экспорт таблиц БД в PDF файлы:



*Рисунок 2. Окно выбора директории для сохранения*

Также логи выводятся не только в файл, но и в консоль:

**

*Рисунок 3. Логирование в консоли работы PDFExporter’a*

Как можно видеть выше, сначала окно выбора директории для сохранения файла было закрыто, а после данные успешно были экспортированы.

# Конфигурационный файл log4j.properties

# Root logger option  
log4j.rootLogger=DEBUG, file, stdout  
  
# Direct log messages to a log file  
log4j.appender.file=org.apache.log4j.RollingFileAppender  
log4j.appender.file.File=./Data/Library/protocol.log  
log4j.appender.file.MaxFileSize=10MB  
log4j.appender.file.MaxBackupIndex=10  
log4j.appender.file.layout=org.apache.log4j.PatternLayout  
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n  
  
# Direct log messages to stdout  
log4j.appender.stdout=org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.Target=System.out  
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

Пояснение конфигураций:

1. Описывает уровень сообщения DEBUG c символическим названием file, на которое будет выводиться сообщение, и stout – на консоль.

2. Указывает, что сообщение будет выводиться в файл.

3. Задаёт имя и путь файлу для записи.

4. Определяет максимальный размер файла.

5. Максимальное резервное копирование.

6. Описывает формат вывода текста.

7. Регулярное выражение, описывающее вывод данных в файл.

8. Указывает, что сообщение будет выводиться также и в консоль

9. Указывает поток вывода

10. Описывает формат вывода текста.

11. Регулярное выражение, описывающее вывод данных в консоль.

# Фрагменты кода, содержащие WARN, INFO, ERROR и DEBUG

**Класс Controller**

import app.\*;  
import org.apache.log4j.Logger;  
  
public class Controller {  
  
 public static final Logger *logger* = Logger.*getLogger*(Controller.class);  
  
 public static void main(String[] args) {  
 *logger*.info("Start app");  
 // Создание и отображение экранной формы  
 App controller= new App();  
 *logger*.info("Finish app");  
 }  
}

**Класс PDFExporter**

package app;  
  
import com.itextpdf.text.\*;  
import com.itextpdf.text.pdf.\*;  
import org.apache.log4j.Logger;  
  
import java.awt.FileDialog;  
import java.io.\*;  
import java.util.Date;  
  
import javax.swing.JFrame;  
import javax.swing.JTable;  
import javax.swing.table.DefaultTableModel;  
// сделать параллельным составление отчётов для каждой из таблиц, а после склеить их в один файл  
public class PDFExporter extends JFrame {  
 public static final Logger *logger* = Logger.*getLogger*(PDFExporter.class);  
 // экспорт выбранной таблицы данных  
 public PDFExporter(JTable TableData)  
 {  
 FileDialog fileDial = new FileDialog(this, "Export PDF", FileDialog.*SAVE*);  
 fileDial.setFile("\*.pdf");  
 fileDial.setVisible(true);  
 if (fileDial.getFile() != null)  
 if (!fileDial.getFile().contains("."))  
 fileDial.setFile(fileDial.getFile()+".pdf");  
 String fileFullName = fileDial.getDirectory() + fileDial.getFile();  
  
 if(!fileFullName.equals("nullnull"))  
 {  
 Document document = new Document(PageSize.*A4*, 15, 15, 15, 15);  
 PdfPTable pdfTable = new PdfPTable(TableData.getColumnCount());  
  
 try {  
 PdfWriter.*getInstance*(document, new FileOutputStream(fileFullName));  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 *logger*.error("File Not Found", e);  
 } catch (DocumentException e) {  
 e.printStackTrace();  
 *logger*.error("IO exception occured", e);  
 }  
  
 Font font = FontFactory.*getFont*("/fonts/DejaVuSans.ttf", "cp1251", BaseFont.*EMBEDDED*, 10);  
  
 // add info  
 Paragraph info = new Paragraph("Report from " + new Date() + " \n\n");  
  
 // table header  
 for(int i =0; i < TableData.getColumnCount(); ++i) {  
 PdfPCell header = new PdfPCell();  
 header.setBackgroundColor(BaseColor.*LIGHT\_GRAY*);  
 header.setBorderWidth(2);  
 header.setPhrase(new Phrase(TableData.getModel().getColumnName(i), font));  
 pdfTable.addCell(header);  
 }  
  
 // table data  
 String data[][] = getTableData(TableData);  
 for(int i = 0; i < TableData.getRowCount(); i++){  
 for(int j =0; j < TableData.getColumnCount(); j++)  
 {  
 String dat = (data[i][j] == null? "" : data[i][j]);  
 pdfTable.addCell(new Phrase(dat, font));  
 }  
 }  
  
 document.open();  
  
 try {  
 document.add(info);  
 document.add(pdfTable);  
 *logger*.debug("File exported successfully");  
 } catch (DocumentException e) {  
 e.printStackTrace();  
 *logger*.error("Write Error", e);  
 }  
  
 document.close();  
 }  
 else {  
 *logger*.warn("User aborted file loading operation");  
 }  
 }  
  
 private String[][] getTableData(JTable table) {  
 DefaultTableModel dtm = (DefaultTableModel) table.getModel();  
 int nRow = dtm.getRowCount(), nCol = dtm.getColumnCount();  
 String[][] tableData = new String[nRow][nCol];  
 for (int i = 0; i < nRow; i++)  
 for (int j = 0; j < nCol; j++)  
 tableData[i][j] = (String) dtm.getValueAt(i, j);  
 return tableData;  
 }  
}

# Выводы

Во время выполнения данной лабораторной работы, были получены навыки работы с методами протоколирования приложения с использованием библиотеки Log4j.