Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Петрозаводский государственный университет» Физико-технический институт Кафедра информационно-измерительных систем и физической электроники

ПРАКТИЧЕСКАЯ РАБОТА
ОТЧЁТ
по предмету «Технология JAVA»

Программа для автоматизации процессов «Первичной Аккредитации» специалистов медицинского института ПетрГУ

Автор работы: студент группы
21512 Н. Ю. Новохатько
_____ 2020 г.
Научный руководитель: доцент
А.В. Соловьёв
_____ 2020 г.

Петрозаводск 2020

Содержание

Введение 3
Постановка задачи 4
Описание объёма выполненной работы 6
Описание алгоритма и методов решения 8
Руководство по сборке и подготовке программы к работе 16
Руководство пользователя программы 18
Отчёт о тестировании 21
Заключение 24
Исходный код 25
Приложение 1 41

Введение

В данной работе будет рассмотрена программа, написанная для реализации задачи автоматизации процессов «Первичной Аккредитации специалистов медицинского института ПетрГУ. Разработка данной программы устранит некоторые сложности, связанные с взаимодействием специалистов разных профессий, задействованных во время аккредитации. Перед выполнением работы нами была поставлена цель:

Создание программы для автоматизации процессов «Первичной Аккредитации» специалистов медицинского института ПетрГУ.

Исходя из цели, нами были поставлены задачи:

1. Создание модуля, позволяющего воспроизводить последовательность сообщений в заданные промежутки времени.
2. Создание модуля, позволяющего отправлять сообщения из модуля последовательности сообщений, отправлять их по сети, получать сообщения из сети, и отображать их в окне.
3. Создание модуля, позволяющего получить из поля путь к файлу со строкой пути к папке с исходными видео файлами, со списком названий и длительностью фрагментов, на которые файлы должны быть обрезаны. Подготовка видео фрагментов путём их нарезки.
4. Создание модуля, который объединит три предыдущих модуля.
5. Скомпилировать программу и предоставить пользователям лёгкий доступ к ней.

Постановка задачи

Требуется создать оконное приложение, состоящее из трёх окон, каждое из которых автоматизирует определённый процесс Аккредитации, и среды, предоставляющей меню для их выполнения.

Для выполнения поставленных задач требуется:

1. Выполнить в модуле воспроизведения последовательности сообщений действия:
    1) Получение списка сообщений и отрезков времени, через которые сообщения должны быть выполнены
    2) Запуск воспроизведения последовательности сообщений по кнопке.
    3) Прерывание воспроизведения последовательности сообщений по кнопке.
    4) В момент воспроизведения каждого сообщения отправление информации о сообщении в модуль отправки и получения сообщений по сети.
2. Выполнить в модуле отправки и получения сообщений по сети следующие действия:
    1) Выбор роли «клиент» или «сервер» через флажок выбора и получение ip-адреса сервера в роли «клиент».
    2) В случае, если выбрана роль «сервер», запуск потока получения сообщений от клиента по определённому порту.
    3) В случае, если выбрана роль «клиент», запуск потока отправки сообщений, получаемых от модуля вопроизведения последовательности.
3. Выполнить в модуле нарезки следующие действия:
    1) Получение из поля пути к списку со строкой пути к исходным файлам и настройками для получения фрагментов нарезанных файлов.

2) Проверка совпадения реального списка файлов нужной папке и файлов их списка, после чего выдача совпавших и не совпавших файлов в таблицу.
3) По нажатию кнопки открытие утилиты ffmpeg для нарезки первого фрагмента первого файла с параметрами (путь к исходному файлу, путь, куда фрагмент будет выгружен, время начала нарезки и длина фрагмента), полученными из входного списка.

4. Выполнить в модуле, объединяющем три предыдущих, следующие действия:
1) По нажатию пункта меню показать окно модуля воспроизведения, с возможностью менять его размер и скрывать.
2) По нажатию пункта меню показать окно модуля сетевого взаимодействия, с возможностью менять его размер и скрывать.
3) По нажатию пункта меню показать окно модуля нарезки, с возможностью менять его размер и скрывать.

Описание объёма выполненной работы

Объём выполненных работ:

1. Для модуля «Воспроизведение сообщений»:
   1) Создание и настройка GUI интерфейса модуля «Воспроизведение сообщений»: bigField2 - поле для получения номера сообщения; bigField1 - поле для получения периода воспроизведения между этим сообщением и следующим; add1 - кнопка добавления нового пункта в последовательности сообщений; add2 - кнопка удаления пункта; add3 - кнопка начала воспроизведения сообщений; add4 - кнопка прерывания воспроизведения сообщений.
   2) Добавление события добавления нового пункта в последовательности сообщений - add1.addActionListener.
   3) Добавление события удаления нового пункта в последовательности сообщений – add2.addActionListener.
   4) Добавление события начала воспроизведения последовательности сообщений – add3.addActionListener.
   5) Добавление события добавления нового пункта в последовательности сообщений – add4.addActionListener.
2. Для модуля «Сетевого взаимодействия»:
   1) Создание и настройка GUI интерфейса модуля «Сетевого взаимодействия»: меню group из флажков serverButton – роль «Сервер», clientButton – роль «Клиент»; add1 – кнопка, запускающая модуль в роли сервера или клиента; add2 – кнопка, прерывающая работу модуля; bigField1 - поле ввода ip-адреса сервера для клиента; textArea – однострочное поле для вывода сообщений, получаемых сервером от клиента.
   2) Добавление события добавления нового пункта в последовательности сообщений - add1.addActionListener.
   3) Добавление события удаления нового пункта в последовательности сообщений – add2.addActionListener.

4) Добавление потока serverModuleThread для получения сообщений.
5) Добавление потока clientModuleThread для отправки сообщений.

3. Для модуля «Нарезки видео файлов»:
   1) Создание и настройка GUI интерфейса модуля «Нарезки видео файлов»: table1 – таблица отображения названий исходных файлов, путей к ним и статусов «найден» / «не найден» / «нарезан» / «не нарезан»; bigField1 – поле для ввода строки с расположением текстового файла с настройками; table – кнопка для заполнения таблицы; obr – кнопка для нарезки первого фрагмента первого файла.
   2) Добавление события заполнения таблицы table1 после чтения списка из bigField1 - table.addActionListener.
   3) Добавление события вызова утилиты ffmpeg для нарезки первого фагмента первого исходного файла из таблицы table1 - obr.addActionListener.
4. Для модуля «Среда формирования окон»:
   1) Создание и настройка GUI интерфейса модуля «Среда формирования окон»: формирование окна intVideo для работы с видео файлами; формирование окна intTime для работы с последовательностью сообщений; формирование окна intEthenet для формирования окна для работы с пересылкой сообщений по принципу «клиент-сервер»; time1 – пункт меню для отображения intTime; video1 – пункт меню для отображения intVideo; ethernet1 – пункт меню для отображения intEthernet; bar – родительское меню, куда должны входить time1, video1, ethernet1.
   2) Добавление события открытия окна time1 - time1.addActionListener.
   3) Добавление события открытия окна video1 - video1.addActionListener.
   4) Добавление события открытия окна ethernet1 - ethernet1.addActionListener.

Описание алгоритма и методов решения

Наименование программы:

Аккредитация_v0.005_Java

ПО, необходимое для функционирования программы:

1. ОС Windows 7.
2. Интерпретатор java версии 1.8.0_161.

Язык, на котором написана программа :

java

Функциональное назначение:

Автоматизация процессов «Воспроизведение сообщений», «Нарезка видео» и «Обмен сообщениями по сети», происходящих во время Первичной Аккредитации медицинского института.

Сведения об ограничениях:

Версия программы реализует только часть процессов и использует тестовые входные данные.

Логическая структура программы:

Смотрите Приложение 1. Блок-схема программы.

Описание каждой функции, входящей в список:

public class InternalFrame() :

1. **public InternalFrame** () – формирует окно с меню доступа к подокнам приложения timeFrame, videoFrame, ethernetFrame. Входные переменные: intTime, intVideo, intEthernet – объекты классов окон (2), (3), (4). Задаёт стандартные события реагирования на кнопки time1 (5), video1 (6), ethernet1 (7).
2. **public class** timeFrame () – класс окна «Воспроизведение сообщений».
3. **public class** videoFrame () – класс окна «Нарезка видео файлов».
4. **public class** ethernetFrame ()– класс окна «Сетевое взаимодействие».
5. time1.addActionListener() – событие нажатия кнопки time1 для показа окна «Воспроизведение сообщений».

6. video1.addActionListener()– событие нажатия кнопки time1 для показа окна «Нарезка видео файлов».
7. ethernet1.addActionListener()– событие нажатия кнопки time1 для показа окна «Сетевое взаимодействие».

public class timeFrame():

1. **public** timeFrame() – формирует меню окна «Воспроизведение сообщений». Задаёт стандартные события нажатия на кнопки add1 - «Добавить интервал в список», add2 - «Удалить из списка», add3 - «Начать проход», add4 - «Закончить проход».
2. **private int** setInterval () – запускается с каждым тиком таймера и проверяет, совпадает ли номер тика с количеством тиков. Входные данные: глоб. переменная interval – количество тиков в данный момент времени; summ – сумма секунд всех интервалов; data2[] – массив количеств тиков между сообщениями. Выходные данные: coutntlist - номер воспроизводимого сообщения.
3. **private void** ethernetSignal (String str, Boolean flag) – отсылает сообщение о воспроизводимом интервале в модуль Сетевое взаимодействие. Входные и выходные данные: str – строка с номер сообщения, flag – флаг новизны данных.
4. add1.addActionListener() – добаляет новую строку в список воспроизведения по нажатию кнопки «Добавить в список». Входные данные: строки из полей bigField1, bigField2. Выходные данные: строка strA, добавленная в список dlm.
5. add2.addActionListener() – удаляет строку из списка воспроизведения по нажатию кнопки «Удалить из списка». Входные данные: строка из поля bigField2. Выходные данные: строка strB, удалённая из списка dlm.
6. add3.addActionListener() – запускает задание объекта timer для отсчёта тиков. Входные данные: list1 – список строчек с интервалами,

получаемый из Scr – отображаемого списка. Выходные данные: номера тиков, получаемые запущенным заданием из функции setInterval().

7. add4.addActionListener – Прекращает работу задания объекта timer для отсчёта тиков. Выходные данные: глоб. переменная interval – количество тиков в данный момент времени.

public class videoFrame():

1. **public** videoFrame() - формирует меню окна «Нарезка видео файлов». Задаёт стандартные события нажатия на кнопки table - «Заполнить таблицу», obr - «Нарезать файл».
2. **private void** readOffile (String way) – читает файл с настройками формата .txt построчно. Входные данные: way - строка пути к файлу с настройками. Выходные данные: глоб. переменная fileStrings – массив строк, считанных из файла.
3. **private void** searchOffile (String way) – функция, осуществляющая начальный запуск функции FilesFromFolder() для поиска .mp4 файлов. Входные данные: way – строка с адресом корневой папки иерархии. Выходные данные объект folder типа File.
4. **private void** FilesFromFolder(File folder) – рекурсивная функция для поиска .mp4 файлов. Входные данные: объект folder типа File – объект пути к корневой папке иерархии. Выходные данные: глоб. Переменная pathStrings – список имён файлов и абсолютных путей к ним.
5. **Private void** compareOffile () – функция, сравнивающая файлы из списка, найденные readOffile() и найденные функцией searchOffile (). Входные данные: глоб. переменная pathStrings – список имён файлов и абсолютных путей к ним. Выходные данные: глоб. Переменная fileEndStrings – массив с параметрами для запуска ffmpeg; pathEndStrings – массив из имён/путей для заполнения таблицы окна.
6. **private Vector<String>** searchOfarray(String str) – получение массива всех строк, идущих после названия исходного файла и до следующего

названия, т.е. строк а настройками начала, длительности и имени нарезаемого фрагмента, нужного для запуска утилиты ffmpeg . Входные данные: str – строка с названием исходного файла, для которого нужны все последующие строки с настройками. Выходные данные: searchVector – массив из строк с настройками для ffmpeg.

7. **private void** addOffile() – добавление данных в таблицу tableModel. Входные данные: pathEndStrings – массив из имён файлов и путей к ним. Выходные данные:pathStr – строки для заполнения таблицы.
8. **private void** convertOffile() - выполняет подготовку к запуску ffmpeg, создавая новый двумерный массив с названием и расположением исходного файла, и названием и расположением фрагмента, временем начала фрагмента, длительностью фрагмента. Входные данные: fileEndStrings – массив с параметрами для запуска ffmpeg; cutFolder – строка пути к корневовой папке в иерархии папок нарезки. Выходные данные: folderPath – строка для двумерного массива с параметрами, формируемого функцией arrayOffolders().
9. **Private void** folderOffile() - функция, создающая иерархию папок с названиями исходных файлов для помещения в них фрагментов нарезанных файлов. Входные данные: convertFolders – массив с названиями папок, соответствующими названиям исходных файлов. Выходные данные: строки для запуска функции createOffolder().
10. **private void** createOffolder(String addr) – функция, создающая директорию. Входные параметры: addr – строка с абсолютным путём, по которому должен быть создан каталог. Выходные параметры: созданные каталоги.
11. **private void** processOffile() – функция, запускающая функцию запуска стороннего процесса ffmpeg. Входные параметры: строка расположения утилиты ffmpeg, массив параметров для запуска нарезки первого видеофайла из таблицы на первый фрагмент. Выходные параметры: строки для запуска функции createOfprocess ().

12. **private method** createOfprocess() - функция, запускающая ffmpeg. Входные параметры: program - строка расположения утилиты ffmpeg, parameters - массив параметров для запуска нарезки первого видеофайла из таблицы на первый фрагмент. Выходные параметры: Запуск стороннего процесса ffmpeg.
13. **private void** endOffile() – функция, осуществляющая запуск потока для слежение за поведением стороннего процесса ffmpeg после его запуска. На данный момент не запрограммирована.
14. **private int** timeOfsubstr(String str) – функция, осуществляющая конвертацию времени из формати 00:00:00 в 000 мс. Входные данные: str – строка формата «00:00:00». Выходные данные: число миллисекунд.
15. **private String** differenceOfsubstr(int start, int finish) – функция, получающая разность начального и конечного времени обрезки видео файла. Входные данные: start – начальное время фрагмента в миллисекундах, finish – конечное время фрагмента в миллисекундах. Выходные данные: строка в формате «00:00:00».
16. **private String** searchOfmenu (String filename) – функция поиска строки пути к папке, совпадающей с именем исходного файла, из которого будет вырезан фрагмент. Входные данные: filename – строка, по которой будет осуществляться поиск, глоб. переменная pathEndStrings – массив, в котором будет осуществляться поиск . Выходные данные: pathName – строка, содержащая найденный путь, или пустая строка, если ничего не найдено.
17. **private String** formOfpath (String filename, String substr) – функция, составляющая абсолютный путь к фрагменту нарезаемого файла. Входные данные: filename – строка, содержащая название разрезаемого файла, substr – строка, содержащая название фрагмента файла, глоб. переменная cutFolder - – строка пути к корневой папке в иерархии

папок нарезки. Выходные данные: childPath – строка, содержащая абсолютный путь к фрагменту.

18. **private Vector <String>** parametersOfstr(String menuName,String childPath, String substr, String convsubstr) – функция, записывающая в массив строк параметры для запуска ffmpeg в правильном порядке. Входные данные: menuName – строка, содержащая путь к исходному файлу, childPath – строка, содержашая путь к фрагменту, substr – начальное время фрагмента в исходном файле, convsubstr – длина вырезаемого фрагмента. Выходные данные:str – вектор с массивом строковых параметров для ffmpeg.
19. **private String** foldOfPath(String filename) - – функция, составляющая абсолютный путь к папке с названием, совпадающим с названием файла, из которого будет вырезаться фрагмент. Входные данные: filename – строка, содержащая название разрезаемого файла, глоб. переменная cutFolder - – строка пути к корневой папке в иерархии папок нарезки. Выходные данные: childPath – строка, содержащая абсолютный путь к папке.
20. **private void** arrayOffolders(String folder) – добавление пути к папке, одноимённой с исходным файлом, в строковый массив. . Входные данные: folder – строковая переменная, содержит путь к папке, глоб. переменная convertFolder – содержит пути к папкам с, имена в которых идентичны именам исходных файлов. Выходные данные: добавленная в convertFolder строка folder.
21. table.addActionListener() – по нажатию кнопки table находит список по ссылке из поля bigField1, читает из списка, находит совпадения в списке и в папке с исходниками, показывает имена файлов, пути к них и статус «совпал/не совпал» в таблице table1.
22. obr.addActionListener() – по нажатию кнопки obr конвертирует запись первого файла из списка в параметры для ffmpeg, создаёт иерархию папок для фрагментов файлов, запускает утилиту ffmpeg с

параметрами, запускает поток ожидания завершения нарезки файла и процесса ffmpeg.

public class ethernetFrame():

1. **public** ethernetFrame () формирует меню окна «Пересылка сообщение по сети». Задаёт стандартные события нажатия на кнопки add1 - «Запустить клиент/сервер», add2 - «Отсоединить клиент/сервер».
2. **public class** serverModuleThread() – получает из сети пересылаемые сообщения и выводит их в многострочное поле textArea. Входные данные: inClient – строковая переменная для получения сообщения от клиента. Выходные данные: строка в поле textArea.
3. **public class** clientModuleThread () – отправляет в сеть сообщения, полученные из модуля «Воспроизведение сообщений. Входные данные: строка с ip-адресом сервера, полученный из поля bigField1, newData – булева переменная, которая ставится в единицу, когда данное только что положено в переменную Data, и ставится в ноль, когда данное отправляется. Выходные данные: отправленная серверу строка с сообщением.
4. add1.addActionListener() - по нажатию кнопки add1, в зависимости от того, нажат ли clientButton или serverButton, запускат модуль в роли «Клиент» или «Сервер».
5. add2.addActionListener()- по нажатию кнопки add2 принудительно отсоединяет модуль от сети. Не запрограммирован.

Используемые технические средства:

1. Процессор Intel Core i3-6300 3,8 ГГц.
2. Оперативная память DDR3 8Гб.
3. Жесткий диск SATA 1Тб.

Способ вызова и загрузки программы:

Программа вызывается путём двойного клика по ярлыку Аккредитация_v0.005_Java.

Входные данные:

Тестовые данные для модулей «Пересылка сообщений по сети» и «Воспроизведение сообщений» содержатся в глоб. переменной data1 модуля «Воспроизведение сообщений».

Тестовые данные для модуля «Нарезка файлов» содержатся в перменных array и arraystr. Тестовое видео для нарезки содержится в папке Videos, идущей в комплекте с программой.

Выходные данные:

Модуль «Воспроизведение сообщений»: метка label3 – вывод времени, оставшегося до завершения прохода, метка label2 – вывод текущего сообщения.

Модуль «Пересылка сообщений по сети» - для модуля в роли «Сервер»: многострочное поле textArea – вывод сообщений, полученных по сети от другого модуля в роли «Клиент».

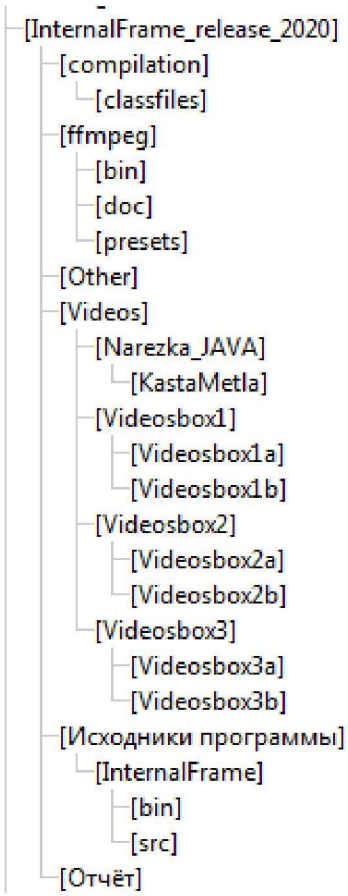
Модуль «Нарезка видео файлов» - table1 – вывод списка имён найденных исходных файлов, путей к ним и статусов, найдены ли они в списке.

После запуска программы ffmpeg информация, выдаваемая её процессом в командной строке.

Руководство по сборке и подготовке программы к работе

В работе системы должны быть задействованы два ПК: один в роли сервера, принимающего и записывающего сообщения, другой – в роли клиента.

Программа состоит из следующей иерархии вложенных папок:



1. Internal_Frame_release_2020 – корневая папка проекта.
2. compilation – папка со скомпилированной программой Аккредитация_v0.005_Java.
3. ffmpeg - папка с консольной утилитой ffmpeg для нарезки файлов.
4. Other – папка с документами – отчётом.
5. Videos – папка, содержащая видео для демонстрации работы модуля нарезки файлов.
6. Исходники программы – папка с исходными текстами программы.

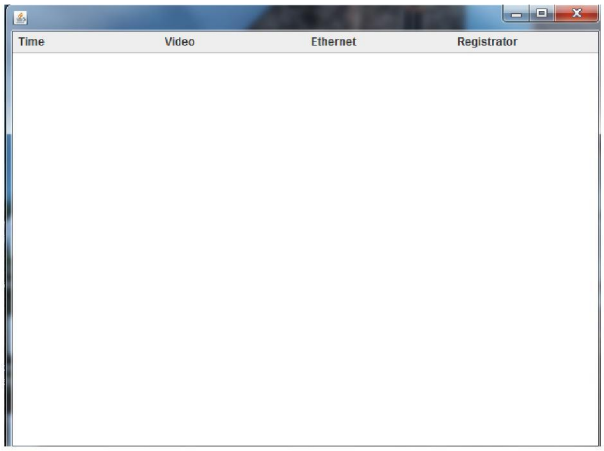
Для успешного запуска программы нужно совершить следующие действия:

1. Скопировать папку InternaFrame_release_2020 и поместить её по адресу “C:\” (в корень логического диска C:, который, обязательно, должен называться именно этой буквой, так как ссылка на утилиту ffmpeg абсолютная).
2. Скачать и установить пакет с java интерпретатором по ссылке: https://www.java.com/ru/. По умолчанию он установится в C:\Program Files\Java\jdk1.8.0_161.
3. Заходим C:\InternalFrame_release_2020\compilation и находим ярлык Аккредитация_v0.005_Java. Кликаем на него правой кнопкой мыши и выбираем «свойства» в выпадающем меню.
4. В поле объект изменяем строку «C:\ProgramData\Oracle\Java\javapath\javaw.exe -classpath C:\InternalFrame_release_2020\compilation\classfiles InternalFrame» на «”C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe” -classpath C:\InternalFrame_release_2020\compilation\classfiles InternalFrame», это позволит запустить программу при помощи интерпретатора.
5. Запускаем программу двойным кликом по ярлыку Аккредитация_v0.005_Java. Ярлык можно перетащить на рабочий стол, для удобства.

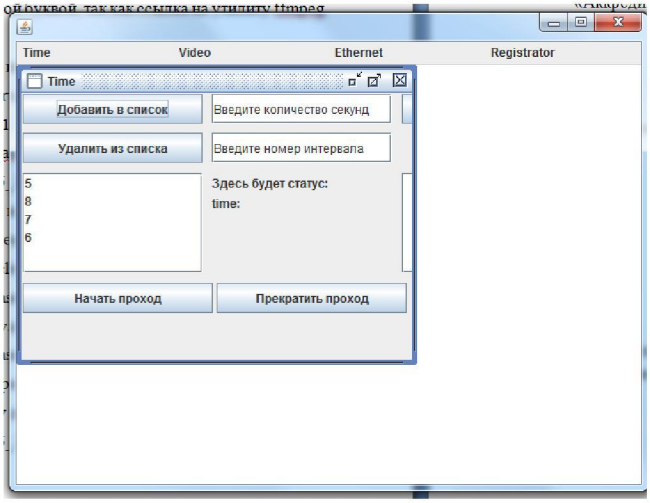
Руководство пользователя программы

1. Кликаем правой кнопкой мыши два раза по ярлычку «Аккредитация_v0.005_Java.

2. Появляется окно с серой полосой меню.



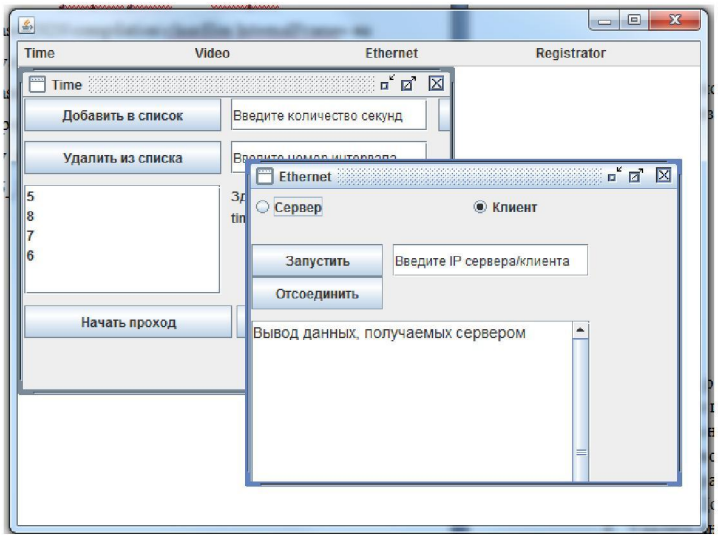
3. Кликаем по пункту меню «Timer», появляется окно модуля «Воспроизведение сообщений»:



4. В списке, расположенном посередине слева, можно видеть тестовые данные в списке.
5. Добавить новый интервал можно, введя в первое поле вверху количество секунд от вводимого интервала до следующего интервала,

и в поле, расположенное под ним, номер интервала. После нажатия кнопки «Добавить в список» он будет добавлен.

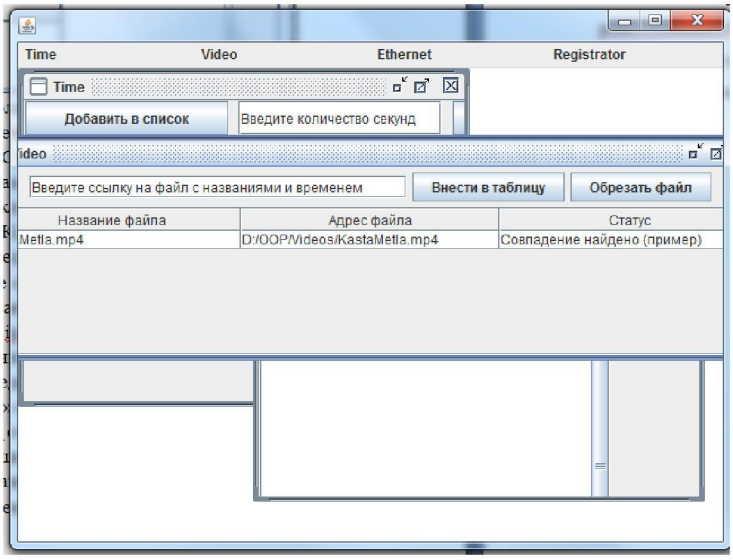
6. Удалить интервал можно, введя номер интервала во второе по счёту сверху поле и нажать на кнопку «Удалить из списка».
7. Чтобы начать проход с воспроизведеним сообщений по времени, нужно нажать на кнопку «Начать проход».
8. Чтобы прекратить проход, нужно нажать на кнопку «Прервать проход».
9. Во время прохода номер сообщения и оставшиеся время до окончания прохода отображаются под ярлыком «Здесь будет статус».
10. Нажав на кнопку «Ethernet», мы увидим окно «Пересылка сообщений по сети»:



11. Нажав один из двух флажков вверху окна, можно выбрать роль запущенного приложения – «Сервер» или «Клиент».
12. Если нажат флажок «Сервер», нажимаем на кнопку «Запустить». Модуль в этой роли самостоятельно запустится и будет ждать подключения к нему клиента.
13. Если нажат флажок «Клиент», в поле рядом вбиваем ip-адрес сервера. (Его можно узнать следующим образом: на компьютере в роли «Сервер» Пуск – поле «найти программы и файлы» - вбиваем cmd – открывается командная строка – вбиваем ipconfig – переписываем в

поле ip-адрес из поля ipv4.) Нажимаем на кнопку «Запустить». Клиент запустится и подключится к серверу.

14. Модули «Воспроизведение сообщений» и «Пересылка сообщений по сети» в роли «Клиент» связаны. Теперь, нажав кнопку «Начать проход» в «Ворспроизведение сообщений» на стороне клиента, мы будем получать номер сообщения ни только в ярлыке «Здесь будет статус», но и в поле «Вывод данных» модуля на стороне сервера.
15. Кликаем по пункту меню «Video», появляется окно модуля «Нарезка сообщений»:



16. В поле «Введите ссылку» в верхней части окна вводим ссылку C:/InternalFrame_release_2020/Videos/textfile_java.txt – в ней расположен файл .txt с настройками.
17. По нажатию кнопки «Ввести в таблицу», таблица из трёх колонок заполнится аналогично приведённому образцу.
18. По нажатию кнопки «Обрезать файл» запустится утилита ffmpeg со всеми нужными параметрами.
19. В случае, если фрагмент файла, на который требуется обрезать исходный файл, уже существует, утилита спосит, переписать его, или нет.: alredy excists.Overwrite? [y/n]. Нужно вести y – переписать.
20. Обрезаный фрагмент должен пофвиться в корневой папке для исходных файлов, в подпапке Narezka_java/имя_исходного_файла/.

Отчёт о тестировании

Данные для тестирования содержатся в тестовых переменных проекта:

data1 - список тестовых интервалов времени для модуля «Воспроизведение сообщений».

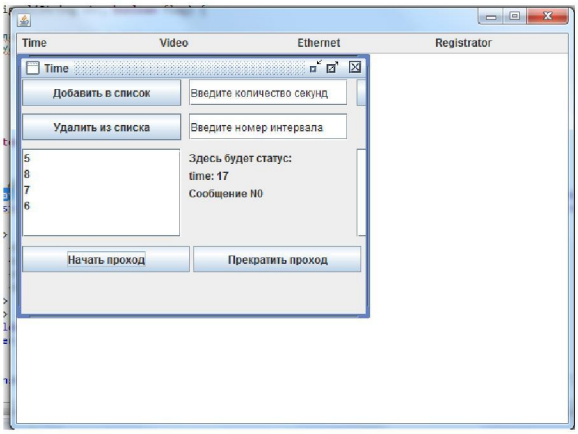
array - список с тестовой строкой для проверки работоспособности таблицы table1.

Тестовые видео формата .mp4 и файл с настройками (textfile_java.txt) находятся в папке Video.

Результаты тестирования:

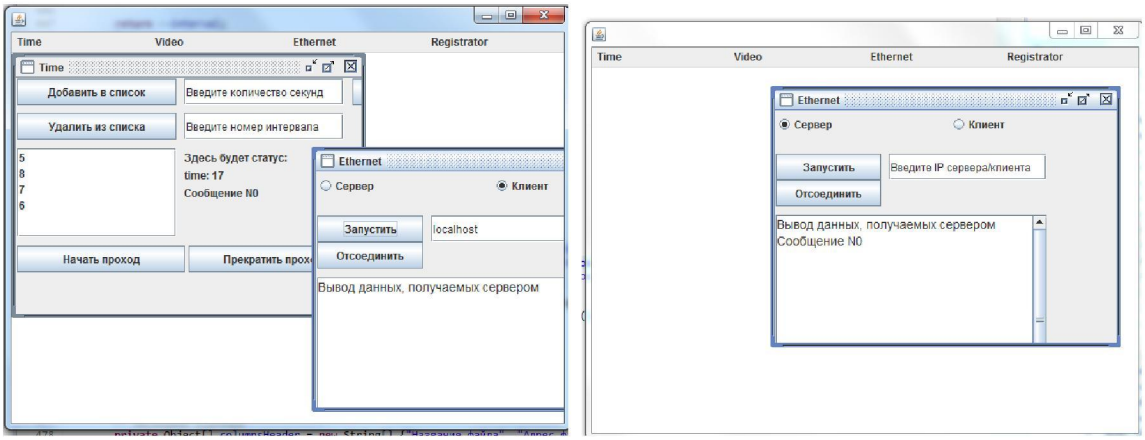
Модуль «Воспроизведение сообщений»:

Все операции, назначенные кнопкам на панели меню (Начать цикл, прервать цикл, удалить строку из списка, добавить строку в список), происходят корректно. При нажатии на кнопку «Начать цикл» в положенное время происходит показ каждого следующего сообщения:



Модуль «Обмен сообщениями по сети»:

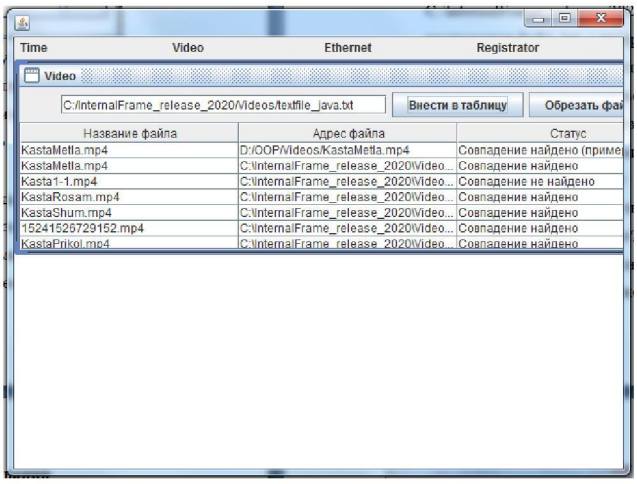
Операция «Запустить» происходит корректно для модуля и в роли «Клиент», и в роли «Сервер». В роли «Клиент» во время проведения прохода с показом сообщений от модуля «Воспроизведение сообщений» происходит пересылка сообщений «Серверу», если введён ip-адрес сервера, или localhost, и нажата кнопка «Соединить. В роли «Сервер», если нажата кнопка «Соединить», происходит вывод полученных сообщений в многострочное поле:



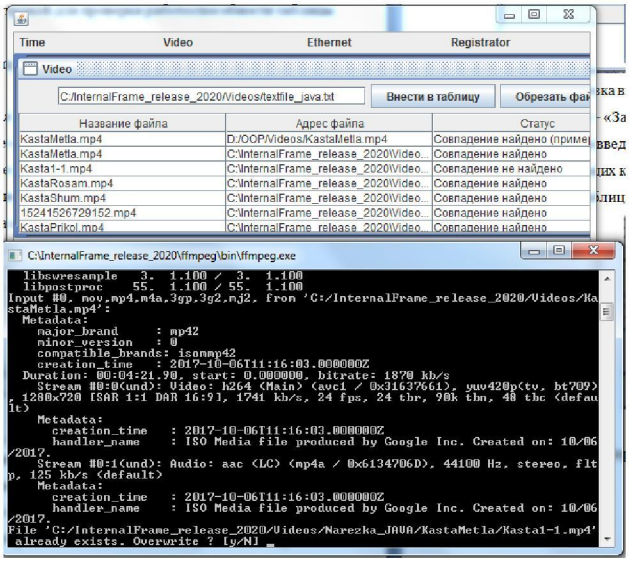
Модуль «Нарезка видео файлов»:

Обе операции – «Заполнение таблицы», «Обрезать файл», происходят корректно при введении тестовой строки с файлом настроек и нажатию соответствующих кнопок.

Заполнение таблицы:



Обрезать файл:



Модуль «Среда формирования окон»:

Все кнопки в меню корректно формируют окна.

Заключение

Цель - Создание программы для автоматизации процессов «Первичной Аккредитации» специалистов медицинского института ПетрГУ – нами выполнена..

Исходя из цели, нами были выполнены следующие задачи:

Создан модуль, позволяющий воспроизводить последовательность сообщений в заданные промежутки времени.

Создан модуль, позволяющий отправлять сообщения из модуля последовательности сообщений, отправлять их по сети, получать сообщения из сети, и отображать их в окне.

Создан модуль, позволяющий получить из поля путь к файлу со строкой пути к папке с исходными видео файлами, со списком названий и длительностью фрагментов, на которые файлы должны быть обрезаны. Подготовлен видео фрагмент путём его нарезки.

Создан модуль, который объединит три предыдущих модуля.

Скомпилирована программа и предоставлен пользователям лёгкий доступ к ней.

Исходный код

```
import javax.swing.*;
import java.util.*;
import java.io.*;
import java.time.*;

import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Timer;
import javax.swing.table.DefaultTableModel;
import java.awt.BorderLayout;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;

//String way = C:/InternalFrame_release_2020/Videos/textfile_java.txt;

public class InternalFrame extends JFrame {

	JDesktopPane desktopPane = new JDesktopPane();

	//**
	videoFrame intVideo = new videoFrame();
	timeFrame intTime = new timeFrame();
	ethernetFrame intEthernet = new ethernetFrame();
	registratorFrame intRegistrator = new registratorFrame();

	JMenuItem time1 = new JMenuItem( "Time" );
	JMenuItem video1 = new JMenuItem( "Video" );
	JMenuItem ethernet1 = new JMenuItem( "Ethernet" );
	JMenuItem registrator1 = new JMenuItem( "Registrator" );

	JMenuBar bar = new JMenuBar();
//

	//Флаг того, что данное новое: true - новое
   private boolean newData;

   public String Data;

	public InternalFrame() {
		//**
		/* Настройка окна Video */
		intVideo.setMaximizable(true);
		intVideo.setIconifiable(true);
		intVideo.setResizable(true);
		intVideo.setClosable(true);
		intVideo.setDefaultCloseOperation(JInternalFrame.HIDE_ON_CLOSE);

		intVideo.setSize(700,200);
		//intSound.setVisible(true);

		desktopPane.add(intVideo);
		//add(desktopPane);

		/* Настройка окна Time */
		intTime.setMaximizable(true);
		intTime.setIconifiable(true);
		intTime.setResizable(true);
		intTime.setClosable(true);
		intTime.setDefaultCloseOperation(JInternalFrame.HIDE_ON_CLOSE);

		intTime.setSize(400,300);
		//intTime.setVisible(true);

		desktopPane.add(intTime);
		//add(desktopPane);

		/* Настройка окна Ethernet */
		intEthernet.setMaximizable(true);
		intEthernet.setIconifiable(true);
		intEthernet.setResizable(true);
		intEthernet.setClosable(true);
```

```
                intEthernet.setDefaultCloseOperation(JInternalFrame.HIDE_ON_CLOSE);

                intEthernet.setSize(400,300);
                //intEthernet.setVisible(true);

                desktopPane.add(intEthernet);

                /* Настройка окна Registrator */
                intRegistrator.setMaximizable(true);
                intRegistrator.setIconifiable(true);
                intRegistrator.setResizable(true);
                intRegistrator.setClosable(true);
                intRegistrator.setDefaultCloseOperation(JInternalFrame.HIDE_ON_CLOSE);

                intRegistrator.setSize(400,300);
                //intSound.setVisible(true);

                desktopPane.add(intRegistrator);
                //add(desktopPane);

                add(desktopPane);

                //

                //**

            bar.add( time1 );
            bar.add( video1 );
            bar.add( ethernet1 );
            bar.add( registrator1 );

            //

    /* Событие "Нажатие кнопки time1" */
            time1.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {

                    System.out.println("time1");

                    intTime.setVisible(true);
               }
            });

            /* Событие "Нажатие кнопки video1" */
            video1.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {

                    System.out.println("video1");

                    intVideo.setVisible(true);

               }
            });

            /* Событие "Нажатие кнопки ethernet1" */
            ethernet1.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {

                    System.out.println("ethernet1");

                    intEthernet.setVisible(true);

               }
            });

            /* Событие "Нажатие кнопки registrator1" */
            registrator1.addActionListener(new ActionListener() {
               public void actionPerformed(ActionEvent e) {

                    System.out.println("registrator1");

                    intRegistrator.setVisible(true);
               }
            });

            setJMenuBar( bar );

        }
```

```
public static void main(String[] args) {
		InternalFrame iFrame = new InternalFrame();
		iFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		iFrame.setSize(640,480);
		iFrame.setVisible(true);

}

public class timeFrame extends JInternalFrame
{

		int interval;
   Timer timer;
   //Сумма секунд всех интервалов
		int summ;
		//Размер массива
		int countlist;
		//Текущий интервал
		int numberlist;
		//Флаг последнего сообщения
		boolean flag;
		boolean flag1;
		boolean flag2;

		String strA;
		String strB;
		int intA;
		int intB;

		// Модель списка
   private DefaultListModel<String> dlm = new DefaultListModel<String>();
   private DefaultListModel<String> dlm1 = new DefaultListModel<String>();
   private LinkedList <String> data1 = new LinkedList<> ();// { "5" ,"8"  ,"7","6"};
   private LinkedList <String> data3 = new LinkedList<> ();// { "5" ,"8"  ,"7","6"};
   int data2[];

   JLabel label2 = new JLabel ("");
   JLabel label3 = new JLabel ("time:");

   public timeFrame()
   {
      super("Time");

      data1.add(0, "5"); data1.add(1, "8"); data1.add(2, "7"); data1.add(3, "6");


      setDefaultCloseOperation(EXIT_ON_CLOSE);
      // Создание панели с возможностью позиционирования
      JPanel contents = new JPanel();
      contents.setLayout(null);

      for (int i = 0; i < data1.size(); i++) {
		dlm.add(i, data1.get(i));
      }

      JList<String> list1 = new JList<String>(dlm);
      JScrollPane Scr = new JScrollPane(list1);
      Scr.setBounds(1, 80, 180, 100);
      contents.add(Scr);

      JList<String> list2 = new JList<String>(dlm1);
      JScrollPane Scr1 = new JScrollPane(list2);
      Scr1.setBounds(380, 80, 180, 100);
      contents.add(Scr1);

      //Создание кнопки
      JButton add1 = new JButton("Добавить в список");
      // 1 - x, 2 - y, 3 - width, 4 - height
      add1.setBounds(1, 1, 180, 30);
      contents.add(add1);
     //Создание кнопки
      JButton add2 = new JButton("Удалить из списка");
      add2.setBounds(1, 40, 180, 30);
      contents.add(add2);
      contents.add(add1);
      //Создание кнопки
       JButton add5 = new JButton("Добавить в аудио");
       add5.setBounds(380, 1, 180, 30);
       contents.add(add5);
```

```
                    //Создание кнопки
                    JButton add3 = new JButton("Начать проход");
                    add3.setBounds(1, 190, 190, 30);
                    contents.add(add3);
                   //Создание кнопки
                    JButton add4 = new JButton("Прекратить проход");
                    add4.setBounds(195, 190, 190, 30);
                    contents.add(add4);

                    //Создание поля
                    JTextField bigField1 = new JTextField("Введите количество секунд", 25);
                    bigField1.setBounds(190, 1, 180, 30);
                    contents.add(bigField1);
                   //Создание поля
                    JTextField bigField2 = new JTextField("Введите номер интервала", 25);
                    bigField2.setBounds(190, 40, 180, 30);
                    contents.add(bigField2);
                   //Создание поля
                    JTextField bigField3 = new JTextField("Введите расположение папки с сообщениями", 25);
                    bigField3.setBounds(570, 1, 180, 30);
                    contents.add(bigField3);
                   //Добавление меток
                    JLabel label1 = new JLabel ("Здесь будет статус:");
                    label1.setBounds(190, 40, 180, 100);
                    contents.add(label1);
                   //Размещение выводящих меток
                    label3.setBounds(190, 60, 180, 100);
                    contents.add(label3);
                   //Размещение выводящих меток
                    label2.setBounds(190, 80, 180, 100);
                    contents.add(label2);

                   ////////

        add1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //dlm.add(dlm.getSize(), "-- Новая запись --");
        //validate();

            boolean flag1, flag2;

            try {
                        strA = bigField1.getText();
                        intA = Integer.parseInt(strA); //читаем с текстового поля текст, преобразуем в Double
                        //a += 2; //тут я прибавляю a = a + 2
            //jTextField1.setText(a.toString()); //вывожу обратно результат в текстовое поле
                        flag1 = true;
            } catch(NumberFormatException intA) {
                        strA = "NumberFormatException strA";
                        flag1 = false;
          //return false;
        } catch(NullPointerException intA) {
          //return false;
            strA = "NullPointerException strA";
            flag1 = false;
        }

            try {
                        strB = bigField2.getText();
                        intB = Integer.parseInt(strB); //читаем с текстового поля текст, преобразуем в Double
                        //a += 2; //тут я прибавляю a = a + 2
            //jTextField1.setText(a.toString()); //вывожу обратно результат в текстовое поле
                        flag2 = true;
            } catch(NumberFormatException intB) {
                        strB = "NumberFormatException strB";
                        flag2 = false;
          //return false;
        } catch(NullPointerException intB) {
          //return false;
            strB = "NullPointerException strB";
            flag2 = false;
        }

//Если оба поля введены верно, то дополняем список
            if ((flag1 == true) && (flag2 == true)) {

                        LinkedList <String> dlm_clone = new LinkedList<> (); //System.out.println(dlm.size());
```

```
                    for (int i = intB; i < dlm.size(); i++)
                    {
                            dlm_clone.add(dlm.get(i)); //Создание копии смещаемой части массива
                            System.out.println("dlm["+i + "]="+dlm.get(i));
                    }

                    for (int i = dlm.size() - 1; i >= intB; i--)
                    {
                      dlm.remove(i); //Удаление скопированных элементов
                    }

                    dlm.add(dlm.size(), strA); //Добавляем нужный элемент

                    int sizeB = dlm.size() - 1;

                    for (int i = 0; i < dlm_clone.size(); i++)
                    {
                            dlm.add(1 + sizeB, dlm_clone.get(i)); //Добавление остальных элементов
                    }

                    for (int i = 0; i < dlm.size(); i++)
                    {
                      System.out.println(dlm.get(i));
                    }

            }

            System.out.println("Добавлен!");
    }
});

add2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

            try {
                    strB = bigField2.getText();
                    intB = Integer.parseInt(strB); //читаем с текстового поля текст, преобразуем в Double
            } catch(NumberFormatException intB) {
                    strB = "NumberFormatException strB";
        } catch(NullPointerException intB) {
          strB = "NullPointerException strB";
        }

            if (dlm.size() > intB)
            {
            dlm.remove(intB); //Удаление скопированных элементов
            }

            System.out.println("Удалён!");
    }
});
//Начать проход
        add3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
            Scanner sc = new Scanner(System.in);
        System.out.print("Input seconds => : ");

        //String secs = "10"; //sc.nextLine();
        int delay = 1000;
        int period = 1000;
        timer = new Timer();

        countlist = list1.getModel().getSize();
        Object item;
        data2 = new int[countlist];
        summ = 0;

        for (int i = 0; i < countlist; i++) {
            //Получили объект строчки полосы прокрутки
          item = list1.getModel().getElementAt(i);
          //Преобразовали объект строки в строку
          String item1 = item.toString();
          //Преобразовали строчку в число
          data2[i] = Integer.parseInt(item1);
          summ += data2[i];
          System.out.println("Item = " + data2[i]);
          if (i == 0) { //(countlist - 1)) {
            numberlist = data2[0];
            System.out.println("numberlist = " + numberlist);
```

```
            }
        }
        System.out.println("summ = " + summ);
        countlist = 0;
        interval = summ;
        System.out.print("Input seconds => : ");
        System.out.println(interval);
        flag = true;
        timer.scheduleAtFixedRate(new TimerTask() {

            public void run() {
              //int tm = setInterval();
              //String time = toString(tm);
              label3.setText("time: "+ setInterval() ); //int k = setInterval();
              //System.out.println(setInterval()); System.out.println("!");

            }
        }, delay, period);

    }
});

add4.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        interval = 1;
        flag = false;
    }
});


    setContentPane(contents);

                ////////
                }

                ////////

                private int setInterval() {

                        if (interval == 1) {
                                if (flag == true)       {
                                        label2.setText("Сообщение N" + countlist); System.out.println("Сообщение N" + countlist);
                                        ethernetSignal("Сообщение N" + countlist, true);
                                }
                        timer.cancel();
                }
                        if (interval == (summ - numberlist)) {
                                label2.setText("Сообщение N" + countlist); System.out.println("Сообщение N" + countlist);
                                ethernetSignal("Сообщение N" + countlist, true);
                                summ -= numberlist;
                                ++countlist; System.out.println(countlist);
                                numberlist = data2[countlist];
                }

                return --interval;
                }

                private void ethernetSignal(String str, boolean flag) {

                        Data = str; //Отсылаем данное окну в роли клиента
                        newData = flag; //Устанавливаем флаг новизны данных
                }

                ///////

            }

            public class videoFrame extends JInternalFrame
            {

                        // Для примера: данные для таблиц
                private Object[][] array = new String[][] {{ "KastaMetla.mp4" , "D:/OOP/Videos/KastaMetla.mp4", "Совпадение найдено (пример)" }};
                private Object[] arraystr = new String[] {"00:01:05", "0:0:55", "D:/OOP/Videos/KastaMetla.mp4", "D:/OOP/Videos/Narezka/KastaMetla/Kasta1-1.mp4"};

                private Vector <String>  fileStrings  = new Vector<String>();
                private Vector <Vector <String>>  pathStrings  = new Vector<Vector <String>>();
                private Vector <Vector <String>>  fileEndStrings;
```

```
    private Vector <Vector <String>>  pathEndStrings;
    private Vector <Vector <String>>  convertStrings;
    private Vector <String>  convertFolders;
    private Vector <String>  pathStr;
    private String videoFolder;
    private String cutFolder;

    // Заголовки столбцов
    private Object[] columnsHeader = new String[] {"Название файла", "Адрес файла", "Статус"}; //, "Массив фрагментов"};

   /// Простая таблица
    JTable table1;
  // Модель данных таблицы
    private DefaultTableModel tableModel;

    public videoFrame() {
      super("Video");//, true, true, true, true );

            // Создание стандартной модели
      tableModel = new DefaultTableModel();
      // Определение столбцов
      tableModel.setColumnIdentifiers(columnsHeader);
      // Наполнение модели данными для примера
      for (int i = 0; i < array.length; i++)
        tableModel.addRow(array[i]);

     /// Простая таблица
      table1 = new JTable(tableModel);
      Box contents = new Box(BoxLayout.Y_AXIS);
      contents.add(new JScrollPane(table1));

      // Создание панели с возможностью позиционирования
      JPanel pnlButtons = new JPanel();

     //Создание поля
      JTextField bigField1 = new JTextField("Введите ссылку на файл с названиями и временем", 30);
      //bigField1.setBounds(130, 50, 180, 30);
      pnlButtons.add(bigField1);

    // Кнопка добавления колонки в модель VideoTest
      JButton table = new JButton("Внести в таблицу");
      pnlButtons.add(table);

    // Кнопка добавления колонки в модель VideoTest
      JButton obr = new JButton("Обрезать файл");
      pnlButtons.add(obr);

    // Слушатель обработки события
      obr.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

          System.out.println("Нажатие кнопки - Обрезать файл");

          System.out.println("pathEndStrings = " + pathEndStrings);
          System.out.println("fileEndStrings = " + fileEndStrings);

          //1. Конвертировать запись первого файла из списка в параметры для ffmpeg.exe

          convertOffile();    System.out.println("convertStrings = " + convertStrings);  System.out.println("convertFolders = " +
convertFolders);

          //2. Создать папку Narezka_java и вложенные папки для нарезаемых файлов

          folderOffile();

          //3. Запустить ffmpeg.exe при помощи процессов и передать параметры

          processOffile();

          //4. Ждать, пока не завершится нарезка файла и прислать подтверждение, статус "Нарезка завершена"

          endOffile();

        }
      });

    // Слушатель обработки события
      table.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
```

```
            System.out.println("Нажатие кнопки - внести в таблицу");

            //1. Прочитать построчно из файла

            //Для примера
            //String way = "D:\\OOP\\Videos\\textfile_java.txt";
            String way = bigField1.getText();
            readOffile(way);

            //2. Выделить в файле путь к папке

            videoFolder = fileStrings.get(0); System.out.println(videoFolder);
            cutFolder = fileStrings.get(1); System.out.println(cutFolder);

            //3. Осуществить поиск в этой папке и найти все файлы .mp4

            searchOffile(videoFolder);

            //4. Сравнить файлы из списка и найденные файлы.

            compareOffile();

            //5. Совпавшие файлы и их пути внести в таблицу

            addOffile();
        }
    });

    getContentPane().add(contents);
    getContentPane().add(pnlButtons, BorderLayout.NORTH);
    // Вывод окна
    //setSize(700, 200);
    //setVisible(true);
  }
/////////

  private void readOffile(String way) {

                    fileStrings = new Vector<String>();

                    try {
      File file = new File(way);
      //создаем объект FileReader для объекта File
      FileReader fr = new FileReader(file);
      //создаем BufferedReader с существующего FileReader для построчного считывания
      BufferedReader reader = new BufferedReader(fr);
      // считаем сначала первую строку
      String line = reader.readLine();
      while (line != null) {
        System.out.println(line);
        //Добавление в массив строк:
        fileStrings.add(line);
        // считываем остальные строки в цикле
        line = reader.readLine();
      }
    } catch (FileNotFoundException a) {
      a.printStackTrace();
    } catch (IOException b) {
      b.printStackTrace();
    }
          }

          private void searchOffile(String way) {

                    File folder = new File(way);

                    pathStrings = new Vector<Vector <String>>();

                    processFilesFromFolder(folder);
          }

          private void processFilesFromFolder(File folder) {

                    File[] folderEntries = folder.listFiles();
            String fileName;
            String filePath;
            boolean end;
```

```
            for (File entry : folderEntries)
            {
               if (entry.isDirectory())
               {
                  processFilesFromFolder(entry);
                  continue;
               } else {
                        // иначе вам попался файл, обрабатывайте его!
                     fileName = entry.getName();
                     //Проверка, соответствует ли формат файла
                     end = fileName.endsWith(".mp4");
                     if (end == true)
                     {
                              filePath = entry.getAbsolutePath(); System.out.println(fileName + " " + filePath);

                              pathStr = new Vector<String>();

                              //Добавление имён файлов и путей в выходной массив
                              pathStr.add(fileName);
                              pathStr.add(filePath);
                              //Не добавляется. Понять почему. Понять, почему глючит!!!
                              pathStrings.add(pathStr); System.out.println("!! pathStr " + pathStr);
                     }
               }
            }
         }

         private void compareOffile() {

                  pathEndStrings = new Vector<Vector <String>>();
                  fileEndStrings = new Vector<Vector <String>>();

                  int iSize = pathStrings.size(); System.out.println("path iSize = " + iSize);
                  Vector<String> fileName = new Vector<String>();
                  Vector<String> searchVector = new Vector<String>();
                  String nameStr;

                  System.out.println("!! pathStrings " + pathStrings);

                  for (int i = 0; i < iSize; i++) {
                           fileName = pathStrings.get(i);
                           nameStr = fileName.get(0);
                           //Поиск совпадения с массивом из списка
                           searchVector = searchOfarray(nameStr); System.out.println("compareOffile for- i = " + i + " nameStr
= " + nameStr); System.out.println("for- searchVector = " + searchVector );

                           int sSize = searchVector.size(); System.out.println("path sSize = " + sSize);
                           if (sSize > 0) {
                                    System.out.println("searchVector is not null!");
                                    //В этом случае мы нашли некое количество подходящих записей
                                    fileName.add("Совпадение найдено");
                                    //Добавляем в конечный массив имен/путей
                                    searchVector.add(0, nameStr);
                                    fileEndStrings.add(searchVector);

                           } else {
                                    System.out.println("searchVector is null!");
                                    //В этом случаемы ничего не нашли. Файла, как в списке, в дирректории не
существует.
                                    fileName.add("Совпадение не найдено");
                           }

                           //Добавляем в конечный массив имен/путей для таблицы
                           pathEndStrings.add(fileName);
                  }

         }

         private Vector<String>searchOfarray(String str) {

                  Vector<String> searchVector = new Vector<String>();
                  int index = fileStrings.indexOf(str); System.out.println("searchOfarray index = " + index);
                  int i = index + 1;
                  int iSize = fileStrings.size(); System.out.println("searchOfarray iSize = " + iSize);
                  String endStr;
                  boolean end;
                  //Если проверка по индексу что то нашла, то
                  if (index > 0) {
```

```
                        while(i < iSize) {
                                endStr = fileStrings.get(i);
                                //Проверка, соответствует ли формат файла
                end = endStr.endsWith(".mp4");
                if (end == false)
                {
                        searchVector.add(endStr);
                        i++;
                } else {
                        i = iSize;
                }
                        }

                }

                return searchVector;
        }

        //Добавление данных в таблицу
        private void addOffile() {

                //Для примера
                //String way = "D:\\OOP\\Videos\\textfile_java.txt";
                tableModel.setRowCount(1);

                Vector <String> pathStr = new Vector<String>();
                int iSize = pathEndStrings.size();

                for (int i = 0; i < iSize; i++) {
                        pathStr = pathEndStrings.get(i);
                        tableModel.addRow(pathStr);
                }
        }

        private void convertOffile() {

                convertStrings  = new Vector<Vector <String>>();
                convertFolders  = new Vector <String>();

                Vector <String> str = fileEndStrings.get(0);
                int strSize = str.size();
                String[] substr;
                String curstr;

                //0. Добавление пути в первую папку в начале массива
                arrayOffolders(cutFolder);

                for (int i = 1; i < strSize; i++) {
                        curstr = str.get(i);
                        substr = curstr.split(" "); System.out.println("substr = [" + substr[0] + "], [" + substr[1] + "], [" +
substr[2] + "]");

                        //1. Конвертировать время начала и время конца отрезка во время начала и длину отрезка,
который нужно вырезать.

                        int subSize = substr.length; //Получаем длину элементов массива
                        int startsubstr;
                        int finishsubstr;
                        String convsubstr;
                        String fileName;
                        String menuName;
                        String childPath;
                        String folderPath;
                        Vector <String> parameters;

                        startsubstr = timeOfsubstr(substr[1]); System.out.println("startsubstr = " + startsubstr);
                        finishsubstr = timeOfsubstr(substr[2]); System.out.println("finishsubstr = " + finishsubstr);

                        convsubstr = differnceOfsubstr(startsubstr, finishsubstr);

                        //2. Найти путь к исходному файлу в массиве.

                        fileName = str.get(0);
                        menuName = searchOfmenu(fileName); System.out.println("menuName = " + menuName);

                        //3. Создать путь к нарезанному файлу из переменной, папки Narezka и имени файла.
                        // Cutfolder + fileName - ".mp4" + "\\" + substr[0]

                        childPath = formOfpath(fileName, substr[0]); System.out.println("childPath = " + childPath);
```

```
                                        //4. Создать массив из строчек с параметрами из первого файла.
                                        //строки с параметрами: menuName, childPath, substr[1], convsubstr

                                        parameters = parametersOfstr(menuName, childPath, substr[1], convsubstr);
System.out.println("parameters = " + parameters);
                                        convertStrings.add(parameters);

                                        //5. Создать массив из строк с путёми для папок, которые тебуется создать
                                        // Cutfolder + fileName - ".mp4" + "\\"

                                        folderPath = foldOfPath(fileName); System.out.println("folderPath = " + folderPath);

                                        //6. Создать массив строчек  с параметрами для создания папок нарезки

                                        arrayOffolders(folderPath);
                                }
                        }

                        private void folderOffile() {

                                int cSize = convertFolders.size();

                                for (int i = 0; i < cSize; i++) {
                                        createOffolder (convertFolders.get(i));
                                }

                        }

                        private void createOffolder (String addr) {
                                // create an abstract pathname (File object)
                File f = new File(addr);

                // Проверка, может ли директория быть создана
                if (f.mkdir()) {
                    System.out.println("Директория создана");
                } else {
                   System.out.println("Директория не создана");
                }
                        }

                        private void processOffile() {
                                //Для примера
                                //String way = "D:\\OOP\\Videos\\textfile_java.txt";
                        createOfprocess("C:/InternalFrame_release_2020/ffmpeg/bin/ffmpeg.exe", convertStrings.get(0));
                        }

                        private void createOfprocess(String program, Vector<String>parameters) {

                                try {
                                        //arguments << "-ss" << "00:01:00" << "-t" << "00:01:00" << "-i" <<
"D:/OOP/Videos/KastaMetla.mp4" << "D:/OOP/Videos/Kasta1-1.mp4";
                                        System.out.println(program + " -ss " + parameters.get(2) + " -t " + parameters.get(3) + " -i " +
parameters.get(0) + " " + parameters.get(1));
                                        //ProcessBuilder builder = new ProcessBuilder("cmd.exe", "/c", "start",
"D:/OOP/NarezkaFailov/ffmpeg/bin/ffmpeg.exe", "-ss", "00:01:00", "-t", "00:01:00", "-i", "D:/OOP/Videos/KastaMetla.mp4",
"D:/OOP/Videos/Narezka_JAVA/KastaMetla/Kasta1-1.mp4");
                                        ProcessBuilder builder = new ProcessBuilder("cmd.exe", "/c", "start", program, "-ss",
parameters.get(2), "-t", parameters.get(3), "-i", parameters.get(0), parameters.get(1));
                                        Process process = builder.start();
                                        } catch (IOException e){ /*Закрытие оператора исключения и с последующем его описанием*/
                                                System.out.println("Процесс не создан!");
                                        }
                        }

                        private void endOffile() {

                        }

                        private int timeOfsubstr(String str) {

                                int convtime;

                                int sec_hour;
                                int sec_min;
                                int sec;

                                Pattern pt = Pattern.compile("(\\d+):(\\d+):(\\d+)");
                                Matcher m = pt.matcher(str);
```

```
                    if(m.find()) {
                    sec_hour = Integer.parseInt(m.group(1))*3600;
                    sec_min = Integer.parseInt(m.group(2))*60;
                    sec = Integer.parseInt(m.group(3)); System.out.println("sec_hour = " + sec_hour + ", sec_min = " + sec_min + ",
sec = " + sec);

                    convtime = sec_hour + sec_min + sec;

                    return convtime;
                    }
                    return 0;
            }

            private String differnceOfsubstr(int start, int finish) {

                    String conv = new String();
                    int intconv = finish - start; System.out.println("intconv = " + intconv);
                    LocalTime time = LocalTime.ofSecondOfDay(intconv);
                    // ISO Format
               DateTimeFormatter timeFormatter = DateTimeFormatter.ISO_LOCAL_TIME;
               conv = time.format(timeFormatter); System.out.println(conv);

                    return conv;

            }

            private String searchOfmenu(String fileName) {
                    // D:\\OOP\\Videos\\textfile_java.txt
                    String pathName = new String();
                    String endName;
                    Vector <String> pathStr;
                    int j;
                    int iSize;
                    int iIndex;

                    iSize = pathEndStrings.size();

                    for (int i = 0; i < iSize; i++) {

                            pathStr = pathEndStrings.get(i);
                            endName = pathStr.get(0); System.out.println("fileName = " + fileName + "   endName = " +
endName);

                            iIndex = fileName.indexOf(endName); System.out.println("iIndex = " + iIndex);

                            if (iIndex > -1) { System.out.println("if true");
                                    pathName = pathStr.get(1);

                                    i = iSize;

                                    return pathName;
                            }
                    }

                    return pathName;
            }

            private String formOfpath(String fileName, String substr) {
                    String childPath = new String();

                    int index = fileName.lastIndexOf(".mp4");
                    //Вырезать подстроку из строки
                    childPath = fileName.substring(0, index);
                    childPath = cutFolder + "/" + childPath + "/" + substr; //System.out.println("childPath = " + childPath);

                    return childPath;
            }

            private Vector <String> parametersOfstr(String menuName,String childPath, String substr, String convsubstr) {

                    Vector <String> str = new Vector <String>();

                    str.add(menuName.replace("\\", "/"));
                    str.add(childPath);
                    str.add(substr);
                    str.add(convsubstr);

                    return str;
            }
```

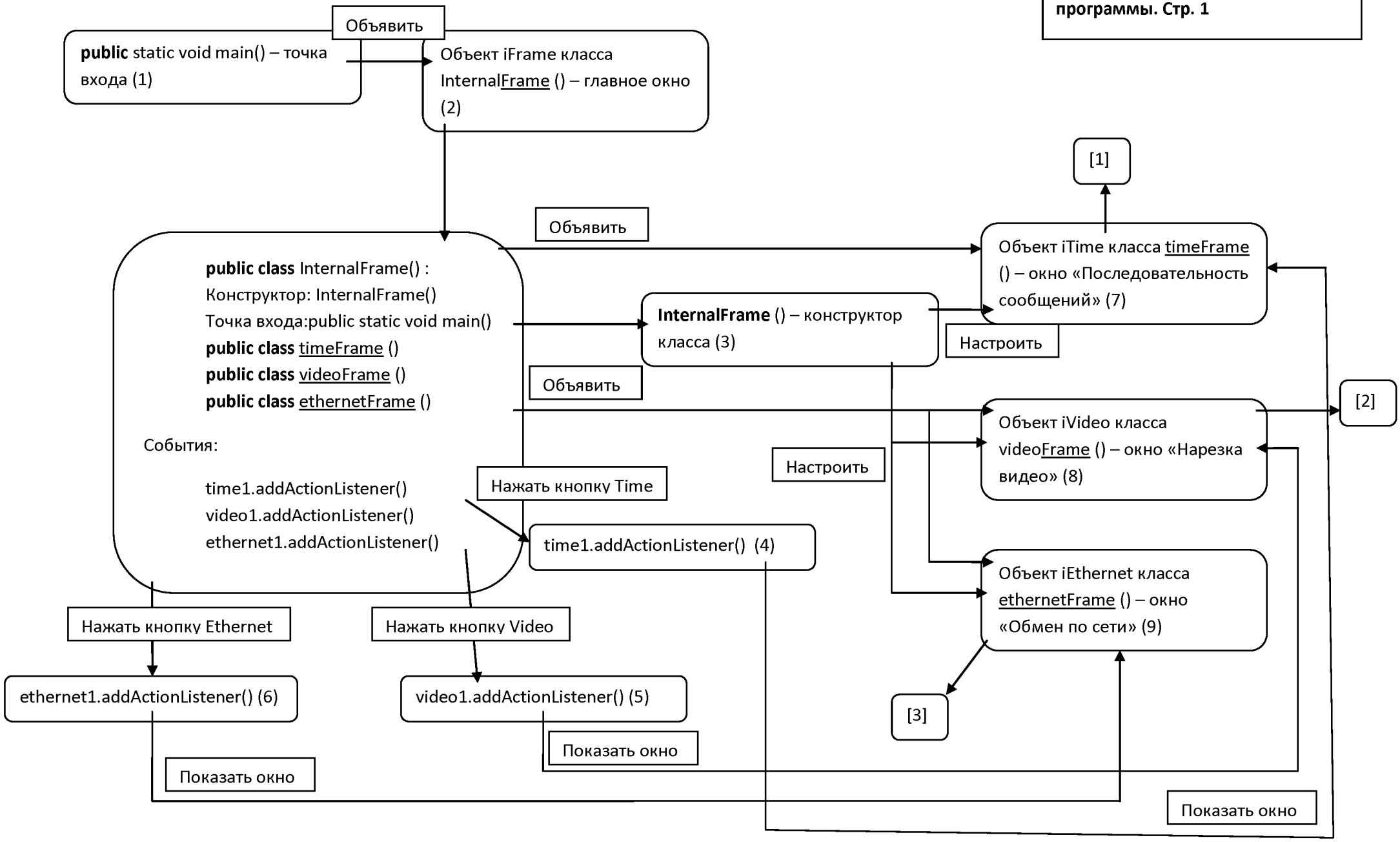
```
		private String foldOfPath(String fileName) {

			String childPath = new String();

			int index = fileName.lastIndexOf(".mp4");
			//Вырезать подстроку из строки
			childPath = fileName.substring(0, index);
			childPath = cutFolder + "/" + childPath + "/";

			return childPath;
		}

		private void arrayOffolders(String folder) {

			int index = convertFolders.indexOf(folder);

			//Если такого пути в папках ещё нет, то
			if(index == -1) {
				convertFolders.add(folder);
			}
		}

////////
	}

	public class ethernetFrame extends JInternalFrame
	{
		//Создание поля
	  JTextField bigField1;

	 //Создание многострочных полей
	  JTextArea textArea;

	  //Флаг разрешения передачи: true - разрешена
	  boolean allowData;

	  //Флаг того, что данное новое: true - новое
	  //private String newData;

	  //public String Data;

	  public ethernetFrame()
	  {
	    super("Ethernet");//, true, true, true, true );

	    setDefaultCloseOperation(EXIT_ON_CLOSE);
	    // Создание панели с возможностью позиционирования
	    JPanel contents = new JPanel();
	    contents.setLayout(null);

			ButtonGroup group = new ButtonGroup();
			JRadioButton serverButton = new JRadioButton("Сервер", false);
			serverButton.setBounds(1, 1, 180, 30);
			group.add(serverButton);
			contents.add(serverButton);

			JRadioButton clientButton = new JRadioButton("Клиент", true);
			clientButton.setBounds(200, 1, 180, 30);
			group.add(clientButton);
			contents.add(clientButton);

			//Создание кнопки
	    JButton add1 = new JButton("Запустить");
	    add1.setBounds(1, 50, 120, 30);
	    contents.add(add1);

	   //Создание кнопки
	    JButton add2 = new JButton("Отсоединить");
	    add2.setBounds(1, 80, 120, 30);
	    contents.add(add2);

			//Создание поля
	    bigField1 = new JTextField("Введите IP сервера/клиента", 25);
	    bigField1.setBounds(130, 50, 180, 30);
	    contents.add(bigField1);

	    //Создание многострочных полей
	    textArea = new JTextArea("Вывод данных, получаемых сервером\n", 20, 10);
```

```
    JScrollPane scrollPane = new JScrollPane(textArea);
    // Шрифт и табуляция
    textArea.setFont(new Font("Dialog", Font.PLAIN, 14));
    textArea.setTabSize(100);
    scrollPane.setBounds(1, 120, 310, 300);
    contents.add(scrollPane);

    //////////Для примера

    //allowData = true;
    newData = false;
    //setnewData("0");

   //Data = "Сообщение 1";

    //////////

    add1.addActionListener(

          new ActionListener() {

            public void actionPerformed( ActionEvent e ) {

             System.out.println("Программа запущена!");

             if (clientButton.isSelected()) {
                        System.out.println("Программа запущена как клиент!");

                        clientModuleThread clientModuleTh = new clientModuleThread();
                        clientModuleTh.start();

             }
             else if (serverButton.isSelected()) {
                        System.out.println("Программа запущена как сервер!");

                        serverModuleThread serverModuleTh = new serverModuleThread();
                        serverModuleTh.start();
             }

            }

          }

        );

    add2.addActionListener(

          new ActionListener() {

            public void actionPerformed( ActionEvent e ) {

             System.out.println("Программа отсоединена!");

            }

          }

        );


                        setContentPane(contents);
}

public class serverModuleThread extends Thread {

                        //Серверный сокет для принятия сообщений с клиента
                        private ServerSocket server;

                        //Соокет для отправки ответных сообщений с клиента
                        private Socket client;

                        private DataInputStream in;

                        private DataOutputStream out;

                        public void run() {
                  System.out.println("Переход внутрь потока модуля-сервера!");

                 //Пример:
```

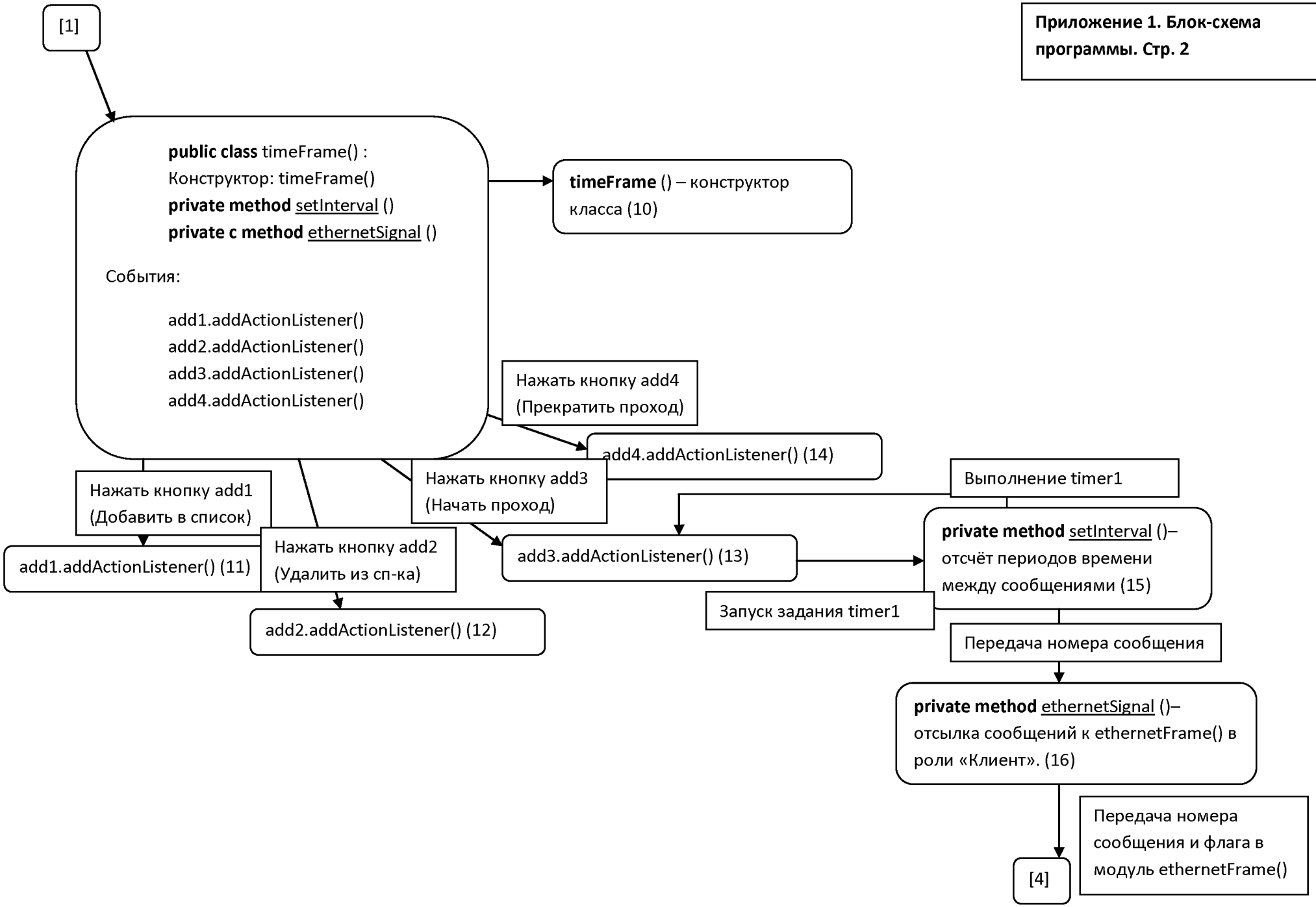
```
                        String inClient = "str";

                        String quit = "quit";

                        try {
                        //Серверный сокет для принятия сообщений с клиента
                        server = new ServerSocket(60);

                        System.out.println("Wait client...");

                        //Ожидание подключения клиента
                        client = server.accept();

                        System.out.println("Client connected!");

                        in = new DataInputStream(client.getInputStream());

                        out = new DataOutputStream(client.getOutputStream());

        //int i = 0;
        while(true) {

                                inClient = in.readUTF();

                                out.writeUTF(quit);

                    if (inClient == "quit") {

                        in.close();
                        out.close();
                        client.close();
                        }
                    else {
                        textArea.append(inClient);
                    }

        }

        } catch(IOException e){ /*Закрытие оператора исключения и с последующем его описанием*/

                        }
    }
}

public class clientModuleThread extends Thread {

            private Socket connect;

            private DataOutputStream out;

            private DataInputStream in;

            public void run() {
          System.out.println("Переход внутрь потока модуля-клиента!");

         //Пример:
                        String str1 = "Сообщение 1";
                        String answer;
                        String newDataflag;

                    try {
                        connect = new Socket(bigField1.getText(),60);

                        out = new DataOutputStream(connect.getOutputStream());

                                                in = new DataInputStream(connect.getInputStream());

                    while(true) {

                        //Блок ухода в сон и ожидания обработки. Без этого не работает посылка сообщений
                        try {
                                Thread.sleep(10);
                        } catch (InterruptedException e) {
                        System.out.println("Работа потока была прервана");
                        break;
                    }

                                                if (newData == true) {
                                            //Отправка данных в клиент в стандартный поток
```

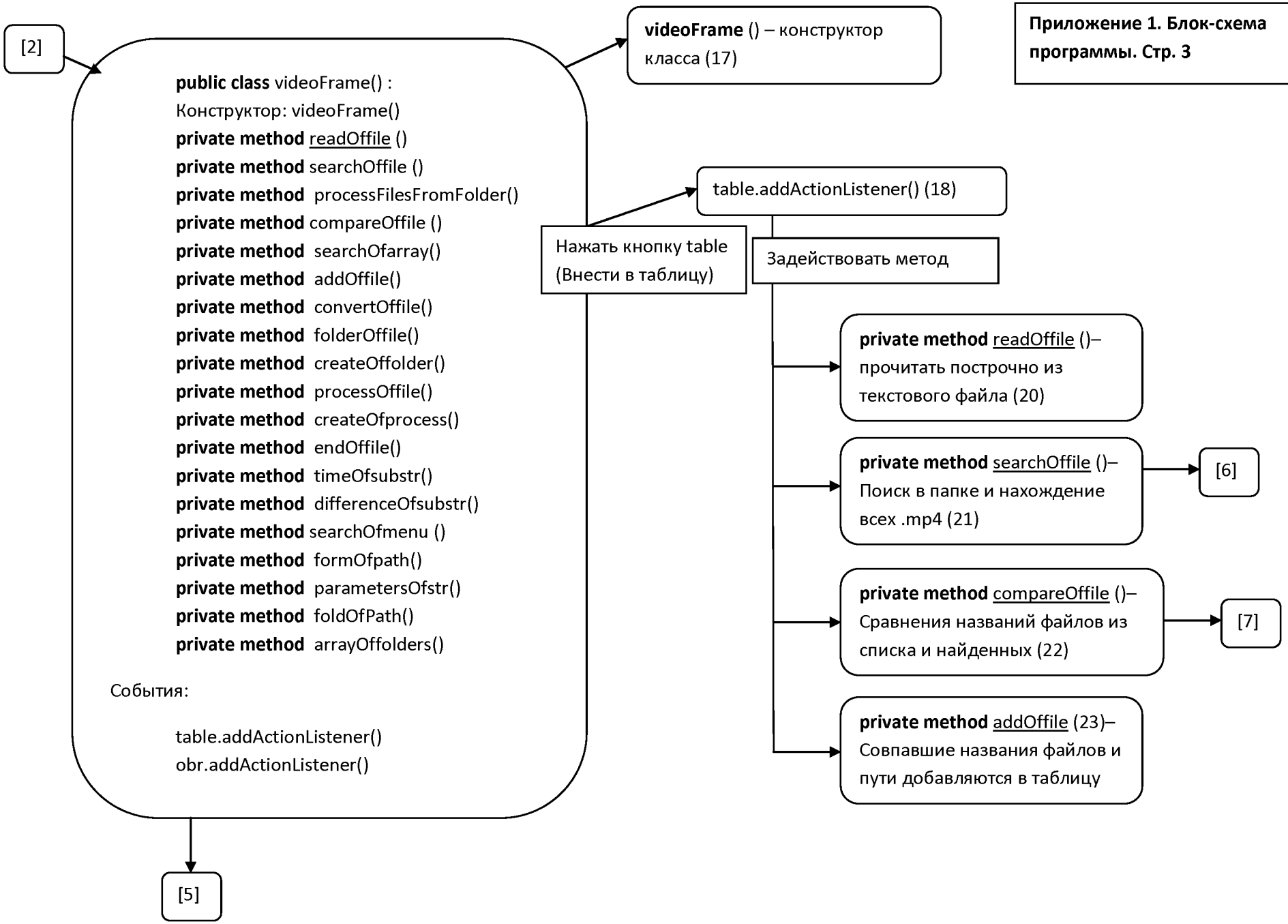
```
                                                                        out.writeUTF(Data + "\n");
                                                                        out.flush();

                                                                    //Ждём ответа от сервера
                                                                    answer = in.readUTF();

                                                                    newData = false;

                                                                    if (answer == "quit") {
                                                                        out.close();
                                                                        connect.close();

                                                                        }

                                                                    }

                                                }

                                                        } catch(IOException e){ /*Закрытие оператора исключения и с последующем его описанием*/
                                                                System.out.println("Работа потока была прервана");
                                                        }

                                                //}
                                        }
                                }
                        }

                public class registratorFrame extends JInternalFrame
                {
                   public registratorFrame()
                   {
                      super("Registrator");

                      JPanel contents = new JPanel();
                      contents.setLayout(null);

                     //Создание кнопки
                      JButton add1 = new JButton("Получить время и видео");
                      add1.setBounds(1, 1, 180, 30);
                      contents.add(add1);

                     //Создание кнопки
                      JButton add2 = new JButton("Сформировать список");
                      add2.setBounds(1, 30, 180, 30);
                      contents.add(add2);

                      setContentPane(contents);

                      /* Событие "Нажатие кнопки "Получить время и видео" */
                      add1.addActionListener(new ActionListener() {
                                public void actionPerformed(ActionEvent e) {

                                        System.out.println("Кнопка получения данных о времени и видео с видеорегистратора");
                                    }
                                });

                      /* Событие "Нажатие кнопки "Сформировать список" */
                      add2.addActionListener(new ActionListener() {
                                public void actionPerformed(ActionEvent e) {

                                        System.out.println("Кнопка формирования лога для нарезки файлов из логов модуля EthernetFrame и полученных\n с регистратора данных о времени.");
                                    }
                                });
                   }
                }
}
```

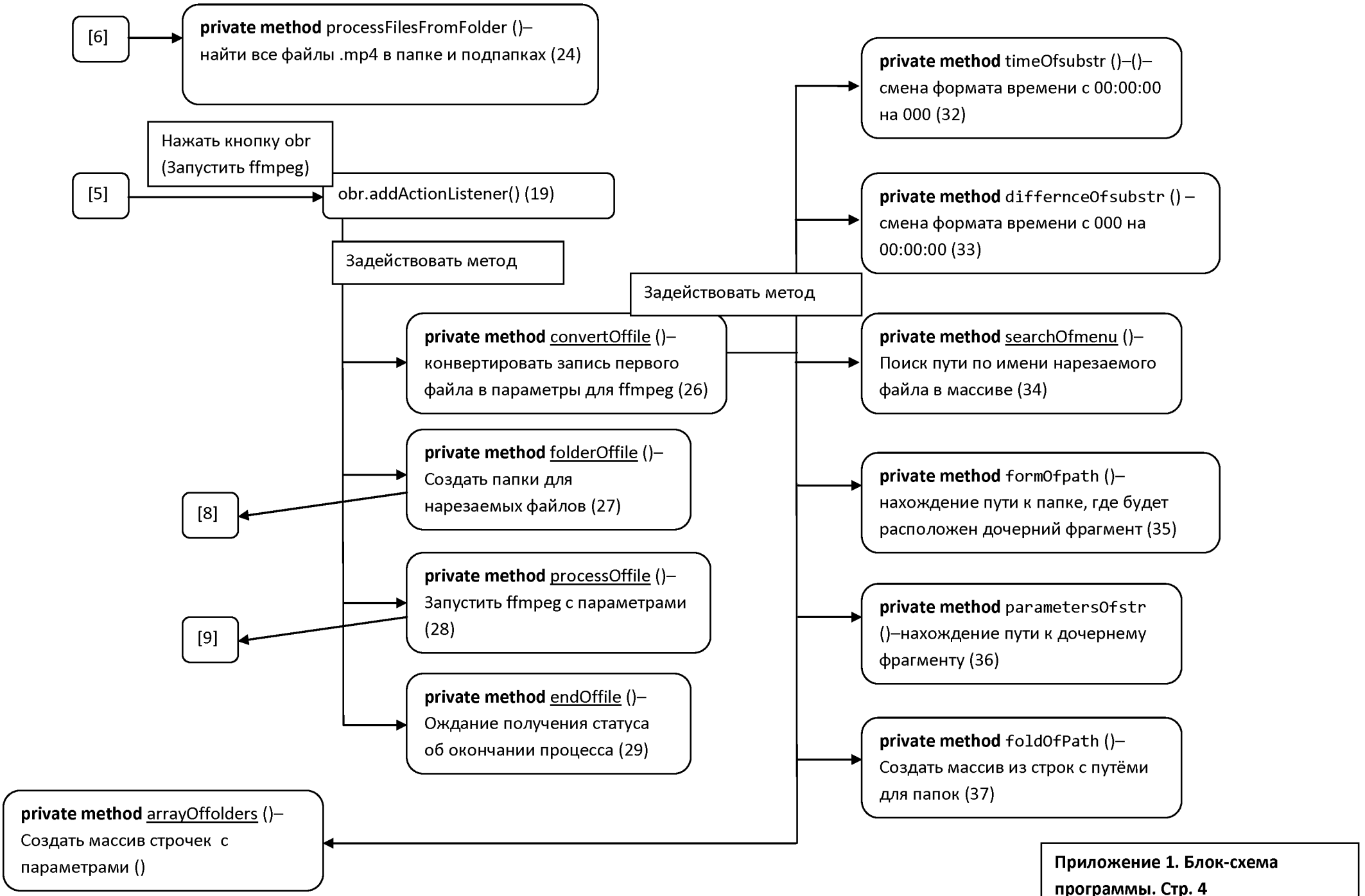
Приложение 1. Блок-схема программы. Стр. 1



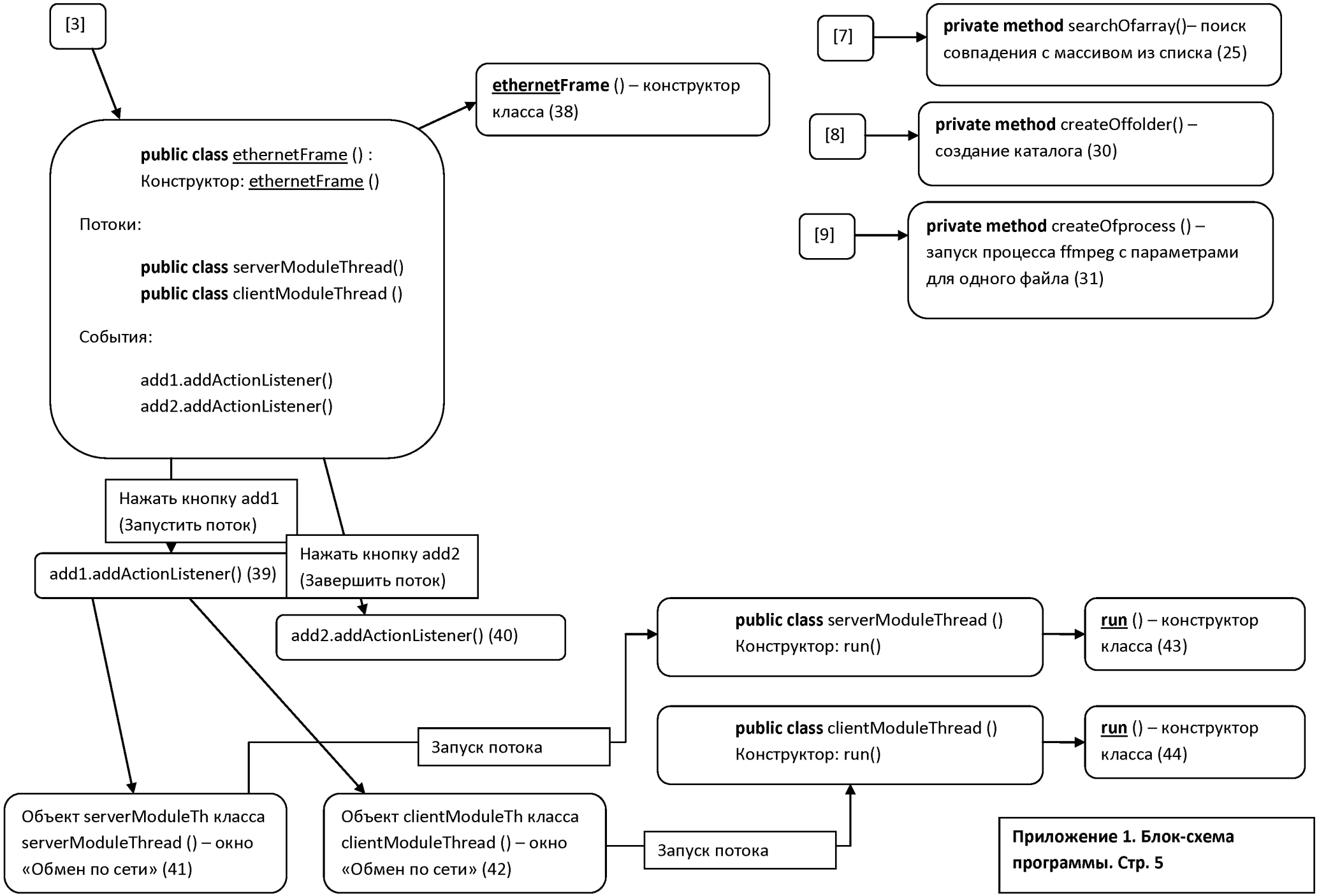
Приложение 1. Блок-схема программы. Стр. 2




[2]
public class videoFrame() :
Конструктор: videoFrame()
private method readOffile ()
private method searchOffile ()
private method processFilesFromFolder()
private method compareOffile ()
private method searchOfarray()
private method addOffile()
private method convertOffile()
private method folderOffile()
private method createOffolder()
private method processOffile()
private method createOfprocess()
private method endOffile()
private method timeOfsubstr()
private method differenceOfsubstr()
private method searchOfmenu ()
private method formOfpath()
private method parametersOfstr()
private method foldOfPath()
private method arrayOffolders()
События:
table.addActionListener()
obr.addActionListener()
[5]
videoFrame () – конструктор класса (17)
table.addActionListener() (18)
Нажать кнопку table (Внести в таблицу)
Задействовать метод
private method readOffile ()– прочитать построчно из текстового файла (20)
private method searchOffile ()– Поиск в папке и нахождение всех .mp4 (21)
[6]
private method compareOffile ()– Сравнения названий файлов из списка и найденных (22)
[7]
private method addOffile (23)– Совпавшие названия файлов и пути добавляются в таблицу



Приложение 1. Блок-схема программы. Стр. 4



Приложение 1. Блок-схема программы. Стр. 5