ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Преподаватель департамента программной инженерии Факультета компьютерных наук  Национального исследовательского университета «Высшая школа экономики»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е.А.Сибирцева  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2016 г. | УТВЕРЖДАЮ Академический руководитель Образовательной программы «Программная инженерия»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.В.Шилов  «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2016 г. |

**Программа прямых видеотрансляций с привязкой по геопозиции с применением линейных фильтров  
Сервер**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**RU. 17701729. 505900-01 12 1-1-ЛУ



Исполнитель: студент группы 142ПИ  
\_\_\_\_\_\_\_\_\_\_\_/А.А.Смилянский/  
«\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2016 г.

**УТВЕРЖДЕНО**  
**RU. 17701729. 505900-01 12 1-1-ЛУ**

**Программа прямых видеотрансляций с привязкой по геопозиции с применением линейных фильтров  
Сервер**

**Текст программы**

RU. 17701729. 505900-01 12 1-1

Листов 45



## СОДЕРЖАНИЕ

[1. Текст программы 3](#_Toc451206453)

[1.1. Класс Controller.java 3](#_Toc451206454)

[1.2. Класс Main.java 3](#_Toc451206455)

[1.3. Класс StreamWindow.java 10](#_Toc451206456)

[1.4. Класс Utils.java 14](#_Toc451206457)

[1.5. Класс BufferManager.java 15](#_Toc451206458)

[1.6. Класс ImageFrameBuffer.java 20](#_Toc451206459)

[1.7. Класс Coordinate.java 22](#_Toc451206460)

[1.8. Класс PictureData.java 23](#_Toc451206461)

[1.9. Класс StreamData.java 24](#_Toc451206462)

[1.10. Класс DataListener.java 25](#_Toc451206463)

[1.11. Класс ErrorListener.java 26](#_Toc451206464)

[1.12. Класс GeoListener.java 26](#_Toc451206465)

[1.13. Класс PoolListener.java 27](#_Toc451206466)

[1.14. Класс ServerListener.java 27](#_Toc451206467)

[1.15. Класс SimpleStreamListener.java 27](#_Toc451206468)

[1.16. Класс StreamListener.java 28](#_Toc451206469)

[1.17. Класс HandleIncomingConnection.java 28](#_Toc451206470)

[1.18. Класс Server.java 31](#_Toc451206471)

[1.19. Класс StreamPool.java 34](#_Toc451206472)

[1.20. Класс UserStream.java 36](#_Toc451206473)

[Лист регистрации изменений 45](#_Toc451206474)

1. **Текст программы**
   1. **Класс Controller.java**

**package** ui.main;  
  
**import** io.Server;  
**import** javafx.event.ActionEvent;  
**import** javafx.fxml.FXML;  
  
**public class** Controller {  
 **private** Main **single**;  
  
 **public** Controller() {  
 **single** = Main.*getInstance*();  
 }  
  
 @FXML  
 **public void** handleStartServerEvent(ActionEvent event) {  
 **switch** (**single**.getServerStatus()){  
 **case** -1:  
 **single**.startServer();  
 **break**;  
 **case** 1:  
 **single**.stopServer();  
 **break**;  
 }  
 }  
  
 @FXML  
 **public void** handleRefreshSettingsEvent(ActionEvent event) {  
 **single**.checkSettings();  
 }  
}

* 1. **Класс Main.java**

**package** ui.main;  
  
**import** data.Listeners.ErrorListener;  
**import** data.Listeners.PoolListener;  
**import** data.Listeners.ServerListener;  
**import** io.Server;  
**import** io.UserStream;  
**import** javafx.application.Application;  
**import** javafx.application.Platform;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.event.ActionEvent;  
**import** javafx.event.EventHandler;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.control.\*;  
**import** javafx.scene.layout.HBox;  
**import** javafx.scene.paint.Color;  
**import** javafx.scene.shape.Circle;  
**import** javafx.scene.text.Text;  
**import** javafx.stage.Stage;  
**import** javafx.stage.WindowEvent;  
**import** ui.stream.StreamWindow;  
  
**import** java.io.BufferedReader;  
**import** java.io.InputStreamReader;  
**import** java.net.URL;  
**import** java.text.SimpleDateFormat;  
**import** java.util.Calendar;  
  
**public class** Main **extends** Application **implements** PoolListener, ServerListener, ErrorListener {  
 */\*\*  
 \* Текущий объект, т.к. сервер может быть лишь один  
 \*/* **private static** Main *single*;  
  
 */\*\*  
 \* Окно проигрывания стрима  
 \*/* **private** StreamWindow **streamWindowWindow**;  
 */\*\*  
 \* Стейдж окна прогирывания стрима  
 \*/* **private** Stage **streamStage**;  
 */\*\*  
 \* Главная сцена  
 \*/* **private** Scene **mainWindow**;  
 */\*\*  
 \* Круг - индикатор доступности сервера  
 \*/* **private** Circle **statusCircle**;  
 */\*\*  
 \* Поле порта сервера  
 \*/* **private** Text **port**;  
 */\*\*  
 \* Поле адреса сервера  
 \*/* **private** Text **ipAdress**;  
 */\*\*  
 \* Логер  
 \*/* **private** TextArea **log**;  
 */\*\*  
 \* Перезапускает проверку настроек  
 \*/* **private** Button **rerunSettingsCheck**;  
 */\*\*  
 \* Запуск сервера  
 \*/* **private** Button **launchServer**;  
  
 **private** ListView<data.Abstractions.StreamData> **list**;  
  
 *// Do not delete this* **private** Server **server**;  
 **private int serverStatus** = -1;  
 **private** ObservableList<data.Abstractions.StreamData> **data** = FXCollections.*observableArrayList*();  
  
  
 **public** Main() {  
  
 }  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception {  
 *single* = **this**;  
  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"main.fxml"**));  
  
 **mainWindow** = **new** Scene(root, 490, 390);  
  
 primaryStage.setTitle(**"StreamEra server"**);  
 primaryStage.setScene(**mainWindow**);  
 primaryStage.setResizable(**false**);  
  
 initForms();  
 checkSettings();  
  
 primaryStage.setOnCloseRequest(**new** EventHandler<WindowEvent>() {  
 @Override  
 **public void** handle(WindowEvent event) {  
 System.exit(0);  
 }  
 });  
  
 **while** (!primaryStage.isShowing()) {  
 **try** {  
 primaryStage.show();  
 } **catch** (Exception e) {  
 Thread.sleep(1000);  
 }  
 }  
  
 }  
  
 */\*\*  
 \* Инициация полей - видов  
 \*/* **public void** initForms() {  
 **if** (mainWindow == **null**) {  
 **return**;  
 }  
  
 *// Статус сервера* statusCircle = (Circle) mainWindow.lookup(**"#crc\_serverStatus"**);  
  
 *// Окно адреса* ipAdress = (Text) mainWindow.lookup(**"#lbl\_ipAdress"**);  
 ipAdress.setText(**"Waiting for ip..."**);  
  
 *// Log* log = (TextArea) mainWindow.lookup(**"#txt\_log"**);  
 log.setEditable(**false**);  
 log.setFocusTraversable(**false**);  
  
 *// Кнопка запуска перепроверки настроек* rerunSettingsCheck = (Button) mainWindow.lookup(**"#btn\_rerun"**);  
  
 *//поле порта сервера* port = (Text) mainWindow.lookup(**"#lbl\_port"**);  
 port.setText(**"Waiting for port..."**);  
  
 *//кнопка запуска сервера* launchServer = (Button) mainWindow.lookup(**"#btn\_launch"**);  
  
 list = (ListView<data.Abstractions.StreamData>) mainWindow.lookup(**"#lv\_liveNow"**);  
  
 list.setItems(data);  
 list.setCellFactory(param -> **new** ListCellX());  
  
  
*// list.setItems(data);  
// list.setCellFactory(list1 -> new StreamData());* setStatusCircle(-1);  
 }  
  
  
 **private class** ListCellX **extends** ListCell<data.Abstractions.StreamData> {  
 **private** Button **deleteButton** = **new** Button(**"Delete"**);  
 **private** Button **startButton** = **new** Button(**"Start"**);  
 **private** Label **name** = **new** Label();  
 **private** HBox **hBox** = **new** HBox();  
  
 **public** ListCellX() {  
 **super**();  
 **hBox**.getChildren().addAll(**startButton**, **deleteButton**, **name**);  
 }  
  
 @Override  
 **protected void** updateItem(data.Abstractions.StreamData item, **boolean** empty) {  
 **super**.updateItem(item, empty);  
 setText(**null**);  
 **if** (item == **null** || empty) {  
 setGraphic(**null**);  
 **return**;  
 }  
 **deleteButton**.setOnAction(**new** EventHandler<ActionEvent>() {  
 @Override  
 **public void** handle(ActionEvent event) {  
 **server**.closeStream(item.getId());  
 **data**.remove(item);  
 **list**.getItems().remove(item);  
 }  
 });  
 **deleteButton**.setStyle(**"-fx-font: 12 arial; -fx-base: #ff4040;"**);  
 **startButton**.setOnAction(**new** EventHandler<ActionEvent>() {  
 @Override  
 **public void** handle(ActionEvent event) {  
 writeLog(**"Stream "** + item.getId() + **" starting..."**);  
  
 **new** Thread() {  
 @Override  
 **public void** run() {  
 UserStream userStream = **server**.openStream(item.getId());  
 **if** (userStream == **null**) {  
 onError(**"Can't find correct stream."**);  
 **return**;  
 }  
  
 *// здесь код не предназначенный для изменения экрана* StreamWindow streamWindow = **new** StreamWindow(item.getPictureData().getWidth(), item.getPictureData().getHeight());  
 Platform.runLater(**new** Runnable() {  
 **public void** run() {  
 **try** {  
 streamWindow.start(**new** Stage());  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 streamWindow.setUserStream(userStream);  
 }  
 }.start();  
 }  
 });  
 startButton.setStyle(**"-fx-font: 12 arial; -fx-base: #67c4a7;"**);  
 **if** (item.getName() != **null** && !item.getName().equals(**"null"**))  
 name.setText(item.getName() + **":\t"** + item.getId());  
 **else** name.setText(item.getId());  
 setGraphic(hBox);  
 }  
 }  
  
  
 *// Server options* **public void** startServer() {  
 server = **new** Server(**this**);  
 server.setPoolListener(**this**); *// изменение пула* server.setErrorListener(**this**); *// ошибки* server.setServerListener(**this**); *// состояние сервера* server.start();  
 }  
  
 **public void** stopServer() {  
 **if** (getServerStatus() == 0 || getServerStatus() == -1) {  
 **return**;  
 }  
 server.closeServer();  
 data.removeAll();  
 list.refresh();  
 }  
  
  
 *// UI options* **public void** checkSettings() {  
 *// Получение и установка public ip* Thread getPublicIp = **new** Thread() {  
 @Override  
 **public void** run() {  
 **super**.run();  
 URL whatismyip = **null**;  
 **try** {  
 port.setText(String.valueOf(8585));  
  
 whatismyip = **new** URL(**"http://checkip.amazonaws.com"**);  
 BufferedReader in = **new** BufferedReader(**new** InputStreamReader(whatismyip.openStream()));  
 String ip = in.readLine(); *//you get the IP as a String* ipAdress.setText(ip);  
 writeLog(**"Current ip received: "** + ip);  
 } **catch** (Exception e) {  
 writeLog(**"Problems with getting ip adress from <http://checkip.amazonaws.com>, check internet connection."**);  
 ipAdress.setText(**"Could not get server ip."**);  
 }  
 }  
 };  
 getPublicIp.start();  
 }  
  
 **public void** writeLog(String message) {  
 *// Запишь timestamp и сообщения в логер* Calendar calendar = Calendar.getInstance();  
 SimpleDateFormat sdf = **new** SimpleDateFormat(**"HH:mm:ss: "**);  
 log.setText(log.getText() + sdf.format(calendar.getTime()) + message + **"\n"**);  
 }  
  
  
 *// Getters - setters* **public static** Main getInstance() {  
 **return** single;  
 }  
  
  
 *// Changing ui* **private void** setStatusCircle(**int** code) {  
 *// -1 = offline  
 // 0 = disconecting or connecting  
 // 1 = online* serverStatus = code;  
 **switch** (code) {  
 **case** -1:  
 statusCircle.setFill(Color.RED);  
 launchServer.setText(**"START SERVER"**);  
 **break**;  
 **case** 0:  
 statusCircle.setFill(Color.WHITE);  
 **break**;  
 **case** 1:  
 statusCircle.setFill(Color.GREEN);  
 launchServer.setText(**"STOP SERVER"**);  
 **break**;  
 **default**:  
 statusCircle.setFill(Color.DARKGREY);  
 }  
 }  
  
 **public int** getServerStatus() {  
 **return** serverStatus;  
 }  
  
 *// Listening for events in pool or server or events for errors* @Override  
 **public void** onStreamAdded(UserStream userStream) {  
 Platform.runLater(**new** Runnable() {  
 @Override  
 **public void** run() {  
 data.add(userStream.getStreamData());  
 list.refresh();  
 }  
 });  
 }  
  
 @Override  
 **public void** onStreamDisconnect(UserStream userStream) {  
 Platform.runLater(**new** Runnable() {  
 @Override  
 **public void** run() {  
 data.remove(userStream.getStreamData());  
 list.refresh();  
 }  
 });  
  
 }  
  
 @Override  
 **public void** onServerThinking() {  
 setStatusCircle(0);  
 }  
  
 @Override  
 **public void** onServerClosed() {  
 setStatusCircle(-1);  
 }  
  
 @Override  
 **public void** onServerOpen() {  
 setStatusCircle(1);  
 }  
  
 @Override  
 **public void** onError(String message) {  
 Platform.runLater(**new** Runnable() {  
 @Override  
 **public void** run() {  
 writeLog(message);  
 }  
 });  
 }  
}

* 1. **Класс StreamWindow.java**

**package** ui.stream;  
  
**import** data.Abstractions.Coordinate;  
**import** data.Abstractions.StreamData;  
**import** data.Listeners.DataListener;  
**import** data.Listeners.GeoListener;  
**import** io.UserStream;  
**import** javafx.application.Application;  
**import** javafx.application.Platform;  
**import** javafx.embed.swing.SwingFXUtils;  
**import** javafx.event.EventHandler;  
**import** javafx.fxml.FXMLLoader;  
**import** javafx.scene.Parent;  
**import** javafx.scene.Scene;  
**import** javafx.scene.image.ImageView;  
**import** javafx.scene.image.WritableImage;  
**import** javafx.scene.layout.VBox;  
**import** javafx.scene.text.Text;  
**import** javafx.stage.Stage;  
**import** javafx.stage.WindowEvent;  
**import** ui.main.Main;  
  
**import** java.awt.image.BufferedImage;  
**import** java.io.IOException;  
**import** java.util.LinkedList;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 05.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public class** StreamWindow **extends** Application **implements** DataListener, GeoListener {  
 */\*\*  
 \* Максимальное кол-во картинок в очереди для отображения.  
 \*/* **private static final int *MAX\_BUFFER*** = 15;  
 */\*\*  
 \* Дефолтная картинка и последний фрейм-картинка.  
 \*/* BufferedImage **mImage**, **mLastFrame**;  
 */\*\*  
 \* Последний фрейм в виде fx  
 \*/* WritableImage **currentFrame**;  
 **private** Scene **streamScene**;  
 **private** Stage **primaryStage**;  
 **private int mWidth**, **mHeight**;  
 */\*\*  
 \* "Очередь" картинок для отображения  
 \*/* **private** LinkedList<BufferedImage> **mQueue** = **new** LinkedList<>();  
  
 *// objects* **private** Main **parent**;  
 **private** StreamData **streamData**;  
 **private** UserStream **userStream**;  
  
 *// UI* **private** Text **latitude**;  
 **private** Text **longitude**;  
 **private** VBox **geoposition**;  
 **private** ImageView **stream**;  
 **private** Coordinate **lastCoordinate**;  
  
  
 **public** StreamWindow(**int** width, **int** height) {  
 **mWidth** = width;  
 **mHeight** = height;  
 }  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
  
 *// start-stop* @Override  
 **public void** start(Stage primaryStage) **throws** Exception {  
 Parent root = FXMLLoader.*load*(getClass().getResource(**"stream.fxml"**));  
*// streamScene = new Scene(root, mWidth - 10, mHeight - 10);* **streamScene** = **new** Scene(root, 520, 390);  
 **this**.**primaryStage** = primaryStage;  
 initforms();  
  
 primaryStage.setOnCloseRequest(**new** EventHandler<WindowEvent>() {  
 @Override  
 **public void** handle(WindowEvent event) {  
 **try** {  
 stop();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 **geoposition**.setOpacity(0);  
  
 primaryStage.setScene(**streamScene**);  
 primaryStage.setTitle(**"Video translation from android client"**);  
 primaryStage.setResizable(**false**);  
 primaryStage.show();  
 }  
  
  
 @Override  
 **public void** stop() **throws** Exception {  
 **if** (userStream == **null**){  
 **return**;  
 }  
  
 **try** {  
 userStream.requestWait();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 userStream.setGeoListener(**null**);  
 userStream.setDataListener(**null**);  
  
 **super**.stop();  
 }  
  
  
 *// func* **private void** initforms() {  
 **if** (streamScene == **null**)  
 **return**;  
 latitude = (Text) streamScene.lookup(**"#txt\_latitude"**);  
 longitude = (Text) streamScene.lookup(**"#txt\_longitude"**);  
 geoposition = (VBox) streamScene.lookup(**"#vbox\_geo"**);  
 stream = (ImageView) streamScene.lookup(**"#view\_stream"**);  
 }  
  
 **private void** repaint() {  
 *// отображаем картинки по очереди из "очереди"* **synchronized** (mQueue) {  
 **if** (mQueue.size() > 0) {  
 mLastFrame = mQueue.poll();  
 }  
 }  
  
  
 *// прорисовываем картинку* **if** (mLastFrame != **null**) {  
 currentFrame = SwingFXUtils.toFXImage(mLastFrame, **null**);  
 **if** (currentFrame == **null** || stream == **null**){  
 **return**;  
 }  
 stream.setImage(currentFrame);  
 } **else if** (mImage != **null**) {  
 *// или заглушку* currentFrame = SwingFXUtils.toFXImage(mImage, **null**);  
 stream.setImage(currentFrame);  
 }  
 }  
  
 **private void** updateUI(BufferedImage bufferedImage) {  
 *// когда картинки не успевают отображаться - некоторые пропускаем* **synchronized** (mQueue) {  
 **if** (mQueue.size() == MAX\_BUFFER) {  
 *// убираем некоторые не успевающие прорисоваться картинки* mLastFrame = mQueue.poll();  
 }  
 *// добавляем в очередь картинку* mQueue.add(bufferedImage);  
 }  
  
 repaint();  
 }  
  
  
 *// Setters* **private void** setNewGeo(Coordinate coordinate) {  
 **this**.lastCoordinate = coordinate;  
 **if** (latitude == **null**){  
 **return**;  
 }  
 latitude.setText(String.valueOf(coordinate.getLatitude()));  
 longitude.setText(String.valueOf(coordinate.getLongitude()));  
 **if** (geoposition.getOpacity() != 1) {  
 geoposition.setOpacity(1);  
 }  
 }  
  
 **public void** setParent(Main parent) {  
 **this**.parent = parent;  
 }  
  
 **public void** setUserStream(UserStream userStream) {  
 userStream.setDataListener(**this**);  
 userStream.setGeoListener(**this**);  
 **this**.userStream = userStream;  
 }  
  
 *// Listeners* @Override  
 **public void** onDirty(BufferedImage bufferedImage) {  
 *// по готовности - обновляем изображение* updateUI(bufferedImage);  
 }  
  
 @Override  
 **public void** onGeoChange(Coordinate coordinate) {  
 setNewGeo(coordinate);  
 }  
}

* 1. **Класс Utils.java**

**package** ui;  
  
*/\*\*  
 \* Методы - утилиты для обработки изображений, принятия соединений и т.д.  
 \*/***public class** Utils {  
  
 */\*\*  
 \* Конвертер Yuv изображения в Rgb. Метод был найден на просторах StackOverflow.  
 \*  
 \** ***@param yuv*** *Yuv изображение  
 \** ***@param width*** *ширина картинки  
 \** ***@param height*** *высота картинки  
 \** ***@return*** *массив-картинка Rgb  
 \** ***@throws*** *NullPointerException  
 \** ***@throws*** *IllegalArgumentException  
 \*/* **public static int**[] convertYUVtoRGB(**byte**[] yuv, **int** width, **int** height)  
 **throws** NullPointerException, IllegalArgumentException {  
 *// выходной размер* **int** sz = width \* height;  
 *// выходной массив* **int**[] out = **new int**[sz];  
  
 **int** i, j;  
 **int** Y, Cr = 0, Cb = 0;  
 **for** (j = 0; j < height; j++) {  
 **int** pixPtr = j \* width;  
 **final int** jDiv2 = j >> 1;  
 **for** (i = 0; i < width; i++) {  
 Y = yuv[pixPtr];  
 **if** (Y < 0)  
 Y += 255;  
 **if** ((i & 0x1) != 1) {  
 **final int** cOff = sz + jDiv2 \* width + (i >> 1) \* 2;  
 Cb = yuv[cOff];  
 **if** (Cb < 0)  
 Cb += 127;  
 **else** Cb -= 128;  
 Cr = yuv[cOff + 1];  
 **if** (Cr < 0)  
 Cr += 127;  
 **else** Cr -= 128;  
 }  
 **int** R = Y + Cr + (Cr >> 2) + (Cr >> 3) + (Cr >> 5);  
 **if** (R < 0)  
 R = 0;  
 **else if** (R > 255)  
 R = 255;  
 **int** G = Y - (Cb >> 2) + (Cb >> 4) + (Cb >> 5) - (Cr >> 1)  
 + (Cr >> 3) + (Cr >> 4) + (Cr >> 5);  
 **if** (G < 0)  
 G = 0;  
 **else if** (G > 255)  
 G = 255;  
 **int** B = Y + Cb + (Cb >> 1) + (Cb >> 2) + (Cb >> 6);  
 **if** (B < 0)  
 B = 0;  
 **else if** (B > 255)  
 B = 255;  
 out[pixPtr++] = 0xff000000 + (B << 16) + (G << 8) + R;  
 }  
 }  
  
 **return** out;  
 }  
}

* 1. **Класс BufferManager.java**

**package** data;  
  
**import** data.Abstractions.PictureData;  
**import** data.Listeners.DataListener;  
  
**import** javax.imageio.ImageIO;  
**import** java.awt.image.BufferedImage;  
**import** java.io.ByteArrayInputStream;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.util.LinkedList;  
**import** java.util.NoSuchElementException;  
  
*//!!! Для данного случая, используется слово фрейм - ещё не обработанный массив байтов или его часть,  
//!!! равная размеру в байтах картинки полученной с камеры на телефоне  
  
  
/\*\*  
 \* Осуществляет работу с буферами фреймов, кадрирование, перевод из YUV в RGB. Общение с View посредством  
 \* <strong>DataListener</strong>. Используется поочерёдная смена буфферов.  
 \*  
 \** ***@author*** *Aleksandr Smilyanskiy  
 \** ***@version*** *1.0  
 \*/***public class** BufferManager **extends** Thread {  
  
 */\*\*  
 \* Количество буферов для содержания и динамической работы с поступающими фреймами  
 \*/* **private static final int** MAX\_BUFFER\_COUNT = 2;  
 */\*\*  
 \* Буферы записи для записи фреймов  
 \*/* **private** ImageFrameBuffer[] mBufferQueue;  
 */\*\*  
 \* Номер текущего заполняемого буфера  
 \*/* **private int** mFillCount = 0;  
 */\*\*  
 \* Количество оставшейся до записи в текущий буфер информации  
 \*/* **private int** mRemained = 0;  
  
 **private** PictureData pictureData;  
 */\*\*  
 \* "Очередь" фреймов для преобразования в картинки  
 \*/* **private** LinkedList<**byte**[]> mYUVQueue = **new** LinkedList<>();  
 */\*\*  
 \* Слушатель о завершении преобразования одного фрейма  
 \*/* **private** DataListener dataListener;  
  
 **private final** LinkedList<BufferedImage> Queue = **new** LinkedList<>();  
  
 **public** BufferManager() {  
  
 }  
  
 */\*\*\*  
 \* Создаёт "преобразователь" фреймов в картинки  
 \*  
 \** ***@param*** *pictureData информация о картинке  
 \*/* **public** BufferManager(PictureData pictureData) {  
 **this**();  
 setPictureData(pictureData);  
  
 mBufferQueue = **new** ImageFrameBuffer[MAX\_BUFFER\_COUNT];  
 **for** (**int** i = 0; i < MAX\_BUFFER\_COUNT; ++i) {  
 mBufferQueue[i] = **new** ImageFrameBuffer(pictureData.getFrameLength(), pictureData.getWidth(), pictureData.getHeight());  
 }  
 }  
  
 */\*\*  
 \* Общий метод направляющий на раскадрирование по фреймам поток последовательно идущих фреймов  
 \*  
 \** ***@param*** *data байт массив - "поток" фреймов  
 \** ***@param*** *len длина новой информации  
 \*/* **public void** fillBuffer(**byte**[] data, **int** len) {  
 *// не проведена инициализация* **if** (pictureData == **null** || !pictureData.checkCorrect()) {  
 **return**;  
 }  
  
 *// исправим инкремент* mFillCount = mFillCount % MAX\_BUFFER\_COUNT;  
  
 *// Если осталось дозаписать в текущий буффер* **if** (mRemained != 0) {  
 *// Если длина поступвшей информации больше оставшейся до записи* **if** (mRemained < len) {  
 *// пишем сколько осталось в текущий буффер* mBufferQueue[mFillCount].fillBuffer(data, 0, mRemained, mYUVQueue);  
 *// меняем буффер* ++mFillCount;  
 **if** (mFillCount == MAX\_BUFFER\_COUNT)  
 mFillCount = 0;  
 *// пишем оставшееся, но с оффсетом для data=mRemained и в другой буффер* mBufferQueue[mFillCount].fillBuffer(data, mRemained, len - mRemained, mYUVQueue);  
 *// запишем сколько осталось дописать в буффер* mRemained = pictureData.getFrameLength() - len + mRemained;  
 } **else if** (mRemained == len) {  
 *// если длина информации для записи равна длине оставшейся информации для записи  
 // заполняем оставшийся буффер* mBufferQueue[mFillCount].fillBuffer(data, 0, mRemained, mYUVQueue);  
 *// обнуляем кол-во необходимой оставшейся информации* mRemained = 0;  
 *// меняем буффер* ++mFillCount;  
 **if** (mFillCount == MAX\_BUFFER\_COUNT)  
 **mFillCount** = 0;  
 } **else** {  
 *// если len<mRemained, то записываем инфо в буффер и уменьшаем кол-во оставшейся до записи информации* **mBufferQueue**[**mFillCount**].fillBuffer(data, 0, len, **mYUVQueue**);  
 **mRemained** = **mRemained** - len;  
 }  
 } **else** {  
 *// заполнения буфера НА кадрирование* **mBufferQueue**[**mFillCount**].fillBuffer(data, 0, len, **mYUVQueue**);  
  
 *// если длина меньше длины одного фрейма - нехватка информации* **if** (len < **pictureData**.getFrameLength()) {  
 *// тогда осталось до полного фрейма - длина фрейма - текущая длина* **mRemained** = **pictureData**.getFrameLength() - len;  
 } **else** {  
 *// иначе - текущий буффер заполнен, идём в следующий* ++**mFillCount**;  
 *// идём на первый буфер если дошли до последнего* **if** (**mFillCount** == ***MAX\_BUFFER\_COUNT***)  
 **mFillCount** = 0;  
 }  
 }  
 }  
  
 */\*\*\*  
 \* Установка слушателя процесса кадрирования и запуск процесса переведения из кадров в картинки  
 \*  
 \** ***@param listener*** *слушатель процесса кадрирования  
 \*/* **public void** setOnDataListener(DataListener listener) {  
 *// Слушатель изменения информации* **dataListener** = listener;  
 *// при установке слушателя тут же начинается превращение из кадров в картинки* **if** (listener != **null**)  
 start();  
 }  
  
 **public void** setPictureData(PictureData pictureData) {  
 **this**.**pictureData** = pictureData;  
 }  
  
 **public** DataListener getDataListener() {  
 **return dataListener**;  
 }  
  
 **public void** completeImageReceived(**byte**[] imageArray) {  
 **try** {  
 BufferedImage bufferedImage;  
 InputStream in = **new** ByteArrayInputStream(imageArray);  
  
 bufferedImage = ImageIO.read(in);  
  
 **synchronized** (Queue) {  
 **if** (Queue.size() > 15) {  
 Queue.poll();  
 }  
 Queue.add(bufferedImage);  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*\*  
 \* Закрытие процесса превращения кадров в картинки  
 \*/* **public void** close() {  
 *// сообщить о желании закрыть поток* interrupt();  
 **try** {  
 *// дождаться прогрузки кадра* join();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*\*  
 \* Процесс переведния кадров в картинки BufferedImage(RGB)  
 \*/* @Override  
 **public void** run() {  
 **super**.run();  
 *// Работает пока не будет прерван специально* **while** (!Thread.currentThread().isInterrupted()) {  
  
 **synchronized** (Queue) {  
 BufferedImage image = **null**;  
 **try** {  
 image = Queue.poll();  
 } **catch** (NoSuchElementException ignored) {  
  
 }  
 **if** (image == **null**) {  
 **continue**;  
 } **else** {  
 **if** (dataListener != **null**) {  
 dataListener.onDirty(image);  
  
 }  
 }  
 }  
  
*// // буффер  
// byte[] data = null;  
//  
// // синхронизируем по потоку кадров  
// synchronized (mYUVQueue) {  
// // Получение последней картинки из "потока" с телефона  
// data = mYUVQueue.poll();  
//  
// // Если последний кадр существует  
// if (data != null) {  
// // получение дампа времени  
// long t = System.currentTimeMillis();  
//  
// BufferedImage bufferedImage = null;  
// // конвертирование потока(массива) в RGB из YUV  
//// int[] rgbArray = Utils.convertYUVtoRGB(data, pictureData.getWidth(), pictureData.getHeight());  
//// // создание картинки  
//// bufferedImage = new BufferedImage(pictureData.getWidth(), pictureData.getHeight(), BufferedImage.TYPE\_USHORT\_565\_RGB);  
//// // портирование массива в картинку  
//// bufferedImage.setRGB(0, 0, pictureData.getWidth(), pictureData.getHeight(), rgbArray, 0, pictureData.getWidth());  
//  
// InputStream in = new ByteArrayInputStream(data);  
// try {  
// bufferedImage = ImageIO.read(in);  
// } catch (IOException e) {  
// e.printStackTrace();  
// }  
//  
// if (dataListener != null)  
// // сообщим о создании очередной картинки  
// dataListener.onDirty(bufferedImage);  
// // время занятое на трансформацию очередного кадра  
// System.out.println("time cost = " + (System.currentTimeMillis() - t));  
// }  
//  
// }* }  
 }  
}

* 1. **Класс ImageFrameBuffer.java**

**package** data;  
  
**import** java.io.ByteArrayOutputStream;  
**import** java.util.LinkedList;  
  
*/\*\*  
 \* Буффер для одного фрейма. Ждёт пока не придёт достаточно информации и начинает запись в "очередь" на  
 \* перекодирование в RGB.  
 \*  
 \** ***@author*** *Aleksandr Smilyanskiy  
 \** ***@version*** *1.0  
 \*/***public class** ImageFrameBuffer {  
  
 */\*\*  
 \* Длина одного фрейма  
 \*/* **private final int mFrameLength**;  
 */\*\*  
 \* Количество текущей заполненной информации  
 \*/* **private int mTotalLength** = 0;  
 */\*\*  
 \* Поток для записи-вывода последовательно идущих фреймов  
 \*/* **private** ByteArrayOutputStream **mByteArrayOutputStream**;  
  
 */\*\*\*  
 \* Создание буфера для фрейма изображения  
 \*  
 \** ***@param frameLength*** *длина одного фрейма  
 \** ***@param width*** *ширина картинки  
 \** ***@param height*** *высота картинки  
 \*/* **public** ImageFrameBuffer(**int** frameLength, **int** width, **int** height) {  
 *// Создаём поток для вывода байтов фрейма* **mByteArrayOutputStream** = **new** ByteArrayOutputStream();  
 *// запоминание длины одного фрейма* **mFrameLength** = frameLength;  
 }  
  
 */\*\*\*  
 \* Запись из потока полследовательно идущих фреймов в "очередь" фреймов-кадров (по сути некоторого рода кадрирование)  
 \*  
 \** ***@param data*** *буфер  
 \** ***@param off*** *смещение в буфере  
 \** ***@param len*** *длина новой информации  
 \** ***@param YUVQueue*** *"очередь" кадров  
 \** ***@return*** *код выподнения (стандартный)  
 \*/* **public int** fillBuffer(**byte**[] data, **int** off, **int** len, LinkedList<**byte**[]> YUVQueue) {  
 *// добавление длины нового массива информации* **mTotalLength** += len;  
 *// запись информации в буффер ожидания записи* **mByteArrayOutputStream**.write(data, off, len);  
  
 *// если фрейм записан в буфер полностью* **if** (**mTotalLength** == **mFrameLength**) {  
  
 *// запись в "очередь" фреймов* **synchronized** (YUVQueue) {  
 *// запись в "очередь" фреймов* YUVQueue.add(**mByteArrayOutputStream**.toByteArray());  
 *// обнуление буффера* **mByteArrayOutputStream**.reset();  
 }  
  
 *// обнуление длины буфера* **mTotalLength** = 0;  
 *// выводим информацию о получении и записи в очередь файла* System.***out***.println(**"received file"**);  
 }  
  
 **return** 0;  
 }  
}

* 1. **Класс Coordinate.java**

**package** data.Abstractions;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 30.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public class** Coordinate {  
 *// Широта и долгота* **double latitude**, **longitude**;  
  
 Coordinate() {  
 }  
  
 **public** Coordinate(**double** latitude, **double** longitude) {  
 **this**();  
 setCoordinates(latitude, longitude);  
 }  
  
 **public** Coordinate(**double**[] data){  
 **this**();  
 setCoordinates(data[0],data[1]);  
 }  
  
  
 *// Сеттеры* **public void** setLatitude(**double** latitude) {  
 **this**.**latitude** = latitude;  
 }  
  
 **public void** setLongitude(**double** longitude) {  
 **this**.**longitude** = longitude;  
 }  
  
 **public void** setCoordinates(**double** latitude, **double** longitude) {  
 **this**.**latitude** = latitude;  
 **this**.**longitude** = longitude;  
 }  
  
 **public double** getLatitude() {  
 **return latitude**;  
 }  
  
 **public double** getLongitude() {  
 **return longitude**;  
 }  
  
 @Override  
 **public boolean** equals(Object obj) {  
 **if** (obj.getClass() != **this**.getClass()){  
 **return false**;  
 }  
 **return this**.**longitude** == ((Coordinate) obj).**longitude** && **this**.**latitude** == ((Coordinate) obj).**latitude**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Latitude: "** + **latitude** + **"\tLongitude: "** + **longitude**;  
 }  
}

* 1. **Класс PictureData.java**

**package** data.Abstractions;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 30.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public class** PictureData {  
 *// ширина кадра, ширина и высота картинки* **private int frameLength**, **width**, **height**;  
  
 **public** PictureData() {  
 }  
  
 **public** PictureData(**int** frameLength, **int** width, **int** height) {  
 **this**();  
 setPictureFormat(frameLength,width,height);  
 }  
  
 **public boolean** checkCorrect(){  
 **return frameLength** > 0 && **width** > 0 && **height** > 0;  
 }  
  
  
 *// Сеттеры* **public void** setPictureFormat(**int** length, **int** width, **int** height){  
 setFrameLength(length);  
 setWidth(width);  
 setHeight(height);  
 }  
  
 **public void** setWidth(**int** width) {  
 **this**.**width** = width;  
 }  
  
 **public void** setFrameLength(**int** frameLength) {  
 **this**.**frameLength** = frameLength;  
 }  
  
 **public void** setHeight(**int** height) {  
 **this**.**height** = height;  
 }  
  
 *// Геттеры* **public int** getFrameLength() {  
 **return frameLength**;  
 }  
  
 **public int** getWidth() {  
 **return width**;  
 }  
  
 **public int** getHeight() {  
 **return height**;  
 }  
}

* 1. **Класс StreamData.java**

**package** data.Abstractions;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 30.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/  
  
/\*\*  
 \* Представляет всю необходимую информацию о стриме  
 \*/***public class** StreamData {  
 *// параметры изображения* **private** PictureData **pictureData**;  
 *// координаты* **private** Coordinate **coordinate**;  
 *// id стрима* **private** String **id**;  
 *// имя стрима* **private** String **name**;  
  
 **public** StreamData() {  
  
 }  
  
 **public** StreamData(PictureData pictureData) {  
 **this**();  
 setPictureData(pictureData);  
 }  
  
 **public** StreamData(String id, String name) {  
 **this**();  
 setId(id);  
 setName(name);  
 }  
  
 **public** StreamData(String id, String name, PictureData pictureData) {  
 **this**();  
 setId(id);  
 setName(name);  
 setPictureData(pictureData);  
 }  
  
 *// Сеттеры* **public void** setCoordinate(Coordinate coordinate) {  
 **this**.**coordinate** = coordinate;  
 }  
  
 **public void** setPictureData(PictureData pictureData) {  
 **this**.**pictureData** = pictureData;  
 }  
  
 **public void** setId(String id) {  
 **if** (**this**.**id** != **null**)  
 **throw new** IllegalArgumentException(**"Id already set."**);  
 **this**.**id** = id;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 *// Геттеры* **public** String getId() {  
 **return id**;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public** Coordinate getCoordinate() {  
 **return coordinate**;  
 }  
  
 **public** PictureData getPictureData() {  
 **return pictureData**;  
 }  
}

* 1. **Класс DataListener.java**

**package** data.Listeners;  
  
**import** java.awt.image.BufferedImage;  
  
*/\*\*  
 \* Слушатель заканчивания процесса перекодирования из YUV в RGB.  
 \*  
 \** ***@author*** *Aleksandr Smilyanskiy  
 \** ***@version*** *1.0  
 \*/***public interface** DataListener {  
 */\*\*  
 \* Информирует о перекодировании очередного фрейма в картинку-RGB BufferedImage  
 \*  
 \** ***@param bufferedImage*** *Получившаяся картинка  
 \*/* **void** onDirty(BufferedImage bufferedImage);  
}

* 1. **Класс ErrorListener.java**

**package** data.Listeners;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 06.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/  
  
/\*\*  
 \* Слушатель ошибок  
 \*/***public interface** ErrorListener {  
 **void** onError(String message);  
}

* 1. **Класс GeoListener.java**

**package** data.Listeners;  
  
**import** data.Abstractions.Coordinate;  
  
**import** java.awt.image.BufferedImage;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 05.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/  
  
/\*\*  
 \* Слушатель изменения геопозиции  
 \*/***public interface** GeoListener {  
 */\*\*  
 \* Информирует о изменении геопозиции стрима  
 \*  
 \** ***@param coordinate*** *новая координата  
 \*/* **void** onGeoChange(Coordinate coordinate);  
}

* 1. **Класс PoolListener.java**

**package** data.Listeners;  
  
**import** io.UserStream;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 06.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/  
  
/\*\*  
 \* Слушатель событий пула стримов  
 \*/***public interface** PoolListener {  
 **void** onStreamAdded(UserStream userStream);  
 **void** onStreamDisconnect(UserStream userStream);  
}

* 1. **Класс ServerListener.java**

**package** data.Listeners;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 30.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public interface** ServerListener {  
 **void** onServerThinking();  
  
 **void** onServerClosed();  
  
 **void** onServerOpen();  
}

* 1. **Класс SimpleStreamListener.java**

**package** data.Listeners;  
  
**import** data.Abstractions.StreamData;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 07.05.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public interface** SimpleStreamListener {  
 **void** onStreamShutdown(StreamData streamData);  
}

* 1. **Класс StreamListener.java**

**package** data.Listeners;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 06.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/  
  
/\*\*  
 \* Слушатель собатий стримов  
 \*/***public interface** StreamListener {  
 **void** onGeopositionChanged(**int** latitude, **int** longitude);  
  
 **void** onStreamResized(**int** width, **int** height);  
  
 **void** onTranslationStatusChanged(**int** status); *// -1 = offline, 0 = paused, 1 = online* **void** onHeartbeatReceived();  
}

* 1. **Класс HandleIncomingConnection.java**

**package** io;  
  
**import** com.google.gson.\*;  
**import** data.Abstractions.PictureData;  
**import** data.Abstractions.StreamData;  
**import** data.BufferManager;  
**import** data.Listeners.ErrorListener;  
  
**import** java.io.\*;  
**import** java.net.Socket;  
**import** java.util.List;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 06.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public class** HandleIncomingConnection **extends** Thread {  
 **private** Socket **socket**;  
 **private** Server **server**;  
 **private** JsonObject **object**;  
 **private** StreamPool **pool**;  
 **private** UserStream **user**;  
 **private** BufferedReader **inputStream**;  
 **private** BufferedWriter **outputStream**;  
 **private** ErrorListener **errorListener**;  
  
 **public** HandleIncomingConnection(Server server, StreamPool pool, Socket incoming) {  
 **socket** = incoming;  
 **errorListener** = server.getErrorListener();  
 **this**.**server** = server;  
 **this**.**pool** = pool;  
 }  
  
 @Override  
 **public void** run() {  
 **super**.run();  
 **try** {  
 *// не должен меняться до конца этой операции* **synchronized** (**socket**) {  
 **outputStream** = **new** BufferedWriter(**new** OutputStreamWriter(**socket**.getOutputStream()));  
 **inputStream** = **new** BufferedReader(**new** InputStreamReader(**socket**.getInputStream()));  
  
  
 *// проверка на json запрос* **boolean** isJson = isIncomingJson();  
  
  
 *// анализируем тип запроса если json* **if** (isJson) {  
 **if** (analyzeIncomingJson()) {  
 *// запрос на регистрацию стрима* **synchronized** (pool) {  
 pool.addUserStream(user);  
 **return**;  
 }  
 } **else** {  
 *// запрос на получение списка стримов* sendStreams(pool.getAllStreams());  
 }  
 }  
  
 *// закрываем соединения если запрос был краткосрочный или неправильный* **try** {  
 inputStream.close();  
 outputStream.close();  
 socket.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 errorListener.onError(**"Error in disconnecting wrong connection."**);  
 }  
 }  
 } **catch** (IOException e) {  
 errorListener.onError(**"Incoming connection error: "** + e.getMessage());  
 e.printStackTrace();  
 }  
 }  
  
 **private boolean** isIncomingJson() {  
 **try** {  
 *// чтение Json* String message = inputStream.readLine();  
 JsonParser parser = **new** JsonParser();  
 JsonElement element = **null**;  
 **try** {  
 **if** (message == **null**){  
 **throw new** JsonParseException(**"Message is null."**);  
 }  
 element = parser.parse(message);  
 } **catch** (JsonParseException e) {  
 errorListener.onError(**"Incoming connection refused: not a json."**);  
 **return false**;  
 }  
 *// нет входного сообщения -> это или неправильное соединение* **if** (element == **null**) {  
 errorListener.onError(**"Incoming connection refused: no message."**);  
 **return false**;  
 }  
  
 object = element.getAsJsonObject();  
 } **catch** (IOException e) {  
 errorListener.onError(**"Incoming connection error: "** + e.getMessage());  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **private boolean** analyzeIncomingJson() {  
 *// класс с информацией о изображении* PictureData pictureData = **new** PictureData();  
  
 *// Если чего-то нет это неправильный запрос или запрос на получение списка стримов* **try** {  
 pictureData.setFrameLength(object.get(**"length"**).getAsInt());  
 pictureData.setWidth(object.get(**"width"**).getAsInt());  
 pictureData.setHeight(object.get(**"height"**).getAsInt());  
 } **catch** (NullPointerException e) {  
 **return false**;  
 }  
 **if** (!pictureData.checkCorrect()) {  
 **return false**;  
 }  
  
 *// Значит это не запрос на регистрацию стрима* user = **new** UserStream(server, socket);  
 user.setPictureData(pictureData);  
 user.setBufferManager(**new** BufferManager(pictureData));  
 **return true**;  
 }  
  
 **private void** sendStreams(List<StreamData> streams) {  
 *// создание сообщения* JsonObject jsonObject = **new** JsonObject();  
 jsonObject.addProperty(**"info"**, **"streamData"**);  
 JsonArray streamList = **new** JsonArray();  
 **for** (StreamData userData : streams) {  
 JsonObject currentUser = **new** JsonObject();  
 currentUser.addProperty(**"name"**, userData.getName());  
 currentUser.addProperty(**"id"**, userData.getId());  
 streamList.add(currentUser);  
 }  
 jsonObject.add(**"streams"**, streamList);  
 *// попытка отправки* **try** {  
 outputStream.write(jsonObject.toString() + **"\n"**);  
 outputStream.flush();  
 } **catch** (IOException e) {  
 errorListener.onError(**"Warning on incoming connection: Can not send stream list, probably user disconnected."**);  
 }  
 }  
}

* 1. **Класс Server.java**

**package** io;  
  
**import** data.Abstractions.StreamData;  
**import** data.BufferManager;  
**import** data.Listeners.\*;  
**import** javafx.application.Platform;  
**import** ui.main.Main;  
  
**import** java.io.IOException;  
**import** java.net.ServerSocket;  
**import** java.net.Socket;  
**import** java.util.List;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 06.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public class** Server **extends** Thread{  
 *// parent* **private** Main **main**;  
  
 *// Компоненты сервера* **private** ServerSocket **serverSocket**;  
 **private** StreamPool **streamPool**;  
 *// Listeners  
// private DataListener dataListener; // converter* **private** PoolListener **poolListener**; *// pool* **private** ErrorListener **errorListener**; *// errors* **private** ServerListener **serverListener**; *// current server  
 // Параметрыы сервера* **private int port** = 8585;  
 **private static final int *MAX\_USERS*** = 10;  
  
 **public** Server(Main main){  
 **super**();  
 **this**.**main** = main;  
 **streamPool** = **new** StreamPool(**this**.**poolListener**);  
 **streamPool**.setErrorListener(**errorListener**);  
 }  
*//  
// public Server(PoolListener poolListener) {  
// this();  
// this.poolListener = poolListener;  
// }  
//  
// public Server(PoolListener poolListener, int port) {  
// this(poolListener);  
// this.port = port;  
// }* @Override  
 **public void** run() {  
 **super**.run();  
 Platform.*runLater*(() -> **main**.onServerThinking());  
 **try** {  
 **serverSocket** = **new** ServerSocket(**port**);  
 Platform.*runLater*(() -> **main**.onServerOpen());  
 **while** (!isInterrupted()) {  
 Socket inputConnection = **serverSocket**.accept();  
  
 *// принимаем и обрабатываем входящее соединение по одному из шаблонов:  
 // 1 - регистрируем стрим  
 // 2 - отсылаем список стримов, закрываем соединение  
 // 3 - ошибочный запрос, закрываем соединение* HandleIncomingConnection handleIncomingConnection = **new** HandleIncomingConnection(**this**, **streamPool**, inputConnection);  
 handleIncomingConnection.start();  
 }  
 } **catch** (IOException e) {  
 errorListener.onError(**"Server going offline."**);  
 }  
 }  
  
 **public** ErrorListener getErrorListener() {  
 **return** errorListener;  
 }  
  
 *// Server options* **public** List<StreamData> getStreams(){  
 **return** streamPool.getAllStreams();  
 }  
 **public void** deleteStream(String id) {  
 streamPool.removeUserStream(id);  
 }  
 **public** UserStream openStream(String id){  
 **if** (!streamPool.heartbeatStream(id)){  
 errorListener.onError(**"Warning in opening stream: user is not active anymore."**);  
 **return null**;  
 }  
 UserStream user = streamPool.getUserStream(id);  
 **try**{  
 user.requestStart();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 errorListener.onError(**"Error in opening stream: user is not active anymore."**);  
 }  
 **return** user;  
 }  
 **public void** closeStream(String id){  
 streamPool.removeUserStream(id);  
 }  
 **public void** heartbeatStreams(){  
 streamPool.heartbeatStreams();  
 }  
 **public void** closeServer(){  
 *// Здесь операции по завершению работы сервера* **try** {  
 serverSocket.close();  
 streamPool.closeAllConnections();  
 } **catch** (IOException e) {  
 errorListener.onError(**"Error on running server: "** + e.getMessage());  
 e.printStackTrace();  
 }  
 Platform.runLater(() -> main.onServerClosed());  
 interrupt();  
 }  
  
  
 *// Listener setters* **public void** setPoolListener(PoolListener poolListener) {  
 **this**.poolListener = poolListener;  
 **if** (streamPool != **null**){  
 streamPool.setPoolListener(poolListener);  
 }  
 }  
 **public void** setErrorListener(ErrorListener errorListener) {  
 **this**.errorListener = errorListener;  
 **if** (streamPool != **null**){  
 streamPool.setErrorListener(errorListener);  
 }  
 }  
 **public void** setServerListener(ServerListener serverListener) {  
 **this**.serverListener = serverListener;  
 }  
}

* 1. **Класс StreamPool.java**

**package** io;  
  
**import** com.google.gson.JsonObject;  
**import** data.Abstractions.StreamData;  
**import** data.Listeners.ErrorListener;  
**import** data.Listeners.PoolListener;  
**import** data.Listeners.SimpleStreamListener;  
**import** data.Listeners.StreamListener;  
  
**import** java.io.IOException;  
**import** java.util.\*;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 05.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public class** StreamPool **implements** SimpleStreamListener{  
 **private final int MAX\_USERS** = 10;  
 **private** HashMap<String, UserStream> **pool**;  
 **private** PoolListener **poolListener**;  
 **private** StreamListener **streamListener**;  
 **private** ErrorListener **errorListener**;  
  
 *// Constructors* **public** StreamPool() {  
 **pool** = **new** HashMap<>();  
 }  
  
 **public** StreamPool(PoolListener poolListener) {  
 **this**();  
 setPoolListener(poolListener);  
 }  
  
  
 *// Pool options* **public boolean** heartbeatStream(String id) {  
 UserStream userStream = **pool**.get(id);  
 **try** {  
 **if** (userStream != **null**) {  
 **if** (userStream.heartbeatStream()) {  
 **return true**;  
 }  
 userStream.interrupt();  
 **pool**.remove(userStream.getStreamData().getId());  
 **if** (**poolListener** != **null**)  
 **poolListener**.onStreamDisconnect(userStream);  
 }  
 } **catch** (IOException e) {  
 userStream.interrupt();  
 **pool**.remove(userStream.getStreamData().getId());  
 e.printStackTrace();  
 }  
 **return false**;  
 }  
  
 **public void** heartbeatStreams() {  
 **for** (UserStream userStream : pool.values()) {  
 **try** {  
 userStream.heartbeatStream();  
 } **catch** (IOException e) {  
 userStream.interrupt();  
 pool.remove(userStream.getStreamData().getId());  
 }  
 }  
 }  
  
 **public** List<StreamData> getAllStreams() {  
 *// !!! при большом колв-е юзеров уменьшить капасити изначальную !!!* List<StreamData> streamDatas = **new** ArrayList<>(MAX\_USERS);  
 **for** (UserStream userStream : pool.values()) {  
 streamDatas.add(userStream.getStreamData());  
 }  
 **return** streamDatas;  
 }  
  
 **public void** closeAllConnections() {  
 **for** (Map.Entry<String, UserStream> entry : pool.entrySet()) {  
 UserStream stream = entry.getValue();  
 *// дисконектит каждого юзера* stream.interrupt();  
 }  
 }  
  
 *// Pool users options* **public int** addUserStream(UserStream userStream) {  
 *// проверка на разрешение добавления юзера* **if** (pool.size() + 1 > MAX\_USERS) {  
 **return** -1;  
 }  
 *// получение id и запись пользователя* String uuid = UUID.randomUUID().toString();  
 userStream.getStreamData().setId(uuid);  
 userStream.start();  
 userStream.setErrorListener(errorListener);  
 userStream.setSimpleStreamListener(**this**);  
 pool.put(uuid, userStream);  
 *// сообщение слушателю о действии* **if** (poolListener != **null**)  
 poolListener.onStreamAdded(userStream);  
 **return** 0;  
 }  
  
 **public int** removeUserStream(String id) {  
 *// проверка на существование стрима* **if** (pool.get(id) == **null**) {  
 **return** -1;  
 }  
 UserStream deletedUserStream = pool.get(id);  
 *// удаление стрима* pool.remove(id);  
 deletedUserStream.closeStream();  
*// boolean is = deletedUserStream.isInterrupted();  
 // сообщение слушателю о действии* **if** (poolListener != **null**)  
 poolListener.onStreamDisconnect(deletedUserStream);  
 **return** 0;  
 }  
  
 **public** UserStream getUserStream(String id) {  
 **return** pool.get(id);  
 }  
  
 *// Pool Listener* **public** PoolListener getPoolListener() {  
 **return** poolListener;  
 }  
  
 **public void** setPoolListener(PoolListener poolListener) {  
 **this**.poolListener = poolListener;  
 }  
  
 **public void** setStreamListener(StreamListener streamListener) {  
 **this**.streamListener = streamListener;  
 }  
  
 **public void** setErrorListener(ErrorListener errorListener) {  
 **this**.errorListener = errorListener;  
 **for** (UserStream userStream: pool.values()){  
 userStream.setErrorListener(errorListener);  
 }  
 }  
  
 @Override  
 **public void** onStreamShutdown(StreamData streamData) {  
 removeUserStream(streamData.getId());  
 }  
}

* 1. **Класс UserStream.java**

**package** io;  
  
**import** com.google.gson.Gson;  
**import** com.google.gson.JsonObject;  
**import** com.google.gson.stream.JsonReader;  
**import** data.Abstractions.Coordinate;  
**import** data.Abstractions.PictureData;  
**import** data.Abstractions.StreamData;  
**import** data.BufferManager;  
**import** data.Listeners.\*;  
  
**import** java.io.\*;  
**import** java.net.Socket;  
**import** java.util.Map;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 05.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***public class** UserStream **extends** Thread {  
 *// Main objects* **private** Socket **socket**;  
 **private** StreamData **streamData**;  
 **private final** Server **server**;  
 **private** BufferManager **bufferManager**;  
  
 *// Streams* **private** BufferedReader **inputStream**;  
 **private** BufferedInputStream **bufferedInputStream**;  
 **private** JsonReader **jsonReader**;  
 **private** BufferedWriter **outputStream**;  
  
 *// Listeners* **private** ErrorListener **errorListener**;  
 **private** StreamListener **streamListener**;  
 *// private DataListener dataListener;* **private** GeoListener **geoListener**;  
 **private** SimpleStreamListener **simpleStreamListener**;  
  
 *// Properties* **private int streamStatus** = -1; *// -1 = offline, 0 = pending or waiting, 1 = online* **private int previousStatus** = -1;  
 **private final** Object **statusWaiter** = **new** Object();  
 **private final** Object **heartbeatWaiter** = **new** Object();  
 **private static final** String ***STOP\_COMMAND*** = **"STOP"**;  
 **private static final int *AWAITING\_TIME*** = 10000;  
  
 *// For GSON Parsing* **private class** DataResponse {  
 **public** Map<String, Data> **descriptor**;  
 }  
  
 **private class** Data {  
 **private byte**[] **data**;  
 **private double**[] **geo**;  
 }  
  
  
 **public** UserStream(Server server, Socket socket) {  
 **this**.**socket** = socket;  
 **this**.**server** = server;  
 **try** {  
 **this**.**inputStream** = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream()));  
 **this**.**bufferedInputStream** = **new** BufferedInputStream(socket.getInputStream());  
 **this**.**jsonReader** = **new** JsonReader(**inputStream**);  
 **this**.**jsonReader**.setLenient(**true**);  
 **this**.**outputStream** = **new** BufferedWriter(**new** OutputStreamWriter(socket.getOutputStream()));  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **streamData** = **new** StreamData();  
 }  
  
  
 @Override  
 **public void** run() {  
 **super**.run();  
 Gson gson = **new** Gson();  
 setStreamStatus(0);  
 **while** (!isInterrupted()) {  
 **try** {  
 **switch** (**streamStatus**) {  
 *// передача картинки осуществляется* **case** 1: {  
 takeImageBuff(gson);  
 **break**;  
 }  
  
 *// ожидание команд* **case** 0: {  
 *// функция пинга со временем по окончании AWAITING\_TIME* **synchronized** (**statusWaiter**) {  
 **try** {  
 **statusWaiter**.wait(***AWAITING\_TIME***);  
 } **catch** (InterruptedException e) {  
 **if** (**errorListener** != **null**) {  
 **errorListener**.onError(**"Waiting for status interrupted."**);  
 }  
 e.printStackTrace();  
 }  
  
 *// если изменился* **if** (**streamStatus** != 0) {  
 **break**;  
 }  
  
 *// отсылка пинга* sendHeartbeat();  
 *// если пинг не прошёл* **if** (!waitForHeartbeat()) {  
 setStreamStatus(-1);  
 **synchronized** (**heartbeatWaiter**) {  
 **heartbeatWaiter**.notify();  
 }  
 **break**;  
 }  
 }  
 **break**;  
 }  
  
 *// отключение* **case** -1:  
 interrupt();  
 }  
  
 } **catch** (IOException e) {  
 *//* ***TODO: 07.05.2016 Here we need to delete the stream* server**.deleteStream(getStreamData().getId());  
 **if** (**streamStatus** != -1) {  
 **if** (**errorListener** != **null**)  
 **errorListener**.onError(**"Error on client <"** + getStreamData().getId() + **">:"** + e.getMessage());  
 e.printStackTrace();  
 }  
 **break**;  
 } **catch** (Exception ignored) {  
 **server**.deleteStream(getStreamData().getId());  
 }  
 }  
 closeStream();  
 }  
  
 *// Запросы* **public void** requestWait() **throws** IOException {  
 **if** (**outputStream** == **null** || **streamStatus** == -1 || **streamStatus** == 0) {  
 **return**;  
 }  
 setStreamStatus(0);  
 *// создаём Json объект для отправки обратно информации о необходимости подождать* JsonObject jsonObject = **new** JsonObject();  
 jsonObject.addProperty(**"state"**, **"wait"**);  
 sendJsonObject(jsonObject);  
 *// ждём до конца потока вещания  
// waitForEnd();* }  
  
 **public void** requestStart() **throws** IOException {  
 **if** (**outputStream** == **null** || **streamStatus** == -1 || **streamStatus** == 1) {  
 **return**;  
 }  
 *// создаём Json объект для отправки обратно информации о необходимости подождать* JsonObject jsonObject = **new** JsonObject();  
 jsonObject.addProperty(**"state"**, **"start"**);  
 sendJsonObject(jsonObject);  
 setStreamStatus(1);  
 }  
  
 **public boolean** heartbeatStream() **throws** IOException {  
 *// проверяет доступность клиента  
 // если сейчас идёт вещание - значит всё в порядке* **if** (**streamStatus** == 1) {  
 **return true**;  
 }  
 *// если вещание закончено* **if** (**streamStatus** == -1) {  
 **return false**;  
 }  
  
 *// иначе - ждём ответа* **synchronized** (**heartbeatWaiter**) {  
 **try** {  
 **heartbeatWaiter**.wait();  
 **if** (**streamStatus** == -1) {  
 **return false**;  
 } **else** {  
 **return true**;  
 }  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
 }  
  
 **private void** sendHeartbeat() **throws** IOException {  
 JsonObject jsonObject = **new** JsonObject();  
 jsonObject.addProperty(**"heartbeat"**, **"request"**);  
 sendJsonObject(jsonObject);  
 }  
  
 **private boolean** waitForHeartbeat() **throws** IOException {  
 **synchronized** (**jsonReader**) {  
 *// чтение строки* **try** {  
 **jsonReader**.beginObject();  
 String name = **jsonReader**.nextName();  
 *// если не ответ* **if** (!**"heartbeat"**.equals(name)) {  
 **jsonReader**.endObject();  
 **return false**;  
 } **else** {  
 *// если правильный ответ* **if** (**"answer"**.equals(**jsonReader**.nextString())) {  
 **jsonReader**.endObject();  
 **synchronized** (**heartbeatWaiter**) {  
 **heartbeatWaiter**.notifyAll();  
 }  
 **return true**;  
 }  
 *// если неправильный ответ* **else** {  
 **jsonReader**.endObject();  
 **return false**;  
 }  
 }  
 } **catch** (Exception e) {  
 **return false**;  
 }  
 }  
 }  
  
 **public void** closeStream() {  
 **try** {  
 **socket**.getOutputStream().close();  
 **try** {  
 **socket**.getInputStream().close();  
 } **catch** (IOException ignored) {  
 }  
 **socket**.close();  
 } **catch** (IOException ignored) {  
 }  
  
 **if** (**streamStatus** != -1) {  
 setStreamStatus(-1);  
 }  
 }  
  
  
 *// Ответы* **public void** sendInfo(String id) **throws** IOException {  
 *//* ***TODO: 07.05.2016 Test this*** JsonObject jsonObject = **new** JsonObject();  
 jsonObject.addProperty(**"id"**, id);  
 sendJsonObject(jsonObject);  
 }  
  
  
 *// Options* **private void** takeImageBuff(Gson gson) **throws** IOException {  
 **if** (**bufferManager** == **null**) {  
 **errorListener**.onError(**"Error, client <"** + getStreamData().getId() + **"> buffer not set."**);  
 setStreamStatus(0);  
 }  
  
 **jsonReader**.beginObject();  
 **byte**[] data = **null**;  
 **double**[] geo = **null**;  
 **while** (**jsonReader**.hasNext()) {  
 **switch** (**jsonReader**.nextName()) {  
 **case "data"**: {  
 data = getByteArray(**jsonReader**.nextString());  
 **break**;  
 }  
 **case "geo"**: {  
 geo = getDoubleArray(**jsonReader**.nextString());  
 **break**;  
 }  
 }  
 }  
 **jsonReader**.endObject();  
 Coordinate newGeo = **new** Coordinate(geo);  
 **if** (**geoListener** != **null**)  
 **geoListener**.onGeoChange(**new** Coordinate(geo));  
 **if** (**bufferManager** != **null**)  
 **bufferManager**.completeImageReceived(data);  
*// bufferManager.fillBuffer(data, data.length);  
  
  
  
// jsonReader.nextName();  
// DataResponse dataResponse;  
// synchronized (jsonReader) {  
// dataResponse = gson.fromJson(jsonReader, DataResponse.class);  
// }  
// byte[] data = dataResponse.descriptor.get("data").data;  
// if (dataResponse.descriptor.containsKey("geo")) {  
// geoListener.onGeoChange(new Coordinate(dataResponse.descriptor.get("geo").geo));  
// }  
// bufferManager.fillBuffer(data, data.length);* }  
  
 **private byte**[] getByteArray(String message) {  
 String[] bytes = message.split(**","**);  
 bytes[0] = bytes[0].replaceFirst(**"\\["**, **""**);  
 bytes[bytes.**length** - 1] = bytes[bytes.**length** - 1].replaceFirst(**"]"**, **""**);  
 **byte**[] actual = **new byte**[bytes.**length**];  
 **for** (**int** i = 0; i < bytes.**length**; i++) {  
 actual[i] = Byte.*parseByte*(bytes[i].replaceFirst(**" "**, **""**));  
 }  
 **return** actual;  
 }  
  
 **private double**[] getDoubleArray(String message) {  
 String[] doubles = message.split(**","**);  
 doubles[0] = doubles[0].replaceFirst(**"\\["**, **""**);  
 doubles[doubles.**length** - 1] = doubles[doubles.**length** - 1].replaceFirst(**"]"**, **""**);  
 doubles[doubles.**length** - 1] = doubles[doubles.**length** - 1].replaceFirst(**" "**, **""**);  
 **double**[] actual = **new double**[doubles.**length**];  
 **for** (**int** i = 0; i < doubles.**length**; i++) {  
 actual[i] = Double.*parseDouble*(doubles[i]);  
 }  
 **return** actual;  
 }  
  
 **private boolean** waitForEnd() **throws** IOException {  
 *// ждёт конца передачи изображений* **synchronized** (**bufferedInputStream**) {  
 **byte**[] stopCommand = ***STOP\_COMMAND***.getBytes();  
 **byte**[] onebyteArray = **new byte**[1];  
 **int** i = 0;  
 **while** (**bufferedInputStream**.read(onebyteArray) != -1) {  
 **if** (onebyteArray[0] == stopCommand[i]) {  
 i += 1;  
 **if** (i == ***STOP\_COMMAND***.length()) {  
 **return true**;  
 }  
 } **else** {  
 i = 0;  
 }  
 }  
 **return false**;  
 }  
 }  
  
  
 *// Helpers* **private void** sendJsonObject(JsonObject jsonObject) **throws** IOException {  
 **synchronized** (**outputStream**) {  
 **outputStream**.write(jsonObject.toString());  
 **outputStream**.flush();  
 }  
 }  
  
  
 *//Setters* **public void** setPictureData(PictureData pictureData) {  
 **streamData**.setPictureData(pictureData);  
 }  
  
 **private void** setStreamStatus(**int** streamStatus) {  
 **previousStatus** = **this**.**streamStatus**;  
 **this**.**streamStatus** = streamStatus;  
 **if** (**previousStatus** == 0 && streamStatus == 1) {  
 **try** {  
 **bufferedInputStream**.skip(**bufferedInputStream**.available());  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 **if** (**streamListener** != **null**) {  
 **streamListener**.onTranslationStatusChanged(streamStatus);  
 }  
 **if** (streamStatus == -1 && **simpleStreamListener** != **null**) {  
 **simpleStreamListener**.onStreamShutdown(getStreamData());  
 }  
 **synchronized** (**statusWaiter**) {  
 **statusWaiter**.notify();  
 }  
 }  
  
 **public void** setStreamListener(StreamListener streamListener) {  
 **this**.**streamListener** = streamListener;  
 }  
  
 **public void** setDataListener(DataListener dataListener) {  
 **bufferManager**.setOnDataListener(dataListener);  
 }  
  
 **public void** setBufferManager(BufferManager bufferManager) {  
 **this**.**bufferManager** = bufferManager;  
 **if** (bufferManager != **null**) {  
 **this**.**bufferManager**.setPictureData(**streamData**.getPictureData());  
 }  
 }  
  
 **public void** setGeoListener(GeoListener geoListener) {  
 **this**.**geoListener** = geoListener;  
 }  
  
 **public void** setErrorListener(ErrorListener errorListener) {  
 **this**.**errorListener** = errorListener;  
 }  
  
 **public void** setSimpleStreamListener(SimpleStreamListener simpleStreamListener) {  
 **this**.**simpleStreamListener** = simpleStreamListener;  
 }  
  
 *//Getters* **public** StreamListener getStreamListener() {  
 **return streamListener**;  
 }  
  
 **public** StreamData getStreamData() {  
 **return streamData**;  
 }  
  
 **public** GeoListener getGeoListener() {  
 **return geoListener**;  
 }  
  
 **public** DataListener getDataListener() {  
 **return bufferManager**.getDataListener();  
 }  
}

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ | | | | | | | | | |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | №  документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
|  | измененных | замененных | Новых | анулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |