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  
Клиент**

**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  
Клиент**

**Текст программы**

RU. 17701729. 505900-01 12 1-1

Листов 34



## СОДЕРЖАНИЕ

[1. Текст программы 3](#_Toc451350581)

[1.1. Класс Coordinate.java 3](#_Toc451350582)

[1.2. Класс PictureData.java 3](#_Toc451350583)

[1.3. Класс StreamData.java 5](#_Toc451350584)

[1.4. Класс ErrorListener.java 7](#_Toc451350585)

[1.5. Класс Client.java 7](#_Toc451350586)

[1.6. Класс CameraManager.java 14](#_Toc451350587)

[1.7. Класс CameraPreview.java 16](#_Toc451350588)

[1.8. Класс ListAdapter.java 23](#_Toc451350589)

[1.9. Класс MainActivity.java 25](#_Toc451350590)

[1.10. Класс StartScreen.java 27](#_Toc451350591)

[1.11. Класс StreamingActivity.java 27](#_Toc451350592)

[Лист регистрации изменений 34](#_Toc451350593)

1. **Текст программы**
   1. **Класс Coordinate.java**

**package** com.example.aleksandr.socketstreamer.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* **int latitude**, **longitude**;  
  
 Coordinate() {  
 }  
  
 Coordinate(**int** latitude, **int** longitude) {  
 **this**();  
 setCoordinates(latitude, longitude);  
 }  
  
  
 *// Сеттеры* **public void** setLatitude(**int** latitude) {  
 **this**.**latitude** = latitude;  
 }  
  
 **public void** setLongitude(**int** longitude) {  
 **this**.**longitude** = longitude;  
 }  
  
 **public void** setCoordinates(**int** latitude, **int** longitude) {  
 **this**.**latitude** = latitude;  
 **this**.**longitude** = longitude;  
 }  
  
 **public int** getLatitude() {  
 **return latitude**;  
 }  
  
 **public int** getLongitude() {  
 **return longitude**;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Latitude: "** + **latitude** + **"\tLongitude: "** + **longitude**;  
 }  
}

* 1. **Класс PictureData.java**

**package** com.example.aleksandr.socketstreamer.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** com.example.aleksandr.socketstreamer.data.Abstractions;  
  
*/\*\*  
 \* Created by Aleksand Smilyanskiy on 30.04.2016.  
 \* "The more we do, the more we can do." ©  
 \*/***import** android.os.Parcel;  
**import** android.os.Parcelable;  
  
*/\*\*  
 \* Представляет всю необходимую информацию о стриме  
 \*/***public class** StreamData **implements** Parcelable{  
 *// 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**;  
 }  
  
  
 *// Parcelable* @Override  
 **public int** describeContents() {  
 **return** 0;  
 }  
  
 @Override  
 **public void** writeToParcel(Parcel dest, **int** flags) {  
 dest.writeString(getId());  
 dest.writeString(getName());  
 }  
  
 *// this is used to regenerate your object. All Parcelables must have a CREATOR that implements these two methods* **public static final** Parcelable.Creator<StreamData> ***CREATOR*** = **new** Parcelable.Creator<StreamData>() {  
 **public** StreamData createFromParcel(Parcel in) {  
 **return new** StreamData(in);  
 }  
  
 **public** StreamData[] newArray(**int** size) {  
 **return new** StreamData[size];  
 }  
 };  
  
 *// example constructor that takes a Parcel and gives you an object populated with it's values* **private** StreamData(Parcel in) {  
 setId(in.readString());  
 setName(in.readString());  
 }  
}

* 1. **Класс ErrorListener.java**

**package** com.example.aleksandr.socketstreamer.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 Client.java**

**package** com.example.aleksandr.socketstreamer.io;  
  
**import** android.location.Location;  
**import** android.location.LocationListener;  
**import** android.os.Bundle;  
**import** android.util.JsonReader;  
  
**import** com.example.aleksandr.socketstreamer.supporting.CameraPreview;  
**import** com.example.aleksandr.socketstreamer.UI.StreamingActivity;  
**import** com.example.aleksandr.socketstreamer.data.Listeners.ErrorListener;  
  
**import** com.google.gson.JsonObject;  
  
  
**import** org.json.JSONException;  
**import** org.json.JSONObject;  
  
  
**import** java.io.BufferedReader;  
**import** java.io.BufferedWriter;  
**import** java.io.IOException;  
**import** java.io.InputStreamReader;  
**import** java.io.OutputStreamWriter;  
**import** java.net.InetSocketAddress;  
**import** java.net.Socket;  
**import** java.util.Arrays;  
  
*/\*\*  
 \* Created by Aleksandr on 03.05.2016.  
 \*/***public class** Client **extends** Thread **implements** LocationListener {  
 *// Main parts* **private** Socket **socket**;  
 **private** StreamingActivity **uiThread**;  
  
 *// streams* **private** BufferedWriter **outputStream**;  
 **private** BufferedReader **bufferedReader**;  
 **private** JsonReader **jsonReader**;  
  
 *// Objects* **private** Location **lastLocation**;  
 **private** Thread **input**;  
 **private** Thread **output**;  
 **private final** Object **statusBell** = **new** Object(); *// слушатель изменения translationStatus* **private final** Object **ipAndHostBell** = **new** Object(); *// слушатель изменения host&ip  
  
 // Paramerers* **private** String **hostIp**;  
 **private int hostPort** = -1;  
 **private** CameraPreview **cameraPreview**;  
 **private int** translationStatus; *// -1 = закончено соединение, 0 = ожидание/не подсоединялось, 1 = активно  
  
 // listeners* **private** ErrorListener **errorListener**;  
  
 *// параметры* **private static final int *SERVER\_TIMEOUT*** = 10000;  
  
  
 *// Constructors* **private** Client(String hostIp, **int** hostPort) {  
 **this**.translationStatus = 0;  
 **this**.**hostIp** = hostIp;  
 **this**.**hostPort** = hostPort;  
 }  
  
 **public** Client(StreamingActivity uiThread, String hostIp, **int** hostPort, CameraPreview cameraPreview) {  
 **this**(hostIp, hostPort);  
 **this**.**cameraPreview** = cameraPreview;  
 **this**.**uiThread** = uiThread;  
 **this**.**errorListener** = (ErrorListener) uiThread;  
 }  
  
  
 *// Options* **public void** close() {  
 setTranslationStatus(-1);  
 **if** (**output** != **null** && **output**.isAlive()) {  
 **output**.interrupt();  
 }  
 **if** (**input** != **null** && **input**.isAlive()) {  
 **input**.interrupt();  
 }  
 **try** {  
 **socket**.close();  
 } **catch** (IOException e) {  
 **if** (**errorListener** != **null**) {  
 **errorListener**.onError(**"Error closing connection."**);  
 }  
 e.printStackTrace();  
 }  
 **synchronized** (**statusBell**) {  
 **statusBell**.notifyAll();  
 }  
 **synchronized** (**ipAndHostBell**) {  
 **ipAndHostBell**.notifyAll();  
 }  
 }  
  
  
 *// helpers* **private boolean** openStream() {  
 **if** (**hostPort** <= 0 || **hostIp** == **null** || **hostIp**.equals(**""**)) {  
 **return false**;  
 }  
 setTranslationStatus(0);  
 **try** {  
 **socket** = **new** Socket();  
 **socket**.connect(**new** InetSocketAddress(**hostIp**, **hostPort**), ***SERVER\_TIMEOUT***);  
  
 **outputStream** = **new** BufferedWriter(**new** OutputStreamWriter(**socket**.getOutputStream()));  
 **bufferedReader** = **new** BufferedReader(**new** InputStreamReader(**socket**.getInputStream()));  
 **jsonReader** = **new** JsonReader(**bufferedReader**);  
 **jsonReader**.setLenient(**true**);  
 } **catch** (IOException e) {  
 **if** (**errorListener** != **null**) {  
 **errorListener**.onError(**"Error on opening stream: server not responding."**);  
 }  
 e.printStackTrace();  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **private boolean** checkConnection() {  
 **return** translationStatus != -1 && translationStatus != 0 && **socket**.isConnected();  
 }  
  
 **private void** pauseStream() {  
 setTranslationStatus(0);  
 }  
  
 **private void** startStream() {  
 setTranslationStatus(1);  
 }  
  
  
 @Override  
 **public void** run() {  
 **super**.run();  
  
 **if** (!openStream()) {  
 **uiThread**.offButton();  
 setTranslationStatus(-1);  
 **return**;  
 }  
  
 **input** = **new** inputAnalyze();  
 **output** = **new** outputData();  
 **input**.start();  
 **output**.start();  
 }  
  
 *// thread for input and output* **class** inputAnalyze **extends** Thread {  
 @Override  
 **public void** run() {  
 **super**.run();  
 **while** (!**this**.isInterrupted()) {  
 **if** (!readMessage()) {  
 **break**;  
 }  
 }  
 }  
  
 **private boolean** readMessage() {  
 **try** {  
 **jsonReader**.beginObject();  
 **switch** (**jsonReader**.nextName()) {  
 **case "state"**: {  
 String condition = **jsonReader**.nextString();  
 **if** (**"start"**.equals(condition)) {  
 startStream();  
 **break**;  
 }  
 **if** (**"wait"**.equals(condition)) {  
 pauseStream();  
 **break**;  
 }  
 *// SMTH new here* **break**;  
 }  
 **case "heartbeat"**: {  
 String condition = **jsonReader**.nextString();  
 **if** (**"request"**.equals(condition)) {  
 sendHeartbeatAnswer();  
 **break**;  
 }  
 *// SMTH new* **break**;  
 }  
 }  
 **jsonReader**.endObject();  
  
 } **catch** (IOException | IllegalStateException e) {  
 **if** (!**socket**.isClosed()) {  
 **uiThread**.offButton();  
 setTranslationStatus(-1);  
 }  
 **return false**;  
 }  
 **return true**;  
 }  
  
 **private void** sendHeartbeatAnswer() {  
 **if** (checkConnection()) {  
 **return**;  
 }  
  
 JsonObject heartbeat = **new** JsonObject();  
 heartbeat.addProperty(**"heartbeat"**, **"answer"**);  
 **try** {  
 **outputStream**.write(heartbeat.toString() + **"\n"**);  
 **outputStream**.flush();  
 } **catch** (IOException e) {  
 **if** (**errorListener** != **null**)  
 **errorListener**.onError(**"Error in sending heartbeat answer: "** + e.getMessage());  
 e.printStackTrace();  
 }  
 }  
 }  
  
 **class** outputData **extends** Thread {  
 **int previousStatus** = 0;  
 **private static final** String ***STOP\_COMMAND*** = **"STOP"**;  
  
 @Override  
 **public void** run() {  
 **super**.run();  
  
 **try** {  
 sendInitialMessage();  
 } **catch** (IOException e) {  
 **if** (**errorListener** != **null**) {  
 **errorListener**.onError(**"Error sending initial message, closing."**);  
 }  
 e.printStackTrace();  
 close();  
 **return**;  
 }  
  
 **while** (!**this**.isInterrupted()) {  
 **try** {  
 **if** (translationStatus == -1){  
 **return**;  
 }  
 **while** (translationStatus == 1) {  
 sendData();  
 }  
 } **catch** (JSONException e) {  
 **if** (**errorListener** != **null**) {  
 **errorListener**.onError(**"Json parsing error."**);  
 }  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 **if** (translationStatus != -1){  
 **if** (errorListener != **null**) {  
 errorListener.onError(**"Sending data error."**);  
 }  
 e.printStackTrace();  
 }  
 }  
 **synchronized** (statusBell) {  
 **try** {  
 statusBell.wait(1000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 **break**;  
 }  
 }  
 }  
 }  
  
 **private void** sendData() **throws** JSONException, IOException {  
 **if** (translationStatus != 1) {  
 **return**;  
 }  
 **byte**[] data = **cameraPreview**.getImageBuffer();  
 String dataS = Arrays.*toString*(data);  
 JSONObject jsonObject = **new** JSONObject();  
 jsonObject.put(**"data"**, dataS);  
 **double**[] geo = **new double**[2];  
 **if** (**lastLocation** != **null**) {  
 geo[0] = **lastLocation**.getAltitude();  
 geo[1] = **lastLocation**.getLatitude();  
 } **else** {  
 geo[0] = geo[1] = -1;  
 }  
 jsonObject.put(**"geo"**, **"["** + geo[0] + **", "** + geo[1] + **"]"**);  
 **outputStream**.write(jsonObject.toString() + **"\n"**);  
 **outputStream**.flush();  
 }  
  
 **private void** sendInitialMessage() **throws** IOException {  
 **if** (translationStatus == -1) {  
 **return**;  
 }  
 JSONObject jsonObject = **new** JSONObject();  
 **try** {  
 jsonObject.put(**"length"**, **cameraPreview**.getPreviewLength());  
 jsonObject.put(**"width"**, **cameraPreview**.getPreviewWidth());  
 jsonObject.put(**"height"**, **cameraPreview**.getPreviewHeight());  
*// jsonObject.put("length", cameraPreview.getAc)* } **catch** (JSONException e) {  
 **if** (**errorListener** != **null**) {  
 **errorListener**.onError(**">>> Error making json in initial message."**);  
 }  
 e.printStackTrace();  
 **return**;  
 }  
  
 **outputStream**.write(jsonObject.toString() + **"\n"**);  
 **outputStream**.flush();  
 }  
  
 **private void** sendStop() **throws** IOException {  
 **if** (translationStatus == -1){  
 **return**;  
 }  
 outputStream.write(Arrays.toString(STOP\_COMMAND.getBytes()));  
 outputStream.flush();  
 }  
 }  
  
 *// Setters-getters* **private void** setTranslationStatus(**int** translationStatus) {  
 **this**.translationStatus = translationStatus;  
 **synchronized** (statusBell) {  
 statusBell.notify();  
 }  
 **synchronized** (ipAndHostBell) {  
 ipAndHostBell.notify();  
 }  
 }  
  
 **public int** getTranslationStatus() {  
 **return** translationStatus;  
 }  
  
 **public void** setHostIp(String hostIp) {  
 **this**.hostIp = hostIp;  
 **synchronized** (ipAndHostBell) {  
 ipAndHostBell.notify();  
 }  
 }  
  
 **public void** setHostPort(**int** hostPort) {  
 **this**.hostPort = hostPort;  
 **synchronized** (ipAndHostBell) {  
 ipAndHostBell.notify();  
 }  
 }  
  
 *// Location Listener* @Override  
 **public void** onLocationChanged(Location location) {  
 **this**.lastLocation = location;  
 }  
  
 @Override  
 **public void** onStatusChanged(String provider, **int** status, Bundle extras) {  
  
 }  
  
 @Override  
 **public void** onProviderEnabled(String provider) {  
  
 }  
  
 @Override  
 **public void** onProviderDisabled(String provider) {  
  
 }  
  
}

* 1. **Класс CameraManager.java**

**package** com.example.aleksandr.socketstreamer.supporting;  
  
**import** android.content.Context;  
**import** android.hardware.Camera;  
**import** android.widget.Toast;  
  
*/\*\*  
 \* Менеджер камеры для управления с hardware.Camera.  
 \*/***public class** CameraManager {  
 */\*\*  
 \* Инстанс камеры  
 \*/* **private** Camera **camera**;  
 */\*\*  
 \* Контекс в котором используется камера  
 \*/* **private** Context **context**;  
  
 */\*\*  
 \* Создание менеджера камеры  
 \*  
 \** ***@param context*** *использованный контекст программы  
 \*/* **public** CameraManager(Context context) {  
 **this**.**context** = context;  
 **camera** = *getCameraInstance*();  
 }  
  
 */\*\*  
 \* Getter для инстанса камеры  
 \*  
 \** ***@return*** *текущий инстанс камеры  
 \*/* **public** Camera getCamera() {  
 **return camera**;  
 }  
  
 */\*\*  
 \* Освобождение камеры  
 \*/* **private void** releaseCamera() {  
 **if** (**camera** != **null**) {  
 **camera**.release();  
 **camera** = **null**;  
 }  
 }  
  
 */\*\*  
 \* При остановке - освобождение камеры  
 \*/* **public void** onPause() {  
 releaseCamera();  
 }  
  
 */\*\*  
 \* При продолжении - восстановление инстанса камеры  
 \*/* **public void** onResume() {  
 **if** (**camera** == **null**) {  
 **camera** = *getCameraInstance*();  
 }  
  
 Toast.*makeText*(**context**, **"preview size = "** + **camera**.getParameters().getPreviewSize().**width** +  
 **", "** + **camera**.getParameters().getPreviewSize().**height**, Toast.***LENGTH\_LONG***).show();  
 }  
  
 */\*\*  
 \* Безопасное получение инстанса камеры  
 \*  
 \** ***@return*** *инстанс камеры  
 \*/* **private static** Camera getCameraInstance() {  
 Camera c = **null**;  
 **try** {  
 c = Camera.*open*();  
 } **catch** (Exception e) {  
 *// Камера недоступна или не может быть включена сейчас* e.printStackTrace();  
 }  
 **return** c;  
 }  
  
}

* 1. **Класс CameraPreview.java**

**package** com.example.aleksandr.socketstreamer.supporting;  
  
**import** android.content.Context;  
**import** android.graphics.Bitmap;  
**import** android.graphics.BitmapFactory;  
**import** android.graphics.Canvas;  
**import** android.graphics.ColorMatrix;  
**import** android.graphics.ColorMatrixColorFilter;  
**import** android.graphics.ImageFormat;  
**import** android.graphics.Matrix;  
**import** android.graphics.Paint;  
**import** android.graphics.Rect;  
**import** android.graphics.YuvImage;  
**import** android.hardware.Camera;  
**import** android.hardware.Camera.Parameters;  
**import** android.hardware.Camera.PreviewCallback;  
**import** android.hardware.Camera.Size;  
**import** android.renderscript.Allocation;  
**import** android.renderscript.Element;  
**import** android.renderscript.RenderScript;  
**import** android.renderscript.ScriptIntrinsicBlur;  
**import** android.util.Log;  
**import** android.view.SurfaceHolder;  
**import** android.view.SurfaceView;  
  
**import** java.io.ByteArrayOutputStream;  
**import** java.io.IOException;  
**import** java.util.LinkedList;  
**import** java.util.List;  
  
*/\*\*  
 \* Представление для камеры. Показывает текущую картинку с камеры.  
 \*/***public class** CameraPreview **extends** SurfaceView **implements** SurfaceHolder.Callback, PreviewCallback {  
 *// Objects* **private** SurfaceHolder **surfaceHolder**;  
 **private** Camera **camera**; *// инстанс камеры* **private** Size **previewSize**; *// размер изображения с камеры* **private int** frameLength; *// длина одного фрейма информации* **private** LinkedList<**byte**[]> **framesQueue** = **new** LinkedList<>(); *// очередь фреймов* **private byte**[] **lastFrame** = **null**; *// последний фрейм в очереди* **private** FrameQueue **frameQueue** = **new** FrameQueue();  
 **private** ColorMatrixColorFilter **filter** = **null**;  
 **private** Paint **paint** = **new** Paint();  
 **private** Filters **currentFilter**;  
 **private** Context **context**;  
  
 *// Properties* **private static final** String ***TAG*** = **"CAMERA\_PREVIEW"**;  
 **private static final int *MAX\_BUFFER*** = 15; *// макс. кол-во фреймов поддерживаемое в очереди* **private static final int *JPEG\_QUALITY*** = 70; *//experimental* **private static final int *BLUR\_RADIUS*** = 9; *// радиус эффекта размытия  
  
 // все возможные фильтры* **public enum** Filters {  
 ***DEFAULT***, ***GREYSCALE***, ***SEPIA***, ***BINARY***, ***BLUR*** }  
  
 *// Blur objects - RenderScript* **private** RenderScript **rs**;  
 **private** ScriptIntrinsicBlur **script**;  
  
  
 **public** CameraPreview(Context context, Camera camera) {  
 **super**(context);  
 *// установка инстанса камеры* **this**.**camera** = camera;  
  
 *// установка отображения и колбэка* **surfaceHolder** = getHolder();  
 **surfaceHolder**.addCallback(**this**);  
  
 *// deprecated setting, but required on Android versions prior to 3.0* **surfaceHolder**.setType(SurfaceHolder.***SURFACE\_TYPE\_PUSH\_BUFFERS***);  
  
 *// получение параметров камеры* Parameters params = **this**.**camera**.getParameters();  
 *// получение поддерживаемых форматов отображения* List<Size> sizes = params.getSupportedPreviewSizes();  
 **for** (Size s : sizes) {  
 Log.*i*(***TAG***, **"preview size = "** + s.**width** + **", "** + s.**height**);  
 }  
  
 *// установка маленького формата превью, т.к. меньше задержек будет* params.setPreviewSize(320, 240);  
 **this**.**camera**.setParameters(params);  
  
 *// получение параметров превью* **previewSize** = **this**.**camera**.getParameters().getPreviewSize();  
 Log.*i*(***TAG***, **"preview size = "** + **previewSize**.**width** + **", "** + **previewSize**.**height**);  
  
 *// получение текущего формата отображения* **int** format = **this**.**camera**.getParameters().getPreviewFormat();  
 *// по формату устанавливается размер фрейма* frameLength = **previewSize**.**width** \* **previewSize**.**height** \* ImageFormat.*getBitsPerPixel*(format) / 8;  
  
  
 *//* ***TODO: 12.05.2016 FOR Filter!, delete if not working*** setWillNotDraw(**false**);  
 }  
  
  
 *// Options* **private void** resetBuff() {  
 *// очищение "очереди" фреймов и последнего фрейма* **synchronized** (**framesQueue**) {  
 **framesQueue**.clear();  
 **lastFrame** = **null**;  
 }  
 }  
  
  
 *// Surface methods* **public void** onPause() {  
 *// при паузе - освободить камеру* **if** (**camera** != **null**) {  
 **camera**.setPreviewCallback(**null**);  
 **camera**.stopPreview();  
 }  
 *// и очистить буфер* resetBuff();  
 }  
  
 **public void** surfaceCreated(SurfaceHolder holder) {  
 **try** {  
 *// установка дисплея дли отображения* **camera**.setPreviewDisplay(holder);  
 *// начало превью* **camera**.startPreview();  
 } **catch** (IOException e) {  
 Log.*d*(***TAG***, **"Error setting camera preview: "** + e.getMessage());  
 }  
 }  
  
 **public void** surfaceDestroyed(SurfaceHolder holder) {  
  
 }  
  
 **public void** surfaceChanged(SurfaceHolder holder, **int** format, **int** w, **int** h) {  
 *// при если текущего отображения нет - менять нечего* **if** (**surfaceHolder**.getSurface() == **null**) {  
 **return**;  
 }  
  
 **try** {  
 *// остановка отображения* **camera**.stopPreview();  
 *// очищение буферов* resetBuff();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
  
 **try** {  
 *// TODO: 01.04.2016 Проверить необходиомсть перезаписывать surfaceHolder  
 // установка прежнего Callback-a* **camera**.setPreviewCallback(**this**);  
 **camera**.setPreviewDisplay(**surfaceHolder**);  
 **camera**.startPreview();  
  
 } **catch** (Exception e) {  
 Log.*d*(***TAG***, **"Error starting camera preview: "** + e.getMessage());  
 }  
 }  
  
 *// кастомная очередь фреймов* **private class** FrameQueue {  
 *// Queue* **private final** LinkedList<**byte**[]> **buffer** = **new** LinkedList<>();  
 *// Instances of variables* **private** ByteArrayOutputStream **convertingStream** = **new** ByteArrayOutputStream();  
 **private** ByteArrayOutputStream **outputStream** = **new** ByteArrayOutputStream();  
 **private** YuvImage **yuvImage**;  
 **private byte**[] **lastFrame**;  
 **private** Bitmap **lastConvertedFrame**;  
 **private** Bitmap **lastScaledFrame**;  
 **private boolean lastFrameConverted** = **false**;  
 **private** Canvas **canvas**;  
  
 **public void** storeData(**byte**[] data) {  
 **synchronized** (**buffer**) {  
 *// если не усевает работать конвертация* **if** (**buffer**.size() == ***MAX\_BUFFER***) {  
 **buffer**.poll();  
 }  
 **buffer**.add(data);  
 **lastFrameConverted** = **false**;  
 }  
 }  
  
 **public synchronized byte**[] getBytedLastFrame() {  
 **if** (!**lastFrameConverted**) {  
 convertLastFrame();  
 }  
 Bitmap bmOverlay = Bitmap.*createBitmap*(**lastConvertedFrame**.getWidth(), **lastConvertedFrame**.getHeight(), lastConvertedFrame.getConfig());  
 canvas = **new** Canvas(bmOverlay);  
 canvas.drawBitmap(lastConvertedFrame, **new** Matrix(), paint);  
 outputStream = **new** ByteArrayOutputStream();  
 bmOverlay.compress(Bitmap.CompressFormat.JPEG, JPEG\_QUALITY, outputStream);  
 **return** outputStream.toByteArray();  
 }  
  
 **public synchronized** Bitmap getBitmapLastFrame() {  
 **if** (!lastFrameConverted) {  
 convertLastFrame();  
 }  
 **return** lastScaledFrame;  
 }  
  
 **private byte**[] getLastUnconvertedFrame() {  
 **synchronized** (buffer) {  
 **if** (buffer.size() > 0) {  
 lastFrame = buffer.poll();  
 }  
 }  
 **return** lastFrame;  
 }  
  
 **private void** convertLastFrame() {  
 **if** (buffer.size() == 0) {  
 **return**;  
 }  
 yuvImage = **new** YuvImage(getLastUnconvertedFrame(), ImageFormat.NV21, previewSize.width, previewSize.height, **null**);  
 convertingStream.reset();  
 yuvImage.compressToJpeg(**new** Rect(0, 0, previewSize.width, previewSize.height), JPEG\_QUALITY, convertingStream);  
 lastConvertedFrame = BitmapFactory.decodeByteArray(convertingStream.toByteArray(), 0, convertingStream.size());  
 **try** {  
 lastScaledFrame = Bitmap.createScaledBitmap(lastConvertedFrame, getWidth(), getHeight(), **true**);  
 lastFrameConverted = **true**;  
 } **catch** (NullPointerException ignored) {  
 lastFrameConverted = **false**;  
 }  
  
 }  
 }  
  
  
 *// Filers* **public void** setFilter(Filters filter) {  
 **switch** (filter) {  
 **case** GREYSCALE: {  
 paint.setColorFilter(**new** ColorMatrixColorFilter(FilterMatrix.getGraysacle()));  
 **break**;  
 }  
 **case** SEPIA: {  
 paint.setColorFilter(**new** ColorMatrixColorFilter(FilterMatrix.getSepia()));  
 **break**;  
 }  
 **case** BINARY: {  
 paint.setColorFilter(**new** ColorMatrixColorFilter(FilterMatrix.getBinary()));  
 **break**;  
 }  
 **case** BLUR: {  
 rs = RenderScript.create(context);  
 script = ScriptIntrinsicBlur.create(rs, Element.U8\_4(rs));  
 script.setRadius(9f);  
 **break**;  
 }  
 **default**: {  
 paint = **new** Paint();  
 **break**;  
 }  
 }  
 currentFilter = filter;  
 }  
  
 **private** Bitmap blur(Bitmap original) {  
 Bitmap bitmap = Bitmap.createBitmap(  
 original.getWidth(), original.getHeight(),  
 Bitmap.Config.ARGB\_8888);  
  
 rs = RenderScript.create(context);  
 script = ScriptIntrinsicBlur.create(rs, Element.U8\_4(rs));  
 Allocation allocIn = Allocation.createFromBitmap(rs, original);  
 Allocation allocOut = Allocation.createFromBitmap(rs, bitmap);  
  
 ScriptIntrinsicBlur blur = ScriptIntrinsicBlur.create(rs, Element.U8\_4(rs));  
 blur.setInput(allocIn);  
 blur.forEach(allocOut);  
  
 allocOut.copyTo(bitmap);  
 rs.destroy();  
 **return** bitmap;  
 }  
  
 *// Setters* **public void** setCamera(Camera camera) {  
 **this**.camera = camera;  
 }  
  
 **public void** setContext(Context context) {  
 **this**.context = context;  
 }  
  
 *// Getters* **public int** getPreviewLength() {  
 **return** frameLength;  
 }  
  
 **public int** getPreviewWidth() {  
 **return** previewSize.width;  
 }  
  
 **public int** getPreviewHeight() {  
 **return** previewSize.height;  
 }  
  
 **public int** getActualWidth() {  
 **return** getMeasuredWidth();  
 }  
  
 **public int** getActualHeight() {  
 **return** getMeasuredHeight();  
 }  
  
 **public byte**[] getImageBuffer() {  
 *// синхронизированное по Queue получение фрейма из "очереди"* **return** frameQueue.getBytedLastFrame();  
*// synchronized (framesQueue) {  
// if (framesQueue.size() > 0) {  
// lastFrame = framesQueue.poll();  
// }  
// }  
//  
// return lastFrame;* }  
  
  
 *// PreviewCallback* @Override  
 **public void** onPreviewFrame(**byte**[] data, Camera camera) {  
*// // запись изображений с камеры в очередь  
// synchronized (framesQueue) {  
// // Если информация не успевает уходить - пропустим последние фреймы  
// if (framesQueue.size() == MAX\_BUFFER) {  
// framesQueue.poll();  
// }  
// // запишем новую информацию  
// framesQueue.add(data);  
//  
// // experimental  
// createBitmap();  
// invalidate();  
// }* frameQueue.storeData(data);  
 invalidate();  
 }  
  
 @Override  
 **protected void** onDraw(Canvas canvas) {  
  
 Bitmap bitmap = frameQueue.getBitmapLastFrame();  
 **if** (bitmap == **null**)  
 **return**;  
 **synchronized** (paint) {  
 **switch** (currentFilter) {  
 **case** BLUR: {  
 canvas.drawBitmap(blur(bitmap), 0, 0, paint);  
 **break**;  
 }  
 **default**: {  
 canvas.drawBitmap(bitmap, 0, 0, paint);  
 **break**;  
 }  
 }  
 }  
 }  
}  
  
**class** FilterMatrix {  
 **static** ColorMatrix getGraysacle() {  
 ColorMatrix colorMatrix = **new** ColorMatrix();  
 colorMatrix.setSaturation(0);  
 **return** colorMatrix;  
 }  
  
 **static** ColorMatrix getSepia() {  
 ColorMatrix colorMatrix = getGraysacle();  
  
 ColorMatrix colorScale = **new** ColorMatrix();  
 colorScale.setScale(1, 1, 0.8f, 1);  
  
 *// Convert to grayscale, then apply brown color* colorMatrix.postConcat(colorScale);  
 **return** colorMatrix;  
 }  
  
 **static** ColorMatrix getBinary() {  
 ColorMatrix colorMatrix = getGraysacle();  
  
 **float** m = 255f;  
 **float** t = -255 \* 128f;  
 ColorMatrix threshold = **new** ColorMatrix(**new float**[]{  
 m, 0, 0, 1, t,  
 0, m, 0, 1, t,  
 0, 0, m, 1, t,  
 0, 0, 0, 1, 0  
 });  
  
 *// Convert to grayscale, then scale and clamp* colorMatrix.postConcat(threshold);  
  
 **return** colorMatrix;  
 }  
}

* 1. **Класс ListAdapter.java**

**package** com.example.aleksandr.socketstreamer.supporting;  
  
**import** android.content.Context;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.BaseAdapter;  
**import** android.widget.TextView;  
  
**import** com.example.aleksandr.socketstreamer.R;  
**import** com.example.aleksandr.socketstreamer.data.Abstractions.StreamData;  
  
**import** java.util.ArrayList;  
  
*/\*\*  
 \* Created by Aleksandr on 10.05.2016.  
 \*/***public class** ListAdapter **extends** BaseAdapter {  
 Context context;  
 ArrayList<StreamData> data;  
 **private static** LayoutInflater *inflater* = **null**;  
  
 **public** ListAdapter(Context context, ArrayList<StreamData> data) {  
 **this**.**context** = context;  
 **this**.**data** = data;  
 *inflater* = (LayoutInflater) context  
 .getSystemService(Context.***LAYOUT\_INFLATER\_SERVICE***);  
 }  
  
 @Override  
 **public int** getCount() {  
 **return data**.size();  
 }  
  
 @Override  
 **public** Object getItem(**int** position) {  
 **return data**.get(position);  
 }  
  
 @Override  
 **public long** getItemId(**int** position) {  
 **return** position;  
 }  
  
 @Override  
 **public** View getView(**int** position, View convertView, ViewGroup parent) {  
 View view = convertView;  
 **if** (view == **null**)  
 view = *inflater*.inflate(R.layout.***item\_row***, **null**);  
 TextView name = (TextView) view.findViewById(R.id.lbl\_nameEnter);  
 TextView id = (TextView) view.findViewById(R.id.lbl\_idEnter);  
 **if** (data.get(position).getName() == **null**){  
 name.setText(**"Не указано"**);  
 }  
 **else**{  
 name.setText(data.get(position).getName());  
 }  
 id.setText(data.get(position).getId());  
 **return** view;  
 }  
}

* 1. **Класс MainActivity.java**

**package** com.example.aleksandr.socketstreamer.UI;  
  
**import** android.app.Activity;  
**import** android.content.Intent;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.ListView;  
**import** android.widget.Toast;  
  
**import** com.example.aleksandr.socketstreamer.R;  
**import** com.example.aleksandr.socketstreamer.data.Abstractions.StreamData;  
**import** com.example.aleksandr.socketstreamer.supporting.ListAdapter;  
**import** com.google.gson.JsonObject;  
  
**import** org.json.JSONException;  
**import** org.json.JSONObject;  
  
**import** java.util.ArrayList;  
  
  
**public class** MainActivity **extends** Activity {  
 *// UI* Button btnLaunchStreamActivity;  
 ListView listView;  
  
 *// Parameters* **private static final int** REQUEST\_CODE\_LOGIN = 1;  
 **private static final int** REQUEST\_CODE\_START = 2;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
  
 Intent loginActivity = **new** Intent(**this**, StartScreen.**class**);  
 startActivityForResult(loginActivity, REQUEST\_CODE\_LOGIN);  
*// FloatingActionButton fabButton = new FloatingActionButton.Builder(this)  
// .withDrawable(R.drawable.camera)  
// .withButtonColor(Color.WHITE)  
// .withGravity(Gravity.BOTTOM | Gravity.RIGHT)  
// .withMargins(0, 0, 16, 16)  
// .create();  
  
 // Установим запуск стрима по нажатию кнопки* btnLaunchStreamActivity = (Button) findViewById(R.id.btnLaunchStreamActivity);  
 btnLaunchStreamActivity.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 launchStream();  
 }  
 });  
  
 listView = (ListView) findViewById(R.id.lv\_streams);  
 }  
  
 **protected void** launchStream() {  
 Intent streamingActivity = **new** Intent(**this**, StreamingActivity.**class**);  
 startActivity(streamingActivity);  
 }  
  
  
 @Override  
 **protected void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
 **if** (data == **null** || resultCode != RESULT\_OK) {  
 finish();  
 **return**;  
 }  
  
 **switch** (requestCode) {  
 **case** REQUEST\_CODE\_LOGIN: {  
 *// onLoginEnd(resultCode, data);* **break**;  
 }  
 **case *REQUEST\_CODE\_START***: {  
 onStreamingActivityEnd(resultCode, data);  
 **break**;  
 }  
 }  
 }  
  
 **protected void** onLoginEnd(**int** resultCode, Intent data) {  
 *// получаем результат входа* Toast.*makeText*(**this**, **"Server exist!"**, Toast.***LENGTH\_SHORT***).show();  
  
 ArrayList<String> streams = data.getStringArrayListExtra(**"streams"**);  
 ArrayList<StreamData> streamDatas = **null**;  
 **try** {  
 streamDatas = parseJsonStreams(streams);  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 Toast.*makeText*(**this**, **"Server send wrong stream info"**, Toast.***LENGTH\_LONG***).show();  
 **return**;  
 }  
  
 **listView**.setAdapter(**new** ListAdapter(**this**, streamDatas));  
 }  
  
 **private** ArrayList<StreamData> parseJsonStreams(ArrayList<String> streams) **throws** JSONException {  
 JSONObject jsonObject = **null**;  
 String name;  
 ArrayList<StreamData> streamDatas = **new** ArrayList<>(streams.size());  
 **for** (String stream : streams) {  
 jsonObject = **new** JSONObject(stream);  
 StreamData streamData = **new** StreamData(jsonObject.getString(**"id"**), **null**);  
 name = jsonObject.getString(**"name"**);  
 **if** (!**"null"**.equals(name)) {  
 streamData.setName(name);  
 }  
 streamDatas.add(streamData);  
 }  
 **return** streamDatas;  
 }  
  
 **protected void** onStreamingActivityEnd(**int** resultCode, Intent data) {  
  
 }  
}

* 1. **Класс StartScreen.java**
  2. **Класс StreamingActivity.java**

**package** com.example.aleksandr.socketstreamer.UI;  
  
**import** android.Manifest;  
**import** android.app.Activity;  
**import** android.app.AlertDialog;  
**import** android.content.Context;  
**import** android.content.DialogInterface;  
**import** android.content.pm.PackageManager;  
**import** android.graphics.ColorMatrix;  
**import** android.graphics.Paint;  
**import** android.hardware.Camera;  
**import** android.location.LocationManager;  
**import** android.os.Bundle;  
**import** android.support.v4.app.ActivityCompat;  
**import** android.view.LayoutInflater;  
**import** android.view.Menu;  
**import** android.view.MenuItem;  
**import** android.view.View;  
**import** android.view.ViewTreeObserver;  
**import** android.view.WindowManager;  
**import** android.widget.EditText;  
**import** android.widget.FrameLayout;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** com.dd.morphingbutton.MorphingButton;  
**import** com.example.aleksandr.socketstreamer.supporting.CameraManager;  
**import** com.example.aleksandr.socketstreamer.supporting.CameraPreview;  
**import** com.example.aleksandr.socketstreamer.R;  
**import** com.example.aleksandr.socketstreamer.data.Listeners.ErrorListener;  
**import** com.example.aleksandr.socketstreamer.io.Client;  
  
*/\*\*  
 \* Активити для проведения трансляций  
 \*/***public class** StreamingActivity **extends** Activity **implements** ErrorListener {  
 *// Parts* **private** Client client;  
  
  
 *// Parameters* **private static final float** LOCATION\_REFRESH\_DISTANCE = 5;  
 **private static final long** LOCATION\_REFRESH\_TIME = 3000;  
 **private final** String DEFAULT\_IP = **"77.94.175.81"**;  
 **private final int** DEFAULT\_PORT = 8585;  
  
 *// Ui* TextView lbl\_port; *// текст-порт* TextView lbl\_ip; *// текст-ip* TextView lbl\_location; *// текст-длкация* MorphingButton btnStart; *// кнопка-старт/стоп  
  
 // Objects* **private** CameraPreview cameraPreview;  
 **private** CameraManager cameraManager;  
 **private** Context context;  
 **private** LocationManager locationManager;  
 **private** MorphingOperations morphingOperations;  
  
 *// Indicators-local values* **private boolean** streamStatus = **false**;  
 **private** String ip = DEFAULT\_IP;  
 **private int** port = DEFAULT\_PORT;  
 **private** StreamFactory streamFactory = **new** StreamFactory();  
  
  
 *// Activity* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_stream);  
  
 getWindow().addFlags(WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON);  
  
 *// контекст программы* context = **this**;  
  
 FrameLayout preview = (FrameLayout) findViewById(R.id.camera\_preview);  
*// ViewTreeObserver vto = preview.getViewTreeObserver();  
// vto.addOnGlobalLayoutListener(new ViewTreeObserver.OnGlobalLayoutListener() {  
// @Override  
// public void onGlobalLayout() {  
// preview.getViewTreeObserver().removeGlobalOnLayoutListener(this);  
//// if (cameraPreview != null) {  
//// cameraPreview.setActualPreviewSize(preview.getMeasuredWidth(), preview.getMeasuredHeight());  
//// }  
// previewWidth = preview.getMeasuredWidth();  
// previewHeight = preview.getMeasuredHeight();  
// if (cameraPreview != null){  
// cameraPreview.setActualPreviewSize(previewWidth,previewHeight);  
// }  
//  
// }  
// });* streamFactory.createCameraPreview();  
 preview.addView(cameraPreview);  
  
  
 *// Установка ip* lbl\_ip = (TextView) findViewById(R.id.lbl\_ip);  
 setIp(DEFAULT\_IP);  
 *// Установка port* lbl\_port = (TextView) findViewById(R.id.lbl\_port);  
 setPort(DEFAULT\_PORT);  
  
 *// Установка локации* lbl\_location = (TextView) findViewById(R.id.lbl\_location);  
  
 *// Morphing button* btnStart = (MorphingButton) findViewById(R.id.button\_capture);  
  
 morphingOperations = **new** MorphingOperations();  
  
 **final** StreamingActivity parent = **this**;  
 *// настройка кнопки* btnStart.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 *// Выключение трансляции* **if** (streamStatus) {  
 btnStart.morph(morphingOperations.off);  
  
 *// выключение трансляции* client.close();  
 streamStatus = **false**;  
 }  
 *// Запуск трансляции* **else** {  
 btnStart.morph(morphingOperations.on);  
  
 *// создание сокет клиента для отправки* streamFactory.createStream(parent);  
 client.start();  
  
 streamStatus = **true**;  
 }  
 }  
 });  
  
  
 *// изначально выключен* btnStart.morph(morphingOperations.off);  
 }  
 @Override  
 **protected void** onPause() {  
 **super**.onPause();  
 closeSocketClient();  
 **cameraPreview**.onPause();  
 **cameraManager**.onPause(); *// release the camera immediately on pause event* **if** (**streamStatus**)  
 **btnStart**.callOnClick();  
 reset();  
 }  
 @Override  
 **protected void** onResume() {  
 **super**.onResume();  
 **cameraManager**.onResume();  
 **cameraPreview**.setCamera(**cameraManager**.getCamera());  
 }  
  
 *// Error listener* @Override  
 **public void** onError(**final** String message) {  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 Toast.*makeText*(**context**, message, Toast.***LENGTH\_SHORT***).show();  
 }  
 });  
 }  
  
 *// operations for morphing button* **class** MorphingOperations {  
 **public final** MorphingButton.Params **off** = MorphingButton.Params.*create*()  
 .duration(500)  
 .cornerRadius((**int**) getResources().getDimension(R.dimen.***btn\_10dp***))  
 .width((**int**) getResources().getDimension(R.dimen.***btn\_200p***))  
 .height((**int**) getResources().getDimension(R.dimen.***btn\_56dp***))  
 .color(R.color.***mb\_blue***)  
 .colorPressed(R.color.***mb\_blue\_dark***)  
 .text(getResources().getString(R.string.***start***));  
 **public final** MorphingButton.Params **on** = MorphingButton.Params.*create*()  
 .duration(500)  
 .cornerRadius((**int**) getResources().getDimension(R.dimen.***btn\_56dp***)) *// 56 dp* .width((**int**) getResources().getDimension(R.dimen.***btn\_56dp***)) *// 56 dp* .height((**int**) getResources().getDimension(R.dimen.***btn\_56dp***)) *// 56 dp* .color(R.color.***mb\_blue***) *// normal state color* .colorPressed(R.color.***mb\_blue\_dark***) *// pressed state color* .icon(R.drawable.***stop***); *// icon* }  
 *// Цепочка вызовов создания стрима* **private class** StreamFactory {  
 **private void** createCameraManager() {  
 **if** (cameraManager != **null**) {  
 **return**;  
 }  
 cameraManager = **new** CameraManager(context);  
 }  
  
 **public void** createCameraPreview() {  
 createCameraManager();  
 **if** (cameraPreview != **null**) {  
 **return**;  
 }  
 cameraPreview = **new** CameraPreview(context, cameraManager.getCamera());  
 cameraPreview.setFilter(CameraPreview.Filters.DEFAULT);  
 cameraPreview.setContext(getApplicationContext());  
*// cameraPreview.setActualPreviewSize(preview.getWidth(), preview.getHeight());  
 // cameraPreview.setActualPreviewSize(previewWidth, previewHeight);* }  
  
 **private void** createLocationManager() {  
 **if** (locationManager != **null**) {  
 **return**;  
 }  
 locationManager = (LocationManager) getSystemService(LOCATION\_SERVICE);  
 }  
  
 **public void** createStream(StreamingActivity streamingActivity) {  
 **if** (client != **null** && client.getTranslationStatus() != -1) {  
 **return**;  
 }  
  
 createCameraPreview();  
 **if** (client == **null** || client.getTranslationStatus() == -1) {  
 client = **new** Client(streamingActivity, ip, port, cameraPreview);  
 }  
 createLocationManager();  
  
 **if** (ActivityCompat.checkSelfPermission(context, Manifest.permission.ACCESS\_FINE\_LOCATION) != PackageManager.PERMISSION\_GRANTED  
 && ActivityCompat.checkSelfPermission(context, Manifest.permission.ACCESS\_COARSE\_LOCATION) != PackageManager.PERMISSION\_GRANTED) {  
 *// ActivityCompat#requestPermissions  
 // here to request the missing permissions, and then overriding  
 // public void onRequestPermissionsResult(int requestCode, String[] permissions,  
 // int[] grantResults)  
 // to handle the case where the user grants the permission. See the documentation  
 // for ActivityCompat#requestPermissions for more details.* **return**;  
 }  
 locationManager.requestLocationUpdates(LocationManager.GPS\_PROVIDER, LOCATION\_REFRESH\_TIME, LOCATION\_REFRESH\_DISTANCE, client);  
 }  
 }  
 *// фильтры  
  
 // Options* **public void** offButton() {  
 runOnUiThread(**new** Runnable() {  
 @Override  
 **public void** run() {  
 btnStart.morph(morphingOperations.off);  
 streamStatus = **false**;  
 }  
 });  
 }  
 **private void** reset() {  
 streamStatus = **false**;  
 }  
 **public void** setPort(**int** port) {  
 **this**.port = port;  
 lbl\_port.setText(String.valueOf(port));  
 }  
 **public void** setIp(String ip) {  
 **this**.ip = ip;  
 lbl\_ip.setText(ip);  
 }  
 **private void** closeSocketClient() {  
 **if** (client == **null**)  
 **return**;  
  
*// client.interrupt();* client.close();  
*// try {  
// mThread.join();  
// } catch (InterruptedException e) {  
// e.printStackTrace();  
// }* client = **null**;  
 streamStatus = **false**;  
 }  
  
 *// Activity menu* @Override  
 **public boolean** onCreateOptionsMenu(Menu menu) {  
 *// Inflate the menu; this adds items to the action bar if it is present.* getMenuInflater().inflate(R.menu.ipcamera, menu);  
 **return true**;  
 }  
 @Override  
 **public boolean** onOptionsItemSelected(MenuItem item) {  
 **int** id = item.getItemId();  
 **switch** (id) {  
 **case** R.id.action\_greyscale:{  
 cameraPreview.setFilter(CameraPreview.Filters.GREYSCALE);  
 **break**;  
 }  
 **case** R.id.action\_sepia:{  
 cameraPreview.setFilter(CameraPreview.Filters.SEPIA);  
 **break**;  
 }  
 **case** R.id.action\_binary:{  
 cameraPreview.setFilter(CameraPreview.Filters.BINARY);  
 **break**;  
 }  
*// case R.id.action\_blur:{  
// cameraPreview.setFilter(CameraPreview.Filters.BLUR);  
// break;  
// }* **default**:{  
 cameraPreview.setFilter(CameraPreview.Filters.DEFAULT);  
 **break**;  
 }  
 }  
 **return super**.onOptionsItemSelected(item);  
 }  
}

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