Spot Saver Main Program

```java
public class MainActivity extends AppCompatActivity {
    private Button takePictureButton;
    private TextureView textureView;
    private static final SparseIntArray ORIENTATIONS = new SparseIntArray();

    static {
        ORIENTATIONS.append(Surface.ROTATION_0, 90);
        ORIENTATIONS.append(Surface.ROTATION_90, 0);
        ORIENTATIONS.append(Surface.ROTATION_180, 270);
        ORIENTATIONS.append(Surface.ROTATION_270, 180);
    }

    private String cameraId;
    protected CameraDevice cameraDevice;
    protected CameraCaptureSession cameraCaptureSessions;
    protected CaptureRequest captureRequest;
    protected CaptureRequest.Builder captureRequestBuilder;
    private Size imageDimension;
    private ImageReader imageReader;
    private File file;
    private static final int REQUEST_CAMERA_PERMISSION = 200;
    private boolean mFlashSupported;
    private Handler mBackgroundHandler;
    private HandlerThread mBackgroundThread;
    private Handler mHandler;
    private int mInterval = 4000;
    private String GOOD_TEXT = "Permit";
    private String lastReadString = "";
    private boolean prevNum = false;
    private boolean prevPermit = false;
    private int skipPicture = 0;
    private int SKIP_TIMES = 2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textureView = (TextureView) findViewById(R.id.textureView);
        assert textureView != null;
        textureView.setSurfaceTextureListener(textureListener);
//        takePictureButton = (Button) findViewById(R.id.btn_takepicture);
        assert takePictureButton != null;
        mHandler = new Handler();
        startRepeatingTask();
    }

    TextureView.SurfaceTextureListener textureListener = new
TextureView.SurfaceTextureListener() {
        @Override
        public void onSurfaceTextureAvailable(SurfaceTexture surface, int width, int
height) {
            //open your camera here
            openCamera();
        }

        @Override
        public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int width, int
height) {
            // Transform you image captured size according to the surface width and
height
        }

        @Override
        public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
            return false;
        }

        @Override
        public void onSurfaceTextureUpdated(SurfaceTexture surface) {
        }
    };
    private final CameraDevice.StateCallback stateCallback = new
CameraDevice.StateCallback() {
        @Override
        public void onOpened(CameraDevice camera) {
            //This is called when the camera is open
            cameraDevice = camera;
            createCameraPreview();
        }

        @Override
        public void onDisconnected(CameraDevice camera) {
            cameraDevice.close();
        }

        @Override
        public void onError(CameraDevice camera, int error) {
            toasty("Camera Error " + error);
            if (cameraDevice != null) {
                cameraDevice.close();
                cameraDevice = null;
            }
        }
    };
    final CameraCaptureSession.CaptureCallback captureCallbackListener = new
CameraCaptureSession.CaptureCallback() {
        @Override
        public void onCaptureCompleted(CameraCaptureSession session, CaptureRequest
request, TotalCaptureResult result) {
            super.onCaptureCompleted(session, request, result);
            createCameraPreview();
        }
    };

    protected void startBackgroundThread() {
        mBackgroundThread = new HandlerThread("Camera Background");
        mBackgroundThread.start();
        mBackgroundHandler = new Handler(mBackgroundThread.getLooper());
    }

    protected void stopBackgroundThread() {
        mBackgroundThread.quitSafely();
        try {
            mBackgroundThread.join();
            mBackgroundThread = null;
            mBackgroundHandler = null;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    protected void takePicture() {
        if (null == cameraDevice) {
//            toasty("cameraDevice is null");
            return;
        }
        CameraManager manager = (CameraManager)
getSystemService(Context.CAMERA_SERVICE);
        try {
            CameraCharacteristics characteristics =
manager.getCameraCharacteristics(cameraDevice.getId());
            Size[] jpegSizes = null;
            if (characteristics != null) {
                jpegSizes =
characteristics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP).getOutputSi
zes(ImageFormat.JPEG);
            }
            int width = 640;
            int height = 480;
            if (jpegSizes != null && 0 < jpegSizes.length) {
                width = jpegSizes[0].getWidth();
                height = jpegSizes[0].getHeight();
//                toasty("width = " + width + " height is " + height);
                width = 1280;
                height = 960;
            }
            ImageReader reader = ImageReader.newInstance(width, height,
ImageFormat.JPEG, 1);
            List<Surface> outputSurfaces = new ArrayList<Surface>(2);
            outputSurfaces.add(reader.getSurface());
            outputSurfaces.add(new Surface(textureView.getSurfaceTexture()));
            final CaptureRequest.Builder captureBuilder =
cameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_STILL_CAPTURE);
            captureBuilder.addTarget(reader.getSurface());
            captureBuilder.set(CaptureRequest.CONTROL_MODE,
CameraMetadata.CONTROL_MODE_AUTO);
            // Orientation
            int rotation = getWindowManager().getDefaultDisplay().getRotation();
            captureBuilder.set(CaptureRequest.JPEG_ORIENTATION,
ORIENTATIONS.get(rotation));
            final File file = new File(Environment.getExternalStorageDirectory() +
"/pic.jpg");
            ImageReader.OnImageAvailableListener readerListener = new
ImageReader.OnImageAvailableListener() {
                @Override
                public void onImageAvailable(ImageReader reader) {
                    Image image = null;
                    try {
                        image = reader.acquireLatestImage();
                        decodeImage(image);
//                        ByteBuffer buffer = image.getPlanes()[0].getBuffer();
//                        byte[] bytes = new byte[buffer.capacity()];
//                        buffer.get(bytes);
//                        save(bytes);
//                    } catch (FileNotFoundException e) {
//                        e.printStackTrace();
//                    } catch (IOException e) {
//                        e.printStackTrace();
                    } finally {
                        if (image != null) {
                            image.close();
                        }
                    }
                }

                private void save(byte[] bytes) throws IOException {
                    OutputStream output = null;
                    try {
                        output = new FileOutputStream(file);
                        output.write(bytes);
                    } finally {
                        if (null != output) {
                            output.close();
                        }
                    }
                }
            };
            reader.setOnImageAvailableListener(readerListener, mBackgroundHandler);
            final CameraCaptureSession.CaptureCallback captureListener = new
CameraCaptureSession.CaptureCallback() {
                @Override
                public void onCaptureCompleted(CameraCaptureSession session,
CaptureRequest request, TotalCaptureResult result) {
                    super.onCaptureCompleted(session, request, result);
//                    toasty( "Saved:" + file);
                    createCameraPreview();
                }
            };
            cameraDevice.createCaptureSession(outputSurfaces, new
CameraCaptureSession.StateCallback() {
                @Override
                public void onConfigured(CameraCaptureSession session) {
                    try {
                        session.capture(captureBuilder.build(), captureListener,
mBackgroundHandler);
                    } catch (CameraAccessException e) {
                        e.printStackTrace();
                    }
                }

                @Override
                public void onConfigureFailed(CameraCaptureSession session) {
                }
            }, mBackgroundHandler);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        }
    }

    protected void createCameraPreview() {
        try {
            SurfaceTexture texture = textureView.getSurfaceTexture();
            assert texture != null;
            texture.setDefaultBufferSize(imageDimension.getWidth(),
imageDimension.getHeight());
            Surface surface = new Surface(texture);
            captureRequestBuilder =
cameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
            captureRequestBuilder.addTarget(surface);
            cameraDevice.createCaptureSession(Arrays.asList(surface), new
CameraCaptureSession.StateCallback() {
                @Override
                public void onConfigured(@NonNull CameraCaptureSession
cameraCaptureSession) {
                    //The camera is already closed
                    if (null == cameraDevice) {
                        return;
                    }
                    // When the session is ready, we start displaying the preview.
                    cameraCaptureSessions = cameraCaptureSession;
                    updatePreview();
                }

                @Override
                public void onConfigureFailed(@NonNull CameraCaptureSession
cameraCaptureSession) {
                    toasty("Configuration change");
                }
            }, null);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        }
    }

    private void openCamera() {
        CameraManager manager = (CameraManager)
getSystemService(Context.CAMERA_SERVICE);
        try {
            cameraId = manager.getCameraIdList()[0];
            CameraCharacteristics characteristics =
manager.getCameraCharacteristics(cameraId);
            StreamConfigurationMap map =
characteristics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
            assert map != null;
            imageDimension = map.getOutputSizes(SurfaceTexture.class)[0];
            // Add permission for camera and let user grant the permission
            if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
!= PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions(MainActivity.this, new
String[]{Manifest.permission.CAMERA, Manifest.permission.WRITE_EXTERNAL_STORAGE},
REQUEST_CAMERA_PERMISSION);
                return;
            }
            manager.openCamera(cameraId, stateCallback, null);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        }
    }

    protected void updatePreview() {
        if (null == cameraDevice) {
            toasty("updatePreview error, return");
        }
        captureRequestBuilder.set(CaptureRequest.CONTROL_MODE,
CameraMetadata.CONTROL_MODE_AUTO);
        try {
            cameraCaptureSessions.setRepeatingRequest(captureRequestBuilder.build(),
null, mBackgroundHandler);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        }
    }

    private void closeCamera() {
        if (null != cameraDevice) {
            cameraDevice.close();
            cameraDevice = null;
        }
```

```java
        if (null != imageReader) {
            imageReader.close();
            imageReader = null;
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        if (requestCode == REQUEST_CAMERA_PERMISSION) {
            if (grantResults[0] == PackageManager.PERMISSION_DENIED) {
                // close the app
                toasty("Sorry, you can't use this app without granting permission");
                finish();
            }
        }
    }

    @Override
    protected void onResume() {
        super.onResume();
        startBackgroundThread();
        if (textureView.isAvailable()) {
            openCamera();
        } else {
            textureView.setSurfaceTextureListener(textureListener);
        }
    }

    @Override
    protected void onPause() {
        //closeCamera();
        stopBackgroundThread();
        super.onPause();
    }

    public void toasty(String text) {
        Context context = getApplicationContext();
        Toast.makeText(context, text, Toast.LENGTH_SHORT).show();
    }

    public void decodeImage(Image image) {
        Context context = getApplicationContext();
        ByteBuffer buffer = image.getPlanes()[0].getBuffer();
        byte[] bytes = new byte[buffer.capacity()];
        buffer.get(bytes);
        Bitmap bitmapImage = BitmapFactory.decodeByteArray(bytes, 0, bytes.length,
null);
//        Bitmap bitmap = BitmapFactory.decodeResource(context.getResources(),
R.drawable.permit);

        TextRecognizer textRecognizer = new TextRecognizer.Builder(context).build();

        if (!textRecognizer.isOperational()) {
            toasty("could not get the text");
        } else {
            Frame frame = new Frame.Builder().setBitmap(bitmapImage).build();
            SparseArray<TextBlock> items = textRecognizer.detect(frame);
            StringBuilder sb = new StringBuilder();

            for (int i = 0; i < items.size(); ++i) {
                TextBlock myItem = items.valueAt(i);
                sb.append(myItem.getValue());
                sb.append("\n");
            }
            lastReadString = sb.toString();
//            toasty(lastReadString);
        }

    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        stopRepeatingTask();
    }

    Runnable mStatusChecker = new Runnable() {
        @Override
        public void run() {
            try {
                updateStatus(); //this function can change value of mInterval.
            } finally {
                // 100% guarantee that this always happens, even if
                // your update method throws an exception
                mHandler.postDelayed(mStatusChecker, mInterval);
            }
        }
    };

    void startRepeatingTask() {
        mStatusChecker.run();
    }

    void stopRepeatingTask() {
        mHandler.removeCallbacks(mStatusChecker);
    }

    public void updateStatus() {
        if(skipPicture > 0) {
            --skipPicture;
            return;
        }
        takePicture();
        playClick();
//        toasty(lastReadString);
        boolean currNum = hasNumber(lastReadString);
        boolean currPermit = hasPermit(lastReadString);
        if (currPermit && prevPermit && prevNum && currNum){
            playGood();
            skipPicture = SKIP_TIMES;
            prevNum = false;
            prevPermit = false;
            return;
        }
        else if(prevNum && currNum){
            playBad();
            skipPicture = SKIP_TIMES;
            prevNum = false;
            prevPermit = false;
            return;
        }
        playClick();
        prevPermit = currPermit;
        prevNum = currNum;
    }
```

```java
    public boolean hasNumber(String text) {
        return text.contains("ACRX") && text.contains("401");
    }

    public boolean hasPermit(String text) {
        return text.contains(GOOD_TEXT);
    }

    public MediaPlayer mp1;

    public void playClick() {
        mp1 = MediaPlayer.create(MainActivity.this, R.raw.click);
        mp1.start();
    }

    public void playGood (){
        mp1 = MediaPlayer.create(MainActivity.this, R.raw.goodcar);
        mp1.start();
    }

    public void playBad (){
        mp1 = MediaPlayer.create(MainActivity.this, R.raw.jerrybadcar);
        mp1.start();
    }
}
```