МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Практическое занятие № 5/4ч.

Разработка мобильных компонент анализа безопасности информационно-аналитических систем

(наименование дисциплины (модуля) в соответствии с учебным планом)

Уровень: бакалавриат
(бакалавриат, магистратура, специалитет)

Форма обучения: очная
(очная, очно-заочная, заочная)

Направление(-я) подготовки: 10.05.04 Информационно-аналитические системы безопасности
(код(-ы) и наименование(-я))

Институт: комплексной безопасности и специального приборостроения ИКБСП
(полное и краткое наименование)

Кафедра: КБ-4 «Прикладные информационные технологии»
(полное и краткое наименование кафедры, реализующей дисциплину (модуль))

Используются в данной редакции с учебного года 2021/22
(учебный год цифрами)

Проверено и согласовано «____» _____20___г. _____ _____
(подпись директора Института/Филиала с расшифровкой)

Москва 2021 г.

ОГЛАВЛЕНИЕ

1 ОСНОВЫ ИСПОЛЬЗОВАНИЯ АППАРАТНЫХ ВОЗМОЖНОСТЕЙ МОБИЛЬНЫХ УСТРОЙСТВ

Наличие в современных телефонах электронных компасов, датчиков равновесия, яркости и близости позволяют реализовывать такие функции как дополненная реальность, ввод данных, основанный на перемещениях в пространстве и многое другое. Датчиковое оборудование делится на несколько категорий: движения, положения и окружающей среды. Доступ к датчикам осуществляется на основе использования класса SensorManager, ссылку на который возможно получить с помощью стандартного метода getSystemService:

```
SensorManager sensorManager =
                (SensorManager)getSystemService(Context.SENSOR_SERVICE);
```

Для того чтобы получить список датчиковой аппаратуры в смартфоне, следует использовать метод getSensorList объекта SensorManager:

```
List<Sensor> sensors =
                        sensorManager.getSensorList(Sensor.TYPE_ALL);
```

Полученный список будет включать все поддерживаемые датчики: как аппаратные, так и виртуальные. Более того, некоторые из них будут иметь различные независимые реализации, отличающиеся количеством потребляемой энергии, задержкой, рабочим диапазоном и точностью.

Также требуется реализация интерфейса *android.hardware.SensorListener*. Интерфейс реализован с помощью класса, который используется для ввода значений датчиков по мере их изменения в режиме реального времени. Приложение реализует данный интерфейс для мониторинга одного или нескольких имеющихся аппаратных датчиков.

Интерфейс включает в себя два необходимых метода:

1. Метод *onSensorChanged(int sensor, float values[])* вызывается каждый раз, когда изменяется показания датчика. Данный метод вызывается только для датчиков, контролируемых данным приложением. В число аргументов метода входит значение, которое указывает, что значение датчика изменилось, и массив значений с плавающей запятой, отражающих собственно значение датчика. Некоторые датчики выдают только одно значение данных, тогда как другие

предоставляют три значения с плавающей запятой. Датчики ориентации и акселерометра возвращают три значения данных.

2. Метод *onAccuracyChanged(int sensor,int accuracy)* вызывается при изменении точности показаний датчика. Аргументами служат два целых числа: одно указывает датчик, а другое соответствует новому значению точности этого датчика.

Для того, чтобы получать события, генерируемые датчиками, требуется зарегистрировать свою реализацию интерфейса SensorEventListener с помощью SensorManager, указать объект Sensor, за которым требуется наблюдать, и частоту, с которой необходимо получать обновления:

```
Sensor defPressureSensor =
            sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
sensorManager.registerListener(workingSensorEventListener,
            defPressureSensor, SensorManager.SENSOR_DELAY_NORMAL);
```

В первой строчке указывается, что будет использоваться датчик – барометр (*TYPE_PRESSURE*). Далее вызывается метод registerListener объекта класса SensorManager для установки параметров датчика.

В классе SensorManager определены четыре статические константы, определяющие частоту обновления:

– *SensorManager.SENSOR_DELAY_FASTEST* — максимальная частота обновления данных;

– *SensorManager.SENSOR_DELAY_GAME* — частота, обычно используемая в играх, поддерживающих гироскоп;

– *SensorManager.SENSOR_DELAY_NORMAL* — частота обновления по умолчанию;

– *SensorManager.SENSOR_DELAY_UI* — частота, подходящая для обновления пользовательского интерфейса.

В таблице 1.1 указаны типы датчиков и описание возвращаемых значений в метод onSensorChanged(int sensor, float values[]).

Таблица 1-1 Значения, возвращаемые датчиками

Тип датчика	Кол-во значений	Содержание значений	Примечание
TYPE_ACCELEROMETER	3	value[0]:ось X (поперечная) value[1]: ось Y (продольная) value[2]:ось Y (вертикальная)	Ускорение (м/с^2) по трём осям. Константы SensorManager.GRAVITY_*
TYPE_GRAVITY	3	value[0]:ось X (поперечная) value[1]: ось Y (продольная) value[2]:ось Y (вертикальная)	Сила тяжести (м/с^2) по трём осям. Константы SensorManager.GRAVITY_*
TYPE_RELATIVE_HUMIDITY	1	value[0]:относительная влажность	Относительная влажность в процентах (%)
TYPE_LINEAR_ACCELERATION	3	value[0]:ось X (поперечная) value[1]: ось Y (продольная) value[2]:ось Y (вертикальная)	Линейное ускорение (м/с^2) по трём осям без учёта силы тяжести
TYPE_GYROSCOPE	3	value[0]:ось X value[1]:ось Y value[2]:ось Z	Скорость вращения (рад/с) по трём осям
TYPE_ROTATION_VECTOR	4	values[0]:x*sin(q/2) values[1]:y*sin(q/2) values[2]:z*sin(q/2) values[3]:cos(q/2)	Положение устройства в пространстве. Описывается в виде угла поворота относительно оси в градусах
TYPE_MAGNETIC_FIELD	3	value[0]:ось X (поперечная) value[1]: ось Y (продольная) value[2]:ось Y (вертикальная)	Внешнее магнитное поле (мкТл)
TYPE_LIGHT	1	value[0]:освещённость	Внешняя освещённость (лк). Константы SensorManager.LIGHT_*
TYPE_PRESSURE	1	value[0]:атм.давление	Атмосферное давление (мбар)
TYPE_PROXIMITY	1	value[0]:расстояние	Расстояние до цели
TYPE_AMBIENT_TEMPERATURE	1	value[0]:температура	Температура воздухах в градусах по Цельсию
TYPE_POSE_6DOF	15	см. документацию	
TYPE_STATIONARY_DETECT	1	value[0]	5 секунд неподвижен
TYPE_MOTION_DETECT	1	value[0]	В движении за последние 5 секунд
TYPE_HEART_BEAT	1	value[0]	

1.1 Задание. Список датчиков.

Требуется создать новый проект *ru.mirea.«фамилия».practice5*. Отобразить существующую датчиковую аппаратуру на устройстве.

В activity_main требуется добавить в файл разметки ListView:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.an-
droid.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <ListView
        android:id="@+id/list_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Для получения доступа к списку сенсоров, присутствующих в смартфоне, следует использовать метод *getSensorList* объекта *SensorManager*:

```
SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

Полученный список будет включать все поддерживаемые датчики: как аппаратные, так и виртуальные (рисунок 1.1). Более того, некоторые из них будут иметь различные независимые реализации, отличающиеся количеством потребляемой энергии, задержкой, рабочим диапазоном и точностью.

Для получения списка всех доступных датчиков конкретного типа необходимо указать соответствующую константу. Например, код возвращает все доступные барометрические датчики. Причем аппаратные реализации окажутся в начале списка, а виртуальные — в конце (правило действует для всех типов датчиков):

```
List<Sensor> pressureList =
                    sensorManager.getSensorList(Sensor.TYPE_PRESSURE);
```

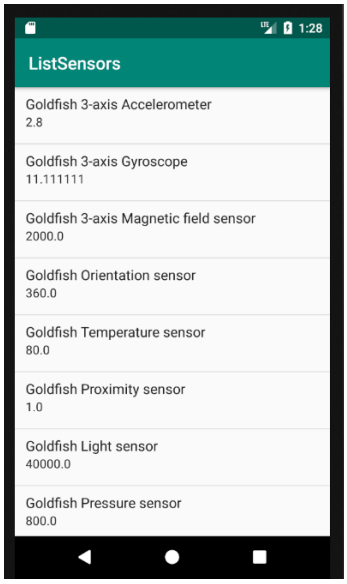


Рисунок 1.1– Список датчиков

Далее приведён код класса MainActivity:

```
public class MainActivity extends AppCompatActivity {
    private SensorManager sensorManager;
    private ListView listCountSensor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listCountSensor = findViewById(R.id.list_view);

        SensorManager sensorManager =
                        (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);

        // создаем список для отображения в ListView найденных датчиков
        ArrayList<HashMap<String, Object>> arrayList = new ArrayList<>();
        HashMap<String, Object> sensorTypeList;
        for (int i = 0; i < sensors.size(); i++) {
            sensorTypeList = new HashMap<>();
            sensorTypeList.put("Name", sensors.get(i).getName());
            sensorTypeList.put("Value", sensors.get(i).getMaximumRange());
            arrayList.add(sensorTypeList);
        }
        // создаем адаптер и устанавливаем тип адаптера – отображение двух полей
        SimpleAdapter mHistory =
                 new SimpleAdapter(this, arrayList, android.R.layout.simple_list_item_2,
                new String[]{"Name", "Value"},
                new int[]{android.R.id.text1, android.R.id.text2});
        listCountSensor.setAdapter(mHistory);
    }
}
```

1.2 Показания акселерометра

Практически любой современный телефон имеет акселерометр, позволяющий определить положение телефона относительно земли, а также ускорение в

пространстве по осям X, Y, Z.

Акселерометр используется для измерения ускорения. Его иногда называют датчиком силы притяжения.

Акселерометры часто выступают в качестве датчиков силы притяжения, так как они не могут определить, чем вызвано ускорение — движением или гравитацией. В результате этого в состоянии покоя акселерометр будет указывать на ускорение по оси Z (вверх/вниз), равное 9,8м/с2 (это значение доступно в виде константы *SensorManager.STANDARD_GRAVITY*).

Ускорение — это производная скорости по времени, поэтому акселерометр определяет, насколько быстро изменяется скорость устройства в заданном направлении. Используя этот датчик, имеется возможность обнаружить движение и, что более полезно, изменение его скорости. Акселерометр не измеряет скорость как таковую, поэтому вы не можете получить скорость движения, основываясь на единичном замере. Вместо этого необходимо учитывать изменения ускорения на протяжении какого-то отрезка времени.

Ускорение может быть измерено в трех направлениях: по оси ординат, по оси абсцисс, а также по вертикальной оси. Менеджер SensorManager сообщает об изменениях в показаниях акселерометра по всем трём направлениям.

Значения, переданные через свойство values объекта SensorEvent, отражают боковое, продольное и вертикальное ускорение (именно в таком порядке).

Рисунок 1.2 иллюстрирует нанесение трех направляющих осей на устройство, находящееся в состоянии покоя, которое в интерпретации SensorManager наступает тогда, когда устройство лежит на плоской поверхности экраном вверх и находится при этом в портретном режиме.

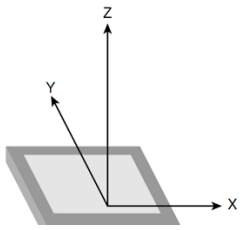


Рисунок 1.2 – Оси координат устройства

Ось X (ординаты). Боковое (влево или вправо) ускорение, положительные значения которого свидетельствуют о движении в направлении правой части устройства, а отрицательные — в направлении левой его части. Например, положительное ускорение по оси X будет, если устройство, лежа экраном вверх, повернется вправо (не отрывая крышку от поверхности).

Ось Y (абсциссы). Ускорение вперед или назад. В первом случае показатели будут больше нуля. Разместив устройство таким образом, как в предыдущем пункте, и подвинув в сторону его верхнюю часть, вы создадите положительное продольное ускорение.

Ось Z (аппликаты). Ускорение вверх или вниз. В первом случае при подъеме устройства значения будут положительными. В состоянии покоя вертикальное ускорение равно 9,8 м/с2 (вследствие силы тяжести).

Изменения ускорения отслеживаются посредством *SensorEventListener*. В классе требуется имплементировать интерфейс:

```
public class MainActivity extends AppCompatActivity
                                    implements SensorEventListener
```

После этого данная строка будет выделена красной чертой. Данное обозначение предупреждает, что данный класс не реализовал внутри тела класса методы, указанные в интерфейсе. Требуется навести курсор и с помощью сочетания клавиш *alt+enter(option+enter)* добавить требуемые методы (onSensorChanged(SensorEvent event) и onAccuracyChanged(Sensor sensor, int accuracy)).

Далее требуется зарегистрировать реализацию интерфейса с помощью SensorManager, используя объект Sensor с типом Sensor.TYPE_ACCELEROMETER, чтобы запрашивать обновления для акселерометра. В следующем листинге регистрируется акселерометр по умолчанию, используя стандартную частоту обновлений.

```
SensorManager sensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
accelerometerSensor = sensorManager
        .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener(this, accelerometerSensor,
        SensorManager.SENSOR_DELAY_NORMAL);
```

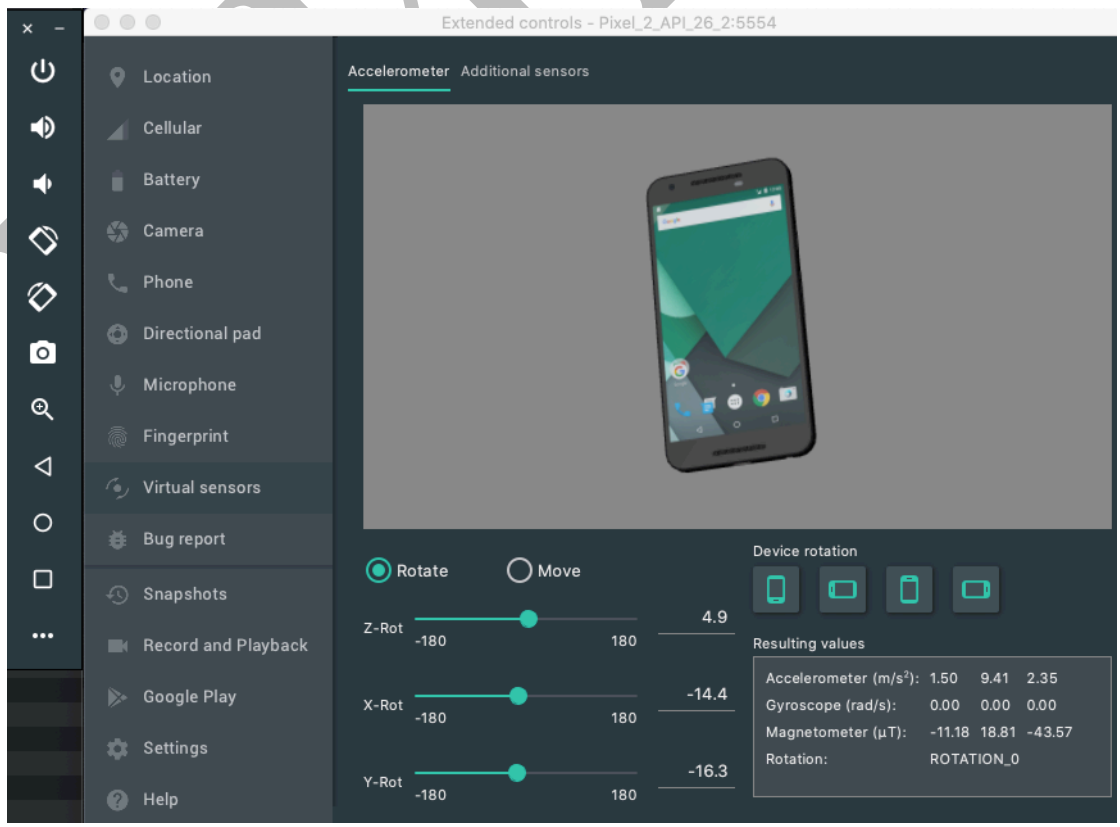
Интерфейс SensorEventListener содержит два обязательных метода. Метод onSensorChanged() срабатывает при измерении ускорения в произвольном направлении.

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        float valueAzimuth = event.values[0];
        float valuePitch = event.values[1];
        float valueRoll = event.values[2];
    }
}
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}
```

Метод onSensorChanged() получает объект SensorEvent, содержащий массив трёх значений типа float, представляющий собой показатели ускорения по всем трём осям. Основываясь на состоянии покоя, когда устройство лежит на задней крышке в портретном режиме, первый элемент означает боковое ускорение, второй — продольное, третий — вертикальное.

Эмулятор AVD имеет набор функций позволяющий изменять показания датчиковой аппаратуры для проведения тестирования приложения.



2 ЗАДАНИЕ.

Создать новый модуль. Название Accelerometer. Требуется создать приложение, отображающее значения акселерометра на главном экране. При вращении устройства значения должны изменяться на главном экране.

1. Требуется разместить 3 текстовых поля в activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.an-
droid.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textViewAzimuth"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textViewPitch"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textViewAzimuth" />

    <TextView
        android:id="@+id/textViewRoll"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textViewPitch" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

2. Создать класс, реализующий интерфейс SensorEventListener. В данном классе требуется: инициализировать датчик в методе onResume(), отменить

регистрацию в методе onPause() для освобождения ресурсов, и отследить изменения в методе onSensorChanged().

```
public class MainActivity extends AppCompatActivity implements SensorEventListener {

    private TextView azimuthTextView;
    private TextView pitchTextView;
    private TextView rollTextView;
    private SensorManager sensorManager;
    private Sensor accelerometerSensor;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        SensorManager sensorManager =
(SensorManager)getSystemService(Context.SENSOR_SERVICE);
        accelerometerSensor = sensorManager
                .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        azimuthTextView = findViewById(R.id.textViewAzimuth);
        pitchTextView = findViewById(R.id.textViewPitch);
        rollTextView = findViewById(R.id.textViewRoll);
    }
    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }

    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(this, accelerometerSensor,
                SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            float valueAzimuth = event.values[0];
            float valuePitch = event.values[1];
            float valueRoll = event.values[2];
            azimuthTextView.setText("Azimuth: " + valueAzimuth);
            pitchTextView.setText("Pitch: " + valuePitch);
            rollTextView.setText("Roll: " + valueRoll);
        }
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }
}
```

3 PERMISSION

Операционная система Android устроена таким образом, что для выполнения некоторых операций или доступа к определенным ресурсам, приложение должно иметь разрешение на это.

Основными двумя типами разрешений, используемых разработчиками приложений, являются *normal* и *dangerous*. Отличие между ними в том, что dangerous разрешения опасны, т.к. могут быть использованы для получения личных данных или информации о пользователе и т.д. Примеры dangerous разрешений - это доступ к контактам, смс или определения местоположения.

Полный список существующих разрешений возможно посмотреть https://developer.android.com/reference/android/Manifest.permission.html.

Характеристика Protection level подскажет насколько опасно это разрешение

Если приложению необходимо получить какое-либо разрешение, то оно должно быть указано в AndroidManifest.xml, в корневом теге <manifest>. Тег разрешения - <uses-permission>.

Далее приведен пример манифеста с разрешениями:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mirea.asd.camera">
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.SEND_SMS" />
```

В данном файле указывается, что приложению понадобятся разрешения на работу с интернет, контактами, bluetooth, локацией, камерой и смс. Пользователю необходимо будет подтвердить, что он предоставляет приложению эти разрешения. До выхода Android 6 пользователь при установке приложения отображался экран следующего вида (рисунок 1.3):

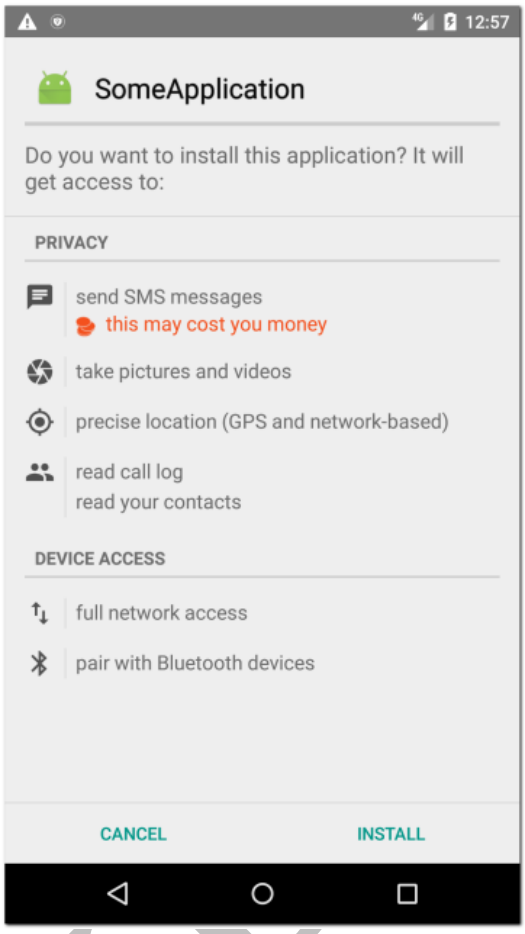


Рисунок 3.1 – Экран установки приложения до Android 6

Система показывает разрешения, которые были прописаны в манифесте. Сначала те, которые могут быть опасными с точки зрения приватности (отправка смс, доступ к камере/местоположению/контактам), а затем - обычные (интернет, bluetooth).

Нажав кнопку Install, пользователь автоматически подтверждает свое согласие, что приложению будут предоставлены данные запрашиваемые разрешения. И далее, когда приложение, например, пытается в коде получить список контактов, то оно без проблем их получает.

Если же в манифесте не указать разрешение READ_CONTACTS, то его не будет и в списке тех разрешений, которые подтверждает пользователь. Соответственно, система не предоставит этому приложению доступ к контактам. При запросе получения списка контактов отобразится ошибка:

java.lang.SecurityException: Permission Denial: opening provider com.android.providers.contacts.ContactsProvider2

С выходом Android 6 механизм подтверждения поменялся. При установке приложения пользователь больше не видит списка запрашиваемых разрешений. Приложение автоматически получает все требуемые normal разрешения, а dangerous разрешения необходимо программно запрашивать в процессе работы приложения.

Т.е. теперь недостаточно указать в манифесте, что требуется, например, доступ к контактам. Когда вы в коде будет производиться запрос списка контактов, то отобразится ошибка *SecurityException: Permission Denial* , потому что явно не был произведен запрос на данное разрешение.

Перед выполнением операции, требующей разрешения, необходимо спросить у системы, есть ли у приложения разрешение на это. Т.е. подтверждал ли пользователь, что он дает приложению данное разрешение. Если разрешение уже есть, то выполняется операция, если нет, то производится запрос разрешения у пользователя.

Проверка текущего статуса разрешения выполняется методом checkSelfPermission:

```
int permissionStatus =
                ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA);
```

Аргументами метода является Context и название разрешения. Данный метод возвращает константу *PackageManager.PERMISSION_GRANTED* (если разрешение есть) или *PackageManager.PERMISSION_DENIED* (если разрешения нет).

Если разрешения нет, то требуется его запросить. Это выполняется методом *requestPermissions*. Схема его работы похожа на метод startActivityForResult. Вызывается метод, передаются данные и request code, а ответ возвращается в определенном onResult методе.

```
if (permissionStatus == PackageManager.PERMISSION_GRANTED) {
    isWork = true;
} else {
    ActivityCompat.requestPermissions(this, new String[] {Manifest.permission.CAMERA},
            REQUEST_CODE_PERMISSION_CAMERA);
}
```

Производится проверка разрешения *CAMERA*. Если оно есть, то выполняется работа с камерой. Иначе запрашиваем разрешение *CAMERA* методом requestPermissions. На вход метод требует Activity, список требуемых разрешений, и

request code. Следует обратить внимание, что для разрешений используется массив.

После вызова метода requestPermissions система отобразит следующее диалоговое окно:

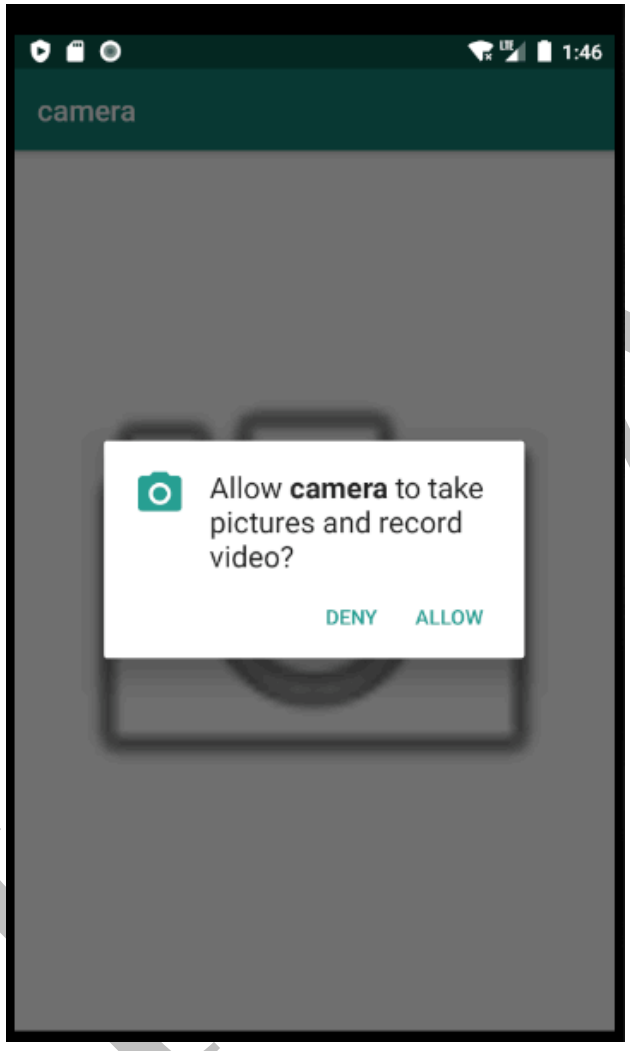


Рисунок 3.2 – Диалоговое окно запроса разрешений

Здесь отображается разрешение, которое запросили методом requestPermissions. Пользователь может либо подтвердить его (ALLOW), либо отказать (DENY). Если будет запрошено сразу несколько разрешений, то на каждое из них будет показан отдельный диалог. И пользователь может какие-то разрешения подтвердить, а какие-то нет.

Решение пользователя возвращается в методе onRequestPermissionsResult, который требуется переопределить в MainActivity.

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
    if (requestCode == REQUEST_CODE_PERMISSION_CAMERA) {
        if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // permission granted
            isWork = true;
        } else {
            isWork = false;
        }
    }
}
```

В первую очередь производится проверка, что requestCode тот же, что был указан в requestPermissions. В массиве *permissions[]* возвращаются названия разрешений, которые были запрошены. В массиве *grantResults[]* возвращаются решения пользователя на запросы разрешений.

Проверяется, что массив ответов не пустой и возвращается первый результат из него (т.к. был запрос одного разрешения). Если пользователь подтвердил разрешение, то выполняется операция. Если же пользователь отказал, то дальнейшие действия зависят от логики приложения.

В результате схема получения разрешения состоит из трех действий:

- проверка текущего состояния разрешения;
- запрос на получение разрешения, если оно еще не было получено;
- обработка ответа на запрос.

Требуется всегда проверять разрешение перед выполнением операции, требующей определенного разрешения. Иногда пользователь может разрешить доступ к данным при запуске приложения, но затем перейти в настройки приложения и отменить их. Если после этого не будет выполнена проверка, то отобразится ошибка при выполнении операции.

4 ЗАДАНИЕ. КАМЕРА

Создать новый модуль: Camera. Требуется запустить приложение «Камера», а полученную фотографию сохранить в папке и отобразить на экране. В activity_main требуется добавить image_view

Если в приложении необходимо произвести снимок или снять видео необязательно для этого создавать отдельное Activity и работать в нем с объектом Camera. Возможно использовать уже существующее в системе приложение.

Для этого приложение должно отправить Intent с *action* = *MediaStore.ACTION_IMAGE_CAPTURE* (фото) или *MediaStore.ACTION_VIDEO_CAPTURE* (видео) и ожидать ответ. Т.е. надо использовать методы startActivityForResult и onActivityResult.

Также, в этот Intent возможно поместить желаемый путь к файлу и туда будет сохранен результат работы камеры. Для этого в Intent необходимо добавить Uri с ключом *MediaStore.EXTRA_OUTPUT*.

Для получения изображения с камеры требуется использовать механизм обмена данными FileProvider. Он применяется специально для совместного использования внутренних файлов приложения. На рисунке 4.1 приведена схема взаимодействия приложения с приложением «Камера».

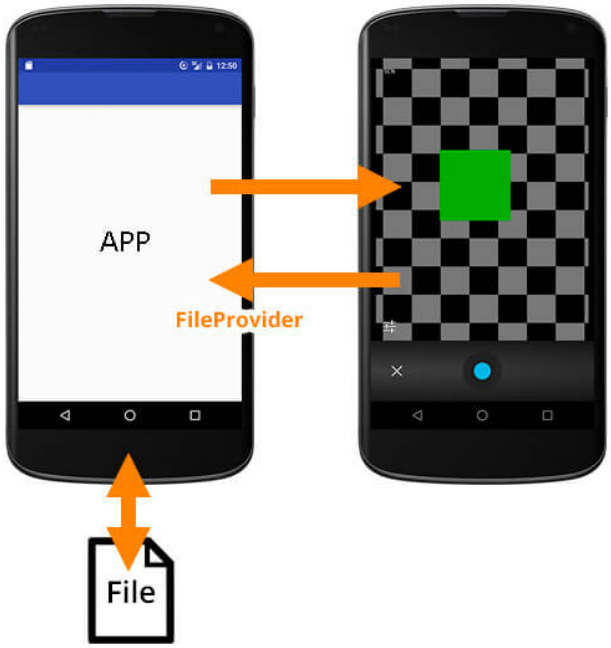


Рисунок 4.1 - Схема получения файлов с использованием FileProvider

Определение FileProvider для приложения требует наличия записи в манифесте. Данная запись определяет полномочия, которые будет использоваться при формировании URI контента, а также имя какого-либо XML файла, который определяет каталоги вашего приложения, используемые для общего доступа.

В манифест файле требуется добавить <provider> элемент, который указывает FileProvider класс, полномочия, и имя XML файла. Также в manifest добавлены необходимые для работы приложения разрешения.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mirea.asd.camera">
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    ...
    <provider
        android:name="androidx.core.content.FileProvider"
        android:authorities="com.mirea.asd.camera.fileprovider"
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data
            android:name="android.support.FILE_PROVIDER_PATHS"
            android:resource="@xml/paths" />
    </provider>
</application>
```

Метка *android:authorities* определяет полномочия URI, который используется для URI контента, сгенерированного FileProvider. Для приложения, требуется указать полномочия, состоящие из значения элемента *android:package* со строкой *«fileprovider»*, добавленной к нему.

Метка *<meta-data>* дочерний элемент <provider> указывает на XML файл, в котором определяются каталоги общего доступа.

Метка *android:resource* указывает путь и имя файла, без .xml расширения.

После добавления FileProvider в манифест приложения, необходимо указать каталоги, которые содержат файлы общего доступа. Для того, чтобы указать каталоги, требуется создать файл filepaths.xml в res/xml/ подкаталоге проекта. В следующем фрагменте приведено содержимое res/xml/filepaths.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <external-path
        name="mediaimages"
        path="." />
</paths>
```

Метка <external-path> обозначает элемент для каталогов общего доступа во внешнем хранилище. После этого имеется полная спецификация о FileProvider , который генерирует URI для контента файлов в external-path/. Когда приложение генерирует URI для контента файла, он содержит авторитетный источник, указанный в <provider> элементе (com.mirea.asd.fileprovider), путь mediaimages/, и имя файла.

Статическая константа *MediaStore.ACTION_IMAGE_CAPTURE* предназначена для создания намерения, которое потом требуется передать методу *startActivityForResult()*. Разместите на форме кнопку и ImageView, в который будем помещать полученный снимок.

```
Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
...
String authorities = getApplicationContext().getPackageName() + ".fileprovider";
imageUri = FileProvider.getUriForFile(this, authorities, photoFile);
cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
startActivityForResult(cameraIntent, CAMERA_REQUEST);
```

Этот код запускает приложение камеры. Полученное с камеры изображение возможно обработать в методе onActivityResult():

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    // Если приложение камера вернула RESULT_OK, то производится установка изображению
imageView
    if (requestCode == CAMERA_REQUEST && resultCode == RESULT_OK) {
        imageView.setImageURI(imageUri);
    }
}
```

Код приложения представлен ниже:

```
public class MainActivity extends AppCompatActivity {
    private static final int REQUEST_CODE_PERMISSION_CAMERA = 100;
    final String TAG = MainActivity.class.getSimpleName();
    private ImageView imageView;
    private static final int CAMERA_REQUEST = 0;
    private boolean isWork = false;
    private Uri imageUri;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imageView = findViewById(R.id.imageView);
        // Выполняется проверка на наличие разрешений на использование камеры и запись в
память
        int cameraPermissionStatus =
                ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA);
        int storagePermissionStatus = ContextCompat.checkSelfPermission(this, Mani-
fest.permission.WRITE_EXTERNAL_STORAGE);
        if (cameraPermissionStatus == PackageManager.PERMISSION_GRANTED && storagePermis-
sionStatus == PackageManager.PERMISSION_GRANTED) {
            isWork = true;
        } else {
            // Выполняется запрос к пользователь на получение необходимых разрешений
            ActivityCompat.requestPermissions(this, new String[] {Manifest.permis-
sion.CAMERA, Manifest.permission.WRITE_EXTERNAL_STORAGE},
                    REQUEST_CODE_PERMISSION_CAMERA);
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        // Если приложение камера вернула RESULT_OK, то производится установка изображе-
нию imageView
        if (requestCode == CAMERA_REQUEST && resultCode == RESULT_OK) {
            imageView.setImageURI(imageUri);
        }
    }

    public void imageViewOnClick(View view) {
        Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        // проверка на наличие разрешений для камеры
        if (cameraIntent.resolveActivity(getPackageManager()) != null && isWork == true)
{
            File photoFile = null;
            try {
                photoFile = createImageFile();
            } catch (IOException e) {
                e.printStackTrace();
            }
            // генерирование пути к файлу на основе authorities
            String authorities = getApplicationContext().getPackageName() + ".filepro-
vider";
            imageUri = FileProvider.getUriForFile(this, authorities, photoFile);
            cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
            startActivityForResult(cameraIntent, CAMERA_REQUEST);
        }
    }
```

```
  /**
     * Производится генерирование имени файла на основе текущего времени и создание файла
     * в директории Pictures на ExternelStorage.
     * class.
     * @return File  возвращается объект File .
     * @exception IOException если возвращается ошибка записи в файл
     */
    private File createImageFile() throws IOException {
        String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
        String imageFileName = "IMAGE_" + timeStamp + "_";
        File storageDirectory =
           Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
        return File.createTempFile(imageFileName, ".jpg", storageDirectory);
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permis-
sions, @NonNull int[] grantResults) {
      // производится проверка полученного результата от пользователя на запрос разреше-
ния Camera
        if (requestCode == REQUEST_CODE_PERMISSION_CAMERA) {
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                // permission granted
                isWork = true;
            } else {
                isWork = false;
            }
        }
    }
}
```

Данный код запустит приложение, работающее с камерой, позволяя пользователю поменять настройки изображения, что освобождает разработчиков от необходимости создавать своё собственное приложение для этих нужд. Вполне возможно, что у пользователя будет установлено несколько приложений, умеющих делать фотографии, тогда сначала появится окно выбора программы.

5 МИКРОФОН. MEDIARECORDER

Внимание!

Данное задание требуется тестировать на реальном устройстве. На сегодняшний день, стандартный эмулятор не имеет полноценной поддержки микрофона!

Создать новый модуль. Название AudioRecord.

Мультимедийный фреймворк Android поддерживает запись и воспроизведение аудио. Требуется разработать приложение для работы с диктафоном, которое будет записывать аудио и сохранять его в локальном хранилище Android-устройства с помощью MediaRecorder из Android SDK.

В файл activity_main.xml добавить три кнопки с идентификаторами для запуска, приостановки/возобновления и остановки записи.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.an-
droid.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/btnStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onRecordStart"
        android:text="@string/start"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.459" />

    <Button
        android:id="@+id/btnStop"
        android:layout_width="wrap_content"
        android:layout_height="48dp"
        android:layout_marginTop="16dp"
        android:onClick="onStopRecord"
        android:text="@string/stop"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btnStart" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

После создания пользовательского интерфейса возможно использовать MediaRecorder для реализации основной функциональности приложения. Но

сначала требуется запросить необходимые разрешения для записи аудио и доступа к локальному хранилищу. В файл AndroidManifest.xml необходимо добавить следующие разрешения:

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Также требуется проверить, одобрил ли пользователь разрешения, прежде чем использовать MediaRecorder. Добавим в MainActivity проверку разрешений:

```
public class MainActivity extends AppCompatActivity {
    private static final String TAG = MainActivity.class.getSimpleName();
    private static final int REQUEST_CODE_PERMISSION = 100;
    private String[] PERMISSIONS = {
            Manifest.permission.WRITE_EXTERNAL_STORAGE,
            Manifest.permission.RECORD_AUDIO
    };
    private boolean isWork;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
// проверка наличия разрешений на выполнение аудиозаписи и сохранения на карту памяти
        isWork = hasPermissions(this, PERMISSIONS);
        if (!isWork) {
            ActivityCompat.requestPermissions(this, PERMISSIONS,
                    REQUEST_CODE_PERMISSION);
        }
    }

    public static boolean hasPermissions(Context context, String... permissions) {
        if (context != null && permissions != null) {
            for (String permission : permissions) {
                if (ActivityCompat.checkSelfPermission(context, permission) != Pack-
ageManager.PERMISSION_GRANTED) {
                    return false;
                }
            }
        }
        return true;
    }
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permis-
sions, @NonNull int[] grantResults) {
        if (requestCode == REQUEST_CODE_PERMISSION) {
            // permission granted
            isWork = grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED;
        }
    }
}
```

Класс android.media.MediaRecorder позволяет делать записи медиафрагментов, включая аудио и видео. MediaRecorder действует как конечный автомат (рисунок 5.1). Задаются различные параметры, такие как устройство-

источник и формат. После установки запись может выполняться как угодно долго, пока не будет остановлена.

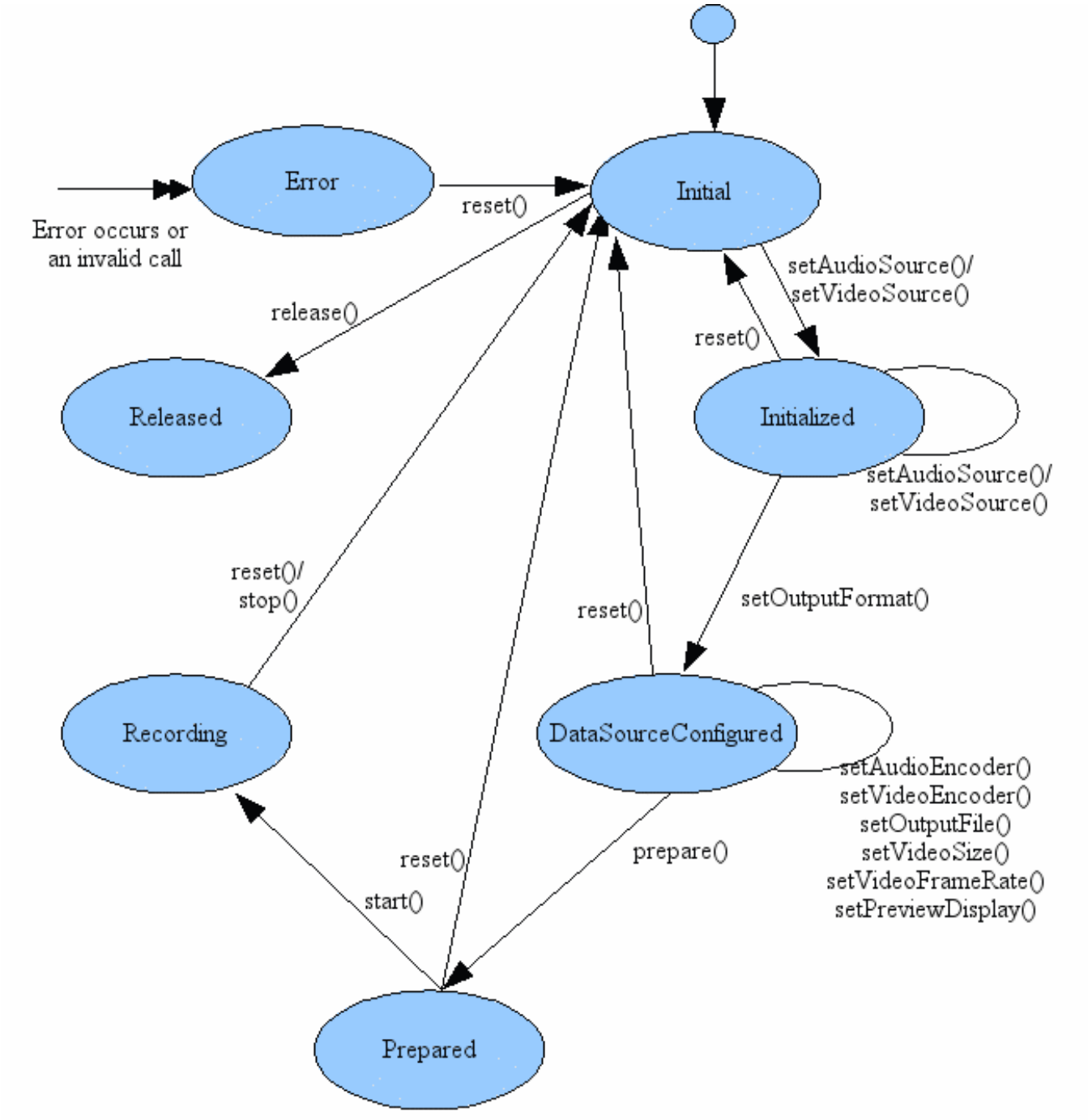


Рисунок 5.1 - Диаграмма состояний MediaRecorder

В созданный ранее класс требуется добавить методы создания и обработки состояний MediaRecorder:

```java
public class MainActivity extends AppCompatActivity {
    private static final String TAG = MainActivity.class.getSimpleName();
    private static final int REQUEST_CODE_PERMISSION = 100;

    private Button startRecordButton;
    private Button stopRecordButton;
    private MediaRecorder mediaRecorder;
    private File audioFile;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        startRecordButton = findViewById(R.id.btnStart);
        stopRecordButton = findViewById(R.id.btnStop);
        // инициализация объекта MediaRecorder
        mediaRecorder = new MediaRecorder();
...
    }

    // нажатие на кнопку старт
    public void onRecordStart(View view) {
        try {
            startRecordButton.setEnabled(false);
            stopRecordButton.setEnabled(true);
            stopRecordButton.requestFocus();
            startRecording();
        } catch (Exception e) {
            Log.e(TAG, "Caught io exception " + e.getMessage());
        }
    }

    // нажатие на копку стоп
    public void onStopRecord(View view) {
        startRecordButton.setEnabled(true);
        stopRecordButton.setEnabled(false);
        startRecordButton.requestFocus();
        stopRecording();
        processAudioFile();
    }

    private void startRecording() throws IOException {
        // проверка доступности sd - карты
        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state) ||
                Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
            Log.d(TAG, "sd-card success");
            // выбор источника звука
            mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
            // выбор формата данных
            mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
            // выбор кодека
            mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
            if (audioFile == null) {
                // создание файла
                audioFile = new File(this.getExternalFilesDir(
                        Environment.DIRECTORY_MUSIC), "mirea.3gp");
            }
            mediaRecorder.setOutputFile(audioFile.getAbsolutePath());
            mediaRecorder.prepare();
            mediaRecorder.start();
            Toast.makeText(this, "Recording started!", Toast.LENGTH_SHORT).show();
        }
    }
```

```
    private void stopRecording() {
        if (mediaRecorder != null) {
            Log.d(TAG, "stopRecording");
            mediaRecorder.stop();
            mediaRecorder.reset();
            mediaRecorder.release();
            Toast.makeText(this, "You are not recording right now!",
                                                    Toast.LENGTH_SHORT).show();

        }
    }
    private void processAudioFile() {
        Log.d(TAG, "processAudioFile");
        ContentValues values = new ContentValues(4);
        long current = System.currentTimeMillis();
        // установка meta данных созданному файлу
        values.put(MediaStore.Audio.Media.TITLE, "audio" + audioFile.getName());
        values.put(MediaStore.Audio.Media.DATE_ADDED, (int) (current / 1000));
        values.put(MediaStore.Audio.Media.MIME_TYPE, "audio/3gpp");
        values.put(MediaStore.Audio.Media.DATA, audioFile.getAbsolutePath());
        ContentResolver contentResolver = getContentResolver();
        Log.d(TAG, "audioFile: " + audioFile.canRead());

        Uri baseUri = MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
        Uri newUri = contentResolver.insert(baseUri, values);
        // оповещение системы о новом файле
        sendBroadcast(new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE, newUri));
    }
}
```

Алгоритм записи:

1. Метод processAudioFile. В первую очередь требуется определить источник (свойство MediaRecorder.AudioSource):

– MIC - встроенный микрофон;

– VOICE_UPLINK - исходящий голосовой поток при телефонном звонке;

– VOICE_DOWNLINK - входящий голосовой поток при телефонном ;

– VOICE_CALL - запись телефонного звонка;

– CAMCORDER - микрофон, связанный с камерой (если доступен);

– VOICE_RECOGNITION - микрофон, используемый для распознавания голоса (если доступен);

– VOICE_COMMUNICATION – аудио поток с микрофона будет "заточен" под VoIP (если доступен).

Если указанный источник не поддерживается текущим устройством, то будет использован микрофон по умолчанию.

2. Далее определяется формат записываемого звука (свойство MediaRecorder.OutputFormat):

– THREE_GPP - формат 3GPP;

– MPEG_4 - формат MPEG4;

– AMR_NB - формат AMR_NB (подходит для записи речи);

– AMR_WB;

– RAW_AMR;

– Определяется тип сжатия звука (свойство MediaRecorder.AudioEncoder);

– AAC;

– AMR_NB;

– AMR_WB;

3. Указывается путь к файлу, в котором будут сохранены аудиоданные (метод setOutputFile())

В методе startRecording() создается и инициализируется экземпляр MediaRecorder. В качестве источника данных выбирается микрофон (MIC). Выходной формат устанавливается в 3GPP (файлы *.3gp) - медиа формат, ориентированный на мобильные устройства. Кодек настроен на AMR_NB - аудиоформат с частотой дискретизации 8 кГц. NB означает узкую полосу частот.

4. Аудиофайл сохраняется на внешней карте. Затем этот файл связывается с экземпляром MediaRecorder, обращаясь к методу setOutputFile. Аудиоданные будут храниться в этом файле.

5. Вызов метода prepare() завершает инициализацию MediaRecorder. Когда нужно начать процесс записи, вызывается метод start().

6. Запись в файл на карте памяти ведется до тех пор, пока не будет вызван метод stop(), который освобождает ресурсы, выделенные экземпляру MediaRecorder.

7. Когда аудиофрагмент записан, возможно выполнить несколько действий:

– добавить аудиозапись в медиатеку на устройстве;

– выполнить шаги по распознаванию звука;

– автоматически загрузить звуковой файл в сетевую папку для обработки.

В этом примере метод processAudioFile() добавляет аудиозапись в медиатеку. Для уведомления встроенного приложения о том, что доступна новая информация,

используется Intent.

6 КОНТРОЛЬНОЕ ЗАДАНИЕ.

В контрольном задании MireaProject, построенном на базе NavigationDrawer, добавить к ранее созданным фрагментам экран аппаратной части со следующим функционалом:

– отображение значений 3 любых датчиков с изменением показаний;

– создание фотографии с отображением результата и последующим сохранением на карту памяти;

– выполнение аудиозаписи и возможность прослушивания данного файла через Service (аналогично заданию с музыкальным файлом);

– добавить в приложение механизмы запроса разрешений.