МИНОБРНАУКИ РОССИИ

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

РТУ МИРЭА

Практическое занятие № 5/4ч.

Разработка мобильных приложений

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

Уровень: бакалавриат
(бакалавриат, магистратура, специалитет)

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

Направление(-я) подготовки: 09.03.02 «Информационные системы и технологии»
(код(-ы) и наименование(-я))

Институт: кибербезопасности и цифровых технологий
(полное и краткое наименование)

Кафедра: КБ-14 «Цифровые технологии обработки данных»
(полное и краткое наименование кафедры, реализующей дисциплину (модуль))

Используются в данной редакции с учебного года 2024/25
(учебный год цифрами)

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

Москва 2025 г.

ОГЛАВЛЕНИЕ

1 ОСНОВЫ ИСПОЛЬЗОВАНИЯ АППАРАТНЫХ ВОЗМОЖНОСТЕЙ МОБИЛЬНЫХ УСТРОЙСТВ

Наличие в современных телефонах электронных компасов, датчиков равновесия, яркости и близости позволяет реализовывать функционал работы с дополненной реальностью, перемещением в пространстве и многим другим. Датчиковое оборудование классифицируется на несколько категорий: движения, положения и окружающей среды. Доступ к датчикам осуществляется на основе использования класса «*SensorManager*», ссылку на который возможно получить с помощью вызова 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 включает как аппаратные, так и виртуальные поддерживаемые датчики. Некоторые из них могут иметь различные независимые реализации, отличающиеся количеством потребляемой энергии, задержкой, рабочим диапазоном и точностью.

Также требуется реализация интерфейса «*android.hardware. SensorEventListener»*. Интерфейс реализован с помощью класса, который используется для ввода значений датчиков по мере их изменения в режиме реального времени. Интерфейс включает в себя два необходимых метода:

1. Метод *onSensorChanged(SensorEvent event)* вызывается каждый раз, когда сенсор (например, акселерометр, гироскоп и т.п.) сообщает об изменении своих данных. Параметр *SensorEvent* содержит: *event.sensor* — объект Sensor, содержащий информацию о типе сенсора, *event.values* — массив float[] с текущими значениями сенсора, *event.timestamp* — время события в наносекундах, *event.accuracy* — точность значений (не для всех сенсоров)

2. Метод «*onAccuracyChanged(Sensor sensor, int accuracy)*» вспомогательный метод, который вызывается, когда изменяется точность данных сенсора. Аргументами метода являются: объект Sensor, для которого изменилась точность, а другое соответствует новому значению точности показаний.

Для получения событий, генерируемых датчиками, требуется реализовать интерфейс «*SensorEventListener*» с помощью созданного объекта класса «SensorManager», указать объект «*Sensor*», за которым требуется выполнять наблюдение, и частоту показаний, с которой необходимо получать обновления.

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

В первой строчке примера кода определён тип сенсора – барометр («*TYPE_PRESSURE*»). Далее вызывается метод «*registerListener*» объекта класса «*SensorManager*» для установки параметров датчика.

В классе «*SensorManager*» определены четыре статические константы, указывающие 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 Описание возвращаемых значений датчикового оборудования

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».Lesson5*. Отобразить в виде списка датчиковое оборудование 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 на мобильном устройстве, следует использовать метод «*getSensorList*» объекта «*SensorManager*»:

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

Для получения списка доступных датчиков конкретного типа необходимо указать соответствующую константу. Например, для получения списка доступных барометрических датчиков используется константа «*Sensor.TYPE_PRESSURE*».

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

Стоит отметить, что аппаратные реализации всегда размещаются в начале списка, а виртуальные замыкают список.

Далее приведён код класса «*MainActivity*», предназначенный для получения списка датчиков и отображении их в компоненте список:

```
private ActivityMainBinding binding;
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  binding = ActivityMainBinding.inflate(getLayoutInflater());
  setContentView(binding.getRoot());
  SensorManager sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
  List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
  ListView listSensor = binding.sensorListView;

  // создаем список для отображения в ListView найденных датчиков
  ArrayList<HashMap<String, Object>> arrayList = new ArrayList<>();
  for (int i = 0; i < sensors.size(); i++) {
    HashMap<String, Object>  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});
  listSensor.setAdapter(mHistory);
```

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.

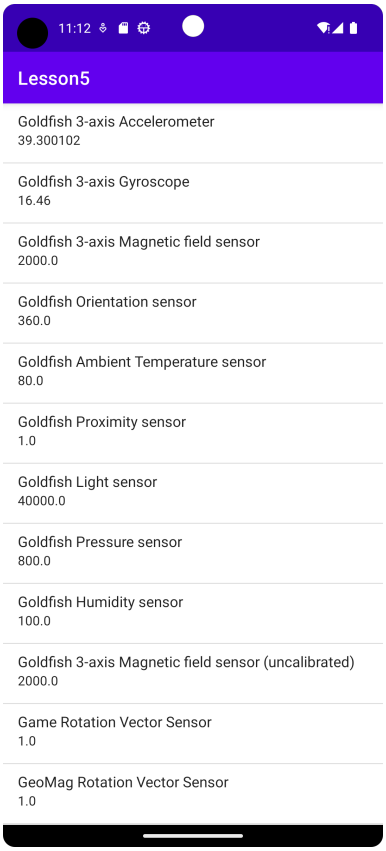

Рисунок 1.1– Список датчиков

1.2 Показания акселерометра

Современные мобильные устройства обладают возможностью получения данных с акселерометра, позволяющего определять 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 зачастую используют 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 является производной скорости по времени, поэтому акселерометр определяет, насколько быстро изменяется скорость устройства в заданном направлении. Датчик позволяет обнаружить движение и изменение его скорости, поэтому отсутствует возможность определения скорости движения, основываясь на единичном замере. Вместо этого требуется учитывать изменения ускорения на протяжении отрезка времени. С помощью акселерометра возможно измерение ускорения 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 аргумент «*values»* объекта «*SensorEvent*», отражают боковое, продольное и вертикальное ускорение.

Рисунок 1.2 отображает три направляющих оси устройства, находящегося в состоянии покоя. Для «*SensorManager*» состояние покоя читается тогда, когда устройство лежит на плоской поверхности экраном вверх и находится при этом в портретном режиме.

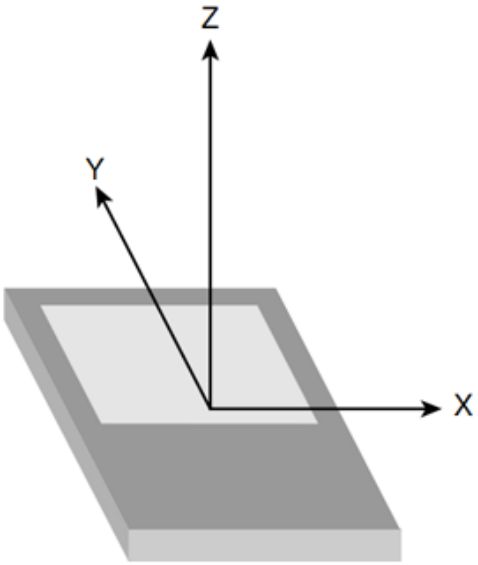

Рисунок 1.2 – Оси координат устройства

По оси X (ординаты) определяется боковое (влево или вправо) ускорение, положительные значения которого свидетельствуют о движении в направлении правой, а отрицательные – в направлении левой части устройства. Например, положительное ускорение по оси X будет, если устройство, лежа экраном вверх, повернется вправо (не отрывая крышку от поверхности). По оси Y (абсциссы) выявляется ускорение вперед или назад, т.е. переместив по направлению вперед создается положительное продольное ускорение. По оси Z (аппликаты) формируется ускорение вверх или вниз, т.е. при подъеме устройства значения будут положительными. В состоянии покоя вертикальное ускорение равно 9,8 м/с2 (вследствие силы тяжести).

Изменения ускорения отслеживаются посредством «*SensorEventListener»,* реализованном в классе наблюдателя за показаниями:

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

После добавления интерфейса 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*» и «*onAccuracyChanged*»). Далее требуется произвести

регистрацию 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) getSystemService(SENSOR_SERVICE);
accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener(this, accelerometer, 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 x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];

        // Пример: Вывод в лог
        Log.d(MainActivity.class.getSimpleName(), "Accelerometer: x=" + x + " y=" + y + " z=" + z);
        // Можно, например, определить, движется ли телефон
    }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    if (sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        switch (accuracy) {
            case SensorManager.SENSOR_STATUS_UNRELIABLE:
                Log.d("Sensor", "Акселерометр: ненадёжные данные");
                break;
            case SensorManager.SENSOR_STATUS_ACCURACY_HIGH:
                Log.d("Sensor", "Акселерометр: высокая точность");
                break;
        }
    }
}
```

Метод «*onSensorChanged*» получает объект «*SensorEvent*», содержащий массив трёх значений типа «*float*», представляющий собой показатели ускорения по всем трём осям. Первый элемент означает боковое ускорение, второй — продольное, третий — вертикальное. Эмулятор «AVD» имеет набор функций, позволяющий изменять показания датчиковой аппаратуры для проведения тестирования приложения.

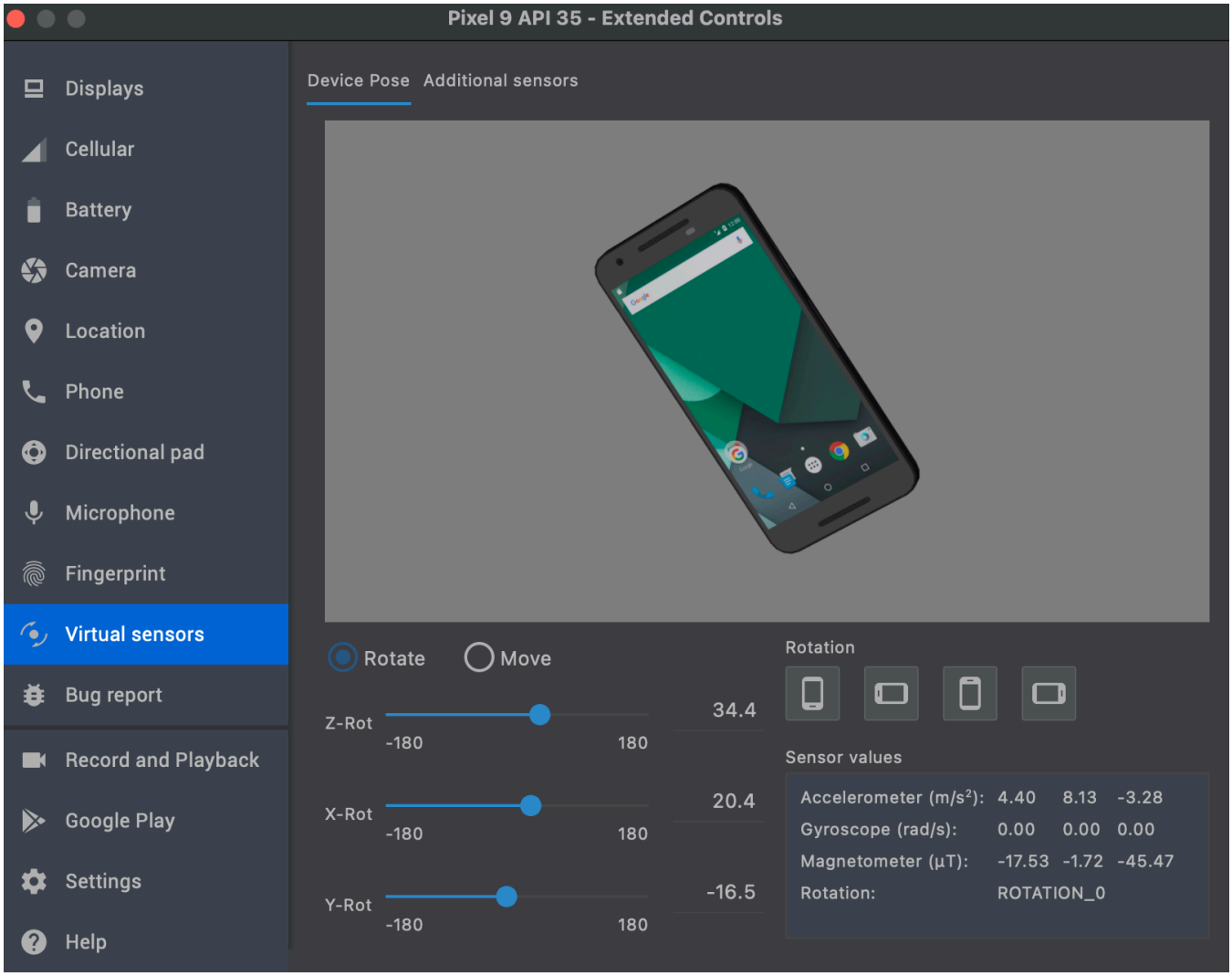

Рисунок 1.3 – Меню настроек эмулятора AVD

2 ЗАДАНИЕ.

Создать новый модуль. В меню «*File* | *New* | *New Module* | *Phone & Tablet Module* | *Empty Activity*». Название модуля «*Accelerometer*». Требуется создать приложение, отображающее значения акселерометра на главном экране. При вращении устройства значения должны изменяться на главном экране.

В первую очередь требуется реализовать экран с 3 текстовыми полями в «*activity_main.xml*».

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    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>
```

Требуется реализовать интерфейс «*SensorEventListener*», а также создать объект класса «*Sensor*». Инициализация датчиков производится в методе «*onResume*», отмена регистрации для освобождения ресурсов в методе «*onPause*» и отслеживание изменений выполняется в методе «*onSensorChanged*».

```
public class MainActivity extends AppCompatActivity  implements SensorEventListener {
   private SensorManager sensorManager;
   private Sensor accelerometer;
   private TextView azimuthTextView;
   private TextView pitchTextView;
   private TextView rollTextView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      EdgeToEdge.enable(this);
      setContentView(R.layout.activity_main);
      ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
         Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
         v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
         return insets;
      });
      sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
      accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
      sensorManager.registerListener(this, accelerometer, SensorManager.SENSOR_DELAY_NORMAL);
      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, accelerometer,
            SensorManager.SENSOR_DELAY_NORMAL);
   }
   @Override
   public void onSensorChanged(SensorEvent event) {
      if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
         float x = event.values[0];
         float y = event.values[1];
         float z = event.values[2];
         azimuthTextView.setText(String.format("Azimuth: %s", x));
         pitchTextView.setText(String.format("Pitch: %s", y));
         rollTextView.setText(String.format("Roll: %s", z)); телефон
      }
   }

   @Override
   public void onAccuracyChanged(Sensor sensor, int accuracy) {
      ...
   }
}
```

3 МЕХАНИЗМ РАЗРЕШЕНИЙ

Операционная система Android для выполнения некоторых операций или доступа к определенным ресурсам устройства проверяет наличие у приложений требуемых разрешений. Основными двумя типами разрешений, используемых разработчиками приложений, являются «*normal*» и «*dangerous*». Отличие между ними заключается в том, что «*dangerous*» разрешения являются опасными, т.к. могут быть использованы для получения личных данных или информации о пользователе. Например, «опасные» разрешения закрывают доступ приложений к контактной книге, смс-сообщениям или определению местоположения устройства. Полный список существующих разрешений представлен на странице https://developer.android.com/reference/android/Manifest.permission.html.

В случаях, если приложению требуется получить какое-либо разрешение, то оно должно быть указано в «*AndroidManifest.xml*», в корневом теге <*manifest*>. Тег разрешения - <*uses-permission*>. Далее приведен пример манифест-файла с разрешениями:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools">

  <uses-feature
    android:name="android.hardware.camera"
    android:required="false" />
  <uses-feature
    android:name="android.hardware.telephony"
    android:required="false" />

  <uses-permission android:name="android.permission.CAMERA" />
  <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE"
    tools:ignore="ScopedStorage" />
  // Проверка – удали меня из кода
  <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.ACCESS_COARSE_LOCATION" />

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

В данном файле указывается, что приложению понадобятся разрешения на работу с интернетом, контактами, «*bluetooth*», локацией, камерой и смс. Пользователю необходимо будет подтвердить, что он предоставляет приложению запрашиваемые разрешения. До выхода ОС «*Android*» 6 пользователь при установке

приложения отображался экран запрос разрешений, представленный на рисунке 3.1.

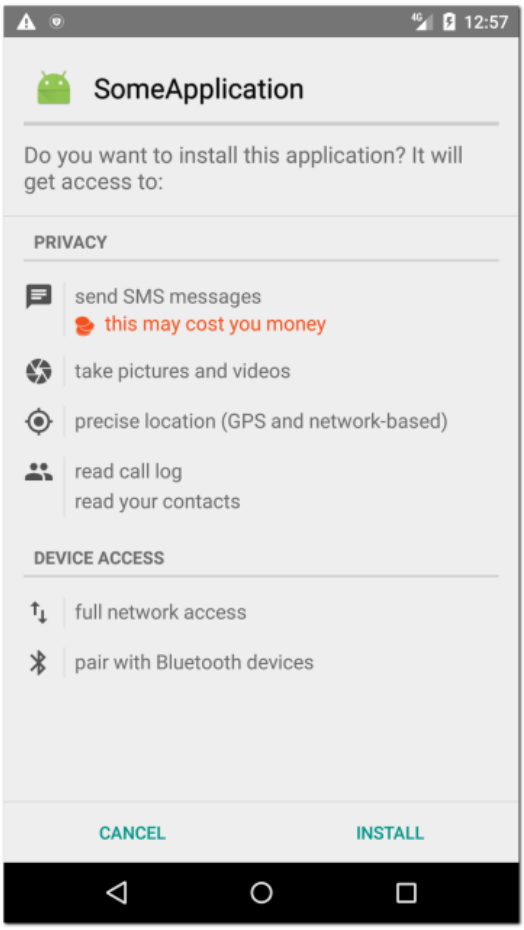

Рисунок 3.1 – Экран установки приложения до Android 6

Система отображает разрешения, которые были указаны в манифест-файле приложения. В первую очередь отображались наиболее опасные с точки зрения приватности (отправка смс, доступ к камере/местоположению/контактам), а затем - обычные (интернет, «*bluetooth*»). После нажатия на кнопку «*Install*», пользователь автоматически подтверждает свое согласие всех требований.

Если же в манифесте не указано разрешение «*READ_CONTACTS*», то его не будет и в списке запрашиваемых разрешений, которые подтверждает пользователь. Следовательно, система не предоставит данному приложению доступ к контактам. При запросе получения списка контактов отобразится ошибка: «***java.lang.SecurityException: Permission Denial: opening provider com.android.providers.contacts.ContactsProvider***»

С выходом ОС «*Android*» 6 механизм подтверждения поменялся. При установке приложения пользователю больше не отображается список

запрашиваемых разрешений. Приложение автоматически получает все требуемые разрешения категории «*normal*», а доступ к ресурсам, закрытыми разрешениями «*dangerous*», обеспечивается запросом к пользователю в процессе выполнения приложения. В случае отсутствия программного запроса, например, списка контактов, отобразится ошибка «*SecurityException: Permission Denial*». Перед осуществлением доступа к ресурсам устройства, выполняется запрос к операционной системе на определение наличия разрешения, т.е. согласие пользователя. Проверка текущего статуса разрешения выполняется методом «*checkSelfPermission*»:

```
int cameraPermissionStatus = ContextCompat.checkSelfPermission(this, android.Manifest.permission.CAMERA);
int storagePermissionStatus = ContextCompat.checkSelfPermission(this, android.Manifest.permission.
    WRITE_EXTERNAL_STORAGE);
```

Аргументами метода является контекст активности и идентификатор разрешения. Данный метод возвращает константу «*PackageManager.PERMISSION_GRANTED*» (имеется разрешение) или «*PackageManager.PERMISSION_DENIED*» (разрешение отсутствует).

В случае имеющегося разрешение, операция выполняется, в противном случае производится запрос разрешения у пользователя с помощью вызова метода «*requestPermissions*».

```
if (cameraPermissionStatus == PackageManager.PERMISSION_GRANTED && storagePermissionStatus
    == PackageManager.PERMISSION_GRANTED) {
  isWork = true;
} else {
  // Выполняется запрос к пользователь на получение необходимых разрешений
  ActivityCompat.requestPermissions(this, new String[] {android.Manifest.permission.CAMERA,
      android.Manifest.permission.WRITE_EXTERNAL_STORAGE}, REQUEST_CODE_PERMISSION);
}
```

Для осуществления запроса требуется указать в качестве аргументов метода контекст активности, массив идентификаторов разрешений и код запроса. После вызова метода «*requestPermissions*», на примере разрешения «*CAMERA*», система отобразит диалоговое окно, представленное на рисунке 3.2.

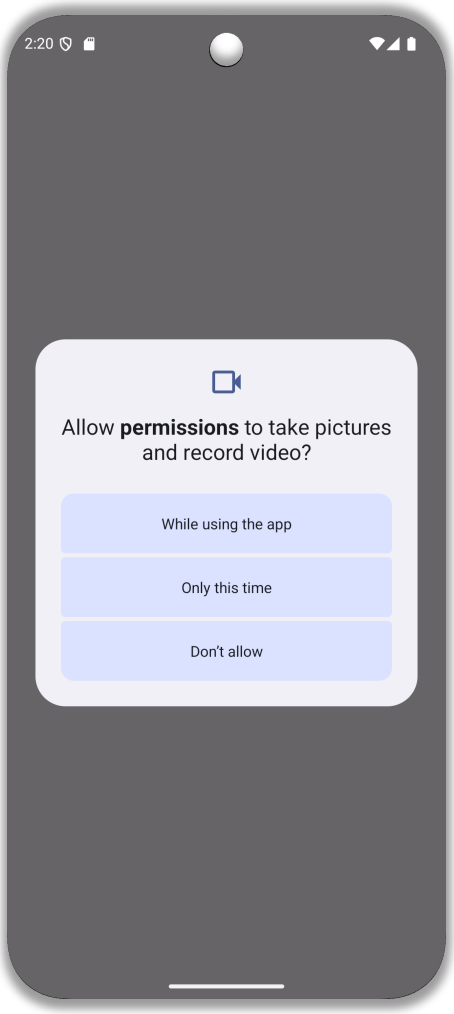

Рисунок 3.2 – Диалоговое окно запроса разрешений

Пользователь может подтвердить запрос на время, когда используется приложение («*While using the app*»), только один раз («*Only this time*»), либо отказать («*Don't allow*»). В случае, когда запрашивается несколько разрешений, для каждого из них будет отображен отдельный диалог, и пользователь имеет возможность производить выборочное разрешение. Обработка результата взаимодействия с пользователем осуществляется в методе «*onRequestPermissionsResult*», который требуется переопределить в активности.

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grant-
Results) {
  super.onRequestPermissionsResult(requestCode, permissions, grantResults);

  if (requestCode == REQUEST_CODE_PERMISSION) {
    if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
      // Разрешение получено — можно получать локацию
      isWork = true;
    } else {
      // Разрешение отклонено — нужно показать объяснение или отключить функцию
    }
  }
}
```

В первую очередь выполняется проверка «*requestCode*», который был указан при вызове метода «*requestPermissions*». В массиве «*permissions*» возвращаются названия запрошенных разрешений, а «*grantResults*» содержит массив решений пользователя на отображенные запросы. Выполняется проверка массива решений на длину и значение первого элемента (пример, с камерой).

Алгоритм получения пользовательского разрешения состоит из трех этапов:

– 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. Пользователь может разрешить доступ к данным при запуске приложения разово, либо затем отозвать его через меню настроек.

4 ЗАДАНИЕ. КАМЕРА

Создать новый модуль. В меню «*File | New | New Module | Phone & Tablet Module | Empty Activity*». Имя модуля «*Camera*». Возможности приложения: вызов системного приложения «камера», сохранение изображения в папку приложения и отображение снимка на экране. В «*activity_main*» требуется добавить «*image_view»* и реализовать обработчик нажатия с помощью *«ViewBinding».*

В случаях, когда в приложении необходимо реализовать функционал камеры необязательно для этого создавать отдельное «*Activity*» и работать в нем с объектом «*Camera*». Возможно использовать уже существующие в системе приложения.

Вызов приложений с функциями камеры осуществляется с помощью намерения со значением действия *«MediaStore.ACTION_IMAGE_CAPTURE»* (фото) или «*MediaStore.ACTION_VIDEO_CAPTURE»* (видео). Для формирования неявного намерения используется метод «*registerForActivityResult*», а получения результата «*onActivityResult*». Дополнительно, в намерении размещается путь для сохранения результата в виде «*Uri*» с ключом «*MediaStore.EXTRA_OUTPUT»*.

С целью безопасной передачи файлов из одного приложения другому приложению, необходимо использовать безопасный дескриптор в виде «URI» содержимого. «*ContentProvider*» является компонентом ОС «*Android*», который инкапсулирует данные и предоставляет их другим приложениям. Например, данные контактов совместно используются другими приложениями, используя «*ContactsProvider*», который является подклассом «*ContentProvider*». «*FileProvider*» также является подклассом «*ContentProvider*» и используется специально для совместного использования внутренних файлов приложения. В «*FileProvider*» компонент генерирует «URI» для содержимого файлов, основанные на спецификации, которая предоставляется в XML. Для реализации «*FileProvider*» требуется выполнить следующие действия: определить «*FileProvider*» в файле AndroidManifest.xml; создать XML-файл, содержащий все пути, которые «*FileProvider*» будет использовать совместно с другими приложениями; связать действительный «URI» в «*Intent*» и активировать его. Для определения

«*FileProvider*» внутри «*AndroidManifest.xml*» используются следующие атрибуты:

– android: authorities – ОС «*Android*» хранит список всех поставщиков, отличая их по полномочиям. Полномочие определяет «*FileProvider*» аналогично, идентификатору приложения, определяющему приложение;

– android:grantUriPermissions – атрибут позволяет безопасно обмениваться внутренним хранилищем вашего приложения используя флаги в намерениях «*FLAG_GRANT_READ_URI_PERMISSION*» или «*FLAG_GRANT_WRITE_URI_PERMISSION*»;

– <meta-data> – определяет путь к XML-файлу, содержащий все пути данных, которые «*FileProvider*» использует с внешними приложениями. XML-файл должен иметь элемент <paths> в качестве его корня. Элемент <*paths*> должен иметь как минимум один дочерний элемент: <*files-path*/> — внутреннее хранилище приложения (*Context.getFilesDir()*); <*cache-path/*> — кэш внутреннего хранилища приложения (*Context.getCacheDir()*); <*external-path/*> — публичное внешнее хранилище (*Environment.getExternalStorageDirectory()*), <*external-files-path/*> — внешнее хранилище приложения (*Context.getExternalFilesDir(null)*), <*external-cache-path/*> — кэш внешнего хранилища приложения, «*Context.getExternalCacheDir()*».

На рисунке 4.1 приведена схема взаимодействия приложения с приложением «Камера».

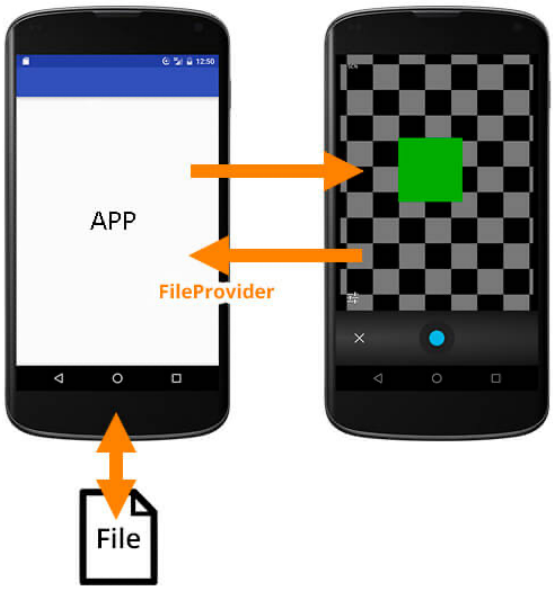

Рисунок 4.1 - Схема получения файлов с использованием FileProvider

На первом шаге в манифест файл добавляется атрибут <*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»* указывает полномочия, состоящие из значения элемента «*android:package»* и строкой *«fileprovider»*,.

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

Метка «*android:resourc»e* указывает путь и имя файла, без .xml расширения.

После добавления «*FileProvider*» в манифест приложения, необходимо указать каталоги, которые содержат файлы общего доступа. Для указания каталогов, требуется создать файл «*paths.xml*» в «*res/xml*» подкаталоге проекта. В следующем фрагменте приведено содержимое «*res/xml/paths.xml*».

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
  <external-files-path name="images" path="Pictures" />
</paths>
```

Метка «*external-files-path*» обозначает элемент для каталогов общего доступа во внешнем хранилище. На данном этапе укзана полная спецификация о «*FileProvider*», генерирующий «URI» для файлов в «*external-files-path*». В результате будет сгенерирован путь до файла, включающий значение «*authorities*» («*com.mirea.fio.camera.fileprovider*»), значение пути в «external-files-path» и имя файла.

Перед вызовом приложения камеры требуется создать файл, в который будет

записан полученный результат.

```
/**
 * 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", Locale.ENGLISH).format(new Date());
   String imageFileName = "IMAGE_" + timeStamp + "_";

   File storageDirectory = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
   return File.createTempFile(imageFileName, ".jpg", storageDirectory);
}
```

После получения файла возможно сгенерировать «URI» на основе значений из манифест-файла.

```
File photoFile = createImageFile();
// генерирование пути к файлу на основе authorities
String authorities = getApplicationContext().getPackageName() + ".fileprovider";
imageUri = FileProvider.getUriForFile(MainActivity.this, authorities, photoFile);
```

Сформированный URI передается в намерение и производится вызов неявного намерения с указанием статической константы «*MediaStore.ACTION_IMAGE_CAPTURE*».

Для реализации приложения требуется использовать механизм разрешений, рассмотренный в предыдущем разделе.

Код, представленный ниже, 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.

```
public class MainActivity extends AppCompatActivity {

  private static final int REQUEST_CODE_PERMISSION = 100;
  private static final int CAMERA_REQUEST = 0;
  private boolean isWork = false;
  private Uri imageUri;
  private ActivityMainBinding binding;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // ДОБАВИТЬ ФАЙЛ РАЗМЕТКИ
    .....

    // ДОБАВИТЬ ПРОВЕРКУ НА НАЛИЧИЕ РАЗРЕШЕНИЙ
    // НА ИСПОЛЬЗОВАНИЕ КАМЕРЫ И ЗАПИСИ В ПАМЯТЬ
    .....
    // Создание функции обработки результата от системного приложения «камера»
    ActivityResultCallback<ActivityResult> callback = new ActivityResultCallback<ActivityResult>() {
      @Override
      public void onActivityResult(ActivityResult result) {
        if (result.getResultCode() == Activity.RESULT_OK) {
          Intent data = result.getData();
          binding.imageView.setImageURI(imageUri);
        }
      }
    };
    ActivityResultLauncher<Intent> cameraActivityResultLauncher =  registerForActivityResult(
        new ActivityResultContracts.StartActivityForResult(),
        callback);

    // Обработчик нажатия на компонент «imageView»
    binding.imageView.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        // проверка на наличие разрешений для камеры
        if (isWork) {
          try {
            File photoFile = createImageFile();
            // генерирование пути к файлу на основе authorities
            String authorities = getApplicationContext().getPackageName() + ".fileprovider";
            imageUri = FileProvider.getUriForFile(MainActivity.this, authorities, photoFile);
            cameraIntent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
            cameraActivityResultLauncher.launch(cameraIntent);
          } catch (IOException e) {
            e.printStackTrace();
          }
        }
      }
    });
  }
  private File createImageFile() throws IOException {
    // реализовать метод создания файла
    .....
    return File.createTempFile(imageFileName, ".jpg", storageDirectory);
  }
```

5 МИКРОФОН. MEDIARECORDER

Создать новый модуль. В меню «*File* | *New* | *New Module* | *Phone & Tablet Module* | *Empty Activity*». Имя модуля «*AudioRecord*». Требуется разработать приложение для работы с диктофоном, позволяющее записывать аудио и его воспроизводить.

Мультимедийные компоненты в наборе инструментариев разработчика ОС «*Android*» поддерживают возможность записи и воспроизведения аудио записей. Для получения доступа к микрофону устройства используется класс «*MediaRecorder*». В первую очередь требуется указать необходимые разрешения для записи аудио и доступа к локальному хранилищу. В манифест-файл необходимо добавить соответствующие разрешения:

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

Также требуется проверить, наличие разрешения пользователя, прежде чем использовать «*MediaRecorder*». Обработка наличия разрешений выполняется в «*MainActivity*» аналогично предыдущему модулю.

```
package ru.mirea.ivanovke.audiorecord;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
    private static final int REQUEST_CODE_PERMISSION = 200;
    private boolean isWork;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        int audioRecordPermissionStatus = ContextCompat.checkSelfPermission(this,
Manifest.permission.RECORD_AUDIO);
        int storagePermissionStatus = ContextCompat.checkSelfPermission(this, android.Manifest.permission.
                WRITE_EXTERNAL_STORAGE);
        if (audioRecordPermissionStatus == PackageManager.PERMISSION_GRANTED && storagePermissionStatus
                == PackageManager.PERMISSION_GRANTED) {
            isWork = true;
        } else {
            // Выполняется запрос к пользователь на получение необходимых разрешений
            ActivityCompat.requestPermissions(this, new String[] {Manifest.permission.RECORD_AUDIO,
                    android.Manifest.permission.WRITE_EXTERNAL_STORAGE}, REQUEST_CODE_PERMISSION);
        }

    }
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[]
grantResults) {
        // производится проверка полученного результата от пользователя на запрос разрешения Camera
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        switch (requestCode){
            case REQUEST_CODE_PERMISSION:
                isWork  = grantResults[0] == PackageManager.PERMISSION_GRANTED;
                break;
        }
        if (!isWork ) finish();
    }
}
```

Класс «*android.media.MediaRecorder*» предназначен для осуществления аудио или видео записи. «*MediaRecorder*» действует как конечный автомат (рисунок 5.1), с возможностью установки различных параметров.

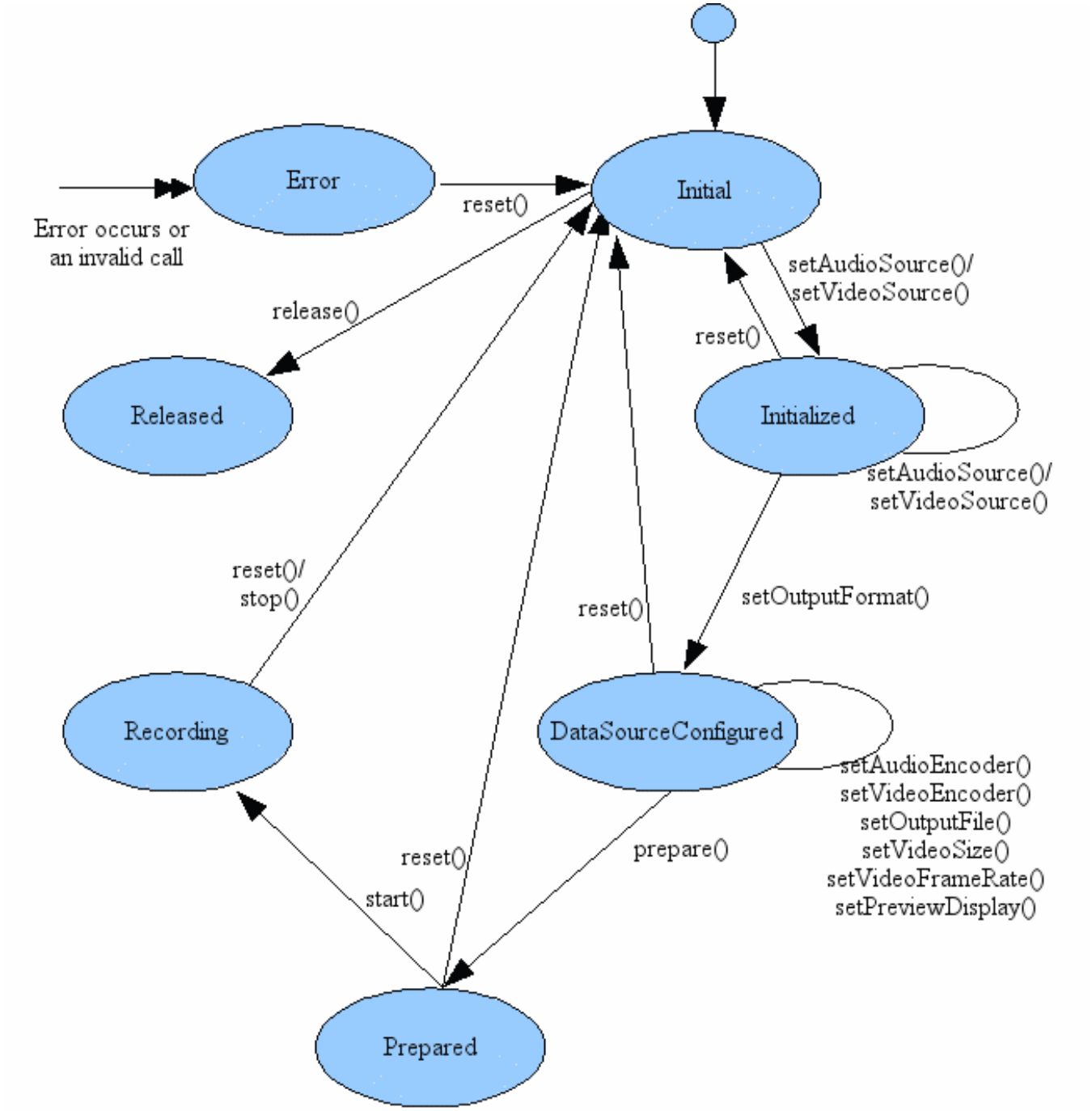

Рисунок 5.1 - Диаграмма состояний MediaRecorder

В файл разметки требуется добавить две кнопки: «Начать запись. *№ студента по списку, группа*» и «Воспроизвести». Для исключения ситуации одновременной записи и воспроизведения требуется вести учет состояния нажатия кнопок. Далее представлен пример, в котором при нажатии одной кнопки становится недоступным другая кнопка.

```
public class MainActivity extends AppCompatActivity{
    private  ActivityMainBinding binding;
    private static final int REQUEST_CODE_PERMISSION = 200;
    private final String TAG = MainActivity.class.getSimpleName();
    private  ActivityMainBinding binding;
    private static final int REQUEST_CODE_PERMISSION = 200;
    private boolean isWork;
    private String fileName = null;
    private Button recordButton = null;
    private Button playButton = null;
    private MediaRecorder recorder = null;
    private MediaPlayer player = null;
    boolean isStartRecording = true;
    boolean isStartPlaying = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // инициализация setContentView()
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        // инициализация кнопок записи и воспроизведения
        recordButton = binding.recordButton;
        playButton = binding.playButton;
        playButton.setEnabled(false);
        recordFilePath = (new File(getExternalFilesDir(Environment.DIRECTORY_MUSIC),
                "/audiorecordtest.3gp")).getAbsolutePath();
        // проверка разрешений на запись аудио и запись на внешнюю память
        ......
       recordButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mStartRecording) {
                    recordButton.setText("Stop recording");
                    playButton.setEnabled(false);
                } else {
                    recordButton.setText("Start recording");
                    playButton.setEnabled(true);

                }
             isStartRecording = !isStartRecording;
            }
        });

        playButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mStartPlaying) {
                    playButton.setText("Stop playing");
                    recordButton.setEnabled(false);
                } else {
                    playButton.setText("Start playing");
                    recordButton.setEnabled(false);
                }
                isStartPlaying = !isStartPlaying;
```

Алгоритм записи файлов содержит несколько этапов. На первом этапе требуется определить источник (свойство «*MediaRecorder.AudioSource*»):

– MIC – встроенный микрофон;

– VOICE_UPLINK – исходящий голосовой поток при телефонном звонке;

– VOICE_DOWNLINK – входящий голосовой поток при телефонном звонке;

– 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 ориентирована под VoIP (если доступен).

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

На третьем этапе определяется тип сжатия звука (свойство «*MediaRecorder.AudioEncoder*»);

– AAC;

– AMR_NB;

– AMR_WB;

Завершающим этапом настройки является указание пути к файлу, в котором будут сохранены аудиоданные (метод «*MediaRecorder.setOutputFile*»). Пример инициализации и настройки «*MediaRecorder»* представлена в методе «*startRecording*».

```
private void startRecording() {
  recorder = new MediaRecorder();
  recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
  recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
  recorder.setOutputFile(recordFilePath);
  recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
  try {
    recorder.prepare();
  } catch (IOException e) {
    Log.e(TAG, "prepare() failed");
  }
  recorder.start();
}
```

В методе «*startRecording*» создается и инициализируется экземпляр «*MediaRecorder*». В качестве источника данных выбирается микрофон (MIC), выходной формат устанавливается «3GPP» (файлы *.3gp), ориентированный на мобильные устройства. Значение кодека установлено «AMR_NB» – аудиоформат с частотой дискретизации 8 кГц. NB означает узкую полосу частот.

Основными положениями записи аудио с помощью «*MediaRecorder*» являются:

1. Аудиофайл сохраняется в памяти и связывается с экземпляром «*MediaRecorder*», обращаясь к методу «*setOutputFile*».

2. Вызов метода «*prepare*» завершает инициализацию «*MediaRecorder*». Для запуска процесса записи требуется вызвать метод «*start*».

3. Запись в файл на карте памяти ведется до тех пор, пока не будет вызван метод «*stop*», который освобождает ресурсы, выделенные экземпляру «*MediaRecorder*».

4. 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 методе «*stopRecording*» представлен пример остановки записи и высвобождение ресурсов после процесса записи.

```
private void stopRecording() {
  recorder.stop();
  recorder.release();
  recorder = null;
}
```

Второй функциональной возможностью приложения является возможность воспроизведения записанного материала из файла. Для работы с медифайлом используется класс «*MediaPlayer*». В методе «*startPlaying*» представлен пример воспроизведения звукового файла.

```
private void startPlaying() {
    player = new MediaPlayer();
    try {
        player.setDataSource(recordFilePath);
        player.prepare();
        player.start();
    } catch (IOException e) {
        Log.e(TAG, "prepare() failed");
    }
}
```

Для остановки и высвобождения ресурсов при воспроизведении медиафайлов представлен метод «*stopPlaying*».

```
private void stopPlaying() {
    player.release();
    player = null;
}
```

Завершающим этапом является вызов методов управления записью и воспроизведением в обработчики нажатий на кнопки.

```
public class MainActivity extends AppCompatActivity{
        ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        recordButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mStartRecording) {
                    ......
                    startRecording();
                } else {
                    ......
                    stopRecording();
                }
                ......
            }
        });
        playButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mStartPlaying) {
                    ......
                    startPlaying();
                } else {
                    ......
                    stopPlaying();
                }
                ......
            }
        });
    }
}
```

6 КОНТРОЛЬНОЕ ЗАДАНИЕ.

В контрольном задании «*MireaProject*» добавить к ранее созданным фрагментам экран аппаратной части со следующим функционалом:

– добавить в приложение механизмы запроса разрешений;

– экран, в котором используется результат с любого из имеющегося датчика для решения какой-либо задачи логической задачи (определение направления на север на экране про определение севера по солнцу, мху и т.д.; влияние высоты гор на организм и т.д.);

– экран, в котором используется результат приложения «камера» для решения какой-либо «творческой» задачи (создание коллажа, заметки, профиля и т.д.);

– экран, в котором используется функционал микрофона для решения какой-либо задачи.