Модуль 3. Основы программирования Android приложений

*Тема 3.5. Сервисы в Android. Типы сенсоров

Оглавление

Тема 3.5. Сервисы в Android. Типы сенсоров.....2
3.5.1. Сервисы в Android 2
3.5.2. Жизненный цикл сервиса. 2
3.5.3. Создание сервиса. 5
3.5.3.1. Создание класса сервиса 5
3.5.3.2. Декларация в манифесте 5
3.5.4. Запуск сервисов. 6
3.5.5. Остановка сервисов.....6
3.5.6. Управление перезагрузкой сервисов 6
Упражнение 3.5.1 7
3.5.7. IntentService 12
3.5.8. Типы сенсоров. 13
Задание 3.5.1.....14
Задание 3.5.2.....14
Список источников 14
Благодарности 15

Тема 3.5. Сервисы в Android. Типы сенсоров.

3.5.1. Сервисы в Android

Сервисы (службы) в Android представляют собой фоновые процессы. Они не имеют интерфейса пользователя (UI) и работают без его вмешательства. Службы являются компонентами приложения, но способны выполнять действия, даже когда оно невидимо, пассивно или, вовсе, закрыто.

В отличие от активностей, сервисы предназначены для длительного существования и обладают более высоким приоритетом, чем скрытые активности. Тем самым, вероятность закрытия сервиса системой, при нехватке ресурсов, куда ниже. К тому же, служба может быть настроена таким образом, что будет автоматически запущена, как только найдутся необходимые ресурсы.

Сегодня, мы можем встретить массу приложений, использующих сервисы. Работая в фоновом режиме службы имеют возможность выполнять сетевые запросы, вызывать уведомления, обрабатывать информацию, проигрывать музыку и выполнять множество иных «закулисных» задач.

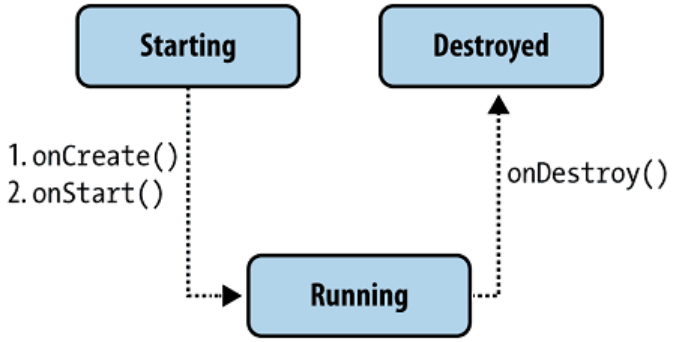
Многие компоненты приложения могут запускать сервисы, обмениваться данными и закрывать их. В роли таких компонентов могут выступать как активности, так и другие сервисы. Так же имеется возможность взаимодействия с широковещательными приемниками. Стоит подчеркнуть, что сервис может быть закрыт как компонентом приложения, так и самостоятельно инициировать собственное завершение.

Обратите внимание, что, не смотря на работу в фоновом режиме – сервисы работают в основном потоке приложения (main UI thread). В случае выполнения «тяжелых» задач, их необходимо запускать в отдельном потоке (3.4), т.к. это может привести к «неприятным» задержкам в UI пользователя.

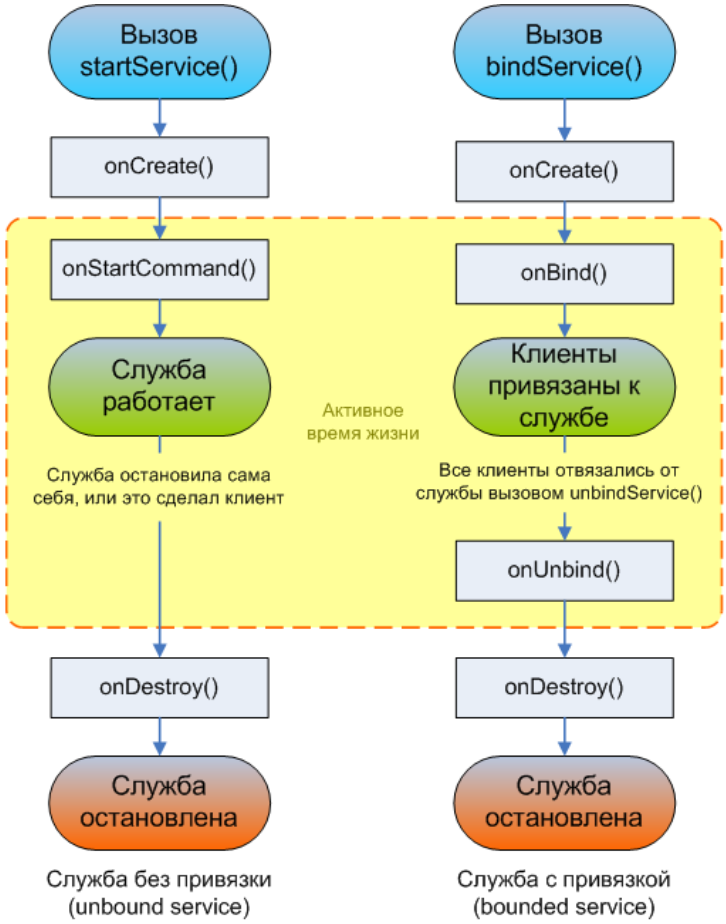
3.5.2. Жизненный цикл сервиса.

Жизненный цикл сервиса, на первый взгляд, куда более прост чем у активности.

- onCreate()
- onStartCommand()
- onDestroy()



На деле же, сервис имеет два вложенных цикла во время жизни и две модели поведения.



- **Полное время жизни** между вызовом onCreate() и возвратом onDestroy(). Как и в activity, служба делает начальную настройку в onCreate() и освобождает все использовавшиеся ресурсы в onDestroy(). Например, служба проигрывания музыки могла создать поток для проигрывания звука в onCreate(), и затем остановку потока в onDestroy(). Методы onCreate() и onDestroy() вызываются для всех служб независимо от того, были ли они созданы вызовом startService() или bindService().
- **Активное время жизни** начинается с вызовом либо onStartCommand(), либо onBind(). Каждый метод получает Intent, который передан соответственно либо через startService(), либо через bindService(). Если служба запущена, активное время жизни заканчивается в тот же момент, что и полное время жизни (служба все равно остается активной после завершения onStartCommand()). Если служба привязана, то активное время жизни заканчивается, когда завершается onUnbind().

Можно установить подключение к работающей службе и использовать это подключение для взаимодействия со службой. Подключение устанавливают вызовом метода Context.bindService() и закрывают вызовом Context.unbindService(). Если служба уже была остановлена, вызов метода bindService() может её запустить.

Методы onCreate() и onDestroy() вызываются для всех служб независимо от того, запускаются ли они через Context.startService() или Context.bindService().

Если служба разрешает другим приложениям связываться с собой, то привязка осуществляется с помощью дополнительных методов обратного вызова:

- IBinder onBind(Intent intent)
- onUnbind(Intent intent)
- onRebind(Intent intent)

В метод обратного вызова onBind() передают объект Intent, который был параметром в методе bindService(), а в метод обратного вызова onUnbind() — объект Intent, который передавали в метод unbindService(). Если служба разрешает связывание, метод onBind() возвращает канал связи, который используют клиенты, чтобы взаимодействовать со службой. Метод обратного вызова onRebind() может быть вызван после onUnbind(), если новый клиент соединяется со службой.

Проанализировав все вышесказанное, можно прийти к выводу, что служба может, по существу, принять 2 формы: ***started*** *(запущена) и* ***bound*** *(привязана).*

Started*. Служба находится в состоянии "started", когда компонент приложения (такой как activity) запустил её вызовом startService(). Будучи запущенной, служба может работать в фоне бесконечно, даже если компонент, который запустил её, был уничтожен. Обычно запущенная служба выполняет одиночную операцию, и не возвращает результат в вызывавший её код. Например, служба может загрузить или выгрузить файл через сеть. Когда эта операция завершена, служба должна остановить саму себя.*

Bound*. Служба находится в состоянии "bound", когда компонент приложения привязался к ней вызовом bindService(). Привязанная служба предоставляет интерфейс типа клиент-сервер, который позволяет компоненту приложения взаимодействовать со службой, отправлять запросы, получать результаты, и даже делать то же самое с другими процессами через IPC. Привязанная служба работает, пока другой компонент приложения привязан к ней. Одновременно к службе может быть привязано несколько компонентов, но когда они все отвязаны (unbind), служба уничтожается.*

Несмотря на то, что в этой документации эти два типа работы службы рассматриваются по отдельности, Ваша служба может работать обоими способами — она может быть запущена (чтобы работать бесконечно) и также позволяет привязку. Это просто вопрос того, реализуете ли Вы или нет методы обратного вызова : onStartCommand() чтобы разрешить компонентам запустить службу и onBind() чтобы разрешить привязку.

3.5.3. Создание сервиса.

3.5.3.1. Создание класса сервиса

Чтобы определить службу, необходимо создать новый класс, расширяющий базовый класс android.app.Service. После, в обязательном порядке, переопределить метод onBind().

```
public class MyService extends Service {

      public class MyService extends Service {

      @Override
      public IBinder onBind(Intent intent) {
            return null;
      }
      }
      @Override
      public IBinder onBind(Intent intent) {
      return null;
      }
}
```

3.5.3.2. Декларация в манифесте

Как и активности (и другие компоненты приложения), вы должны декларировать все службы в файле манифеста приложения. Чтобы декларировать службу, добавьте элемент <service> в качестве дочернего для элемента <application>.

```
<                    manifest                    ...                    >
  ...
  <                   application                    ...                 >
      <service                  android:name=".MyService"                />
      ...
  <                             /application                              >
< /manifest >
```

Стоит подчеркнуть, что тег <service> может включать целый набор атрибутов. Например, такой как enabled, дает возможность другим приложениям получать доступ к вашему сервису. Подробнее со списком атрибутов и правилом их использования можно ознакомиться по ссылке.

При использовании Android Studio, для создания сервисов можно использовать пункты меню New | Service | Service. Это позволит автоматически создать класс наследник Service и задекларировать его в манифесте приложения.

3.5.4. Запуск сервисов.

Вы можете запустить службу из activity или другого компонента приложения путем передачи Intent (указывающего службу для запуска) в startService(). Система Android вызывает метод службы onStartCommand() и передает ей Intent (вы никогда не должны вызывать onStartCommand() напрямую). Например, activity может запустить службу примера в предыдущей секции (HelloService), используя explicit intent с startService().

```
Intent intent = new Intent(this, MyService.class);
startService(intent);
```

Чтобы обеспечить безопасность приложения, всегда используйте явное намерение (explicit intent), когда запускаете службу или делаете привязку к ней, и не декларируйте intent-фильтров для службы. Если это критически важно, что вы допускаете некоторую неоднозначность в запуске службы, вы можете предоставить фильтры intent для ваших служб и исключить имя компонента из Intent, но тогда вы должны установить пакет для intent вызовом setPackage(), который предоставляет достаточное разрешение неоднозначности для целевой службы.

3.5.5. Остановка сервисов

Запущенная служба должна обслуживать свой жизненный цикл, т. е. система не остановит, не удалит службу за исключением ситуаций, требующих освобождения памяти системы, и служба продолжит работать после завершения метода onStartCommand(). Таким образом, служба должна остановить саму себя вызовом stopSelf(), или другой компонент может остановить службу вызовом stopService().

Как только поступил запрос на остановку через stopSelf() или stopService(), система уничтожит службу так быстро, как это только возможно.

Предупреждение: важно, чтобы ваше приложение останавливало свои службы, когда они завершили работу, чтобы избежать потери ресурсов системы, и сохранить энергию батареи. Если это необходимо, другие компоненты могут остановить службу вызовом stopService(). Даже если вы разрешили привязку службы, вы всегда должны останавливать службу самостоятельно, если она когда-либо получила вызов onStartCommand().

3.5.6. Управление перезагрузкой сервисов

В большинстве случаев, при работе с сервисами, необходимо переопределять метод onStartCommand(). Он вызывается каждый раз, когда сервис стартует с помощью метода startService(), поэтому может быть выполнен несколько раз на протяжении работы. Вы должны убедиться, что ваш сервис это предусматривает.

Обратите внимание, что метод onStartCommand() должен вернуть целое число (integer). Эта целая величина описывает, как система должна продолжить работу службы, когда нужно уничтожить

службу. Возвращаемое значение из onStartCommand() должно быть одной из следующих констант:

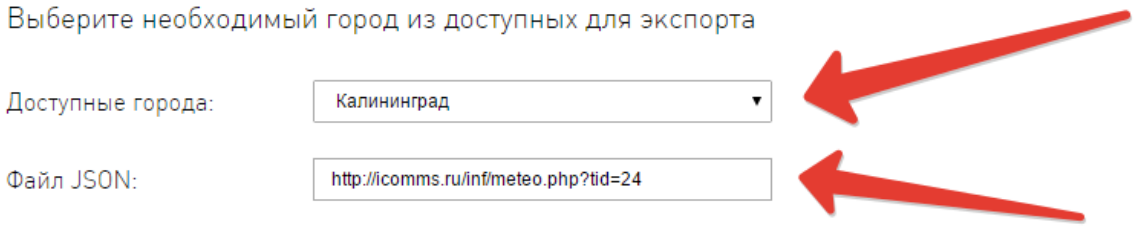
- **START_STICKY** - Описывает стандартное поведение. Похоже на то, как был реализован метод onStart() в Android 2.0. Если вы вернете это значение, обработчик onStartCommand() будет вызываться при повторном запуске сервиса после преждевременного завершения работы. Обратите внимание, что аргумент Intent, передаваемый в onStartCommand(), получит значение null. Данный режим обычно используется для сервисов, которые сами обрабатывают свои состояния, явно стартуя и завершая свою работу при необходимости (с помощью методов startService() и stopService()). Это относится к сервисам, которые проигрывают музыку или выполняют другие задачи в фоновом режиме
- **START_NOT_STICKY** - Этот режим используется в сервисах, которые запускаются для выполнения конкретных действий или команд. Как правило, такие сервисы используют stopSelf() для прекращения работы, как только команда выполнена. После преждевременного прекращения работы сервисы, работающие в данном режиме, повторно запускаются только в том случае, если получат вызовы. Если с момента завершения работы Сервиса не был запущен метод startService(), он остановится без вызова обработчика onStartCommand(). Данный режим идеально подходит для сервисов, которые обрабатывают конкретные запросы, особенно это касается регулярного выполнения заданных действий (например, обновления или сетевые запросы). Вместо того, чтобы перезапускать сервис при нехватке ресурсов, часто более целесообразно позволить ему остановиться и повторить попытку запуска по прошествии запланированного интервала
- **START_REDELIVER_INTENT** - В некоторых случаях нужно убедиться, что команды, которые вы посылаете сервису, выполнены. Этот режим — комбинация предыдущих двух. Если система преждевременно завершила работу сервиса, он запустится повторно, но только когда будет сделан явный запрос на запуск или если процесс завершился до вызова метода stopSelf(). В последнем случае вызовется обработчик onStartCommand(), он получит первоначальное намерение, обработка которого не завершилась должным образом.

Такой подход позволяет методу onStartCommand() быстро завершить работу и даёт возможность контролировать поведение сервиса при его повторном запуске.

Упражнение 3.5.1

Реализуйте приложение, показывающее погоду в вашем городе.

Первое, что необходимо для решения этой задачи – это источник данных о погоде в вашем городе. Подобную информацию можно получить в интернете, но представлена она там в виде структуры понятной пользователю, а не программе. Для удобства работы целесообразно использовать API любого погодного сервиса, для получения информации в виде XML или JSON. Для данной задачи вполне подойдет (http://icomms.ru/pogoda_xml_json.html). Там же, на сайт, есть информация о формате выдаваемого JSON. Прежде чем приступить к написанию самого приложения выберите свой город и скопируйте ссылку в удобное для вас место. Стоит подчеркнуть, что при желании, есть возможность использовать любой другой сервис или вовсе парсить HTML страницу, но, как правило, все это более трудоемкий процесс.



Далее, создадим проект с одной активностью. Первоначально, просто создадим сервис и покажем полученные о ссылке данные. В качестве разметки установим TextView с возможностью прокрутки, т.к. первоначальный набор данных будет достаточно большим (прогноз на 2 недели вперед).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical"
      android:paddingBottom="@dimen/activity_vertical_margin"
      android:paddingLeft="@dimen/activity_horizontal_margin"
      android:paddingRight="@dimen/activity_horizontal_margin"
      android:paddingTop="@dimen/activity_vertical_margin"
      tools:context=".MainActivity">

      <ScrollView
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:fillViewport="true"
      android:scrollbars="vertical">

       <TextView
            android:id="@+id/temp_txt"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center" />

      </ScrollView>

</LinearLayout>
```

При написании сервиса нам понадобиться вспомнить, что для загрузки данных из Интернет, мы можем использовать класс URL. Также, не стоит забывать, что сегодня Android запрещает работу с сетью в основном потоке, а также, работа с Интернет требует добавления разрешения в манифест приложения.

Создадим сервис.

```
import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.IBinder;
import android.widget.Toast;
```

```
import java.io.InputStream;
import java.net.URL;
import java.util.Scanner;

public class GisService extends Service {

	public static final String CHANNEL = "GIS_SERVICE";
	public static final String INFO = "INFO";

	@Override
	public void onCreate() {
	Toast.makeText(this, "Служба создана",
			Toast.LENGTH_SHORT).show(); // сообщение о создании
службы
	}

	@Override
	public int onStartCommand(Intent intent, int flags, int startId) {

	// сообщение о запуске службы
	Toast.makeText(this, "Служба запущена",
			Toast.LENGTH_SHORT).show();

	// создаем объект нашего AsyncTask (необходимо для работы с сетью)
	GisAsyncTask t = new GisAsyncTask();
	t.execute(); // запускаем AsyncTask

	return START_NOT_STICKY;
	}

	@Override
	public void onDestroy() {
	//сообщение о остановке службы
	Toast.makeText(this, "Служба остановлена",
			Toast.LENGTH_SHORT).show();
	}

	@Override
	public IBinder onBind(Intent intent) {
	return null;
	}

	//поток работы с сетью
	private class GisAsyncTask extends AsyncTask<Void, Void, String> {

	@Override
	protected void onPostExecute(String aVoid) {
		Intent i = new Intent(CHANNEL); // интент для отправки ответа
		i.putExtra(INFO, aVoid); // добавляем в интент данные
		sendBroadcast(i); // рассылаем
```

```
	}

	@Override
	protected String doInBackground(Void... voids) {
		String result;
		try {
			//загружаем данные
			URL url =
				new URL("http://icomms.ru/inf/meteo.php?tid=24");

			// "оборачиваем" для удобства чтения
			Scanner in = new Scanner((InputStream) url.getContent());

			// читаем и добавляем имя json массива
			result = "{\"gis\":" + in.nextLine() + "}";

		} catch (Exception e) {
			result = e.toString();
		}
		return result;
	}
	}
}
```

Прописываем разрешение на работу сетью и регистрируем сервис в манифесте.

```
<                    manifest                    ...                    >
  ...
  <uses-permission android:name="android.permission.INTERNET" />
  ...
  <                    application                    ...                    >
      <service android:name=".GisService"
          android:enabled="true"
          android:exported="true"/>
  <                              /application                              >
< /manifest >
```

Код MainActivity

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;
```

```
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class MainActivity extends AppCompatActivity {

      private TextView tempText;

      @Override
      protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      tempText = (TextView) findViewById(R.id.temp_txt);
      registerReceiver(receiver, new IntentFilter(GisService.CHANNEL));
      Intent intent = new Intent(this, GisService.class);
      startService(intent);
      }

      @Override
      protected void onDestroy() {
      super.onDestroy();
      Intent intent = new Intent(this, GisService.class);
      stopService(intent);
      }

      protected BroadcastReceiver receiver = new BroadcastReceiver() {
      @Override
      public void onReceive(Context context, Intent intent) {
            try {
                  JSONObject json = new
JSONObject(intent.getStringExtra(GisService.INFO));
                  JSONArray gisArray = json.getJSONArray("gis");
                  tempText.setText(gisArray.toString());
            } catch (JSONException e) {
                  Toast.makeText(MainActivity.this, "Неверный формат JSON",
                        Toast.LENGTH_LONG).show();
            }
      }
      };
}
```

В целом, все происходящие внутри главной активности, не должно вызвать у вас вопросов, за исключением, может быть, одного. Для обработки полученных результатов здесь были использованы стандартные классы для работы с JSON. Для подробного ознакомления можно воспользоваться ссылкой http://java-help.ru/android-json.

В результате, данное приложение позволяет нам пролистывать JSON массив, с данным погоды в выбранном вами городе на две недели. Конечно, такой формат представления данных неприемлем для конечного пользователя. Следовательно, для завершения данного приложения,

необходимо «распарсить» уже полученный JSON массив и представить все данные в понятной пользователю структуре. (Для работы с JSON – используйте функции, описанные по ссылке http://java-help.ru/android-json).

3.5.7. IntentService

Поскольку большинству запущенных служб не нужно обрабатывать несколько запросов одновременно (на самом деле это довольно опасная ситуация для многопоточного сценария), то наверно будет лучшим решением реализовать вашу службу на основе класса **IntentService**. IntentService делает следующее:

- создает по умолчанию рабочий поток (worker thread), который выполняет все intent-ы, переданные в onStartCommand(), отдельно от главного потока вашего приложения;
- создает рабочую очередь (work queue) которая передает один intent за раз в вашу реализацию onHandleIntent(), так что вам не нужно беспокоиться о многопоточности;
- останавливает службу после обработки всех стартовых запросов, так что вам не нужно вызывать stopSelf();
- предоставляет реализацию по умолчанию onBind(), которая возвращает null (по умолчанию привязка невозможна);
- предоставляет реализацию по умолчанию onStartCommand(), которая отправляет intent в рабочую очередь (work queue), и затем в Вашу реализацию onHandleIntent().

Все это, в целом, приводит к тому, что все, что вам нужно сделать - только реализовать onHandleIntent(), чтобы выполнить работу.

3.5.8. Типы сенсоров.

Android-устройства имеют множество сенсоров, к которым программист может получить доступ, причем с появлением новых моделей их разнообразие постоянно растет.

- Акселерометр
- Барометр
- Гироскоп
- Датчик освещения
- Датчик магнитных полей
- Датчик поднесения телефона к голове
- Датчик температуры аппарата
- Датчик температуры окружающей среды
- Измеритель относительной влажности
- И другие.

Конкретный набор сенсоров зависит от Android-устройства, но в большинстве смартфонов и планшетов присутствуют самые основные - акселерометр и гироскоп.

Для обработки событий, связанными с датчиками, в классе активности нужно реализовать интерфейс **SensorEventListener:**

```
public class MainActivity extends Activity implements SensorEventListener
```

В активности должны появиться следующие методы:

```
@Override
 // Изменение точности показаний датчика
public void onAccuracyChanged(Sensor sensor, int accuracy) {
  }
@Override
  protected void onResume() {
  }
@Override
  protected void onPause() {
  }
@Override
// Изменение показаний датчиков
  public void onSensorChanged(SensorEvent event) {
  }
```

В методе **OnCreate** получим объект типа **SensorManager** – менеджер сенсоров устройства. С помощью этого объекта можно получить доступ к нужному сенсору. Далее показан фрагмент класса активности с получением сенсора – акселерометра:

```
public class MainActivity extends Activity, implements SensorEventListener
{
      private final SensorManager mSensorManager;
     private final Sensor mAccelerometer;
```

```
      public SensorActivity() {
         mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
      mAccelerometer =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
      }

      protected void onResume() {
      super.onResume();
         mSensorManager.registerListener(this, mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
      }

      protected void onPause() {
      super.onPause();
         mSensorManager.unregisterListener(this);
      }

      public void onAccuracyChanged(Sensor sensor, int accuracy) {
      }

      public void onSensorChanged(SensorEvent event) {
      }
 }
```

Получение значений с датчика осуществляется с помощью объекта типа **SensorEvent,** который передается внутрь метода **onSensorChanged.**

За подробностями работы с **SensorEvent** можно обратиться к официальной справочной системе http://developer.android.com/reference/android/hardware/SensorEvent.html

Задание 3.5.1.

Реализуйте приложение отображающие на экране информацию об углах наклона аппарата в 3-х плоскостях.

Задание 3.5.2.

Реализуйте приложение, которое при падении освещения проигрывает мистическую музыку. При восстановлении освещения - дорожка останавливается (для воспроизведения музыки используйте медиа плеер внутри сервиса).

Список источников

1. http://metanit.com/java/android/16.1.php
2. http://microsin.net/programming/android/services.html
3. http://developer.alexanderklimov.ru/android/theory/services-theory.php

4. http://www.cyberforum.ru/android-dev/thread1108200.html
5. http://icomms.ru/pogoda_xml_json.html
6. http://developer.android.com/intl/ru/guide/topics/manifest/service-element.html
7. http://habrahabr.ru/company/eastbanctech/blog/192998
8. http://developer.android.com/reference/android/hardware/SensorEvent.html
9. http://habrahabr.ru/post/137678/
10. http://java-help.ru/android-json

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю IT ШКОЛЫ SAMSUNG Бабошкину Артуру Олеговичу.