Лабораторная работа № 10. Разработка мобильных приложения с функцией геолокации и сетевым взаимодействием

Цель лабораторной работы:

- Разработка приложения, демонстрирующего геолокационные возможности.
- Изучить работу с потоками. Научиться работать с мультимедиа файлами.
- Изучить работу с классом AsyncTask. Так как данный класс помечен как deprecated, начиная с API 30, рекомендуется в заданиях для самостоятальной работы использовать классы из пакета java.util.concurrent для проектов на Java и корутины для проектов на Kotlin в случае использования API 30 и выше.

Задачи лабораторной работы:

- Разработать приложение, получающее координаты устройства и отслеживающее их изменения.
- Разработать погодное приложение.
- Разработать приложение с подключением к сервису LastFM и получением сведений об артистах и композициях

Table of Contents

В данной работе рассмотрим процесс создания приложения, отслеживающего изменения координат устройства.

10.1 Разработка приложения, получающего координаты устройства и отслеживающего их изменение

Создадим приложение.

Далее будем править java файл, определяющий класс активности приложения. Внесем в этот класс следующие дополнения:

- Нам потребуется экземпляр класса LocationManager, поэтому в методе onCreate() запишем следующую конструкцию:

```
LocationManager mlocManager =
  (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

экземпляр класса LocationManager, как и большинство системных сервисов, создается с помощью вызова метода getSystemService().

- Укажем, что в приложении необходимо получать обновления координат, сделаем это с помощью вызова метода requestLocationUpdates() класса LocationManager:

```
mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
              0, 0, mlocListener);
```

первый параметр метода указывает способ получения координат, для этого используются константы класса LocationManager:

GPS_PROVIDER	- сообщает, что координаты определяются с помощью GPS;
NETWORK_ PROVIDER	- сообщает, что координаты определяются с использованием сигналов сотовых вышек и WiFi.

В случае, когда необходимо получать координаты и с GPS, и от сотовых вышек необходимо вызвать метод requestLocationUpdates() дважды: один раз первый параметр должен быть равен GPS_PROVIDER, второй раз - NETWORK_ PROVIDER.

Второй и третий параметры метода управляют частотой обновлений. Второй определяет минимальное время между извещениями об обновлениях, третий - минимальное изменение расстояния между извещениями об обновлениях. Если оба эти параметра имеют нулевое значение, то извещения об обновлениях появляются настолько часто, насколько это возможно.

Последний параметр указывает на слушателя, который получает вызовы при обновлениях координат. В нашем случае слушателем является переменная mlocListener - реализация интерфейса LocationListener.

- Теперь необходимо объявить переменную mlocListener:

```
LocationListener mlocListener = new LocationListener(){
  public void onLocationChanged
(Location location) {
    tvLat.append(" "+location.getLatitude());
    tvLon.append(" "+location.getLongitude());
  }
  public void onStatusChanged(String provider, int status,
```

```
                Bundle extras) {}
  public void onProviderEnabled(String provider) {}
  public void onProviderDisabled(String provider) {}
};
```

Переменная mlocListener инициализируется реализацией интерфейса LocationListener. Этот интерфейс содержит 4 метода, каждый из которых должен быть определен в реализации интерфейса. Нас интересует метод onLocationChanged(), который вызывается каждый раз при изменении показаний GPS датчика, в этом методе всего лишь выполняется вывод полученных координат в информационные поля.

- Чтобы разрешить получать обновления координат с помощью сигналов от сотовых вышек (NETWORK_PROVIDER) или с GPS датчика (GPS_PROVIDER), необходимо в файле манифеста добавить строку:

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

(для работы с сигналами сотовых вышек) или

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

(для работы GPS датчиком).

Без этих полномочий приложение "сломается" во время выполнения, когда попробует запросить обновление координат. Если в приложении в качестве источника координат используются и NETWORK_PROVIDER, и GPS_PROVIDER, в манифесте достаточно запросить только полномочия на ACESS_FINE_LOCATION. Тогда как ACCESS_COARSE_LOCATION позволяет работать только с NETWORK_PROVIDER.

В листинге 10.1 представлен код приложения, позволяющего получать координаты устройства и отслеживать их изменения, используя GPS приемник.

После создания приложения, разумеется, необходимо протестировать его работу. Проще всего это сделать, используя реальное устройство под управлением Android, но если устройства под рукой нет, можно использовать виртуальное устройство. Однако при этом необходимо решить вопрос, каким образом имитировать передачу данных о местоположении на эмулятор. Проще всего воспользоваться для этого DDMS.

В Android Studio откройте компоновку (Perspective) DDMS, в этой компоновке выберите вкладку **Emulator Control**. С помощью инструмента DDMS, можно имитировать обновление данных о местоположении несколькими способами:

- вручную передать значения широты и долготы на виртуальное устройство;
- использовать GPX файл, описывающий маршрут для считывания эмулятором;
- использовать KML файл, описывающий отдельные метки местности для последовательной передачи на виртуальное устройство.

В этой части работы рассмотрели вопросы разработки приложений, способных получать координаты устройства и отслеживать их изменения. Разработанное приложение реагирует на изменения координат устройства и выдает новые координаты в соответствующие информационные поля.

Отличия доставки сообщений о местоположении

Данные местоположения в Android предоставляются системной службой LocationManager. Эта служба предоставляет обновленную позиционную информацию всем приложениям, для которых она представляет интерес. Доставка обновлений осуществляется одним из двух способов.

Первый (и пожалуй, наиболее прямолинейный) способ доставки проходит через интерфейс LocationListener, который описан в лабораторной работе выше. Использование LocationListener для получения обновлений удобно в тех случаях, когда данные местоположения должны поступать только одному компоненту вашего приложения.

Второй способ доставки через интерфейс PendingIntent API, который позволит приложению RunTracker отслеживать местоположение пользователя независимо от состояния (или наличия) пользовательского интерфейса. Использование закрепляемой службы не рекомендуется из-за тяжеловесности. Рекомендуется использовать интерфейс PendingIntent API, появившийся в Android 2.3 (Gingerbread).

Запрашивая информацию местоположения с использованием PendingIntent, фактически приказываем LocationManager отправлять некую разновидность Intent в будущем. Таким образом, компоненты приложения (и даже весь процесс) могут прекратить существование, а LocationManager будет доставлять интенты, пока не будет отправлен приказ ему остановиться, запустив новые компоненты, которые отреагируют на интенты нужным образом. Например, такая схема позволит предотвратить поглощение приложением лишних ресурсов, в то время как оно активно отслеживает местоположение устройства.

```
package com.example.lab5_4_geolocation;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
```

```
import android.app.Activity;
import android.content.Context;
import android.widget.TextView;

public class MainActivity extends Activity {

  TextView tvOut;
  TextView tvLon;
  TextView tvLat;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    tvOut = (TextView)findViewById(R.id.textView1);
    tvLon = (TextView)findViewById(R.id.longitude);
    tvLat = (TextView)findViewById(R.id.latitude);

    //Получаем сервис
    LocationManager mlocManager = (LocationManager)
                  getSystemService(Context.LOCATION_SERVICE);

    LocationListener mlocListener = new LocationListener(){
      public void onLocationChanged(Location location) {

      //Called when a new location is found by the network location provider.
        tvLat.append(" "+location.getLatitude());
        tvLon.append(" "+location.getLongitude());
      }

      public void onStatusChanged(String provider, int status, Bundle extras) {}

      public void onProviderEnabled(String provider) {}

      public void onProviderDisabled(String provider) {}

    };

    //Подписываемся на изменения в показаниях датчика

    mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
                  mlocListener);

    //Если gps включен, то ... , иначе вывести "GPS is not turned on..."
    if (mlocManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
      tvOut.setText("GPS is turned on...");

    } else {
      tvOut.setText("GPS is not turned on...");
    }
  }
}
```

Листинг 10.1. Получение геолокационных данных

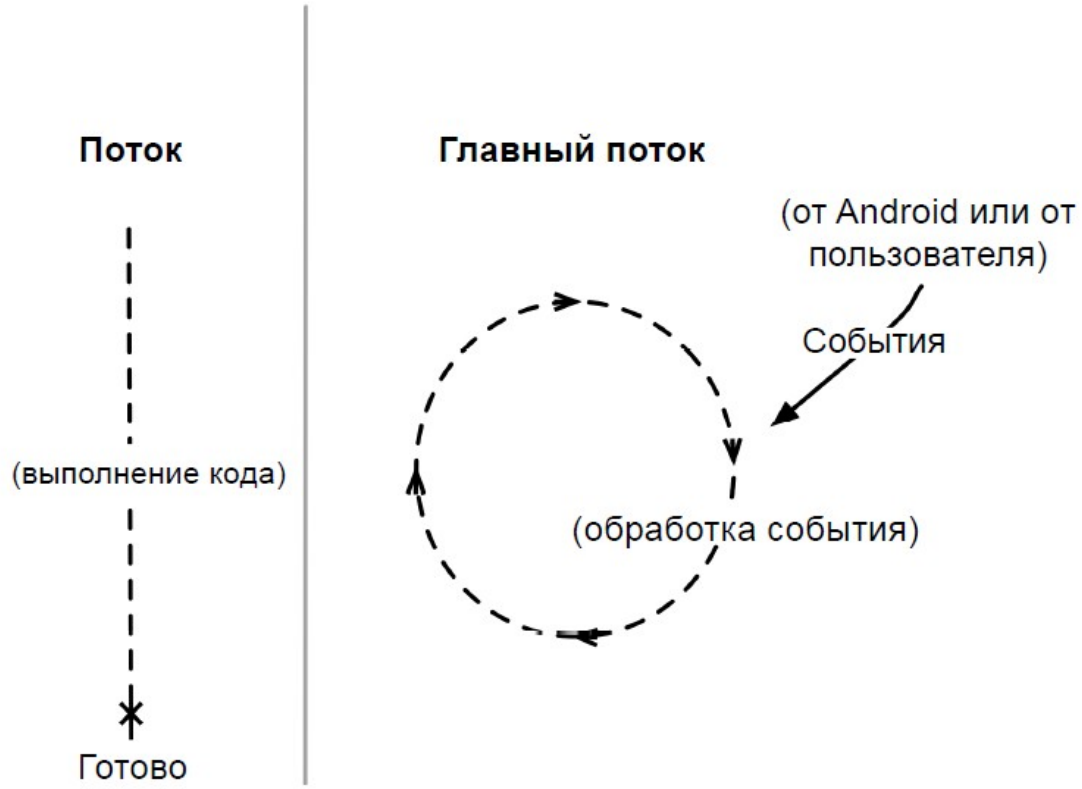
10.2 Взаимодействие с сервером

Теоретические сведения

Познакомьтесь с документацией по процессам и потокам —

https://developer.android.com/guide/components/processes-and-threads?hl=ru.

Главный и обычный потоки



10.1. Графическое представление потоков

Асинхронные потоки в Android

AsyncTask-это специальный абстрактный класс, предоставляющий набор методов для реализации:

- onPreExecute-для размещения инициализирующего кода (UI поток)
- doInBackground-для размещения тяжелого кода, который будет выполняться в другом потоке
- onProgressUpdate-для информирования о прогрессе (UI поток)
- onPostExecute-для обработки результата, возвращенного doInBackground(UI поток) и вспомогательных методов
- isCancelled-для получения информации об отмене задачи
- publishProgress-для перевода сообщения о прогрессе в UI поток с последующим вызовом onProgressUpdate

Последовательность выполнения методов AsyncTask

onPreExecute

doInBackground

onPostExecute

publishProgress

onProgressUpdate

Правила использования AsyncTask:

Объект AsyncTaskдолжен быть создан в UI-потоке

- Метод execute должен быть вызван в UI-потоке
- Метод executeможет быть запущен только один раз
- Не вызывайте методы onPreExecute, doInBackground, onPostExecute и onProgressUpdate

Передача данных в AsyncTask

Объявляем класс

Class MyAsyncTaskextends AsyncTask<String, Integer, Long>{

...

}

- Первый параметр используется методом doInBackground protectedLong doInBackground(String... urls)
- Второй параметр используется методом onProgressUpdate protectedvoidonProgressUpdate(Integer... progress)
- Третий параметр используется методом onPostExecute protectedvoidonPostExecute(Longresult)

Промежуточные данные

Последовательность действий для передачи промежуточных данных в основной поток программы:

- В методе doInBackgroundвызываем метод publishProgress
- В методе onProgressUpdate обрабатываем переданный в publishProgress параметр и выводим прогресс

Метод get

- Возвращает результат выполнения метода doInBackground
- Вызывается из UI потока

MyAsyncTaskat = newMyAsyncTask();

...

result= at.get();

Критерии оценивания

4 — приложения на основе заданий 1-2 на Java из лабораторной работы и ответы на контрольные вопросы;

5-6 — приложения на основе заданий 1-4 из лабораторной работы и ответы на контрольные вопросы;

7-8 — приложения на основе заданий 1-3, 5 на Java, задание 4 на Kotlin из лабораторной работы и ответы на контрольные вопросы;

9 — приложения на основе заданий 1-5 на Java и Kotlin из лабораторной работы, дополнительный функционал и ответы на контрольные вопросы.

Задания для самостоятельной работы

Для реализации концепции параллельного программирования, начиная с API 30 рекомендуется использовать для проекта на java классы из пакета java.util.concurrent (https://developer.android.com/reference/java/util/concurrent/package-summary), так как класс AsyncTask помечен как deprecated (https://developer.android.com/reference/android/os/AsyncTask?hl=ru). В случае реализации проекта на Kotlin рекомендуется познакомиться с концепцией корутин (https://developer.android.com/topic/libraries/architecture/coroutines?hl=ru). При реализации самостоятельных заданий рекомендуются учитывать версию API и в зависимости от выбранного API использовать AsyncTask или классы из пакета java.util.concurrent для проекта на Java и корутины для проекта на Kotlin.

Задание 1

Реализовать на Java приложение RunTracker, которое позволяет отслеживать текущее местоположение и имеет две управляющие кнопки в интерфейсе. Фрагменты коды приложения представлены в разделе 10.1 текущей лабораторной работы. Пример интерфейса представлен на рисунке на странице 3 раздела 10.1. Пользовательский интерфейс отвечает за вывод простых данных о текущем местоположении. В нем имеются кнопки для запуска и остановки текущей серии. В качестве разметки использовать ConstraintLayout.

Задание 2

Внести изменения в приложение, реализованное в задании 1, используя один из двух вариантов хранения данных маршрута для передачи на виртуальное устройство:

а) GPX-файл

б) KML-файл

Задание 3

Реализовать приложение RunTracker из задания 1, изменив разметку на

TableLayout и LinerLayout и применив PendingIntent API.

В макете использовать виджет TableLayout, чтобы разместить элементы интерфейса на экране. TableLayout состоит из пяти виджетов TableRow и одного LinearLayout . Каждый виджет TableRow содержит два виджета TextView : в первом выводится метка, а второй заполняется данными во время выполнения. LinearLayout содержит два виджета Button.

Для доставки обновлений о метостоположении в системную службу LocationManager использовать PendingIntent API. Отличия использования интерфейса LocationListener и PendingIntent API описаны в п. 10.1.

Задание 4

Рассмотрите пример передачи данных, пример вывода промежуточных данных:

Пример передачи данных

```
MyAsyncTaskat = newMyAsyncTask();
at.execute(“url1”, “url2”);
doInBackground(String... urls)
```

Пример вывода промежуточных данных

```
@Override
Protected void doInBackground(String... urls) {
        try{
                int cnt= 0;
                for(Stringurl: urls) {
                        // обрабатываем первый параметр
                                ...
                        // выводим промежуточные результаты
                        cnt++;
                        publishProgress(cnt);
        }
        TimeUnit.SECONDS.sleep(1);
} catch(InterruptedExceptione) {
        e.printStackTrace();
        }
        return null;
```

```
}
@Override
        protected void onProgressUpdate(Integer... values) {
                super.onProgressUpdate(values);
                tv.setText("обработано " + values[0] + " параметров");
}
```

Реализуйте погодное приложение на Java или Kotlin с использованием AsyncTask, получением ответа от API и извлечением ответа JSON. Приложение должно иметь минимум 2 встроенных города (Областной город по месту рождения, любой город название которого начинается на первую букву Вашей фамилии на русском или английском языке). Пользователь может добавить свой город. Приложение должно

a) выводить города с указанием температуры,

b) должно уметь показывать более подробную информацию по городу

c) уметь показывать прогноз погоды (3 или 7 дней). http://developer.yahoo.com/weather/, или на http://openweathermap.org/API, или любой другой на выбор.

Задание 5

На основании изученных примеров разработать приложение на Kotlin с подключением к сервису Last FM (или другому музыкальному сервису), сохранять сведения о 2-3 артистах и его популярных композициях в базу данных и выводить список из 10 популярных произведений артиста при поиске по его личным данным. Для сохранения песни и названия необходимо создать базу данных, содержащую таблицу со следующими полями:

1) ID

2) Исполнитель

3) Название трека

4) Количество прослушиваний

5) Количество пользователей, прослушавших композицию

При включении приложения необходимо производить проверку подключения к Интернету. В случае если подключение отсутствует – выводить всплывающее сообщение (Toast) с предупреждением о запуске в автономном режиме (доступен только просмотр внесённых ранее записей).

Приложение должно содержать операцию (activity), позволяющее просматривать внесённые в базу данных записи.

Контрольные вопросы

1. Какой интерфейс используется для получения уведомлений от LocationManager, когда местоположение изменилось?
2. Как расшифровывается аббревиатура NMEA? И для чего применяется?
3. Как расшифровывается аббревиатура GNSS?
4. Расшифруйте аббревиатуры GPX, KML. Для каких задач применяются файлы формата GPX и KML? Приведите примеры файлов формата GPX и KML.
5. Когда рекомендуется применять для определения местоположения PendingAPI вместо LocationManager?
6. Как называется класс данных, использующийся для представления географического местоположения?
7. Какими данными описывается местоположение?
8. Какие строки необходимо добавить в AndroidManifest.xml, чтобы приложение для определения местоположения получило доступ к Интернет?
9. Для каких задач применяется класс AsyncTask?
10. Для чего предназначены методы AsyncTask, такие как onPreExecute(), doInBackground(Params), onPostExecute(Result)?