8. Лабораторная работа № 2 «Датчики в Android»

8.1. Цель лабораторной работы

Научиться использовать датчики в программных приложениях под Android.

8.2. Введение

Android, как система, ориентированная на мобильный сегмент рынка, имеет мощную поддержку разнообразных датчиков, а также датчика GPS. Далее мы рассмотрим два датчика – акселерометр (датчик ускорения) и GPS датчик географических координат.

Для выполнения работы необходимы базовые знания языка Java и Android API. Самое важное требование – необходимо Android-устройство (смартфон или планшет) с акселерометром и GPS.

8.3. Инструкция по выполнению лабораторной работы

8.3.1. Подготовка

Приложение, которое будет подробно рассмотрено, написано в IDE Eclipse Indigo, поэтому используйте именно эту версию или более позднюю. Также предполагается, что вы уже ознакомились с возможностью разрабатывать под Android в Eclipse и установили все требуемые плагины.

8.3.2. Выполнение работы

Проделайте следующие шаги, чтобы посмотреть предварительно подготовленное приложение в действии.

Приложение может быть запущено с использованием виртуальной машины Davlik.

1). Импорт приложения.

Запустите Eclipse, выберите пункт меню File, в списке выполните команду Import, в появившемся окне во вкладке General, щелкните

на Existing project into workspace. Появиться диалоговое окно, в котором вы можете указать путь до проекта приложения либо в виде архива, либо в виде простой папки. *Примеры находится в каталоге* lab02 файла labAtom21.rar. Затем нажимаете на кнопку OK. В Packageexplorer у вас появится проект с исходным кодом.

2). Запуск приложения.

Нажмите на стрелку рядом с кнопкой «Run», в раскрывшемся списке выберите:

```
Runas->Androidapplication
```

В результате действий проект перестроится и запустится приложение. После загрузке эмулятора можно убедиться в работоспособности программы.

Датчик акселерометра. Для работы с обычными датчиками в Android API предназначен пакет `android.hardware`, содержащий такие классы как `Sensor`, `SensorManager` и `SensorEventListener`. Мы воспользуемся ими для работы с акселерометром в примере Sensor1. Класс `SensorManager` используется для доступа ко всем датчикам системы, создадим объект этого класса при создании нашей `Activity`:

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

Затем используем `SensorManager` для получения датчика желаемого типа:

```
mSensor                                          =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCEL
EROMETER);
```

Так же, как и в других ОС, создание объекта датчика не означает обязательного изменения состояния датчика аппаратного – датчик уже давно может быть активирован и использоваться другими приложениями, мы лишь создаем еще один приемник данных для него.

Мы можем подписаться на информацию от вновь созданного датчика прямо в методе `onCreate()`, где мы только что его создали, но правильнее делать это в методе `onResume()` и отписываться от датчика в методе `onPause()`. Это сэкономит ресурсы системы в случае, если она по каким-то причинам

деактивирует наше приложение. Подписка на данные датчика выглядит так:

```
sensorManager.registerListener(this, mSensor,

SensorManager.SENSOR_DELAY_NORMAL);
```

Третий параметр указывает желаемую скорость обновления данных – скорость, необходимая для игр, будет бесполезно нагружать систему в фоновом приложении.

Наконец, данные от датчика необходимо как-то обрабатывать. Для этого мы реализовали в `Activity` интерфейс `SensorEventListener` и его метод `onSensorChanged()`. Этот метод будет вызван системой только тогда, когда показания датчика изменятся, таким образом это реализация алгоритма прерываний, а не поллинга.

Единственным параметром метода является сообщение `SensorEvent`, перед обработкой необходимо проверить, что это сообщение именно от акселерометра (никто не мешает подписать один интерфейс на несколько разных датчиков):

```
if (event.sensor.getType() ==
Sensor.TYPE_ACCELEROMETER)
…
```

Если проверка прошла успешно, то можно использовать данные, упакованные в вещественный массив `event.values` – первые три значения в массиве дают соответственно ускорения по осям x, y, z устройства (рис. 15.3.1). В нашем примере мы вычисляем величину суммарного ускорения и определяем, во сколько раз она превосходит ускорение свободного падения:

```
float accelationSquareRoot = (x * x + y * y + z
* z)/

(SensorManager.GRAVITY_EARTH*SensorManager.GRAVI
TY_EARTH);
```

Если ускорение превосходит некий предел, то мы считаем, что устройство встряхнули, выводим сообщение об этом и меняем цвет экрана для наглядности. Для того, чтобы избежать множественных ложных срабатываний в момент встряхивания, после первого определения алгоритм отключается на 200 миллисекунд.

Рассмотрев основные детали построения приложения, перейдем к обзору кода.

В главном методе `onCreate()`,в самом начале функцией `getWindow` получаем текущее окно для `activity` с параметрами (флагами) во весь экран. Теперь зададим в качестве содержимого `Activity`, созданный нами интерфейс, с помощью вызова метода `setContentView` («Путь к файлу через класс ресурсов»). Очень часто при раздельном описании интерфейса и логики приложения, необходимо получить доступ к конкретному элементу интерфейса, чтобы назначить ему определенную функцию. В таком случае, пожалуй, лучшим способом является использование метода `findViewById` (идентификатор). Параметр идентификатор – это имя компонента, определенное в файле интерфейса. Следующими вызовами мы получаем поля по x, y и z, а `setBackgroundColor` задает цвет по умолчанию – зеленый. Так как класс `SensorManager` предоставляет доступ к сенсорам устройства, то методом `getSystemService` мы получаем экземпляр этого класса. И, естественно, чтобывыбрать одининтересующий нас сенсор, нам следующей строчкой нужно присвоить нашей переменной `mSensor - Accelerometer`, что делается вызовом метода `getDefaultSensor`. На этом метод `onCreate` завершен.

Так как мы хотим реализовать эффект «Shake» смены картинки при резком движении (в нашем случае `Red` на `Green` и наоборот) то необходимо реализовать соответствующие действия при изменении показаний сенсора. Именно поэтому, мы реализуем метод `onSensorChanged` интерфейса `SensorEventListener`. По просту говоря, этот интерфейс отвечает за получение уведомлений от класса `SensorManager` когда меняется значение сенсора и содержит в себе кроме `onSensorChanged` еще и `onAccurancyChanged` (который отвечает за точность изменений в показании датчика). Последний метод в данной лабораторной работе мы не реализуем потому, как на не интересует точность измерений сенсора.

Разберем метод `onSensorChanged`. В самом начале происходит проверка события сенсора на совпадение с интересующим нас датчиком сенсора (самый первый `if`), если все верно, то метод выполняется. Далее мы получаем (методом `event.values`) и выводим на экран значения координат. Далее чтобы организовать

эффект смены цвета мы осуществляем проверку «текущего времени (`actualTime`)» с «предыдущим временем (`lastUpdate`)». И если их разность меньше 200 милисекунд, то мы выходим из метода (мы снова зайдем в метод как только показания датчика изменяться), если же разность больше 200 милисекунд, то мы меняем (методом `SetBackgroundColor`) цвет с зеленого на красный. На этом метод благополучно завершен.

Дальше по коду мы стандартным образом задаем модель поведения для методов `onResume` и `onPause`. А именно в первом методе мы регистрируем наш класс как слушатель для ориентации и акселерометра, т. е для решаемой задачи (методом `registerListener`). Второй метод «снимает» регистрацию со слушателя событий.

Датчик GPS. Работа с датчиком GPS в Android отличается от работы с другими датчиками. Для геолокации в Android API предназначен пакет `android.location`, содержащий такие классы как `Location`, `LocationListener` и `LocationManager`. Специфика объясняется тем, что задача определения координат может решаться не только с помощью датчика GPS, но и другими способами, однако мы далее будем обсуждать только датчик GPS. Соответствующий пример называется Sensor2.

По аналогии с обычным датчиком, источниками геолокационной информации ведает класс `LocationManager`, создадим его при создании приложения:

```
lm = (LocationManager)
getSystemService(LOCATION_SERVICE);
```

По аналогичной схеме в методах `onResume()` и `onPause()` соответственно подпишемся и отпишемся от этого датчика:

```
lm.requestLocationUpdates(LocationManager.GPS_PR
OVIDER, 1000, 1f, this);
lm.removeUpdates(this);
```

Обратите внимание, что мы не создаем никакого отдельного объекта для датчика GPS, как мы это делали для акселерометра. Также сигнатура вызываемого метода не полностью аналогична `registerListener()` – в первом параметре передается имя поставщика геоданных (в данном случае это датчик GPS).

Для получения данных нужно реализовать интерфейс `LocationListener`. Его главный метод `onLocationChanged()`

вызывается всякий раз, когда изменяются координаты, параметром передаются новые:

```
public void onLocationChanged(Location location)
{
      …
      long = location.getLongitude();
      …
}
```

В отличие от обычных датчиков, датчик GPS может быть выключен и включен обратно пользователем. Для отслеживания этой ситуации в интерфейс добавлены методы `onProviderEnabled()` и `onProviderDisabled()`. Программист может использовать их для оповещения пользователя о состоянии датчика GPS.

Кроме того, датчик GPS может перестать работать без явной команды пользователя, например, внутри помещения или тоннеля. Для отслеживания этих изменений используется метод `onStatusChanged()`. Этот метод наиболее полезен для обработки ситуации временной неработоспособности датчика.

Рассмотрев основные детали построения приложения, перейдем к обзору кода.

Еще до метода `onCreate` происходит что-то интересное. Объявляется переменная `tab` нужная нам для ведения `logcat`, переменная `lm` (класса `LocationManager`) – менеджер для управления службой GPS и переменная `sb` (класса `StringBuilder`) – хранилище большого количества текста, как одна большая строка. Затем идет `onCreate`.

В нем мы сначала задаем в качестве содержимого `Activity`, созданный нами интерфейс, с помощью вызова метода `setContentView` («Путь к файлу через класс ресурсов»). Далее методом `findViewById` (идентификатор) инициализируем текстовое представление. Надо сказать, что параметр идентификатора – это имя компонента, определенное в файле интерфейса. Затем методом `getSystemService` мы инициализируем менеджер для доступа к GPS службе. На этом метод `onCreate` завершен.

Затем следуют методы `onResume` и `onPause`. В первом мы методом `requestLocationUpdates` добавляем слушатель события и получаем уведомление каждые 1000 милисекунд или 10 секунд. Во втором методе мы отключаем службу (методом `removeUpdates`) GPS, если приложение переходит в паузу.

Так как нам важны изменения показания датчика GPS («ловит – не ловит»), то мы используем интерфейс `LocationListener`, который отвечает за получения уведомлений от `LocationManager`, когда меняетсялокация. Таким образом мы должны реализовать четыре метода этого интерфейса: `onLocationChanged` (вызывается всякий раз, когда меняется местоположение), `onProviderDisabled` (вызывается, когда в приложении недоступен GPS, если устройство никогда не использовало службу GPS, то этот метод вызовется автоматически и предоставит пользователю возможность включить GPS с помощью интерфейса), `onProviderEnabled` (этот метод вызовется если приложению доступна служба GPS, или вызовется после того, как пользователь включит GPS) и `onStatusChanged` (вызывается всякий раз когда GPS программно терпит изменения (плохой сигнал, устройство «накрылось» и т. п., например, этот метод вызывается когда GPS служба не в состоянии установить местоположение или недавно стала доступной после периода недоступности).

Рассмотрим каждый из методов более подробно. `onLocationChanged` принимает в качестве аргумента локацию и в данной лабораторной метод осуществляет “сбор” информации в строку `sb` и вывод ее в `txtInfo`. Сбор информаии определяется стандартными функциями (например, `location.getLongitude` возвращает текущую долготу и т. д). В методе `onProviderDisabled` самое интересное это создание действия для вызова настроек, конкретно GPS-настроек. Вообще класс `Intent` служит для создания абстрактного описания операций, которые должны быть выполнены. Таким образом в конструкторе мы передаем `ImplicitIntents`, то есть неявные указания. Затем вызываем `activity` для данного описания. На этом `onProviderDisabled` закончился. Далее идет `onProviderEnabled`. Здесь нет ничего сложного – просто выводиться информация для текущего объекта исполнения о том, что GPS доступен. И в самом конце метод `onStatusChanged` –

здесь три case, каждый из которых выводит разную информацию о изменении Status на экран:

1 case – это вывод на экран Out of Service,

2 case – Temporarily Unavailable,

3 case – Available.

В конце идет onStop.

Датчик ориентации. Работа с датчиком ориентации показана в примере Sensor3.

Начнем разбор исходного кода приложения с рассмотрения файла графического интерфейса. Файл графического интерфейса в этом приложении достаточно простой. В нем определен LinearLayout (менеджер размещения с вертикально-позиционирующей политикой по умолчанию) а также три текстовых поля, которые отображают текущие показания датчика: азимут, угол относительно горизонтальной плоскости, угол относительно вертикальной плоскости. Каждому из текстовых полей присвоен идентификатор, чтобы в коде программы иметь возможность использовать данный компонент. На этом, пожалуй, завершим рассмотрение файла GUI.

Исходный код программы расположен в трех файлах OrientationListener.java, SensorOrientationActivity.java, OrientationManager.java, в отличие от предыдущих программ. Логика приложения разделена в нашем случае на три части: первые два файла описывают событийную модель, последний – реализует бизнес-логику приложения.

Начнем рассмотрение программы с файла OrientationListener.java. В этом файле определяется интерфейс для работы с датчиком. В файле описаны методы наиболее частого расположения устройства: onTopUp(), … . Смысл этих методов описан в коде. Также определен метод onOrientationChanged() – стандартный метод интерфейса, который вызывается, всякий раз, когда изменяются показания датчика ориентации устройства.

Рассмотрим теперь файл SensorOrientationActivity.java, в котором класс SensorOrientationActivity реализует рассмотренный ранее интерфейс. В классе мы определим переменную CONTEXT класса Context. Класс Context представляет общую информацию о среде приложения. Это

абстрактный класс, чья реализация обеспечивается операционной системой Android. С помощью него можно получить доступ к специфичным ресурсам и классам Android. В нашем приложении он содержит ссылку на объект `this` (`Activity`, которое реализует интерфейс). В классе `SensorOrientationActivity` определен метод `onCreate()`, в котором устанавливается `View`, и сохраняется ссылка `this`. В методах `onResume()` и `onDestroy()` происходят соответственно регистрация слушателя на событие «Ориентация», реализация бизнес-логики приложения (ответы на события) и отписка от события. В методе `onOrientationChanged()` в текстовые поля, определенные в файле “main.xml”, выводятся полученные данные с датчика. Реализация методов `onBottom()`, `onRight()`,… однообразна, при выполнении этих функций пользователю показывается всплывающее окно с текстовой информацией.

В классе `OrientationManager` описана основная логика программы. Класс содержит три переменные для реализации логики. В методе `startListening()` происходит инициализация объекта датчика-ориентации, с помощью класса `SensorManager`. Далее, мы регистрируем слушатель для события “Ориентация”. Объект класса `SensorEventListener` инициализируется с помощью анонимного конструктора, реализация которого очень наглядна и проста.

8.4. Задания для самостоятельной работы

1. Адаптировать пример работы с акселерометром так, чтобы он по-разному реагировал на встряхивания в разных направлениях или на последовательность встряхиваний.
2. Добавить в пример с GPS вывод протокола NMEA.
3. Добавить в пример с GPS вывод информации о доступных спутниках.