**Программирование жестов в Android**

**Как заменить кнопки на жесты смахивания в своем мобильном Android-приложении**

Пользовательский интерфейс мобильного приложения нужно строить, исходя из того, что мобильные пользователи часто отвлекаются, бывают заняты и ограничены в эргономическом плане. Эндрю Гловер объясняет ключевые факторы, отличающие мобильные приложения от Web-приложений, а затем показывает, как создать пользовательский интерфейс мобильного приложения, в котором кнопки уступают место жестам.

При создании мобильных приложений, как и при написании Web-приложений, программист должен ориентироваться на конечный результат (и пользователя). Постоянно держите в голове назначение своего приложения и способ работы с ним. Информацию какого типа отображает приложение, каковы его функции и как пользователи будут обращаться к тому и другому? Чтобы обеспечить успех, необходимо уделить достаточно внимания работе пользователей с этим мобильным приложением.

**Об этом цикле статей**

Мобильные приложения растут как грибы после дождя, и специалистам по их разработке есть где применить свои знания. Предлагаемый цикл статей знакомит с мобильными технологиями тех программистов, которые только начинают осваивать эту сферу. Мы начнем с написания "родных" приложений на языке Java, а затем добавим в инструментарий JVM-языки, платформы сценариев, инструменты HTML5/CSS/JavaScript и многое другое. Читатель, шаг за шагом, будет овладевать навыками, необходимыми для выполнения практически любого сценария разработки мобильных приложений.

В отличие от разработки традиционного GUI для настольных и Web-приложений, в случае мобильных приложений действует правило: *чем меньше, тем лучше*. Конструируя интерфейс мобильного приложения, *мыслите категориями простоты и легкости*. Большинство мобильных устройств имеет небольшие размеры (если только это не Samsung Note 4 — самый большой мобильный телефон – он же планшет, ― какой я когда-либо видел). Малый размер ― одна из причин повсеместного распространения этих устройств, которые везде носят с собой и могут использовать в любом месте. С этим связано еще одно важное наблюдение: в большинстве случаев внимание мобильного пользователя не приковано к приложению, с которым он работает!

Некоторые мобильные приложения проектируются специально для планшетов и предназначены для делового использования (например, врачами для обращения к историям болезни). В большинстве же своем мобильные приложения предназначены для крошечных устройств, пользователи которых одновременно делают что-то еще. Стоя в очереди за продуктами, я могу несколько раз сыграть в Angry Birds. Время высадки из самолета после продолжительного полета можно использовать для того, чтобы проверить электронную почту. Однако если для загрузки злых птиц или сообщений электронной почты недостаточно пары жестов пальцами, то я, скорее всего, не стану с этим возиться.

Еще одно, что заметно отличает мобильные приложения от традиционных Web- и настольных приложений, ― это их количество: на каждое Web-приложение легко наберется 100 мобильных. Ваше приложение должно предлагать какой-то действительно стоящий сервис. Оно должно быть простым и привлекательным. Если вы ожидаете, что пользователь станет читать руководство, то создадите проблему для него, которая в конечном итоге станет проблемой для вас. И не важно, на кого рассчитано приложение ― на врача, которому нужно получить доступ к истории болезни, или на того, кто играет в "Разрежь веревку" в зале ожидания. Если настройка мобильного приложения занимает больше пары минут, то пользователи, скорее всего, отправятся в app store за другим.

**(Hello) Overheard Word**

Те, кто прочел [первую статью](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses1/index.html) этого цикла, знают, как настроить среду разработки Android в Eclipse, и уже настроили свой Android SDK для платформы Android версии 4.2. Они разработали свое первое, простое Android-приложение ― разновидность классического Hello World. В этом месяце мы продолжим этот путь и разработаем более оригинальное приложение.

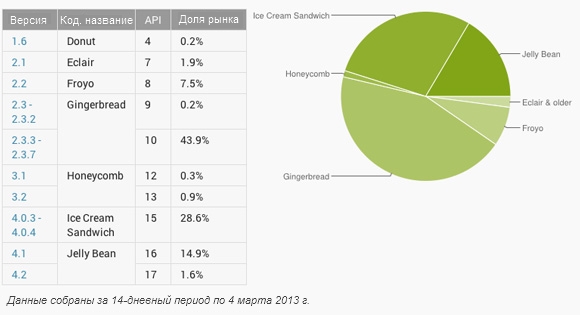
Мой пример приложения, Overheard Word (Подслушанное слово), предназначен для того, чтобы весело и легко разучивать новые слова и практиковать их использование в контексте — кстати, это два моих любимых занятия. Пользователь приложения узнаёт несколько новых слов и отвечает на вопросы викторины. Интерфейс состоит из экрана дисплея и двух кнопок. На дисплей выводятся слова и соответствующие определения, а кнопки служат для навигации.

"Подслушанное слово" ― это веселое и простое мобильное приложение для тех, кто любит играть в слова (я имею в виду*поклонников* лексики и *ценителей* слова). Еще важнее то, что это пример создания легитимного Android-приложения, которое можно развернуть на реальном Android-устройстве.

**Целевая аудитория**

Прежде чем разрабатывать приложение, хотелось бы оценить его целевой рынок. Приложение, созданное нами в первой статье, нацелено на Android 4.2, или API версии 17. Взгляните на последнюю таблицу распространения версий Android, предоставленную Google (см. рисунок 1).

**Рисунок 1. Распределение версий Android**



**Планшеты против телефонов**

Наиболее вероятная причина роста числа пользователей Android версии 15 (см. [Рисунок 1](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses2/index.html#fig1)) ― успешные продажи планшетов. В настоящее время большинство планшетов, независимо от производителя, работает с Android 4.x. Если только вы специально не создаете приложение для Android-планшета, то ориентируйте его на API Android версий 9 и 10, на долю которых в настоящее время приходится наибольшая часть рынка мобильных устройств.

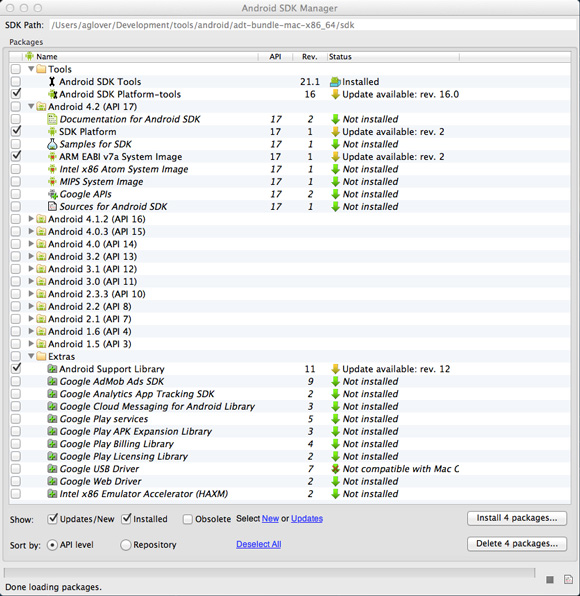
[Рисунок 1](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses2/index.html#fig1) показывает, что сегодня примерно половина всех Android-устройств работает на Android 2.3.x (API версий 9 и 10)! Так не лучше ли разработать приложение именно для этого рынка?

[**В начало**](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses2/index.html#ibm-pcon)

**Начинаем новый проект**

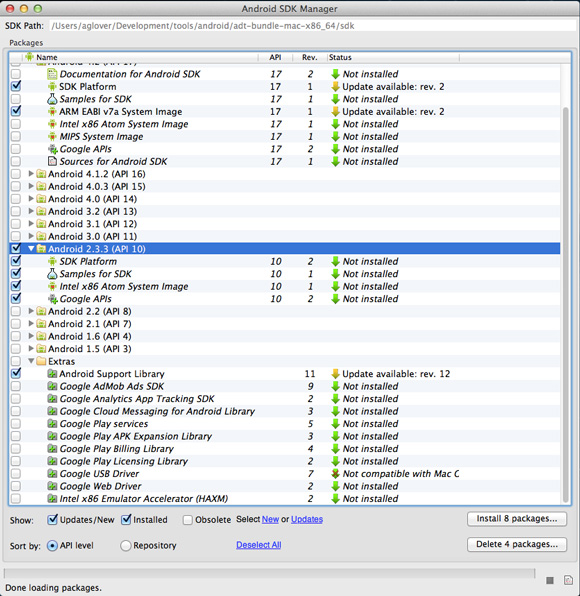
В прошлом месяце мы установили API версии 4.2. Теперь давайте установим Android 2.3.3 (API 10). Запустив Android SDK Manager, как показано на рисунке 2, видим несколько готовых к установке обновлений.

**Рисунок 2. Локальная установка Android SDK с единственной установленной версией**



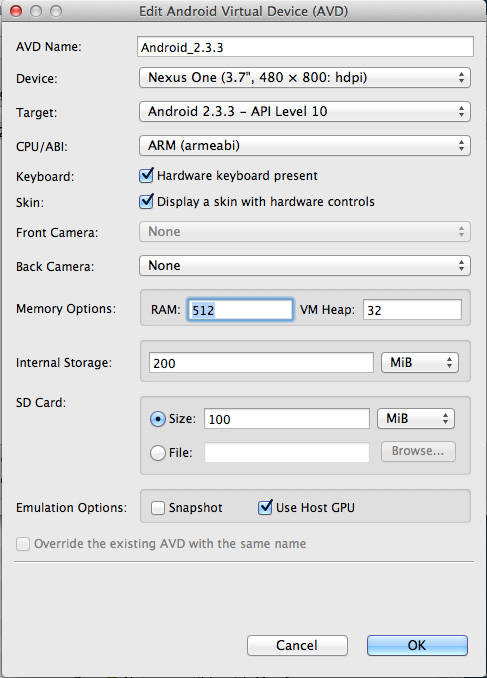
Если вы решили следовать за мной, выберите вариант **Android 2.3.3 (API 10)** и сохраните все рекомендуемые обновления, указанные менеджером.

**Рисунок 3. Установка Android 2.3.3 (API 10)**



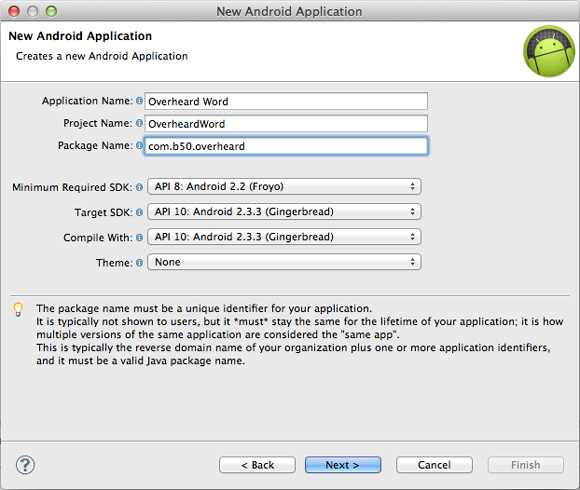
Загрузив новый API для Android, необходимо создать совместимый эмулятор, или виртуальное устройство Android (AVD). В Android SDK Manager выберите меню **Tools** с параметром **Manage AVDs**. Убедитесь в том, что вы создаете AVD, ориентированное на API уровня 10.

**Рисунок 4. Создание AVD**



Далее, создайте проект **New Android Application** и присвойте ему имя. Мой проект называется Overheard Word. Сориентируйте приложение на API 10, чтобы сделать его доступным для максимального количества устройств. Рассмотрите [Рисунок 5](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses2/index.html#fig5) и обратите внимание на то, что я также выбрал компиляцию с API 10. Компилировать можно и с новой версией Android, но я предпочитаю программировать и компилировать с одним и тем же API. (Если компилировать с более поздним API, то нужно следить за тем, чтобы не использовались никакие его возможности, которые могут создать проблемы совместимости.)

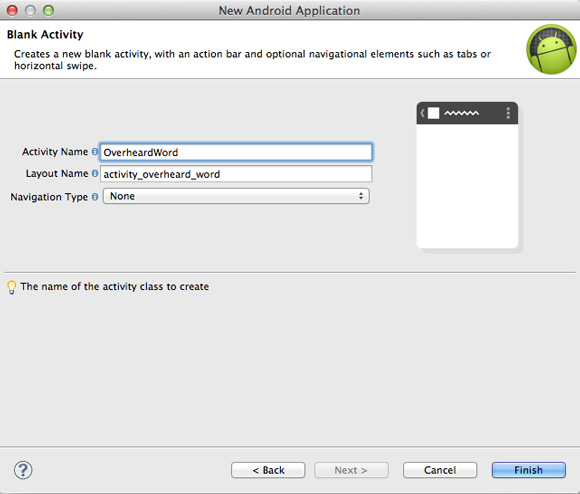
**Рисунок 5. Создание нового проекта Android в Eclipse**



Нажав кнопку **Next**, не делайте свое приложение библиотекой проекта. Сохраните [те же значения по умолчанию, что и для приложения Hello World](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses1/index.html#N100F7). Вы всегда сможете внести небольшие изменения (например, заменить файл ярлыка приложения).

Последний шаг установки ― присвоение имени своему первому действию. Я назвал его OverheardWord, а не Main, и то же имя использовал для макета. Если вы тоже используете API 10 в качестве своей целевой версии, то в заключительном диалоговом окне у вас будет несколько вариантов выбора способа навигации (**Navigation Type**). Пока не беспокойтесь об этом; я покажу, как создать меню навигации, в одной из последующих статей. Для этого же приложения оставьте для параметра **Navigation Type** значение **none**.

**Рисунок 6. Создание действия по умолчанию**



Нажмите кнопку **Finish,** чтобы создать проект, который выглядит так же, как проект по умолчанию HelloWorld, с которым мы работали в [прошлый раз](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses1/index.html).

[**В начало**](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses2/index.html#ibm-pcon)

**Создание UI Android**

Теперь можно создать простой пользовательский интерфейс. Как вы, наверное, помните, UI Android в конечном итоге определяется как XML-документ, который вам, вероятно, покажется знакомым. Android использует те же *макеты*, которые применяют при создании традиционного GUI Java™. В макетах Android можно определить способ отображения устройством визуальных компонентов приложения. Стиль макета может быть:

* *линейным*, когда компоненты организованы по горизонтали или по вертикали (как колонки в газете);
* *относительным*, когда компоненты располагаются друг относительно друга (виджет 2 справа от виджета 1 и т.д.);
* *табличным*, когда компоненты отображаются в столбцах и строках.

Бывают и другие стили макета, но для начала достаточно этих трех.

В макете можно определить *виджеты* ― обычные компоненты, которые можно найти в любом GUI. Платформа Android предлагает основные виджеты, такие как кнопки, текстовые поля и раскрывающиеся списки, а также более сложные ― для просмотра картинок, вращения и т.п.

Как я уже говорил, для пользовательского интерфейса Overheard Word требуются:

* три текстовых поля (для хранения слова, названия соответствующей ему части речи и его определения);
* две кнопки (для выбора нового слова и ответа на вопрос).

**Определение макета**

Первым шагом к созданию пользовательского интерфейса Android является определение макета с виджетами. В листинге 1 мы сначала определяем макет типа LinearLayout с тремя виджетами TextViews. Затем создаем подмакет (также типа LinearLayout) с двумя кнопками Button.

**Листинг 1. Определение макета LinearLayout в Android**

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/LinearLayout1"

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=".OverheardWord" >

<TextView

android:id="@+id/word\_study\_word"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginBottom="10dp"

android:layout\_marginTop="60dp"

android:textColor="@color/black"

android:textSize="30sp" />

<TextView

android:id="@+id/word\_study\_part\_of\_speech"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginBottom="10dp"

android:layout\_marginLeft="20dp"

android:layout\_marginRight="20dp"

android:textColor="@color/black"

android:textSize="18sp" />

<TextView

android:id="@+id/word\_study\_definition"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginLeft="40dp"

android:layout\_marginRight="40dp"

android:textColor="@color/black"

android:textSize="18sp" />

<LinearLayout

android:id="@+id/widget62"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginLeft="5dp"

android:layout\_marginRight="5dp" >

<Button

android:id="@+id/next\_word"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginTop="20dp"

android:text="@string/next\_word" />

<Button

android:id="@+id/take\_quiz"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_marginLeft="50dp"

android:layout\_marginTop="20dp"

android:text="@string/take\_quiz" />

</LinearLayout>

</LinearLayout>

Как видно по XML в листинге 1, каждый виджет имеет ряд свойств, которые влияют на его внешний вид. Например, у каждой из двух кнопок есть элемент text, строка которого указывает на *элемент ресурсов*. Этот элемент определяется в *файле ресурсов* с именемstrings.xml, который хранится в папке res/values. [Листинг 2](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses2/index.html#listing2) иллюстрирует файл strings.xml.

**Листинг 2. Файл ресурсов, определяющий цвет кнопок и надписи на них**

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">Overheard Word</string>

<color name="white">#ffffff</color>

<color name="black">#000000</color>

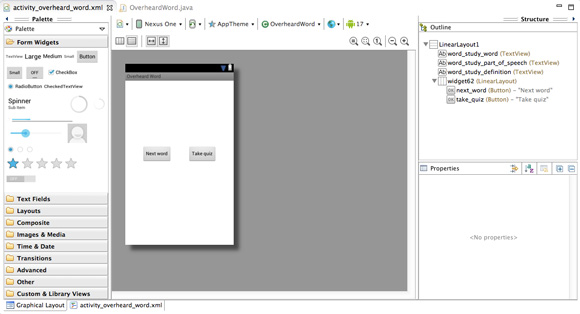
<string name="next\_word">Next word</string>

<string name="take\_quiz">Take quiz</string>

</resources>

Чтобы увидеть, как выглядит UI на данный момент, можно запустить экземпляр эмулятора. Но мы вместо этого выберем вкладку**Graphical Layout** в Eclipse, как показано на рисунке 7.

**Рисунок 7. Работа Overheard Word в Eclipse**



**Замещающий текст**

При просмотре пользовательского интерфейса несколько смущает пустое пространство вместо атрибутов слова. Это не проблема: прежде чем приступить к программированию, добавим образец текста для более реалистичного представления.

Сначала добавим к каждому из виджетов TextView атрибут android:text, так же, как я это сделал для надписей на кнопках. Пока добавим замещающий текст.

**Листинг 3. Замещающий текст для android:text**

<TextView

android:id="@+id/word\_study\_part\_of\_speech"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:layout\_marginBottom="10dp"

android:layout\_marginLeft="20dp"

android:layout\_marginRight="20dp"

android:textColor="@color/black"

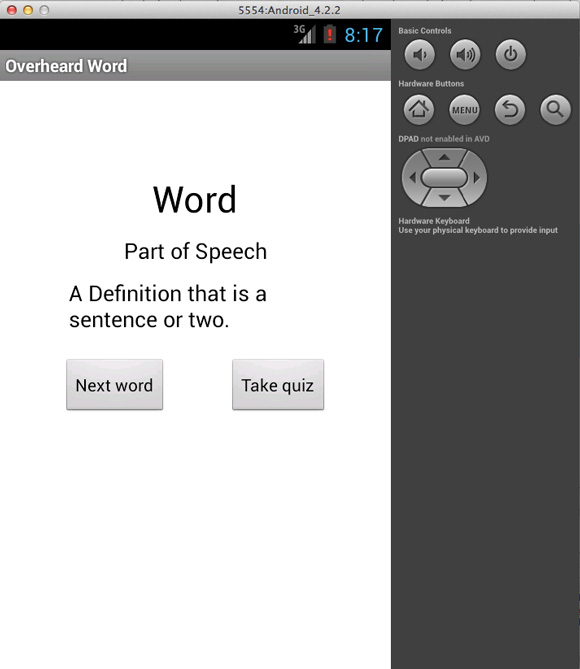
android:textSize="18sp"

android:text="Part of Speech"/>

Как видно из листинга 3, я добавил в виджет TextViewword\_study\_part\_of\_speech образец текста ("Part of Speech"). И обновил некоторые из элементов стилевого оформления, увеличив размер текста, определив его цвет и отцентрировав виджет в макете. Атрибуты в листинге 3 ― общие для определения виджетов в Android-приложении; создавая другие приложения, вы узнает и полюбите их. Я определил образец текста непосредственно в своем файле макета, а не в файле ресурсов, потому что это временные текстовые значения, используемые только в процессе проектирования.

При запуске приложения в Eclipse будет видно, что увидят пользователи.

**Рисунок 8. Работа Overheard Word в AVD**



В дизайне пользовательского интерфейса все еще что-то не так. Догадываетесь, что именно? Да-да, эти неуклюжие кнопки! Необходимы ли они? Часто ли пользователь нажимает кнопки в мобильных приложениях? Не удобнее ли смахнуть пальцем влево или вправо, чтобы увидеть новые слова, и вниз, чтобы ответить на вопрос?

Удалив кнопки, мы очистим свой UI и лучше приспособим приложение Overheard Word к ожиданиям большинства пользователей мобильных устройств. Как вы скоро увидите, жесты смахивания реализуются довольно легко.

**Программирование жестов в Android**

Мы определили начальный экран приложения Overheard Word и теперь готовы добавить некоторое поведение. Опыт разработки традиционных Java GUI и здесь применим непосредственно к Android ― сначала реализуем прослушивающие процессы, которые реагируют на события. Если бы мы оставили две кнопки, то могли бы использовать метод OnClickListener для присвоения действия каждому нажатию кнопки, а затем добавить его к методу setOnClickListener виджета каждой кнопки.

**Анонимные классы**

Анонимные классы — это классы, локальные по отношению к данному классу, но без имени: можно определять анонимные классы*и создавать их экземпляры* в одном выражении. Я считаю их удобными и элегантными и часто использую, особенно при программировании поведения интерфейса пользователя.

Хотя, безусловно, можно использовать OnClickListener, я, скорее, склоняюсь к нескольким файлам классов для разных виджетов. Мне не нравится весь этот беспорядок, и я предпочитаю альтернативную стратегии реализации анонимных классов.

В традиционном программировании GUI понятие жестов, таких как *смахивание*, отсутствует, однако их реализация аналогична программированию нажатия кнопки. Начнем с *процесса прослушивания касания*, который входит в состав базовой платформы Android, а затем добавим соответствующее поведение. Для обнаружения движения смахивания пальцем нужно вычислить разность между координатами *начала* и *конца* жеста, а также его скорость. (Я нахожу здесь полезными декартовы координаты: смахивание влево и вправо происходит по оси X, а вверх и вниз ― по оси Y.)

Жест может происходить в любом месте интерфейса. При подключении такого процесса прослушивания к любому компоненту интерфейса приложения начинайте с макета (который в Android представлен как View) и добавьте самый простой виджет кнопки. Я использую метод Android SimpleOnGestureListener для создания начальной реализации жеста, а затем подключаю его к View с помощью метода onTouchListener. Подключившись к экземпляру своего представления, я могу обнаруживать жесты смахивания по*всему* интерфейсу, а не только в пределах его компонента, такого как компонент для просмотра изображений.

Однако предварительно нужно написать класс, способный обнаруживать движение пальцев на мобильном устройстве.

**SwipeDetector**

Обнаружить нажатие кнопки относительно легко. Операционная система обычно абстрагирует координаты границ кнопки, чтобы программисту не нужно было заниматься их определением вручную. Программирование распознавания движения смахивания и реагирования на него в приложении немного сложнее. Android многое перекладывает на программиста, зато оставляет вам огромную свободу. Вы можете обнаруживать один жест, как мы делаем здесь. Но можете обнаруживать и другие движения пальцев, такие как потирание, которое иногда используют в играх.

**Мышление мобильными категориями**

Как ни странно, процесс прослушивания мобильного приложения определяется глаголом *click* (щелкать). Пользователи мобильных устройств *нажимают*кнопки, а не *щалкают* на них, как в случае настольной мыши. Разработка для мобильных устройств предполагает мышление мобильными категориями, что означает переосмысление традиционных Web- и настольных механизмов, таких как нажатие кнопок, меню Пуск и даже мастера GUI.

Для программирования обнаружения жеста смахивания приходится прибегать к геометрии. Если *начальная координата* касания по оси X минус *конечная координата*по той же оси превышает заданное *расстояние*, и если *скорость* этого движения превышает другое заданное значение, то есть все основания предположить, что пользователь сделал жест смахивания пальцем справа налево. Смахивание вверх и вниз рассчитываются аналогичным образом, но по оси Y.

Подобную логику легко абстрагировать в простой класс, который выполняет элементарные расчеты. SimpleOnGestureListener содержит метод onFling. МетодonFling принимает два значения типа MotionEvent (*начало* и *конец*), представляя значений скорости по оси X, и два значения типа float ― для представления значений скорости по оси Y. Таким образом, классSwipeDetector в листинге 4 содержит четыре метода, которые разграничивают весь диапазон движений смахивания: влево, вправо, вверх и вниз.

**Листинг 4. SwipeDetector**

import android.view.MotionEvent;

public class SwipeDetector {

private int swipe\_distance;

private int swipe\_velocity;

private static final int SWIPE\_MIN\_DISTANCE = 120;

private static final int SWIPE\_THRESHOLD\_VELOCITY = 200;

public SwipeDetector(int distance, int velocity) {

super();

this.swipe\_distance = distance;

this.swipe\_velocity = velocity;

}

public SwipeDetector() {

super();

this.swipe\_distance = SWIPE\_MIN\_DISTANCE;

this.swipe\_velocity = SWIPE\_THRESHOLD\_VELOCITY;

}

public boolean isSwipeDown(MotionEvent e1, MotionEvent e2, float velocityY) {

return isSwipe(e2.getY(), e1.getY(), velocityY);

}

public boolean isSwipeUp(MotionEvent e1, MotionEvent e2, float velocityY) {

return isSwipe(e1.getY(), e2.getY(), velocityY);

}

public boolean isSwipeLeft(MotionEvent e1, MotionEvent e2, float velocityX) {

return isSwipe(e1.getX(), e2.getX(), velocityX);

}

public boolean isSwipeRight(MotionEvent e1, MotionEvent e2, float velocityX) {

return isSwipe(e2.getX(), e1.getX(), velocityX);

}

private boolean isSwipeDistance(float coordinateA, float coordinateB) {

return (coordinateA - coordinateB) > this.swipe\_distance;

}

private boolean isSwipeSpeed(float velocity) {

return Math.abs(velocity) > this.swipe\_velocity;

}

private boolean isSwipe(float coordinateA, float coordinateB, float velocity) {

return isSwipeDistance(coordinateA, coordinateB)

&& isSwipeSpeed(velocity);

}

}

Теперь, когда у нас есть удобный класс, говорящий, что выполнен тот или иной жест, его можно включить в наш пользовательский интерфейс. Напомню, что вначале мы создали действие по умолчанию, которое назвали OverheardWord. Мы сохраним все детали кода по умолчанию, которые дает Eclipse, но кое-что добавим. Для прослушивания действия смахивания и реагирования на него определим метод Android GestureDetector, который принимает реализацию OnGestureListener. К счастью, разработчики Android создали удобный класс, который можно реализовать через анонимный класс SimpleOnGestureListener. Потом мы переопределим метод onFling.

**Обнаружение смахивания в пользовательском интерфейсе**

В методе действия onCreate приступаем к созданию детектора жестов GestureDetector с процессом SimpleOnGestureListener, в котором используется класс SwipeDetector.

Следующим шагом нужно написать код сообщения для указания того, что событие по жесту смахивания сработало. Такое маленькое, короткоживущее сообщение в Android называется Toast В некоторых случаях Toast может состоять всего из одной строки кода.

**Листинг 5. GestureDetector**

private GestureDetector initGestureDetector() {

return new GestureDetector(new SimpleOnGestureListener() {

private SwipeDetector detector = new SwipeDetector();

public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,

float velocityY) {

try {

if (detector.isSwipeDown(e1, e2, velocityY)) {

return false;

} else if (detector.isSwipeUp(e1, e2, velocityY)) {

showToast("Up Swipe");

}else if (detector.isSwipeLeft(e1, e2, velocityX)) {

showToast("Left Swipe");

} else if (detector.isSwipeRight(e1, e2, velocityX)) {

showToast("Right Swipe");

}

} catch (Exception e) {} //for now, ignore

return false;

}

private void showToast(String phrase){

Toast.makeText(getApplicationContext(), phrase, Toast.LENGTH\_SHORT).show();

}

});

}

Теперь у нас есть метод initGestureDetector, который вычисляет диапазон и скорость движения смахивания. Во-первых, он создает анонимный экземпляр SimpleOnGestureListener и обеспечивает реализацию метода onFling. Заметьте, как методonFling использует SwipeDetector. Например, если происходит распознавание смахивания вверх, то Toast указывает на это движение.

Следующий шаг ― включение действия обнаружения жеста смахивания в пользовательский интерфейс. Начнем с регистрации нескольких прослушивающих процессов с экземпляром View.

**Листинг 6. Регистрация обнаружения жеста смахивания в представлении View**

private GestureDetector gestureDetector;

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_overheard\_word);

gestureDetector = initGestureDetector();

View view = findViewById(R.id.LinearLayout1);

view.setOnTouchListener(new View.OnTouchListener() {

public boolean onTouch(View v, MotionEvent event) {

return gestureDetector.onTouchEvent(event);

}

});

view.setOnClickListener(new OnClickListener() {

public void onClick(View arg0) {

}

});

}

В методе onCreate я получаю дескриптор для фактического экземпляра View посредством его Id. Большинство Java-разработчиков оценит эту удобную особенность *Android*.

**Метод поиска представления по ID**

Помните, как мы определили XML-файл для своего пользовательского интерфейса вместе с соответствующими значениями в различных файлах ресурсов (таких как strings.xml)? При компиляции Android-приложения этот XML-файл превращается в сгенерированный код, и ему присваиваются соответствующие Id в классе R. Прежде чем пойти дальше, проверьте класс R в каталогеgen своего проекта.

Посмотрите на XML-файл макета (как вы помните, я назвал его activity\_overheard\_word.xml). В этом XML-файле у каждого виджета есть атрибут id. Например, id макета моего приложения будет android:id="@+id/LinearLayout1", или для краткостиLinearLayout1. Имя Id Eclipse генерирует автоматически, но его можно изменить. Важно, что у idLinearLayout1 есть соответствующее свойство в классе R, как показано в листинге 7.

**Листинг 7. Идентификаторы виджетов в классе R**

public static final class id {

public static final int LinearLayout1=0x7f090000;

public static final int action\_settings=0x7f090004;

public static final int word\_study\_definition=0x7f090003;

public static final int word\_study\_part\_of\_speech=0x7f090002;

public static final int word\_study\_word=0x7f090001;

}

Объединяя содержимое XML-файла с соответствующим файлом R, можно вызывать виджеты без необходимости анализа XML. Поэтому, возвращаясь к методу onCreate, можно вызвать виджет View по его id: R.id.LinearLayout1. Метод findViewByIdпредоставляется платформой Android при расширении Activity.

**Смахивание пальцем работает!**

Когда у нас есть экземпляр View, к нему можно добавить свой экземпляр gestureDetector посредством метода setOnTouchListener. В листинге 6 еще один анонимный класс теперь обрабатывает все действия касания. Когда пользователь касается экрана устройства, это событие срабатывает и воздействует на gestureDetector. Я также реализовал OnClickListener и установил его в методе setOnClickListener; но в этом случае у анонимного класса нет поведения.

В результате получился совсем не сложный интерфейс, который реагирует на жесты смахивания пальцем. Таких жестов три: влево, вправо и вверх. Попробуйте этот код в своем AVD. Для имитации смахивания нажмите кнопку мыши и перетащите ее влево, вправо или вверх.

Теперь пора проверить это приложение в действии на реальном устройстве.

[**В начало**](http://www.ibm.com/developerworks/ru/library/j-mobileforthemasses2/index.html#ibm-pcon)

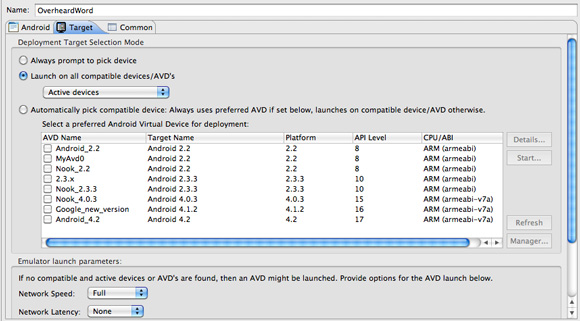
**Развертывание в процессе разработки**

В ответ на распространение вирусов, троянских коней и вредоносных программ, этого бича ничего не подозревающих пользователей, Apple и Google ввели форму подписи кода сторонних приложений, работающих на их устройствах. Теоретически подписи служат гарантией, что данное приложение ― из авторитетного источника и что никто не изменил двоичный файл, который вы устанавливаете. На практике же коды подписи Google и Apple различаются. Например, подписывая Android-приложение, я могу утверждать, что я ― известный государственный банк. Подписывая приложение для iOS или Windows Phone 8, этого не сделаешь.

В режиме разработки о подписании кода можно не беспокоиться. Android и Eclipse создают подписанный двоичный код с ключом разработчика, не известным программисту. Так что поместить приложение на устройство в режиме разработки нетрудно. Простейший способ ― подключить Android-устройство к порту USB своего компьютера.

Затем перейдите в Eclipse, щелкните правой кнопкой мыши на проекте и выберите **Run As menu**, а затем **Run Configurations**. Eclipse отображает диалоговое окно конфигурации, где можно выбрать проект и перейти на вкладку Target, как показано в листинге 9.

**Рисунок 9. Выбор устройства для запуска приложения**



Можно выбрать режим **Always prompt to pick device** (всегда предлагать выбрать устройство) или **Launch on all compatible devices/AVD's** (запустить на всех совместимых устройствах/AVD's). В последнем случае выберите параметр **Active Devices** и нажмите кнопку **Run**. Через несколько мгновений на вашем устройстве появится симпатичное приложение. Смахните пальцем влево, вправо, вверх, и вы увидите простые сообщения, подтверждающие, что все работает.

**Параметры безопасности**

Если из системы электронной почты или Dropbox приложение загрузить не удается, перейдите в окно настройки устройства и включите режим **Unknown Sources** (неизвестные источники) в разделе **Security Settings** (параметры безопасности). После этого изменения можно будет устанавливать приложения*не* из Google Play, в том числе те, которые вы разрабатываете.

Еще один способ установки приложения ― отправить себе файл .apk по электронной почте, а затем открыть его на устройстве Android и следовать инструкциям в диалоговом окне автоматической установки. Иначе, можно загрузить файл .apk в такую службу, как Dropbox, затем открыть Dropbox на своем устройстве и установить приложение оттуда. Файл .apk приложения находится в каталоге bin проекта, куда Eclipse также помещает соответствующие двоичные файлы.

Имейте в виду, что эти механизмы установки предназначены лишь для тестирования в процессе разработки, а не для целей распространения. Для коммерческого распространения приложения требуется несколько дополнительных шагов, о которых я расскажу в следующей статье.

**Заключение**

При разработке мобильных приложений нужно мыслить категориями простоты и легкости. В этом месяце вы узнали, как запрограммировать мобильное приложение, чтобы оно реагировало на жесты смахивания, а не кнопки, привычные пользователям настольных и Web-приложений. В некоторых случаях мобильное приложение все же может иметь одну-две удобные кнопки. Но они должны в точности соответствовать задаче пользователя. Для нашего сверхпростого приложения Overheard Word кнопки не нужны, а разработанные нами жесты смахивания пальцем вверх-вниз интуитивно понятны большинству пользователей мобильных приложений.

Мы кратко рассмотрели здесь процесс развертывания мобильного приложения, но дошли только до тестирования — вам еще предстоит освоить коммерческое распространение приложений. Большинство пользователей не устанавливает приложения из ненадежных источников, поэтому наиболее успешные приложения распространяются через посредника, такого как Google Play, iTunes App Store или Amazon Appstore для Android. В следующей статье я покажу процесс, необходимый для соблюдения стандартов безопасности магазинов приложений. Этот процесс идет несколько дальше простого подписания приложения. Зато когда ваше приложение размещено у крупного дистрибьютора, оно становится доступным *всему миру*.