万人のためのモバイル: スワイプしてみよう！Android でのジェスチャーのプログラミング

Android モバイル・アプリのボタンをスワイプに交換する

Andrew Glover

2013年 6月 27日

たいていのモバイル・ユーザーは、他のことに気を取られていて、いろいろと忙しい上に、人間工学的に制約されています。そんなモバイル・ユーザーに合わせてモバイル・アプリの UI を作成してください。著者の Andrew Glover はこの記事で、モバイル・アプリが Web アプリケーションとは異なる主な点を取り上げた後、ボタンをクリックする代わりに、スワイプ・ジェスチャーでナビゲートするモバイル・アプリの UI を作成する方法を説明します。

このシリーズの他の記事を見る

はじめに

この連載について

モバイル・アプリの配布が急増するなか、モバイル開発のスキルを要する市場が急成長しています。この developerWorks の連載では、プログラミングの経験は積んでいても、モバイルの世界でのプログラミングには馴染みのない開発者を対象に、モバイルの世界でのプログラミングをわかりやすく紹介します。

まず Java コードでネイティブ・アプリを作成することから始め、JVM 言語、スクリプティング・フレームワーク、HTML5/CSS/JavaScript、サード・パーティー・ツールなどを使いこなせるようにモバイル開発のスキルを磨いていきます。ステップ・バイ・ステップで、ほとんどどのようなモバイル開発シナリオにも対応できるだけのスキルをマスターしてください。

モバイル・アプリを作成するときには、Web アプリケーションを作成するときと同じく、常に最終的な結果 (そしてユーザー) を意識してください。作成するアプリの目的を理解し、人々がどのようにそのアプリを利用するかを念頭に置いて、アプリが表示する情報のタイプ、アプリに持たせる機能、そしてユーザーがこの情報と機能にアクセスする手段について考える必要があります。作成するモバイル・アプリを成功に導くには、アプリのユーザー・エクスペリエンスに十二分の注意を払うことが不可欠です。

デスクトップ・アプリケーションや Web アプリケーション向けの従来の GUI 開発とは異なり、モバイル・アプリでのルールは、「少ないほうが、より多くを得られる」です。モバイル・アプリのインターフェースを設計するときには、「シンプルかつ容易に使えるものを考える」ようにしてください。ほとんどのモバイル端末はこぢんまりとしたサイズです (Samsung Note 4 を扱っているとしたら話は別です。Samsung Note 4 は今まで私が目にしたなかで最も大きな携帯電話で、タブレットと呼んだ方がよいのかもしれません)。サイズが小さいことは、モバイル端末が普及するのに欠かせない要素であり、サイズが小さいがために、人々は常にモバイル端末を持ち歩き、場所を問わずに使用するようになっています。このことから、モバイル・アプリに関するもう 1 つの重要な見解が導かれます。それは、ほとんどのユーザーはアプリを使うときに、それだけに専念してはいないということです！

モバイル・アプリのなかには、ビジネス・ユース (例えば、医者が患者の記録にアクセスする場合など) のみを目的としてタブレット専用に作成されているものもあります。しかし大半のモバイル・アプリが想定しているユーザーは、率直に言ってしまえば、小型の端末で何か他のことをしながらアプリを利用するユーザーです。私の例で言えば、食料品店で精算するために並んで待っている間、Angry Birds (訳注: Angry Birds は iPhone 向けに開発され、後に Android 版と PC 版がリリースされたモバイル・ゲーム) で暇つぶしをしようとするかもしれません。長時間のフライトの後、飛行機から降りるまでの時間を利用して、e-メールをチェックする可能性もあります。けれども Angry Birds を立ち上げたり、e-メール·メッセージを開いたりするまでに何回もタップやスワイプをしなければならないとしたら、それらを使うのはあきらめてしまうと思います。

従来の Web アプリケーションやデスクトップ・アプリケーションと、モバイル・アプリとで明らかに異なるもう 1 つの点は容量です。どの Web アプリケーションでも、その容量は優にモバイル・アプリ 100 個分に相当します。モバイル・アプリを作成する場合は、使う価値があるサービスを提供するにとどまらず、それ以上のものを提供してください。使いやすいだけにとどまらず、使わずにはいられないようなモバイル・アプリにしてください。アプリを使用するユーザーに RTM (Read The Manual: マニュアルを読むこと) を期待するようであれば、ユーザーにとって面倒な問題を作り出しているということであり、結局は自分の問題になって返ってきます。それはアプリのユーザーが患者の記録にアクセスする医者であろうと、待合室での時間を Cut-the-Rope (訳注: Cut-the-Rope は iOS 向けに開発され、後に他の OS 向けにもリリースされた物理パズル・ゲーム) でつぶす人であろうと変わりません。モバイル・アプリを最初に立ち上げるまでに何分もかかるとしたら、ユーザーはアプリケーション・ストアでそれよりも短時間で立ち上げられるアプリを探すことでしょう。

Overheard Word アプリ

この新しい連載の第 1 回では、Eclipse に Android 開発環境をセットアップする方法を学び、Android バージョン 4.2 用の Android SDK を構成しました。また、最初の基本的な Android アプリとして、昔ながらの Hello World! のバリエーションを作成しました。今回はその過程の続きとして、前回に比べると少し変わったアプリを設計します。

この記事のサンプル・アプリである Overheard Word は、楽しく簡単に新しい単語を学んで、文脈に沿った使い方を練習するためのアプリです (偶然にも、これらは私の好きなことのうちの 2 つです)。このアプリでは、ユーザーがいくつかの単語を学習してからクイズに挑戦します。インターフェースは、ディスプレイ画面と 2 つのボタンからなります。ディスプレイには単語とその定義を表示する一方、2 つのボタンはユーザーがアプリをナビゲートするために使用します。

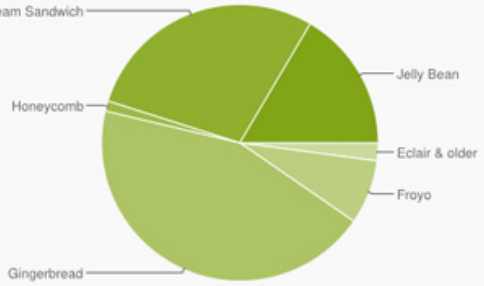
Overheard Word は、言葉に関心がある人 (つまり、ボキャブラリーが豊富な人や単語をよく知っている人) のためのシンプルで楽しいモバイル・アプリですが、さらに重要なこととして、このアプリは正真正銘の Android アプリを作成するサンプルであるため、実際の Android 端末にデプロイすることができます。

アプリのターゲットを定める

アプリを設計する前に、アプリがターゲットとする市場を判断したいと思います。第 1 回で作成したアプリのターゲットは、Android 4.2 (API バージョン 17) でした。Google で報告している最新の Android の配布状況を調べてください (図 1 を参照)。

図 1. バージョン別 Android 配布状況

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	1.9%
2.2	Froyo	8	7.5%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	43.9%
3.1	Honeycomb	12	0.3%
3.2		13	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	28.6%
4.1	Jelly Bean	16	14.9%
4.2		17	1.6%



Data collected during a 14-day period ending on March 4, 2013

タブレットとスマートフォン

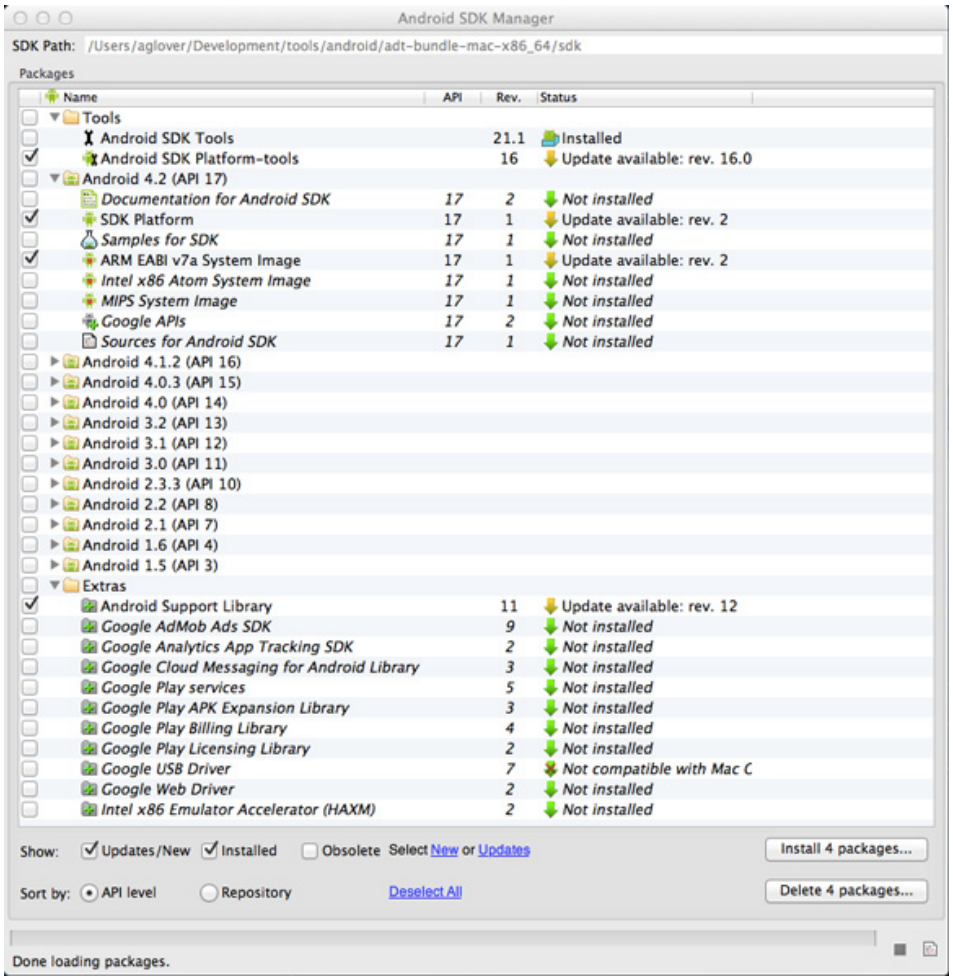
図 1 で Android バージョン 15 あたりのユーザーが増加しているのは、タブレットが売れたことが最も影響しています。メーカーによらず、ほとんどのタブレットは現在 Android 4.x を実行しています。Android タブレットに特定したアプリを作成するのでない限り、アプリのターゲットは、現在モバイル端末市場の大部分を占める Android API バージョン 9 および 10 にしてください。

図 1 には、現在のところ、Android 2.3.x (API バージョン 9 および 10) が Android 端末市場の約半分を占めていることが示されています！全 Android 端末の半分が Android 2.3.x を実行しているとしたら、その市場をターゲットに作成するのが妥当だと思いませんか？

新規プロジェクトを開始する

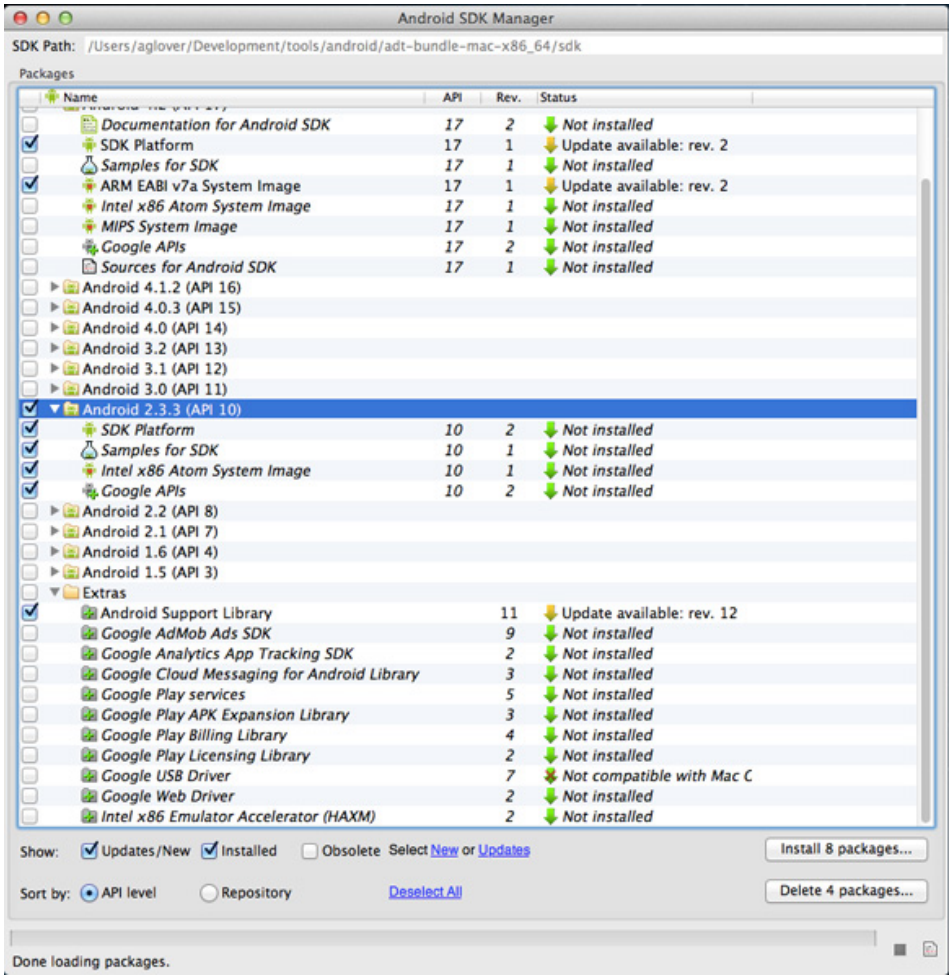
前回の記事の終わりの時点でインストールされていた API のバージョンは 4.2 です。今回は、Android 2.3.3 (API 10) をインストールしようと思います。Android SDK Manager を起動すると (図 2 を参照)、インストールできる更新がいくつかあることがわかります。

図 2. 1 つのバージョンだけがインストールされた状態のローカル Android SDK インストール済み環境



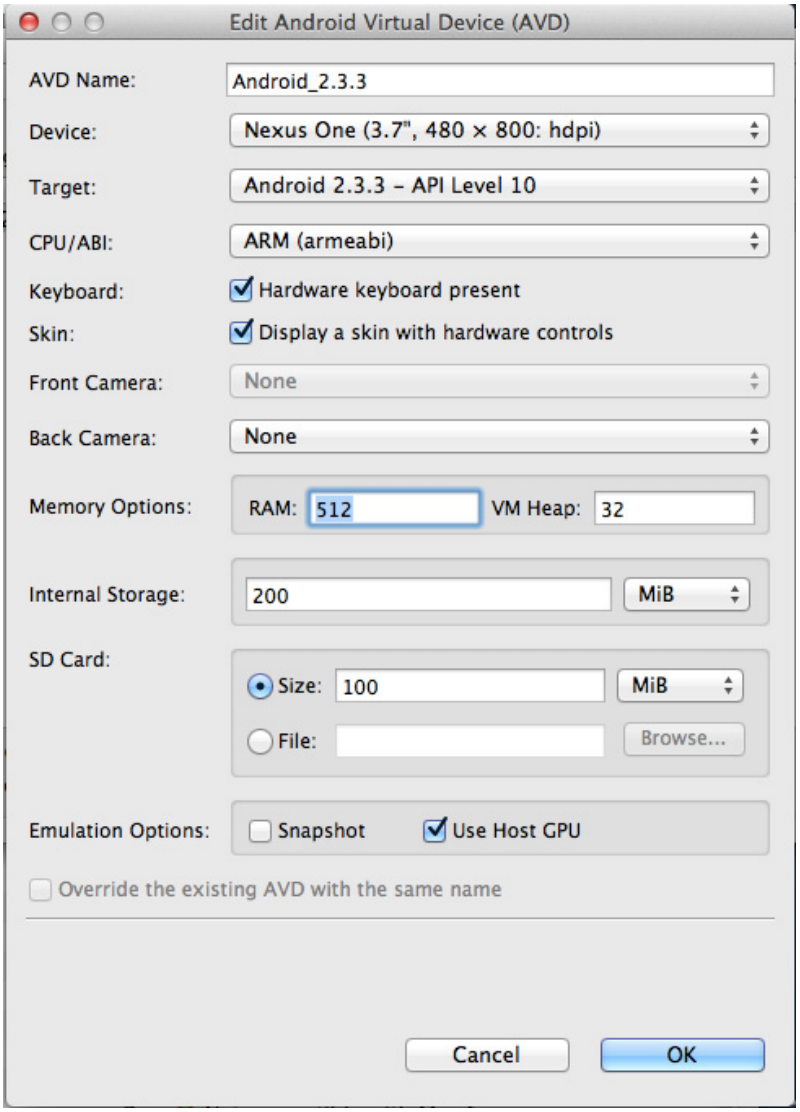
私と同じ手順に従う場合は、「Android 2.3.3 (API 10)」インストール・オプションをクリックして、Android SDK Manager が示唆する推奨の更新をそのまま適用してください。

図 3. Android 2.3.3 (API 10) をインストールする



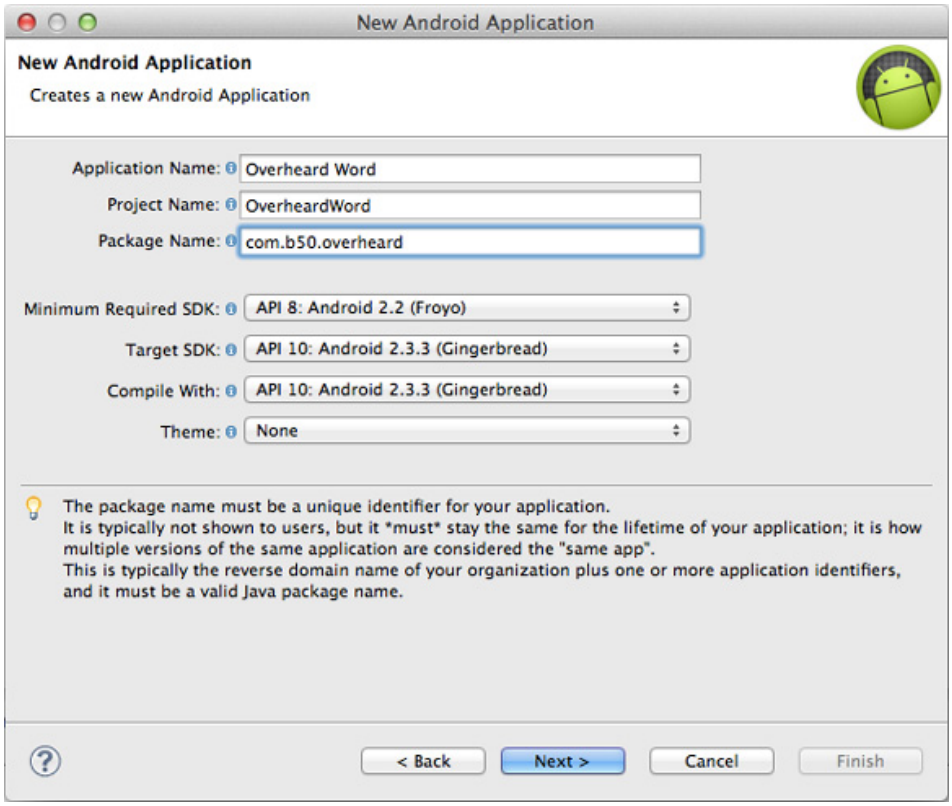
Android の新規 API をダウンロードした後は、それに対応するエミュレーターつまり AVD (Android Virtual Device) も作成する必要があります。Android SDK Manager で「Tools (ツール)」メニューを選択し、「Manage AVDs (AVD の管理)」を選択します。必ず API Level 10 をターゲットにした AVD を作成してください。

図 4. AVD を作成する



次に、「New Android Application (新規 Android アプリケーション)」ダイアログで、新規プロジェクトを作成し、そのプロジェクトに名前を付けます。私が作成するプロジェクトのアプリケーション名は、「Overheard Word」です。このアプリをできるだけ多くの端末で利用できるように、アプリのターゲットを API 10 に設定します。図 5 のスクリーン・キャプチャーをよく見ると、コンパイルに使用するバージョンにも API 10 を選択しています。これよりも新しいバージョンの Android でコンパイルすることもできますが、私はプログラミングとコンパイルには同じ API を使用するようにしています (新しいバージョンの API でコンパイルする場合、その API で追加された機能をアプリで使用しないように注意しなければなりません。これを怠ると、互換性の問題が発生する可能性があります)。

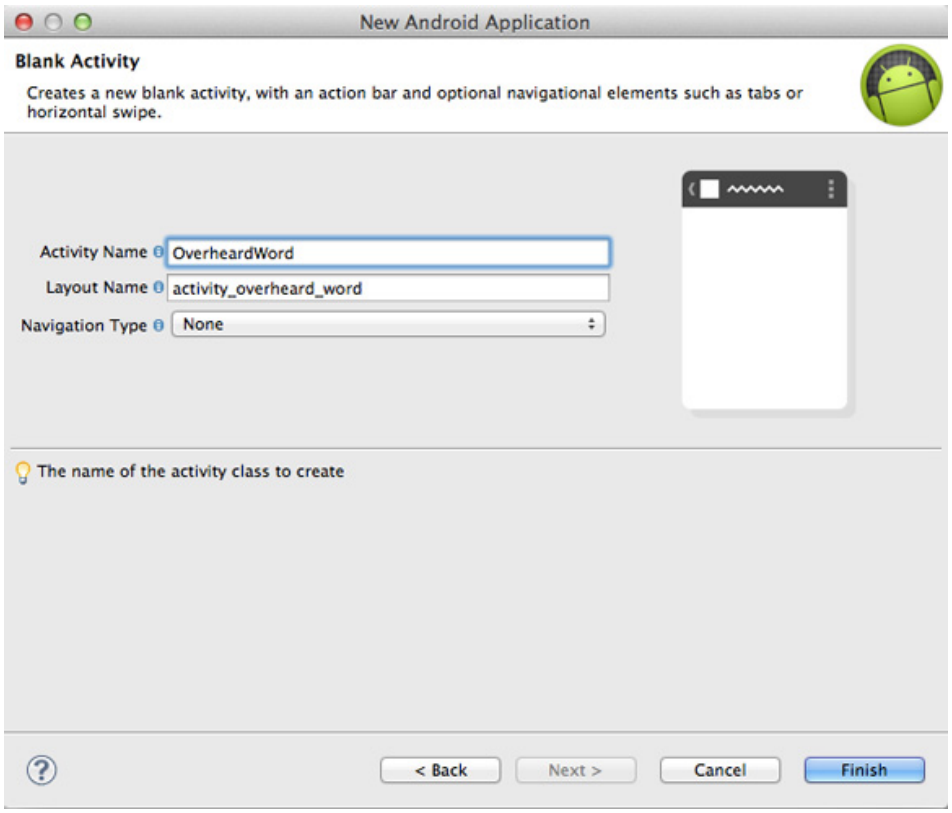
図 5. Eclipse で新規 Android プロジェクトを作成する



「Next (次へ)」をクリックするときには、アプリをライブラリー・プロジェクトにしていないことを確認してください。次に表示されるダイアログでは、Hello World! アプリに設定したのと同じデフォルト値を使用します。小さな変更 (アプリのアイコン・ファイルに対する変更など) は、後からいつでも追加することができます。

セットアップの最後のステップは、最初のアクティビティーに名前を付けることです。この例では、アクティビティーに付ける名前は「`Main`」ではなく、「`OverheardWord`」にして、レイアウトにも同様の名前を付けます。ターゲットとするバージョンとして API 10 を使用する場合、最後のダイアログにいくつかの「Navigation Type (ナビゲーション・タイプ)」オプションが示されます。この時点では、ナビゲーション・タイプについては気にしないでください。後で、ナビゲーションのメニューを作成する方法を説明します。このアプリでは、「Navigation Type (ナビゲーション・タイプ)」の選択を「None (なし)」のままにしておきます。

図 6. デフォルト・アクティビティーを作成する



「Finish (終了)」をクリックしてプロジェクトを作成します。このプロジェクトは、前回の記事で取り組んだデフォルトの `HelloWorld!` プロジェクトにそっくりです。

Android UI を作成する

これから、シンプルな UI の作成に取り掛かります。覚えているかもしれませんが、Android UI は最終的に XML 文書として定義されます。XML 文書の扱いには、皆さんも慣れていることでしょう。従来の Java GUI 開発を行ったことがある方は、レイアウトのことを覚えているかと思いますが、Android でもレイアウトを使用します。Android のレイアウトを使用して、端末がアプリのビジュアル・コンポーネントを配置する方法を定義 (実際には示唆) することができます。レイアウトには、以下のスタイルがあります。

- Linear: コンポーネントを縦方向または横方向に並べます (新聞のコラムと同様)。
- Relative: 複数のコンポーネントを相対的に配置します (ウィジェット 1 の右側にウィジェット 2 を並べるなど)。
- Table: コンポーネントを列と行の表形式で表示します。

この他にも使用できるレイアウト・スタイルはありますが、確実に上記の 3 つが出発点となります！

レイアウトの内部には、ウィジェットを定義することができます。ウィジェットは、あらゆる GUI で目にする通常のコンポーネントです。Android プラットフォームには、ボタンやテキスト・ボックス、ドロップダウン・リストなどの基本的なウィジェットの他、画像ビューアーやスピンホイールなどの高度なウィジェットも用意されています。

Overheard Word アプリの UI には、すでに以下のコンポーネントを使用することに決めています。

- 3 つのテキスト・フィールド (単語、単語の品詞、単語の定義を格納)
- 2 つのボタン (新しい単語を選択するためのボタンと、クイズに挑戦するためのボタン)

レイアウトを定義する

Android UI を組み立てる際の最初のステップは、UI のレイアウトを、そこに含めるウィジェットと併せて定義することです。リスト 1 では、最初に 3 つの `TextView` を配置する `LinearLayout` を定義しています。次に、2 つのボタンを配置するサブレイアウト (別の `LinearLayout`) を作成します。

リスト 1. Android の LinearLayout を定義する

```
<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>
```

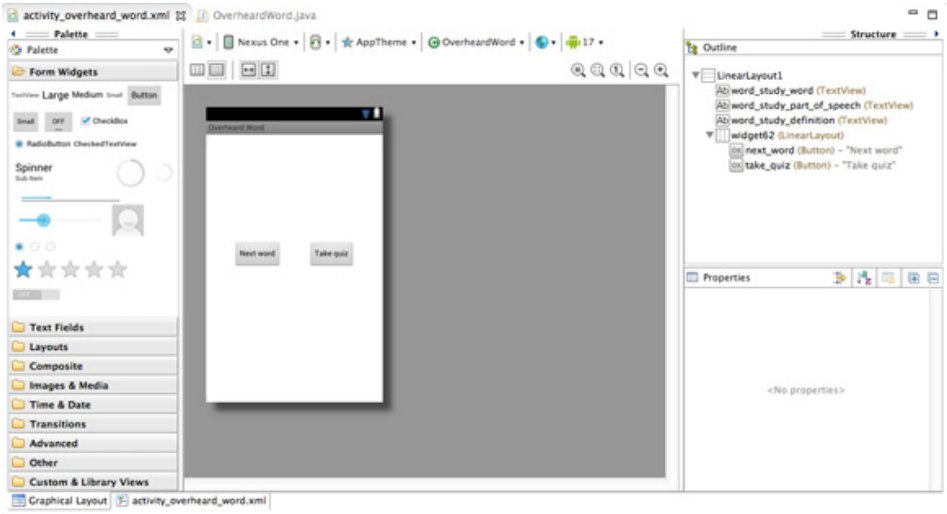
リスト 1 の XML を見るとわかるとおり、各ウィジェットにはいくつものプロパティーがあります。これらのプロパティーが、最終的にそのルック・アンド・フィールに影響を与えます。例えば、2 つのボタンのそれぞれに 1 つの `text` 要素があり、その要素のストリングがリソース要素を指しています。そのリソース要素が定義されているファイルは、`res/values` ディレクトリーに格納された `strings.xml` という名前のリソース・ファイルです。リスト 2 に、`strings.xml` ファイルを記載します。

リスト 2. ボタンの色とラベルを定義するリソース・ファイル

```
<?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>
```

UI が現状でどのように表示されるかを確認するには、エミュレーター・インスタンスを起動するという方法もありますが、代わりに Eclipse で「Graphical Layout (グラフィカル・レイアウト)」タブをクリックします (図 7 を参照)。

図 7. Eclipse で実行中の Overheard Word アプリ



プレースホルダー・テキスト

UI のプレビューを表示してみて、単語の属性用のホワイト・スペースに少し混乱しているところですが、問題はありません。これからサンプル・テキストを追加して、コードを作成する前に実際にテキストがどのように表示されるかを把握できるようにします。

まず、ボタンのラベルと同じように、`TextView` ウィジェットのそれぞれに `android:text` 属性を追加します。今のところは、以下のようにプレースホルダー・テキストを追加しておきます。

リスト 3. android:text を使用してプレースホルダーを作成する

```
<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"/>
```

`### 3` に示されているように、`word_study_part_of_speechTextView` には、サンプル・テキスト ("Part of Speech") を追加しました。さらに、一部のスタイル要素を更新するために、テキストのサイズを大きくして、テキストの色を定義し、ウィジェットをレイアウト内でセンタリングしています。リスト 3 に示されている属性は、Android アプリのウィジェットを定義する際に共通して使用する属性です。アプリの作成を繰り返していくうちに、これらの属性を理解するようになり、愛着がわいてくると思います。また、サンプル・テキストをリソース・ファイルにではなく、レイアウト・ファイルに直接定義している理由は、設計プロセスの間は、テキストの値は一時的なものだからです。

Eclipse からアプリを実行すると、ユーザーがこのアプリを起動したときに表示される内容を確認できます。

図 8. AVD で実行中の Overheard Word アプリ

この UI 設計には、まだしっくりこないところがあります。その理由は何なのか、見当がつきますか？そうです、それは煩わしいボタンの存在です！ボタンは必要でしょうか？通常、モバイル・アプリでユーザーはボタンをクリックするのでしょうか？おそらくこのようなモバイル・アプリでは、ユーザーが左か右にスワイプすると新しい単語を表示することができ、下にスワイプするとクイズに挑戦することができるのが一般的です。

ボタンをなくすことで、UI がすっきりとして、Overheard Word アプリをより適切にほとんどのモバイル・ユーザーが期待するようなアプリにすることができます。この後すぐにわかるように、スワイプの振る舞いを実装するのは簡単です。

Android でのジェスチャーをコーディングする

Overheard Word アプリの初期画面を定義したので、振る舞いを追加する準備は整いました。ここでは私が従来の Java GUI 開発で培った経験を直接 Android に適用し、イベントに応答するリスナーを最初に実装します。画面に 2 つのボタンをそのまま残すのであれば、`OnClickListener` を

使用して、それぞれのボタンのクリック･アクションに振る舞いを追加した上で、各リスナーを該当するボタン・ウィジェットの `setOnClickListener` メソッドに追加することになるでしょう。

匿名クラス

匿名クラスとは、あるクラスにローカルなクラスで名前を持たないクラスのことです。匿名クラスの定義とインスタンス化は、同じ 1 つの表現で同時に行うことができます。匿名クラスは便利で簡潔なので、特にユーザー・インターフェースの振る舞いをプログラミングする際には、私は数多くの匿名クラスを作成しています。

もちろん `OnClickListener` を使用することもできますが、その場合、複数のウィジェットに複数のクラスができるという結果になります。コードをクラスであふれさせることはしたくはないので、代わりの手段として匿名クラスという実装ストラテジーを採用します。

スワイプのようなジェスチャーは従来の GUI プログラミングにはない概念ですが、ジャスチャーの実装は、ボタン・クリックをコーディングする場合と似ています。出発点となるのは、Android プラットフォームの一部となっているタッチ・リスナーです。次に、該当する振る舞いをタッチ・リスナーに指定します。指でスワイプする動きを検出するために、ジェスチャーの開始点と終了点の違いに加え、スワイプの速度も計算します (この場合、デカルト座標を考えると役に立つと思います。つまり、左右のスワイプは X 軸に対応し、上下のスワイプは Y 軸に対応します)。

ジェスチャーは、インターフェース上のあらゆる場所で行われる可能性があります。この類のリスナーをアプリのインターフェースのコンポーネントに追加するときには、(Android では `View` として表現される) レイアウトに最も単純なボタン・ウィジェットを含めるところから始めます。ここでは、Android の `SimpleOnGestureListener` を使用して最初のジェスチャー実装を作成し、その実装を `onTouchListener` メソッドを介して `View` に追加します。ビューのインスタンスに追加することにより、小さい写真ビューアーなどといったインターフェースのコンポーネント内だけでなく、インターフェース全体でスワイプを検出できるようになります。

ただしその前に、モバイル端末上での指の動きを検出できるクラスを作成しなければなりません。

SwipeDetector

ボタンのクリックを検出するのは、比較的簡単です。一般に、オペレーティング・システムではボタンの境界を表す座標を抽象化しているため、その座標の詳細を手動で定義する手間を省けます。一方、スワイプ動作を認識して、それに応答するアプリをプログラミングするのは少し難解な作業です。Android ではプログラマーに委ねられている部分が多いため、このジョブは少し複雑ではあるものの、プログラマーにかなりの柔軟性が与えられます。例えば、私がここで行っているのと同じく単一のスワイプを検出することも、これ以外の指の動作 (例えば、ゲームに使用されることがある、画面をこする動きなど) を検出することもできます。

モバイルの考え方

モバイル・アプリのリスナーが「click (クリック)」という動詞で定義されるのは気になりませんか？この前確かめたときは、ユーザーがモバイル端末のボタンを操作する場合、デスクトップ PC で操作するときのように「クリック」するのではなく、ユーザーはボタンを「押す」動作をしていました。モバイル開発者になるということには、モバイルの世界の観点で考えることも含まれます。つまり、ボタンのクリック、スタート・メニュー、さらには GUI

ウィザードといった従来の Web アプリケーションやデスクトップ・アプリケーションのメカニズムについて再考するということです。

スワイプを検出するためのプログラミングには、デカルト座標の計算が必要です。具体的に言うと、X 軸上のタッチ開始座標から X 軸上のタッチ終了座標を引き算した値が、事前に定義済みの距離よりも大きく、なおかつスワイプ動作の速度も事前に定義済みの値より大きい場合には、ユーザーがアプリのディスプレイ上で指を右から左へスワイプしたと見なすことが、合理的な理由から可能です。上下のスワイプについても、Y 軸上で同じように計算されます。

この種のロジックを、基本的な計算を自動的に行ってくれる単純なクラスに抽象化するのは簡単です。`SimpleOnGestureListener` に、`onFling` というメソッドがあります。`onFling` メソッドは、X 軸方向の速度を表す 2 つの MotionEvent 型の引数 (動作の「開始」と「終了」に対応) と、Y 軸方向の速度を表す 2 つの `float` 型の引数を取ります。したがって、リスト 4 に記載する `SwipeDetector` クラスにはスワイプのすべての動作 (左、右、上、下) を表現する 4 つのメソッドがあります。

リスト 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);
    }
}
```

特定のジェスチャーが発生したかどうかを知らせてくれる重宝なクラスを用意できたので、次はこのクラスを UI に組み込みます。最初にデフォルトのアクティビティーを作成し、それに `OverheardWord` という名前を付けたことを思い出してください。Eclipse によって提供されたデフォルトのコードはそっくりそのまま残しますが、そこにいくつかのコードを追加します。スワイプをリッスンし、それに応答するために、`OnGestureListener` の実装を引数に取る Android の `GestureDetector` を定義します。ありがたいことに、Android チームではコンビニエンス・クラスを作成してくれていて、このクラスを `SimpleOnGestureListener` という匿名クラスによって実装することができます。その上で、`onFling` メソッドをオーバーライドします。

UI でのスワイプ検出

作成したアクティビティーの `onCreate` メソッド内で、最初に `SimpleOnGestureListener` を使って `GestureDetector` を作成します。このディテクターで、前に作成した `SwipeDetector` クラスを使用します。

次のステップでは、ユーザーのスワイプ操作によってイベントが起動されたことを示す小さなダイアログを作成します。このような短時間だけ表示される小さなダイアログは、Android ではトースト (`Toast`) と呼ばれています。トースト (`Toast`) は、場合によってはたった 1 行のコードからなることもあります。

リスト 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();
        }
    });
}
```

ユーザーによるスワイプの範囲と速度を計算する `initGestureDetector` メソッドが作成できました。このメソッドは、`SimpleOnGestureListener` の匿名インスタンスを作成してから、`onFling` メソッドを実装します。`onFling` メソッドで `SwipeDetector` をどのように使用しているかに注目してください。例えば、上方向のスワイプで一致が検出されたとすると、その動きを示すためにトースト (`Toast`) が表示されます。

次のステップは、スワイプ検出の振る舞いを UI に組み込むことです。それにはまず、いくつかのリスナーを `View` のインスタンスに登録するところから始めます。

リスト 6. スワイプ検出を View に登録する

```
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) {
        }
    });
}
```

`onCreate` メソッド内では、`ID` を使用して実際の `View` インスタンスへのハンドルを取得しています。Java 開発者の大部分は、この重宝な Android 式のやり方を高く評価しています。

ID を基準にビューを検索する方法

UI の XML を定義したときに、それに対応する値も各種のリソース·ファイル (`strings.xml` など) に定義したことを覚えていますか？ Android アプリをコンパイルすると、その XML が生成済みコードに変換され、対応する `ID` が `R` というクラスに格納されます。先に進む前に、プロジェクトの `gen` ディレクトリーにある `R` クラスを調べてください。

レイアウトの XML ファイルを見てください (覚えているかと思いますが、私はこのファイルに `activity_overheard_word.xml` という名前を付けました)。この XML ファイル内では、すべてのウィジェットに `id` 属性があります。例えば、このサンプル・アプリのレイアウトの `id` は `android:id="@+id/LinearLayout1"` (省略形は `LinearLayout1`) です。`id` 名は Eclipse によって自動的に生成されますが、必要に応じて変更することもできます。重要な点は、`R` クラスには、`id` (`LinearLayout1`) に対応するプロパティーが含まれていることです (リスト 7 を参照)。

リスト 7. R クラス内のウィジェット ID

```
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;
}
```

XML ファイルの内容とそれに対応する `R` ファイルを組み合わせれば、XML を構文解析しなくてもウィジェットを参照することができます。作成したアクティビティーの `onCreate` メソッドに話を戻すと、`View` ウィジェットへの参照を取得するために、その `ID` である `R.id.LinearLayout1` を使用しています。`Activity` を継承すると、Android プラットフォームによって `findViewById` メソッドが提供されます。

指によるスワイプが適切に処理されるようになりました！

`View` インスタンスを取得した後は、`setOnTouchListener` を介して、`gestureDetector` インスタンスに処理をさせることができます。リスト 6 では、別の匿名クラスがタッチの振る舞いのすべてを処理しています。ユーザーが端末の画面にタッチすると、タッチ・イベントが起動されて `gestureDetector` に処理が委ねられます。また、`OnClickListener` を実装して、このリスナーを `setOnClickListener` メソッド内で設定しています。ただし、この場合、匿名クラスには振る舞いがありません。

以上のように、指によるスワイプに応答するインターフェースは、比較的単純なものに仕上がりました。この場合のスワイプは、左、右、上の 3 方向です。このコードを AVD で試してみてください。スワイプをシミュレートするには、マウスをクリックしながら左、右、上にドラッグします。

それでは、この開発中のアプリを実際の端末で実行してみましょう。

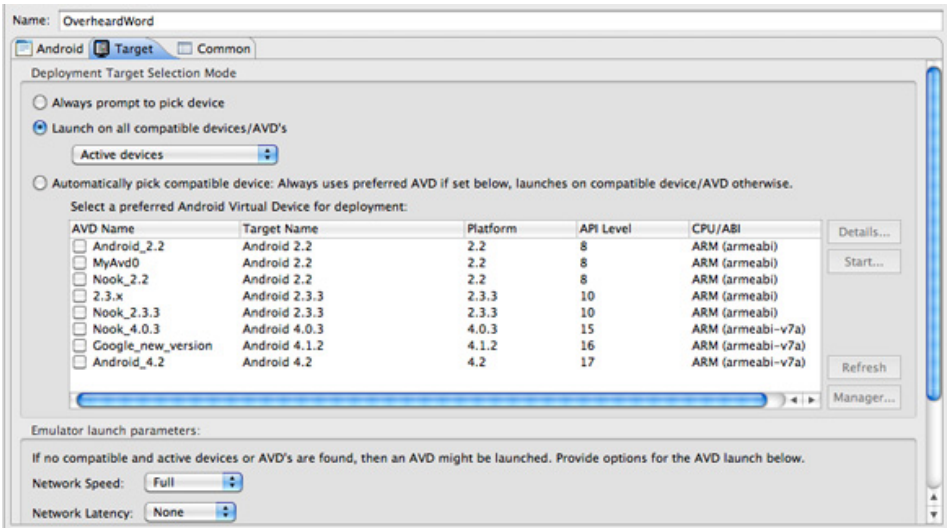
開発モードでデプロイする

ウィルス、トロイの木馬、そして無防備なユーザーを悩ませるマルウェアなどの拡散に対処するために、Apple と Google の両社では、それぞれの端末で実行するサード・パーティー・アプリに対して、コード・サイニングの形式を導入しています。理論的には、コード・サイニングは、そのアプリが信頼できる人によって作成されていることに関する保険、そしてアプリのインストール前に誰もバイナリー・ファイルを改ざんしていないことに関する保険の役割を果たします。ただし、実際には Google と Apple のセキュリティー署名用のコードはそれぞれに異なります。私が自分で作成した Android アプリに署名するときには、例えば自分が有名な国立銀行であると主張することができますが、iOS や Windows Phone 8 用のアプリに署名するときには、それは不可能です。

開発モードでは、コード・サイニングについて心配する必要はありません。実のところ、Android と Eclipse は、開発者に知られることのない開発者キーを使って署名されたバイナリーをビルドします。したがって、開発モードでアプリを端末にデプロイするのは難しいことではありません。このようなアプリを端末にデプロイする最も簡単な方法は、Android 端末をコンピューターの USB ポートに接続することです。

次に、Eclipse でプロジェクトを右クリックし、「Run As (実行)」メニュー・オプション、「Run Configurations (実行構成)」の順に選択します。この操作によって Eclipse に表示される構成ダイアログで、プロジェクトを選択し、「Target (ターゲット)」タブをクリックします (図 9 を参照)。

図 9. アプリを実行する端末を選択する



「Always prompt to pick device (デバイスを選択するときに常にプロンプトを表示)」を選択することや (ユーザーが選択できるようにしておきたい場合)、「Launch on all compatible devices/AVD's (互換するすべてのデバイス/AVD で起動)」を選択することができます。後者を選択する場合には、私と同じように「Active Devices (アクティブ・デバイス)」オプションを選択してから「Run (実行)」をクリックしてください。この記事の説明に従ってコードを作成した場合、ほどなくすると端末に見事なアプリが表示されます。左、右、上にスワイプして、すべてが機能していることを裏付けるダイアログが表示されることを確認してください。

セキュリティー設定

e-メールまたは Dropbox からアプリをロードできない場合は、端末の設定画面にアクセスして、「Security Settings (セキュリティー設定)」で「Unknown Sources (不明なソース)」を有効にしてください。このように変更すると、Google Play からではなく、アプリ (開発中のアプリを含む) をインストールできるようになります。

開発中のアプリをデプロイする別の方法として、アプリの `.apk` ファイルを自分宛に e-メールで送信し、そのメールを Android 端末で開いて自動インストール・ダイアログに従うこともできます。あるいは、`.apk` ファイルを Dropbox などのサービスにアップロードしてから端末で Dropbox を開き、そこからインストールするという方法もあります。作成したアプリの `.apk` ファイルは、プロジェクトの `bin` ディレクトリーにあります。このディレクトリーには、Eclipse によって対応するバイナリーも格納されます。

これらのデプロイメント・メカニズムは、開発のテスト用であって、配布目的ではないことに注意してください。アプリを配布して儲けるためには、さらに追加で行わなければならないステップがあります。それについては、今後の記事で説明します。

まとめ

モバイル・アプリを設計することは、シンプルかつ容易に使えるものを考えるということでもあります。今回は、デスクトップ・アプリケーションや Web アプリケーションのユーザーが一般的に使用するボタンのクリックではなく、スワイプ・ジェスチャーに応答するモバイル・アプリをプログラミングする方法を説明しました。場合によっては、便利なボタンが 1 つや 2 つあった方が、モバイル・アプリにとってメリットがあるかもしれません。それらのボタンが、ユーザーがアプリを利用する目的に適っていることを確認してください。私の超シンプルな Overheard Word アプリの場合、ボタンは必要ありません。この記事で完成させた上下のスワイプ・ナビゲーションは、ほとんどのモバイル・アプリ・ユーザーにとって直観的なものです。

モバイル・アプリのデプロイメントについても簡単に取り上げましたが、テスト用のデプロイメントという範囲でしか説明していません。皆さんにはまだ、本番アプリをデプロイする必要が残っています。大抵のユーザーは、信用できないアプリをインストールしません。これが、成功を収めているアプリのほとんどが、Google Play、iTunes App Store、または Amazon Appstore for Android などの手段によって配布されている理由です。今後の記事では、アプリケーション・ストアのセキュリティー標準を満たすためのプロセスを紹介します。そのプロセスには、アプリに署名を付ける以外にも、多くのステップがあります。ただし重要な点は、作成したアプリが主要なディストリビューターでホストされるようになった後は、そのアプリが世界中で利用可能になることです。

関連トピック

- 「万人のためのモバイル: Android の手ほどき」(Andrew Glover 著、developerWorks、2013年3月): まず始めに、この実用的入門編で Android 開発環境をセットアップして、最初のアプリを作成してください。
- 「Android 開発入門」(Frank Ableson 著、developerWorks、2009年5月): この入門記事では、Android アプリケーションのアーキテクチャーについて概説しています。
- 「モバイル Web のためのユーザー・インターフェースの設計」(James L. Lentz 著、developerWorks、2011年7月): モバイル Web のユーザビリティーに関する課題を詳しく探り、モバイル Web アプリケーションを設計する際のベスト・プラクティスを検討してください。
- 「C-Swipe: An Ergonomic Solution To Navigation Fragmentation On Android」(Greg Nudelman 著、Smashing Magazine、2013年3月25日): モバイル端末でのスワイプ・ナビゲーションの人間工学について、親指による半円形のスワイプを検出するためのパターンを含め、詳しく学んでください。
- 「Swipe Views: Android developers」: Android のドキュメントで、スワイプ・ナビゲーションの他のパターンも学んでください。
- 「The Latest Infographics: Mobile Business Statistics For 2012」(Mark Fidelman 著、Forbes.com、2012年5月): 2012年中間期のビジネス統計では、Apple CEO の Tim Cook 氏も述べているようにモバイル・アプリの開発および配布がかつてないほど急増していることが明らかになっています。
- developerWorks テクニカル・イベント: これらのセッションで最新の技術情報を入手してください。
- Twitter での developerWorks: 今すぐ登録して developerWorks のツイートをフォローしてください。
- Android のダウンロード: Android SDK は、Android 向けアプリの作成、テスト、デバッグに必要な API ライブラリーと開発ツールを提供します。
- IBM developer kits のダウンロード: システムを更新して、最新のツールとテクノロジーを入手してください。