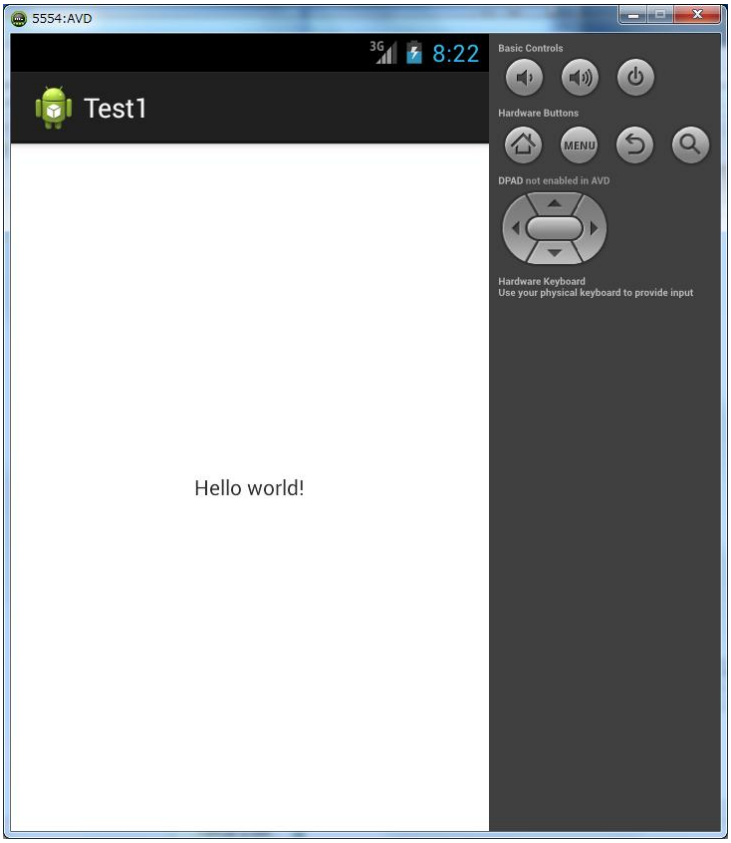
Android 4.2(android-sdk_r21.0.1-windows)+Eclipse 4.2での Androidアプリの作り方

以下の作業が終わって Eclipse で Android アプリを開発する環境が整備されているものとして話を進めます。

・Android SDK のインストール
・Eclipse のインストール
・Eclipse への Android Plugin（ADT）のインストール
・Android アプリを実行するために必要な AVD（Android Virtual Device）の作成

この章ではデフォルトのスケルトン（システムが自動生成するプログラムの骨格）を使って「Hello world!」というメッセージを TextView（テキストビュー）に表示するプログラムの作り方の手順を説明します。また作成されたファイルの意味と役割を説明します。



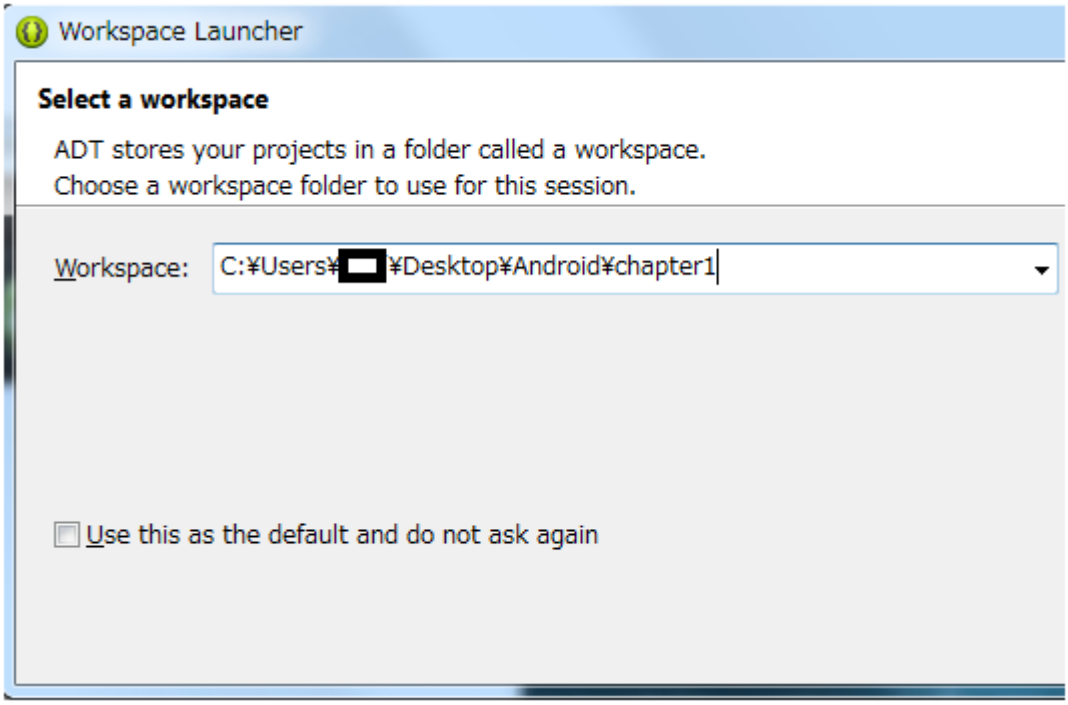
「注」Eclipse や Android SDK のバージョンにより作業手順が異なる場合があります。

1-1　プロジェクトの作り方

Android アプリはプロジェクトで管理します。プロジェクトを保管するためのフォルダをワークスペースと呼びます。

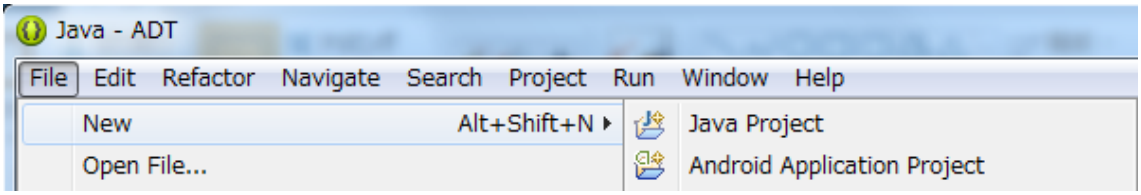
1.　ワークスペースの作成

Eclipse を起動し「Workspace Launcher」画面でワークスペース名を入力します。ここでは、デスクトップの「Android」フォルダにワークスペースを「chapter1」として作成します。「Use this as the default and do not ask again」にチェックが入っているとワークスペースの設定画面は出ません。



2.　プロジェクトの作成

①「File」－「New」－「Android Application Project」を選択します。



②アプリケーション名等の入力

・Application Name、Project Name、Package Name

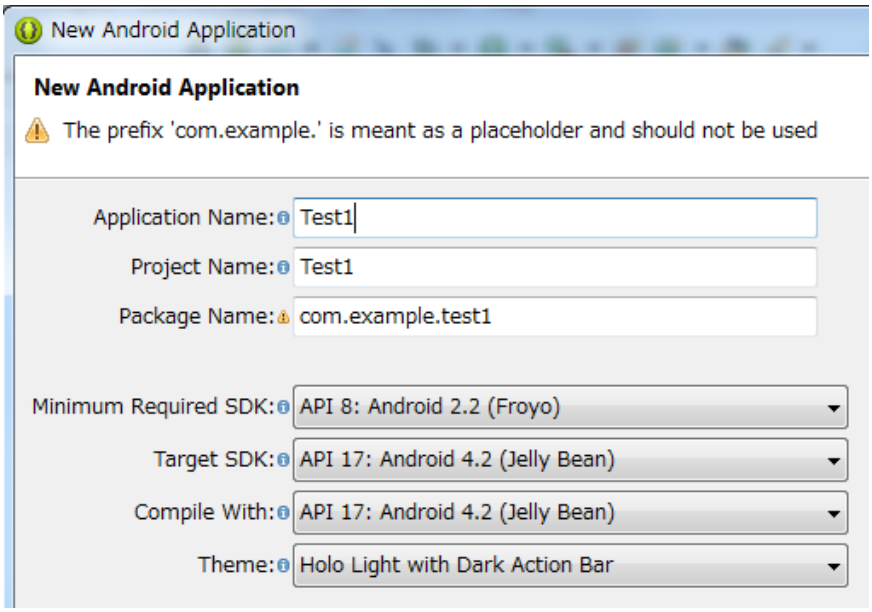
Application Name を「Test1」と入力します。自動で Project Name は「Test1」、Package Name は「com.example.test1」　となりますので、そのままとします。

１つのJavaアプリを構成する各種ファイルを管理するための基本をプロジェクトと呼びます。プロジェクト名のフォルダ内に各種ファイルが格納されます。

・ SDK バージョン

AndroidSDK のバージョン（Target SDK と Compile With）を選択します。ここでは「API 17:Android 4.2」を選択しました。

Minimum Required SDK はアプリケーションが動作する最低の Android SDK バージョンです。ここでは「API 8:Android 2.2」を選択しました。



「注」SDK バージョンと API レベル

Android SDK バージョン	API レベル	コードネーム
4.2	17	Jelly Bean
4.1	16	Jelly Bean
4.0.3	15	Ice Cream Sandwich
4.0	14	Ice Cream Sandwich
3.2	13	Honeycomb
3.1	12	
3.0	11	
2.3.4(2.3.3)	10	Gingerbread
2.3	9	
2.2	8	Froyo
2.1	7	Eclair

2.0.1	6	
2.0	5	
1.6	4	Donut
1.5	3	Cupcake
1.1	2	非公開
1.0	1	非公開

③Configure Project

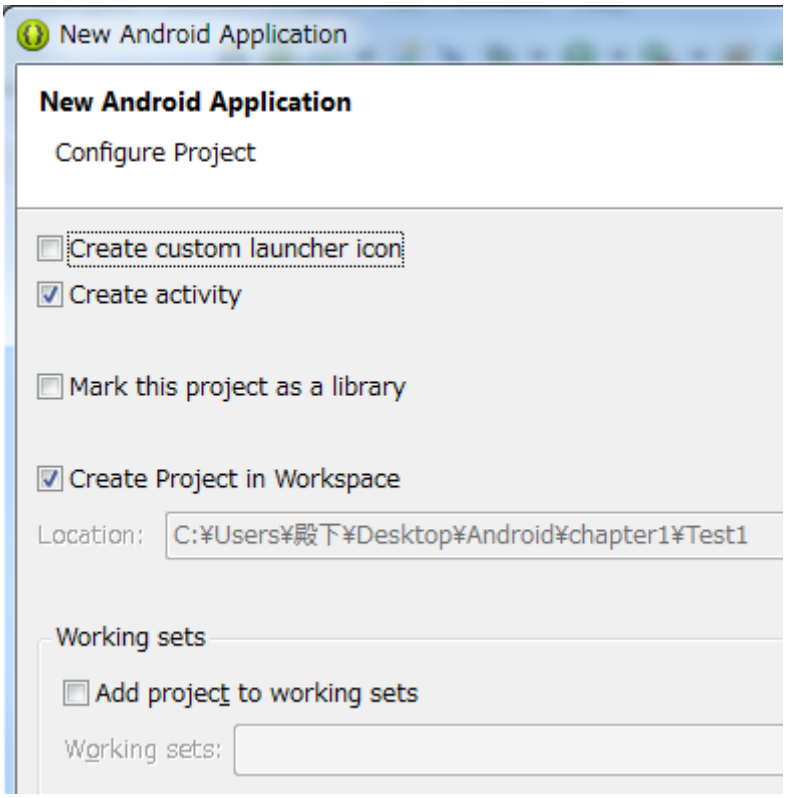
プロジェクトの構成を指定します。

・Create custom launcher icon

独自に launcher icon を作る場合はチェックを入れます（デフォルト）。ここではデフォルトのアイコンを使用するのでチェックを外します。

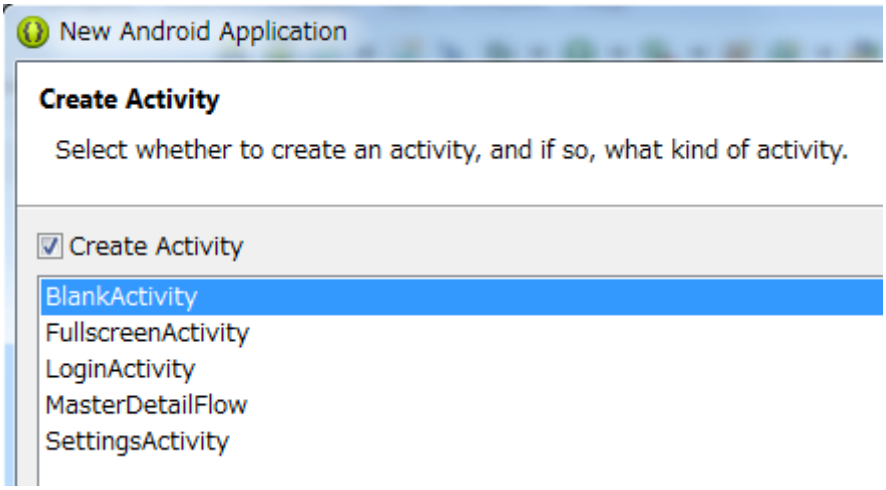
・Create activity

Android 画面の基本は Activity で、このクラスを自動的に作る場合に□にチェックを入れます（デフォルト）。



④アクティビティの作成

BlankActivity（通常のアクティビティ）を選択します。



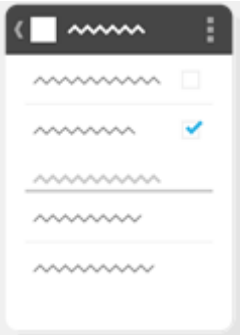
「注」Android 4.2 では以下の 5 種類のアクティビティを選択できるようになりました。


BlankActivity

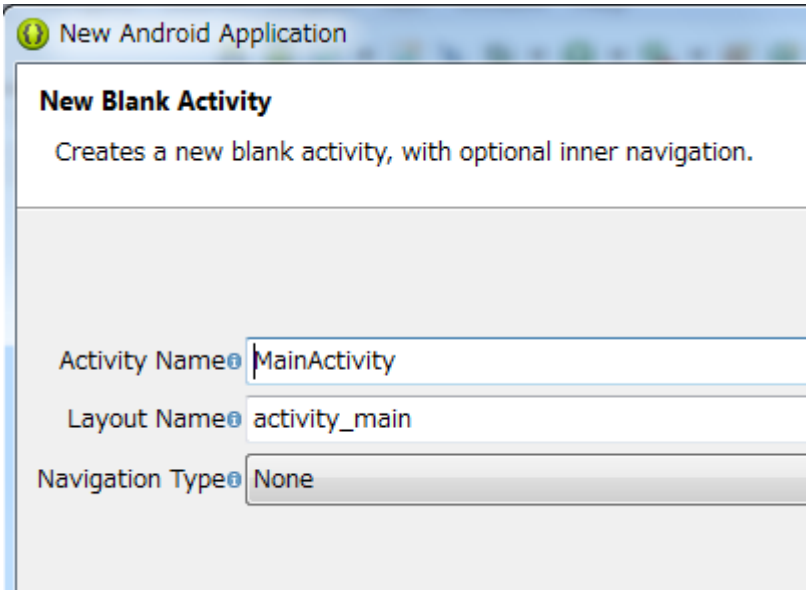

FullscreenActivity


LoginActivity


MasterDetailFlow


SettingsActivity

⑤Activity Name、Layout Name の入力
デフォルトで以下のような名前になっていますので、このままとします。古いバージョンでは Activity Name は「Test1Activity」、Layout Name は「main」となっていました。
Activity Name:MainActivity
Layout Name:activity_main



1-2　各種名前の意味

プロジェクトを作る過程で出てきた各種名前の意味をまとめて説明します。

1.　ワークスペース名

複数のプロジェクトを保管する一番元になるフォルダの名前です。

2.　プロジェクト名

１つのJavaアプリを構成する各種ファイルを管理するための基本をプロジェクトと呼びます。プロジェクト名のフォルダ内に各種ファイルが格納されます。

3.　アプリケーション名

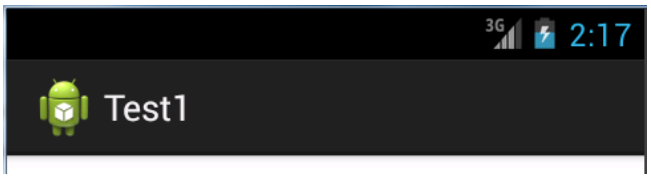
Java アプリの中に埋め込まれる名前をアプリケーション名と呼びます。アプリケーション名は「app_name」の値として次のように res/values/string.xml 内に埋め込まれます。

```
<resources>

    <string name="app_name">Test1</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>

</resources>
```

実行時にタイトルバーにこのアプリケーション名が表示されます。



4.　パッケージ名

　関連するクラスやインタフェースを１つにまとめた単位をパッケージと呼びます。パッケージにまとめることにより、機能ごとのカテゴリとして階層構造で分類することができます。異なるパッケージでは、同じ名前のクラスやインターフェースがあってもそれぞれ別なものとして扱えるため、名前の衝突を防ぐことができます。

　パッケージ名は「.」で区切って階層構造の名前を付けます。パッケージ名はすべて小文字にする慣習があります。各社が同じような名前でアプリを開発したときに名前の衝突が起きないように、会社のドメイン名などを入れるのが一般的です。ある会社のドメイン名がたとえば、「kasai.co.jp」であるとパッケージ名は,ドメイン名を逆順にならべた「jp.co.kasai.」を先頭にし、それぞれのパッケージ名を階層構造で付けて

「jp.co.kasai.android.camaera」や「jp.co.kasai.java.util」などとなります。この他に「com.会社名.プロジェクト名.機能名」のような命名規則も多く見られます。

5.　クラス名

Java アプリは複数のクラスで構成されますが、public 指定されたクラスがそのアプリを代表するクラス名となります。「1-1　プロジェクトの作り方」－「2.　プロジェクトの作成」－「⑤Activity Name、Layout Name の入力」で指定した Activity クラスの名前を使って以下のようなクラスが生成されます。そしてこのクラス名と同じ名前の Java ソースファイル「MainActivity.java」が作成されます。

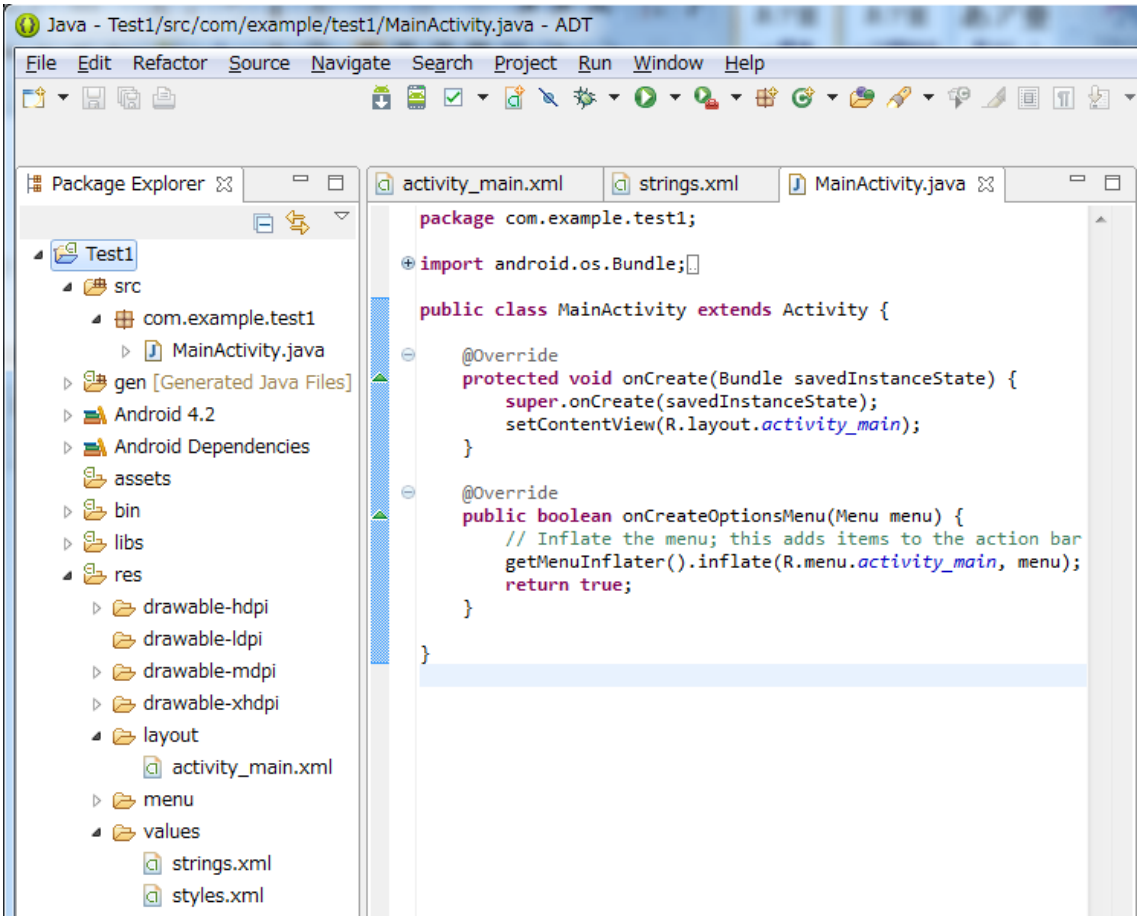
```
public class MainActivity extends Activity {
```

「注」本書での命名規則

本書のアプリは小規模のため「ワークスペース名＝プロジェクト名＝アプリケーション名」とし、パッケージ名は「com.example.プロジェクト名をすべて小文字にしたもの」とします。

1-3　作成されたファイル

1-1 で示したプロジェクトの作成作業が終了すると、デフォルトのスケルトン・ファイルとして MainActivity.java、activity_main.xml、string.xml などが作成されます。



1.　ソースファイル（MainActivity.java）

Java のソースファイルです。「1-1　プロジェクトの作り方」－「2.　プロジェクトの作成」－「⑤Activity Name、Layout Name の入力」で指定した Activity クラス名に「.java」を付けたものがファイル名になります。

```
package com.example.test1;←パッケージ名

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {←クラス名
```

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

2. レイアウトファイル（activity_main.xml）

ウイジェットを配置するレイアウトを定義した XML ファイルです。res/layout フォルダに activity_main.xml という名前で保存されます。

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

</RelativeLayout>
```

3. 文字列リソース・ファイル（string.xml）

アプリケーション名などの文字列テキストのリソースを定義した XML ファイルです。res/values フォルダに string.xml という名前で保存されます。

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

    <string name="app_name">Test1</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>

</resources>
```

「注」リソースとは
文字列、イメージ、レイアウトなどのプログラムから分離されたデータをリソース（resource : 資源）と呼びます。Android で使う主なリソースとして以下の 3 種類があります。リソースは res フォルダの下にリソースの種類ごとに配置されます。
・drawable
イメージのリソースです。異なる解像度やディスプレイサイズに対応するために xhdpi(eXtra High)、hdpi（High）、mdpi（Middle）、ldpi（Low）の 4 種類の dpi（dot per inch）リソースが用意されていて、実行している端末に合った dpi のフォルダから自動的に画像を取得します。
・layout
レイアウトのリソースです。たとえば activity_main.xml などを置きます。
・values
文字列や色、サイズなどの数値データのリソースです。たとえば string.xml などを置きます。

「注」文字列リソース
文字列などはわざわざリソースとして定義せずにコードに直接記述することも可能ですが、たとえば同じ文字列を繰り返し使っているプログラムがあった場合、あとでその文字列に変更が発生するとその修正箇所は多くなってしまいます。リソースを使えばリソースの定義部分だけを変更すれば済みます。ただ小規模プログラムではリソース本来の効果より定義場所と使用場所の 2 箇所を管理することの方が煩雑になるので、本書のプログラムでは文字列リソースは基本的に使用しません。⇒「1-5　表示内容を変えてみる」－「3.　直接文字列」参照。

4. 生成されたフォルダとファイルの意味

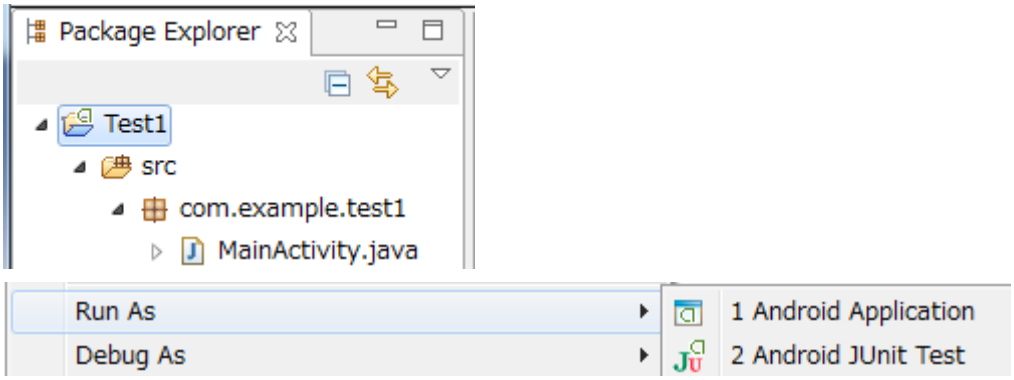
生成されたフォルダとファイルの意味は以下です。

フォルダまたはファイル	内容
src	ソースファイルが作成されるフォルダ
gen	リソース定義ファイル（R.java）が自動生成されるフォルダ
Android4.2	指定した Android バージョンのライブラリ情報。この名前のフォルダがあるわけではなく、Eclipse が内部情報として持っている
assets	バイトストリームとして読み込むファイルの保存するフォルダ
res	画像や画面レイアウトに関するリソース・ファイルを保存するフォルダ
AndroidManifest.xml	実行に必要な情報が記述されたマニフェスト・ファイル
project.properties	デフォルトのプロパティ・ファイル

1-4　作成したプログラムの実行

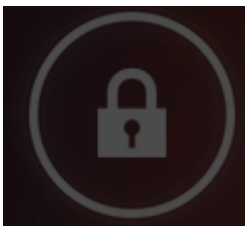
以下の手順で作成したプログラムをエミュレータ上で実行します。

1. パッケージ・エクスプローラー内のプロジェクト名の上で右クリックし、「Run As」－「Android Application」を選択します。

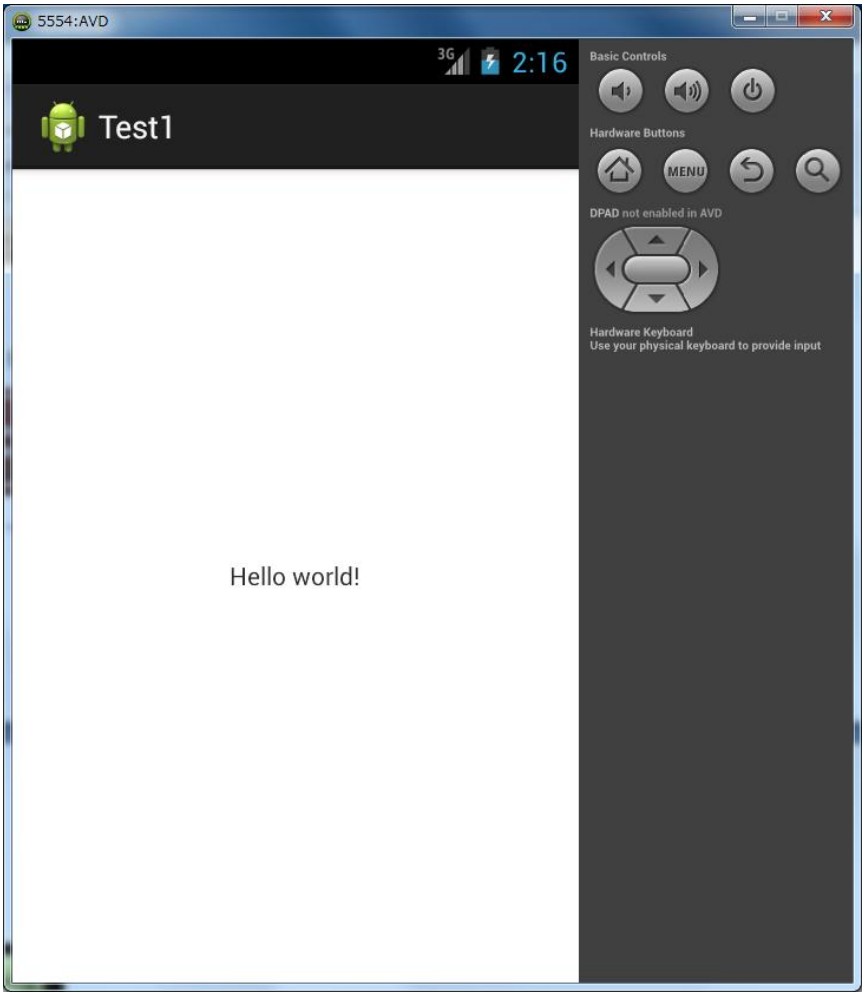


「注」以下の実行ボタンでもよいです。

2. Android 実行画面が現れたらロックを解除します。



3. 表示された実行結果



「注」自動実行しない場合は Home 画面の以下を選択します。

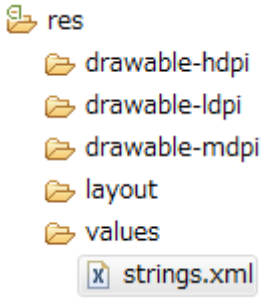
ランチャーに登録されているアプリ Test1 を選択します。

1-5　表示内容を変えてみる

実行結果の「Hello world!」はプログラムのどこに記述されているのでしょうか。それは「res」－「values」フォルダの「strings.xml」ファイルの中に記述されています。

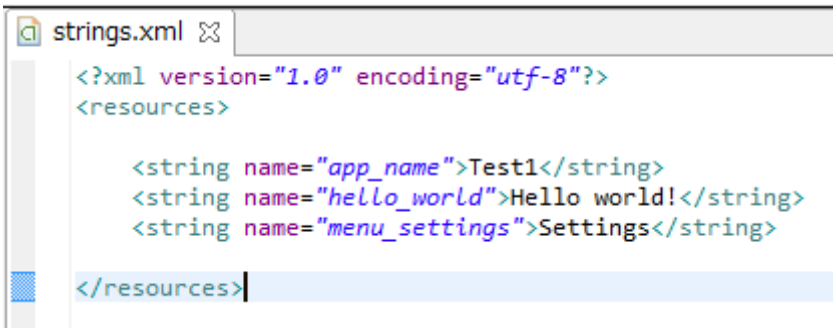
1.　リソースの内容

①「res」－「values」－「strings.xml」を選択



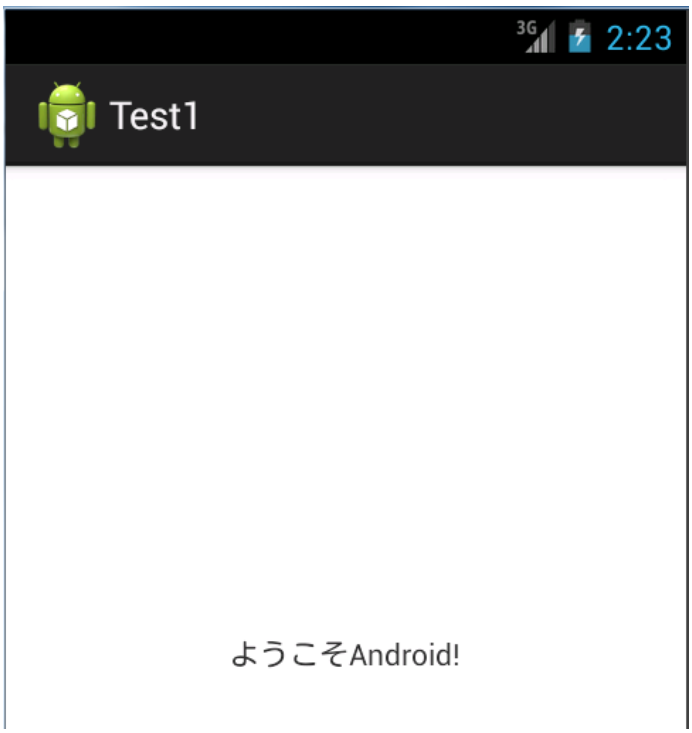
②string.xml の内容を表示

デフォルトで"app_name"と"hello_world"という名前の文字列リソースがそれぞれ、「Test1」と「Hello world!」に定義されています。「Test1」は「1-1　プロジェクトの作り方」－「2.　プロジェクトの作成」－「②アプリケーション名等の入力」で指定したアプリケーション名です。



ここに記述されている「Hello world!」を「ようこそ Android!」に変えて実行し直せば次のように表示されます。

```
<string name="hello_world">ようこそAndroid!</string>
```



2. activity_main.xml の TextView の内容

Android ではテキストビューやボタンなどの GUI 部品をウイジェット（Widget）と呼んでいます。テキストビューは activity_main.xml の中で次のように<TextView>タグを使って定義します。

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />
```

テキストビューに表示する内容は text プロパティに「android:text="@string/hello_world"」のように指定します。

「@string/hello_world」のように先頭に「@」を置いた場合は""で囲まれた文字列の内容を表示するのではなく、リソース string の hello_world 項目で定義されている内容を表示します。

3. 直接文字列

文字列リソースを指定せずに、直接文字列を「android:text="ようこそ Android!"」のように指定することもできます。

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="ようこそAndroid!" />
```

本書では、直接文字列を指定する方式とします。

1-6　Android アプリの典型的な Java コードの意味

Android アプリを作って行く上で必要な Java の基礎的な知識を先に作成した MainActivity.java を参考に説明します。

```
package com.example.test1;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

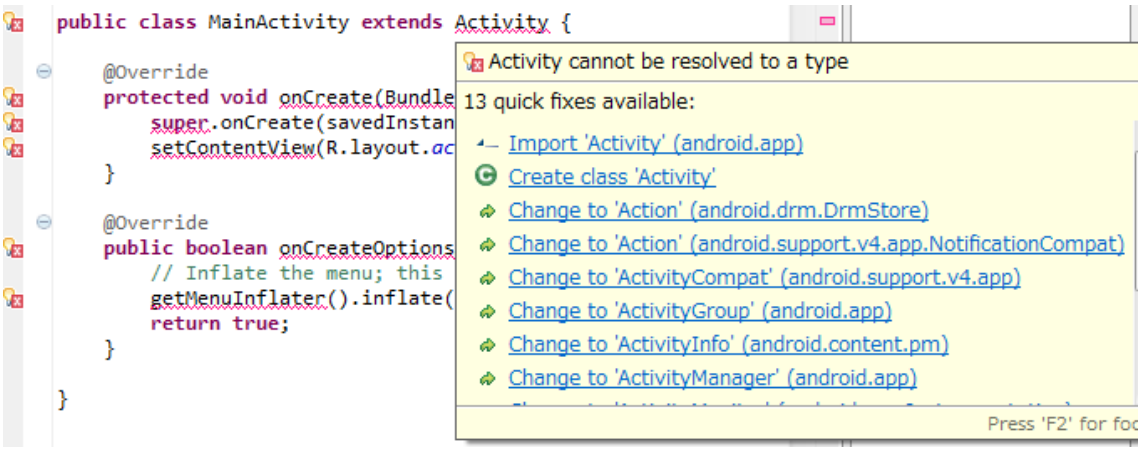
1.　package 文

「package com.example.test1;」でこのプログラム（MainActivity.java）が com.example.test1 という名前のパッケージに含まれるものであることを示します。

2.　import 文

プログラムで使用しているクラスを定義している API をインポートします。インポート（英語の意味は輸入）とは外部で定義されているクラスをこのプログラムで使用することをコンパイラに通知することです。この例では Activity クラスを使用するので「import android.app.Activity;」で Activity クラスをインポートしています。import 文の書式には

次の 2 つがあります。
①import パッケージ名.クラス名
指定したパッケージの指定したクラスだけをインポートします。
②import パッケージ名.*
指定したパッケージに含まれるすべてのクラスをインポートします。
たとえば 3 章で説明する Button や TextView などのウイジェットは「android.widget」パッケージで定義されています。これらのクラスをインポートするには、「import android.widget.Button」や「import android.widget.TextView」のように個々にインポートしてもよいし、「import android.widget.*」のようにまとめてインポートしてもよいです。
「注」自動インポート機能
もし「import android.app.Activity;」がないとプログラム中の「Activity」の箇所がエラーを示す赤の波線が付きます。そこにマウスを合わせるとエラーを解決するための候補が表示されますので、「Import 'Activity'(android.app)」を選択すると、「import android.app.Activity;」文が自動的に入ります。



3. Activity クラス

Android の画面を示す基本クラスで、ウイジェットやグラフィックスの表示、イベントの取得などのユーザインターフェース（UI）を提供します。

4. ユーザクラスの定義とクラス継承

Java のプログラムはクラスという単位で構成されます。クラスの定義は「class ユーザクラス名」で行います。このクラスが API のすでにあるクラスを継承する場合は「extends 元になるクラス名」とします。以下の例では Activity クラスを継承した MainActivity というユーザー定義クラスが定義されます。通常 1 つのアプリの中に複数のクラスがありますが、その中でそのアプリを代表するクラスには「public」指定をして外部から参照できるよ

うにします。ソースファイルもこの代表クラス名と同じ名前の「MainActivity.java」となります。

```
public class MainActivity extends Activity {
    クラスの内容
}
```

5. コメント

コメントは人間がプログラムの意味を理解しやすいように付ける注釈で、プログラム上（コンパイラ）はコメントは意味を持ちません。コメントは以下の 2 種類の書き方があります。/*と*/は複数行に渡ってコメントを書くことができます。//は 1 行だけのコメントです。

```
/*  コメント
    コメント
 */
// コメント
```

6. メソッドと引数

クラスに所属し、そのクラス内の各種処理を行うものをメソッドと呼びます。メソッドの定義は次のように行います。

```
アクセス属性 型 メソッド名(仮引数,仮引数,・・・){
    メソッドの内容
}
```

生成された MainActivity クラスには 2 つの Activity クラスに属するメソッドがデフォルトで生成されます。

1 つは onCreate メソッドで、Activity クラスが生成されるときに呼び出されます。先頭に「on」の付くメソッドは何らかのイベントの発生で呼びだれるメソッドです。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

もう 1 つは onCreateOptionsMenu メソッドでオプションメニューの作成時に呼び出されます。オプションメニューを使用しなければ必要ありませんので、このまま置いておいても良いし、削除しても構いません。本書では削除しています。

```
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
}
```

on～メソッドはイベントの発生で呼び出されますが、他のメソッドはユーザがプログラム中で呼び出します。ユーザがメソッドを呼び出すには「オブジェクト.メソッド(実引数, 実引数・・・);」とします。引数はメソッドとメソッドの呼び出し元との間でデータの授受を行うもので、メソッド定義部に指定するものを仮引数、メソッド呼び出し部に指定するものを実引数と呼びます。
たとえば以下の呼び出しでは super がオブジェクト、onCreate がメソッド、savedInstanceState が実引数です。

```
super.onCreate(savedInstanceState);
```

ところが以下の呼び出しでは「オブジェクト.」がありません。これは setContentView メソッドが、現在のクラス（Activity クラス）のメソッドであるため「オブジェクト.」を省略できるのです。あえてオブジェクトを明示するなら「this.」とします。

```
setContentView(R.layout.activity_main);
```

7. アクセス属性

クラスやオブジェクトが他のプログラムや自身のプログラム内からどのようにアクセスできるかをアクセス属性と呼び以下の 3 つがあります。オブジェクトに対しアクセス属性を指定しない場合は同じパッケージ内なら public と同じです。オブジェクトとは変数やクラスのオブジェクトを指しますが詳しいことは後で説明します。
①public
どこからでもこのクラスまたはオブジェクトを参照できる
②protected
オブジェクトを定義しているクラスとサブクラスから参照できる
③private
オブジェクトを定義しているクラスのみから参照できる

8. 型

人間は 5 と 5.0 は同じようなものと考えますが、コンピュータは前者を整数型(int)、後者を浮動小数点型(float あるいは double)として異なる型として扱います。型はオブジェクトとメソッドに指定します。メソッドの型はそのメソッドが呼び出し元に返すオブジェクトの型ですが、何も変えさないメソッドの型を特別に「void」型と呼びます。

9. @Override

@で始まる命令をアノテーション（annotation）と呼びます。アノテーションはコンパイラに対する指示を与える注釈という意味です。「@Override」を付けたメソッドはそのメソッドがスーパークラスのメソッドをオーバーライドしていることをコンパイラに明示しています。「@Override」はコンパイラに対する指示なので、置かなくてもよいのですが、「@Override」を付けることでオーバーライドされたメソッドであることが明確になるのと同時にメソッド名の単純な打ち間違いを防止するメリットがあります。ただし、オーバーライドではないメソッドにこの「@Override」を付けるとエラーとなります。

10. オーバーライド

スーパークラスで定義されているメソッドをサブクラスにおいて同じ名前（引数の個数と型も同じ）で再定義することをオーバーライドと呼びます。

11. コンストラクタ

コンストラクタは、そのクラスのオブジェクトが生成されるときに呼び出される特別な初期化メソッドでクラス名と同じ名前を付けます。コンストラクタは new 演算子を用いて呼び出します。

12. super

super はスーパークラスを示すオブジェクトで、サブクラスからスーパークラスのコンストラクタやメソッドを呼び出すときに使用します。「super.onCreate(savedInstanceState);」でスーパークラスの onCreate メソッドを呼び出しています。通常、スーパークラスからオーバーライドしたメソッドは先頭でスーパークラスのメソッドを呼び出して規定の処理をした後で、ユーザー独自の処理を置きます。引数の「savedInstanceState」は「public void onCreate(Bundle savedInstanceState)」で受けた引数をそのまま指定してしています。Activity クラスのメソッドについては後述します。

13. this

super がスーパークラスを示すオブジェクトなのに対し、this は自分自身のオブジェクトを指します。メソッドを呼び出す場合「オブジェクト名.メソッド名()」としますが、自分自身のメソッドを呼び出す場合オブジェクト名を省略して「メソッド名()」とできますが、自分自身のオブジェクトを明示したい場合は「this.メソッド名()」とします。
「setContentView(R.layout.main);」は自分自身の中で定義（継承している）しているメソッドなので、オブジェクトを指定していませんが、明示するなら「this.setContentView(R.layout.main);」となります。