■DroidSlot52 の解説　　配列や繰り返し処理を活用した例　5 章（P.197）

本書の中で少しずつ作ってきた DroidSlot アプリケーションのコードをさらに簡潔にしたのが、ここで示すコードです。イベントを Activity で処理するとともに、配列と繰り返し処理をうまく使って、コードを短くしています。

実行例は本書のサンプルと同じなので省略します。さっそく、コードを見てみましょう。

Java ▶ src/com.example.Sample/DroidSlotActivity.java

```
 1: package com.example.Sample;
 2:
 3: import java.util.Random;
 4:
 5: import android.app.Activity;
 6: import android.os.Bundle;
 7: import android.view.View;
 8: import android.view.View.OnClickListener;
 9: import android.widget.Button;
10: import android.widget.ImageView;
11: import android.widget.Toast;
12:
13: public class DroidSlotActivity extends Activity implements OnClickListener{ ……クリックイベントをアクティビティで処理する
14:     /** Called when the activity is first created. */
15:     int[] droidImageId = new int[]{R.id.droidimageid1, R.id.droidimageid2, R.id.droidimageid3}; ……ドロイドくんを表示するのに使う ImageView のリソース ID を配列に入れる
16:     int[] slotButtonId = new int[]{R.id.slotbutton1, R.id.slotbutton2, R.id.slotbutton3}; ……Button のリソース ID を配列に入れる
17:     int[] droidSide = new int[]{-1, -1, -1}; ……ドロイドくんの向き。表示されていないときは -1 とする
18:     ImageView[] droidImage = new ImageView[3]; …… ①3 つある ImageView の参照を配列として取り扱えるようにする
19:     Button[] b = new Button[3]; …… ②3 つある Button の参照を配列として取り扱う
20:     Button retryButton;
21:     final Random r = new Random();
22:     @Override
23:     public void onCreate(Bundle savedInstanceState) {
```

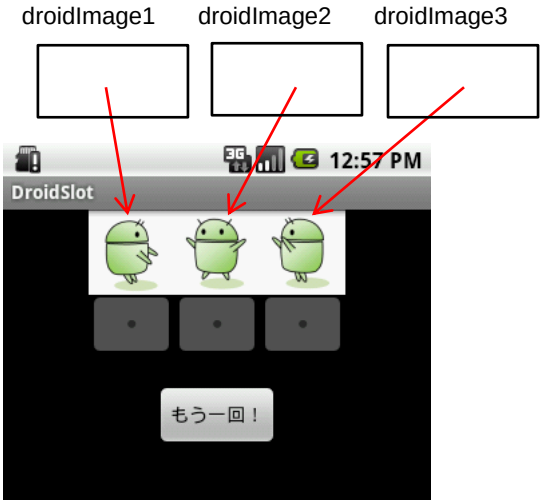
```
24:         super.onCreate(savedInstanceState);
25:         setContentView(R.layout.main);
26:         for(int i=0;i<3;i++){   …… ③繰り返し処理を使って、ImageView と Button をすべて
参照
27:             droidImage[i] = (ImageView)this.findViewById(droidImageId[i]);
28:             b[i] = (Button)this.findViewById(slotButtonId[i]);
29:             b[i].setOnClickListener(this); …… ④OnClickListener も順にすべて設定
30:         }
31:         retryButton = (Button) this.findViewById(R.id.retrybutton);
32:         retryButton.setOnClickListener(this);
33:         retryButton.setVisibility(View.INVISIBLE);
34:     }
35:     public void onClick(View v) {
36:          switch(v.getId()){ …… ⑤クリックされたウィジェットで場合分け
37:         case R.id.slotbutton1:
38:             flipImage(0); break;
39:         case R.id.slotbutton2:
40:             flipImage(1); break;
41:         case R.id.slotbutton3:
42:             flipImage(2); break;
43:         case R.id.retrybutton:
44:             for(int i=0;i<3;i++){
45:                 droidImage[i].setImageResource(R.drawable.star);
46:                 b[i].setEnabled(true);
47:                 droidSide[i] = -1;
48:             }
49:             retryButton.setVisibility(View.INVISIBLE);
50:         }
51:     }
52:     private void flipImage(int i){ …… 引数で指定された ImageView にドロイドくんを表示
53:         droidSide[i] = r.nextInt(4);
54:         droidImage[i].setImageResource(getDrawableId(droidSide[i]));
55:         checkSlot();
56:         b[i].setEnabled(false);
57:     }
58:     private int getDrawableId(int side){
```

```
59:         int[] drawableId = new int[]{R.drawable.droid_front, R.drawable.droid_back,
60:                 R.drawable.droid_left, R.drawable.droid_right};
61:         return drawableId[side];
62:     }
63:     private void checkSlot(){
64:         if(droidSide[0] == droidSide[1] && droidSide[0] == droidSide[2]){
65:             Toast.makeText(this, "おめでとう！揃いました",
Toast.LENGTH_SHORT).show();
66:             retryButton.setVisibility(View.VISIBLE);
67:         } else if(droidSide[0] != -1 && droidSide[1] != -1 && droidSide[2] != -1){
68:             retryButton.setVisibility(View.VISIBLE);
69:         }
70:     }
71: }
```

18 行目を見てください。これまでは、droidImage1、droidImage2、droidImage3 という別の変数を使って ImaveView ウィジェットの参照を取り扱ってきましたが、①で、それらを 1 つの配列で取り扱えるようにしました。droidImage[0]がこれまでの droidImage1 にあたります。

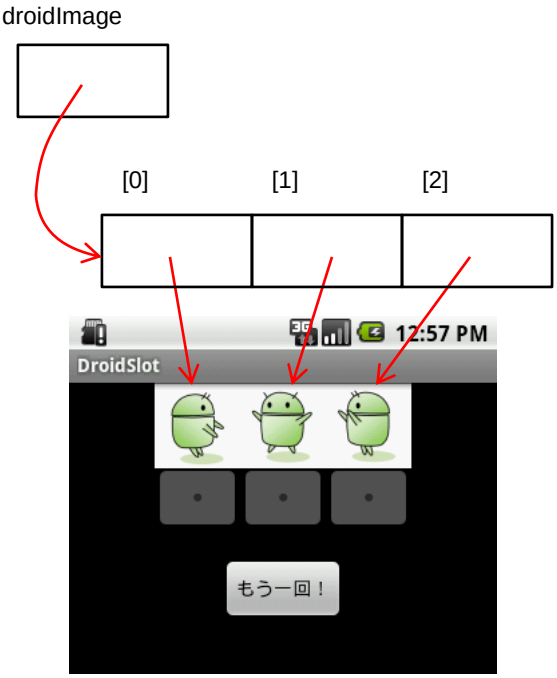
19 行目も同様のコードです。3 つある Button ウィジェットの参照も、ここで、b という 1 つの配列で取り扱えるようにします。b[0]がこれまでの b1 にあたるわけです。

図で表すと以下のような感じになります。最初の図がこれまでの方法です。別々の変数を使って ImageView を参照しています。



次の図が、配列を使ったここでのサンプルのイメージです。変数は droidImage だけですが、

配列になっているので、[0]などのインデックスを使って個々の要素を区別します。インデックスの値を 0 から 2 まで順に変えれば、ImageView の参照を順に取得したり、ImageView を順に操作したりできるといわうけです。



あとは、これまでのコードとほぼ同じです。何度も言うようですが、droidImage1 の代わりに droidImage[0]が使われ、b1 の代わりに b[0]が使われているだけのことです。

■GrapicsTest の解説　　四角や円などの図形を描く　　5 章（P.180）

Android アプリケーションでは、画面の上にさまざまなものが表示されるので、画面に表示されているものは必要に応じて再描画されなくてはなりません。たとえば、画面上に何かを表示したあと、その背後に回っていた部分が再び前面に表示されたとします。そこに四角や円などの図形があったとすると、（いったん背後に回って消されたわけですから、前面に戻ったときには）再描画する必要があります。

しかし、そのたびごとに、自分で四角や円などの図形を描画するのはとても面倒です。実際、そういう方法は使われません。その代わりに、onDraw メソッドに、図形を描画するメソッドを書いておきます。

Android アプリケーションでは、ビューの描画が必要になると onDraw メソッドが自動的に呼び出されるのです。したがって、**View クラスの onDraw メソッドをオーバーライド**し、Canvas クラスの drawRect メソッドや drawOval メソッドを使って、四角や円をそれぞれ描くようにしておけば、ビューの描画が必要になるたびに、それらのメソッドが実行され、図形が描画されます。

なお、強制的に描画を実行するには、ビューの invalidate メソッドを呼び出します。

まず、四角形を描画する例を使って、図形の描画方法を見ていきましょう。

●GraphicsTest の実行例

以下のような設定で新しいアプリケーションを作成してください。

項目名	設定する内容
プロジェクト名	GraphicsTest
ビルド・ターゲット	Android 2.1
アプリケーション名	GraphicsTest
パッケージ名	com.example.Sample
アクティビティ名	GraphicsTestActivity

View クラスの onDraw メソッドをオーバーライドするには、View クラスをサブクラス化する必要があります。GraphicsTestActivity.java ファイルを以下のように編集しましょう。

Java ▶ src/com.example.Sample/GraphicsTestActivity.java

```
 1: package com.example.Sample;
 2:
 3: import android.app.Activity;
 4: import android.content.Context;
 5: import android.graphics.Canvas;
 6: import android.graphics.Color;
 7: import android.graphics.Paint;
 8: import android.os.Bundle;
 9: import android.view.View;
10:
11: public class GraphicsTestActivity extends Activity {
12:     /** Called when the activity is first created. */
13:     @Override
14:     public void onCreate(Bundle savedInstanceState) {
15:         super.onCreate(savedInstanceState);
16:         MyView mv = new MyView(this); …… ③MyViewクラスのオブジェクトを作成する
17:         setContentView(mv); …… ④表示に使うビューとして、上で作ったオブジェクトを指定
18:     }
19: }
20: class MyView extends View{     …… ①Viewクラスの子クラスを作る（サブクラス化する）
21:     public MyView(Context context) {     …… コンストラクター
22:         super(context);          …… Viewクラスのコンストラクターをそのまま実行する
23:     }
24:     public void onDraw(Canvas c){          …… ②描画が必要になると自動的に呼び出される。
Canvasは描画のためのメソッドを使うためのクラス
25:         Paint p = new Paint();    …… 描画色やスタイルなどを利用するために使われるPaintク
ラスのオブジェクトを作る
26:         p.setColor(Color.rgb(255, 255, 0));          …… 描画色を黄色とする
27:         c.drawRect(10, 20, 80, 50, p);     …… CanvasクラスのdrawRectメソッドを使って四
角形を描く
28:     }
```

25 行目～27 行目が描画のためのコードです。この部分に何を描画したいかを書けば、ほかの部分の意味は分からなくてもいろいろな図形が描画できます。

しくみは以下の通りです。

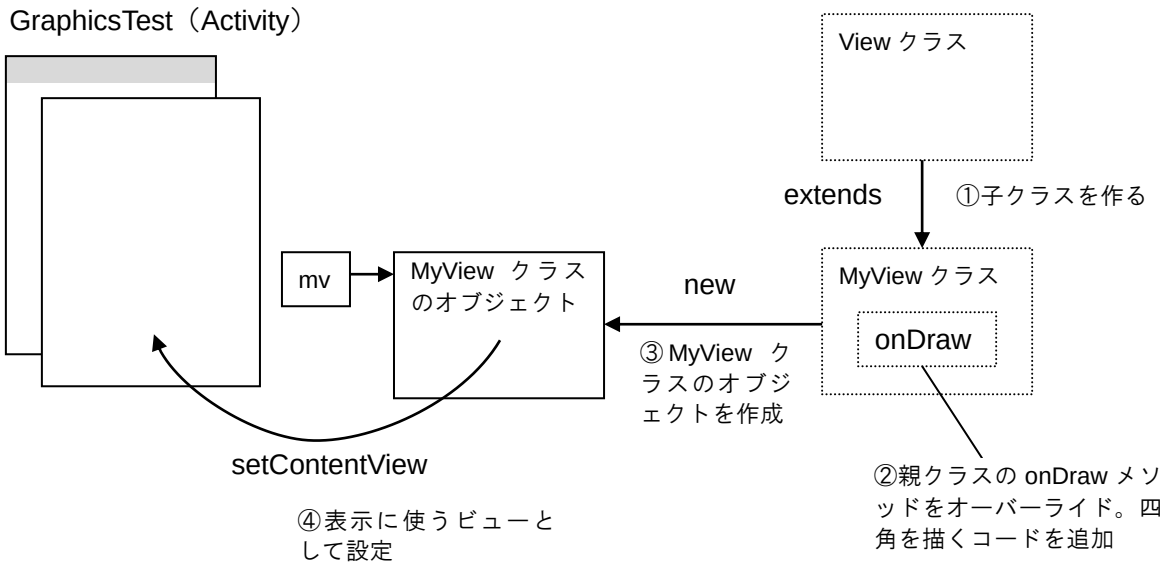
①では View クラスの子クラスとして MyView クラスを作成しています。これは②で onDraw メソッドをオーバーライドするためです。親クラスの持つ onDraw メソッドを書き換えて、MyView クラスでは描画の必要が生じたときに四角形を描くようにします。

③では MyView クラスのオブジェクトを作成し、mv という変数で参照できるようにしました。

④で、画面に表示するビューとして③で作ったオブジェクトを指定しています。これまでは main.xml で作成されたレイアウトを指定していましたが、このように View クラス（やその子クラス）のオブジェクトを指定することもできます。

＜ひとこと＞
setContentView メソッドの引数に指定できるのはビューです。main.xml で定義したレイアウトが指定できたのは、LinearLayout などのレイアウトが View の子クラスだからです。

これまでの説明を図にしてみると、以下のようになります。



}

次に、MyView クラスの中身について詳しく見ていきます。21 行目を見ると、MyView クラスの中に MyView メソッドが定義されていることに気づくと思います。このような、クラス名と同じ名前のメソッドはコンストラクターと呼ばれ、クラスからオブジェクトが作られるときに実行されます。

ここでは、super と書かれているので、親クラス（View クラス）のコンストラクターをそのまま実行します。オブジェクトが作られるときに、子クラス（MyView クラス）では特別なことはしない、ということになります。

24 行目には onDraw メソッドの定義が書かれています。これが、親クラスの onDraw メソッドをオーバーライドするためのコードです。親クラスの onDraw メソッドでは特に何も描画されませんが、子クラスの onDraw メソッドでは四角を描画するというわけです。onDraw メソッドに引数として渡される Canvas は図形を描画のためのメソッドを利用するためのクラスです。単純に「Canvas は描画の道具」と考えてもらってもさしつかえありません。

描画の必要が生じたら onDraw メソッドが自動的に呼び出され、そのときに描画の道具も渡してもらえるというわけです。

25 行目と 26 行目は後回しにして、27 行目を見てください。Canvas クラスの drawRect メソッドは四角形を描画するメソッドです。このメソッドには四角形の座標だけでなく、どんな色を使うかといったことも指定します。

```
c.drawRect(10, 20, 80, 50, p);
```

始点のＸ座標　始点のＹ座標　終点のＸ座標　終点のＹ座標　色などの描画の方

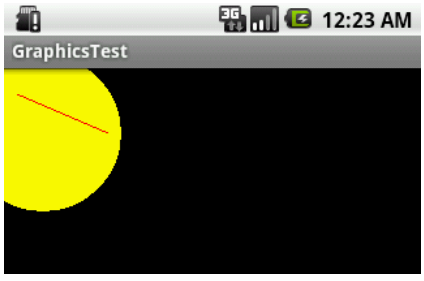
描画の方法としては Paint クラスのオブジェクトを指定します。Paint クラスでは、色のほか、透明度、フォントなども指定できます。そのために、25 行目と 26 行目で Paint クラスのオブジェクトを作成し、setColor メソッドで色を設定していたというわけです。

なお、MyView クラスは、GraphicsTestActivity.java ファイルとは別のファイルに書いてもかまいません。その場合は、ファイル名を MyView.java とし、クラス定義の最初の部分を、

```
public class MyView extends View{
```

とします。　**オマケアプリの GraphicsTest はこの方法を使って書いてあります。**また、四角形の代わりに円と線分を描くものにしてあります。実行例は以下のようになります。

●円と線分を描画した例



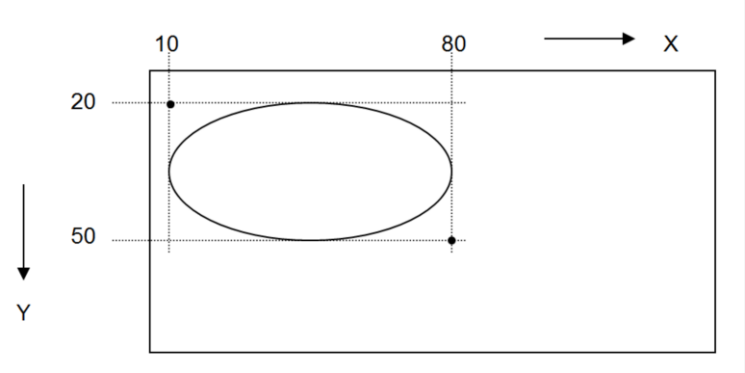
円や線分の描くためのメソッドについては次の＜ひとこと＞を見てください。

＜ひとこと＞

円を描画するには、drawOval メソッドを使います。例えば、

```
c.drawOval(new RectF(10, 20, 80 ,50), p);
```

と書けば、下の図のような円（楕円）が描画されます。RectF は浮動小数点数の座標を使って四角い範囲を表すクラスです。



drawOval メソッドでも幅と高さが同じになるようにすれば真円が描けますが、drawCircle メソッドを使えば、中心と半径の指定で真円が描けます。たとえば、

```
c.drawCircle(30, 50, 60, p);
```

とすると、中心の X 位置が 30、Y 位置が 50、半径が 60 の真円が描けます。

線分を描画するには、drawLine メソッドを使います。例えば、

```
c.drawLine(10, 20, 80 ,50, p);
```

とすると、(10,20)の位置から(80,50)の位置までの線分が引かれます。上の図の左上の小さな●から右下の小さな●までの線分になります。

ほかにも以下のような描画のメソッドが使えます。

・drawArc メソッド　　円弧を描画する
・drawLines メソッド　　複数の線を描画する
・drawRoundRect 角丸四角形を描画する
・drawPath メソッド　　指定したパス（複数の点や図形）を結ぶ線を描く
・drawText メソッド　　テキストを描画する
・drawTextOnPath メソッド　　パスに沿ってテキストを描画する

＜ひとこと＞

ボタンをクリックしたときに、図形を描画する場合には、ボタンの OnClickListener の onClick メソッドの中には、普通、drawRect などの描画メソッドは書きません。描画のメソッドはあくまでもここで見たように View の onDraw メソッドの中に書きます。onClick メソッドの中では、invalidate メソッドを呼び出し、再描画が必要なことをアプリケーションに知らせます（すると自動的に描画が実行されます）。b という変数が Button ウィジェットを参照しているものとすれば、以下のようなコードになります。

```
b.setOnClickListener(new OnClickListener(){
    public void onClick(View v) {
        mv.invalidate(); …… 再描画を指示する（自動的にonDrawメソッドが実行される）
    });
```

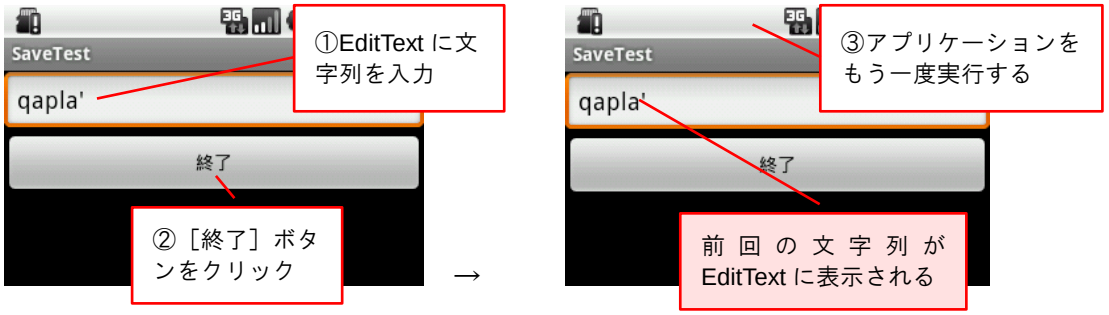
ただし、サブクラス化した View と Button ウィジェットを画面上にレイアウトしたり、描画される内容を設定するためのコードが必要になります。

■SaveTest の解説　　設定を保存する　5 章（P.201）

整数や文字列などの簡単な値を保存するには、SharedPreferences.Editor クラスの putInt メソッドや putString メソッドを使います。一方、保存した設定値を読みだすには、SharedPreferences クラスの getInt メソッドや getString メソッドを使います。

設定値の保存と読み出しの動作を確認するために、EditText ウィジェットに入力した文字列を保存する例を見てみましょう。

●SaveTest の実行例



以下のような設定で新しいアプリケーションを作成してください。

項目名	設定する内容
プロジェクト名	SaveTest
ビルド・ターゲット	Android 2.1
アプリケーション名	SaveTest
パッケージ名	com.example.Sample
アクティビティ名	SaveTestActivity

必要なウィジェットは EditText と Button だけなので、main.xml ファイルは以下のようになります。

xml ▶ res/layout/main.xml

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3:       android:orientation="vertical"
4:       android:layout_width="fill_parent"
5:       android:layout_height="fill_parent"
6:       >
7: <EditText
8:       android:id="@+id/freetext"
```

```
 9:      android:layout_width="fill_parent"
10:      android:layout_height="wrap_content"
11:      android:singleLine="true"
12:      />
13: <Button
14:      android:id="@+id/exitbutton"
15:      android:layout_width="fill_parent"
16:      android:layout_height="wrap_content"
17:      android:text="終了"
18:      />
19: </LinearLayout>
```

文字列を保存するので、putString メソッドを書いて設定を保存し、getString メソッドを使って設定を読み込みます。

設定を保存するタイミングは、アクティビティが一時停止したとき（onPause）とします。保存された設定を読み出すタイミングはアクティビティが作成されたとき（onCreate）としましょう。それぞれ、onPause メソッドと onCreate メソッドをオーバーライドするだけです。

Java ▶ src/com.example.Sample/SaveTestActivity.java

```
1: package com.example.Sample;
2:
3: import android.app.Activity;
4: import android.content.SharedPreferences;
5: import android.os.Bundle;
6: import android.view.View;
7: import android.view.View.OnClickListener;
8: import android.widget.Button;
9: import android.widget.EditText;
10:
11: public class SaveTestActivity extends Activity {
12:      /** Called when the activity is first created. */
13:      EditText t;
14:      @Override
15:      public void onCreate(Bundle savedInstanceState) {
```

```
16:         super.onCreate(savedInstanceState);
17:         setContentView(R.layout.main);
18:         SharedPreferences sp = getSharedPreferences("savedtext", MODE_PRIVATE); ……
①SharedPreferencesへの参照を得る
19:         t = (EditText) this.findViewById(R.id.freetext);
20:         t.setText(sp.getString("mykey", ""));          …… ②文字列を読み出す
21:         Button b = (Button)this.findViewById(R.id.exitbutton);
22:         b.setOnClickListener(new OnClickListener(){
23:             public void onClick(View arg0) {
24:                 finish();          …… ③アプリケーションを終了する
25:             }
26:         });
27:     }
28:     @Override
29:     protected void onPause() {
30:         super.onPause();
31:         SharedPreferences sp = getSharedPreferences("savedtext", MODE_PRIVATE); ……
④
32:         SharedPreferences.Editor sped = sp.edit();   ……⑤SharedPreferences.Editor への参
照を得る
33:         t = (EditText) this.findViewById(R.id.freetext);
34:         sped.putString("mykey", t.getText().toString());   ……⑥文字列を保存する
35:         sped.commit(); ……⑦変更を反映する
36:     }
37: }
```

●設定を読み出すためのコード

16 行目～28 行目は、アクティビティが作られるときに自動的に実行される onCreate メソッドです。ここには、起動時に設定を読み出す処理を書く必要があります。

18 行目の①では、getSharedPreferences メソッドを使って、設定の保存や読み出しに使う SharedPreferences クラスのオブジェクトへの参照を取得しています。

getSharedPreferences メソッドの書き方は以下の通りです。

```
getSharedPreferences("savedtext", MODE_PRIVATE);
```

設定の保存に使うファイル名　操作モード

操作モードに MODE_PRIVATE を指定すると、呼び出した Android アプリケーションのみから操作できるようになります。ほかの Android アプリケーションからの読み出しを許可するには MODE_WORLD_READABLE を指定し、ほかの Android アプリケーションからの書きこみを許可するには MODE_WORLD_WRITABLE を指定します。

20 行目の②では、getString メソッドで文字列を読み出しています。引数には識別のためのキー値と、読み出しができなかったときの既定値を指定します。識別のためのキーは、設定を保存するときに指定します（後述）。もし、設定がまだ保存されていなかったり、何らかの不具合があって読み出せなかったときには、""（空文字列）を返すものとしています。

読み出した文字列は setText メソッドを使って、EditText ウィジェットに表示します。以上で、設定の読み出しから表示までが完了です。

ここでは、23 行目と 24 行目にも注目してください。ボタンがクリックされたら、③の finish メソッドが実行されます。これでアプリケーションが確実に終了するわけです。

●設定を保存するためのコード

続けて、設定を保存するコードを見てみましょう。29 行目以降の onPause メソッドです。［HOME］ボタンが押されるなどして、アクティビティが一時停止するときには、onPause メソッドが自動的に呼び出されます。そこのタイミングで設定を保存しておきます。

31 行目の④は①と同じです。つまり、getSharedPreferences メソッドを使って、設定の保存や読み出しに使う SharedPreferences クラスのオブジェクトへの参照を取得します

32 行目の⑤では、edit メソッドを使って、設定を編集するための SharedPrefereneces.Editor クラスのオブジェクトへの参照を取得しています。設定を読み出すときには、SharedPreferences クラスのオブジェクトをそのまま使いましたが、設定を保存するときには、設定を編集するためのオブジェクトを利用します。

34 行目の⑥では、putString メソッドを使って、EditText ウィジェットに入力されている文字列を保存します。このとき、”mykey”というキーを付けて、どういう設定なのかが識別できるようにします。

35行目の⑦のcommitメソッドは重要です。これは、設定の変更を反映するためのメソッドです。つまり、このメソッドが実行されてはじめて、設定が保存されるのです。このコードを忘れると設定が保存されないので、注意が必要です。

＜ひとこと＞

試しに 35 行目の先頭に//を入力して、この行をコメントにしてしまい、commit メソッドが実行されないようにしてみてください。EditText に文字列を入力して、アプリケーションを終了してもその設定は保存されません。ふたたびアプリケーションを実行しても、EditText には、直前に入力した文字列は表示されないはずです（それ以前の設定が表示されるか何も表示されないでしょう）。