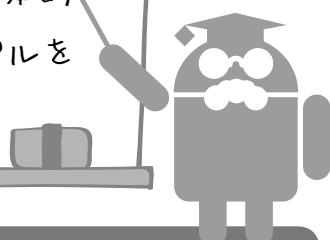


# 第 7 章

## Androidの APIについて学ぶ

著：小泉勝志郎

AndroidのAPIを学ぶ第一歩として、Androidの画面を構成する部品について学びます。Androidの画面部品の基本的な構造を理解しながら、代表的な画面部品のサンプルを見てみましょう。



## 7-1-1 各種のビューの使い方

この節では、以下の要素について学んで行きます。

- ①Androidの画面部品の基本
- ②長い文章をスクロールで表示
- ③画像を表示する
- ④「Spinner」を使って複数の選択肢から選択する
- ⑤「ListView」を使ったリスト表示

## 部品利用でAndroidアプリの画面を作る

APIは、「あるコンピュータプログラム（ソフトウェア）の機能や管理するデータなどを、外部の他のプログラムから呼び出して利用するための手順やデータ形式などを定めた規約のこと」（IT用語辞典「e-Words」）などと説明されます。最近では「Web API」のように、サービス間をつなぐ仕組みもAPIと呼ばれますが、ここでは「Androidの基本的な機能を利用する手段」と受け取ってください。

いくつかのAndroidアプリケーションを見ていると、長い表示ではスクロールしたり、複数の項目から選択できるような箇所では似たような動作をするものがあります。これは、アプリケーションの作成時に共通した部品を使っているためです。AndroidのAPIには、画面を構成するものが数多く用意されていますが、その中でもよく使う部品について、使い方を学んでいきましょう。ここで解説する部品を用いるだけでも、見栄えと使い勝手が両立したアプリの作成が可能となります。



## 7-1-2 ViewとViewGroup

Androidアプリケーションで表示される画面には、テキストやボタンといった「ビュー」(View)と呼ばれる部品が表示されています。これらの部品は、Activityに対して「setContentView」メソッドを使うことで配置できます。ここでは、部品として配置されるView及び「ViewGroup」について、まずは簡単に説明します。

「android.view.View」は、Androidの画面部品の基本となるクラスです。今まで学んできた画面部品はすべて、android.view.Viewを継承したサブクラスです。これらは「ウイジェット」とも呼ばれます。

```
java.lang.Object
└ android.view.View

public class View extends Object implements Drawable,
Callback, KeyEvent.Callback, AccessibilityEventSource
```

表1:android.view.Viewのクラス階層

Viewクラスには多くのサブクラスがあります。すでに学んでいるように、画面上に文字を表示したい場合は「android.widget.TextView」クラスを使います。プロジェクトを作ったときに「Hello world!」と表示していたのも、TextViewでした。TextViewのクラス階層は以下のようになります。

```
java.lang.Object
└ android.view.View
  └ android.widget.TextView

public class TextView extends View implements
ViewTreeObserver.OnPreDrawListener
```

表2:android.widget.TextViewのクラス階層

Androidの画面には、複数の画面部品が配置されています。通常、1つの画面に表示される部品は1つではなく、テキストボックスやボタンなどが複数並べて配置されています。ですが、ActivityでsetContentViewメソッドを使って配置できるビューは1つだけです。そこで、まずはビューの一種である「ViewGroup」に、ボタンなどの必要なビューを配置した上で、ViewGroupをActivityに追加します。つまり、複数の画面部品をまとめて扱う方法がViewGroupなのです。

ViewGroupでは、子ビューの配置方法などを指定して、ビューの追加ができます。ViewGroupは“abstract”な(抽象的な)クラスなので、実際にはViewGroupそのものではなく、ViewGroupのサブクラスである、「LinearLayout」、「FrameLayout」、「RelativeLayout」、「TableLayout」などのクラスを使ってビューを配置します。

ViewGroupは次のように定義されています。

```

java.lang.Object
└ android.view.View
  └ android.view.ViewGroup

public class ViewGroup extends View implements
ViewManager, ViewParent

```

表3:android.view.ViewGroup

ViewGroupもViewクラスのサブクラスであり、ビューの1つです。そして、View Groupクラスには追加するビューのレイアウト方法ごとに、複数のサブクラスが用意されています。たとえば「LinearLayout」クラスは、ViewGroupに追加したウェジットを縦または横に並べていきます。また「TableLayout」クラスの場合は、ViewGroupに追加したウェジットを表のように並べていきます。

このように、目的に応じたViewGroupを使用することで、Activityの中にビューを思い通りの形にレイアウトすることができます。また、ViewGroupもビューの一種ですので、ViewGroupの中に別のViewGroupを追加することもできます。階層のようにViewGroupを積み重ねていくことで、より複雑なレイアウトを実現することが可能です。

レイアウトについては後の講座で詳しく扱うので、ここではViewGroupのサブクラスの例として「ScrollView」を用いてみましょう。



## 7-1-3 ScrollView で文字数が多い日も安心！

まず、前回までの講義と同じように、プロジェクトを作成してみましょう。

アプリケーション名:	Scroll
プロジェクト名:	Scroll
パッケージ名:	jp.techinstitute.scroll
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表4:Scrollプロジェクトの構成

さて、プロジェクトを作成したばかりでは、何の変哲もない「Hello world!」です。ここで「strings.xml」をいじって文字数をすごく多くしてみましょう。name属性が「hello\_world」のところに、とにかくたくさんの文字を書いてみましょう。

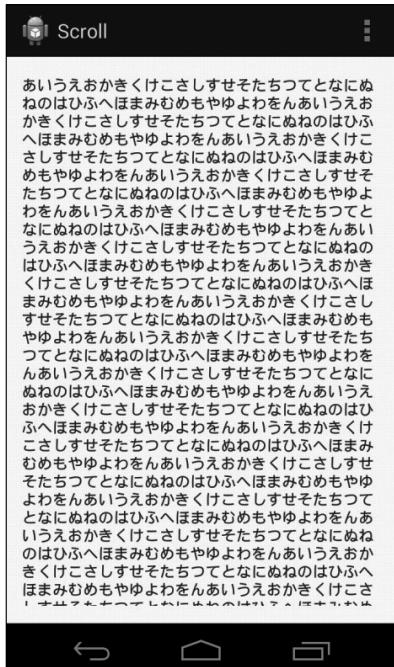
**strings.xml**

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

    <string name="app_name">SpinnerXml</string>
    <string name="hello_world">Hello world!</string><!-- ここにとにかくたくさんの中を書く -->
    <string name="action_settings">Settings</string>

</resources>
```

そうすると、次の**図1**のような画面になり、文字が多すぎて文章をすべて表示することができなくなっています。



**図1:** 文字が多すぎて文章が下に切れてしまっている

そこで、「fragment\_main.xml」を以下のように書き換えてみましょう。

**fragment\_main.xml**

```
<ScrollView 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"
    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="jp.techinstitute.scroll.MainActivity$PlaceholderFragme
nt" >
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</ScrollView>
```

上記のように書き換えたら、実行してみましょう。

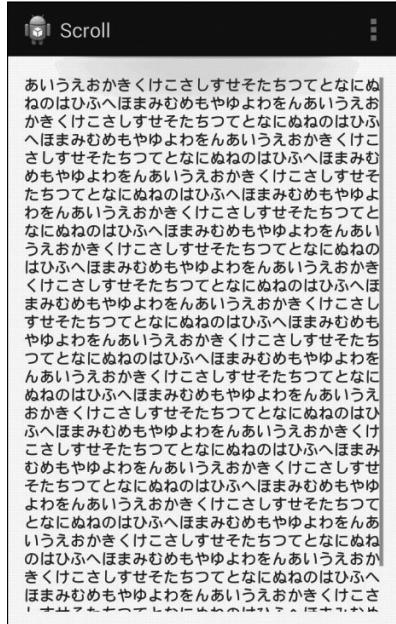


図2:右にスクロールバーが表示され、最後まで読める

図2のように、スクロールして全体を見られるようになりました。これで、文字数が多い場合でも大丈夫です。

### check!

#### スクロールビューについて

リスト化されたView(ListView)や、グリッド表示のView(Grid View)は、もともとスクロールするための機能を内包しています。そのため、ここで説明した処理は必要ありません。長いフォームなどの場合や、長い文章を読ませたい場合などに使うとよいでしょう。

画面をスクロールさせるためのViewは、2種類あります。縦方向の「ScrollView」と、横方向の「HorizontalScrollView」です。

ScrollViewはViewGroupですが、子要素は1つしか置けません。2つ以上の要素を置くとエラーになります。

複数置きたい場合は、ScrollViewの子要素に今後の講義で取り上げる「LinearLayout」や「RelativeLayout」などを配置します。そのレイアウトの下には、いくつでも子要素を配置できます。ちなみに、グラフィカルレイアウトからScrollViewを配置すると、最初から子要素にLinearLayoutが配置されます。



## 7-1-4 ImageView、アプリは見た目が9割！

今までの講義では、多少のレイアウト要素はあるにせよ、基本的には文字の表示ばかりで、ボタンがちょっと出てきた程度でした。しかし、人と同じくアプリも見た目が9割！です。アプリに画像を表示して、華やかにしましょう。ここでは、画像をリソースに組み込む方法と、画像URLから読み込む方法を紹介します。

アプリケーション名:	ImageId
プロジェクト名:	ImageId
パッケージ名:	jp.techinstitute.imageid
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表5:ImageIdプロジェクトの構成

ImageViewに、XMLからではなく、プログラムからリソースに登録された画像に対するリソースIDを指定して、ImageViewクラスのオブジェクトを作成してみましょう。

前準備として、画像ファイルを「drawable」で始まるフォルダーに置いて、リソースとして登録します。drawableで始まるフォルダーは、「drawable-hdpi」、「drawable-ldpi」など複数存在していますが、解像度によって適切なものが選ばれます。合致する解像度に指定された画像がない場合は、近い解像度のフォルダーにあるものが選択されます。画像のフォーマットはPNGファイルもしくはJPEGファイルである必要があります。

ここでは、とりあえずプロジェクト作成時にデフォルトで格納されている「ic\_launcher」を用います。

「fragment\_main.xml」を開いてください。ここで図3のようにグラフィカルレイアウトでImageViewを選択し、レイアウトへドラッグ＆ドロップしましょう。



図3: グラフィカルレイアウトで、ImageViewを選択

そうすると図4のようなダイアログが出てきて、画像を指定することができます。

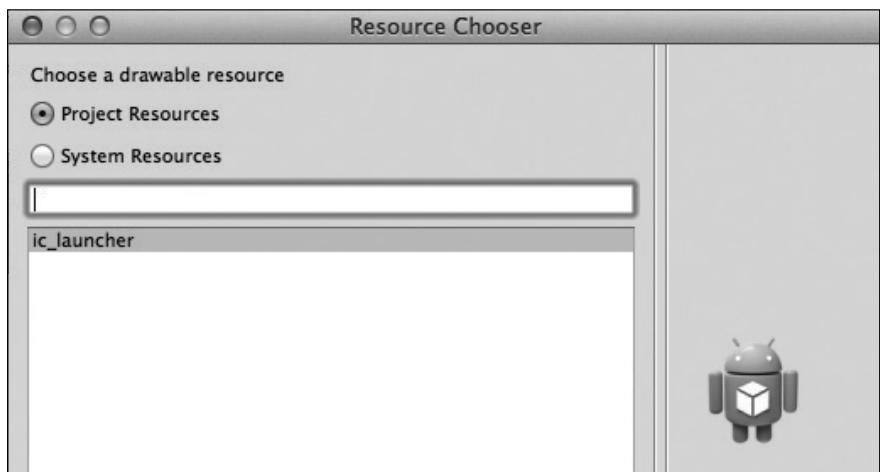


図4:画像選択のダイアログ

ここで「ic\_launcher」を選択すれば、画像の指定は終了です。xmlに警告が出てきますが、これは画像の説明を記入する「android:contentDescription」属性がないためで、無視してかまいません。

これで無事に作業は完了しましたが、学習のために、あえてプログラムから画像を指定してみましょう。まず、fragment\_main.xmlから「`android:src="@drawable/ic_launcher"`」を削除して、以下のように変更します。

#### fragment\_main.xml

```
<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"
    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="jp.techinstitute.imageid.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="50dp"
        />

</RelativeLayout>
```

これに「ImageView」クラスで用意されている「setImageResource」メソッドを使います。

```
setImageResource
public void setImageResource(int resid)
Parameters:
```

表6: setImageResource。『resid』は画像リソースに対応するリソースID

1番目の引数には、リソースIDを指定します。drawableで始まるフォルダーに「pic.png」というファイルを置いている場合は「R.drawable.pic」と指定します。今回はここに「ic\_launcher」を指定します。実際にMainActivity.javaを変更してみましょう。

#### MainActivity.java

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_main, container,
        false);

    // fragment_mainから ImageView オブジェクトを取得
    ImageView iv = (ImageView) rootView.findViewById(R.id.imageView1);

    // ImageViewに ic_launcher をセット
    iv.setImageResource(R.drawable.ic_launcher);

    return rootView;
}
```

では実行してみましょう。図5のように画像が表示されたでしょうか？

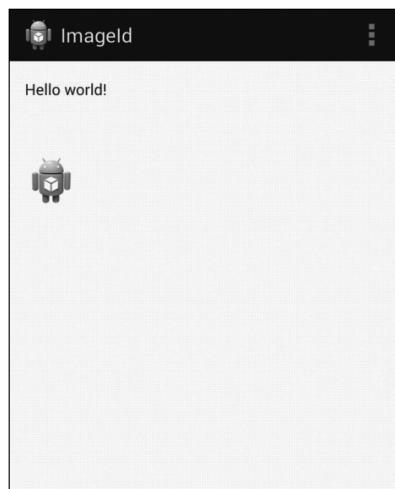


図5: 実行した結果、ic\_launcherの画像が表示されている



#### インターネット上の画像の利用について

Androidでは、インターネット上にある画像を取得して、表示することもできます。ただし、そのやり方は単純ではなく、後の講義で学ぶ「非同期処理」が必要となります。ネット上には、簡単に記述できるように見えるサンプルコードがよくあります。ですが、

Android 3.0以降ではメインスレッドからHTTP通信(ウェブの通信)を行えないようになっているため、そのままでは実行できません。



## 7-1-5 独自の情報収集

Androidには、複数の選択肢から1つを選択するための機能として、「スピナー」(Spinner)というものがあります。ウェブでのプルダウンメニューに当たるもので、都道府県を一覧から選ぶなど、Spinnerの名前は知らずとも、みなさんもよく目にしていると思います。ここでは2つのサンプルを用いて、Spinnerへの選択肢のセットとその取得の方法、イベントのキャッチについて学びます。

まずはXMLファイルから選択肢をSpinnerにセットする方法、そしてSpinnerで選択された選択肢を取得するサンプルを見て行きましょう。このサンプルアプリは、Spinnerで選択したものを、「Toast」で表示します。

アプリケーション名:	SpinnerXml
プロジェクト名:	SpinnerXml
パッケージ名:	jp.techinstitute.spinnerxml
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表7:SpinnerXmlプロジェクトの構成

fragment\_main.xmlファイルを開くと、図6のようにグラフィカルレイアウトにSpinnerが表示されます。

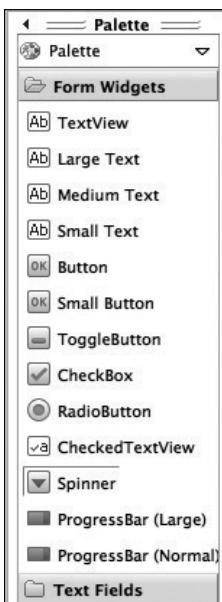


図6: グラフィカルレイアウトで、Spinnerを選択

このグラフィカルレイアウトから、Spinnerをドラッグ＆ドロップしてください。ただし、これだけでは足りません。Spinner要素の「prompt」属性にはポップアップで表示

されるダイアログのタイトルを、「entries」属性には選択肢の「配列」を記載しなければなりません。選択肢は複数あるため、これまでのstring要素と異なり、複数個のまとまりを記述できる「string-array」要素を用います。

#### fragment\_main.xml

```
<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"
    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="jp.techinstitute.spinnerxml.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <Spinner
        android:id="@+id/spinner1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="50dp" />

</RelativeLayout>
```

ただし、このままではfragment\_main.xmlはエラーのままでです。そこでstrings.xmlに、足りない項目であるentry属性に入れるためのstring-array要素とprompt属性用のstring要素を足します。

#### strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">SpinnerXml</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string-array name="fruits">
        <item>リンゴ</item>
        <item>みかん</item>
        <item>梨</item>
        <item>ドリアン</item>
    </string-array>
    <string name="spinnerSample">くだもの選択</string>
</resources>
```

とりあえずここまで実行してみましょう。図7のようにSpinnerが表示されるはずで

す。タップしてみましょう。



図7:Spinnerをタップすると、  
このようにリスト表示される

まだ選択時のイベントを記述していないので、選択しても何も起りません。MainActivity.javaに、イベント処理を記述して行きましょう。以下のソースコードをMainActivity.java内のonCreateViewメソッドに反映してください。

#### MainActivity.java

```
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup container,  
    Bundle savedInstanceState) {  
    View rootView = inflater.inflate(R.layout.fragment_main, container,  
        false);  
  
    // fragment_main から Spinner オブジェクトを取得  
    Spinner spinner = (Spinner) rootView.findViewById(R.id.spinner1);  
    spinner.setOnItemSelectedListener(new OnItemSelectedListener() {  
  
        // アイテムが選択された時の動作  
        @Override  
        public void onItemSelected(AdapterView<?> parent, View view, int position, long id)  
        {  
            // 引数の parent がイベントが発生した Spinner のオブジェクト  
            Spinner spinner = (Spinner) parent;  
            String str = spinner.getSelectedItem().toString();  
            Toast.makeText(getActivity(), str + " が選択されました。", Toast.LENGTH_SHORT).show();  
        }  
  
        // 何も選択されなかつた時の動作  
        @Override  
        public void onNothingSelected(AdapterView<?> arg0) {  
  
        }  
    });  
  
    return rootView;  
}
```

では、あらためて実行してみましょう。今度は、選択肢を選ぶとその内容がToastで図8のように表示されます。上手く行きましたか？ソースコード中に「Item」（アイテム）という単語がありますが、これは選択肢のことです。



図8:Spinnerから選択すると、その内容がToastで表示される

さて、ソースコードの中を見て行きましょう。まずSpinnerのイベントですが、これは「setOnItemSelectedListener」メソッドに「android.widget.AdapterView.OnItemSelectedListener」を実装したクラスを引数として渡すと、選択時に実装クラスのonItemSelectedメソッドが呼ばれます。

ここでは「匿名クラス」と呼ばれる手法を使って「OnItemSelectedListener」の実装を行っています。「onItemSelected」メソッドは第1引数parentがイベントの発生したListViewのオブジェクト、第2引数viewが選択されたアイテム、第3引数positionが何番目の選択肢が選択されたか、第4引数idがリソースIDとなっています。この渡ってくる引数を利用して、選択された際の処理を記述します。

どの選択肢が選択されているかは、Spinnerの「getSelectedItem」メソッドで分かります。戻り値の定義がObjectなので、toStringを加えて文字列化しましょう。

続いて、Spinnerの中身をプログラムからセットする方法を学びましょう。「連絡先の名前一覧がSpinnerに表示される」のような、「状態によって選択肢の中身が変わる」場合は、プログラムから選択肢をセットする必要があります。連絡先は増えたり減ったりしますからね。「状態によって選択肢の中身が変わる」ことを「動的」と言います。では、まずプロジェクトを作成しましょう。

「匿名クラス」とは“その場限りの実装”を実現するための手法で、イベント処理の際にはよく用いられます。スーパークラスやインターフェイスに対して「newスーパークラス名( コンストラクタの引数 ) { /\* サブクラスとしての実装 \*/ }」とすることで、「名前のない」クラスを作ることができます。

アプリケーション名:	SpinnerAdapter
プロジェクト名:	SpinnerAdapter
パッケージ名:	jp.techinstitute.spinneradapter
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表8:SpinnerAdapterプロジェクトの構成

SpinnerXmlの時と同じように、fragment\_main.xmlでSpinnerをドラッグ＆ドロップしてください。SpinnerXmlのときと違ってプログラムから内容をセットするため、fragment\_main.xmlではSpinnerの中身を定義しません。strings.xmlも修正しません。

#### fragment\_main.xml

```
<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"
    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="jp.techinstitute.spinneradapter.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <Spinner
        android:id="@+id/spinner1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="50dp" />

</RelativeLayout>
```

ソースコードからアイテムをセットするには「`android.widget.ArrayAdapter`」クラスを用います。このArrayAdapterのオブジェクトに、「`add`」メソッドで要素を追加して行きます。SpinnerXml同様に、`MainActivity.java`の`onCreateView`メ

ソッドの記述を以下のようにします。イベント処理はSpinnerXmlで行っているので、ここではイベント処理記述はしていません。

```
MainActivity.java
```

```
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup container,  
    Bundle savedInstanceState) {  
    View rootView = inflater.inflate(R.layout.fragment_main, container,  
        false);  
  
    // ArrayAdapter オブジェクトを作成して選択肢アイテムをセット  
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(getActivity(),  
        android.R.layout.simple_spinner_item);  
    adapter.add("りんご");  
    adapter.add("みかん");  
    adapter.add("梨");  
    adapter.add("ドリアン");  
  
    // Spinner に作成した ArrayAdapter をセット  
    Spinner spinner = (Spinner) rootView.findViewById(R.id.spinner1);  
    spinner.setAdapter(adapter);  
  
    return rootView;  
}
```

同じように、図9の形で展開されますね。ちなみに、ArrayAdapterについている<String>は、「ジェネリクス」と呼ばれるものです。このArrayAdapterに入るものは、すべてString型であるという意味です。



図9: プログラムからセットしても、Spinnerをタップする  
とこのように表示される



## 7-1-6 ListViewは応用範囲がひろい！

「ListView」は非常に多く用いられる画面部品です。Android端末で表示されるお馴染みの設定画面でも、ListViewが使われています。ここでは、シンプルなListViewに配列で内容を設定する方法、そしてListViewをカスタマイズして表示する方法を学びます。

最初のサンプルでは、ListViewの項目設定は配列で記述しています。項目をクリックすると、Toastで選択内容を表示するというものです。動作もコードもSpinner XMLでやったものに似ているので、理解しやすいかと思います。

アプリケーション名:	ListViewClick
プロジェクト名:	ListViewClick
パッケージ名:	jp.techinstitute.imageid
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表9:ListViewClickプロジェクトの構成

まずは画面を作るために、fragment\_main.xmlを開きましょう。図10のように、ListViewはグラフィカルレイアウトではCompositeのところにあります。これをそのままドラッグ＆ドロップで画面に配置してください。

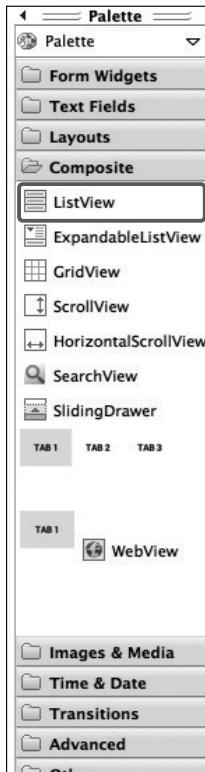


図10: グラフィカルレイアウトの「Composite」から、ListViewを選択

このとき、fragment\_main.xmlのソースコードは、以下のようにになります。

### fragment\_main.xml

```
<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"
    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="jp.techinstitute.listviewclick.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_below="@+id/textView1" />

</RelativeLayout>
```

そして、画面の外観は図11のようになります。

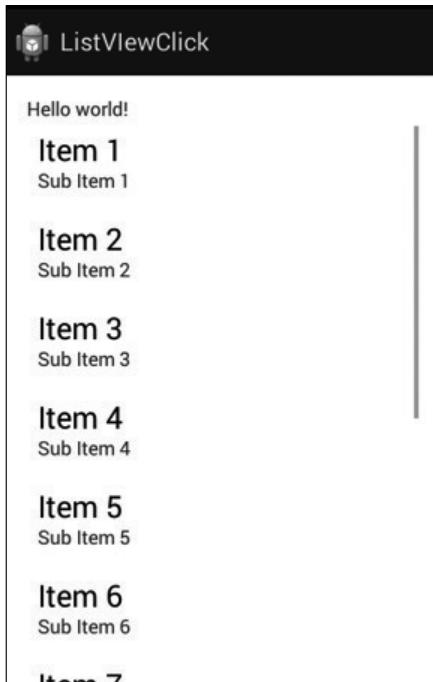


図11:画面上でのListViewの見え方

この時点でも実行しても、ListViewには何も表示されません。内容を表示するためにはMainActivity.javaにListViewの内容セット処理を記述しなければなりません。このサンプルでは、ListViewにセットするのは文字列のみです。

### MainActivity.java

```
@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup container,  
    Bundle savedInstanceState) {  
    View rootView = inflater.inflate(R.layout.fragment_main, container,  
        false);  
  
    //ListViewに表示させる項目を記述  
    String[] item = { "りんご", "みかん", "梨", "ドリアン" };  
    // ArrayAdapter を用いて item( 上で記述した項目 ) を ListView にセット  
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(getActivity(),  
        android.R.layout.simple_list_item_1, item);  
  
    ListView listView = (ListView) rootView.findViewById(R.id.listView1);  
    listView.setAdapter(adapter);  
  
    listView.setOnItemClickListener(new OnItemClickListener() {  
  
        @Override  
        public void onItemClick(AdapterView<?> parent, View view,  
            int position, long id) {  
            String item = (String) parent.getItemAtPosition(position);  
            Toast.makeText(getApplicationContext(), item + " を選択しました。", Toast.LENGTH_SHORT).  
show();  
        }  
  
    });  
  
    return rootView;  
}
```

では実行してみましょう。行を選択すると、図12のようにその内容が表示されます。



図12:ListViewでの項目と選択時の表示

ソースコードを解説します。「ArrayAdapter」を使うのはSpinnerAdapterのときと同じですね。SpinnerAdapterでは1項目ずつ足していましたが、ここでは配列でまとめて定義し、それをセットしています。

ListViewのイベントですがこれは「setOnItemClickListener」メソッドに「android.widget.AdapterView.OnItemClickListener」を実装したクラスを引数として渡すと、選択時に実装クラスの「onItemClick」メソッドが呼ばれます。onItemClickメソッドは、第1引数parentがイベントの発生したListViewのオブジェクト、第2引数viewが選択されたアイテム、第3引数positionが何番目の選択肢が選択されたか、第4引数idがリソースIDとなっています。この渡ってくる引数を利用して選択された際の処理を記述するのは、Spinnerと同様です。

ListViewにSpinnerに似た印象があるかもしれませんね。ListViewではカスタマイズにもチャレンジします。

ここでは、画像が左に、そして文字列が右に表示されるTwitterもどきのサンプルを作成してみます。このサンプルでは、作成するファイル数が多いので気を抜かないように！

アプリケーション名:	ListViewCustom
プロジェクト名:	ListViewCustom
パッケージ名:	jp.techinstitute.listviewcustom
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表10:ListViewCustomプロジェクトの構成

ListViewClickのときと同様にfragment\_main.xmlを開き、ListViewを足しましょう。

ここからは、独自に作る部分になります。まず、ListView項目のレイアウトを作成します。項目に用いるレイアウトも、同じくXMLで記述できます。ここで記述内容が、リストの各々の行のレイアウトということになります。ここでは、「item\_layout.xml」というファイル名にします。図13のようにlayoutフォルダーを右クリックし、「New」→「Android XML File」と選択してください。

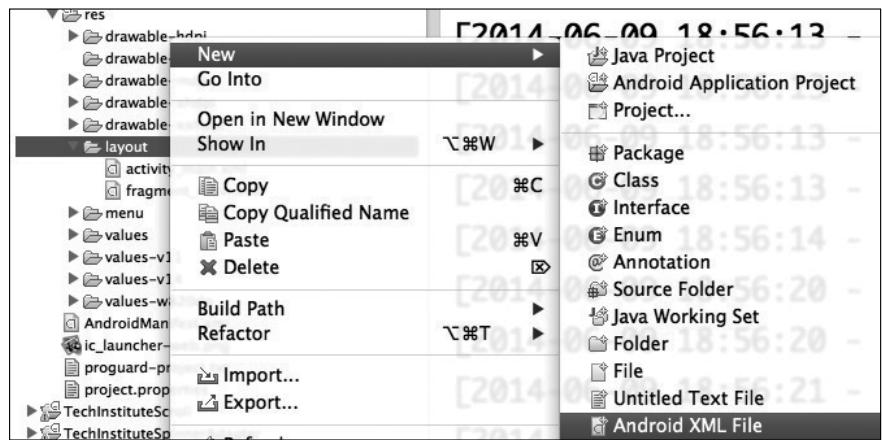


図13:メニューからAndroid XML Fileを追加

選択すると、図14のようなダイアログが出てきます。ここでは、「Resource Type」に「Layout」、「File」に「item\_layout」、「Root Element」に「LinearLayout」と入力してください。



図14:ListViewXML作成ダイアログに、ファイル名等を入力

作成したら、以下のようにitem\_layout.xmlの中身を書き換えてください。ImageViewとTextViewが足され、LinearLayoutのandroid:orientationが「horizontal」になっています。つまり横並びで表示されるようにしています。

**item\_layout.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />
</LinearLayout>
```

続いてデータを格納するための「`ItemData`」クラスを作成します。以前の講義で学んだ、クラスの新規作成を用います。このクラスは、各行に表示する画像とテキストを格納するためのものです。

**ItemData.java**

```
package jp.techinstitute.listviewcustom;

import android.graphics.Bitmap;

public class ItemData {
    private Bitmap _image;
    private String _text;

    public void setImage(Bitmap image) {
        _image = image;
    }

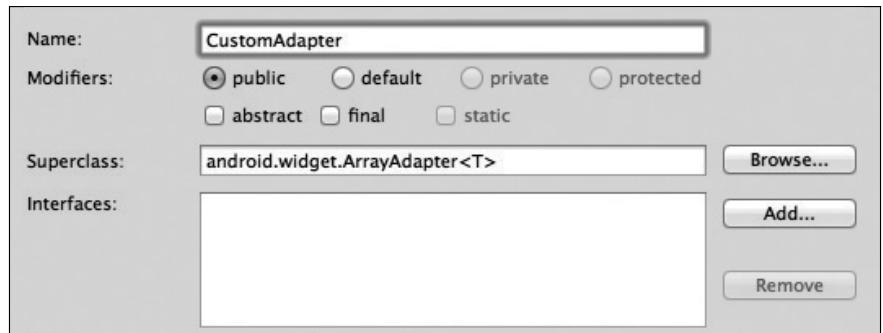
    public Bitmap getImage() {
        return _image;
    }

    public void setText(String text) {
        _text = text;
    }

    public String getText() {
        return _text;
    }
}
```

先ほどは、`ArrayAdapter<String>`のように文字列だけを表示していましたが、`ListView`をカスタマイズして表示するために、独自のアイテム用のクラス`ItemData`を扱える「`CustomAdapter`」クラスを作成します。このクラスは`ArrayAdapter`を継承することで、`ListView`にセットが可能になります。

早速作成してみましょう。先ほどと同じように、まず`CustomAdapter`クラスを作成してください。**図15**のようにSuper Classに`android.widget.ArrayAdapter`を指定しましょう。



**図15:**Super Classに「`android.widget.ArrayAdapter`」を指定

そして、`CustomAdapter.java`のコードを、以下のように記述します。

## CustomAdapter.java

```
package jp.techinstitute.listviewcustom;

import java.util.List;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class CustomAdapter extends ArrayAdapter<ItemData> {
    // アイテムのレイアウト取得用
    private LayoutInflater layoutInflater_;

    public CustomAdapter(Context context, int textViewResourceId, List<ItemData> objects) {
        super(context, textViewResourceId, objects);

        layoutInflater_ = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // 表示する行 (position) のデータを得る
        ItemData item = (ItemData)getItem(position);

        // convertViewは使い回しされている可能性があるので null の時だけ新しく作る
        if (null == convertView) {
            convertView = layoutInflater_.inflate(R.layout.item_layout, null);
        }

        // ItemData のデータを各 View にセットする
        ImageView imageView;
        imageView = (ImageView)convertView.findViewById(R.id.image);
        imageView.setImageBitmap(item.getImage());

        TextView textView;
        textView = (TextView)convertView.findViewById(R.id.text);
        textView.setText(item.getText());

        return convertView;
    }
}
```

CustomAdapter.javaのソースコードを解説します。ArrayAdapterクラスにある「getView」メソッドの戻り値が、ListViewの各行に表示するアイテムの「表示」となります。つまり、このgetViewメソッドをオーバーライドして、目的通りにカスタマイズされた見た目にすれば良いわけです。具体的には、「CustomData」クラスから必要なデータを取り出して、Viewを作る処理を記述します。

このgetViewメソッドは、各行を表示しようとした時に呼ばれます。どの行を表示しようとしているかは引数の「position」で分かれます。

処理の流れは、まずgetItemメソッドを利用してListの中から表示しようとしている行のアイテムを取得します。次に、「convertView」が「null」かどうかを判断しています。スクロールするほどの行がある場合は、メモリーの消費を抑えるためにViewを使い回すためです。そのため、nullでない場合は使い回して、nullの場合のみ新しくViewを生成します。

あとはitem\_layout.xmlから作成したレイアウトにあるViewの各要素に、Item Dataオブジェクトに格納されている画像と文字列をセットし、それをgetViewメソッドの戻り値としてリターンします。

ここまで来たらあと一步! MainActivity.javaの記述です。onCreateViewメソッドに次の記述を加えます。カスタマイズしたアイテムのデータを渡すために、そのItemDataのクラスのListを作成して、新しく作った ArrayAdapterの継承クラスのコンストラクタに与えて生成します。そして、ListViewの「setAdapter」メソッドでセットすれば完了です。

## MainActivity.java

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.fragment_main, container,
        false);

    // リソースに準備した画像ファイルから Bitmap を作成しておく
    Bitmap image;
    // 画像は今回デフォルトで格納されている ic_launcher を使用
    image = BitmapFactory.decodeResource(getResources(), R.drawable.ic_launcher);

    // データの作成
    List<ItemData> items = new ArrayList<ItemData>();
    ItemData item1 = new ItemData();
    item1.setImaga(image);
    item1.setText("リンゴ");

    ItemData item2 = new ItemData();
    item2.setImaga(image);
    item2.setText("みかん");

    ItemData item3 = new ItemData();
    item3.setImaga(image);
    item3.setText("梨");

    ItemData item4 = new ItemData();
    item4.setImaga(image);
    item4.setText("ドリアン");

    items.add(item1);
    items.add(item2);
    items.add(item3);
    items.add(item4);

    // CustomAdapter にデータのリストをセット
    CustomAdapter customAdapater = new CustomAdapter(getActivity(), 0, items);

    ListView listView = (ListView)rootView.findViewById(R.id.listView1);
    listView.setAdapter(customAdapater);

    return rootView;
}
```

では、実行してみましょう。



図16:カスタマイズしたListViewが完成

図16のように、ちょっとTwitterアプリっぽい見た目の画面を作れました。

## まとめ

今回の講義ではAndroidの画面を構成する部品について、これまでの講義で出てきていないものを中心にまとめ、また画面構成の基本となるViewとViewGroupについて学びました。Viewの応用となる独自Viewの作成、およびViewGroupの代表となるLayoutについては今後の講義で学びます。

- ・ **ViewGroupの簡単なサンプルとしてのScrollView。**テキストが長くなる場合やフォームにスクロールが必要な場合に重宝します
- ・ **画像を表示するImageView。**これがあるのとないのでは、画面の華やかさが違います。リソースに画像を入れて表示する方法も学習
- ・ **複数の選択肢を表示する方法としてのSpinner。**XMLのみで簡単に選択肢を記述する方法と内容を後から変更できるプログラムによる指定方法を学びました
- ・ **Androidで非常に頻繁に見かける画面部品であるListView。**内容のセット方法とイベントの取得方法、そしてカスタマイズしたListViewの表示。これによってアプリで表現できる幅が大きく広がります

# 7-2 AndroidのAPIについて学ぶ(2)

著：小泉勝志郎

第7章

AndroidのAPIについて学ぶ

7-1では、AndroidのAPIを学ぶ第一歩として、Androidの画面部品について学びました。この節では、Androidの新しい画面構築法である「Fragment」について学んでいきます。



## 7-2-1 Fragmentって何？

この節では、以下の要素を学んでいきます。

### ①Fragmentの基本

- ②スマホとタブレットでの画面切り替え
- ③タブによる画面切り替え
- ④ダイアログ画面の表示

「Fragment」は、サイズによって大きくUIが異なる画面を実現するために生まれたもので、「Android Development Tools r22.6.1」以降、EclipseでのAndroidプロジェクト自動生成コードも、Fragmentを用いる前提となっています。タブ表示やアラートダイアログもFragmentを利用したものになっています。

## Fragmentの基本を知る

Android端末には、さまざまな画面サイズのものがあります。またスマートフォンだけではなくタブレットもあり、さらには横向きなのか縦向きなのかといった表示の違いもあります。すべての端末に向けて同じUI(ユーザーインターフェイス)というのでは、不都合が生じることもあるでしょう。このため、アプリの画面設計で意識しておくべき重要な項目に、「複数の異なるタイプのディスプレイでも等しく使いやすい画面設計」があります。

たとえば、「タブレットでは左にリストがあって右に詳細のある画面だが、スマートフォンではリスト画面から項目を選択すると詳細画面へ移動するように変わる」というUIをよく見かけませんか？

以前は、そういったUIを実現するのに、レイアウトを解像度や端末の種別ごとに作成することで対応していました。画面サイズ等が大きく異なる場合には、それぞれに適した操作性を提供する必要があり、プログラム内で処理を切り分けっていました。

ですが、Android 3.0で追加された「Fragment」(フラグメント)によって、よりスマートに解決できるようになりました。Fragmentとは、英語で“破片”という意味です。Fragmentを活用することで、多様なディスプレイタイプに合わせたレイアウトを

作成できます。

また、Fragmentクラスには画面のパーツ機能だけではなく、拡張されたFragmentクラスも提供されています。よく見かけるアラートダイアログも、Fragmentで書くようになっているのです。この節では、そのDialogFragmentを取り上げます。

クラス名	概要
<b>DialogFragment</b>	アラート等のダイアログを提供するために拡張されたFragmentのサブクラス
<b>ListFragment</b>	一覧(ListView)の表示をするために拡張されたFragmentのサブクラス
<b>PreferenceFragment</b>	アプリケーションの設定情報を扱うために拡張されたFragmentのサブクラス
<b>WebViewFragment</b>	ブラウザ(WebView)のために拡張されたFragmentのサブクラス

表1: フラグメントのサブクラス

Fragmentを使用する場合は、Activityが画面全体を表し、FragmentはそのActivity上に配置されるパーツ、といった構造になります。先ほどのタブレットとスマートフォンでの表示の違いの例で言えば、「メニューFragment」と「詳細Fragment」があって、タブレットでは両方が1画面内に、スマートフォンではそれぞれ別画面に表示されていることになります。

Activityのライフサイクルを覚えていますか? Fragmentは、以前学んだActivityのライフサイクルとは異なる、独自のライフサイクルを持っています。このため、Fragment自体が入力イベントを受け取ったり、Activityの実行中にFragmentを追加したり取り除いたりすることができます。Fragmentのライフサイクルを記述したのが、図1です。

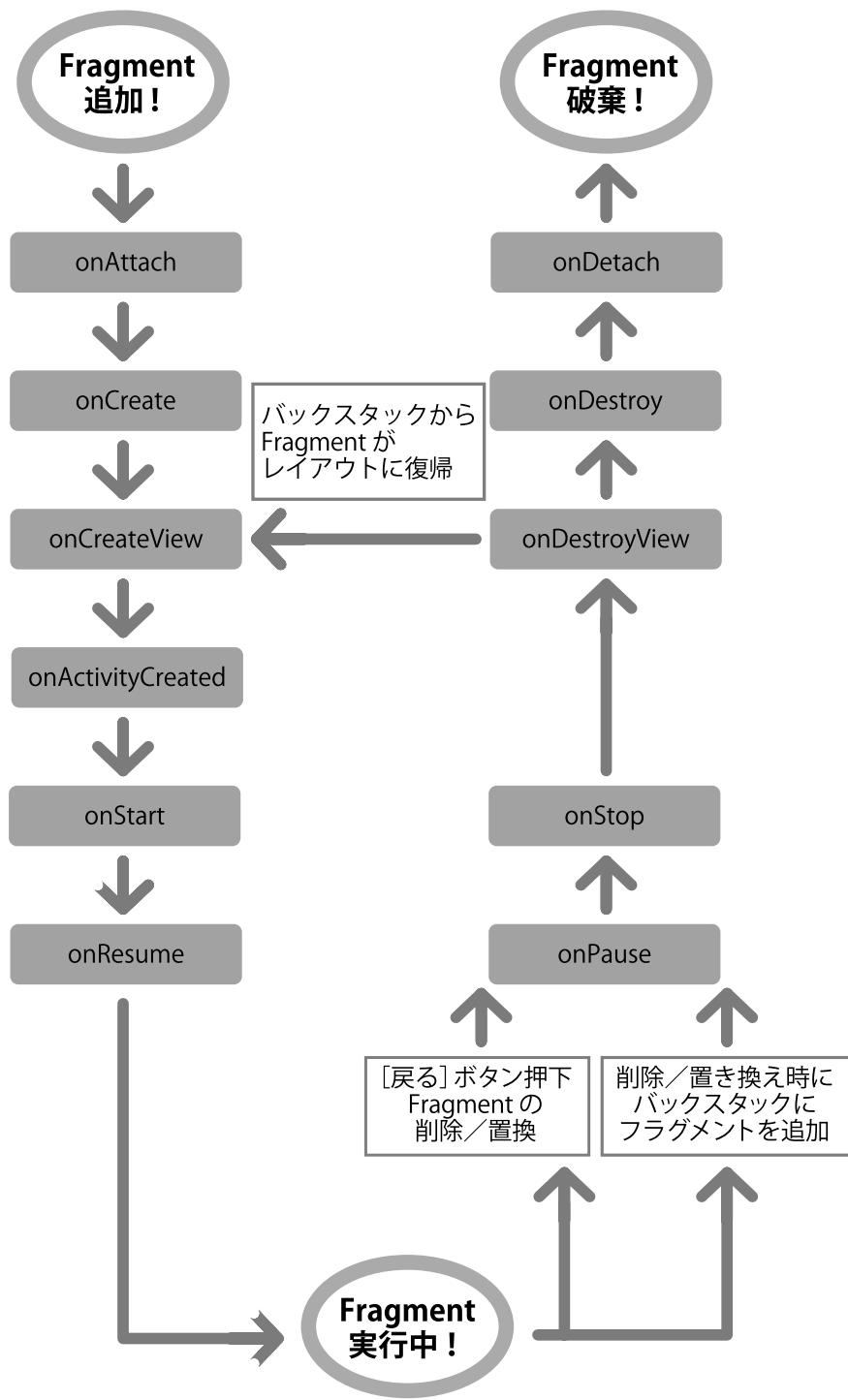


図1:Fragmentのライフサイクル

この図をメソッドに対応させて説明したのが、次の表2になります。

メソッド名	概要
<code>onAttach()</code>	<code>Activity</code> に関連付けされた際に一度だけ呼び出される
<code>onCreate()</code>	<code>Fragment</code> の初期化処理を行う
<code>onCreateView()</code>	<code>Fragment</code> のView階層を生成し戻り値として返す。これが <code>Fragment</code> の表示となる。
<code>onActivityCreated()</code>	親となる <code>Activity</code> の「 <code>onCreate</code> 」の終了時に呼び出される。
<code>onStart()</code>	<code>Activity</code> の「 <code>onStart()</code> 」時に開始される。
<code>onResume()</code>	<code>Activity</code> の「 <code>onResume()</code> 」時に開始される
<code>onPause()</code>	<code>Activity</code> が「 <code>onPause()</code> 」になった場合や、 <code>Fragment</code> が更新されて、操作を受け付けなくなった場合に呼び出される
<code>onStop()</code>	フォアグラウンドでなくなった場合に呼び出される
<code>onDestroyView()</code>	<code>Fragment</code> の内部のViewリソースの整理を行う
<code>onDestroy()</code>	<code>Fragment</code> が破棄されるとき、最後に呼び出される
<code>onDetach()</code>	<code>Activity</code> の関連付けから外されたときに呼び出される

表2: フラグメントのメソッド

`Activity`のライフサイクルとは異なる独自のライフサイクルを持っているとはいっても、`Fragment`は`Activity`上に配置されているため、`Activity`のライフサイクルの影響も受けます。たとえば、`Activity`が一時的に停止されると`Fragment`も同様に一時停止状態となり、`Activity`が破棄されると`Fragment`も破棄されます。

`Activity`と`Fragment`のライフサイクルを対応させると、表3のようになります。`Activity`の特定のタイミングで、対応するメソッドが`Fragment`で呼び出される、という仕組みになっています。

<code>Activity</code> の状態	<code>Fragment</code> で呼び出されるメソッド
Created	<code>onAttach()</code>
	<code>onCreate()</code>
	<code>onCreateView()</code>
	<code>onActivityCreated()</code>
Started	<code>onStart()</code>
Resumed	<code>onResume()</code>
Paused	<code>onPause()</code>
Stopped	<code>onStop()</code>
Destroyed	<code>onDestroyView()</code>
	<code>onDestroy()</code>
	<code>onDetach()</code>

表3:`Activity`と`Fragment`の対応

なお、FragmentはActivityにある「FragmentManager」によって管理されていて、新たにFragmentを追加する場合は、FragmentManagerにFragmentオブジェクトを追加するという方法で行います。これから実際にプログラムを組んでみましょう。



## 7-2-2 自動生成プロジェクトの中の Fragment

Android Development Tools r22.6.1以降、新しくAndroidプロジェクトを作成した際には、Fragmentを使用する形でプロジェクトが生成されます。生成されたばかりの状態を見てるために、新たに「FragmentDummy」という名称のプロジェクトを作ってみましょう。

アプリケーション名:	FragmentDummy
プロジェクト名:	FragmentDummy
パッケージ名:	jp.techinstitute.fragmentdummy
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表4:FragmentDummyプロジェクトの構成

まずは、activity\_main.xmlを見てみます。

### activity\_main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="jp.techinstitute.fragmentdummy.MainActivity"
    tools:ignore="MergeRootFrame" />
```

activity\_main.xmlの中の「FrameLayout」に「container」というidが振られています。次に、生成されたばかりのMainActivity.javaを見てみましょう。Activityが生成されたときに起動されるonCreateはどうなっているでしょうか？

```
MainActivity.java  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
     super.onCreate(savedInstanceState);  
     setContentView(R.layout.activity_main);  
  
     if (savedInstanceState == null) {  
         getSupportFragmentManager().beginTransaction()  
             .add(R.id.container, new  
PlaceholderFragment()).commit();  
     }  
 }
```

4行目の「setContentView(R.layout.activity\_main)」で、activity\_main.xmlを画面レイアウトとして設定しています。

順にソースを見てみると、「Bundle savedInstanceState」は、アプリ終了時の状態を保存しているものです。続くif節では、初回起動時にどうするかという処理をしています。この中の「getSupportFragmentManager()」は、現在のActivityに関連するFragmentManagerを取得するものです。先に述べたように、FragmentはFragmentManagerで管理されます。

ここでの処理をもう少し詳しく見てみましょう。Fragmentでの画面遷移に相当する部分には、「トランザクション」という考え方方が用いられています。トランザクションとは“分けることができない一連の流れ”を指し、その一連の流れが終わったあとに、処理が反映されます。「beginTransaction」で取得したFragmentTransactionのメソッドを呼び出しても、即座にそれらが適用されるわけではありません。その後に「commit」メソッドを実行することで適用されます。

再びソースを見てみましょう。beginTransactionで取得したFragmentTransactionのaddメソッドで、「PlaceHolderFragment」というFragmentを、「R.id.container」というidで定義されたlayout、つまりactivity\_main.xmlの「FrameLayout」に紐付けています。その後にcommitメソッドを実行することで反映しています。

このPlaceHolderFragmentとは、同じくMainActivity.javaの中にあるインナークラスです。

**MainActivity.java**

```
public static class PlaceholderFragment extends Fragment {

    public PlaceholderFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_main, container,
                                         false);
        return rootView;
    }
}
```

PlaceholderFragmentは、「`android.app.Fragment`」を継承したインナーカラスです。`onCreateView()`メソッドの引数に「`LayoutInflater`」オブジェクトが提供されていて、「`inflate()`」メソッドでレイアウトXMLファイルと関連付けを行うことができます。

`LayoutInflater`は、他のxmlリソースのViewを取り扱える仕掛けです。第1引数に紐付けを行うレイアウトXMLファイルのリソースIDを指定します。第3引数の「`attachToRoot`」は、`true`にするか`false`にするかで、戻り値となるルートビュー(View)が変わります。

`true`とした場合は第2引数の`ViewGroup`が、`false`とした場合は第1引数のリソースIDで指定したXMLファイルがルートビューとなります。`inflate`メソッドに、ここでは`R.layout.fragment_main`、つまり`fragment_main.xml`を読み込んで、そのViewを返しています。

`MainActivity.java`から、画面レイアウトとして`activity_main.xml`を読み込み、その中の`FrameLayout`に`fragment_main.xml`をはめ込んでいる、という流れになります。

この形で自動生成ソースコードが作られるようになったのはAndroid Development Tools r22.6.1以降で、ネット上に多くのサンプルコードは、これより前の自動生成ソースコードを元にしているため、`activity_main.xml`に直接オブジェクトを配置しているものが多くあります。ネットの情報を参考にする際は、その点を注意しましょう。

## 縦画面と横画面でのレイアウト変更をFragmentで実現してみる

では、実際にFragmentを使って行きましょう。「タブレットでは左にリストがあって右に詳細のある画面だが、スマートフォンではリスト画面から項目を選択すると詳細画面へ移動するようになる」と実際にやってみたいところですが、スマートフォンとタブレットの両方を持っている方は少ないですよね。そこで、「縦向きと横向きでレイアウトが変わる」というのをやってみましょう。

仕様を以下のようにします。

- ・縦のときはリストが表示されて、選択するとToastに選択内容が表示される
- ・横のときは左にリストが表示されて、選択すると右に選択内容が表示される

アプリケーション名:	FragmentTablet
プロジェクト名:	FragmentTablet
パッケージ名:	jp.techinstitute.fragmentdummy
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表5:FragmentTabletプロジェクトの構成

縦のときと横のときでは画面のレイアウトが変わるので、メイン画面のレイアウトは2種類必要になりそうです。また画面のパーツは、大きく分けるとリストと詳細の2つになります。ということは、Fragmentもリストと詳細の2種類になりますね。

今回、リストにはFragmentのサブクラスである「ListFragment」を使ってみます。ですので、ListFragmentを継承したMyListFragmentクラスを作成しましょう。この部分のソースコードは、後ほど記述します。

まずは縦と横のレイアウトXMLの違いを見てみましょう。fragment要素内のclass属性に「jp.techinstitute.fragmenttablet.MyListFragment」を記述しています。ここにこれから作成するMyListFragmentクラスが入ります。

```
layout/activity_main.xml
<LinearLayout
    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"
    android:orientation="vertical">
    <fragment
        android:id="@+id/list1"
        class="jp.techinstitute.fragmenttablet.MyListFragment"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

画面を横にしたときのレイアウトは、layoutフォルダーが格納されているresフォルダーにlayout-landフォルダーを作り、そこに記述します。ファイル名は同じactivity\_main.xmlです。

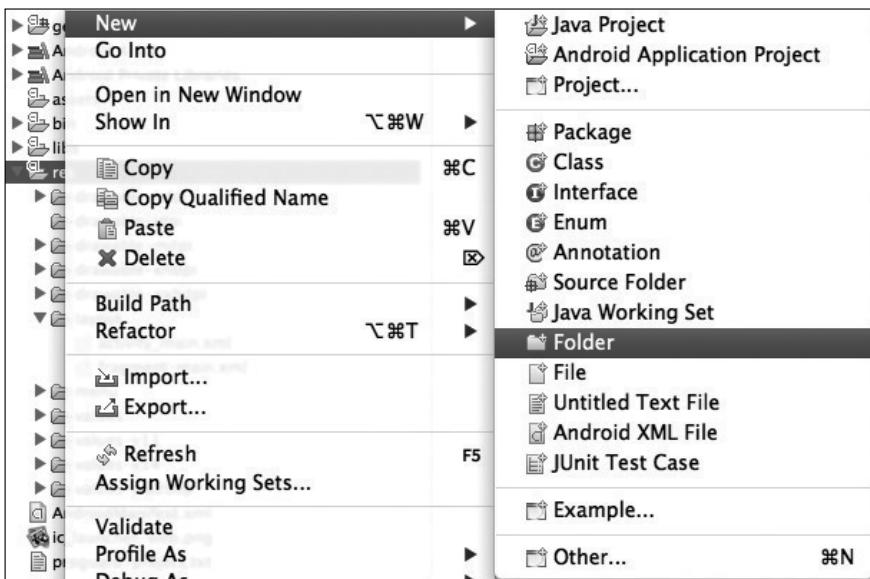


図2:layout-landフォルダーをres配下に作成

## layout-land/activity\_main.xml

```
<LinearLayout
    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"
    android:orientation="horizontal">
    <fragment
        android:id="@+id/list1"
        class="jp.techinstitute.fragmenttablet.MyListFragment"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent"/>
    <View
        android:layout_width="5dp"
        android:layout_height="match_parent"
        android:background="#0000aa"/>
    <FrameLayout
        android:id="@+id/detail"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" >
    </FrameLayout>
</LinearLayout>
```

タブレット端末も持っている方は、layout-landではなくlayout-largeにしてみましょう。これで、スマートフォンとタブレットでレイアウトが異なるのを実現できます。このlayout-land配下のactivity\_main.xmlと、layout配下のactivity\_main.xmlとの違いは、詳細を表示する右側のFrameLayoutがあるかどうかです。間にあるViewは枠線代わりです。

そしてMainActivity.javaですが、今回はシンプルにするために、以下のようにActivityクラスを継承するものにします。

```

MainActivity.java

package jp.techinstitute.fragmenttablet;

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

public class MainActivity extends Activity {

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

    }

}

```

次に、画面右の詳細部分になる「DetailFragment.java」を作成します。Fragmentを継承したDetailFragmentクラスです。クラスを新規追加し、「Super class」の欄で「Browse」をクリックし、以下のようにFragmentを選択してください（図3）。

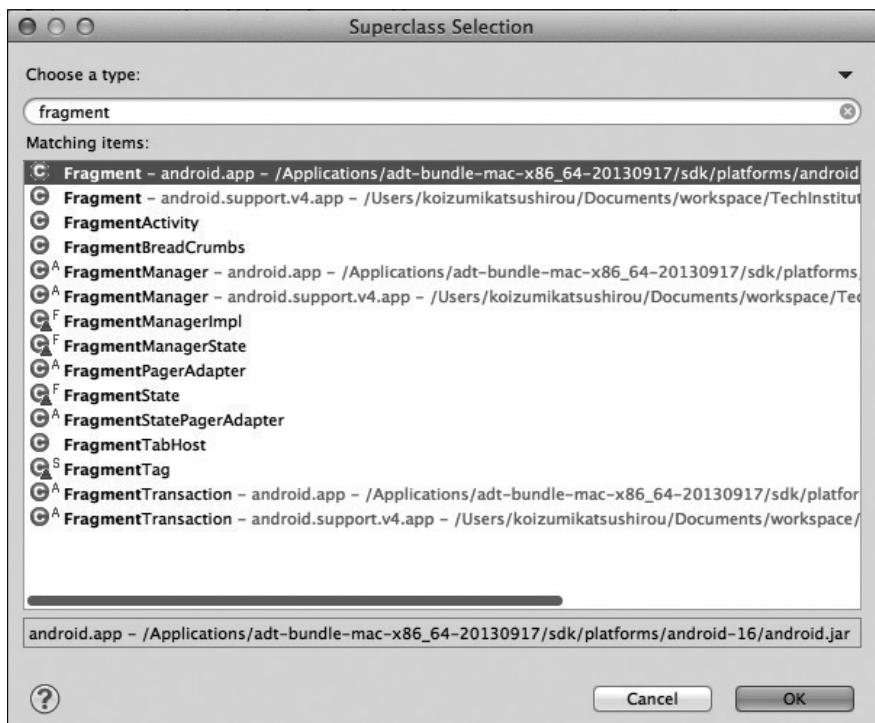


図3:スーパークラスにFragmentを選択

以下の図4のようになればOKですので、FinishをクリックしてDetailFragmentクラスを作ってください。

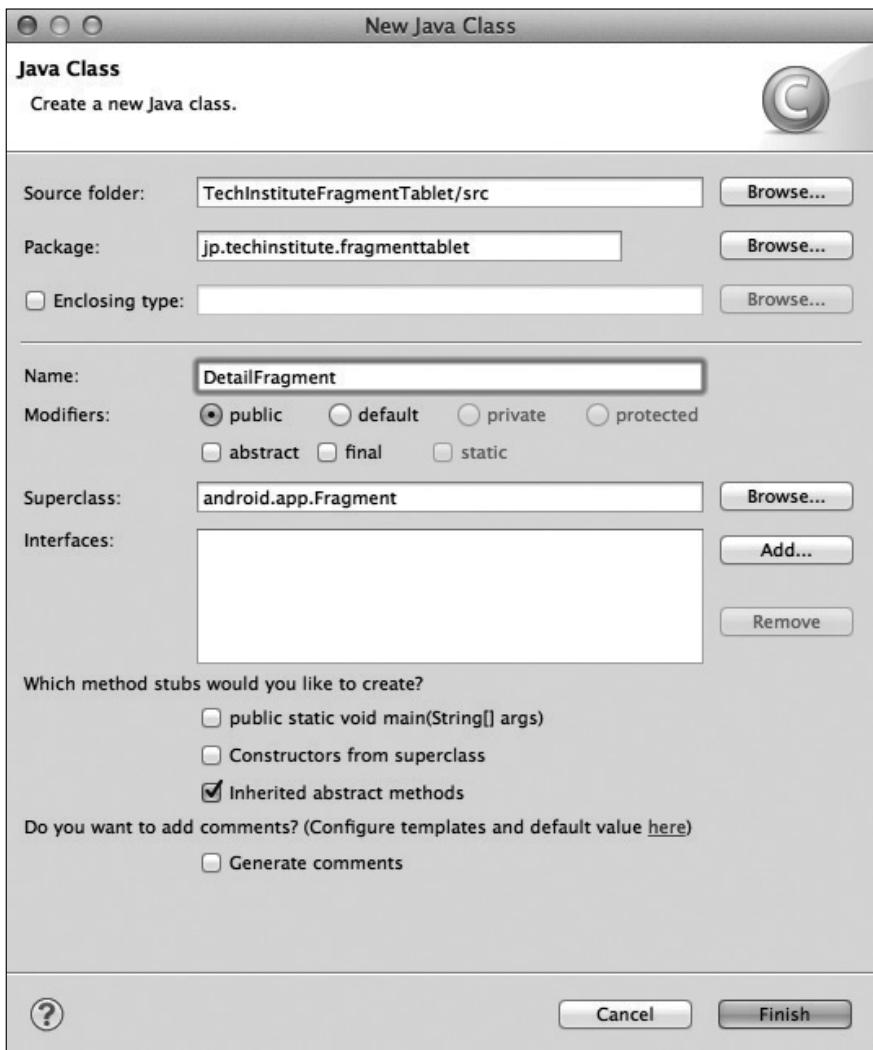


図4:DetailFragment作成

### DetailFragment.java

```
package jp.techinstitute.fragmenttablet;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class DetailFragment extends Fragment {

    // フラグメントのビュー生成時に呼ばれる
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle bundle) {
        if (container==null){
            return null;
        }
        TextView rootView=new TextView(getActivity());
        rootView.setTextSize(18);
        rootView.setText(getArguments().getString("item") + "が選択されました");
        return rootView;
    }
}
```

Fragmentを自分で作るには、このDetailFragmentのように、Fragmentを継承したクラスを作成します。Fragmentの中身はどう作るのかですが、**表3**にあるように、onCreateViewメソッドは戻り値としてFragmentのUI(画面)を返します。つまり、Fragmentの中身はonCreateViewメソッドの戻り値となるように、onCreateViewを上書きしてメソッドを作成する必要があります。今回は単純にTextViewだけにしてみました。

「 getArguments」メソッドの戻り値はBundleクラスのオブジェクトで、これがFragmentに対して渡されたパラメーターとなっています。ここでは「item」というキーでパラメーターとして設定された文字列を取り出し、TextViewに設定しています。続いてリスト部分の作成です。ListFragmentを継承した「MyListFragment」クラスを作ってみましょう。クラスを新規追加し、Super classの欄でBrowseをクリックして、以下のようにListFragmentを選択してください。

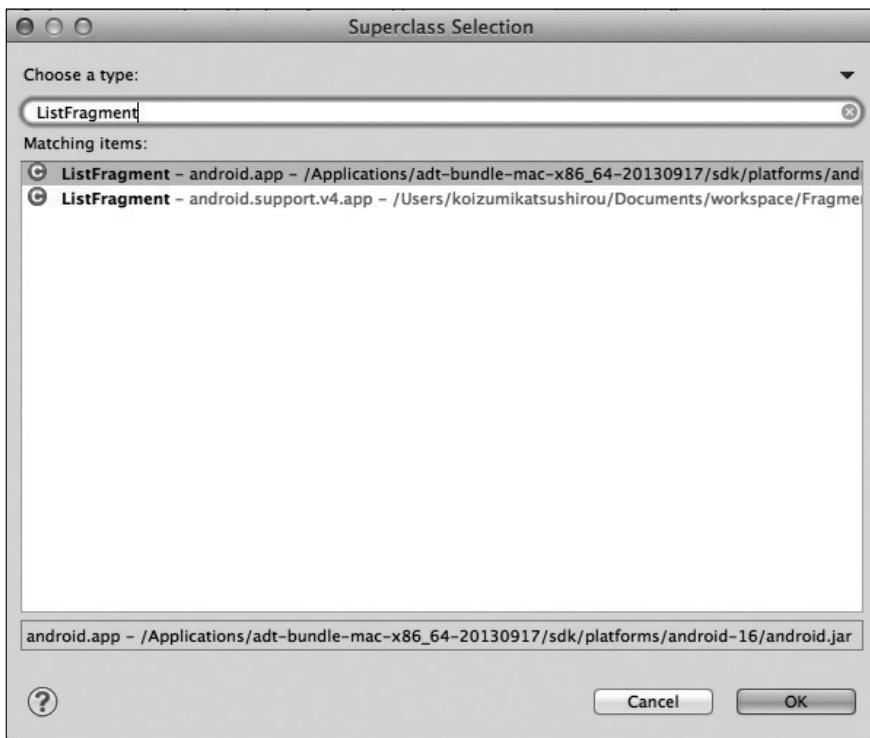


図5:スーパークラスにListFragmentを選択

以下の図6のようになればOKですので、FinishをクリックしてMyListFragmentクラスを作ってください。

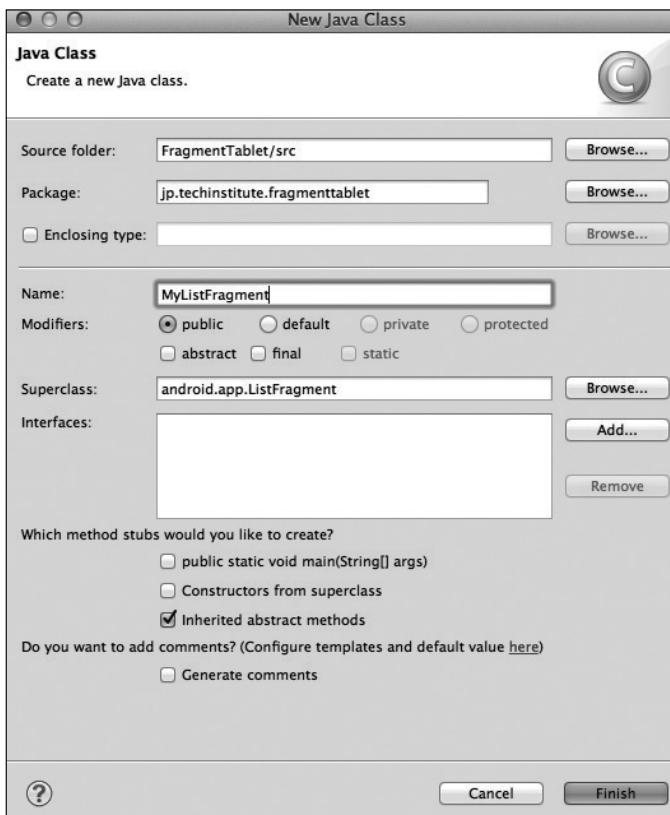


図6:MyListFragment  
作成

### MyListFragment.java

```
package jp.techinstitute.fragmenttablet;

import android.app.FragmentTransaction;
import android.app.ListFragment;
import android.os.Bundle;
import android.app.FragmentManager;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class MyListFragment extends ListFragment {

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        String[] Data = {
            "りんご",
            "みかん",
            "梨",
            "ドリアン"
        };

        ArrayAdapter<String> adapter =
            new ArrayAdapter<String>(
                getActivity(),
                android.R.layout.simple_list_item_1,
                Data);

        setListAdapter(adapter);
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        super.onListItemClick(l, v, position, id);

        // MainActivityのオブジェクトを取得
        MainActivity activity = ( MainActivity ) getActivity();

        // 選択されたアイテム
        String itemStr = l.getItemAtPosition(position).toString();

        // R.id.detailがあるか否かで場合分け
        boolean hasDetail = false;
        View detail = activity.findViewById(R.id.detail);
        if( detail == null ){
            hasDetail = false;
        } else {
            hasDetail = true;
        }

        // R.id.detailがある場合
    }
}
```

```
if( hasDetail ){
    // Activity の FragmentManager から FragmentTransaction を取得
    FragmentManager fm = activity.getFragmentManager();
    FragmentTransaction t = fm.beginTransaction();

    // DetailFragment に渡す
    DetailFragment fragment = new DetailFragment();
    Bundle bundle = new Bundle();
    bundle.putString("item", itemStr);
    fragment.setArguments(bundle);

    // レイアウトに Fragment をセット
    t.replace(R.id.detail, fragment);
    t.commit();
} else {
    // R.id.detailがない場合は Toast で呼び出し
    Toast.makeText(getActivity(), itemStr, Toast.LENGTH_SHORT).show();
}
}
```

ListFragmentは、前節で学習したListViewをもつFragmentで、Fragmentクラスのサブクラスです。ListFragmentを継承した独自クラスを作成し、メソッドを上書きして使います。今回上書きしているメソッドは2つ。Activityが作られたときに呼び出される「onActivityCreated」と、リストのアイテムがクリックされたときに呼び出される「onListItemClick」です。

onActivityCreatedには、リストの中身をつくる処理が書かれています。ListViewと同様に ArrayAdapterに配列をセットしますが、ArrayAdapterはListFragmentのsetListAdapterにセットします。

ListFragmentのアイテムがクリックされたときに呼び出されるのが、onListItemClickメソッドです。引数は、前節のOnItemClickListenerのonItemClickと同様です。この中では、縦画面か横画面かを「R.id.detail」があるかどうかで判別しています。

R.id.detailがある場合は、画面右のFrameLayoutにFragmentをセットします。7-2-2で学んだように、FragmentはActivityの持つFragmentManagerに管理され、変更はFragmentTransactionに対して行います。ここでは、独自に作成するDetailFragmentをセットします。

Fragmentに値を渡すには、Bundleオブジェクトを「setArguments」でセットします。Bundleオブジェクトには、各プリミティブ型とその配列、および文字列とその配列がセット可能です。ここでは、「item」をキーにリストから選択された値をセットしています。

Fragmentに渡す値のセットが終わったら、FragmentTransactionにDetail Fragmentをセットします。ここでは「replace」メソッドでセットしています。replaceメ

ソッドは、すでにFragmentがセットされいたら削除し、新しくFragmentを追加するメソッドです。

FragmentTransactionクラスの主なメソッドを、以下の表6にまとめたのでご覧ください。

分類	メソッド
add/remove系	<pre>add(int containerViewId, Fragment fragment, String tag)</pre> <ul style="list-style-type: none"> <li>Activity に fragment を追加する。</li> <li>fragment がすでに Activity に追加されている場合にも起こらない</li> <li>fragment を containerViewId の ViewGroup に追加</li> <li>addメソッド後に呼び出されるライフサイクル           <ul style="list-style-type: none"> <li>onAttach()</li> <li>→ onCreate()</li> <li>→ onCreateView()</li> <li>→ onActivityCreated()</li> <li>→ onStart()</li> <li>→ onResume()</li> </ul> </li> </ul>
	<pre>remove(Fragment fragment)</pre> <ul style="list-style-type: none"> <li>Activity から fragment を削除する。</li> <li>fragment がすでに Activity から削除されている場合にも起こらない。</li> <li>fragment の View が container に追加されている場合、その View は container から削除される。</li> <li>removeメソッド後に呼び出されるライフサイクル           <ul style="list-style-type: none"> <li>onPause()</li> <li>→ onStop()</li> <li>→ onDestoryView()</li> <li>→ onDestory()</li> <li>→ onDetach()</li> </ul> </li> </ul>
show/hide系	<pre>replace(int containerViewId, Fragment fragment, String tag)</pre> <ul style="list-style-type: none"> <li>container にすでに追加されている Fragment を置き換える。</li> <li>処理としては、remove(Fragment) を呼び、それから add(containerId, fragment, tag) を呼ぶのと同じ</li> <li>replaceメソッド後に呼び出されるライフサイクル</li> <li>置き換えられる fragment : removeに同じ</li> <li>置き換える fragment : addに同じ</li> </ul>
	<pre>show(Fragment fragment)</pre> <ul style="list-style-type: none"> <li>hidden 状態になっている fragment (の View)を表示する</li> <li>fragment のライフサイクル状態は変わらない</li> </ul>
attach/detach系	<pre>hide(Fragment fragment)</pre> <ul style="list-style-type: none"> <li>存在している fragment を隠す(hidden 状態にする)</li> <li>fragment のライフサイクル状態は変わらない</li> </ul>
	<pre>attach(Fragment fragment)</pre> <ul style="list-style-type: none"> <li>detach() によって画面から取り外されている fragment を再度取り付ける</li> <li>onCreateView() は呼ばれるが onAttach(), onCreate() は呼ばれない</li> <li>attachメソッド後に呼び出されるライフサイクル           <ul style="list-style-type: none"> <li>onCreateView()</li> <li>→ onActivityCreated()</li> <li>→ onStart()</li> <li>→ onResume()</li> </ul> </li> </ul>
	<pre>detach(Fragment fragment)</pre> <ul style="list-style-type: none"> <li>画面から fragment を取り外す</li> <li>onDestoryView() は呼ばれるが onDestory(), onDetach() は呼ばれない</li> <li>detachメソッド後に呼び出されるライフサイクル           <ul style="list-style-type: none"> <li>onPause()</li> <li>→ onStop()</li> <li>→ onDestoryView()</li> </ul> </li> </ul>

表6:FragmentTransactionの主なメソッド

では実行してみましょう。画面が縦のときは画面の中にはリストのみ。リストから選択すると、選択内容がToastで表示されます(図7)。画面が横のときは画面左にリスト、右に表示欄になります。リストから選択すると、選択内容が画面右にテキストで表示されます(図8)。



図7:縦画面のとき。選択内容はToastで表示



図8:横画面のとき。選択内容は画面右にテキストで表示



## 7-2-3 アラートダイアログを作ってみよう

Androidアプリで、ひんぱんに見かけるアラートダイアログ。図9のような、OKボタンとCancelボタンが表示される画面を皆さんも見たことがあるでしょう。このアラートダイアログも、Fragmentで作ることができます。



図9:キャプション:アラートダイアログ

では早速プロジェクトの作成から始めてみましょう。

アプリケーション名:	AlertFragment
プロジェクト名:	AlertFragment
パッケージ名:	jp.techinstitute.alertfragment
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表7:AlertFragmentプロジェクトの構成

まず、fragment\_main.xmlにアラートダイアログを呼び出すためのボタンを追加しましょう。グラフィカルレイアウトからButtonを選択し、画面にドラッグ＆ドロップしてください。

#### fragment\_main.xml

```
<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"
    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="jp.techinstitute.alertfragment.MainActivity$PlaceholderFragment" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/textView1"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="48dp"
        android:text="Button" />

</RelativeLayout>
```

続いて、MainActivity.javaにアラートダイアログの記述をします。

**MainActivity.java**

```
package jp.techinstitute.alertfragment;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.app.Fragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

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

        if (savedInstanceState == null) {
            getFragmentManager().beginTransaction()
                .add(R.id.container, new PlaceholderFragment()).commit();
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    /**
     * A placeholder fragment containing a simple view.

```

```

/*
public static class PlaceholderFragment extends Fragment {

    public PlaceholderFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_main, container,
                                         false);

        Button button = (Button) rootView.findViewById(R.id.button1);
        button.setOnClickListener(
            new OnClickListener() {

                @Override
                public void onClick(View v) {
                    MyDialogFragment dialog = new
MyDialogFragment();
                    dialog.show(getFragmentManager(), "dialog");

                }
            }
        );
    }

    return rootView;
}
}

public void doPositiveClick() {
    Toast.makeText(this, "OK ボタン", Toast.LENGTH_SHORT).show();
}

public void doNegativeClick() {
    Toast.makeText(this, "Cancel ボタン", Toast.LENGTH_SHORT).show();
}

public static class MyDialogFragment extends DialogFragment {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("ここにタイトルが入ります");
        builder.setMessage("ここにメッセージが入ります");
        builder.setPositiveButton("OK",
new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                MainActivity activity = (MainActivity) getActivity();
                activity.doPositiveClick();
            }
        });
    }
}

```

```
        }

    });

    builder.setNegativeButton("Cancel",

        new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                MainActivity activity = (MainActivity) getActivity();
                activity.doPositiveClick();
            }
        });
}

return builder.create();

}

}
```

ボタンをクリックして、アラートダイアログを表示する流れから見て行きましょう。on Clickの中でMyDialogFragmentのインスタンスdialogを作成し、showメソッドでダイアログを表示しています。

アラートダイアログは、DialogFragmentを継承したMyDialogFragmentを作成し、その中に具体的にどういうダイアログにするかを記述します。

AlertDialog.Builderクラスのインスタンスbuilderにタイトルや表示するメッセージを記述して行きます。ここで使用しているAlertDialog.Builderクラスのメソッドは以下の表8を参照してください。

メソッド名	引数	説明
setTitle	setTitle(CharSequence title)	表示されるダイアログのタイトルを設定。文字列、リソースIDを引数に指定可能。
	setTitle(int titleId)	
setMessage	setMessage(CharSequence message)	表示されるダイアログのメッセージを設定。文字列、リソースIDを引数に指定可能。
	setMessage(int messageId)	
setPositiveButton	setPositiveButton (CharSequence text, DialogInterface.OnClickListener listener)	OKボタンのタイトルとボタン押下時に呼び出されるOnClickListenerを設定。第1引数には文字列、リソースIDを引数に指定可能。
	setPositiveButton (int textId, DialogInterface.OnClickListener listener)	
setNegativeButton	setNegativeButton (CharSequence text, DialogInterface.OnClickListener listener)	キャンセルボタンのタイトルとボタン押下時に呼び出されるOnClickListenerを設定。第1引数には文字列、リソースIDを引数に指定可能。
	setNegativeButton (int textId, DialogInterface.OnClickListener listener)	
show	show()	アラートダイアログを表示する。

表8:AlertDialog.Builderの主なメソッド

次に「setPositiveButton」メソッド、「setNegativeButton」メソッドを見てみましょう。ここには、ボタンに表示される文字と、ボタンを押したときに実行される処理が記述されます。

ここでは、ボタンタイトルはそれぞれ「OK」と「Cancel」とし、クリック時にはDialog Interface.OnClickListenerを実装したクラスのonClickが呼び出されるようになっています。この中では、MainActivityのdoPositiveClickとdoNegativeClickを呼び出しています。doPositiveClickとdoNegativeClickを呼び出せるようにするために、getActivityで取得したActivityをMainActivityにキャストする必要がある点にご注意ください。

さて、実行してみましょう。ボタンを押すと図10のようにアラートダイアログが表示されればOKです。



図10:キャプション:アラートダイアログ



## 7-2-4 タブで画面切り替え

Androidには、タブで画面の表示を切り替える機能があります。タブは直感的で分かりやすいため、多くのアプリで利用されています。皆さんも図11のようにタブを切り替える画面を見たことがあるのではないですか？



図11:タブ画面

本節では、「ActionBar」（ロゴやボタンなどを配置できるツールバー）にFragmentを切り替えるタブコントロールを実装する方法で、タブ画面を作って行きます。タブには、タイトル文字列やアイコン画像をセットすることが可能で、画面をより直感的なものにできます。



### タブはActionBarで作るのがおすすめ

以前は、タブ形式のナビゲーションは「TabHost」と「TabWidget」を用いて実装されました。しかしながら、ActionBarが導入されてからは、こちらのタブナビゲーションを利用することが推奨されています。

大きな利点として、タブ表示に十分な幅がない場合自動的にタブが非表示になり、代わりにアクションアイテム（ボタン）として表示されるようになります。つまり、ActionBarのタブナビゲーションを利用することで、より多くのスクリーンサイズに対応可能となるわけです。

では、プロジェクトを作成して行きましょう。

アプリケーション名:	TabFragment
プロジェクト名:	TabFragment
パッケージ名:	jp.techinstitute.alertfragment
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表9:TabFragmentプロジェクトの構成

アクションバータブのイベントを制御するにはActionBar.TabListenerを実装したクラスを作成する必要があります。上書きすべき各メソッドは、表10のようになっています。ここにタブが選択／非選択時になったときの処理を書きます。

メソッド	概要
onTabSelected(Tab tab, FragmentTransaction ft)	タブが選択された時に呼ばれる
onTabUnselected(Tab tab, FragmentTransaction ft)	タブが非選択になった時に呼ばれる
onTabReselected(Tab tab, FragmentTransaction ft)	同じタブが再度選択された時に呼ばれる

表10:ActionBar.TabListenerのメソッド

各メソッドには、第1引数としてイベントを受け取ったActionBar.Tabインスタンス、第2引数としてFragmentTransactionインスタンスとなっています。

今回はActionBar.TabListenerを実装したクラスMyTabListenerを作成します。

## MyTabListener.java

```
package jp.techinstitute.tabfragment;

import android.app.ActionBar.Tab;
import android.app.Fragment;
import android.app.ActionBar;
import android.app.FragmentTransaction;

public class MyTabListener implements ActionBar.TabListener {
    private Fragment mFragment;

    // 新規タブを作成する際にragmentインスタンスと一緒に渡す
    public MyTabListener(Fragment fragment) {
        mFragment = fragment;
    }

    @Override
    public void onTabSelected(Tab tab, FragmentTransaction ft) {
        // タブが選択された時の処理
        // Fragmentを追加する
        ft.add(R.id.fragment_content, mFragment, null);
    }

    @Override
    public void onTabUnselected(Tab tab, FragmentTransaction ft) {
        // タブが切り替えられて非選択になった時の処理
        // Fragmentを削除する
        ft.remove(mFragment);
    }

    @Override
    public void onTabReselected(Tab tab, FragmentTransaction ft) {
        // 同じタブを再度タップされた時の処理
        // ここでは何も記述しない
    }
}
```

タブの切り替え時にFragmentの追加・削除を行えるようにする必要があるので、MyTabListenerのコンストラクタにてタブのFragmentを渡しておき、表10のメソッドを上書きしてfragmentの追加や削除を行うよう実装しました。

肝心のタブの中身のレイアウトも作りましょう。UIが切り替わっていることがわかるようするために、タブ1にはボタンを1つ、タブ2にはボタンを2つ置いてみました。それぞれ、fragment\_tab1.xml、fragment\_tab2.xmlです。

fragment\_tab1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Tab1 が選択されています "/>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="50dp"
        android:text="Button" />

</RelativeLayout>
```

## fragment\_tab2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Tab2 が選択されています "/>

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="50dp"
        android:text="Button" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button1"
        android:layout_marginTop="50dp"
        android:text="Button" />

</RelativeLayout>
```

続いてActivityです。MainActivityを以下のように実装してみましょう。

### MainActivity.java

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    // ActionBar を取得してモードをタブモードへセット  
    final ActionBar actionBar = getActionBar();  
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);  
    // タイトルを非表示  
    actionBarsetDisplayShowTitleEnabled(false);  
  
    // Fragment を生成して Tab へセット  
    Fragment fragment1 = new TabFragment1();  
    // 新しく生成した Tab インスタンスにタイトル文字列、アイコン、リスナーをセット  
    ActionBar.Tab tab1 = actionBar.newTab();  
    tab1.setText("Tab1");  
    tab1.setIcon(R.drawable.ic_launcher);  
    tab1.setTabListener(new MyTabListener(fragment1));  
    actionBar.addTab(tab1);  
  
    // Tab2 も同様に作成  
    Fragment fragment2 = new TabFragment2();  
    ActionBar.Tab tab2 = actionBar.newTab();  
    tab2.setText("Tab2");  
    tab2.setIcon(R.drawable.ic_launcher);  
    tab2.setTabListener(new MyTabListener(fragment2));  
    actionBar.addTab(tab2);  
}
```

タブ表示にするためのステップを見て行きましょう。ActivityクラスにてgetActionBar()を使って取得したActionBarインスタンスに対して、「setNavigationMode(ActionBar.NAVIGATION\_MODE\_TABS)」を設定することで、ActionBarをタブ表示モードへ切り替えることができます。

ActionBarに対してタブを追加するには、「actionBar.newTab()」で取得したインスタンスをaddTab()にて追加します。この時、タブに対してタイトル及びアイコン画像をセットすることができます。

また、タブの切り替え制御を行うために、上記で作成したActionBar.TabListenerをセットします。これらのActionBar.Tabのメソッドには表11を参照してください。

メソッド名	引数	説明
setTitle	setTitle(CharSequence title)	表示されるダイアログのタイトルを設定。文字列、リソースIDを引数に指定可能。
	setTitle(int titleId)	
setMessage	setMessage(CharSequence message)	表示されるダイアログのメッセージを設定。文字列、リソースIDを引数に指定可能。
	setMessage(int messageId)	
setPositiveButton	setPositiveButton (CharSequence text, DialogInterface.OnClickListener listener)	OKボタンのタイトルとボタン押下時に呼び出されるOnClickListenerを設定。第1引数には文字列、リソースIDを引数に指定可能。
	setPositiveButton (int textId, DialogInterface.OnClickListener listener)	
setNegativeButton	setNegativeButton (CharSequence text, DialogInterface.OnClickListener listener)	キャンセルボタンのタイトルとボタン押下時に呼び出されるOnClickListenerを設定。第1引数には文字列、リソースIDを引数に指定可能。
	setNegativeButton (int textId, DialogInterface.OnClickListener listener)	
show	show()	アラートダイアログを表示する。

表11:ActionBar.Tabの主なメソッド

これでタブを実装することができました。実行してみましょう。「TAB1」と「TAB2」の、2つのタブが表示されるようになりました。(図12)

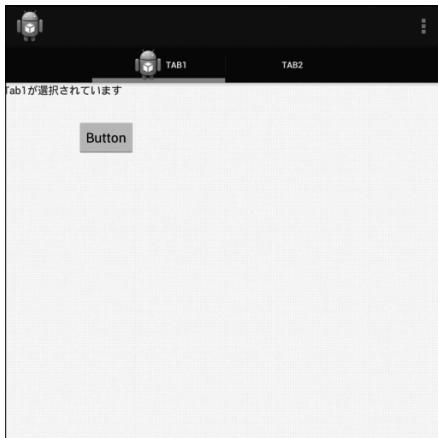


図12:実行結果

## まとめ

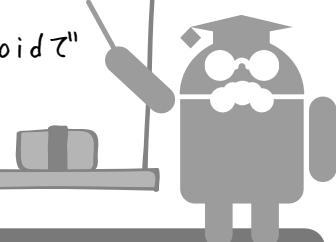
今回の講義では、Fragmentについて扱いました。Android Development Tools r22.6.1以降は、自動生成されるコードがFragmentを使用する前提となっています。今後もFragmentを用いて画面を作るという方向性は変わらないと思われる所以、ぜひこの機会にFragmentに慣れておいてください。

- ・自動生成されるコードがどういう意味を持つのか
- ・画面の解像度や比率によってUIを変更するにはどうするのか
- ・画面をどうFragmentに分けるのか
- ・Activityとはどう対応しているのか
- ・Fragmentに値を渡すときはどうすれば良いのか
- ・FragmentのサブクラスとしてListFragmentとDialogFragmentについて今回は学び、またタブの実現方法、ActionBarを用いた方法を体験

# 7-3 AndroidのAPIについて学ぶ(3)

著: 小泉勝志郎

7-2では、AndroidのAPIの中からFragmentについて学びました。画面構成の幅が、大きく広がったと思います。ここでは、写真の撮影を通じてさまざまなAPIを使うことで、Androidで実現できることが増えることを体感していただきます。



## 7-3-1 各種のビューの使い方

この節では、以下について学んでいきます。

- ・カメラでの写真撮影との撮影データの処理
- ・SurfaceView

Androidスマートフォンに付いているカメラ。スマートフォンの用途としては、メールやSNSの次に使うものになっているのではないでしょうか。

このカメラ機能も、APIを使うことで自分のアプリからも利用できます。まずは写真を撮って、その画像を表示するプログラムを作ってみましょう。特に準備のいらない暗黙的Intentと、その戻り画像を利用したものを作成します。

アプリケーション名:	CameraIntent
プロジェクト名:	CameraIntent
パッケージ名:	jp.techinstitute.cameraintent
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表1: CameraIntentプロジェクトの構成

まずは画面の作成です。今回は簡単に作成するためにFragmentを利用しないで画面を作成して行きます。activity\_main.xmlに、図1のようにカメラを起動す

るためのButtonと、撮影した画像を表示するImageViewを配置します。

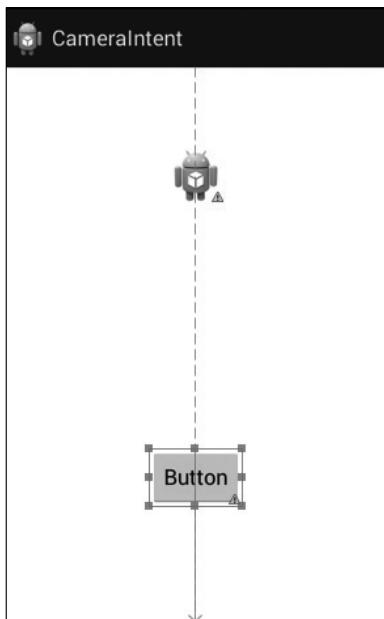


図1:ButtonとImageViewを配置

#### activity\_main.xml

```
<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"
    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="jp.techinstitute.cameralintent.MainActivity$PlaceholderFragment" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp"
        android:src="@drawable/ic_launcher" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="80dp"
        android:text="Button" />

</RelativeLayout>
```

ここでは、暗黙的Intentを利用してカメラを起動してみます。カメラの呼び出しは以下のように「MediaStore.ACTION\_IMAGE\_CAPTURE」を指定して行います。

#### 暗黙 Intent を利用してカメラを起動

```
// カメラでの撮影画面を暗黙 Intent で呼び出し  
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
```

この処理を、ボタンを押したときに呼び出すようにします。そして写真を撮った後、撮影画像をImageViewで表示するように作っていくのですが、撮影した結果として、小さいサイズのBitmapオブジェクト(撮影した画像データ)しか返ってきません。

結果を受け取るために、「startActivityForResult(intent, requestCode)」を使います。第2引数のrequestCodeは、カメラ画面が終わった後のonActivityResultの第1引数に渡される値です。複数のActivity呼び出しがあったときに、識別できるようにするための数値で、ここでは0を渡しています

#### MainActivity.java

```
package jp.techinstitute.cameraintent;  
  
public class MainActivity extends Activity {  
    Button button1;  
    ImageView imageView1;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        button1 = (Button) findViewById(R.id.button1);  
        imageView1 = (ImageView) findViewById(R.id.imageView1);  
  
        button1.setOnClickListener(new View.OnClickListener() {  
  
            @Override  
            public void onClick(View v) {  
  
                // カメラでの撮影画面を暗黙 Intent で呼び出し  
                Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
  
                // 結果を受け取るために startActivityForResult(intent, requestCode) を使  
                // う  
                startActivityForResult(intent, 0);  
            }  
        });  
    }  
}
```

続いて、カメラ画面から戻ってきたときの処理であるonActivityResultを記述しましょう。「右クリック」→「Source」→「Override」を選択すると、図2のような画面になります。ここでonActivityResultを選択してください。

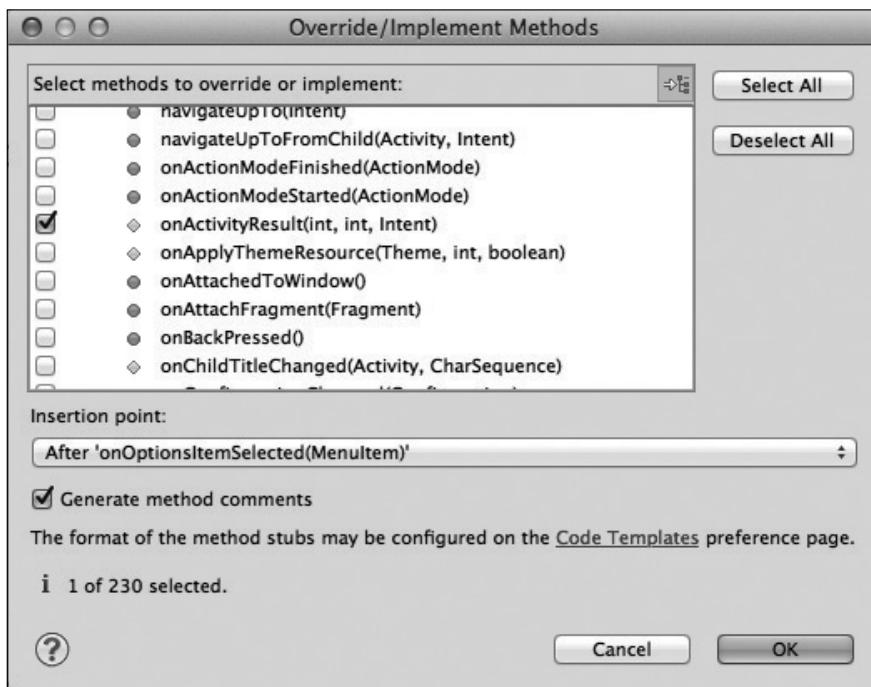


図2:onActivityResultを選択

onActivityResultで、撮影結果のビットマップデータを受け取るために、キーが「data」のExtraを受け取ります。

### MainActivity.java

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if( requestCode == 0 && resultCode == Activity.RESULT_OK ){
        Bitmap image = (Bitmap) data.getExtras().get("data");
        imageView1.setImageBitmap(image);
    }
}
```

撮影結果をR.id.imageView1に反映しています。では、ここで実行してみましょう。図3のような画面が表示されます。

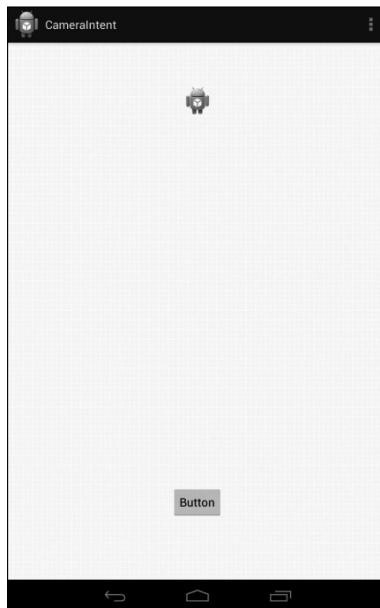


図3:実行結果の初期表示

そしてボタンを押してみると、図4のようにカメラの撮影画面が立ち上がります。



図4:撮影画面が起動

撮影が終わると、図5のように撮影した画像が、画面にはめ込まれます。

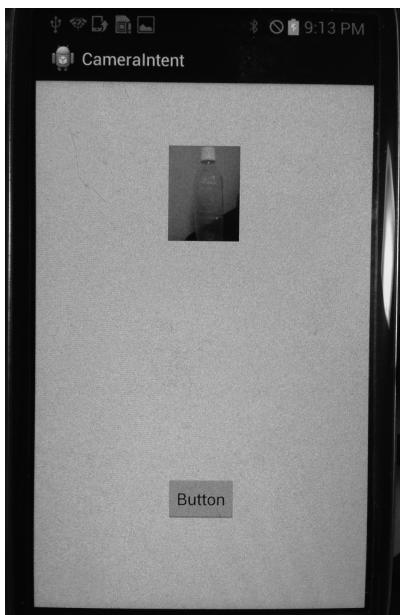


図5:撮影結果の画面

意外と簡単にできたのではないか? ただし、このソースコードで取得できる画像データは小さいサイズのもので、大きな解像度のものは得られません。解決のためには一工夫が必要になりますので、新しくプロジェクトを作成して試してみましょう。

アプリケーション名:	CameraEx
プロジェクト名:	CameraEx
パッケージ名:	jp.techinstitute.cameraex
最小必須SDK:	API16:Anndroid4.1(Jelly Bean)
ターゲットSDK:	API16:Anndroid4.1(Jelly Bean)
次でコンパイル:	API16:Anndroid4.1(Jelly Bean)
アクティビティ名:	MainActivity
レイアウト名:	activity_main
フラグメントレイアウト名:	fragment_main
ナビゲーションタイプ:	なし
テーマ:	Holo Light with Dark Action Bar
アクティビティのタイプ:	Blank Activity with Fragment

表2:CameraExプロジェクトの構成

標準のカメラアプリでは、intentのExtraDataに「MediaStore.EXTRA\_OUTPUT」を指定したかどうかによって、画像の取得の仕方が変わります。大きい画像を取得したい場合は、カメラアプリを呼び出すIntentのパラメーターMediaStore.EXTRA\_OUTPUTにUriオブジェクトを指定します。

```
intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
```

以下、最初に作ったCameraIntentプロジェクトとの相違点を中心に見て行きましょう。まずは下準備として、activity\_main.xmlでアプリの動作を端末の横向き

に変更します。今回はImageViewのサイズをあらかじめ固定しておきました。

### activity\_main.xml

```
<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"
    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="jp.techinstitute.cameraintent.MainActivity$PlaceholderFragment" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="50dp"
        android:src="@drawable/ic_launcher" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="80dp"
        android:text="Button" />

</RelativeLayout>
```

実は、Androidでの撮影アプリを作成する際の「カメラの向き」は、機種依存が多く発生する鬼門です。今回は、画像を取得する手法にフォーカスを当てる意図で、アプリの向きを端末横向きに固定しました。

次に、以下のように MediaStore.EXTRA\_OUTPUTにUriオブジェクトを指定します。OnActivityResultから扱えるようにするために、ここではフィールドにしました。Uri オブジェクトは、コンテンツプロバイダーにあらかじめ写真情報を格納する際に確保しておきます。格納メソッドであるinsertの戻り値が、このUriオブジェクトとなっています。コンテンツプロバイダーについては、後述のコラムをご覧ください。

**MainActivity.java**

```
public static final int REQUEST_CAMERA = 100;
Uri imageUri;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button button1 = (Button) findViewById(R.id.button1);
    button1.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {

            // カメラでの撮影画面を暗黙 Intent で呼び出し
            Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            intent.setAction(MediaStore.ACTION_IMAGE_CAPTURE);
            intent.addCategory(Intent.CATEGORY_DEFAULT);
            imageUri = insertPhotoUri();

            // URI を MediaStore.EXTRA_OUTPUT にセット
            intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);

            // 結果を受け取るために startActivityForResult(intent, requestCode) を使う
            startActivityForResult(intent, REQUEST_CAMERA);
        }
    });
}
```

撮影後は、MediaStore.EXTRA\_OUTPUTで指定したimageUriに画像データが書き込まれているので、表示するにはimageUriを参照するだけです。生成したUriをOnActivityResultにてImageViewへ渡します。ただし、これは撮影された解像度のままの画像データなので、撮影サイズによっては画像サイズが非常に大きい場合があります。今回はImageViewのサイズをあらかじめ固定しておきました。

**MainActivity.java**

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    ImageView imageView1 = (ImageView) findViewById(R.id.imageView1);
    if( requestCode == REQUEST_CAMERA && resultCode == Activity.RESULT_OK ){
        imageView1.setImageURI(imageUri);
    }
}
```

肝心のURI生成処理は、URI生成のinsertPhotoUriメソッドとディレクトリ生成処理のbuildPhotoDirメソッドの、二段階の構成です。

まずはbuildPhotoDirメソッドで、SDカード内の画像保存用のディレクトリを取得します。パッケージ名のフォルダーがあればそれを返し、なければ作ってそのパスを返します。

その後、insertPhotoUriメソッドでコンテンツプロバイダーへ画像ファイルの登録処理を行い、URIを取得します。コンテンツプロバイダーに画像情報を登録することで、標準の画像ギャラリーからも撮影した写真を閲覧できるようになります。

### MainActivity.java

```
private String buildPhotoDir() {
    String dirPath = "";
    File photoDir = null;
    File extStorageDir = Environment.getExternalStorageDirectory();
    if (extStorageDir.canWrite()) {
        photoDir = new File(extStorageDir.getPath() + "/" + getPackageName());
        if (!photoDir.exists()) {
            photoDir.mkdirs();
        }
        if (photoDir.canWrite()) {
            dirPath = photoDir.getPath();
        }
    }
    return dirPath;
}

private Uri insertPhotoUri() {
    long currentTimeMillis = System.currentTimeMillis();

    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd_HHmss");
    String title = dateFormat.format(new Date(currentTimeMillis));
    String fileName = "IMG_" + title + ".jpg";
    String path = buildPhotoDir() + "/" + fileName;
    File file = new File(path);

    // コンテンツプロバイダーへのセット
    ContentValues values = new ContentValues();
    values.put(Images.Media.TITLE, title);
    values.put(Images.Media.DISPLAY_NAME, fileName);
    values.put(Images.Media.MIME_TYPE, "image/jpeg");
    values.put(Images.Media.DATA, path);
    values.put(Images.Media.DATE_TAKEN, currentTimeMillis);
    if (file.exists()) {
        values.put(Images.Media.SIZE, file.length());
    }
    Uri uri = getContentResolver().insert(Images.Media.EXTERNAL_CONTENT_URI, values);
    return uri;
}
```

check!

## コンテンツプロバイダーについて

すでにコンテンツプロバイダーに触っていますが、写真の情報もコンテンツプロバイダーに格納されています。コンテンツプロバイダーとは、Androidにおいて、異なるアプリケーション間でデータのやりとりを行うためのしくみです。多岐にわたって使用されていて、たとえば連絡先もコンテンツプロバイダーにあります。同様に、写真の情報もコンテンツプロバイダーに格納されています。

SDカードに保存するときの注意点は、画像を保存してもAndroidデータベースに登録しないと、再起動するまでギャラリー等のアプリで見られないことです。そのため、保存後にファイルパスをAndroidデータベースに登録する必要があります。

最後は、AndroidManifest.xmlです。今回はSDカードを使用していることへのパーミッションと、画面を横向きにすることへの対応が入ります。

AndroidManifest.xml

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="jp.techinstitute.cameraex.MainActivity"
        android:screenOrientation="landscape"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

では、実行してみましょう。実行すると図6のような画面が表示され、ボタンを押すと図7のように撮影画面に切り替わります。ここで撮影したものは、ギャラリーアプリからも確認できます。

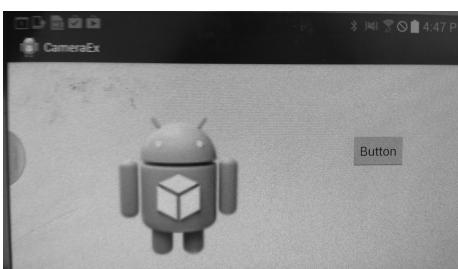


図6:実行画面。ここのボタンを押すと、



図7:写真撮影画面に切り替わる



## 7-3-2 カメラプレビューから写真を撮ってみよう！

今まででは、カメラアプリを呼び出しての写真撮影でした。今度は撮影プレビューを画面に出してみましょう。ここでは「SurfaceView」にカメラのプレビュー画面を出します。

今回主役となるのは「`android.hardware.Camera`」クラスです。

メソッド名	説明
<code>getNumberOfCameras()</code>	<code>Android</code> 端末が持つカメラの数を取得する
<code>getCameraInfo(i, camerainfo)</code>	指定デ <small>No</small> のカメラの <code>CameraInfo</code> を取得する。
<code>open()</code>	カメラのオープン
<code>release()</code>	カメラの解放
<code>setPreviewCallback()</code>	プレビュー完了時のコールバックメソッドの登録
<code>Camera.PreviewCallback</code>	プレビュー完了時に呼び出されるコールバックメソッド
<code>Camera.PreviewCallback.onPreviewFrame (byte[] data, Camera camera)</code>	プレビュー更新時のフレーム情報
<code>startPreview()</code>	プレビューの開始
<code>stopPreview()</code>	プレビューの停止

表3:Cameraクラスの主なメソッド

それでは、新しく「`CameraPreview`」プロジェクトを作成します。

アプリケーション名:	<code>CameraPreview</code>
プロジェクト名:	<code>CameraPreview</code>
パッケージ名:	<code>jp.techinstitute.cameralpreview</code>
最小必須SDK:	<code>API16:Anndroid4.1(Jelly Bean)</code>
ターゲットSDK:	<code>API16:Anndroid4.1(Jelly Bean)</code>
次でコンパイル:	<code>API16:Anndroid4.1(Jelly Bean)</code>
アクティビティ名:	<code>MainActivity</code>
レイアウト名:	<code>activity_main</code>
フラグメントレイアウト名:	<code>fragment_main</code>
ナビゲーションタイプ:	なし
テーマ:	<code>Holo Light with Dark Action Bar</code>
アクティビティのタイプ:	<code>Blank Activity with Fragment</code>

表4:CameraPreviewプロジェクトの構成

今回は、`SurfaceView`をそのまま画面に表示するので、レイアウトXMLファイルは使用しません。

`SurfaceView`の詳しい説明は別の講義で行います。ゲームのような、描画内容がひんぱんに変わる処理を行いたい場合に使用される、高速な描画機能です。また「`SurfaceHolder`」は、`SurfaceView`での描画を実現するために、描画内容を管理するクラスです。

「`SurfaceHolder.Callback`」というインターフェースを実装すると、`SurfaceHold`

erが生成されたタイミングで、surfaceCreatedメソッドが呼び出されます。そのタイミングで setPreviewDisplay(SurfaceHolder holder)メソッドをコールし、カメラのプレビューを開始します。

まずはSurfaceViewを継承したCameraViewを作成します。

### CameraView.java

```
public class CameraView extends SurfaceView
    implements SurfaceHolder.Callback, Camera.PictureCallback {

    private SurfaceHolder holder;
    private Camera camera;
    private MainActivity activity;

    public CameraView(Context context) {
        super(context);
        holder = getHolder();
        holder.addCallback(this);
        activity = (MainActivity)context;
    }

    public void surfaceCreated(SurfaceHolder holder) {
        try {
            // カメラをオープン
            camera = Camera.open();
            // プレビューディスプレイ（表示先）を設定
            camera.setPreviewDisplay(holder);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int width,
        int height) {
        // 画面に変化があったときは startPreview
        camera.startPreview();
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
        // プレビューを停止
        camera.stopPreview();
        // カメラをリリース
        camera.release();
        camera=null;
    }
}
```

CameraViewクラスには、「SurfaceHolder.Callback」を実装しています。SurfaceViewが作成された時点では呼び出される「surfaceCreated」メソッドにてCamera.open()を実行し、カメラデバイスのインスタンスを確保します。「setPreviewDisplay」に「holder」を渡し、表示先をSurfaceViewにしています。画面に変化があったときに呼び出される「surfaceChanged」には、カメラデバイスの「sta

rtPreview」を記述します。そして、SurfaceViewが破棄されるときに呼ばれる「surfaceDestroyed」には、プレビューを停止するstopPreviewメソッドと、カメラデバイスを解放する「release()」メソッドを記述します。

続いて、MainActivityです。「setContentView」にCameraViewのインスタンスを渡しているのみです。

#### MainActivity.java

```
public class MainActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(new CameraView(this));  
    }  
}
```

AndroidManifest.xmlファイルには、CameraExの時と同様の「android.permission.WRITE\_EXTERNAL\_STORAGE」の他に、カメラのパーミッション「android.permission.CAMERA」を新しく追加します。また、CameraExの時と同様に、android:screenOrientationはlandscapeにします。

#### AndroidManifest.xml

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />  
<uses-permission android:name="android.permission.CAMERA" />  
  
<application  
    android:allowBackup="true"  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme" >  
    <activity  
        android:name="jp.techinstitute.cameralibrary.MainActivity"  
        android:label="@string/app_name"  
        android:screenOrientation="landscape" >  
        <intent-filter>  
            <action android:name="android.intent.action.MAIN" />  
            <category android:name="android.intent.category.LAUNCHER" />  
        </intent-filter>  
    </activity>  
</application>
```

では、ここで実行してみましょう。カメラのプレビュー画面が表示されましたか？

プレビュー画面の表示ならこれだけで良いのですが、まだ撮影機能が備わっていないません。MainActivity.javaにCameraExプロジェクトのときと同様に、保存ファイルのパスを生成し、ContentProviderへ登録します。違いは、byteの配列を受け取って、それをファイルとして書き出しているところです。

## MainActivity.java

```

public Uri insertPhoto( byte[] data ) throws IOException {
    long currentTimeMillis = System.currentTimeMillis();

    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyyMMdd_HHmmss");
    String title = dateFormat.format(new Date(currentTimeMillis));
    String fileName = "IMG_" + title + ".jpg";
    String path = buildPhotoDir() + "/" + fileName;
    File file = new File(path);

    FileOutputStream out=null;
    try {
        out=new FileOutputStream(file);
        out.write(data);
        out.close();
    } catch (Exception e) {
        if (out!=null) {
            out.close();
        }
    }

    // コンテンツプロバイダーへのセット
    ContentValues values = new ContentValues();
    values.put(Images.Media.TITLE, title);
    values.put(Images.Media.DISPLAY_NAME, fileName);
    values.put(Images.Media.MIME_TYPE, "image/jpeg");
    values.put(Images.Media.DATA, path);
    values.put(Images.Media.DATE_TAKEN, currentTimeMillis);
    if (file.exists()) {
        values.put(Images.Media.SIZE, file.length());
    }
    Uri uri = getContentResolver().insert(Images.Media.EXTERNAL_CONTENT_URI, values);
    return uri;
}

```

写真の撮影は、画面をタッチしたときに行われるようにしましょう。SurfaceViewへのTouchEvent処理の中に記述します。

## CameraView.java

```

@Override
public boolean onTouchEvent(MotionEvent event) {
    // タッチダウンイベントで
    if (event.getAction()==MotionEvent.ACTION_DOWN) {
        camera.takePicture(null,null,null,this);
    }
    return true;
}

```

画面へのタッチダウンイベントで、「takePicture」メソッドを呼び出すことで撮影した画像を取得します。

各引数には、「コールバック」と呼ばれるものを設定します。コールバックとは「呼

び出し先から実行してもらいたい処理」を設定することで、ここでは写真撮影後に実行したい処理を渡すことになります。具体的には、「Camera.PictureCallback」インターフェイスを実装したクラスのオブジェクトを渡します。ここではActivityに実装しているので、thisです

引数はそれぞれ、第1引数は撮影直後に呼ばれるコールバックの設定、第2引数は撮影された生データを受け取るコールバックの設定、第3引数は撮影されたJPEGデータを受け取るコールバックの設定となっています。

今回はJPEGデータを保存するので、第3引数のみコールバックを設定しました。

コールバックの中で、JPEGデータを受け取って、SDカードに保存します。以下は、コールバックの記述です。Camera.PictureCallbackの実装であるonPictureTakenメソッドをオーバーライドします。

#### CameraView.java

```
public void onPictureTaken(byte[] data, Camera camera) {  
    try {  
        activity.insertPhoto(data);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    // プレビュー再開  
    camera.startPreview();  
}
```

MainActivityの「insertPhoto」メソッドに、受け取ったJPEGデータを渡して、SDカードに保存します。

また、takePictureメソッドを呼び出すと プレビューが停止されてしまいます。プレビューを再開するためには、ここでstartPreviewを呼び出します。

それでは、実行してみましょう。画面をタッチすると写真が撮影され、撮影された内容はギャラリーから閲覧できます。ここでの流れは、以下の図8のようになります。

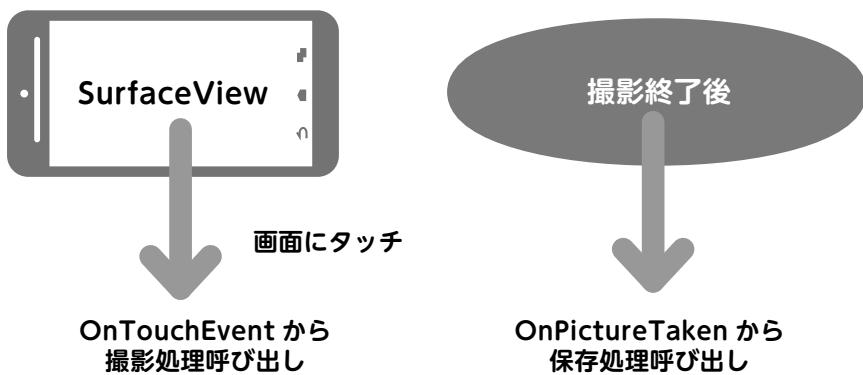


図8:プレビュー画面からの撮影・保存の流れ

アプリケーションの画面をSurfaceViewにし、そこにカメラのプレビューを表示。画面にタッチすると撮影処理を実行し、実行後に保存処理が行われるという流れです。

## まとめ

この節では、写真撮影処理について学びました。写真撮影にもAndroid特有のクセがあり、画像を活用する際にも一手間がかかります。この節では、それを通じてコンテンツプロバイダーやSurfaceViewについても学びました。

APIはAndroidがもつ様々な機能への入り口です。紹介しきれなかったものもたくさんありますが、ここで得た知識を活用することで、自分で新しい機能の使い方を調べて学んで行くことも可能になるでしょう。たとえば、撮影関連では「顔認識処理」という面白い機能が簡単に使えます。ぜひ自分で調べてみてください。

これから応用編でも、講義で教わったことだけではなく、自分で掘り下げてみることをやっていきましょう!