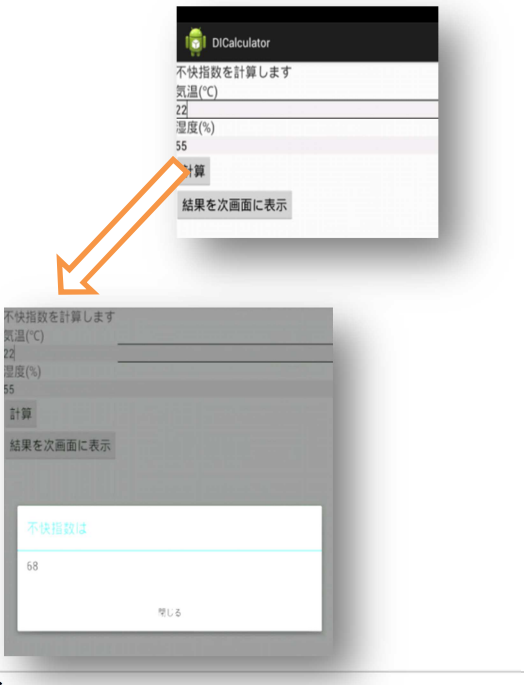
4　ボタンが押されたらイベントを取り扱う

ボタンの押下時に、入力された内容を確認するダイアログを表示するようにします。



4.1　リソースを追記する

網掛け部分を追加してください。

ファイル名：　res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
・・・<省略>・・・
    <string name="button_calculate">計算</string>
    <string name="label_di_description">不快指数は</string>
    <string name="button_close_dialog">閉じる</string>
</resources>
```

4.2　ボタン押下時のイベントに応答する処理を追記する

網掛け部分を追加してください。

ファイル名：　src/jp.edu.mie /DICalculatorActivity.java

```
public class DICalculatorActivity extends Activity
{
    private int temperature;//温度
    private int humidity;//湿度
    private EditText texttemperature;//温度入力値
    private EditText texthumidity;//湿度入力値
    private int di;//不快指数
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);//レイアウトの表示
```

```
        //リソースオブジェクトの作成
        texttemperature = (EditText)findViewById(R.id.text_temperature);
        texthumidity = (EditText)findViewById(R.id.text_humidity);//          ①
        Button button = (Button)findViewById(R.id.button_calculate);

        button.setOnClickListener(new calculateClickListener()); //②
    }
    class calculateClickListener implements OnClickListener //③
    {

        public void onClick(View view)
        {
            calculate();
            final AlertDialog.Builder builder =
                        new AlertDialog.Builder(DICalculatorActivity.this);
            builder.setTitle(R.string.label_di_description);
④           builder.setMessage(String.valueOf(di));
            builder.setPositiveButton(R.string.button_close_dialog,
                                      new calculateDialogClickListener());
            builder.show();
        }
      class calculateDialogClickListener
                              implements DialogInterface.OnClickListener//⑤
        {
            @Override
           public void onClick(DialogInterface dialog, int whichButton)//⑥
            {
                setResult(RESULT_OK);  //⑦処理が成功した
            }
        }
    }
    void calculate()
    {
        temperature =
                  Integer.parseInt(texttemperature.getText().toString());
⑧       humidity = Integer.parseInt(texthumidity.getText().toString());
⑨       di = (int) (0.81 * temperature + 0.01 * humidity
                                    * (0.99 * temperature - 14.3) + 46.3);
    }
}
```

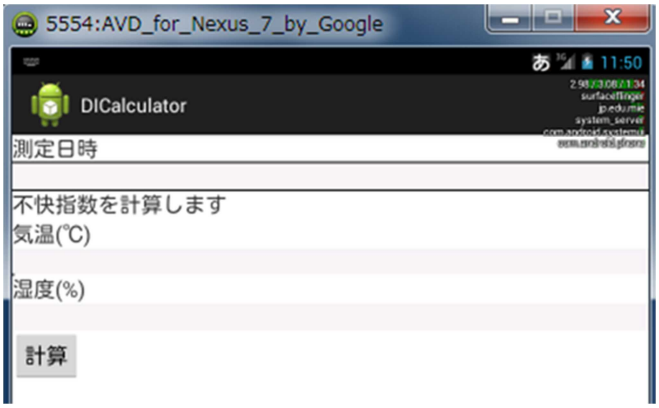
演習

【演習 4】この時点で実行してみる。

DI の計算結果が表示されます。

4.2.1 処理の流れ

（1）　レイアウトの表示



（2）①　気温に２２、湿度に５５を入力すると

texttemperature　２２

texthumidity　５５

が文字列として代入される。

（3）⑧　計算ボタンをクリック（タップ）すると

temperature　２２

humidity　５５

が数値として代入される。

（4）⑨　不快指数を計算する。

di　６８

（5）④　アラートダイアログを表示する。

a　インスタンスの生成

b　諸項目の設定

c　コールバックリスナーの登録

d　表示

（6）⑥　閉じるをタップしてアラートダイアログを閉じる。

4.2.2　ソースの解説

①texttemperature = (EditText)findViewById(R.id.text_temperature);
texthumidity = (EditText)findViewById(R.id.text_humidity);
Button button = (Button)findViewById(R.id.button_calculate);

「温度」と「湿度」の Edittext のインスタンスをリソースから取得しています。

texttemperature = (EditText)findViewById(R.id.text_temperature);

Activity クラスで定義されている findViewById(リソース ID)というメソッド(this が省略されている）は、ソースコード外のリソースを利用する時に使います。つまり、引数として指定された id からビューを探すメソッドです。this は自分自身、この場合には、DICalculatorActivity クラスになります。つまり、findViewById は、Activity クラスで定義されているメソッドで、findViewById(0x7f070001);とも書かれます。

このコードは、EditText を参照する texttemperature 変数を定義し、その中身として this.findViewById メソッドの結果を代入するものです。

findViewById は、さまざまなビューへの参照を返すため、View クラスを返すメソッドになっているので、EditText クラスのオブジェクトを参照する texttemperature に代入するには､データ型が一致していないので､戻り値をそのビューのクラス(EditText)でキャストしなければなりません。

この引数には「R.id.text_temperature」が指定されています。「R」はこのアプリケーション自身のリソースをまとめたオブジェクトで､「id」はその中の id の定義になります。つまり､これはリソースとして定義されている main.xml の「text_temperature」という名前の id を表します。これは､xml の id 定義である「@+id/text_temperature」と対応します。

xml ファイルで「@+id/X」という型式で定義された id はプログラムからは､「R.id.X」で参照できます。

Button button = (Button)findViewById(R.id.button_calculate);

リソースの名前が「button_calculate」のビューのオブジェクトを取得します。このビューはButton クラスのオブジェクトですのでButton クラスでキャストして取得しています。

②button.setOnClickListener(new calculateClickListener());

リスナーを登録します。

このソースは次のソース 2 行を 1 行にまとめたものです。

calculateClickListener c = new calculateClickListener();

button.setOnClickListener(c);

これは、button がクリックされたら calculateClickListener()を実行しなさいという意味です。

ボタンでクリック処理を行うためにはまずボタンがクリックされた時に発生するイベントを受け取るようにする必要があります。クリックイベントを受け取るようにするには「Button」クラスの親クラスである「View」クラスで用意されている「setOnClickListener」メソッドを使います。

引数には「OnClickListener」インタフェースを実装したクラスのオブジェクトを指定します。イベント発生時に呼び出されるメソッドは「onClick」メソッドとなります。

③class calculateClickListener implements OnClickListener
public void onClick(View view)

OnClickListener インターフェースの onClick メソッドを calculateClickListener クラスの中で実装するという意味です。

④final AlertDialog.Builder builder =
new AlertDialog.Builder(DICalculatorActivity.this);
builder.setTitle(R.string.label_di_description);
builder.setMessage(String.valueOf(di));
builder.setPositiveButton(R.string.button_close_dialog,
new calculateDialogClickListener());
builder.show();

AlertDialog.Builder クラスのインスタンス(builder)に対して計算結果を渡し、ダイアログを表示しています。　alert：アラート

AlertDialog は、「OK/キャンセル」による確認や質問の答えを入力するダイアログなど、簡単な入力インターフェースとして利用できます。

・builder.setTitle(R.string.label_di_description);

アラートダイアログのタイトルを設定します。

・builder.setMessage(String.valueOf(di));

アラートダイアログのメッセージを設定します。

・builder.setPositiveButton(R.string.button_close_dialog,
new calculateDialogClickListener());

アラートダイアログの閉じるボタンがクリックされた時に呼び出されるコールバックリスナーを登録します。

・builder.show();

アラートダイアログを表示します。

・final AlertDialog.Builder builder =
new AlertDialog.Builder(DICalculatorActivity.this);

final は変数の内容を変更しないという宣言です。

⑤class calculateDialogClickListener implements DialogInterface.OnClickListener

「DialogInterface.OnClickListener」インタフェースはダイアログに含まれるボタンのクリック処理に使用されます。クリック時に呼び出されるメソッドは「onClick」メソッドとなります。

⑥public void onClick(DialogInterface dialog, int whichButton)

1 番目の引数にはクリックが発生した「android.content.DialogInterface」インターフェースを実装したクラスのオブジェクトが渡されてきます。インターフェースを実装したクラスとしては「AlertDialog」クラス，「DatePickerDialog」クラス，「Dialog」ク

ラス，「ProgressDialog」クラス，「TimePickerDialog」があります。イベント発生元のダイアログを判別するのに使用します。

2番目引数にはクリックが発生したボタンの種類が渡されてきます。ボタンの種類は「android.content.DialogInterface」インターフェースで定義されており次のどちらかです。

・android.content.DialogInterface.BUTTON1

・android.content.DialogInterface.BUTTON2

⑦setResult(RESULT_OK);

setResult は、メインアクティビティに実行の結果を通知するために必要なものです。RESULT_OK は、インテントとしては成功したことを返すためのものです。

⑧humidity = Integer.parseInt(texthumidity.getText().toString());

EditText のインスタンスから入力されている文字列を取得し、数値へ変換しています。エディットテキストのテキストを取得します。

⑨di = (int) (0.81 * temperature + 0.01 * humidity
* (0.99 * temperature - 14.3) + 46.3);

DI を計算しています。

4.2.3 イベントドリブンの考え方（event-driven）

一般にアプリケーションは、ボタンを押す、キーを押すなど、外から様々な働きかけに応じて処理を行います。このような「外からの働きかけ」のことを**イベント**と呼びます。

Android のよく使うイベントには次のような種類があります。

イベントの種類	イベントの説明
クリックイベント	ボタンのクリックやチェックボックスの選択
タッチイベント	スクリーンにタッチやドラッグ
キーイベント	キーボードからのキー入力

タッチイベントには次のような種類があります。

日本語表記	英語表記	操作方法
タップ	Tap Single tap	画面を指先(ペン先)で1回叩く
ダブルタップ 二回タップ	Double tap	画面を指先(ペン先)で2回叩く(突く)
ロングタップ ロングプレス 長押し	Long press Long tap	画面を指先(ペン先)で長く押す
フリック	Flick	画面に触れた指先(ペン先)を少しスライドさせ素早く払う(弾く)ようにタッチ
右フリック	Flick Right Fling right Right flick	左から右にフリック→
左フリック	Flick Left Fling left Left flick	右から左にフリック←
上フリック	Flick up Fling up Up flick	下から上にフリック↑

下フリック	Flick down Fling down Down flick	上から下にフリック↓
マルチタッチ マルチタップ	Multi-tap Multitech	同時に 2 本以上の指で操作出来ること
ピンチ操作 ピンチズーム	Pinch Pinch to Zoom Pinch-Zoom Pinch and Zoom	同時に 2 本の指で操作すること
ピンチイン	Pinch-in Pinch-close	2 本の指の間隔を縮めて画面をつまむようにする動作
ピンチアウト	Pinch-out Pinch-open	2 本の指の間隔を広げる動作

「キーが入力された」「マウスがクリックされた」といったイベントが発生する場合、そのイベントに対応する処理をするためのメソッドを用意する必要があります。それを**イベントハンドラ**といいます。そのメソッドを持つクラスが**イベントリスナー**です。

【クラス】イベントリスナー　setOnClickListener（イベントリスナー）

≪メソッド≫イベントハンドラ
≪メソッド≫イベントハンドラ

Java では、イベントハンドラの集合がイベントリスナーです。各イベントごとに、イベントリスナークラスを作り、その中にイベントハンドラのコードを書いていきます。

プログラムは基本的には上から下へと続けて実行されます。しかし、スマートフォンや Windows のような GUI 環境では、「ボタンをクリックした」「文字を入力した」といったユーザのアクションに対して処理が実行されます。

プログラムというのは待機するということができないはずですが、なぜ、スマートフォンのアプリはアクションが起こるまで待っていることができるのでしょうか。

タッチやマウス、キーボードの操作のようなアクションのことを**イベント**といいます。Android のアプリは、起動後には何かイベントが発生しないかを監視しています。つまり、アイドル状態で待っていることになります。

```
イベントを待つ待機ループ処理　　①

アプリが起動された(){
    処理
}
マウスが押された(){
    処理
}
キーボードが押された(){
    処理
}
アプリの終了(){
    処理
}
```

①の部分でイベントの発生を監視しています。イベントを感知したら、各イベントに割り当てられたそれぞれの関数を呼び出し、各関数内でアプリを呼び出します。関数の処理が完了したら、またイベント待ちのループに戻ります。これを**イベントドリブン**といいます。

一般に①で動いているプログラムはアプリ（ＯＳ）ですが、プログラマが作る部分ではありません。プログラマは、その下の各関数の中身だけを作成します。

例えば、ボタンが押されたら、まず液晶ディスプレイのデバイスドライバを介して、ＯＳが押された場所を感知して、そのボタンオブジェクトに対して、押されたことをメッセージで通知します。ここで初めて、ボタンオブジェクトは自分が押されたことに気が付きます。

そのあと、ボタンオブジェクトは、あらかじめ登録していた別のオブジェクトに対して、あらかじめ決めていた動作（メソッド）を指示します。

＜実際の処理＞

```
        //ボタンオブジェクトが別のオブジェクトを登録
        button.setOnClickListener(new calculateClickListener());
    }
    class calculateClickListener implements OnClickListener
    {
        calculate();
        public void onClick(View view)
        {
            final AlertDialog.Builder builder = new
                        AlertDialog.Builder(DICalculatorActivity.this);
            builder.setTitle(R.string.label_di_description);
            builder.setMessage(String.valueOf(di));
            ・・・
```

onClick メソッドの引数 View には、イベントリスナーが呼び出されるときに、この引数にはどのビューでイベントが起こったかということが渡されます。

4.2.4　内部クラスと匿名クラス（無名クラス）

クラスの宣言を別のクラスの内部で行うことができます。このようなクラスを**内部クラス**といいます。内部クラスは外側のクラス（外部クラス）の private 修飾子のついたフィールドとメソッドにもアクセスできます。

Java 言語では、あるインタフェースを実装したクラスの宣言をしつつ、そのクラスのインスタンスを生成することができます。 このようにして生成されたインスタンスは、インタフェース名を引数の型にもつオブジェクトとして、メソッドの引数に使用できます。またこうして宣言された名前のないクラスを**匿名クラス**（**無名クラス**）といいます。

（１）自分自身の Activity にインタフェースを実装する方法（内部クラスを使わない方法）

```
public class DICalculatorActivity extends Activity  implements OnClickListener
{
     ・・・
    protected void onCreate(Bundle savedInstanceState)
    {
        ・・・
        Button button = (Button)findViewById(R.id.button_calculate);
        button.setOnClickListener(this);
    }
    void calculate()
    {
    ・・・
    }
    @Override
    public void onClick(View view)
    {
```

```
        ・・・
    }
    ・・・
}
```

（2）内部クラスを使う方法

```
public class DICalculatorActivity extends Activity
{
     ・・・
    protected void onCreate(Bundle savedInstanceState)
    {
        ・・・
        Button button = (Button)findViewById(R.id.button_calculate);
        button.setOnClickListener(new calculateClickListener());
    }
    class calculateClickListener implements OnClickListener
    {
        calculate();
        public void onClick(View view)
        {
            ・・・
        }
        ・・・
    }
}
```

（3）匿名クラス（無名クラス）を使う方法

匿名クラスを使う場合には、対象となるボタン毎にクリック時の処理を記述していきます。イベント発生元のビューを意識せずに個々のボタン毎に処理を記述できます。

記述方法の最初に new がついていることでも示されていますが、匿名クラスはその時点でインスタンス化されます。

button.setOnClickListener(new　・・・{クラスの定義});

```
public class DICalculatorActivity extends Activity
{
    ・・・
    protected void onCreate(Bundle savedInstanceState)
    {
        ・・・
        Button button = (Button)findViewById(R.id.button_calculate);
        button.setOnClickListener(new OnClickListener
        {
            public void onClick(View view)
            {
                ・・・
            }
        });
    }
}
```