５　複数の画面を使う（インテント　Intent）

インテントは､アクティビティ間やアプリケーション間でやり取りするメッセージの役割を果たすものです｡インテントを送る事で､アプリケーション内の画面遷移や他アプリケーションの呼び出しを行います｡主にアクティビティを起動する際のパラメータに使われます。

intent:意思

付加情報(配列、文字列、整数型など)を別アプリケーションに通知でき、他の Activity の呼び出し方法としてよく使われます。

インテントには「明示的インテント」と「暗黙的インテント」の２種類が存在します。明示的インテントでは、アクティビティを直接指定して起動させます。一方、暗黙的インテントは明示的に起動させるアクティビティを指定せずにインテントのパラメータに起動するアクティビティをある程度推測できるだけの情報(インテントフィルター)を入れておくと、該当するアクティビティが起動するようになっています。

■インテントの概念図

インテントを用いて、DI 計算の最初の画面で入力された計算結果を別画面に表示するようにします。

Android アプリの画面遷移では Activity を積み重ねていく方法が主に使われます。呼び出し元の DICalculatorActivity の上に新たな ResultActivity を置くことにより、別の画面に移ったように見せる手法です。

■表示方法

手順①DICalculatorActivity 画面を表示する。

手順②「結果を次画面に表示」ボタンをクリックすると ResultActivity 画面が表示される。
DICalculatorActivity 画面は ResultActivity 画面の下で待機する。

手順③「閉じる」ボタンをクリックすると
ResultActivity 画面が閉じられ、待機していた DICalculatorActivity 画面が表示される。

①

②

③

5.1　リソースの追記

網掛けの箇所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_close_dialog">閉じる</string>
    <string name="button_show_next_activity">結果を次画面に表示</string>
    <string name="label_to_55">寒い</string>
    <string name="label_55_60">肌寒い</string>
    <string name="label_60_65">何も感じない</string>
    <string name="label_65_70">快い</string>                ※<読み方>こころよい
    <string name="label_70_75">暑くない</string>
    <string name="label_75_80">やや暑い</string>
    <string name="label_80_85">暑くて汗が出る</string>
    <string name="label_from_85">暑くてたまらない</string>
</resources>
```

5.2　レイアウトの追記

main.xml の最後に「結果を次画面へ表示」するボタンを追加します。

ファイル名：　res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
・・・<省略>・・・
    <Button
        android:id="@+id/button_show_next_activity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="@string/button_show_next_activity"
        android:textSize="20sp" />
</LinearLayout>
```

今回新たに作成する結果表示用アクティビティが利用するレイアウトを定義します。

ファイル名： res/layout/result.xml（**新規作成**）

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/label_di_description"
        android:textSize="20sp"   />
    <TextView
        android:id="@+id/label_divalue"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#F0F0F0"
        android:textIsSelectable="true"  <!--①-->
        android:textSize="20sp"   />
    <TextView
        android:id="@+id/lable_status"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#F9F9F9"
        android:textIsSelectable="true"
        android:textSize="20sp"   />
    <Button
        android:id="@+id/button_close"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_close_dialog"   />
</LinearLayout>
```

①android:textIsSelectable="true"

TextView のテキストを選択可能にします。

5.3 入力用アクティビティ(DICalculatorActivity)の追記・修正

網掛けの箇所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 di;
    private Button button;
    private Button buttonShowNextActivity;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        ・・・
```

```
        button = (Button)findViewById(R.id.button_calculate);
        button.setOnClickListener(new calculateClickListener());
        buttonShowNextActivity
                    = (Button)findViewById(R.id.button_show_next_activity);
        buttonShowNextActivity.setOnClickListener
                                            (new calculateClickListener());
    }
    class calculateClickListener implements OnClickListener
    {
        @Override
        public void onClick(View view)
        {
            calculate();
            if(view == button) //①
            {
                final 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();
            }else if(view == buttonShowNextActivity)  //②
            {
               Intent intent = new Intent(getApplicationContext(),
                                                        ResultActivity.class);  //③
               intent.putExtra("DI", di); //④
               startActivityForResult(intent, 0);  //⑤
            }
        }
        class calculateDialogClickListener
                                implements DialogInterface.OnClickListener
        {
            ・・・
        }
    }
    void calculate()
    {
        ・・・
    }
}
```

①if(view == button)

button（「計算」ボタン）が押されたときに実行するイベントリスナーです。計算結果はアラートダイアログボックスに表示されます。

②}else if(view == buttonShowNextActivity)

buttonShowNextActivity（「結果を次画面に表示」ボタン）が押されたときに実行するイベントリスナーです。計算結果は別画面に表示されます。

③Intent intent = new Intent(getApplicationContext(),ResultActivity.class);

インテントのインスタンスを生成し、そこに遷移先の Activity に関する情報を格納します。ここでは、元画面と次画面を明示的に指定しています。このことを**明示的インテント**と呼びます。Intent クラスは、Activity に関する様々な情報を保持しています。異なる Activity から同じ Intent オブジェクトにアクセスできることが特徴です。

④intent.putExtra("DI", di);

計算した「ＤＩ」の値を付加情報に登録します。ここで登録したものは、インテントを受け取った次画面で自由に参照することができます。付加情報は、文字列をキーとしたMapとして保持されているため、キーとなる文字列（"DI"）と値のペアで設定します。

⑤startActivityForResult(intent, 0);

startActivityForResultメソッドを使って、ResultActivityを起動します。第一引数には、Intentを指定します。startActivityForResultは、引数で与えられたIntentが保持しているActivityを起動します。このIntentには、生成時にResultActivity.classを与えたので、startActivityForResultにより、ResultActivity.classが起動され、端末上には、２番目の画面が表示されます。

startActivityForResultメソッドは、遷移先のActivityが終了したときに、遷移元のActivityに何らかの結果を伝えることができます。このため、第二引数にはリクエストコードを指定します。今回は０を渡しています。リクエストコードは、再び親のActivityへ戻ってきたときに、どのActivityから戻ってきたのかを区別するための値で、０以上の数値を指定します。

Activityクラスの startActivityForResultメソッド	Intentでの情報の受け渡しを開始（返り値有り）
Activityクラスの startActivityメソッド	Intentでの情報の受け渡しを開始（返り値無し）

5.4　結果表示用のアクティビティ(ResultActivity)の作成

「ResultActivity.java」という名称でAndroidファイルを新規作成し、結果を表示するアクティビティを定義します。

・新規作成の手順

①Eclipse画面のsrcの「jp.edu.mie」をクリックする。

②「ファイル」－「新規」－「その他」－「Androidアクティビティー」

③「次へ」

ファイル名：　src/ jp.edu.mie /ResultActivity.java　（**新規作成**）

```
public class ResultActivity extends Activity
{
    private int di;//不快指数
    private Intent intent;//インテント
    private Button button;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.result);
        TextView diValue = (TextView)findViewById(R.id.label_divalue);
        TextView status = (TextView)findViewById(R.id.lable_status);
        button = (Button)findViewById(R.id.button_close);
        button.setOnClickListener(new closeClickListener());

        intent = getIntent(); //①
        if (intent != null) {
            di = intent.getIntExtra("DI",0); //②
            diValue.setText(String.valueOf(di));//③
            if (di <= 55) {
                status.setText(R.string.label_to_55);
```

```
            } else if (di <= 60) {
                status.setText(R.string.label_55_60);
            } else if (di <= 65) {
                status.setText(R.string.label_60_65);
            } else if (di <= 70) {
                status.setText(R.string.label_65_70);
            } else if (di <= 75) {
                status.setText(R.string.label_70_75);
            } else if (di <= 80) {
                status.setText(R.string.label_75_80);
            } else if (di <= 85) {
                status.setText(R.string.label_80_85);
            } else {
                status.setText(R.string.label_from_85);
            }
        }
    }
    class closeClickListener implements OnClickListener
    {
        @Override
        public void onClick(View view)
        {
            setResult(RESULT_OK, intent); //④遷移元に渡す値、処理が成功した
            finish();  //自分自身を終了する
        }
    }
}
```

①intent = getIntent();

遷移元画面（DICalculatorActivity）で生成した Intent から付加情報を取得します。

②di = intent.getIntExtra("DI",0); //②

遷移元画面（DICalculatorActivity）から渡された不快指数の値(di)を付加情報より取得します。getIntExtra メソッドの第二引数には、対応するキーが存在しないときの戻り値を指定します。

③diValue.setText(String.valueOf(di));

不快指数の値(di)と判定結果をヴィジェットに設定します。

Stringクラスで用意されている valueOf メソッドを使って文字列に変換し、TextView 型の diValue に設定しています。

④setResult(RESULT_OK, intent);

結果コードを設定しています。今回は、Activity クラスで元から用意されている RESULT_OK という定数を使っています。

5.5 数値から文字列への変換方法

数値から文字列への変換方法には３つあります。

5.5.1 toString メソッドを使用する。

基本データ型をラッパークラスのオブジェクトに変換した後で、各ラッパークラスで用意されている toString メソッドを使って文字列に変換する方法です。toString メソッドはオブジェクトの文字列表現を返します。

```
diValue.setText(di.toString());
```

5.5.2 valueOf メソッドを使用する。

String クラスで用意されている valueOf メソッドを使って文字列に変換する方法です。valueOf メソッドは対象となる値の種類毎に次のようなメソッドが用意されています。

```
static String valueOf(boolean b)
static String valueOf(char c)
static String valueOf(double d)
static String valueOf(float f)
static String valueOf(int i)
static String valueOf(long l)
```

valueOf メソッドは引数に指定された値の文字列表現を返します。例えば int 型の値を文字列に変換する場合は次のように記述します。

```
int i = 84;

String str = String.valueOf(i);
```

この場合、引数にはラッパークラスのオブジェクトではなく基本データ型の値を直接指定することに注意して下さい。

なお short 型と byte 型の引数を取るメソッドが用意さえていませんが、どちらのデータ型の値を valueOf メソッドの引数に指定しても文字列表現を取得できます。恐らく自動的に int 型へ変換されてメソッドが実行されていると考えられます。

5.5.3 ＋演算子を使用する

「+」演算子は文字列と数値が対象だった場合には数値を文字列に変換してから連結を行います。その為、基本データ型の値と空文字""を「+」演算子で連結することで数値を文字列に変換することができます。

例えば次のように記述します。

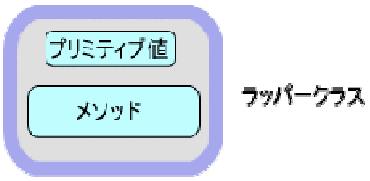
```
int i = 84;

String str = "" + i;
```

「+」演算子の対象の値の 1 つが文字列なので、もう一方の値である数値を文字列に変換してから連結します。連結する左側の文字列は空文字なので結果的に数値を文字列に変換したものが左辺に代入されることになります。

5.6 ラッパークラスとは

java では基本データ型として、byte, short, long, integer, float, char, double, boolean が規定されています。これらをプリミティブ型といいます。



プリミティブ型はオブジェクトではないので、それ自身のメソッドを持っていません。ところがオブジェクトを使ってゆくと、これらの基本型を操作しなければならない局面が出てきます。

そのためプリミティブ型をオブジェクトとして操作するクラスが各種定義されています。これらはプリミティブ型を包み込むという意味でラッパークラスといいます。

ラッパークラスは表で示されるように、プリミティブ型の種類によっていくつかの種類があります。

プリミティブ型	ラッパークラス
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

オブジェクトですので、それぞれ固有のプリミティ値を持ち、それを操作するメソッドを持っています。 使い方は色々ありますが１例として、数字の文字列(String 型) "123""123.45" はそのままでは計算ができませんが、ラッパークラスで数字に変換することによって数値計算することができます。

基本データ型は NULL データは代入できません。オブジェクト型なら NULL が代入できます。ラッパークラスは表で見られるように、頭が大文字になっています。

(注)String 型は当初からオブジェクトですので特にラッパークラスには入れていません。

5.7 マニフェストファイルの定義

マニフェストファイル（AndroidManifest.xml）は、Android アプリケーションに必ず必要なファイルです。Android アプリケーションにはどのようなコンポーネント（Activity やサービス）があるのか、また、それらがどのような性質で、どのような振る舞いをするのかを記述するためのものです。また、アプリケーションに必要なセキュリティもここで定義します。

ファイル名： AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"    //①
    package="jp.edu.mie"
    android:versionCode="1"   //②
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".DICalculatorActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".ResultActivity"
                  android:label="@string/app_name" >
        </activity>

    </application>
```

```
</manifest>
```

①xmlns:android="http://schemas.android.com/apk/res/android"

android のタグをあらわす名前空間です。android の部分は、プレフィックス(prefix)と呼ばれています。

②android:versionCode="1"

プレフィックスの android で表される名前空間に属するタグとなります。

5.7.1 マニフェストファイルの構造

マニフェストファイル（AndroidManifest.xml）は必ず manifest 要素から始まります。manifest 要素の下には次のような要素があります。

要素	用途
uses-permission	Android アプリケーションが正常に動作するために必要なアクセス権を示す。
permission	Android アプリケーションのデータやロジックを使うために、Activity やサービスが他のアプリケーションに要求するアクセス権を宣言する。
instrumentation	ログの記録や監視の目的で、Activity の開始といった重要なシステムイベントで呼び出されるコードを示す。
uses-library	マッピングサービスなど、オプションの Android コンポーネントに接続する。
intent-filter	Activity などの各アプリケーションとのつなぎ役をする Intent の仕様を記述します。
uses-sdk	実行に必要な Android SDK のバージョンを示す。
application	マニフェストが説明するアプリケーションの内部構造を定義する。
activity	アクティビティの登録
service	サービスの登録

(1)manifest タグの属性

属性	用途
android:package	アプリケーションのデフォルトのパッケージ。このパッケージで Android Market に登録する。
android:versionCode	内部的なアプリケーションのバージョン番号。アップデートの際は、この番号が公開されているバージョンよりも大きくなければ Android Market に登録できない。
android:versionName	ユーザーに見せるアプリケーションのバージョン。Android Market での紹介にこのバージョンが表示される。

(2)application タグの属性

属性	用途
android:icon	アプリケーションアイコン。このアイコンが Android Market の紹介で表示される。
android:label	アプリケーションの名前。このラベルが各コンポーネントのデフォルトのラベルとなる。

(3)activity タグの属性

属性	用途
android:name	Activity のクラス名。デフォルトのパッケージの相対パスで指定することもできる。
android:label	Activity のラベル名。

(4)intent-filter タグ

Activity などの各アプリケーションとのつなぎ役をする Intent の仕様を記述します。

アプリケーションが起動した時に最初に呼び出されるアクティビティの指定は、

```
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER"  />
```

とします。

(5)uses-sdk タグの属性

属性	用途
android:mminSdkVersion	最小の API レベル。この以下のバージョンが動作する端末からは Android Market で検索しても見つからない。指定しない場合は１が設定される。

5.8 ログの出力方法

ログにはデバック用に変数の値を出力したり実行時エラーのエラー内容を出力したりといった用途が考えられますが、後でログを確認する時に便利なように書き出される内容をいくつかの種類に分類して書き出す事ができるようになっています。

```
Log.d("MyApp", "x=" + x);
```

1 番目の引数にタグを表す文字列を指定し、2 番目の引数にログとして出力する文字列を指定します。 タグに指定した文字列を使ってログをフィルタすることが出来ます。特定のアプリケーションが出力したログだけを確認したい場合などのためにタグにはアプリケーションを表す任意の文字列などを指定しておくと便利です。 2 番目の引数には例えばデバック情報であれば変数の値や、エラー情報であれば例外発生時のエラー内容などを文字列として指定します。