

## 本日の内容

### TouchEvent

今回マニュアル感がとても強いです。

Android アプリ内で出来ることを増やす目的で授業を行います。

TouchEvent(公式ドキュメント)

<https://developer.android.com/training/gestures/movement?hl=ja>

### ■Touch とは

Android アプリ内に作成する、メニューボタンのようなものです。

使用する場合このオーバーフローメニューに対してもレイアウトファイルが必要になります。

### ■プロジェクト作成

- Empty Activity
- プロジェクト名：TryTouchEvent

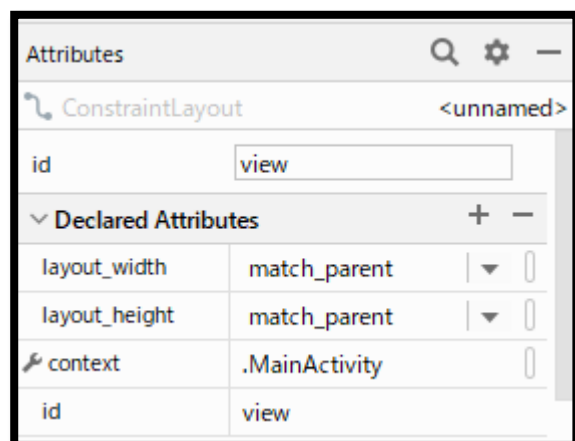
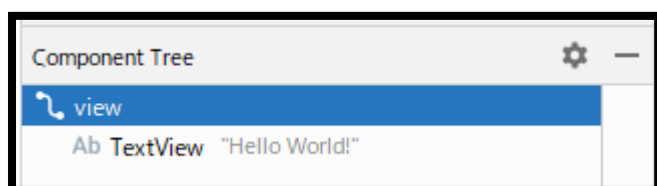
=====

### ■activity\_main.xml

今回、画面全体に対してタッチされたかの判定を取得したいため、

画面全体が該当するルートタグ(ConstraintLayout)に対して、タッチイベントの検知を行います。

PG 上で識別できるよう id を付与しておきましょう



## ■ MainActivity.java

先程の ConstraintLayout に対して処理を行えるよう、「変数の定義」と「view との関連付け」を行います

```
public class MainActivity extends AppCompatActivity {  
    private ConstraintLayout view; // 画面全体のview  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        // Viewの関連付け  
        view = findViewById(R.id.view);  
    }  
}
```

## ■ View.OnTouchListener

タッチしたかを判定するインターフェイスを実装します。

```
public class MainActivity extends AppCompatActivity implements View.OnTouchListener {  
    private ConstraintLayout view; // 画面全体のview  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        // Viewの関連付け  
        view = findViewById(R.id.view);  
    }  
}
```

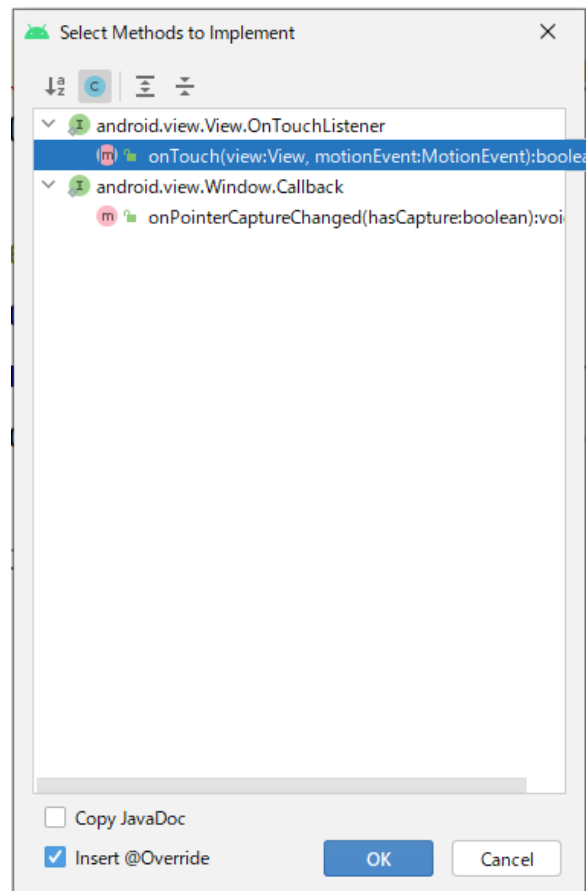
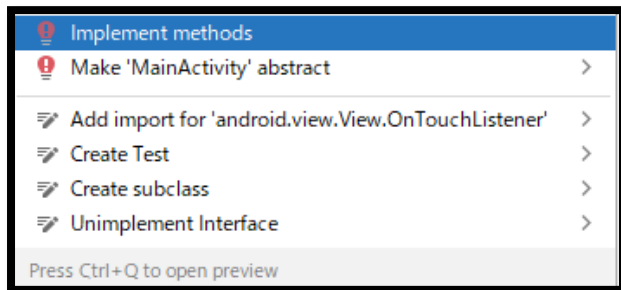
しかし、赤波線が出ていますね、これは必要なメソッドが実装されていないからです。

View.OnTouchListener をクリックした状態で「Alt」 + 「Enter」で

必要なメソッドを自動的に実装させましょう

## スマートフォンアプリ開発演習

今回は onTouch メソッドを実装します。



```
public class MainActivity extends AppCompatActivity implements View.OnTouchListener {
    private ConstraintLayout view; // 画面全体のview

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

        // Viewの関連付け
        view = findViewById(R.id.view);
    }

    @Override
    public boolean onTouch(View view, MotionEvent motionEvent) {
        return false;
    }
}
```

実装された onTouch メソッド内の戻り値を true に変更しておきましょう。

```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {
    // イベント処理終了:true 他イベント処理も行う:false
    return true;
}
```

onTouch メソッドを実装しましたが、このままでは何をタッチしたらこのメソッドを呼ぶのかが決まていません。

画面(ConstraintLayout)をタッチしたら、onTouch メソッドが呼ばれるようにしましょう

```
public class MainActivity extends AppCompatActivity implements View.OnTouchListener{
    private ConstraintLayout view; // 画面全体のview

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

        // Viewの関連付け
        view = findViewById(R.id.view);

        // タッチ検知の登録
        view.setOnTouchListener(this);
    }
}
```

これでタッチを検知出来るようになったので、ログ処理を記述して画面タッチが認識されているか確認してみましょう！

```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {
    Log.d( tag: "debug", msg: "onTouch() call");

    // イベント処理終了:true 他イベント処理も行う:false
    return true;
}
```

## スマートフォンアプリ開発演習

下部の Run タブを開いた状態で、Android の画面を 1 回クリックしてみましょう



何故か 2 回反応していますね。これは一体・・・？

### ■onTouch で渡される引数

先程の謎の鍵は引数にあります。

```
public boolean onTouch(View view, MotionEvent motionEvent) { }
```

第 1 引数：View view,

これは、タッチされた view が入っています。

第 2 引数： MotionEvent motionEvent

ここにはどのような操作をしたかが入っています。

私たちはタップしただけですが、

Android には

- ・ 画面に指が触れたアクション
- ・ 画面から指を離れたアクション

この 2 種類の処理が行われていたのです。

なので触れた時と離れた時で分岐をさせる必要があります。

どのアクションだったかは[motionEvent.getAction()]で取得可能です。

```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {
    // 操作の種類で分岐
    switch (motionEvent.getAction()){
        // 画面に触れた時
        case MotionEvent.ACTION_DOWN:
            Log.d( tag: "debug", msg: "onTouch() ACTION_DOWN");
            break;
        // 画面から離れた時
        case MotionEvent.ACTION_UP:
            Log.d( tag: "debug", msg: "onTouch() ACTION_UP");
    }

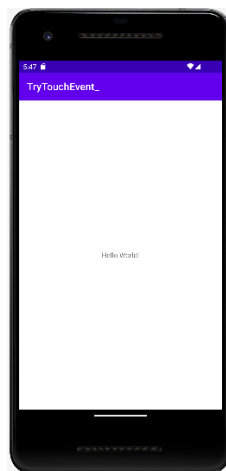
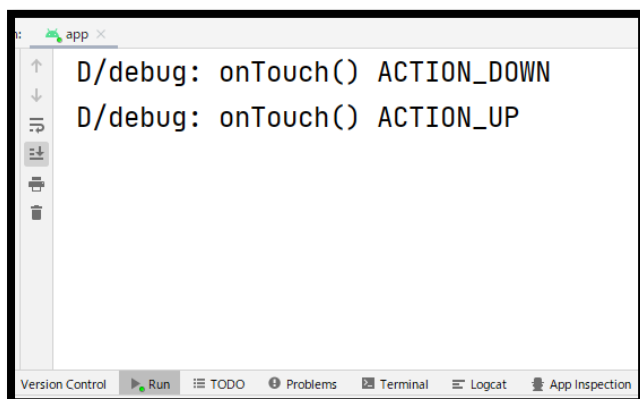
    // イベント処理終了:true 他イベント処理も行う:false
    return true;
}
```

実行してみると

押したときに ACTION\_DOWN

離れたときに ACTION\_UP

のログが出力されていますね！



## ■座標の取得

今回はフリック操作で方向を検知したいのですが、始点と終点がわからないと方向が求められません。

それぞれの座標を取得していきます。

まずは座標値を格納する変数の定義から。

```
public class MainActivity extends AppCompatActivity implements
    private ConstraintLayout view; // 画面全体のview
    private float downX; // 画面を触ったX軸位置
    private float downY; // 画面を触ったY軸位置
    private float upX; // 画面から離れたX軸位置
    private float upY; // 画面から離れたY軸位置

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

次にそれぞれの座標を取得します。

取得する方法は

motionEvent.getX()で X 軸

motionEvent.getY()で Y 軸

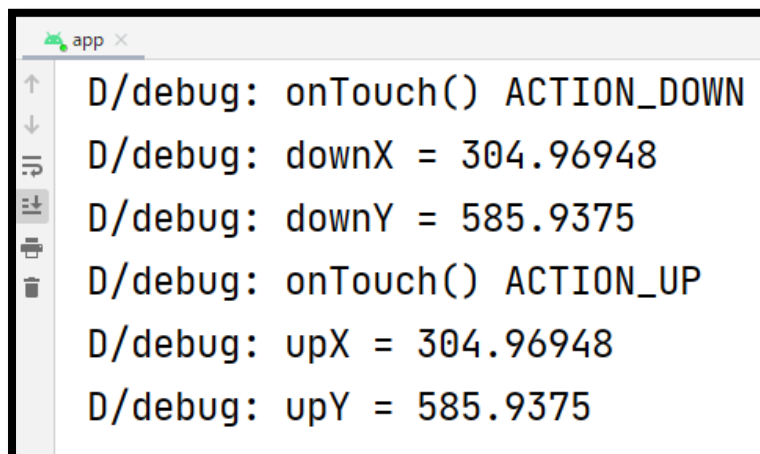
上記に値を取得します。その際,float 型で取得されます。

```
public boolean onTouch(View view, MotionEvent motionEvent) {
    // 操作の種類で分岐
    switch (motionEvent.getAction()){
        // 画面に触れた時
        case MotionEvent.ACTION_DOWN:
            downX = motionEvent.getX(); // タッチしたX軸位置を取得
            downY = motionEvent.getY(); // タッチしたY軸位置を取得
            Log.d( tag: "debug", msg: "onTouch() ACTION_DOWN");
            break;
        // 画面から離れた時
        case MotionEvent.ACTION_UP:
            upX = motionEvent.getX(); // 指を離れたX軸位置を取得
            upY = motionEvent.getY(); // 指を離れたY軸位置を取得
            Log.d( tag: "debug", msg: "onTouch() ACTION_UP");
    }
    // イベント処理終了:true 他イベント処理も行う:false
    return true;
}
```

開発者も座標がわかるようにログの出力するように追記しておきましょう。

```
public boolean onTouch(View view, MotionEvent motionEvent) {  
    // 操作の種類で分岐  
    switch (motionEvent.getAction()){  
        // 画面に触れた時  
        case MotionEvent.ACTION_DOWN:  
            downX = motionEvent.getX();    // タッチしたX軸位置を取得  
            downY = motionEvent.getY();    // タッチしたY軸位置を取得  
            Log.d( tag: "debug", msg: "onTouch() ACTION_DOWN");  
            Log.d( tag: "debug", msg: "downX = "+downX);  
            Log.d( tag: "debug", msg: "downY = "+downY);  
            break;  
        // 画面から離れた時  
        case MotionEvent.ACTION_UP:  
            upX = motionEvent.getX();    // 指を離れたX軸位置を取得  
            upY = motionEvent.getY();    // 指を離れたY軸位置を取得  
            Log.d( tag: "debug", msg: "onTouch() ACTION_UP");  
            Log.d( tag: "debug", msg: "upX = "+upX);  
            Log.d( tag: "debug", msg: "upY = "+upY);  
    }  
    // イベント処理終了:true 他イベント処理も行う:false  
    return true;  
}
```

タップした時のログ



```
app ×  
↑  
↓  
D/debug: onTouch() ACTION_DOWN  
D/debug: downX = 304.96948  
D/debug: downY = 585.9375  
D/debug: onTouch() ACTION_UP  
D/debug: upX = 304.96948  
D/debug: upY = 585.9375
```



```
app x
D/debug: onTouch() ACTION_DOWN
D/debug: downX = 176.98975
D/debug: downY = 904.9219
D/debug: onTouch() ACTION_UP
D/debug: upX = 849.9463
D/debug: upY = 661.9336
```

この座標を元に、上下左右どの方向にフリックしたのかを計算し分岐する必要があります。  
先に管理しやすいよう、方向の変数と格納用の変数を定義しておきましょう。

```
public class MainActivity extends AppCompatActivity implements
    private ConstraintLayout view; // 画面全体のview
    private float downX; // 画面を触ったX軸位置
    private float downY; // 画面を触ったY軸位置
    private float upX; // 画面から離れたX軸位置
    private float upY; // 画面から離れたY軸位置
    private final int UP_FLICK = 0; // 上方向
    private final int RIGHT_FLICK = 1; // 右方向
    private final int DOWN_FLICK = 2; // 下方向
    private final int LEFT_FLICK = 3; // 左方向
    private int myFlick; // フリックした方向の格納用
```

フリック方向を計算するためのメソッドを定義していきましょう。

```
// フリック方向の計算
private int getFlickVector(){
    float buffer = 200; // フリックの遊び領域
    int flickVector = -1; // フリックの方向

    return flickVector;
}
```

まずは左右方向の確認を行います。

平行に左右へフリックするとは限らない為、ある程度上下にぶれても判定できるようにします。

上下の判定は Math.abs() を活用し、2つの値の差を取得します。

```
// フリック方向の計算
private int getFlickVector(){
    float buffer = 200; // フリックの遊び領域
    int flickVector = -1; // フリックの方向

    // 右方向にフリック かつ 縦幅がbuffer以下の時
    if ((downX < upX) && Math.abs(downY - upY) < buffer) {
        flickVector = RIGHT_FLICK;
    }
    // 左方向にフリック かつ 縦幅がbuffer以下の時
    }else if (upX < downX && Math.abs(downY - upY) < buffer) {
        flickVector = LEFT_FLICK;
    }

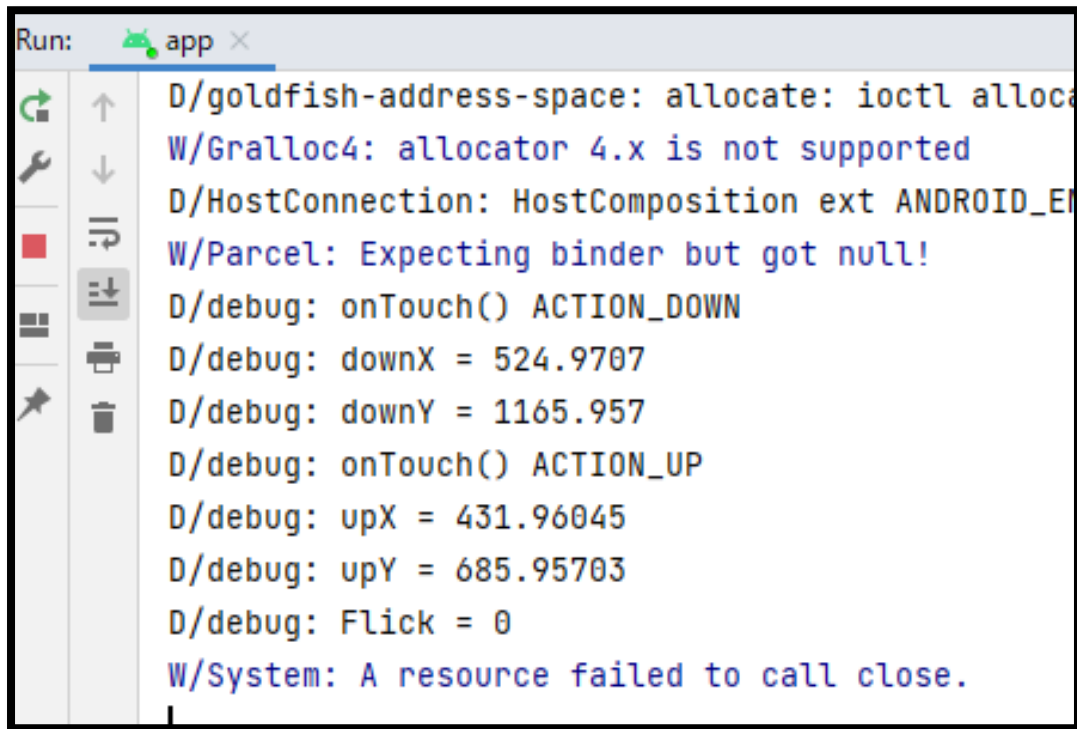
    return flickVector;
}
```

上下方向の判定も追加します。

```
// フリック方向の計算
private int getFlickVector(){
    float buffer = 200; // フリックの遊び領域
    int flickVector = -1; // フリックの方向
    // 右方向にフリック かつ 縦幅がbuffer以下の時
    if ((downX < upX) && Math.abs(downY - upY) < buffer) {
        flickVector = RIGHT_FLICK;
    }
    // 左方向にフリック かつ 縦幅がbuffer以下の時
    } else if (upX < downX && Math.abs(downY - upY) < buffer) {
        flickVector = LEFT_FLICK;
    }
    // 下方向にフリック
    } else if (downY < upY) {
        flickVector = DOWN_FLICK;
    }
    // 上方向にフリック
    } else if (upY < downY) {
        flickVector = UP_FLICK;
    }
    return flickVector;
}
```

方向を判別出来ているか確認の為にログを出力させてみましょう

```
public boolean onTouch(View view, MotionEvent motionEvent) {  
    // 操作の種類で分岐  
    switch (motionEvent.getAction()){  
        // 画面に触れた時  
        case MotionEvent.ACTION_DOWN:  
            downX = motionEvent.getX();    // タッチしたX軸位置を取得  
            downY = motionEvent.getY();    // タッチしたY軸位置を取得  
            Log.d( tag: "debug", msg: "onTouch() ACTION_DOWN");  
            Log.d( tag: "debug", msg: "downX = "+downX);  
            Log.d( tag: "debug", msg: "downY = "+downY);  
            break;  
        // 画面から離れた時  
        case MotionEvent.ACTION_UP:  
            upX = motionEvent.getX();    // 指を離れたX軸位置を取得  
            upY = motionEvent.getY();    // 指を離れたY軸位置を取得  
            Log.d( tag: "debug", msg: "onTouch() ACTION_UP");  
            Log.d( tag: "debug", msg: "upX = "+upX);  
            Log.d( tag: "debug", msg: "upY = "+upY);  
            Log.d( tag: "debug", msg: "Flick = "+getFlickVector());  
    }  
    // イベント処理終了:true 他イベント処理も行う:false  
    return true;  
}
```

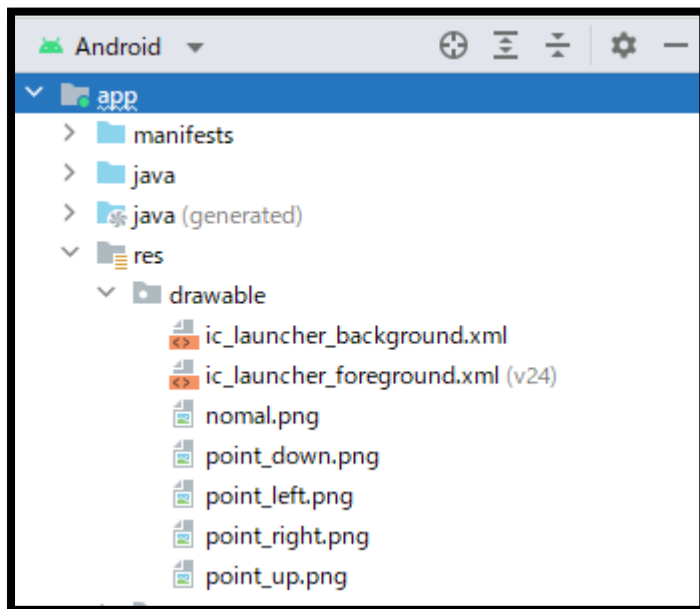


The screenshot shows the 'Run' window in Android Studio with the 'app' tab selected. The logcat output displays various system and application messages. On the left side of the window, there is a vertical toolbar with icons for running, debugging, and other development actions. The log messages include:

```
D/goldfish-address-space: allocate: ioctl allocat
W/Gralloc4: allocator 4.x is not supported
D/HostConnection: HostComposition ext ANDROID_E
W/Parcel: Expecting binder but got null!
D/debug: onTouch() ACTION_DOWN
D/debug: downX = 524.9707
D/debug: downY = 1165.957
D/debug: onTouch() ACTION_UP
D/debug: upX = 431.96045
D/debug: upY = 685.95703
D/debug: Flick = 0
W/System: A resource failed to call close.
```

方向も判定できるようになっていますね

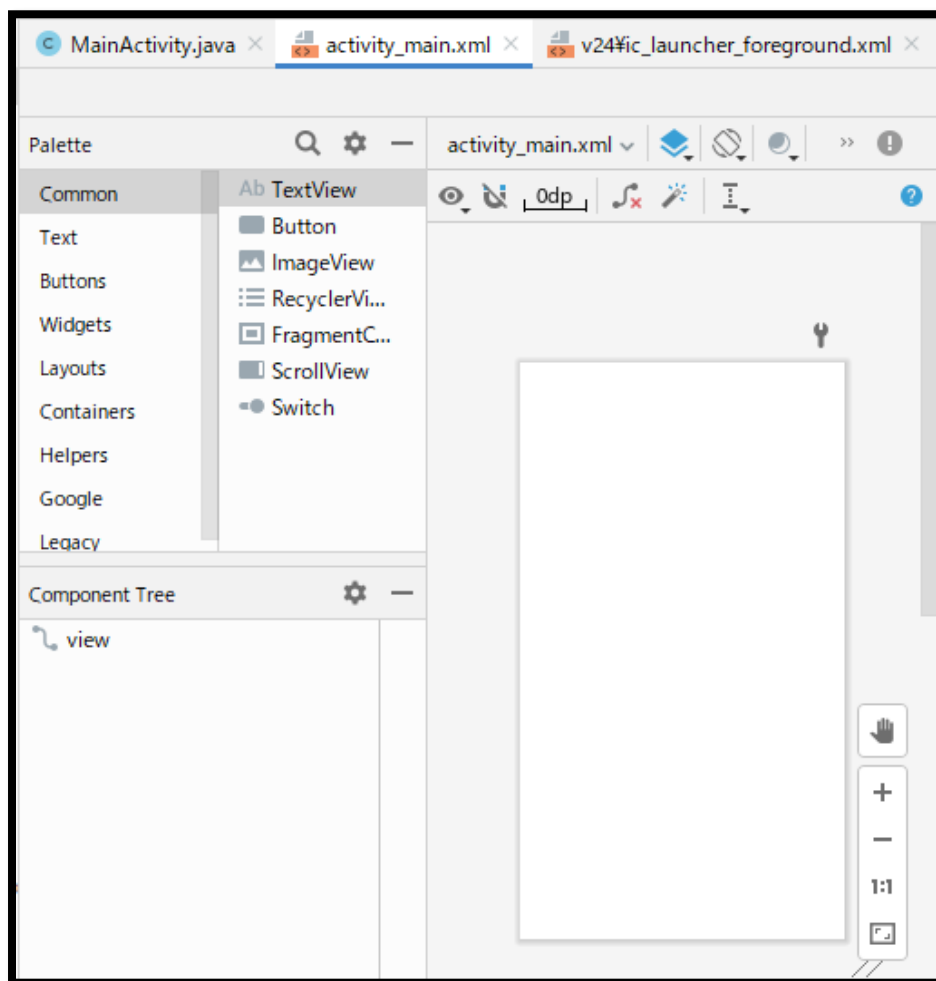
次に、CPU とバトル出来るように画像を追加していきましょう



## スマートフォンアプリ開発演習

### ■Activity\_main.xml

いったん textView を削除します

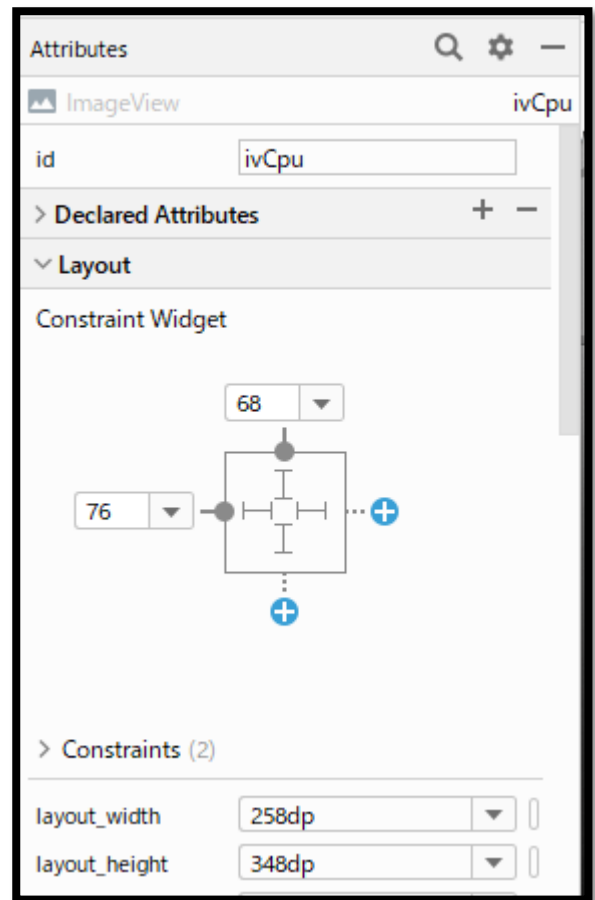
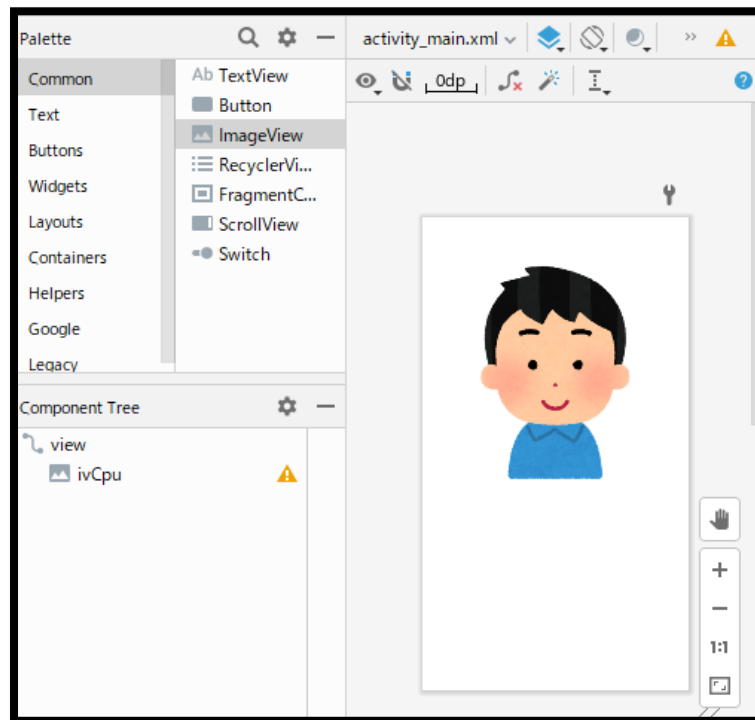


ImageView を配置しましょう

Id: ivCpu に変更

位置制約も忘れずに設定

## スマートフォンアプリ開発演習



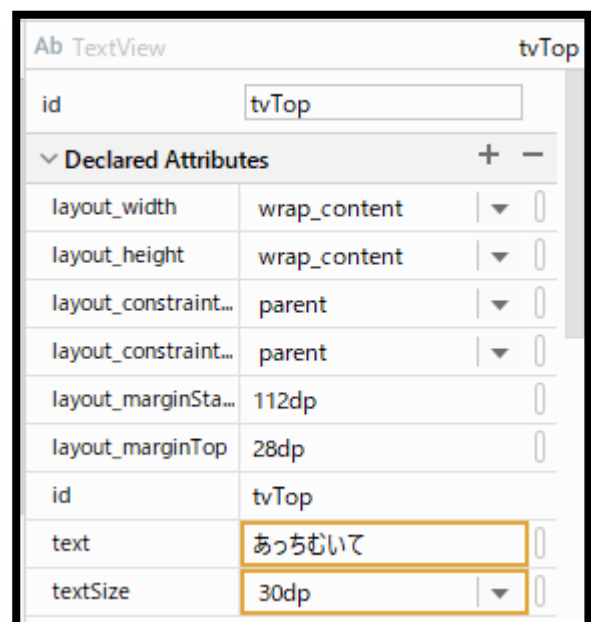
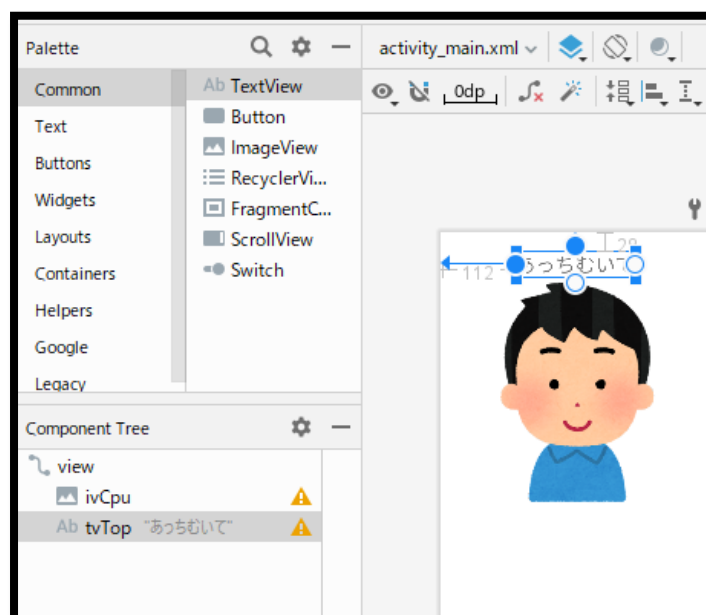
トップの文字用の TextView を配置

Id : tvTop

TextSize : 30dp

Text : あっちむいて

位置制約も忘れずに



## スマートフォンアプリ開発演習

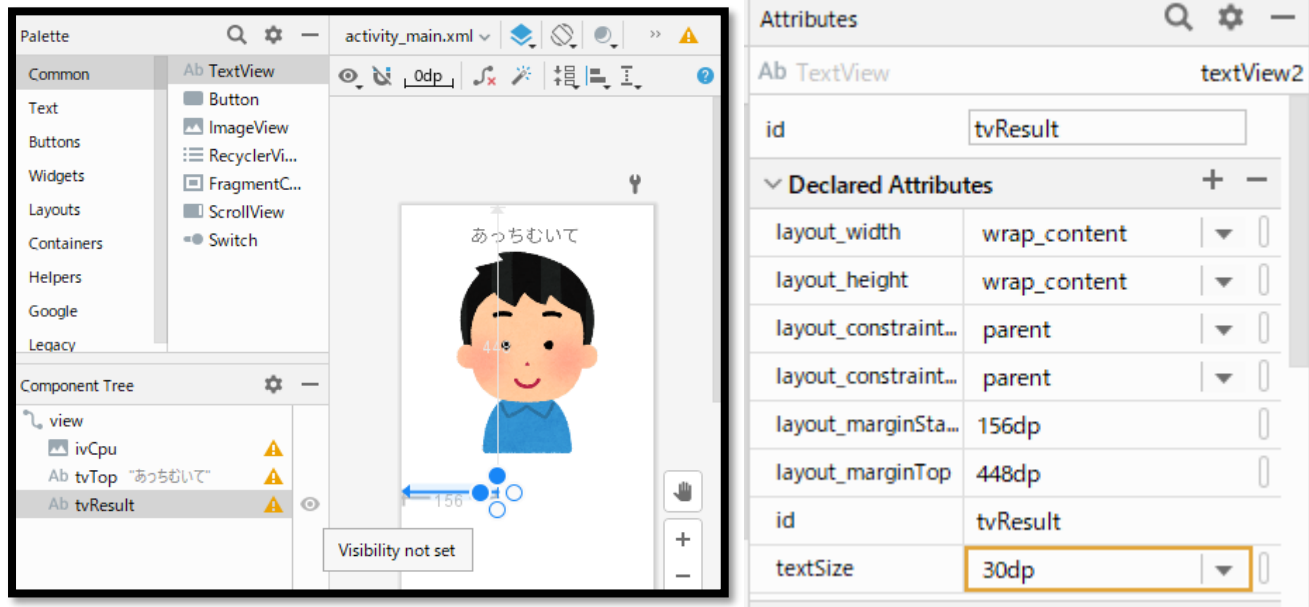
結果表示用の TextView も配置

Id : tvResult

TextSize : 30dp

Text :

位置制約も忘れずに



レイアウトの準備ができたので、

クリック後に画像と結果を表示するように処理を追加していきましょう



表示させる画像 id の配列を作成し取り出しやすくしておきましょう

```
public class MainActivity extends AppCompatActivity implements View.OnTouchListener{
    private ConstraintLayout view; // 画面全体のview
    private float downX; // 画面を触ったX軸位置
    private float downY; // 画面を触ったY軸位置
    private float upX; // 画面から離れたX軸位置
    private float upY; // 画面から離れたY軸位置
    private final int UP_FLICK = 0; // 上方向
    private final int RIGHT_FLICK = 1; // 右方向
    private final int DOWN_FLICK = 2; // 下方向
    private final int LEFT_FLICK = 3; // 左方向
    private int myFlick; // フリックした方向の格納用

    // CPUの画像配列
    private final int CPU_IMAGES[] = {R.drawable.point_up,R.drawable.point_right,
        R.drawable.point_down,R.drawable.point_left};

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

CPU の画像を差し替えれるように

ImageView の宣言と関連付けを行います

```
public class MainActivity extends AppCompatActivity implements View.OnTouchListener{
    private ConstraintLayout view; // 画面全体のview
    private float downX; // 画面を触ったX軸位置
    private float downY; // 画面を触ったY軸位置
    private float upX; // 画面から離れたX軸位置
    private float upY; // 画面から離れたY軸位置
    private final int UP_FLICK = 0; // 上方向
    private final int RIGHT_FLICK = 1; // 右方向
    private final int DOWN_FLICK = 2; // 下方向
    private final int LEFT_FLICK = 3; // 左方向
    private int myFlick; // フリックした方向の格納用

    // CPUの画像配列
    private final int CPU_IMAGES[] = {R.drawable.point_up,R.drawable.point_right,
        R.drawable.point_down,R.drawable.point_left};

    private ImageView ivCpu; // cpuの画像
}
```

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

    // Viewの関連付け
    view = findViewById(R.id.view);
    ivCpu = findViewById(R.id.ivCpu);

    // タッチ検知の登録
    view.setOnClickListener(this);
}
```

勝敗判定メソッドを作成し、CPU の内容によって画像を差し替えましょう。

```
// 勝敗処理メソッド
public void gudge(){
    // フリックした方向
    int myFlick = getFlickVector();
    // cpuの方向
    int cpuSelect = (int)(Math.random()*CPU_IMAGES.length);
    // cpuの画像を差し替え
    ivCpu.setImageResource(CPU_IMAGES[cpuSelect]);
    // 一致していれば勝ち(true)
    boolean result =(myFlick == cpuSelect);
}
```

しかし、このままでは勝敗がわからないので、TextView に結果を表示するよう処理を追加します。

## スマートフォンアプリ開発演習

### TextView の宣言

```
public class MainActivity extends AppCompatActivity implements View.OnTouchListener{
    private ConstraintLayout view; // 画面全体のview
    private float downX; // 画面を触ったX軸位置
    private float downY; // 画面を触ったY軸位置
    private float upX; // 画面から離れたX軸位置
    private float upY; // 画面から離れたY軸位置
    private final int UP_FLICK = 0; // 上方向
    private final int RIGHT_FLICK = 1; // 右方向
    private final int DOWN_FLICK = 2; // 下方向
    private final int LEFT_FLICK = 3; // 左方向
    private int myFlick; // フリックした方向の格納用
    // CPUの画像配列
    private final int CPU_IMAGES[] = {R.drawable.point_up,R.drawable.point_right,
        R.drawable.point_down,R.drawable.point_left};
    private ImageView ivCpu; // cpuの画像
    private TextView tvResult; // 結果表示用
```

### View との関連付け

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

    // Viewの関連付け
    view = findViewById(R.id.view);
    ivCpu = findViewById(R.id.ivCpu);
    tvResult = findViewById(R.id.tvResult);

    // タッチ検知の登録
    view.setOnTouchListener(this);
}
```

結果を TextView に表示させます。

```
// 勝敗処理メソッド
public void gudge(){
    // フリックした方向
    int myFlick = getFlickVector();
    // cpuの方向
    int cpuSelect = (int)(Math.random()*CPU_IMAGES.length);
    // cpuの画像を差し替え
    ivCpu.setImageResource(CPU_IMAGES[cpuSelect]);
    // 一致していれば勝ち(true)
    boolean result =(myFlick == cpuSelect);
    // 結果を表示
    tvResult.setText(result?"あなたの勝ち":"引き分け");
}
```

最後に勝敗勝利をどのタイミングで行うかですが、  
フリックして、指を離れたタイミングで行うようにしてみましょう

```
@Override
public boolean onTouch(View view, MotionEvent motionEvent) {
    // 操作の種類で分岐
    switch (motionEvent.getAction()){
        // 画面に触れた時
        case MotionEvent.ACTION_DOWN:
            downX = motionEvent.getX(); // タッチしたX軸位置を取得
            downY = motionEvent.getY(); // タッチしたY軸位置を取得
            Log.d( tag: "debug", msg: "onTouch() ACTION_DOWN");
            Log.d( tag: "debug", msg: "downX = "+downX);
            Log.d( tag: "debug", msg: "downY = "+downY);
            break;
        // 画面から離れた時
        case MotionEvent.ACTION_UP:
            upX = motionEvent.getX(); // 指を離れたX軸位置を取得
            upY = motionEvent.getY(); // 指を離れたY軸位置を取得
            Log.d( tag: "debug", msg: "onTouch() ACTION_UP");
            Log.d( tag: "debug", msg: "upX = "+upX);
            Log.d( tag: "debug", msg: "upY = "+upY);
            Log.d( tag: "debug", msg: "Flick = "+getFlickVector());
            gudge();
    }
}
```

## スマートフォンアプリ開発演習

