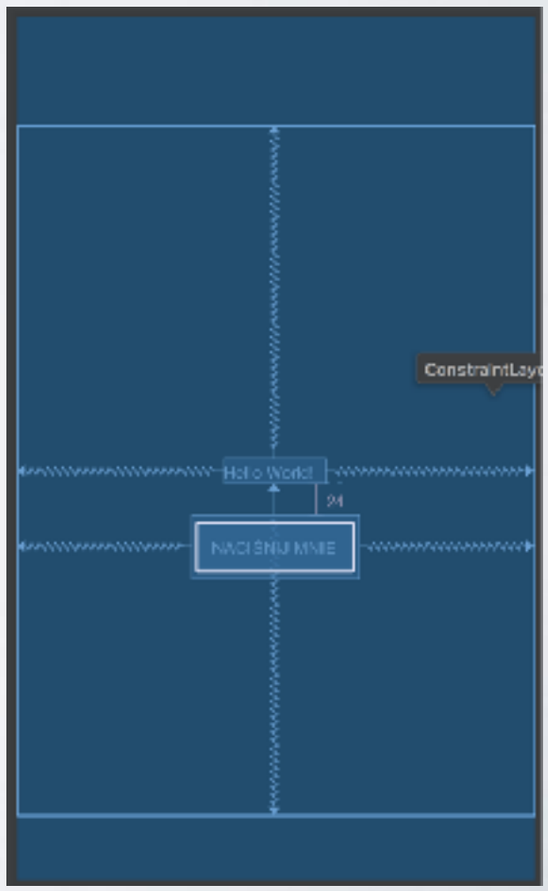
WSTĘP DO ANDROIDA

Laboratorium 2

Systemy i aplikacje bez granic

I

- Utworzyć nowy projekt typu Empty Activity
- Pod napisem Hello World dodać przycisk Button
- Zmienić id TextView na statusText
- i przycisku na pressMeButton



- W klasie widoku musimy dodać import na projekt

```
import kotlinx.android.synthetic.main.activity_main.*
```

- oraz obsługę zdarzenia

```
pressMeButton.setOnClickListener {
    statusText.text = "Naciśnięto"
}
```

- Uruchomić

- Możemy jeszcze dodać

```
pressMeButton.setOnLongClickListener {
    statusText.text = "Długie wciśnięcie"
    true
}
```

II

- Utworzyć nowy projekt typu EmptyActivity o nazwie Dotyk

- W aktywności umieścić dwie kontrolki TextView nad sobą, jako textView1 i textView2

II

- Zmienić id ConstraintLayout na myLayout
- Dodać następujące importy

```
import android.view.MotionEvent
import android.view.View
import kotlinx.android.synthetic.main.activity_main.*
```

II

- Na koniec metody onCreate dodać następujący kod:

```
myLayout.setOnTouchListener { v: View,m: MotionEvent ->

    handleTouch(m)
     ^setOnTouchListener true
}
```

```
private fun handleTouch(m: MotionEvent)
{
    val pointerCount = m.pointerCount

    for (i in 0 until pointerCount)
    {
        val x = m.getX(i)
        val y = m.getY(i)
        val id = m.getPointerId(i)
        val action = m.actionMasked
        val actionIndex = m.actionIndex
        var actionString: String

        when (action)
        {
            MotionEvent.ACTION_DOWN -> actionString = "Dół"
            MotionEvent.ACTION_UP -> actionString = "Góra"
            MotionEvent.ACTION_POINTER_DOWN -> actionString = "Wskaźnik dół"
            MotionEvent.ACTION_POINTER_UP -> actionString = "Wskaźnik góra"
            MotionEvent.ACTION_MOVE -> actionString = "Ruch"
            else -> actionString = ""
        }

        val touchStatus =
                "Akcja: $actionString Współrzędne: $actionIndex ID: $id X: $x Y: $y"

        if (id == 0)
            textView1.text = touchStatus
        else
            textView2.text = touchStatus
    }
}
```

III

- Tworzymy nowy projekt typu Empty Activity
- Zmienić id Text View na statusText
- Dodać importy

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.GestureDetector
import android.view.MotionEvent
import kotlinx.android.synthetic.main.activity_main.*
import android.support.v4.view.GestureDetectorCompat
```

III

- Klasa oprócz dziedziczenie musi implementować interfejsy (będzie odpowiedzialna za obsługę gestów)

```
class MainActivity : AppCompatActivity(),
        GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener {
```

- Dodajemy do klasy pole

```
var gDetector: GestureDetectorCompat? = null
```

III

- W onCreate dopisujemy

```
this.gDetector = GestureDetectorCompat( context: this,  listener: this)
gDetector?.setOnDoubleTapListener(this)
```

- Przeciążamy metodę onTouchEvent

```
override fun onTouchEvent(event: MotionEvent): Boolean {
    this.gDetector?.onTouchEvent(event)
    // Be sure to call the superclass implementation
    return super.onTouchEvent(event)
}
```

III

- Dodajemy metody obsługi gestów

```
override fun onDown(event: MotionEvent): Boolean {
    statusText.text = "onDown"
    return true
}

override fun onFling(event1: MotionEvent, event2: MotionEvent,
                     velocityX: Float, velocityY: Float): Boolean {
    statusText.text = "onFling"
    return true
}

override fun onLongPress(event: MotionEvent) {
    statusText.text = "onLongPress"
}

override fun onScroll(e1: MotionEvent, e2: MotionEvent,
                      distanceX: Float, distanceY: Float): Boolean {
    statusText.text = "onScroll"
    return true
}
```

|||

```
override fun onShowPress(event: MotionEvent) {
    statusText.text = "onShowPress"
}

override fun onSingleTapUp(event: MotionEvent): Boolean {
    statusText.text = "onSingleTapUp"
    return true
}

override fun onDoubleTap(event: MotionEvent): Boolean {
    statusText.text = "onDoubleTap"
    return true
}

override fun onDoubleTapEvent(event: MotionEvent): Boolean {
    statusText.text = "onDoubleTapEvent"
    return true
}

override fun onSingleTapConfirmed(event: MotionEvent): Boolean {
    statusText.text = "onSingleTapConfirmed"
    return true
}
```

IV

- Wracamy do kodu z przykładu I
- W metodzie obsługi klawisza umieścić kod

i uruchomić

```
var i = 0
while (i <= 20) {
    try {
        Thread.sleep( millis: 1000)
        i++
    } catch (e: Exception) {
    }

}
myTextView.text = "Skończyłem"
```

IV

- Uaktualnić importy:

```
import android.os.AsyncTask
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.util.Log
import kotlinx.android.synthetic.main.activity_main.*
```

- Dodać 2 pola w klasie aktywności:

```
private var task=MyTask()
private var running=false
```

IV

- Dodać wewnętrzną klasę do aktywności

```
private inner class MyTask : AsyncTask<String, Int, String>() {

    override fun onPreExecute() {
        running=true
        myTextView.text="Zaczynam"
        pressMe.text="Anuluj"
        Log.i( tag: "As", msg: "PreExecute")
    }
```

```
override fun doInBackground(vararg params: String): String {
    Log.i( tag: "As", msg: "Zaczynam")
    var i = 0
    while (i <= 20) {
        try {
            Thread.sleep( millis: 1000)
            if (isCancelled())
            {
                Log.i( tag: "As", msg: "Przerywam")
                return "Anulowałeś mnie"
            }
            publishProgress(i)
            i++
        }
        catch (e: Exception) {
            return(e.localizedMessage)
        }
    }
    Log.i( tag: "As", msg: "Kończę")
    return "Skończyłem"
}
```

```
    override fun onProgressUpdate(vararg values: Int?) {
        super.onProgressUpdate(*values)
        val counter = values.get(0)
        myTextView.text = "Licznik = $counter"
        Log.i( tag: "As", msg: "Update")
    }

    override fun onPostExecute(result: String) {
        myTextView.text = result
        running=false
        pressMe.text = "Naciśnij mnie"
        Log.i( tag: "As", msg: "Skończyłem")
    }

    override fun onCancelled(result: String?) {
        Log.i( tag: "As", msg: "Zaczynam anulować")
        super.onCancelled(result)
        pressMe.text = "Naciśnij mnie"
        myTextView.text = "Anulowane"
        Log.i( tag: "As", msg: "Anulowane")
        running=false
    }
}
```

IV

- Zmienić metodę obsługi przycisku

```
fun buttonClick(view: View) {
    if (running) {
        Log.i( tag: "As",  msg: "Guzik anuluje")
        task.cancel( mayInterruptIfRunning: false)
    }
    else {
        Log.i( tag: "As", msg: "Uruchamiam")
        task=MyTask()
        task.execute()
    }

}
```