

# Hitparáda aktivít

Aktivity, View  
Intent, Layout



Peter Borovanský  
KAI, I-18

borovan 'at' ii.fmph.uniba.sk



# Odovzdávanie riešení zadaní

---

- **Odovzdávajte .zip, ktorý v koreni obsahuje**
- **- README s popisom ovládania, resp. vami implementované detaily nespomenuté v zadaní,**
- **- projekt AS s vymazaným build adresárom,**
- **- vytiahnutým .apk súborom do koreňa .zipu, aby ho opravujúci nemuseli loviť v útrobách vášho projektu**
- **Za nedodržanie tohoto formátu sa strhávajú body, ktoré sú úmerné času cvičiaceho, aby**
- **- zo zdrojáku pochopil ovládanie vašej apky (ak ste zabudli README),**
- **- kompilovaním vášho projektu, ak ste neposkytli .apk**
- **Code review robíme pre vás, tak nám pomôžte. Ďakujeme za pochopenie.**

# Príklad jednoduchéj aplikácie

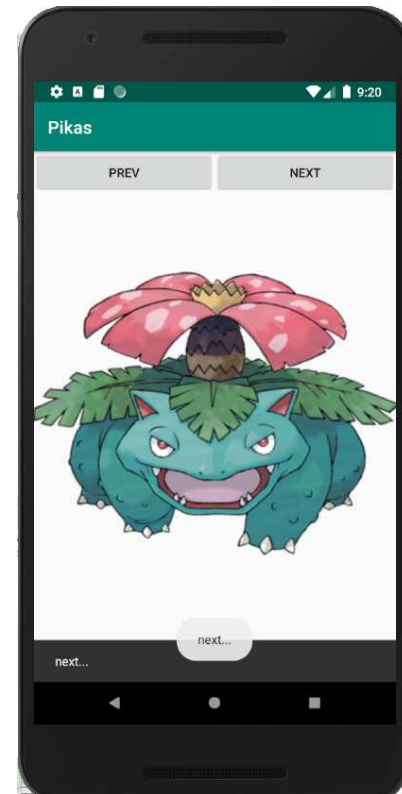
(RECAP predošlej prednášky)

## Ilustrovali sme:

- príklad návrhu (*vyklikania*) jednoduchého GUI (single activity app)
- logovanie udalostí ako efektívny prostriedok ladenia pomocou
  - `Log.d(...)`
  - `Toast.make(...)`
  - `Snackbar.make(...)`
- používanie Image/Vector Asset (drawable/mipmap)
- používanie resource editora (pri definovaní strings.xml)
- používanie layout editora pri tvorbe rozhrania (ešte bude)
- eventhandler (`.setOnClickListener`) previazané cez
  - `findViewById<Button> (R.id. quitBtn)`
  - `prevBtn.setOnClickListener { }`
  - `property android:onClick="nextOnClickListener"`

## Nestihli sme:

- aktivitu a jej životný cyklus
- previazanie View Binding





# Logovanie

(RECAP 3.prednášky)

Tri najbežnejšie spôsoby:

- Log – loguje do okna Logcat, filtrujte podľa **TAGu** metódy `Log.d(TAG,`
  - definujte si **TAG** ako konštantu
- Toast – potrebuje **Context** (zjednodušená aktivita, v ktorej sa toastuje)
  - nezabudnite na volanie `.show()`
- Snackbar – umožňuje reagovať akciou, potrebuje View ako prvý argument

```
prevBtn2.setOnClickListener({  
    → Toast.makeText(this, "prev...", Toast.LENGTH_SHORT).show()  
    → Log.d(TAG, "prev...")  
    → Snackbar.make(it, "next...", Snackbar.LENGTH_SHORT)  
        .setAction("Action", null).show()  
    alebo  
        .setAction(R.string.action,  
            View.OnClickListener { nextOnClickListener(it) }) .show()  
    ...  
})
```

# Pikas

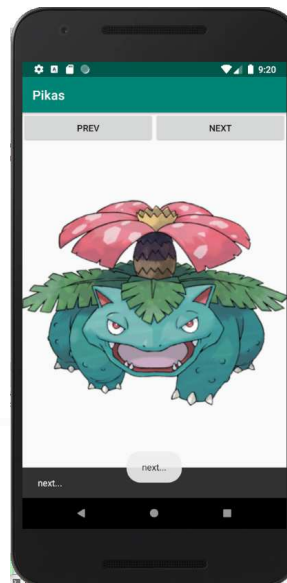
(RECAP 3.prednášky)

activity entry point

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
    var i = 0  
    var imgs = arrayOf(  
        ContextCompat.getDrawable(applicationContext,  
            R.drawable.butterfree),  
        ...  
    )  
    imageView2.setImageDrawable(imgs[i])  
    prevBtn2.setOnClickListener({  
        Toast.makeText(this, "prev...", Toast.LENGTH_SHORT).show()  
        if (--i < 0) i += imgs.size  
        imageView2.setImageDrawable(imgs[i])  
    })  
    nextBtn2.setOnClickListener({  
        Toast.makeText(this, "next...", Toast.LENGTH_LONG).show()  
        i = (++i)%imgs.size  
        imageView2.setImageDrawable(imgs[i])  
    })  
}
```

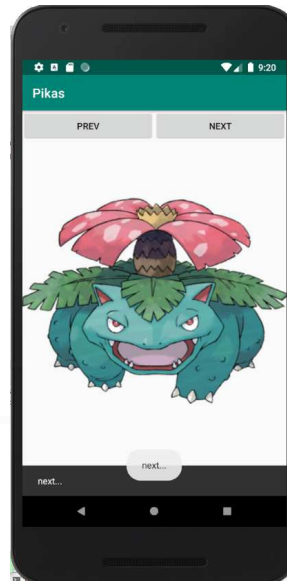
View(s)

logovanie



# Pikas

(stav sa mieša s views a logikou – riešenie pride MVVM)



```
const final val TAG = "PIKAS"
var i = 0
var imgs = arrayOf<Drawable?>() }
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    imgs = arrayOf(ContextCompat.getDrawable(applicationContext,
                                                R.drawable.butterfree), ... )

    imageView2.setImageDrawable(imgs[i])
    prevBtn2.setOnClickListener { // it:View -> { ... }
        if (--i < 0) i += imgs.size
        imageView2.setImageDrawable(imgs[i])
    }
}

// prepojene cez property android:onClick="nextOnClickListener"
fun nextOnClickListener(v: View) {
    i = (++i) % imgs.size
    imageView2.setImageDrawable(imgs[i])
}
```

State

## ▼ Common Attributes

|         |                |   |
|---------|----------------|---|
| style   | @style/mystyle | ▼ |
| onClick | clickOnNext    | ▼ |

# View Binding

- findViewById() as Button, findViewById<Button>() - klasické, „javish“ riešenie
- syntetic – kotlin-android-extensions plugin – deprecated od 2020
- ďalší spôsob prepojenia komponentov (View) z .xml layoutu s kódom
- **pozor:** nepliet' si to s Data Binding, to príde s JetPack library, to je zložitejšie

1) do build.gradle pridajte pod

```
android {  
    buildFeatures {  
        viewBinding = true  
    }  
}
```

2) v samotnej Activity NAHRADÍTE

```
setContentView(R.layout.activity_main)
```

za

```
val binding = ActivityMainBinding.inflate(layoutInflater)  
setContentView(binding.root)
```

3) miesto referencie nejakého View, napr. `imageView2`, použijete `binding.imageView2`

4) ak mimo metódy onCreateView potrebujete premennú `binding`, urobte ju `lateinit` `var`

```
lateinit var binding : ActivityMainBinding
```

5) ak sa vaša aktivita nevolá MainA..., tak nahrad'te zelené za jej meno

6) objavte, čo je `apply`, resp. iné scoping functions

```
android {  
    buildFeatures {  
        viewBinding = true  
    }  
    compileSdkVersion 30  
    defaultConfig {  
        applicationId "com.example.pikas"  
        minSdkVersion 23  
        targetSdkVersion 30  
    }  
}
```



# View Binding

příklad apply

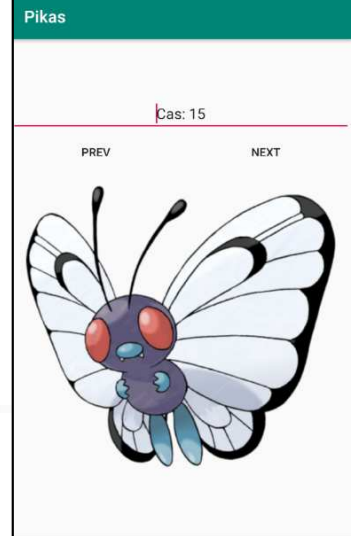
```
binding.imageView2.setImageDrawable(imgs[i])
binding.prevBtn2.setOnClickListener {
    Toast.makeText(this,
        "prev...",
        Toast.LENGTH_SHORT
    ).show()
    if (--i < 0) i += imgs.size
    binding.imageView2.setImageDrawable(imgs[i])
}
```

```
binding.apply {
    imageView2.setImageDrawable(imgs[i])
    prevBtn2.setOnClickListener {
        Toast.makeText(this@MainActivity,
            "prev...",
            Toast.LENGTH_SHORT
        ).show()
        if (--i < 0) i += imgs.size
        imageView2.setImageDrawable(imgs[i])
    }
}
```



# Pikas

(asynchrónnosť - timer)



pomocou `java.util.Timer`

```
Timer("tik-tak").schedule(1000, 1000) { // delay, period
    Log.d(TAG, "onTICK")
    cas++
    runOnUiThread {binding.time.setText("Cas: $cas")}
}.run()
```

- nezabudnite na `.run()`
- `runOnUiThread`
  - má argument `java.lang.Runnable`, ktorý vykoná v hlavnom GUI vlákne

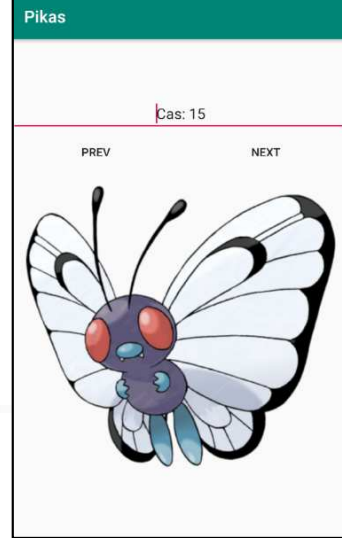
zabitie timera:

```
override fun onPause() {
    super.onPause()
    timer.cancel()
}
```

# Pikas

(asynchrónnosť – count down)

`pomocou android.os.CountDownTimer`



```
object:CountDownTimer(20000, 1000) { // 20sek, tik po 1sek  
    // how long, period
```

tik

```
    override fun onTick(millisUntilFinished: Long) {  
        Log.d(TAG, "onTICK")  
        runOnUiThread {  
            time.setText("Cas: ${millisUntilFinished/1000}") }  
        }  
    }
```

game  
over

```
    override fun onFinish() {  
        Log.d(TAG, "onFinish")  
        finish()  
        exitProcess(0)  
    }
```

```
}.start()
```

ukončenie appky



# Handler

Na prenos dát z background vlákna do UI vlákna možno použiť Handle, ktorý beží a akcie vykoná v GUI vlákne

```
val handler = Handler(Looper.getMainLooper())

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val runnable = Runnable {
        updateSomeGUI() // some actions
    }
    ...
    handler.postDelayed(runnable, 5000) // +milis

    val now = SystemClock.uptimeMillis()
    val uptime = now + (15*1000-now % (15*1000)) // 15 sec
    handler.postAtTime(runnable, uptime)

    handler.post { Runnable { updateSomeGUI() } }
}
```



# Handler

## prenos parametrov

Na prenos dát z background vlákna do UI vlákna možno použiť Handler, ktorý beží a akcie vykoná v GUI vlákne

```
val handler = Handler()
val handler2 = object : Handler() {
    override fun handleMessage(msg: Message) {
        super.handleMessage(msg)
        Log.d("HANDLER", "msg = ${msg.arg1}")
        Log.d("HANDLER", "msg = ${msg.data.getString("time")}")
    }
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

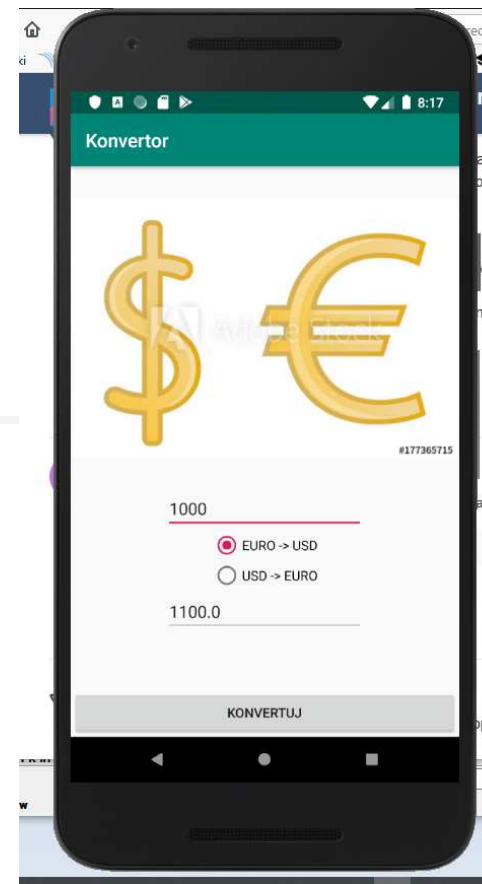
    val runnable = Runnable {
        updateSomeGUI()           // some actions
        val msg = Message()
        msg.arg1 = 1               // arg1, arg2 : Int
        val bundle = Bundle()     // pre všetko zložitejšie slúži bundle = dictionary
        val dateFormat = SimpleDateFormat("HH:mm:ss MM/dd/yyyy", Locale.US)
        bundle.putString("time", dateFormat.format(Date()))
        msg.data = bundle
        handler2.sendMessage(msg)
    }

    handler.postDelayed(runnable, 5000) // millis
    handler.post { Runnable { updateSomeGUI() } }
}

private fun updateSomeGUI() {    ... }
```

# Konvertor EURO USD

(logika)



Jednoduchá aplikácia na konverziu kurzov USD EURO

- s modifikovateľným TextView pre zadanie sumy, reálneho čísla
- RadioButtonom pre výber smeru konverzie
- s nemodifikovateľným poľom pre výsledok
- Button Konvertuj pre vykonanie akcie

```
override fun onCreate(savedInstanceState: Bundle?)  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
    convertBtn.setOnClickListener({  
        Toast.makeText(this, "convert", Toast.LENGTH_SHORT).show();  
        if (inputText.text.isNotEmpty()) {  
            val input = inputText.text.toString().toFloat();  
            var output = input  
            val exchangeRate = 0.96f  
            if (eur2usd.isChecked) output = exchangeRate * output  
            if (usd2eur.isChecked) output = output / exchangeRate  
            outputText.setText("$output") // set
```

Klik na Konvertuj

Konvertor.zip

# Konvertor EURO USD

(setOnClickListener)

*// very old fashion*

```
val cBtn = findViewById<Button>(R.id.convertBtn)
cBtn.setOnClickListener( { v -> convert(v) } )
cBtn.setOnClickListener { convert(it) }
```

*// old fashion*

```
convertBtn.setOnClickListener { v -> convert(v) }
convertBtn.setOnClickListener { convert(it) }
```

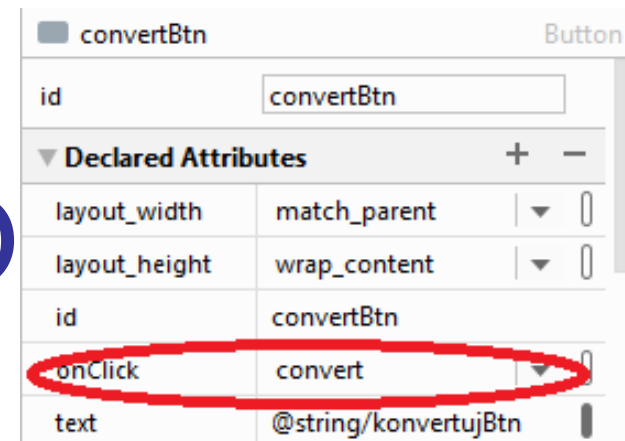
```
fun convert(v: View) {
```

```
    Toast.makeText(this, "convert", Toast.LENGTH_SHORT).show()
    if (inputText.text.isNotEmpty()) {
        val input = inputText.text.toString().toFloat()
        var output = input
        val exchangeRate = 0.96f
        if (eur2usd.isChecked) output = exchangeRate * output
        if (usd2eur.isChecked) output = output / exchangeRate
        outputText.setText("${output.format(2)}")
    }
```

```
fun Float.format(digits: Int) =
    java.lang.String.format("%.${digits}f", this)
```

extension  
metóda  
Float



convertBtn

Button

id

convertBtn

▼ Declared Attributes

+

-

layout\_width

match\_parent

▼

0

layout\_height

wrap\_content

▼

0

id

convertBtn

onClick

convert

▼

0

text

@string/konvertujBtn

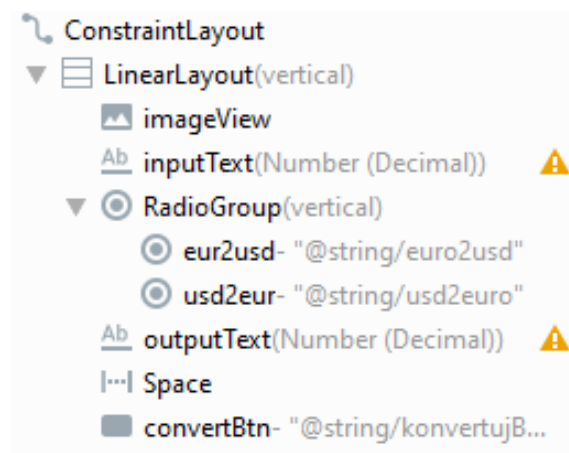
0

# Konvertor EURO USD

(layout)

```
<LinearLayout
    <ImageView .../>
    <EditText .../>
    <RadioGroup
        <RadioButton .../>
        <RadioButton .../>
    </RadioGroup>
    <EditText .../>
    <Space .../>
    <Button .../>
</LinearLayout>
```

Už pomerne jednoduchý návrh produkuje „košaté“, vnorené návrhové štruktúry – riešenie: ConstraintLayout



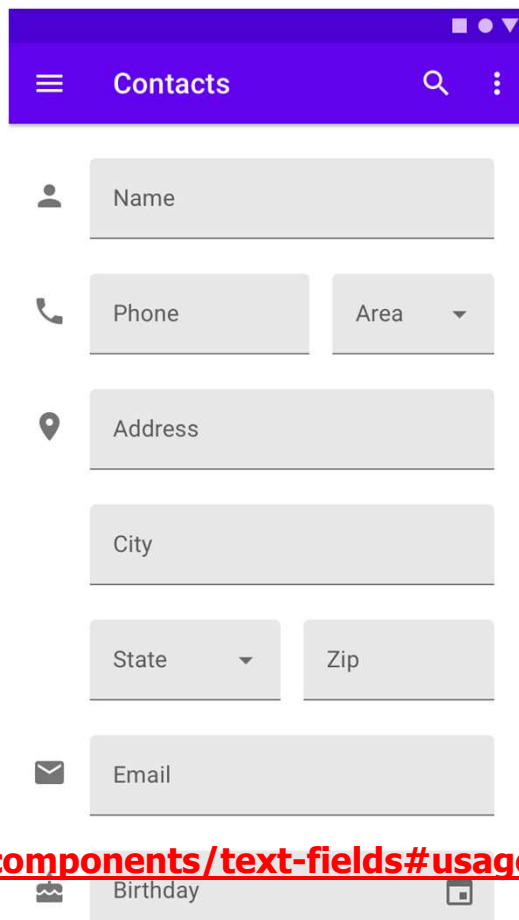
# Text Fields

## prvý dotyk s Material Design

Material Design je Google knižnica GUI komponentov unifikovaná pre Android, iOS, Flutter, web, ...

```
dependencies {  
  implementation 'com.google.android.material:material:1.4.0'
```

- zahŕňa Button, Text fields, SnackBars, Sliders, a mnoho ďalších vizuálnych komponentov Views



Contacts

Name

Phone Area

Address

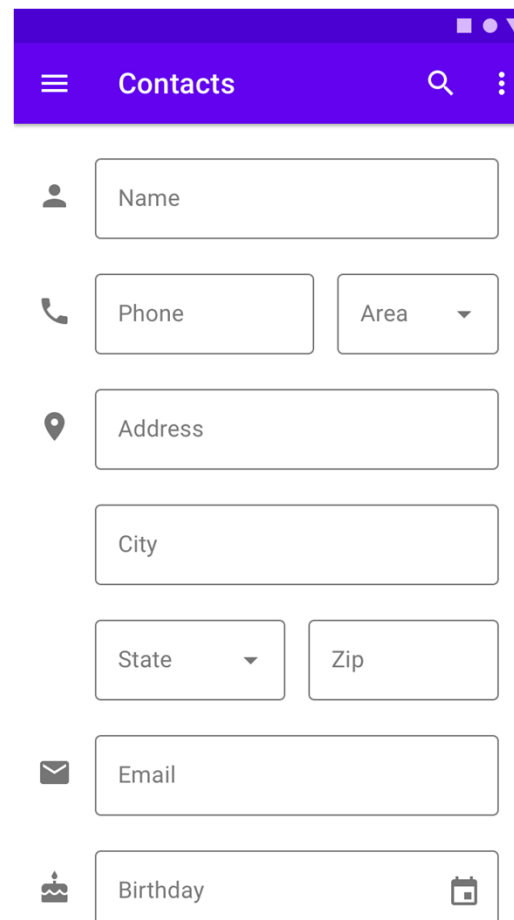
City

State Zip

Email

Birthday

The image shows a mobile app interface for 'Contacts' using the Material Design style. It features a purple header bar with a hamburger menu, the title 'Contacts', a search icon, and a three-dot menu icon. Below the header, there are several text input fields with icons to their left: a person icon for 'Name', a phone icon for 'Phone', a location pin for 'Address', an envelope for 'Email', and a birthday cake for 'Birthday'. The 'Phone' field is split into 'Phone' and 'Area' with a dropdown arrow. The 'City' field is a single line. The 'State' and 'Zip' fields are split with dropdown arrows. The 'Email' field is a single line. The 'Birthday' field is a single line with a calendar icon on the right.



Contacts

Name

Phone Area

Address

City

State Zip

Email

Birthday

The image shows the same mobile app interface as the left, but using the default Android Studio style. The header bar is purple with the same icons. The text input fields are white with a thin border and no icons to their left. The 'Phone' field is split into 'Phone' and 'Area' with a dropdown arrow. The 'City' field is a single line. The 'State' and 'Zip' fields are split with dropdown arrows. The 'Email' field is a single line. The 'Birthday' field is a single line with a calendar icon on the right.



# TextInput[Layout/EditText]

```
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:startIconDrawable="@drawable/ic_launcher_foreground"
    app:startIconContentDescription="@string/iconDescription"
    app:startIconCheckable="true"
    app:endIconMode="clear_text"
    app:counterEnabled="true"
    app:counterMaxLength="15"
    app:errorEnabled="true">
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/userTV"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/userHint"
        android:maxLength="15"
        android:inputType="textPersonName" />
</com.google.android.material.textfield.TextInputLayout>
```

<https://material.io/components/text-fields#usage>





# TextWatcher

```
val textWatcher = object : TextWatcher {    // singleton
    override fun beforeTextChanged(s: CharSequence, ...) { }
    override fun afterTextChanged(s: Editable?) { }
    override fun onTextChanged(s: CharSequence?, ...) {
        button.isEnabled =
            emailTV.text?.isEmpty()?:false &&
            userTV.text?.isEmpty()?:false &&
            passwordTV.text?.isEmpty()?:false
        button.isEnabled =
            if (emailTV.text != null && userTV.text != null &&
                passwordTV.text != null)
                emailTV.text!!.isEmpty() &&
                userTV.text!!.isEmpty() &&
                passwordTV.text!!.isEmpty()
            else
                false
    }
}

emailTV.addTextChangedListener(textWatcher)
userTV.addTextChangedListener(textWatcher)
passwordTV.addTextChangedListener(textWatcher)
```

# Kotlin – pokračovanie

Cheat sheets

- <https://www.programming-idioms.org/cheatsheet/Kotlin>
- <https://github.com/vmandro/Prednasky/tree/master/Kotlin>

## The billion-dollar mistake

I call it my billion-dollar mistake. It was the invention of the **null** reference in 1965...This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

Kotlin Null Safety

Sir Tony Hoare

FRS FREng



Tony Hoare in 2011

|                    |  |
|--------------------|--|
| <b>Born</b>        | Charles Antony Richard Hoare<br>11 January 1934 (age 85)<br>Colombo, British Ceylon  |
| <b>Residence</b>   | Cambridge  |
| <b>Other names</b> | C. A. R. Hoare   |
| <b>Alma mater</b>  | University of Oxford (BA)<br>Moscow State University   |
| <b>Known for</b>   | Quicksort<br>Quickselect<br>Hoare logic<br>Null reference<br>Communicating Sequential Processes<br>Structured programming  |
| <b>Awards</b>      | Turing Award (1980)<br>Harry H. Goode Memorial Award (1981)<br>Faraday Medal (1985)<br>Computer Pioneer Award (1990)<br>Kyoto Prize (2000)<br>IEEE John von Neumann Medal (2011) |

# Nullables

To, čo je

- Optional v Java, resp.
- Option v Scala, resp. kdekade iné inde

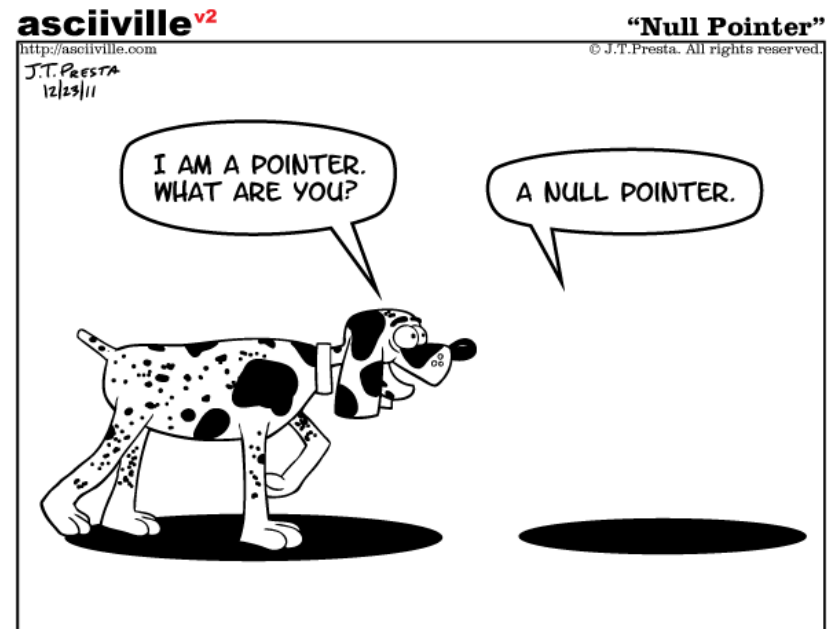
Napr. `String?` je typ pre reťazec alebo null

Ale `String` je typ len pre SKUTOČNÝ REĹAZEC, not-null

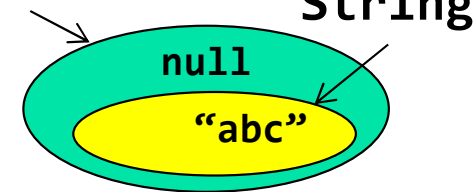
Preto `a:String?` nemôžete priradiť do `b:String`, lebo čo, ak by `a == null`

Ak ste skalo-pevne presvedčený, že hodnota `a:String?` `!= null`, môžete opatrne použiť BANG-BANG (`!!`) operátor a oklamať type-checker  
`val b:String = a!!`

Ak ale neviete, či `a:String?` `== null`, tak použijete tzv. **Elvis operátor**  
`val c:String = a?:"default, ak je prázdny reťazec"`



`String?`






# Nullables

(ďalšie operátory na konverziu medzi type a type?)



- 
- Elvis operátor  
`obj?:default` = if (obj == null) default else obj
  - Safe call operátor (Elvis na Žižku)  
`obj?.m()` = if (obj == null) null else obj.m()
  - Not-null assertion (bang-bang !!)  
`obj!!` = if (obj != null) obj else N.P.E. – null pointer Ex.
  - Safe cast  
`obj as? T` = if (obj typeof T) obj else null  
`obj as T` = if (!obj typeof T) cast exception
  - let  
`obj?.let {...it...}` = if (obj != null) {...it <- obj...}

# Nullables

(ešte raz, podrobnejšie)



NULL-SAFETY

V Jave je typ `String` skutočný reťazec alebo `null`

V Kotlini `String` je **LEN skutočný reťazec** a `null` nepatrí do typu `String`

Existuje `String?` čo je `String` alebo `null`, vo všeobecnosti:  $T? = T \cup \text{null}$

`T?` Podobne vo Swingu, Java `Optional[T]`, Scala `Option[T]`

```
fun foo(str : String?) {  
    println(str)  
    if (str != null) println(str.toUpperCase())  
    println(str?.toUpperCase()) // safe call operátor  
                                // x?.m == if (x != null) x.m else null  
}
```

```
fun stringLen(s: String?): Int = s?.length?:0 // Elvis operátor  
if (if (s == null) then null else s.length) == null then 0 else s.length
```

```
fun nonEmptystringLen(s: String?): Int {  
    val sNotNull: String = s!! // určite nebude null,  
    // ak bude tak exception kotlin.KotlinNullPointerException  
    return sNotNull.length  
}
```

# Životný cyklus apky

(prvý – zjednodušený nástrel)

global: 0  
local: 0  
shared: 0

Alt-Insert = Generate Override Implemented Methods:

- `override fun onDestroy()`
- `override fun onPause()`
- `override fun onRestart()`
- `override fun onRestoreInstanceState(Bundle savedInstanceState)`
- `override fun onResume()`
- `override fun onSaveInstanceState(Bundle outState)`
- `override fun onStart()`
- `override fun onStop()`

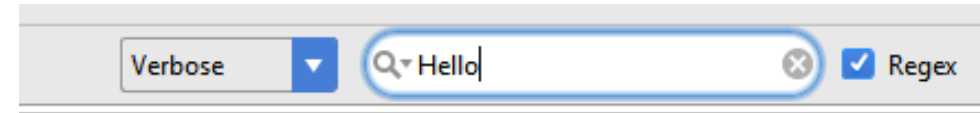
- do každej metódy dáme kontrolný výpis, aby sme pochopili životný cyklus

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    Log.d("CYKLUS", "onCreate") // LOGUJTE, LOGUJTE, LOGUJTE
}
```

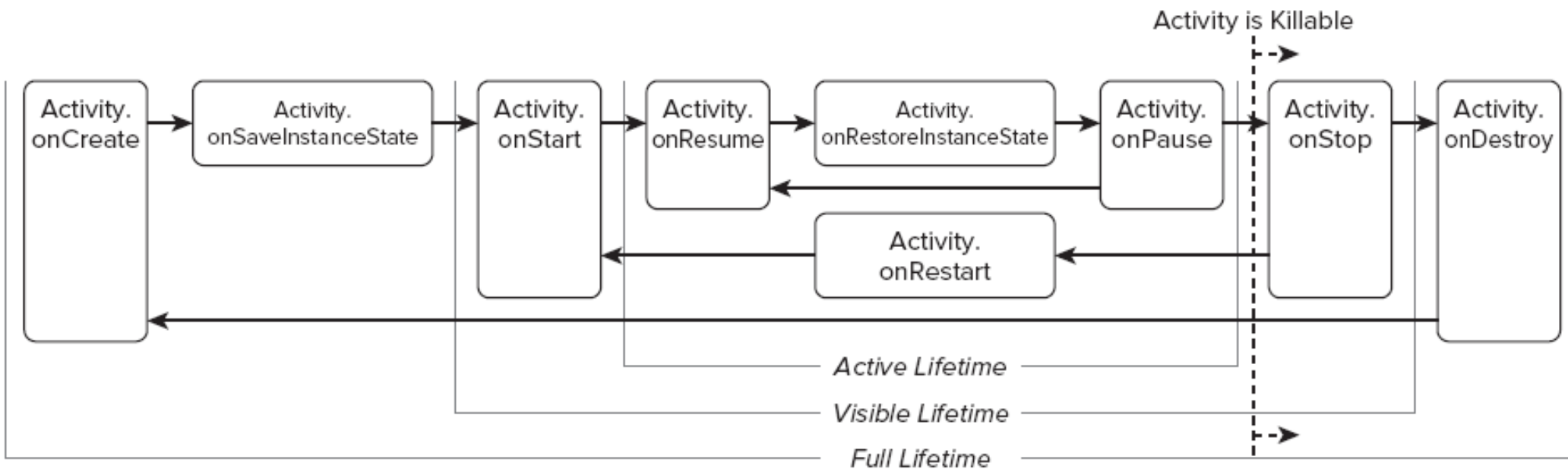
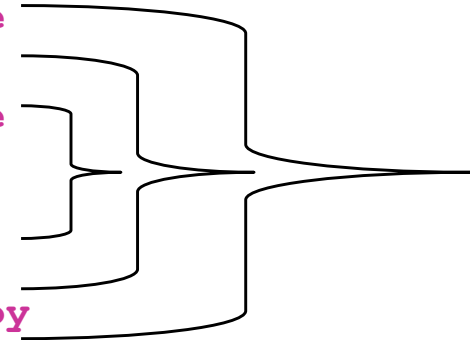
tag vhodný na filtrovanie najlepšie definovať ako konštantu

# LogCat

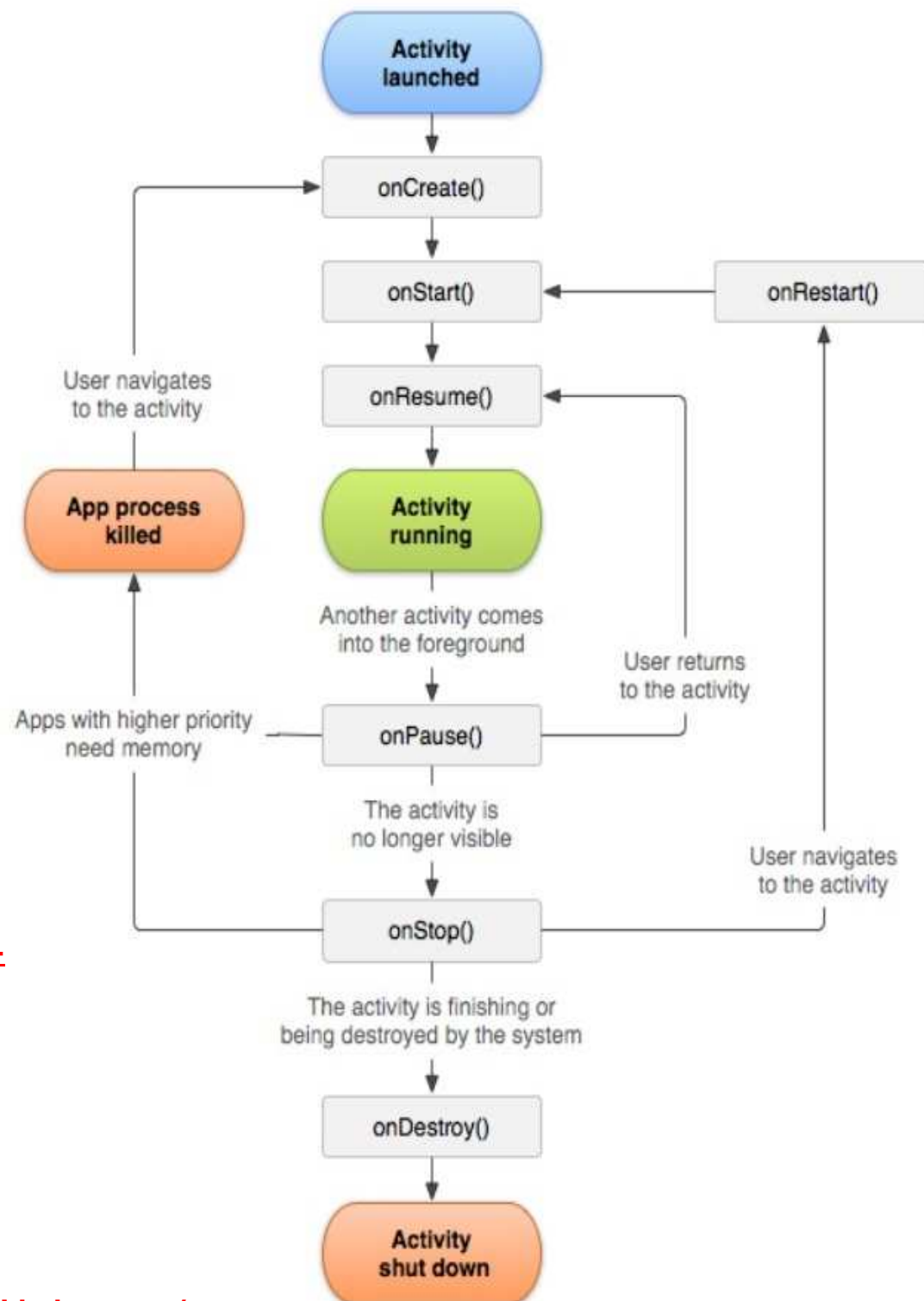
(Filtrovane logov)



- 10-13 12:55:41.091: D/Hello(405): onCreate
- 10-13 12:55:41.091: D/Hello(405): onStart
- 10-13 12:55:41.100: D/Hello(405): onResume
- kill
- 10-13 12:56:45.061: D/Hello(405): onPause
- 10-13 12:56:45.681: D/Hello(405): onStop
- 10-13 12:56:45.681: D/Hello(405): onDestroy







<https://media.geeksforgeeks.org/wp-content/uploads/20191125171002/Activity-Lifecycle-in-Android-Demo-App.mp4>

<https://www.geeksforgeeks.org/activity-lifecycle-in-android-with-demo-app/>

# Persistencia

(prvý dotyk)

global: 0  
local: 0  
shared: 0

- **globalCounter** je premenná, ktorá sa
  - pri **onSaveInstanceState** uloží do Bundle (`Map<String, Value>`)
  - pri **onCreate(savedInstanceState: Bundle?)** príde táto Bundle ako argument
- **localCounter** je bežná lokálna triedna premená v MainActivity
- **sharedCounter** je premenná, ktorá sa ukladá
  - pri **onPause** sa uloží do **SharedPreferences** (`Map<String, Value>`)
  - pri **onResume** sa prečíta zo **SharedPreferences**
- všetky tri premenné sa inkrementujú v metóde **onPause**

Zistíte, že:

- aktivita, ak zmení orientáciu, tak sa reštartne, vytvorí sa nová inštancia a zavolá sa **onCreate**. Preto premenná **localCounter** sa vynuluje.
- ak si chcete niečo uchovať aj po zmene orientácie aktivity, treba to uložiť do bundle, zapíšete to tam v **onSaveInstanceState** a prečítate v **onCreate**
- ak si chcete niečo uchovať aj po reštarte aplikácie, treba to uložiť do **SharedPreferences**



# Bundle?

---

Bundle má metódy [put/get][Int/Boolean/Char/Float/Any/...]

```
override fun onRestoreInstanceState(  
    savedInstanceState: Bundle?) {  
    super.onRestoreInstanceState(savedInstanceState)  
    globalCounter = savedInstanceState?.getInt("COUNTER")?:0  
    ... OLD SCHOOL:  
    if (savedInstanceState != null &&  
        savedInstanceState.getInt("COUNTER") != null) {  
        globalCounter = savedInstanceState!!.getInt("COUNTER")!!  
    } else  
        globalCounter = 0  
}  
  
override fun onSaveInstanceState(outState: Bundle?,  
    outPersistentState: PersistableBundle?) {  
    super.onSaveInstanceState(outState, outPersistentState)  
    outState?.putInt("COUNTER", globalCounter)  
    ...
```



# SharedPreferences

SharedPreferences má metody get[Int/Boolean/Char/Float/Any/...]

```
private lateinit var preferences: SharedPreferences
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    preferences = getSharedPreferences("lifecycle",
                                     Context.MODE_PRIVATE)
}
override fun onResume() {
    sharedCounter = preferences.getInt("kluc", 0)
}
override fun onPause() {
    preferences.edit {
        putInt("kluc",
              sharedCounter)
        apply()
    }
}
```

```
val editor = preferences.edit()
editor.putInt("kluc",
             sharedCounter)
editor.apply()
```

# Čo je Kotlin ?

Kotlin is the New Official Language of Android



Android

+



Kotlin



Kotlin Island

3



# GUI komponenty

## Layout

- LinearLayout (Vertical/Horizontal)
- RelativeLayout, ConstraintLayout

## View, ViewGroup

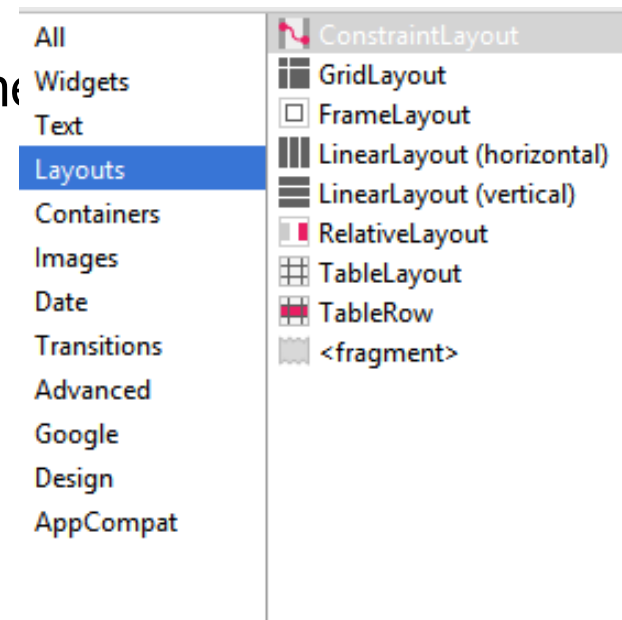
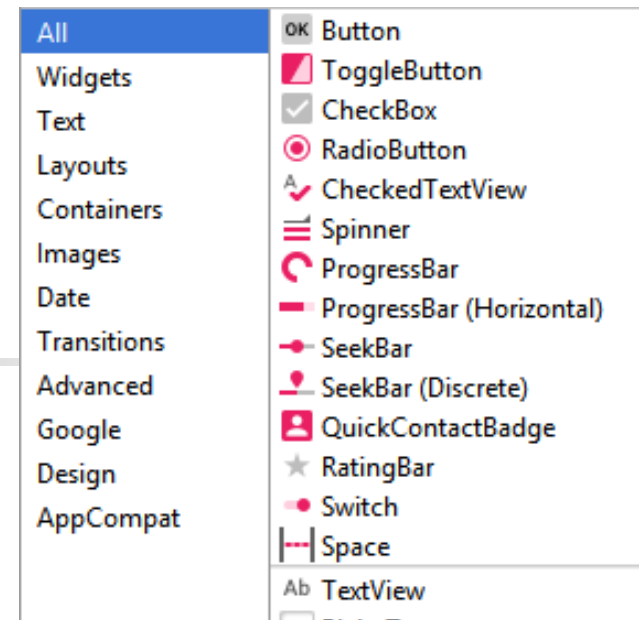
- všetky viditeľné komponenty (widgets)

**Activity** - analógia Screenu (MIT), resp. Form/Frame  
najznámejšie podtriedy

- ListActivity pre ListView, zobrazenie zoznamu
- MapActivity pre MapView (zobrazenie mapy)

## Fragment (>= API level 11)



- reusable UI components



# Layouts

(match\_parent, wrap\_content)



- FrameLayout – objekty umiestni v ľavom hornom rohu
- LinearLayout – horizontálny/vertikálny  
- RelativeLayout – dovoľí umiestniť objekty relatívne k pozíciám iných objektov
- ConstraintLayout (support library, API 9, od Android Studio 2.2)
- GridLayout (od API Level 14)

## <FrameLayout

```
android:id="@+id/FrameLayout1"  
android:layout_width="match_parent"  
android:layout_height="match_parent"
```

## <ImageView

```
android:id="@+id/imageView1"  
android:layout_width="match_parent" --roztiahni podľa  
android:layout_height="match_parent" -- rodičovského  
android:src="@drawable/ic_launcher" />
```



## </FrameLayout>



# LinearLayout

```
<LinearLayout
```

```
    android:orientation="vertical"
```



```
    <LinearLayout
```

```
        android:orientation="horizontal"
```



```
        <TextView
```

```
            android:id="@+id/lb1"
```

```
            android:text="@string/login"/>
```

```
        <EditText
```

```
            android:id="@+id/logintv"
```

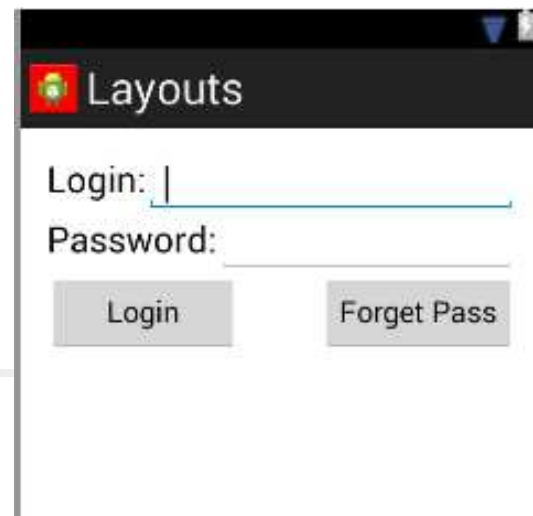
```
            android:layout_width="match_parent" --rozťahni
```

```
            android:layout_height="wrap_content" --na výšku fontu
```

```
            android:inputType="textEmailAddress" /> -- filter
```

```
    </LinearLayout>
```

... podobne pre password





# LinearLayout

(weight, gravity, align with the base line)

<LinearLayout ... Pokračovanie

<LinearLayout

android:orientation="horizontal"

<Button

android:id="@+id/logBtn"

**android:layout\_weight="50"**

android:text="@string/Login"/>

<Space

**android:layout\_weight="50" />**

<Button

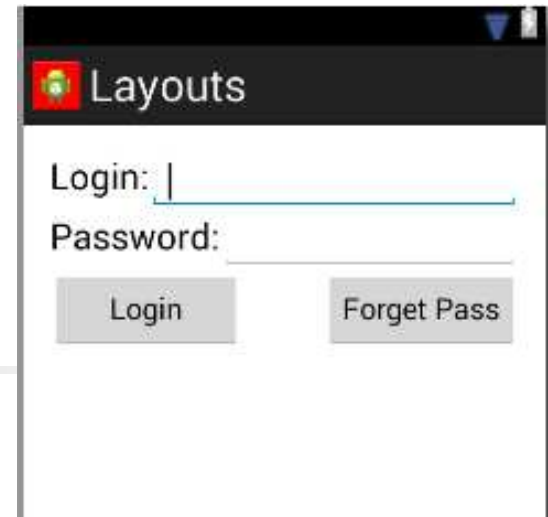
android:id="@+id/forgetPass"

**android:layout\_weight="50"**

android:text="@string/forget" />

</LinearLayout>

</LinearLayout>



# RelativeLayout

```
<RelativeLayout
```

```
<Button
```

```
    android:id="@+id/button1"
```

```
    android:layout_alignParentLeft="true"
```

```
    android:layout_alignParentTop="true"/>
```

```
<Button
```

```
    android:id="@+id/button2"
```

```
    android:layout_below="@+id/button1"
```

```
    android:layout_toRightOf="@+id/button1"/>
```

```
... <Button
```

```
    android:id="@+id/button4"
```

```
    android:layout_alignLeft="@+id/button1"
```

```
    android:layout_below="@+id/button3"
```

```
    android:layout_toLeftOf="@+id/button3" />
```

```
</RelativeLayout>
```



# RelativeLayout

```
<RelativeLayout> ... skrátené...  
<EditText
```

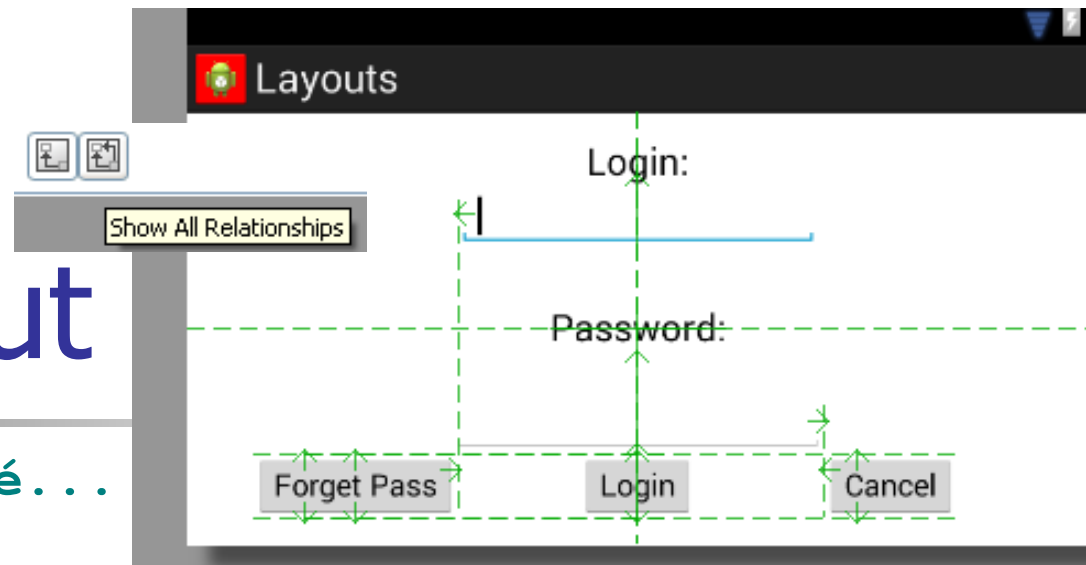
```
    android:id="@+id/passwdtv"  
    android:layout_below="@+id/pass"  
    android:layout_centerHorizontal="true"/>
```

```
<Button
```

```
    android:id="@+id/loginBtn"  
    android:layout_below="@+id/passwdtv"  
    android:text="@string/Login" />
```

```
<Button
```

```
    android:id="@+id/forgetBtn"  
    android:layout_alignBottom="@+id/loginBtn"  
    android:layout_alignTop="@+id/loginBtn"  
    android:layout_toLeftOf="@+id/passwdtv"  
    android:text="@string/forget" />
```



# GridLayout

`<GridLayout`

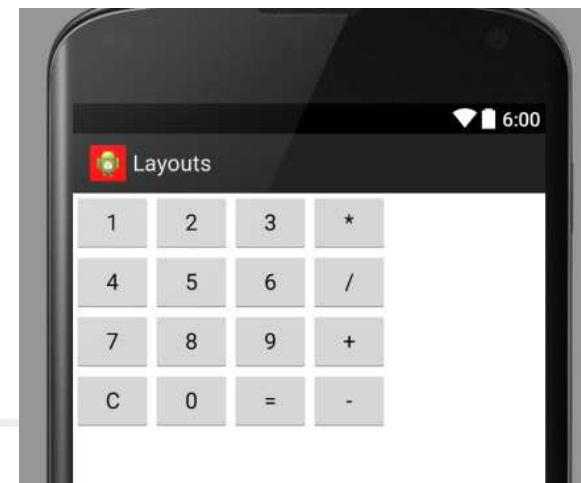
```
android:layout_width="wrap_content"  
android:layout_height="match_parent"  
android:columnCount="4"  
android:rowCount="4">
```

`<Button`

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:text="1"  
android:id="@+id/button1"  
android:layout_row="0"  
android:layout_column="0" />
```

`<Button ...`

```
android:layout_row="0"  
android:layout_column="1" />
```

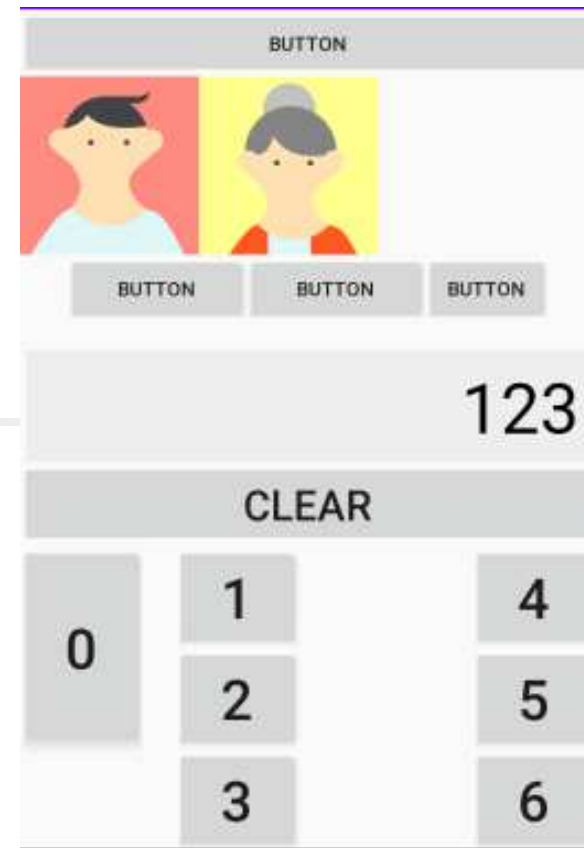


# Table vs. GridLayout

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <Button
        android:id="@+id/button7"
        android:text="Button" />

    <TableRow
        . . .
```



```
<GridLayout
    android:id="@+id/grid"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:columnCount="4"
    android:rowCount="6">

    <TextView
        android:layout_columnSpan="4"
        android:layout_gravity="right"/>
```

# Dynamický vs. statický layout

Layout môžete vyklikať (niekedy náročné) alebo naprogramovať

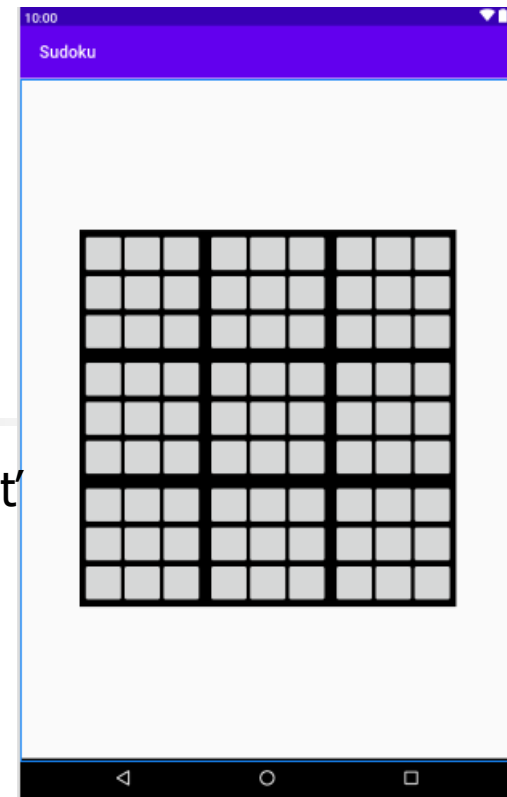
```
<GridLayout  
    android:id="@+id/bigGrid"  
    android:columnCount="3"  
    android:rowCount="3">
```

```
    <GridLayout  
        android:columnCount="3"  
        android:rowCount="3">
```

```
        <Button  
            android:id="@+id/button00"  
            android:layout_width="@dimen/buttonSize"  
            android:layout_height="@dimen/buttonSize"  
            android:layout_margin="@dimen/buttonmargin"  
            android:onClick="onClick"  
            android:text="" />
```

```
    </GridLayout>
```

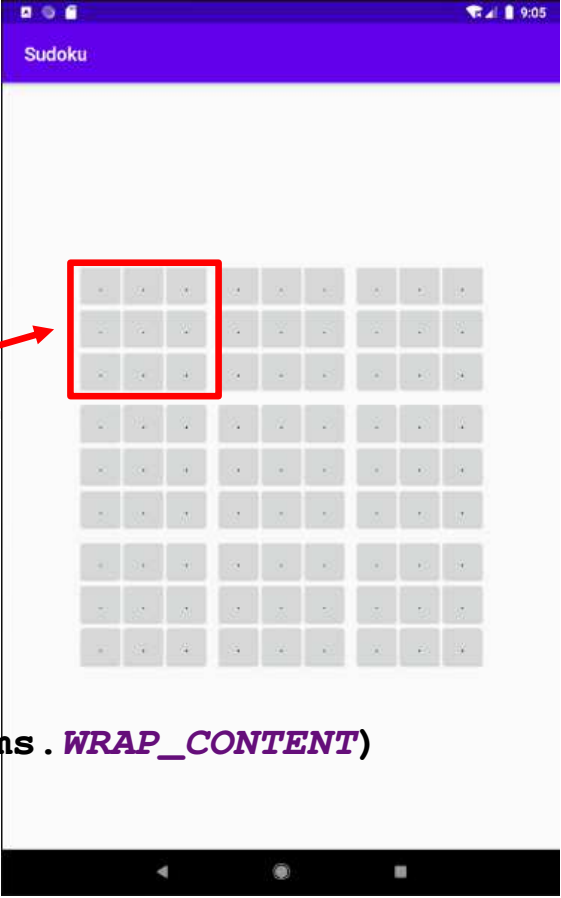
```
</GridLayout>
```



# Dynamický vs. statický layout

```
val smallGrid = GridLayout(this)
smallGrid.columnCount = SIZE
smallGrid.rowCount = SIZE
val smallGridParams = ViewGroup.MarginLayoutParams(
    ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT)
smallGrid.layoutParams = smallGridParams

for (row in 0 until smallGrid.rowCount) {
    for (col in 0 until smallGrid.columnCount) {
        val button = Button(this)
        button.id = ... + 3*3*3*rowb + 3*3*row + 3*colb + col
        button.text = "."
        val buttonSize = resources.getDimension(R.dimen.buttonSize).toInt()
        val buttonParams = ViewGroup.MarginLayoutParams(buttonSize, buttonSize)
        button.layoutParams = buttonParams
        button.setOnClickListener { v -> onClick(v) }
        smallGrid.addView(button)
    }
}
bigGrid.addView(smallGrid)
```



```
fun onClick(v: View) {
    Log.d("SUDOKU", "clicked on ${v.id}")
}
```

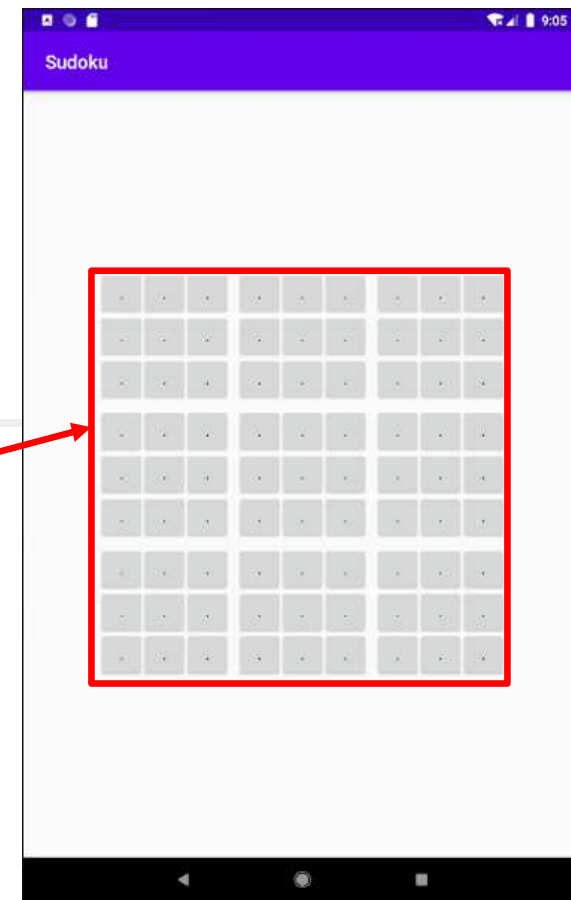
# Dynamický vs. statický layout

```
val SIZE = 3
```

```
val bigGrid = GridLayout(this)
bigGrid.columnCount = SIZE
bigGrid.rowCount = SIZE
bigGrid.layoutParams = ViewGroup.LayoutParams(
    ViewGroup.LayoutParams.WRAP_CONTENT,
    ViewGroup.LayoutParams.WRAP_CONTENT)
for (rowb in 0 until bigGrid.rowCount) {
    for (colb in 0 until bigGrid.columnCount) {
        val smallGrid = GridLayout(this)

        ... celý kód z predošlého slajdu

        bigGrid.addView(smallGrid)
    }
}
ll.addView(bigGrid)
```



```
<LinearLayout
    android:id="@+id/ll"
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```





# Puzzle 8

```

buttons = arrayOf(
    arrayOf(bindings.button00, bindings.button01, bindings.button02),
    arrayOf(bindings.button10, bindings.button11, bindings.button12),
    arrayOf(bindings.button20, bindings.button21, bindings.button22)
)

```

```

fun shuffleBtn(view: View) {
    bindings.shuffleBtn.isEnabled = false
    bindings.solveBtn.isEnabled = false

    object : CountDownTimer(10000, 100) {
        override fun onTick(p0: Long) {
            swap one pair
        }
        override fun onFinish() {
            bindings.shuffleBtn.isEnabled = true
            bindings.solveBtn.isEnabled = true
        }
    }.start()
}

```

|   |   |   |
|---|---|---|
| 1 | 2 | 7 |
| . | 5 | 3 |
| 4 | 6 | 8 |

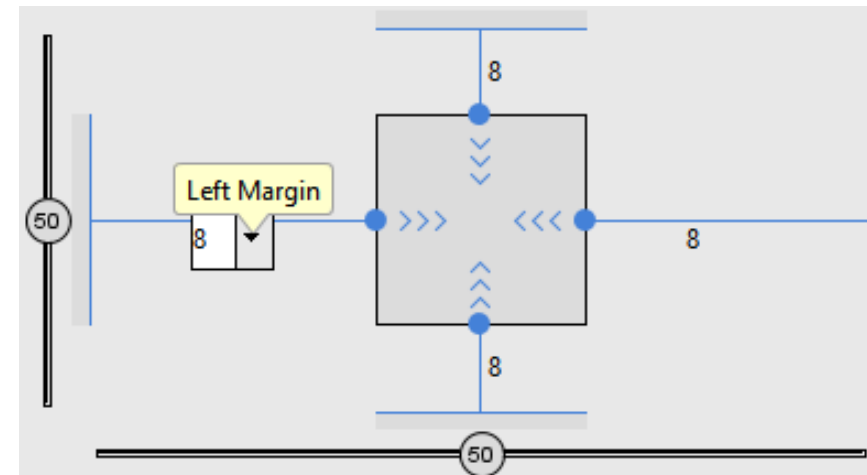
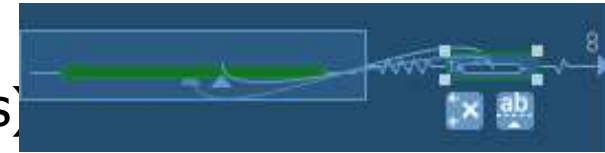
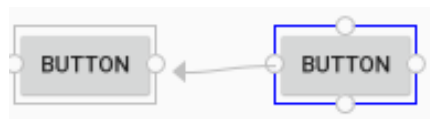
SHUFFLE

SOLVE

# Constraint Layout

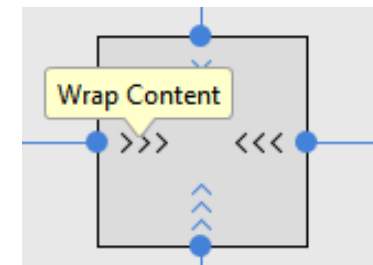
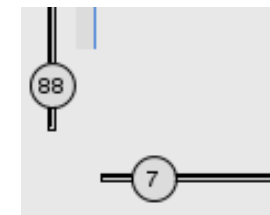
Je zovšeobecnením RelativeLayout umožňuje nastaviť väzby, či obmedzenia (constraints)

- relatívnu pozíciu
- spoločná baseline pre text
- okraje
- wrap/match content/fixná veľkosť
- vychýlenie (bias)



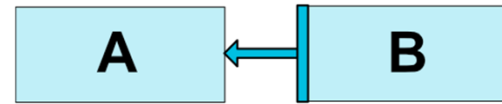
<https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout>

<https://www.youtube.com/watch?v=z53Ed0ddxgM>

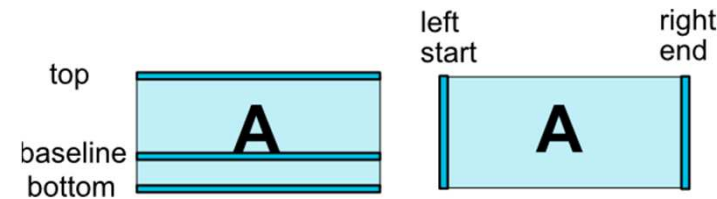


# Niektoré možnosti

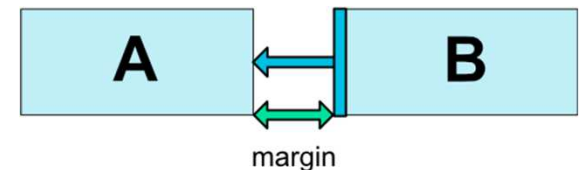
```
<Button android:id="@+id/buttonA" ... />
<Button android:id="@+id/buttonB" ...
    app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```



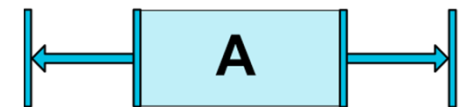
`layout_constraintBaseline_toBaselineOf`



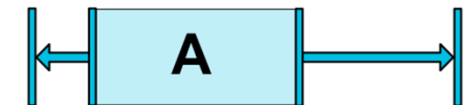
`android:layout_marginLeft`



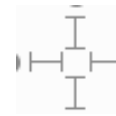
`layout_constraintVertical_bias`



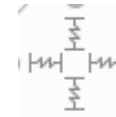
- je ich d'aleko viac, ale zaujímavejšie je to v designeri



# Constraint Layout



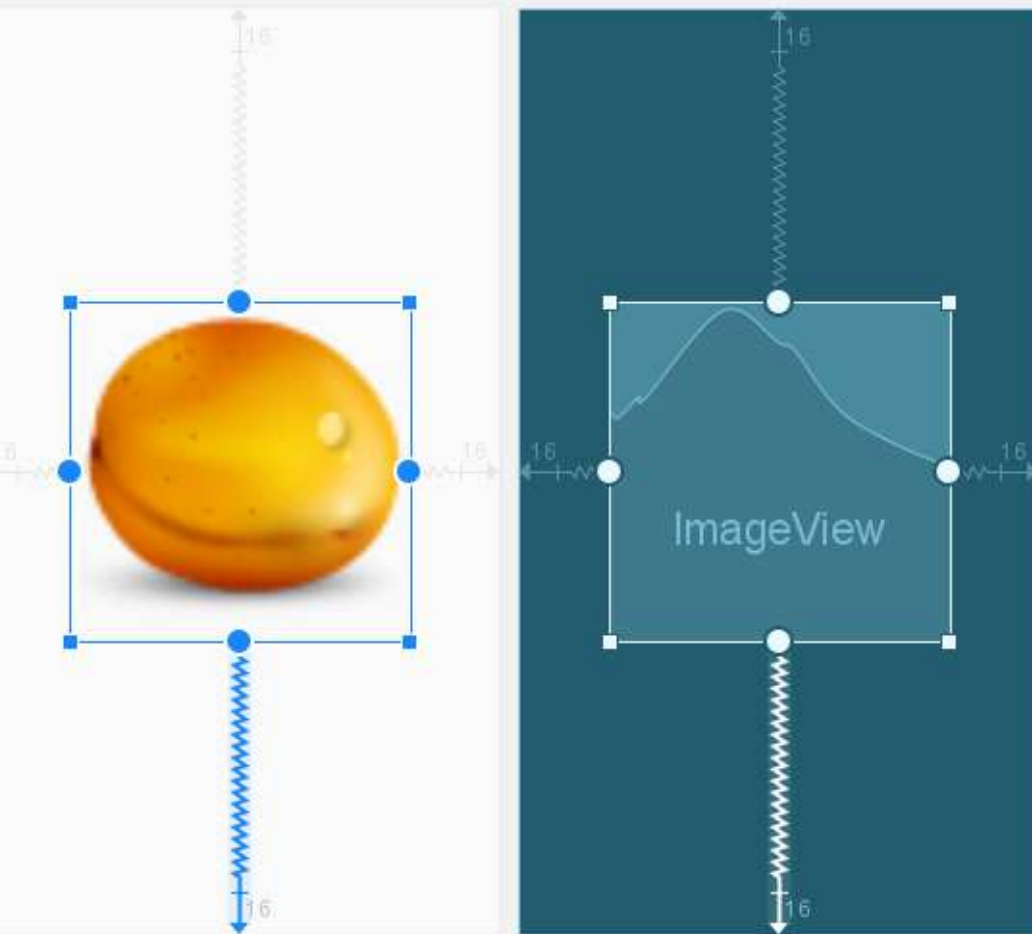
fixed size in dp ☹️



match parent



wrap content



weights

imageView2

srcCompat @drawable/apricot

Layout

Constraint Widget

margins

size

Constraints

- Start → StartOf parent (16dp)
- End → EndOf parent (16dp)
- Top → TopOf parent (16dp)
- Bottom → BottomOf parent (16dp)

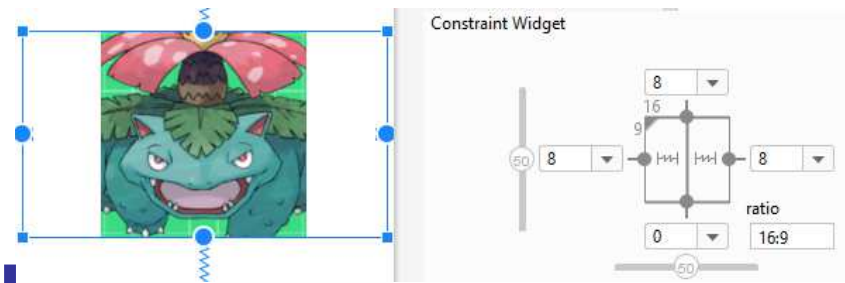
layout\_width 269dp

layout\_height 268dp

visibility

visibility

# Constraint Layout



- Convert View
- Blueprint vs. Design móde
- Show constraints – zobrazí constraints v Blueprint resp. Design móde
- remove constraint (ctrl-)
- Horizontálne vertikálne constraints nemožno miešať
- Komponent musí mať aspoň jeden horizontálny a vertikálny constraint
- Clear All Constraints – zmaže všetky
- Okraje - `layout_marginStart`
- Infer Constraints –
- `wrap_content/match_constraint=0dp` (deprecated `match_parent`)
- `layout_constraintWidth_min`, `layout_constraintWidth_max`
- base line
- ImageView – ratio - `layout_constraintDimensionRatio`
- `rotation[X,Y,Z]`, `scale[X,Y,Z]`, `translation[X,Y,Z]`



# Constraint Layout

---

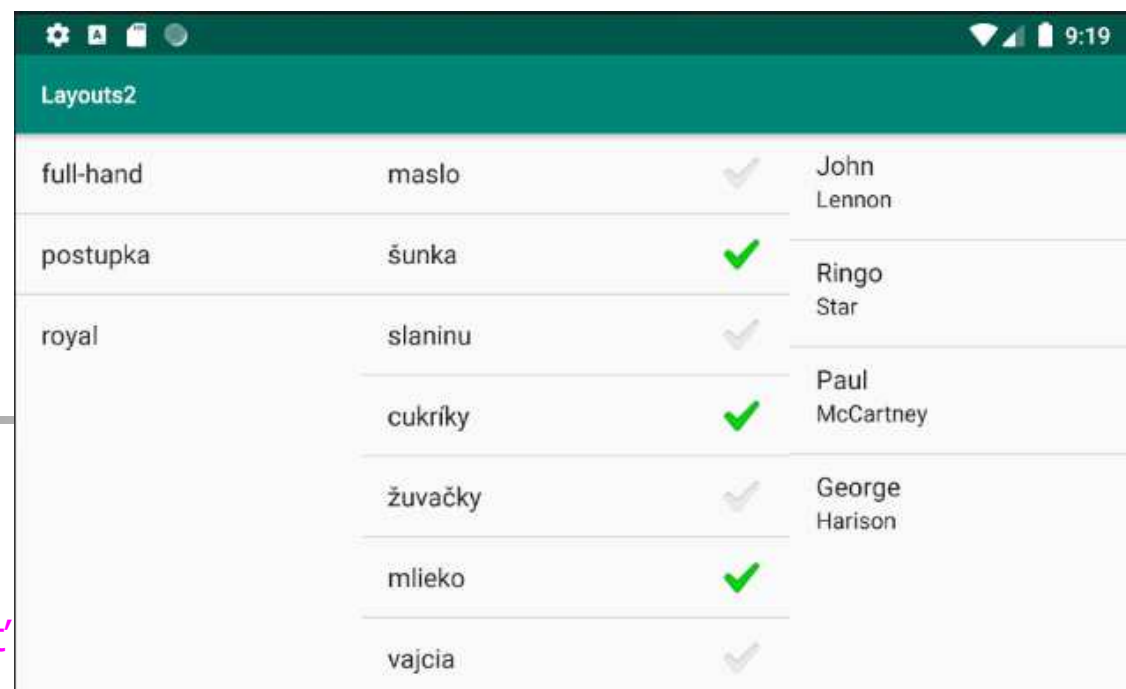
- zret'azenie – chain
- zarovnanie - alignment
- centrovanie
- guidelines
- Barriers
- Groups

# ListView

(variabilita)

ListView a ListActivity zobrazujú zoznam položiek a môžu mať

- preddefinovaný štýl
  - môžu/nemusia sa nám páčiť
  - ale sú ready to use
- user defined
  - narobíme sa pri ich definícii



| full-hand | maslo   | ✓ | John Lennon    |
|-----------|---------|---|----------------|
| postupka  | šunka   | ✓ | Ringo Star     |
| royal     | slaninu | ✓ | Paul McCartney |
|           | cukríky | ✓ | George Harison |
|           | žuvačky | ✓ | George Harison |
|           | mlieko  | ✓ | George Harison |
|           | vajcia  | ✓ | George Harison |

Rôzne inštancie ListView  
`simple_list_item_1`,  
`simple_list_item_activated_1`  
`simple_list_item_checked`  
`simple_list_item_2`  
....

## Odchyťavanie udalostí v ListView

```
com.example.layouts2 D/ZOZNAM: beatles click: 2:{krstne=Paul, priezvv=McCartney}
com.example.layouts2 D/ZOZNAM: beatles click: 1:{krstne=Ringo, priezvv=Star}
com.example.layouts2 D/ZOZNAM: beatles click: 3:{krstne=George, priezvv=Harison}
com.example.layouts2 D/ZOZNAM: check click: 3:cukríky
com.example.layouts2 D/ZOZNAM: check click: 4:žuvačky
com.example.layouts2 D/ZOZNAM: item click: 1:postupka
com.example.layouts2 D/ZOZNAM: item click: 2:royal
com.example.layouts2 D/ZOZNAM: item click: 0:full-hand
com.example.layouts2 D/ZOZNAM: check click: 2:slaninu
```

# ListView

(simple\_list\_item\_1)

|           |         |   |
|-----------|---------|---|
| full-hand | žuvačky | ✓ |
| postupka  | mlieko  | ✓ |
| royal     | vajcia  | ✓ |
|           | pečivo  | ✓ |

simple\_list\_item\_1

imple\_list\_item\_checked

```
// poker - simple_list_item1 view
listView1.adapter = ArrayAdapter<String>(
    this,
    android.R.layout.simple_list_item_1, // jednoriadkový
    // simple_list_item_activated_1
    resources.getStringArray(R.array.poker) // hodnoty
)

// listView1.choiceMode = ListView.CHOICE_MODE_MULTIPLE

listView1.setOnItemClickListener {
    adapterView, view, index, l -> // View.OnItemClickListener
    val hodnota = adapterView.getItemAtPosition(index)
    Log.d(TAG, "item click: $index:$hodnota")
}
```



# ListView

(simple\_list\_item\_checked)

|           |         |   |
|-----------|---------|---|
| full-hand | žuvačky | ✓ |
| postupka  | mlieko  | ✓ |
| royal     | vajcia  | ✓ |
|           | pečivo  | ✓ |

simple\_list\_item\_1

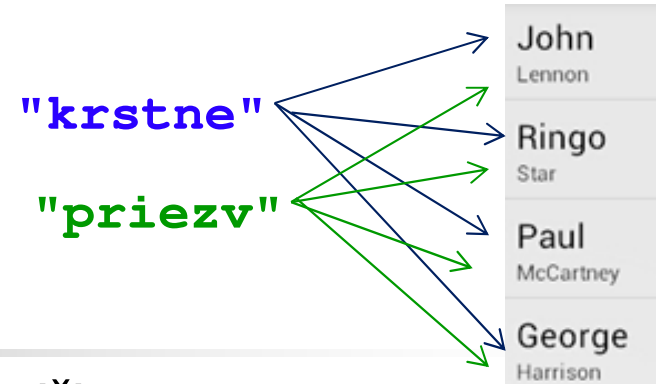
simple\_list\_item\_checked

```
// nákup - checked box list view
listView2.adapter = ArrayAdapter<String>(
    this,
    android.R.layout.simple_list_item_checked, //2riadkový
    resources.getStringArray(R.array.nakup)
)

listView2.setOnItemClickListener {
    adapterView, view, index, l ->
        val hodnota = adapterView.getItemAtPosition(index)
        (view as CheckedTextView).toggle() // prekresli
        Log.d(TAG, "check click: $index:$hodnota")
}
```

# ListView 2

(simple\_list\_item\_2)



Naplniť iný, napr. dvojriadkový ListView je náročnejšie *//beatles list view*

```
val pairs = listOf( // hodnoty sú zoznam máp klúč->hodnota
    mapOf("krstne" to "John", "priezv" to "Lennon"), mapOf("krstne" to "Ringo", "priezv" to "Star"),
    mapOf("krstne" to "Paul", "priezv" to "McCartney"), mapOf("krstne" to "George", "priezv" to "Harison")
)

listView3.adapter = SimpleAdapter(this,
    pairs, // hodnoty
    android.R.layout.simple_list_item_2, // format ListView
    arrayOf("krstne", "priezv"), // zoznam klúčov
    arrayOf(android.R.id.text1, android.R.id.text2) // riadky
    .toIntArray()
)

listView3.setOnItemClickListener {
    adapterView, view, index, l ->
    val hodnota = adapterView.getItemAtPosition(index)
    Log.d(TAG, "beatles click: $index:$hodnota:" +
        "${(hodnota as Map<String, String>)[\"krstne\"]
}
```

# Rôzne preddefinované ListView

(prehľad)

The image displays four screenshots of Android applications, each showing a different ListView implementation. Each screenshot has a status bar at the top with a 3G signal icon, a battery icon, and a time display.

- SimpleListItemSingleChoice:** Shows a list titled "BuiltInViews" with six items: Vegetables, Fruits, Flower Buds, Legumes, Bulbs, and Tubers. Each item has a radio button to its right. The "Fruits" item is selected, indicated by a blue dot in the center of the radio button.
- TwoLineListItem:** Shows a list titled "BuiltInViews" with six items: Vegetables (65 items), Fruits (17 items), Flower Buds (5 items), Legumes (33 items), Bulbs (18 items), and Tubers (43 items). Each item has a small icon to its left.
- ActivityListItem:** Shows a list titled "BuiltInViews" with six items: Vegetables, Fruits, Flower Buds, Legumes, Bulbs, and Tubers. Each item has a small icon to its left.
- SimpleExpandableListItem:** Shows a list titled "BuiltInExpandableViews" with four expandable items: Vegetables, Squash, Zucchini, and Carrots. Each item has a small icon to its left. The "Vegetables" item is expanded, showing a list of sub-items: Squash (33 units), Zucchini (6 units), Carrots (20 units), Fruit, and Herbs.

Below each screenshot, the corresponding ListView implementation is labeled:

- SimpleListItemSingleChoice
- TwoLineListItem
- ActivityListItem
- SimpleExpandableListItem

# Domáca úloha 2

(jedna z 3 alternatív, na budúcu prednášku bude Menus)

Debilníček

|                                    |          |
|------------------------------------|----------|
| zavolať svokre, že ju obdivujem... | 21/11/12 |
| vencit Harryho                     | 21/11/12 |
| pivo                               | 21/11/12 |
| syr                                | 21/11/12 |
| mlieko                             | 21/11/12 |

Vytvorte (malú) aplikáciu zvanú Debilníček, resp. Nákupný košík:

- umožní poznamenať si, veci, predmety, činnosti do tzv. ToDo listu,
- dovoľí nastaviť deadline na splnenie činnosti pomocou dátumu/času,
- ak to bude verzia nákupný košík, tak aj počet predmetov,
- umožní ich vymazať, resp. označiť za vybavené/nakúpené, resp. vymazať všetky vybavené,
- kontroluje deadline, a upozorní správou, zvukom na prešvihnutý deadline,
- pri vypnutí aplikácie si zoznam zapamätá, pri otvorení sa zoznam obnoví

+

Add New Item

×

Remove Item

←

🏠

📄

☰

🔍

↓

🖼️

⚠️

✉️

20:32

📶

🔋

🔌

🔔

🔑