



Fragment

Peter Borovanský
KAI, I-18

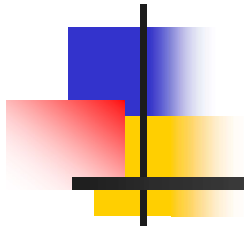
borovan 'at' ii.fmph.uniba.sk





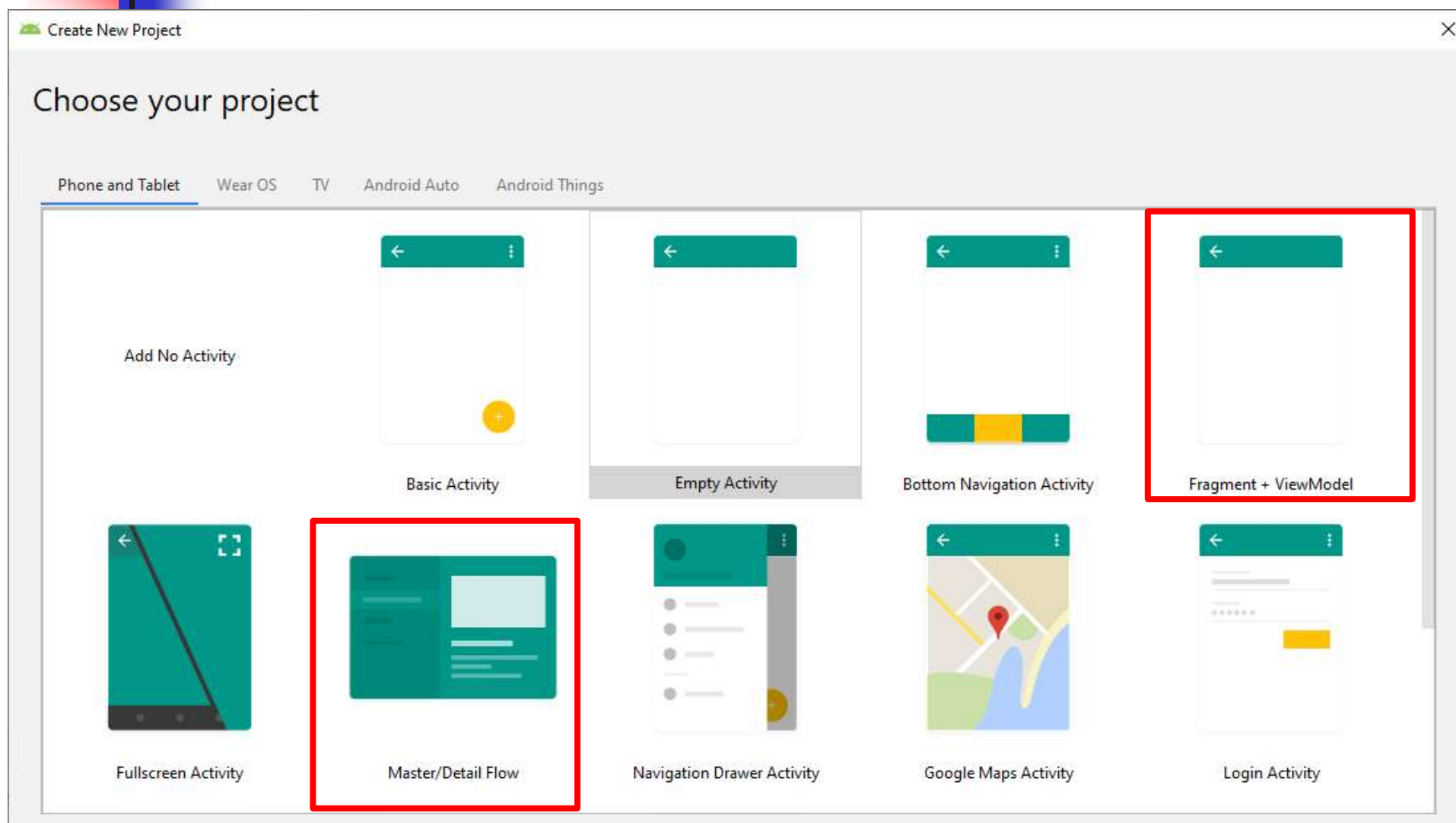
Hitparáda DU2

(Hall of Fame)

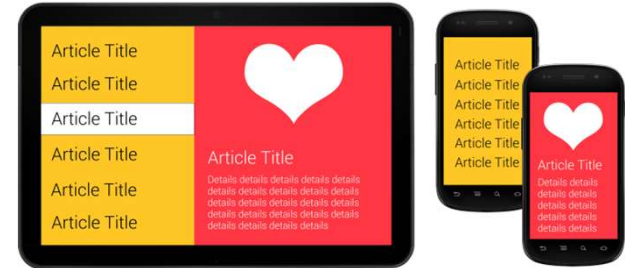


<https://github.com/vmandro/Riesenia/tree/master/DU02/LindaJ/todolist>

O čo to dnes bude

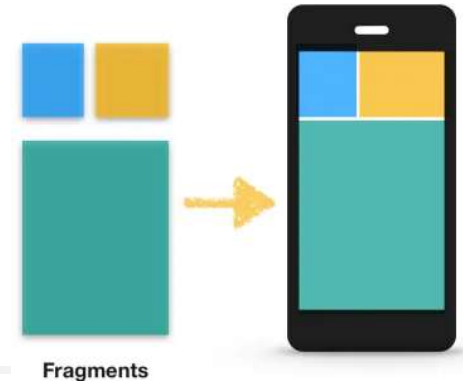


O čo to dnes bude



- **Fragment** ako základný stavebný kameň zložitejšej aplikácie
 - fragment je samostatne existujúca modul časť aplikácie majúca layout aj správanie
 - je to podtrieda FragmentActivity (nie AppCompatActivity)
 - svoj layout má definovaný v .xml
 - princípy fungovania
 - jednoduché používanie existujúcich Dialog Fragmentov
- **Master-Detail** aplikácia
 - Master je napr. zoznam všetkých objektov, Detail je detail jedného z nich
- Návrhové vzory a Android
 - Model View Controller (MVC)
 - **Model View ViewModel** (MVVM)
 - **JetPack v AndroidX** (androidx.* packages)

Fragmenty



- **fragment** predstavuje ucelenú časť GUI, podobne ako aktivita
- fragment má, podobne ako aktivita, životný cyklus, ale zložitejší
- hlavným cieľom fragmentu je jeho znovu-použiteľnosť (reusability)
- každý fragment má svoju aktivitu, ktorá si ho pri inicializácii pripojí (attach)
 - aktivita si vkladá do seba jeden, alebo viac fragmentov, ktoré môžu komunikovať
- Koexistencia fragmentu a aktivity je zložitejšia ako život aktivity
- vzťah fragment-aktivita je typu many-many
 - Fragment môže hostovať v rôznych aktivitách, o tom je reusability fragmentu
- aktivita môže obsahovať/kombinovať viacero fragmentov, dvomi spôsobmi
 - **staticky** (sú navrhnuté v layout .xml-súboroch)
 - **dynamicky** (vzniknú dynamicky v kóde pomocou konštruktora podtriedy Fragmentu)

Fragmenty

(história a následky)



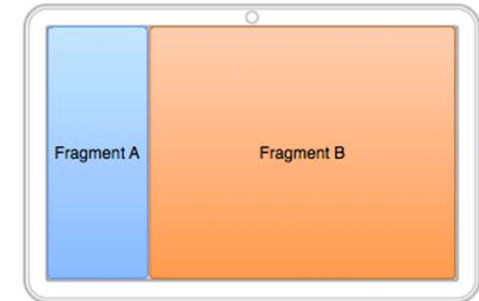
- fragmenty sú podporované od Android 3.1 (API 11)
- ak naše minSDK < 11, použijeme Support Library
<https://developer.android.com/topic/libraries/support-library/index.html>
- knižnice podporujúce Fragment sú:
 - ~~android.app.Fragment~~ (This class was deprecated in API level 28)
 - `android.support.v4.app` (od API 26-July,2017, min.API level 14)
 - a najnovšie Android Jetpack, balíky `androidx.*` od Android 9.0 (API level 28)

Pozor na miešanie importov z rôznych knižníc:

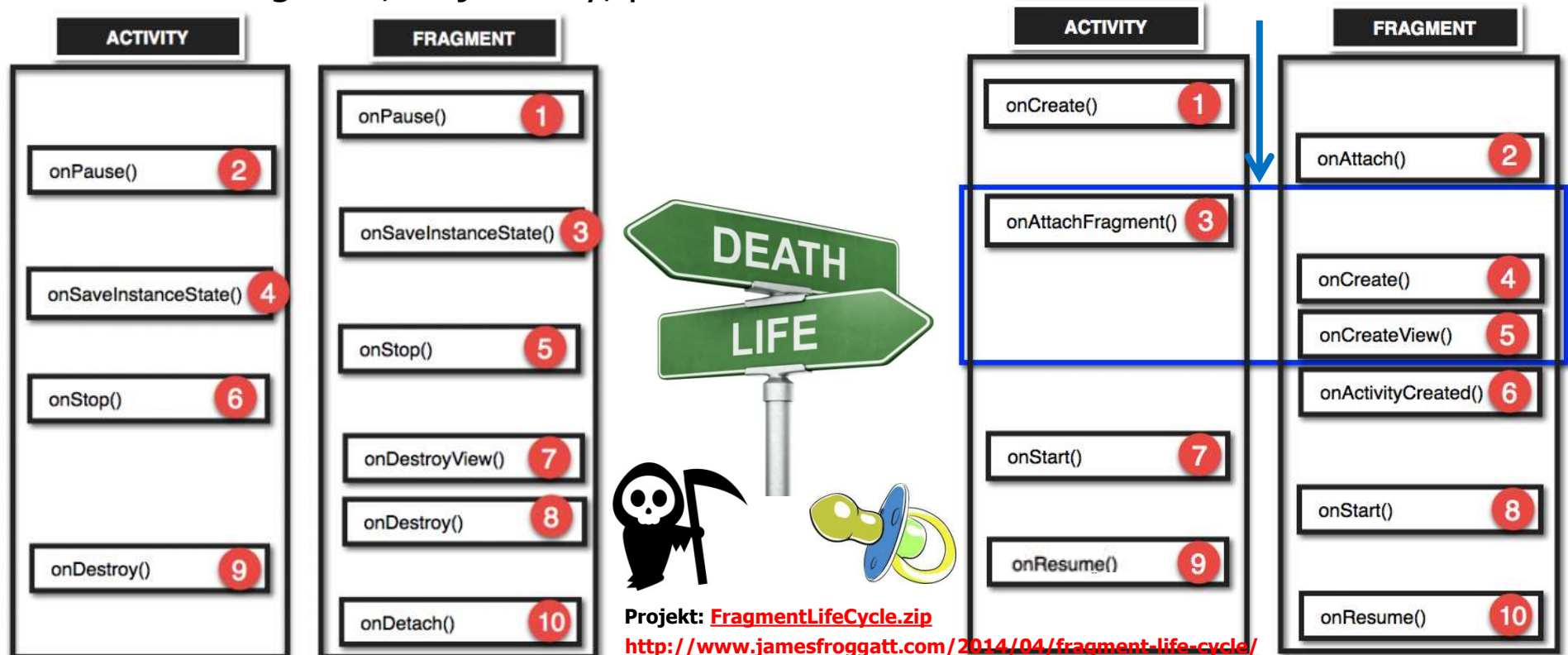
- `android.app.Fragment`
- `!= android.support.v4.app.Fragment`
- `!= androidx.fragment.app.Fragment`
- Stav fragmentu (životný cyklus extrémne stručne):
 - definujeme podtriedu triedy Fragment, kým nezavoláme konštruktor, tak *neexistuje nič* !
 - po `FragmentSubClass()`, existuje inštancia fragmentu ako objekt, *nevidíme nič*
 - aktivita linkuje (*attachne*) fragment, *nevidíme nič*, ale aspoň fragment vie, že má aktivitu
 - fragment sa zobrazí na obrazovke, a *vidíme ho a existuje*

Život fragmentu

(je zložitejší ako u aktivita)

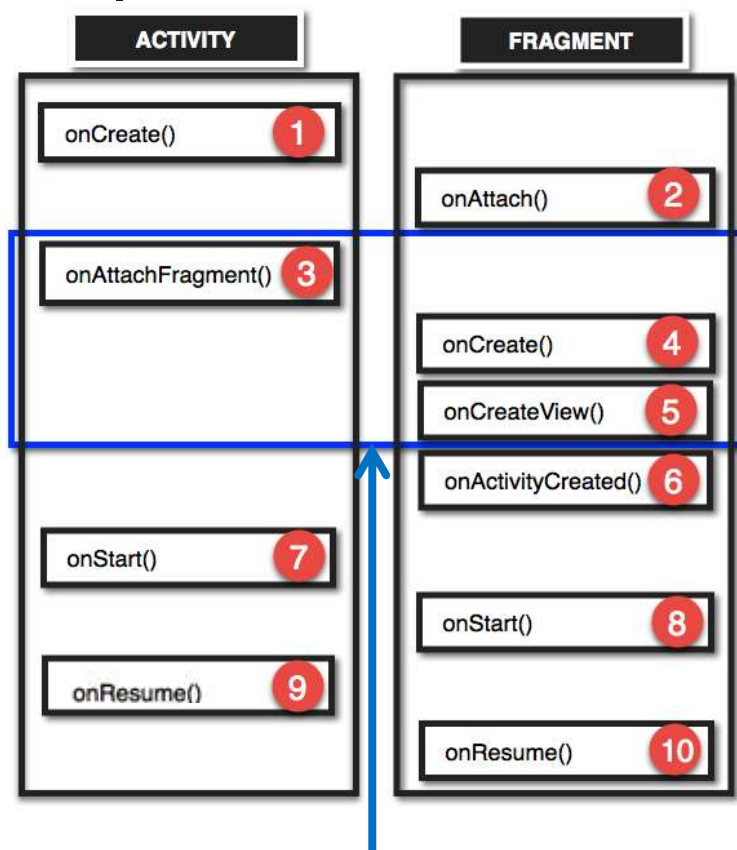


- fragment predstavuje ucelenú časť GUI, podobne ako aktivita
- fragment má svoju aktivitu, ktorá ho pripojí (predpokladajme vzťah 1:1)
- ...aktivita môže obsahovať/kombinovať (aj dynamicky) viacero fragmentov
- fragment, ak je dobrý, používa ho viacero aktivít (reusability)



Vznik fragmentu

(venujeme sa vzniku, nie zániku)

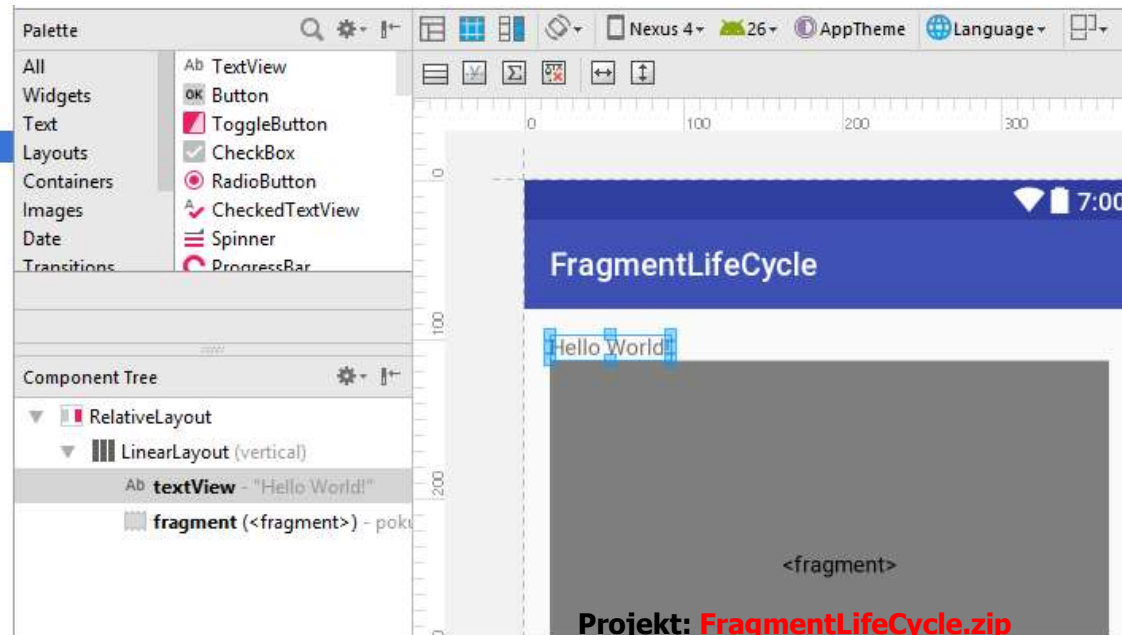
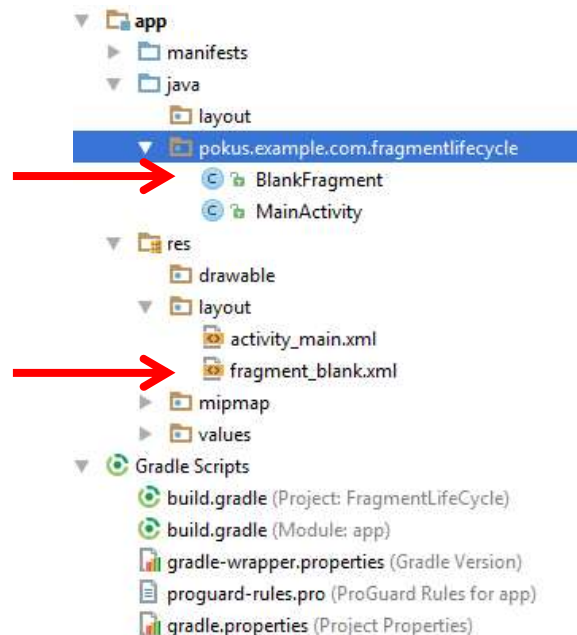


1. **onCreate v aktivite**: Najčastejšie obsahuje `setContentView`, ktorá definuje layout aktivity
2. **onAttach vo fragmente**: dostaneme pointer na aktivitu, do ktorej je vkladáný, uložíme si ho...
3. **onAttachFragment v aktivite**: dozvie sa, že fragment bol attach-nutý do aktivity
4. **onCreate vo fragmente**: aktivita `onCreate` nemusí byť ukončená, preto nie je dovolené adresovať UI komponenty z aktivity
5. **onCreateView vo fragmente**: fragmentu určíme layout, inflater inflatuje
6. **onActivityCreated vo fragmente**: už konečne vidíme UI komponenty aj z aktivity
7. **onStart v aktivite**
8. **onStart vo fragmente**
9. **onResume v aktivite**
10. **onResume vo fragmente**

Život fragmentu

(jeden fragment v aktivite)

```
<RelativeLayout>
  <LinearLayout>
    <TextView ...android:text="Hello World!"/>
    <fragment android:id="@+id/fragment"
      android:name="com.example.fragmentlifecycle.BlankFragment"/>
  </LinearLayout>
</RelativeLayout>
```



Život fragmentu

(onSaveInstanceState)

- napr. zmena orientácie displaya
- ak **fragment**/**aktivita** zaniká, môžeme si zapamäť jej stav cez Bundle v onSaveInstanceState

```
override fun onSaveInstanceState(  
    savedInstanceState? : Bundle) {  
    super.onSaveInstanceState(savedInstanceState);  
    savedInstanceState?.putString("key", "value")  
    savedInstanceState?.putInt("score", ...)  
    savedInstanceState?.putLong("time", ...)  
}
```



- a následne reštaurovať:

```
override fun onCreate(savedInstanceState?:Bundle) {  
    super.onCreate(savedInstanceState);  
    savedInstanceState?.getString("key")  
    savedInstanceState?.getInt("score")  
    savedInstanceState?.getLong("time")  
    ... }  
}
```

on Attach
on Create
on CreateView
on Activity Created
on Start
on Resume

on Pause
on Save Instance State
on Stop
on Destroy View
on Destroy
on Detach
on Attach
on Create
on CreateView
on Activity Created
on Start
on Resume

bez onSaveInstanceState

on Pause
on Stop
on Destroy View
on Destroy
on Detach



Back


Zmena orientácie

```
on Create ACTIVITY
on Attach Fragment
on Create Fragment
on CreateView Fragment
on Activity Created Fragment
on Start ACTIVITY
on Start Fragment
on Resume ACTIVITY
on Resume Fragment
on Pause Fragment
on Pause ACTIVITY
on Save Instance State Fragment
on Save Instance State ACTIVITY
on Stop Fragment
on Stop ACTIVITY
on Destroy View Fragment
on Destroy Fragment
on Detach Fragment
on Destroy ACTIVITY
on Create ACTIVITY
on Attach Fragment
on Create Fragment
on CreateView Fragment
on Activity Created Fragment
on Start ACTIVITY
on Start Fragment
on Restore Instance State ACTIVITY
on Resume ACTIVITY
on Resume Fragment
```

Život fragmentu (detail)

restart home screen

```
on Pause Fragment
on Pause ACTIVITY
on Save Instance State Fragment
on Save Instance State ACTIVITY
on Stop Fragment
on Stop ACTIVITY
on Restart ACTIVITY
on Start ACTIVITY
on Start Fragment
on Resume ACTIVITY
on Resume Fragment
```

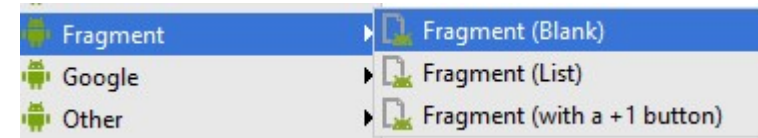
keď aktivitu/fragment dáme na pozadie  , tak sa:

- nevolá **onDestroy**,
- pri opätovnom spustní sa nevolá **onCreate**, ale **onRestart**

Statický fragment

(existuje jeho layout)

- vytvoríme podtriedu Fragment
- AS nám pomôže File/New/Fragment
- vytvoríme dva fragmenty First/Second fragment, a rôzne ofarbíme ich

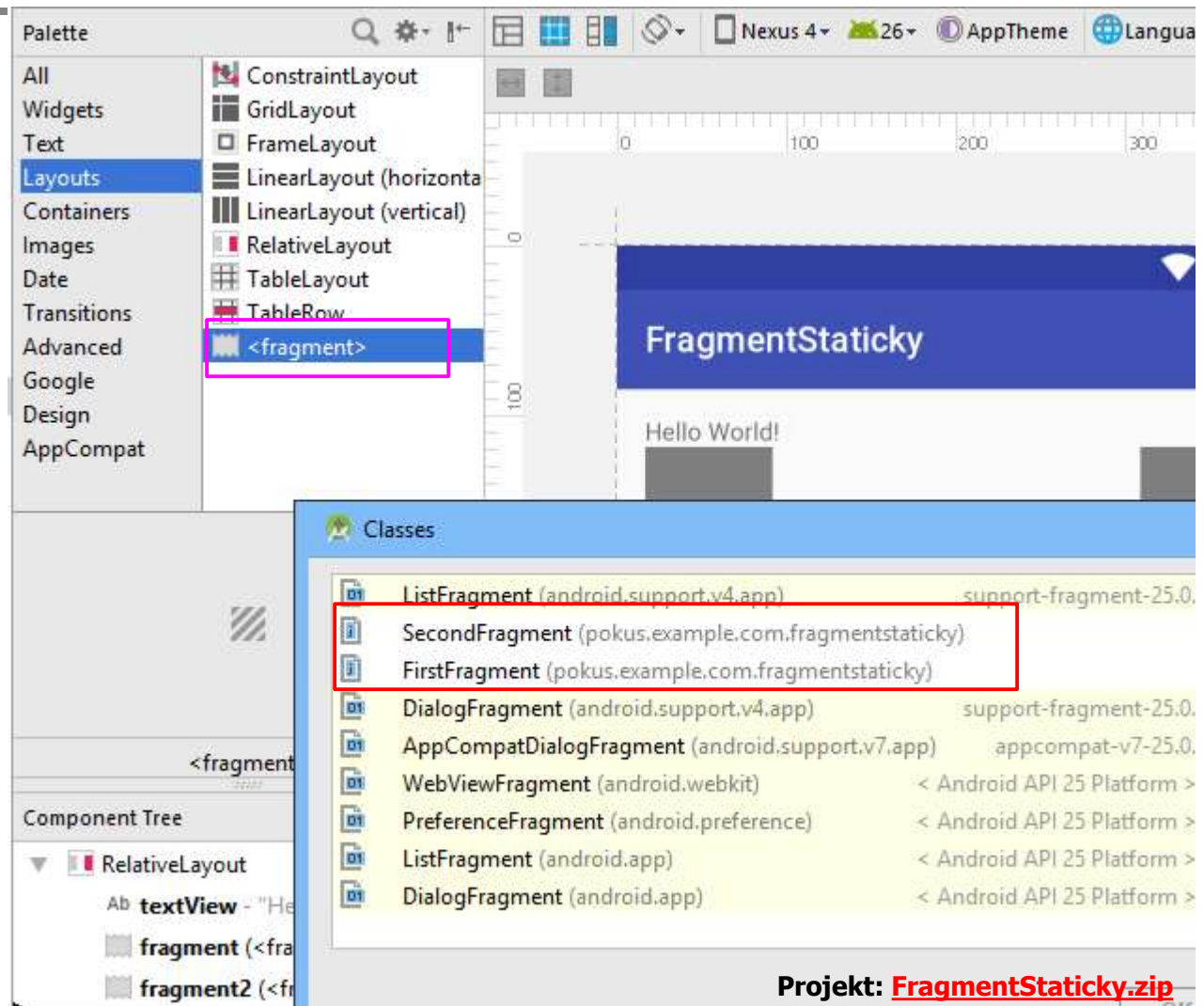


fragment_first.xml

```
<FrameLayout xmlns:android=http://schemas.android.com/apk/res/android
  xmlns:tools=http://schemas.android.com/tools
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="pokus.example.com.fragmentstaticky.FirstFragment">
  <!-- TODO: Update blank fragment layout -->
  <TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorAccent"
    android:text="Hello from fist fragment" />
</FrameLayout>
```

Statický fragment

Ked' potom editujeme layout aktivity, tak môžeme doň vložiť `<fragment>` a v detailnejšej ponuke nájdeme nami vytvorené fragmenty



Statický fragment

(jednoduchá verzia)

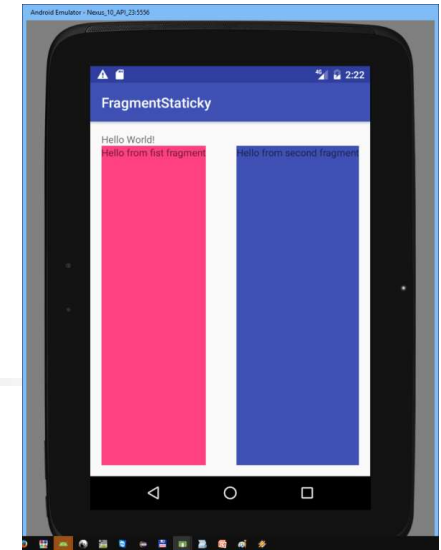
```
class FirstFragment : Fragment() {
    lateinit var mainActivity: MainActivity

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // onCreateView: fragmentu určíme layout, inflater inflatuje
    }

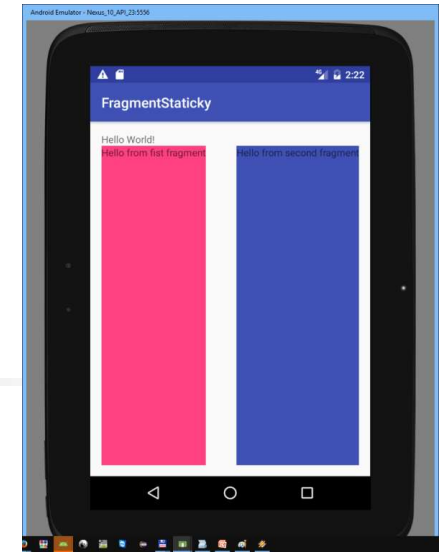
    override fun onCreateView(inflater: LayoutInflater,
                              container: ViewGroup?,
                              savedInstanceState: Bundle?): View? {
        return inflater.inflate(R.layout.fragment_first,
                                container, false)
    }

    override fun onAttach(context: Context) {
        super.onAttach(context)
        mainActivity = context as MainActivity
    }
}
```



Statický fragment

(reálne dostanete – ak si ho necháte vygenerovať)



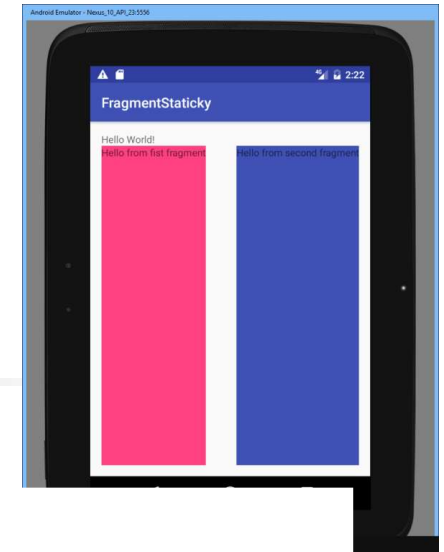
```
private const val ARG_PARAM1 = "param1"
private const val ARG_PARAM2 = "param2" // raz mená vašich parametrov
```

```
class BlankFragment1 : Fragment() {
    private var param1: String? = null // premenné, kam sa načítajú
    private var param2: String? = null
```

```
→ override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    param1 = arguments?.getString(ARG_PARAM1) // tu sa načítajú
    param2 = arguments?.getString(ARG_PARAM2)
}
```

Statický fragment

(reálne dostanete)



Companion object je Singleton Pattern

```
companion object {  
    @JvmStatic  
    fun newInstance(param1: String, param2: String) =  
        BlankFragment1().apply {  
            arguments = Bundle().apply {  
                putString(ARG_PARAM1, param1)  
                putString(ARG_PARAM2, param2)  
            }  
        }  
}
```

inštanciu by ste vyrobili:

```
val bf = BlankFragment1.newInstance("value1", "value2")
```

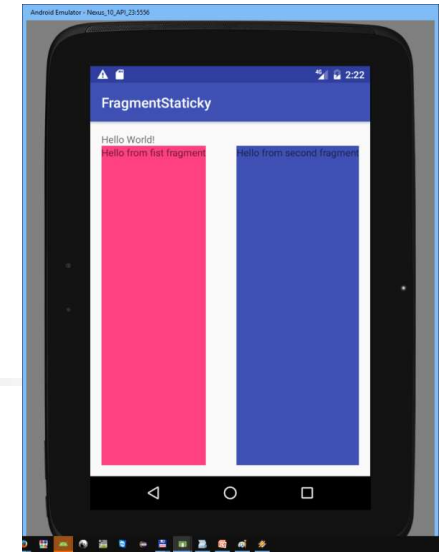

Statický fragment

(reálne dostanete)

```
definujete akýkoľvek listener na komunikáciu s aktivitou
interface OnFragmentManagerListener {
    fun onFragmentManagerInteraction(uri: Uri)
}
// definujete premennú, kam si uložíte pointer na rodičovskú aktivitu,
// ktorá musí implementovať váš listener
private var listener: OnFragmentManagerListener? = null

fun onPressed(uri: Uri) {
    listener?.onFragmentManagerInteraction(uri)
}

override fun onAttach(context: Context) {
    super.onAttach(context) // aktivita, ktorá ho attachuje, musí
    if (context is OnFragmentManagerListener) { // spíňať
        listener = context // interface, a uložíte si pointer na ňu
    } else { // inak fail
        throw RuntimeException(context.toString() +
            " must implement OnFragmentManagerListener")
    }
}
```





Slajder Fragement

fragment_slajder.xml

```
<RelativeLayout >

    <EditText
        android:id="@+id/editText"
        ...
    />

    <SeekBar
        android:id="@+id/seekBar"
        ...
    />

    <Button
        android:id="@+id/button"
        ...
    />
</RelativeLayout>
```

fragment_text.xml

```
<RelativeLayout >

    <TextView
        android:id="@+id/textView"
        ...
    />
</RelativeLayout>
```

activity_main.xml

```
<RelativeLayout >
    <fragment
        android:id="@+id/fragmentSlajder"
    />
    <fragment
        android:id="@+id/fragmentTextView"
    />
</RelativeLayout>
```



Slajder Fragement

```
class SlajderFragment : Fragment() {
    var slajder = 50

    interface Listener {
        fun onClick(position: Int, text : String)
    }
    lateinit var activityCallback : Listener
    override fun onAttach(context: Context) {
        super.onAttach(context)
        try { activityCallback = context as Listener
        } catch (e : ClassCastException) {
            throw ClassCastException(context.toString() + " does not implement Listener")
        }
    }
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle? ): View? {
        return inflater.inflate(R.layout.fragment_slajder, container, false)
    }
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        seekBar.setProgress(slajder)
        seekBar.setOnSeekBarChangeListener (
            object : SeekBar.OnSeekBarChangeListener {
                override fun onProgressChanged(sb: SeekBar, progress: Int, fromUser: Boolean) {
                    slajder = progress
                }
            }
        )
        button.setOnClickListener{ v -> activityCallback.onClick(slajder, editText.text.toString())}
    }
}
```



Slajder Fragement

```
class TextViewFragment : Fragment() {  
  
    override fun onCreateView(  
        inflater: LayoutInflater,  
        container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View? {  
        return inflater.inflate(R.layout.fragment_text,  
                                container, false)  
    }  
  
    fun changeText(fontsize : Int, text : String) {  
return        textView.textSize = fontsize.toFloat()  
        textView.text = text  
    }  
}
```

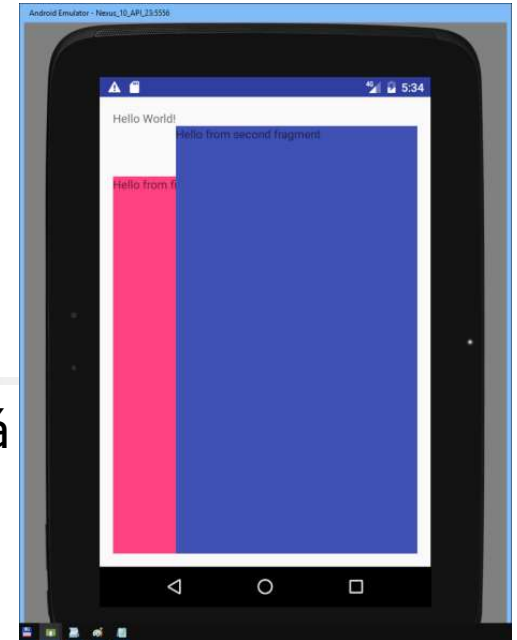


Slajder Fragement

```
class MainActivity : FragmentActivity(), SlajderFragment.Listener {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
    }  
  
    override fun onClick(fontSize: Int, text: String) {  
        val textViewFragment =  
            supportFragmentManager.findFragmentById(  
                R.id.fragmentTextView) as TextViewFragment  
        textViewFragment.changeText(fontSize, text)  
    }  
}
```

Dynamický fragment

- dynamická práca s fragmentmi je častejšia ako statická
- adresovanie fragmentu používame:
 - *supportFragmentManager* (nie *FragmentManager*)
 - `findFragmentById()`
 - `findFragmentByTag()`



```
val sfr = supportFragmentManager // nie FragmentManager android.app.*
        .findFragmentById(R.id.frameLayout2) as SecondFragment
alebo
        .findFragmentByTag("tag2") as SecondFragment
sfr.setText(s)
```

```
<RelativeLayout // layout activity je zjednodušený, pozri kód
    <TextView android:text="Hello World!" android:id="@+id/textView" />
    <FrameLayout android:id="@+id/frameLayout1" </FrameLayout>
    <FrameLayout android:id="@+id/frameLayout2" </FrameLayout> } placeholders
</RelativeLayout>
```



Dynamický fragment

- dynamická práca s fragmentmi je častejšia ako statická
 - vytvorenie inštancie podtriedy Fragment
 - podslanie argumentov fragmentu
 - získanie referencie na fragment
 - beginTransaction()
 - add()
 - commit()



Dynamický fragment

aktivita môže mať viac fragmentov, ktoré spravuje *supportFragmentManager*

- pridávanie/rušenie/modifikácia fragmentu je vždy cez *FragmentManager*:

```
val ft = supportFragmentManager.beginTransaction()
val firstFragment = FirstFragment()

    val bundle = Bundle()
    bundle.putInt("init", 10) // posielanie argumentu/ov do fragmentu
    firstFragment.arguments = bundle

ft.add(R.id.frameLayout1, firstFragment, "tag1")
ft.add(R.id.frameLayout2, SecondFragment(), "tag2")
ft.commit()
```

vo fragmente získame context activity a hodnotu poslaných argumentov

```
override fun onAttach(context: Context) {
    super.onAttach(context)
    state = arguments?.getInt("init", 0)?:0 // získanie argumentu
    mainActivity = context as Updater
}
```




Dynamický fragment

```
val firstFragment = FirstFragment()
    val bundle = Bundle()
    bundle.putInt("init", 10) // posielanie argumentu/ov do fragmentu
    firstFragment.arguments = bundle
```

supportFragmentManager

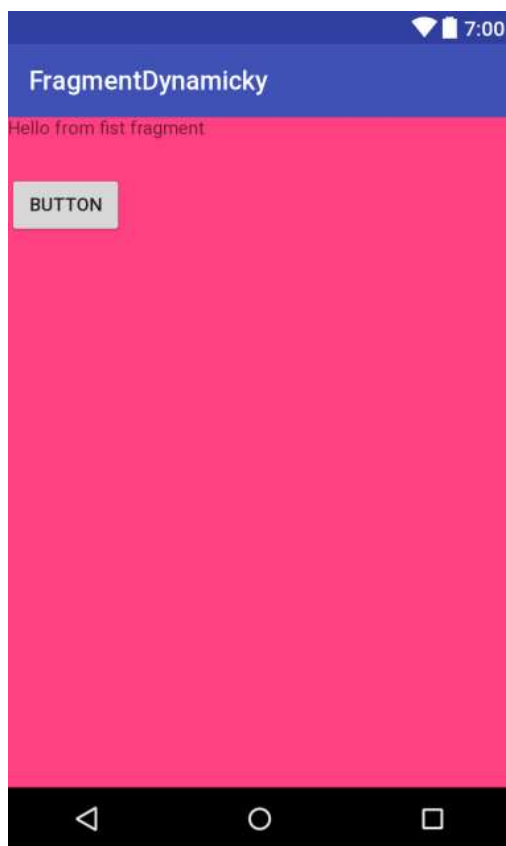
```
.beginTransaction()
.add(R.id.frameLayout1, firstFragment, "tag1")
.commit()

.remove(firstFragment)

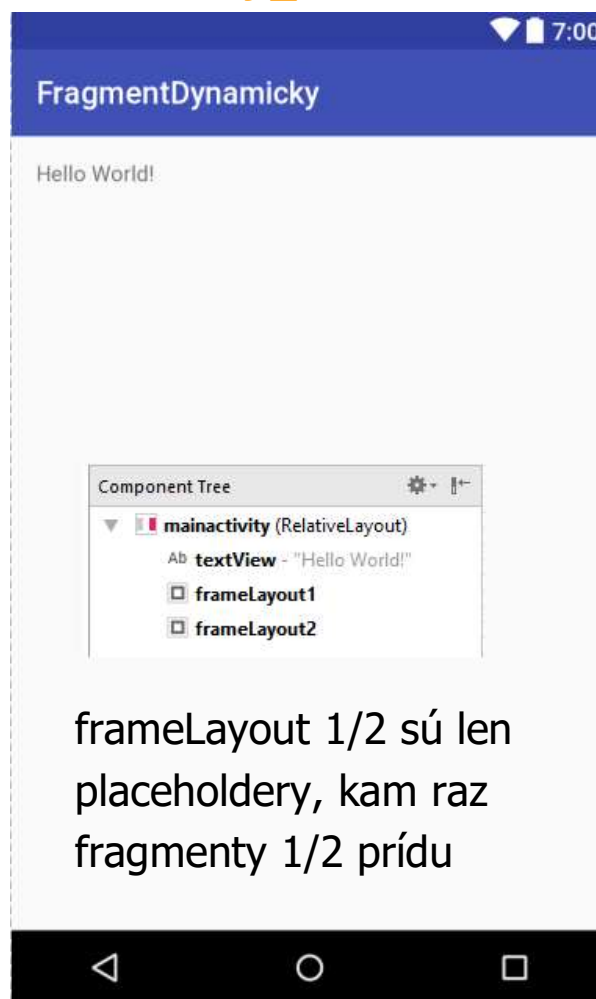
.replace(R.id.frameLayout1, firstFragment)
```

Dynamický fragment

fragment_first.xml



activity_main.xml



fragment_second.xml



Komunikácia medzi fragmentami



Nikdy nie fragment<->fragment, ale nepriamo cez ich spoločnú aktivitu !

MainActivity implementuje náš Update interface

```
interface Updater {  
    fun update(s:String); // medzi aktivitami chceme posielat' string  
}  
  
class MainActivity : FragmentActivity(), Updater {  
    override fun update(s:String) { ←  
        textView.text = s // TextView v bielej aktivite  
        val sfr =  
            supportFragmentManager // nájdi druhý/modrý fragment  
                .findFragmentById(R.id.frameLayout2) as SecondFragment  
        alebo  
            supportFragmentManager  
                .findFragmentByTag("tag2") as SecondFragment  
        sfr.setText(s)  
    } ←
```

Komunikácia medzi fragmentmi



Nikdy nie fragment<->fragment, ale nepriamo cez ich spoločnú aktivitu

FirstFragment volá náš update do main activity

```
class FirstFragment : Fragment() {  
    lateinit var mainActivity: Updater  
    private var state = 0  
  
    override fun onAttach(context: Context) {  
        super.onAttach(context)  
        state = arguments?.getInt("init", 0)?:0  
        mainActivity = context as Updater  
    }  
  
    override fun onActivityCreated(savedInstanceState: Bundle?) {  
        super.onActivityCreated(savedInstanceState)  
        button.setOnClickListener {  
            ← mainActivity.update("state:" + state++) }  
        }  
    }
```

Komunikácia medzi fragmentmi



Nikdy nie fragment<->fragment, ale nepriamo cez ich spoločnú aktivitu

SecondFragment

```
class SecondFragment : Fragment() {
```

```
    fun setFText(s: String) {  
        largeTextView.text = s  
    }
```



Komunikácia medzi fragmentmi

(sumarizácia)

```
class FirstFragment {
    var ma : Updater
    var state ...
    // API < 23
    onAttach(Activity a) {
        ma = a as Updater
    }
    // API >= 23
    onAttach(Context ctx) {
        ma = ctx as Updater
    }
    onActivityCreated(...){
        Button = ...
        ..onClick() {
            ...ma.update(state)
        }
    }
}
```

```
class
    MainActivity : Updater {

    fun update(state){
        f=supportFragmentManager().
        findFragmentById/Tag()
        f.setFText(state)
    }
}
```

```
interface Updater {
    fun update(state)
}
```

```
class
    SecondFragment {

    setFText(state){
        ...
    }
}
```

Ak by chceli komunikovať obojsmerne, tak **SecondF** tiež si musí odložiť referenciu na aktivitu a komunikovať cez ňu, referencia z fragmentu na jeho aktivitu je **getActivity()**

Komunikácia medzi fragmentmi

(nech zostane skryté, čo môže zostať skryté)

```
class FirstFragment {  
    interface Updater {  
        fun update(state)  
    }  
}
```

```
var ma : Updater  
var state ...  
  
onAttach(Activity a) {  
onAttach(Context a) {  
    ma = a as Updater  
}  
onActivityCreated(...){  
    Button = ...  
    ..onClick() {  
        ...ma.update(state)  
    }  
}
```

```
class MainActivity :  
    FirstFragment.Updater {  
  
    void update(state){  
        f=supportFragmentManager().  
            findFragmentById/Tag()  
        f.setFText(state)  
    }  
}
```

```
class  
    SecondFragment {  
  
    setFText(state){  
        ...  
    }  
}
```

Interface Updater súvisí len s FirstFragment a MainActivity, takže v niektorej z nich by mal byť ukrytý

Aktivita fragmentu



```
<LinearLayout
    android:orientation="vertical" >

    <Button
        android:id="@+id/fragment1"
        android:text="Show Fragment 1" />

    <Button
        android:id="@+id/fragment2"
        android:text="Show Fragment 2" />

    <FrameLayout // sem dynamicky vložíme jeden z fragmentov
        android:id="@+id/fragment_place"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

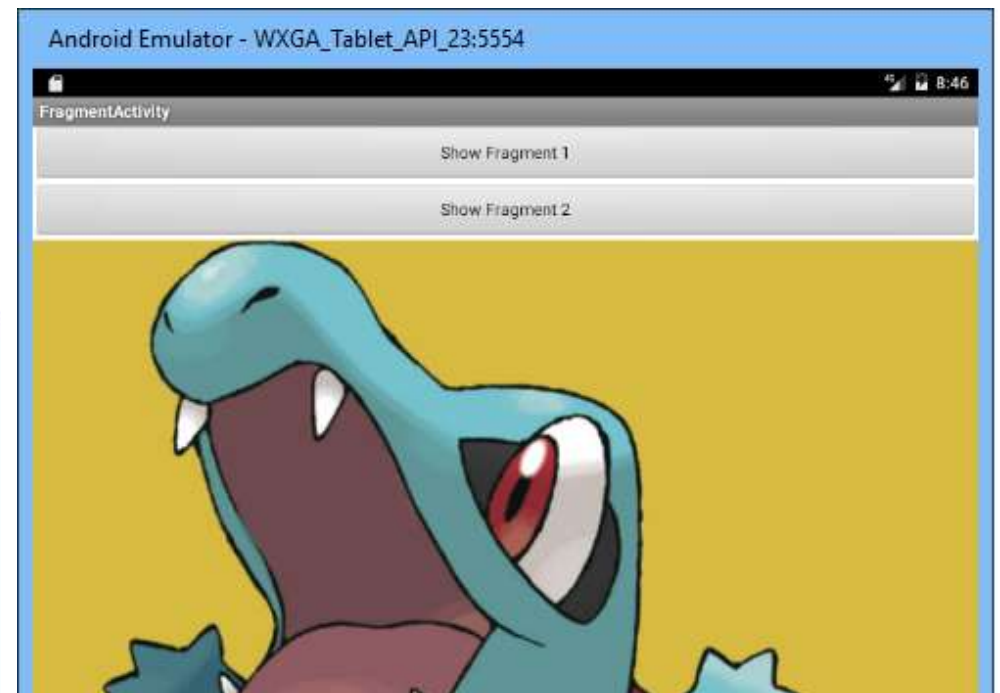

Fragmenty

```
<LinearLayout ...FragmentButtons
    android:orientation="horizontal"
    <Button
        android:text="Previous"
        android:id="@+id/prevBtn"/>
    <Button
        android:text="Next"
        android:id="@+id/nextBtn"
    />
    <Button
        android:text="Quit"
        android:id="@+id/quitBtn"
    />
```

```
<LinearLayout ...FragmentImage
    android:orientation="vertical">
    <ImageView
        android:id="@+id/imageView"
    />
</LinearLayout>
```

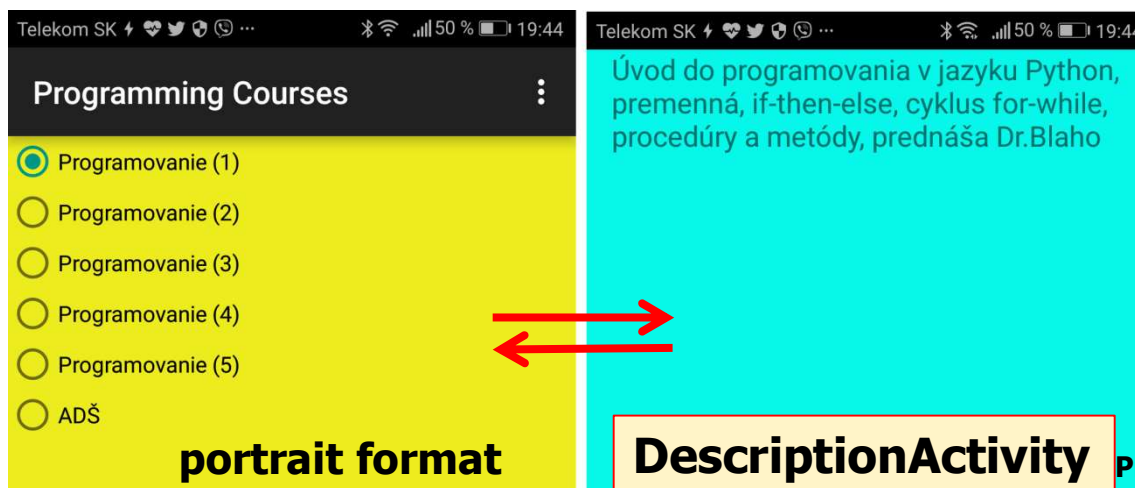
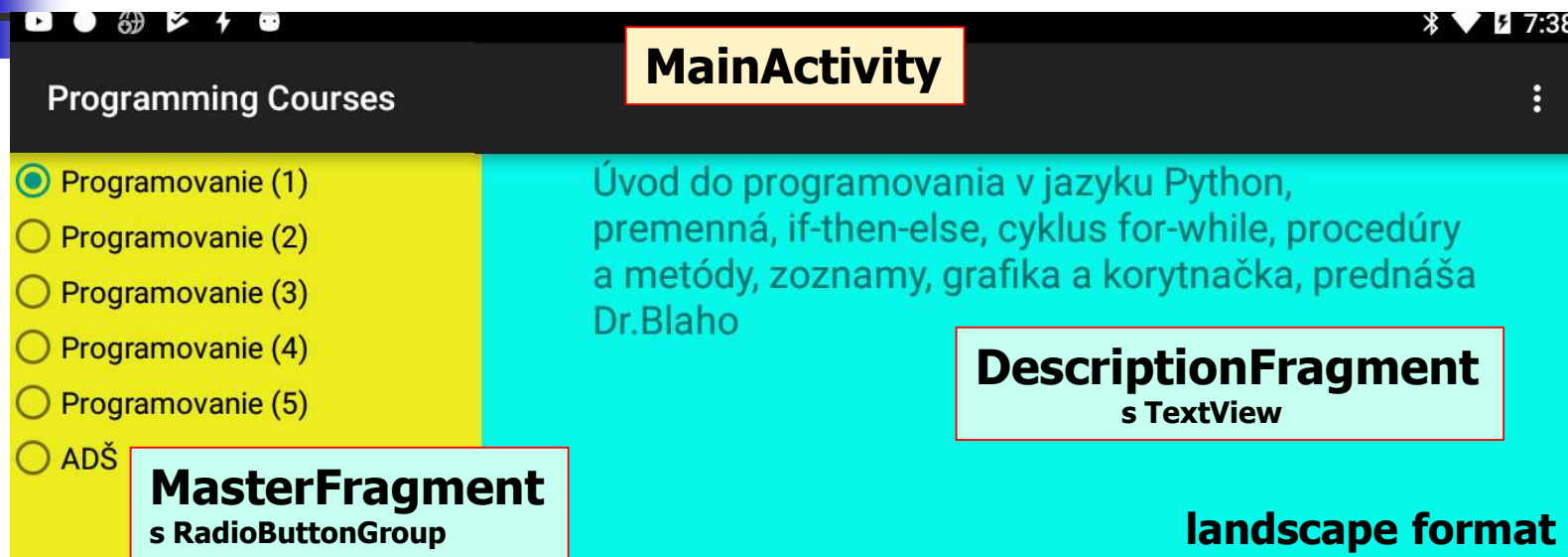


Projekt: [FragmentPikas.zip](#)



Master Detail

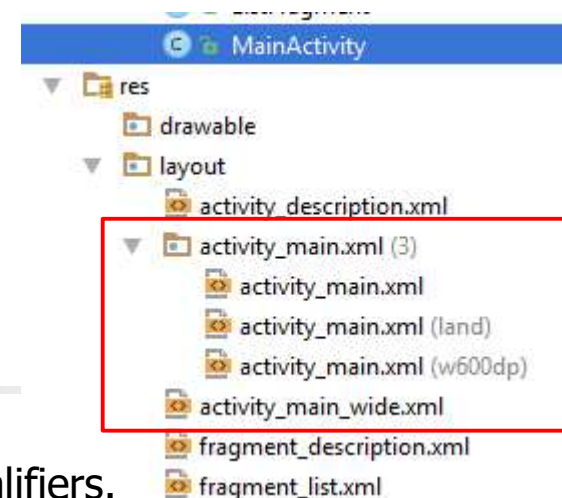
(MainActivity)



Master Detail

(MainActivity)

- aktivita/fragment môžu mať rôzne zobrazenia/layouts, napr. podľa orientácie, resp. rozlíšenia displaya, tzv. qualifiers.
- Kľúčom je Android Resource Directory, ak na zdrojáku aktivity klikneme pravým, pomôže vám vygenerovať špecializované layouts aktivity podľa zobraz. parametrov



activity_main_wide.xml

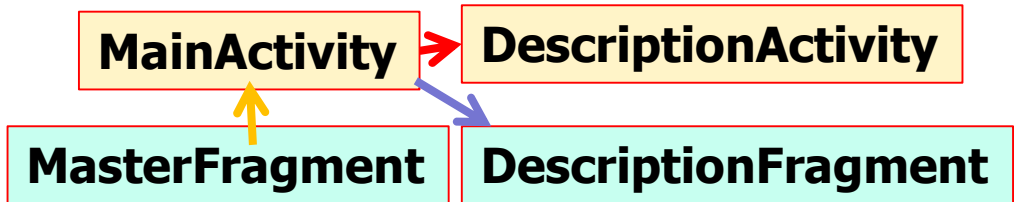
```
<LinearLayout ...  
    android:orientation="horizontal"  
    <fragment ...  
        tools:layout="@layout/fragment_list"/>  
    <fragment ...  
        tools:layout="@layout/fragment_description"/>  
</LinearLayout>
```

activity_main.xml

```
<LinearLayout ...  
    android:orientation="vertical"  
    <fragment  
        android:layout_width="match_parent"  
        android:id="@+id/fragmentTitles"/>  
</LinearLayout>
```

Master Detail

(MainActivity)



```
class MainActivity : AppCompatActivity(), ListFragment.Updater {  
  
    override fun update(selectedIndex: Int) {  
        val descriptionFragment = supportFragmentManager.  
            findFragmentById(R.id.fragmentDescription)  
            as? DescriptionFragment  
        if (descriptionFragment == null ||  
            !descriptionFragment.isVisible) {  
            if (!mCreating) {  
                val intent = Intent(this,  
                    DescriptionActivity::class.java)  
                intent.putExtra("selectedIndex", selectedIndex)  
                startActivity(intent)  
            }  
        } else {  
            descriptionFragment.setDetail(selectedIndex)  
        }  
    }  
}
```

Master Detail

(MasterFragment)

MainActivity

DescriptionActivity

MasterFragment

DescriptionFragment

```
class ListFragment:Fragment(),RadioGroup.OnCheckedChangeListener {
    internal interface Updater {
        fun update(selectedIndex: Int)
    }
    override fun onCheckedChanged(group:RadioGroup,checkedId:Int) {
        var selectedIndex = -1
        when (checkedId) {
            R.id.prog1ID -> selectedIndex = 0
            R.id.prog2ID -> selectedIndex = 1
            R.id.prog3ID -> selectedIndex = 2
            R.id.prog4ID -> selectedIndex = 3
            R.id.prog5ID -> selectedIndex = 4
            R.id.adsID   -> selectedIndex = 5
        }
        val listener = activity as Updater
        → listener.update(selectedIndex)
    }
}
```

MainActivity

DescriptionActivity

MasterFragment

DescriptionFragment



Master Detail

(DescriptionFragment)

```
class DescriptionFragment : Fragment() {  
    lateinit var tv: TextView  
    override fun onCreateView(inflater: LayoutInflater,  
                              container: ViewGroup?,  
                              savedInstanceState: Bundle?): View? {  
        val view = inflater.inflate(  
            R.layout.fragment_description,  
            container, false)  
        tv = view.findViewById(R.id.descriptionID) as TextView  
        return view  
    }  
}
```

→

```
fun setDetail(index: Int) {  
    val descriptions =  
        resources.getStringArray(  
            R.array.course_full_descriptions)  
    val course = descriptions[index]  
    tv.text = course  
}
```

```
<string-array  
    name="course_full_descriptions">  
        <item>@string/prog1Detail</item>  
        <item>@string/prog2Detail</item>  
        <item>@string/prog3Detail</item>  
        <item>@string/prog4Detail</item>  
        <item>@string/prog5Detail</item>  
        <item>@string/adsDetail</item>  
    </string-array>
```

MainActivity

DescriptionActivity

MasterFragment

DescriptionFragment

Master Detail

(DescriptionActivity)

```
class DescriptionActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_description)  
        val intent = intent  
        val selectedIndex = intent.getIntExtra("selectedIndex", -1)  
        if (selectedIndex != -1) {  
            val descriptionFragment = supportFragmentManager  
                .findFragmentById(R.id.fragment_description)  
                as DescriptionFragment  
            descriptionFragment.setDetail(selectedIndex)  
        }  
    }  
}
```

MasterDetail

(veľké rozlíšenie)

nechajte AS vygenerovať M/D projekt, a pokúste sa pochopiť kód

itemListActivity

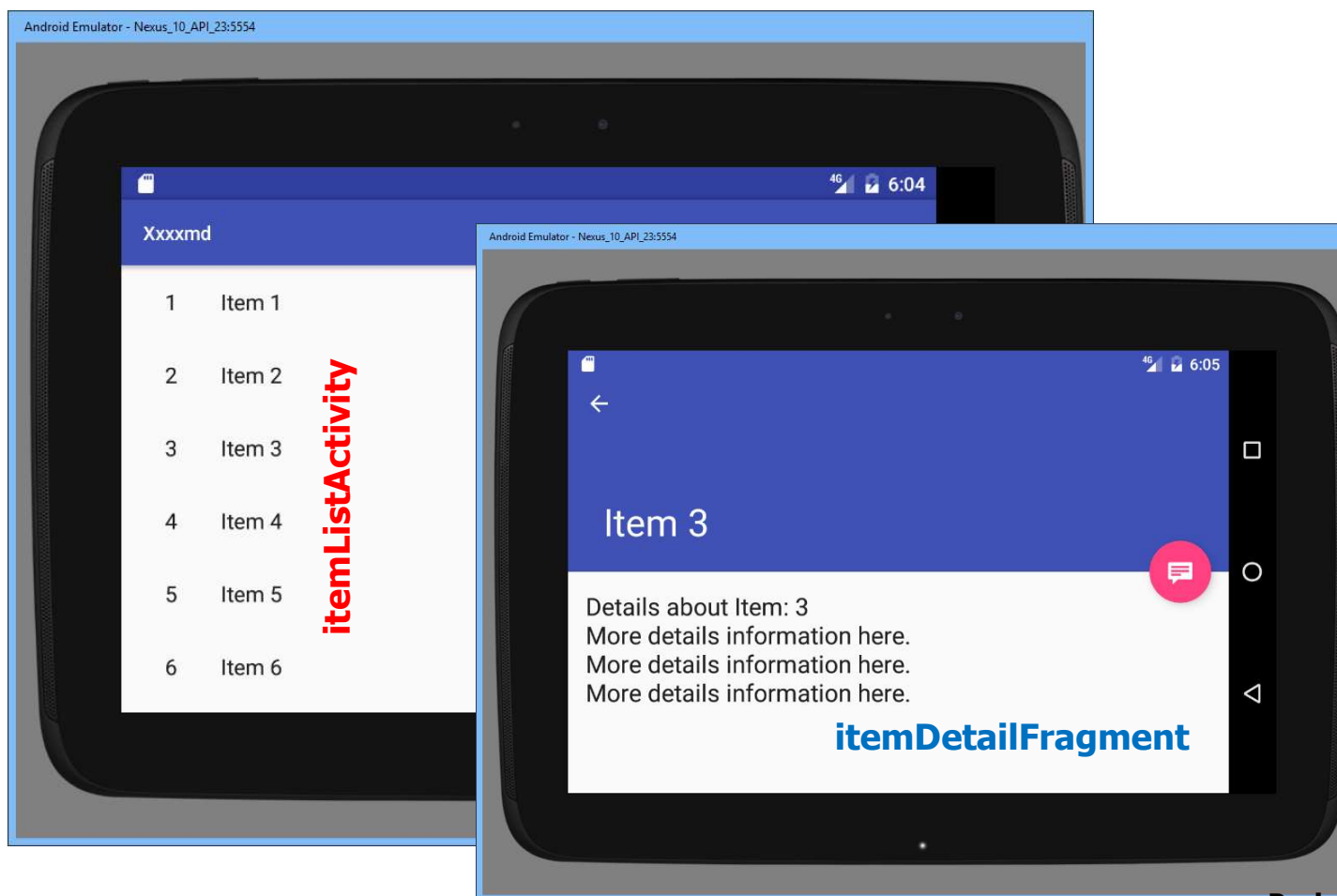


: MasterDetail.zip

MasterDetail

(malé rozlíšenie)

pre iné rozlíšenie dostanete iný look

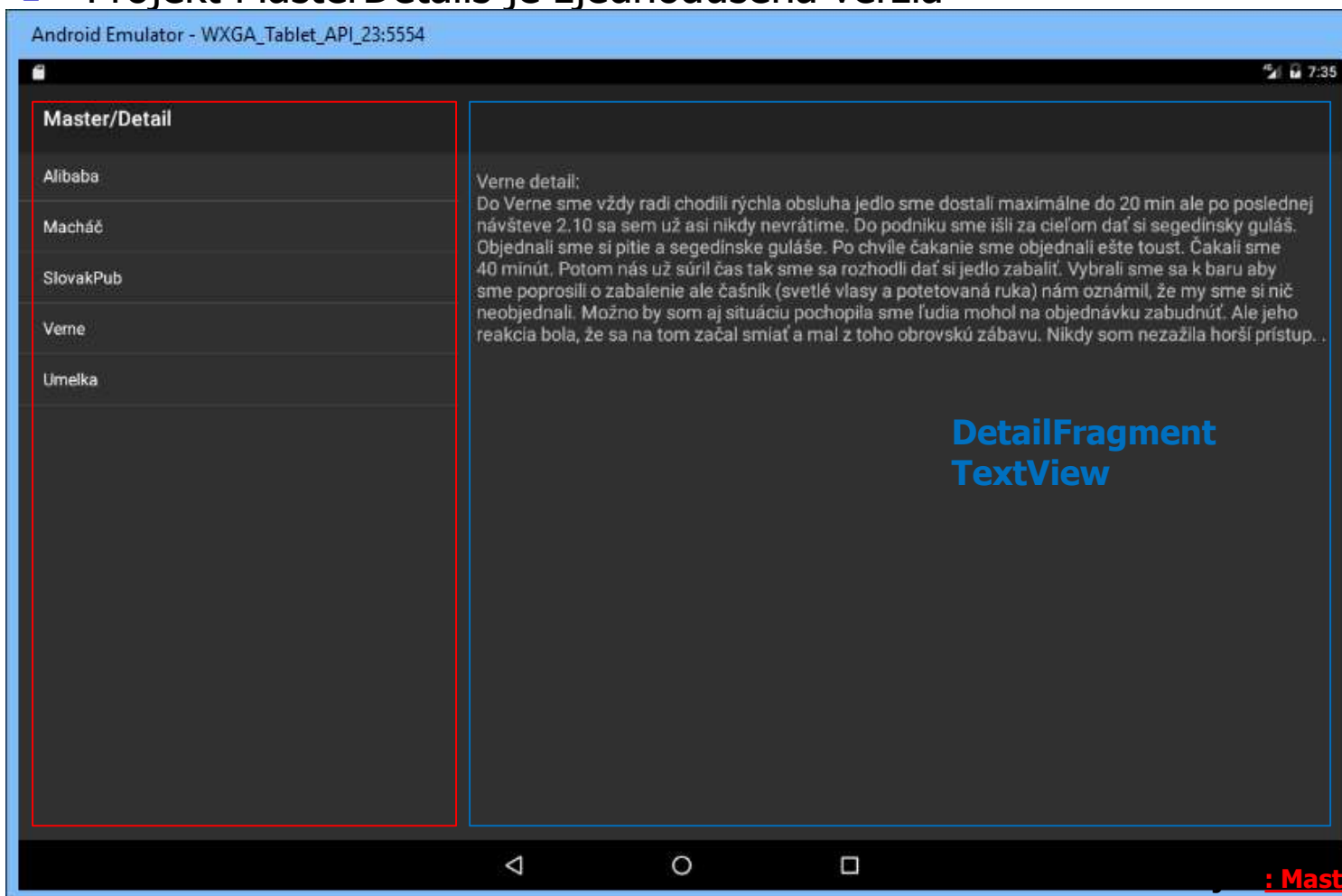


MasterDetails

(veľké rozlíšenie)

- Projekt MasterDetails je zjednodušená verzia

MasterFragment
ListView



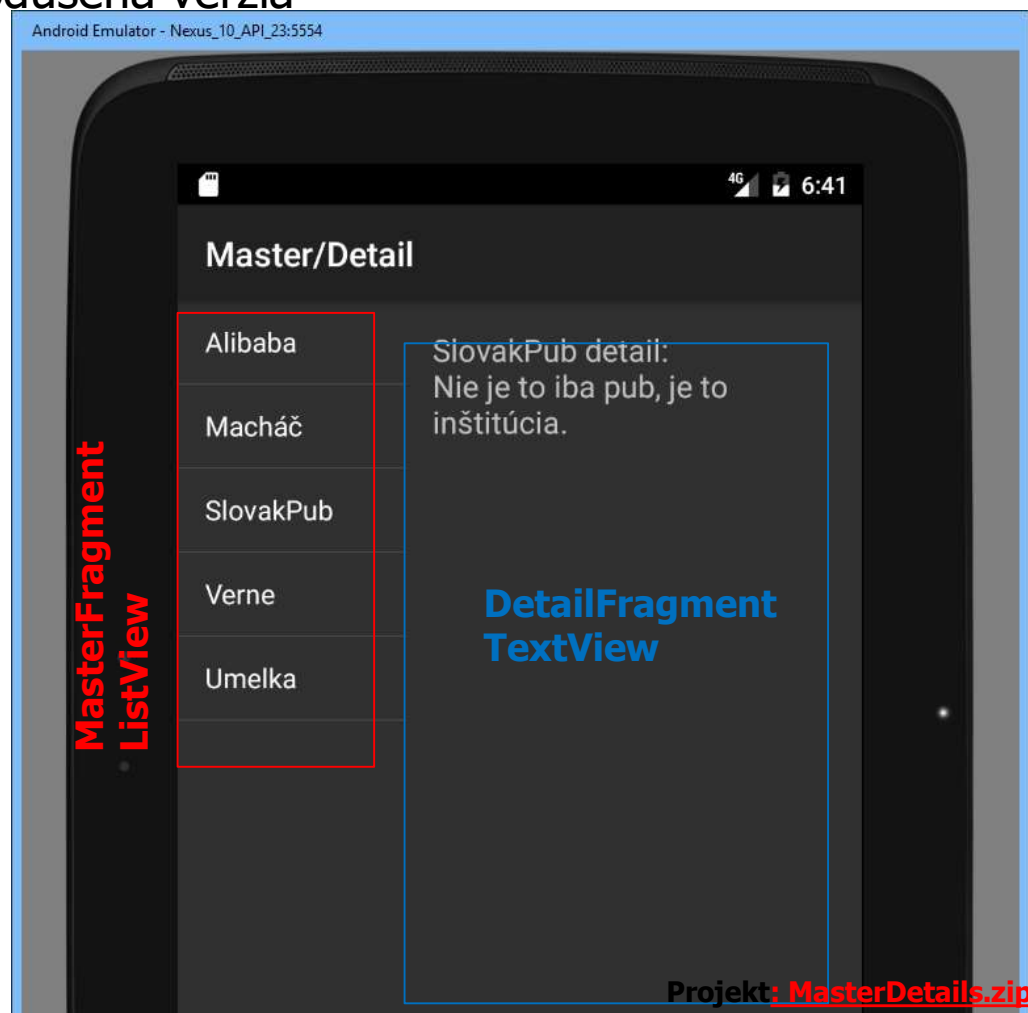
MasterDetails

(malé rozlíšenie)

Projekt MasterDetails je zjednodušená verzia

Problémy:

- pri zmene orientácie aktivity/fragmentu príde k strate dát/nastavení aktivity/fragmentu
- pri menšom rozlíšení by sme privítali iný layout fragmentov v móde landscape/portrait





Perzistencia dát fragmentu

- potrebujeme uložiť index v ListView, na ktorom sme stáli do Bundle savedInstanceState
- pri onCreateView fragmentu opätovne obnovíme index zo savedInstanceState

```
class DetailFragment : Fragment() {  
    private var index = -1  
  
    // toto sa zavolá pred restartom aktivity/fragmentu  
    override fun onSaveInstanceState(outState: Bundle) {  
        super.onSaveInstanceState(outState)  
        outState.putInt("INDEX", index)  
    }  
  
    // bundle outstate sa odpamätá až do event.volania/reštartu a/f  
    override fun onCreateView(inflater: LayoutInflater,  
        container: ViewGroup?, savedInstanceState: Bundle?): View? {  
        index = savedInstanceState?.getInt("INDEX")?:-1  
        return  
            inflater.inflate(R.layout.detail_view, container, false)  
    }  
    // bundle je dictionary resp. HashMap<String, Object>
```



Argumenty fragmentu

(fragment môže dostať argumenty od aktivity – tiež Bundle)

```
public class DetailFragment extends Fragment {  
    // fragment môže dostať bundle argumentov aj od aktivity  
    override fun onStart() {  
        super.onStart()  
        val args = arguments  
        if (args != null) {  
            updateDetailView(args.getInt("INDEX"))  
        } else if (index != -1) {  
            updateDetailView(index)  
        }  
    }  
}
```

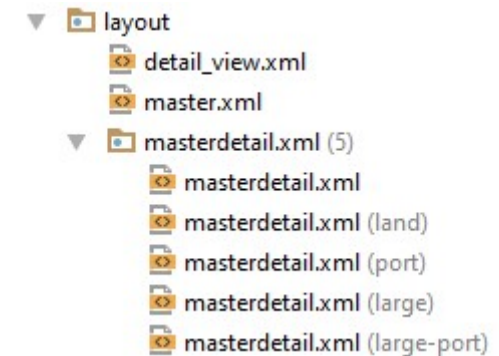
Bundle je
HashMap<String, Object>

// Pri vytvorení fragmentu, ak aktivita chce odovzdať bundle argumentov vznikajúcemu fragmentu

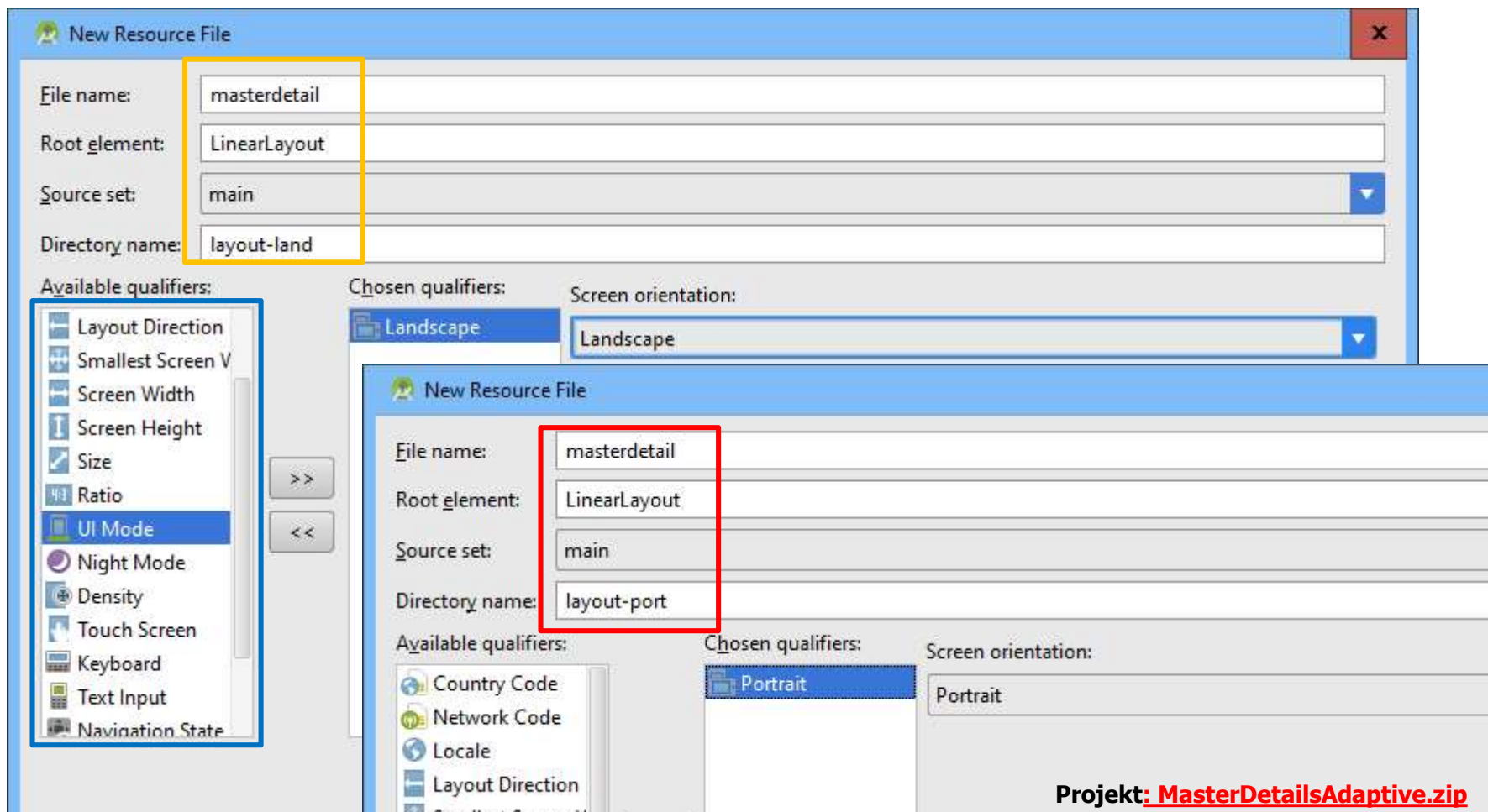
```
val newFragment = DetailFragment()  
val args = Bundle()  
args.putInt("INDEX", index)  
newFragment.arguments = args
```

Adaptívny layout

Ak pre rôzne rozlíšenia a orientácie display (...qualifiers) chceme iné layouty

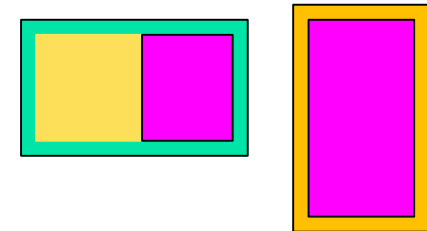


qualifiers



Projekt: [MasterDetailsAdaptive.zip](#)

Flexibilný layout

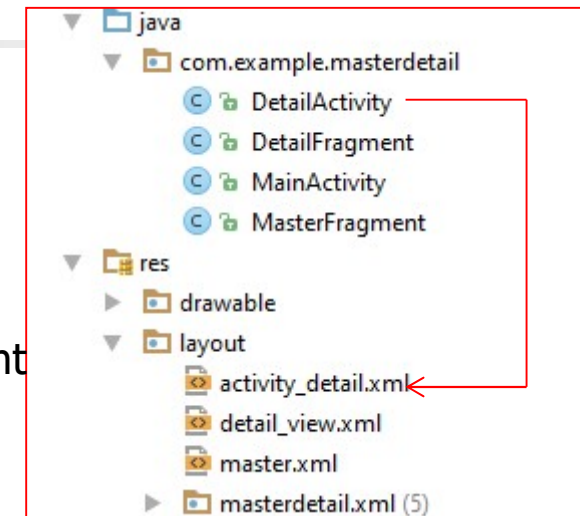


Landscape

- MainActivity
 - First/MasterFragment
 - Second/DetailFragment

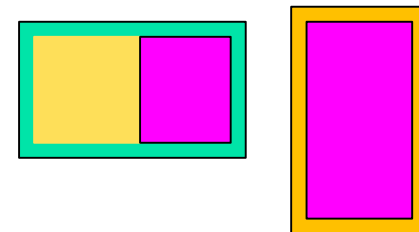
Portrait

- MainActivity
 - First/MasterFragment
- DetailActivity
 - Second/DetailFragment



```
public void update(int index) {  
    int orientation=getResources().getConfiguration().orientation;  
    if (orientation== Configuration.ORIENTATION_LANDSCAPE) {  
        ... to, čo sme robili predtým  
    } else { // Configuration.ORIENTATION_PORTRAIT  
        Intent in = new Intent(this, DetailActivity.class);  
        in.putExtra("YINDEX",index);  
        startActivity(in);  
    }  
}
```

Flexibilný layout



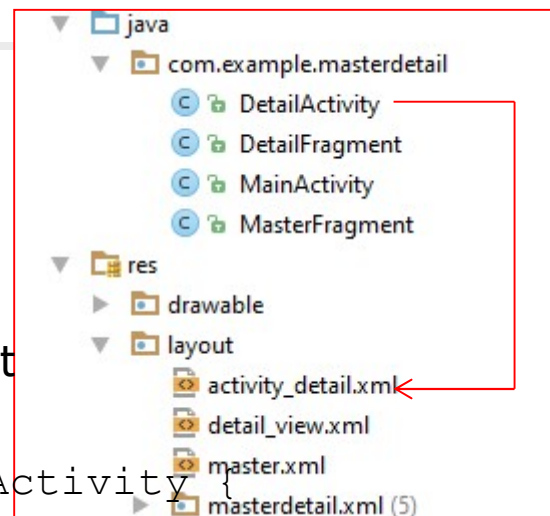
Landscape

- MainActivity
 - First/MasterFragment
 - Second/DetailFragment

Portrait

- MainActivity
 - First/MasterFragment
- DetailActivity
 - Second/DetailFragment

```
public class DetailActivity extends FragmentActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
        Intent in = getIntent();
        int yndex = in.getIntExtra("YNDEX", 0);
        FragmentManager fm = getSupportFragmentManager();
        DetailFragment detailfr =
            (DetailFragment) fm.findFragmentById(R.id.detail_fragment);
        if (detailfr != null) {
            detailfr.updateDetailView(yndex);
        }
    }
}
```



R.layout.yes_no_layout

Do you really want to quit ?

YES

NO

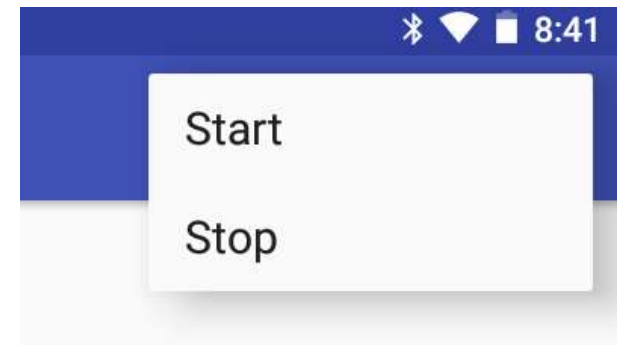
Dialog Fragment

(podtrieda Fragment)

```
class YesNoDialog : DialogFragment() {
    lateinit var updater : Updater
    override fun onAttach(activity: Activity) {
        super.onAttach(activity)
        updater = activity as Updater
    }
    override fun onCreateView(inflater: LayoutInflater,
                               container: ViewGroup?,
                               savedInstanceState: Bundle?): View? {
        isCancelable = false // neda sa zrusit dialog
        val view = inflater.inflate(R.layout.yes_no_layout,
                                    container, false)
        (view.findViewById(R.id.yesBtn) as Button)
            .setOnClickListener {
                updater.sendMessage("yes pressed")
                dismiss() // zmizne dialog
            }
        return view
    }
}
```

Dialog Fragment

(volanie v MainActivity)

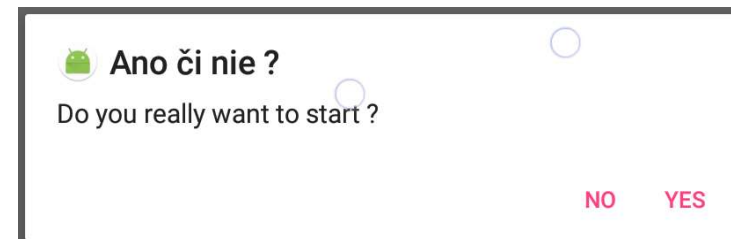


```
class MainActivity : AppCompatActivity(), YesNoDialog.Updater {  
  
    override fun onOptionsItemSelected(item: MenuItem): Boolean {  
        when (item.itemId) {  
            ...  
            R.id.StopID -> {  
                YesNoDialog().show(supportFragmentManager, "Yes or No ?")  
                return true  
            }  
        }  
        return super.onOptionsItemSelected(item)  
    }  
  
    override fun sendMessage(msg: String) {  
        if (msg == "yes pressed")  
            this@MainActivity.finish()  
    }  
}
```

Ak bolo Yes na really want?

Alert Dialog

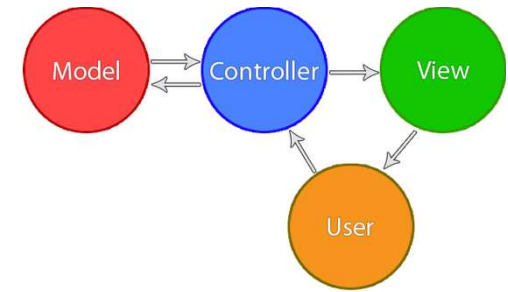
(musí to ist' aj jednoduchšie – varenie z polotovarov)



```
R.id.StartID -> {  
    val builder = AlertDialog.Builder(this@MainActivity)  
    builder.setTitle("Ano či nie ?")  
        .setMessage("Do you really want to start ?")  
        .setIcon(R.mipmap.ic_launcher_round)  
        .setCancelable(false)  
        .setPositiveButton(R.string.yesText)  
            { dialogInterface, i -> Toast.makeText(this@MainActivity,  
                "Start it", Toast.LENGTH_SHORT).show() }  
        .setNegativeButton(R.string.noText)  
            { dialogInterface, i -> Toast.makeText(this@MainActivity,  
                "DO NOT Start it", Toast.LENGTH_SHORT).show() }  
        .setNeutralButton(R.string.whoKnowsText)  
            { dialogInterface, i -> Toast.makeText(this@MainActivity,  
                "DO NOTHING", Toast.LENGTH_SHORT).show() }  
    val alertDialog = builder.create()  
    alertDialog.show()  
    return true  
}
```

Architektonický *mess*

(MVC)

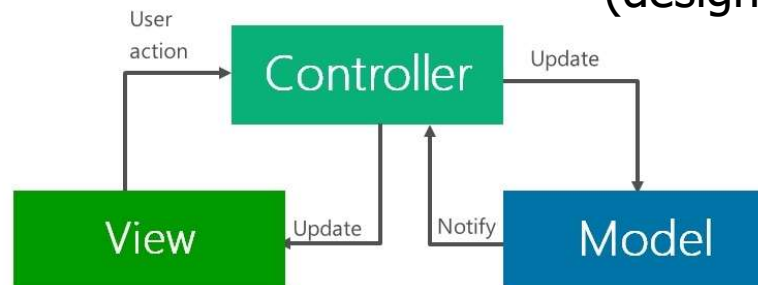


- vzniká, ak vizuálne komponenty (Views) sú zviazané s dátovými objektami a opačne

```
prev.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        i++;  
        i %= imgs.length;  
        iv.setImageDrawable(imgs[i]);  
    }  
});
```

- preto sa pri návrhu GUI používajú návrhové vzory, Model-View-Controller (design patterns)

3 Tier Architecture - iOS



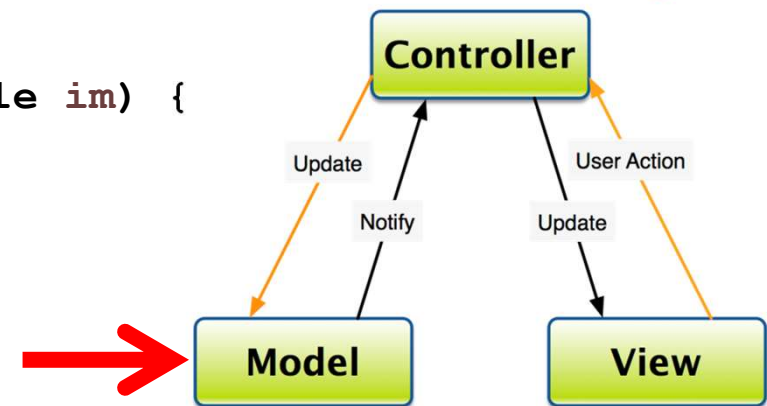
- motto: the architecture of most Android-apps is a mess.

<http://doridori.github.io/Android-Architecture-MV%3F/#sthash.SiE5eude.IQq3XhmU.dpbs>

Model View Controller (MVC)

(model – len data, netuší nič o ich prezentácii)

```
public class Model extends Observable {  
    int indx = 0;           // actual picture on the screen  
    ArrayList<Drawable> list = new ArrayList<Drawable>(); // all pics  
  
    public void addDrawableImage(Drawable im) {  
        list.add(im);  
    }  
  
    public Drawable getDrawable() {  
        return list.get(indx);  
    }  
  
    public void nextValue() {  
        indx++;  
        indx %= list.size();  
        setChanged();  
        notifyObservers();  
    }  
  
    public void prevValue() {  
        indx--;  
        if (indx < 0)  
            indx = list.size() - 1;  
        setChanged();  
        notifyObservers();  
    }  
}
```



Model View Controller (MVC)

(controller – komunikuje medzi modelom a view)

```
public class Controller extends ... implements Observer {
```

```
mModel = new Model();
```

```
mModel.addObserver(this);
```

```
mModel.addDrawableImage(getResources().getDrawable(R.drawable.pok0));
```

```
mModel.addDrawableImage(getResources().getDrawable(R.drawable.pok1));
```

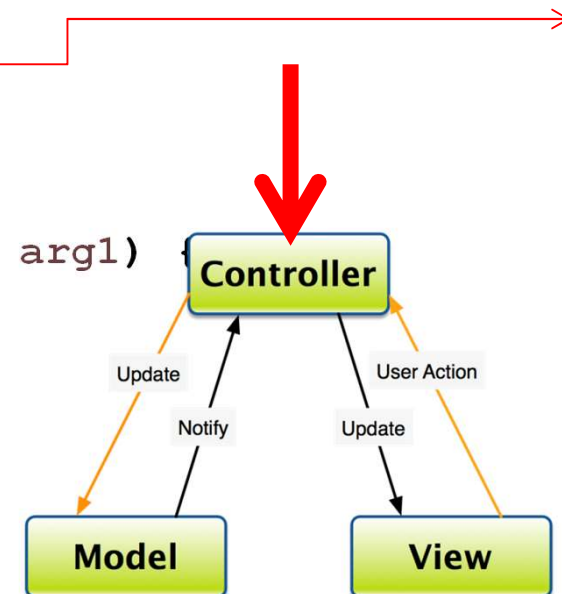
```
mView = new myView(this);
```

```
@Override
```

```
public void update(Observable arg0, Object arg1) {
```

```
    mView.update(mModel.getDrawable());
```

```
}
```



Model View Controller (MVC)

(view)

```
public class myView {  
    final Controller controller;  
    ImageView iv;  
    Button prev, next;  
  
    public myView(Controller c) {  
        this.controller = c;  
        iv = (ImageView)mainActivity.findViewById(R.id.imageView1);  
        Button prev = (Button)mainActivity.findViewById(R.id.prevBtn);  
        prev.setOnClickListener(new OnClickListener() {  
            @Override  
            public void onClick(android.view.View v) {  
                controller.mModel.prevValue();  
            }  
        });  
        public void update(android.graphics.drawable.Drawable im) {  
            iv.setImageDrawable(im);  
        }  
    }  
}
```

