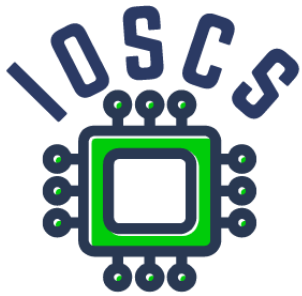


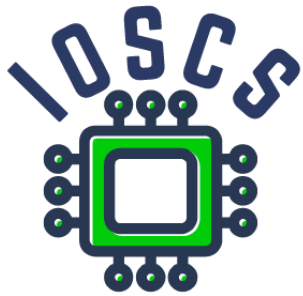
Project: Innovative Open Source Courses for Computer Science

Vývoj mobilních aplikací
Prezentace

Radosław Maciaszczyk
West Pomeranian University of Technology in Szczecin

30.05.2021

Innovative Open Source Courses for Computer Science



This teaching material was written as one of the outputs of the project "Innovative Open Source Courses for Computer Science", funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564. The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland) and is implemented in partnership with Mendel University in Brno (Czech Republic) and University of Žilina (Slovak Republic). The project implementation timeline is September 2019 to December 2022.

Project information

Project was implemented under the Erasmus+.
Project name: **"Innovative Open Source courses for Computer Science curriculum"**
Project nr: **2019-1-PL01-KA203-065564**
Key Action: **KA2 – Cooperation for innovation and the exchange of good practices**
Action Type: **KA203 – Strategic Partnerships for higher education**

Consortium
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE
MENDELOVA UNIVERZITA V BRNE
ZILINSKA UNIVERZITA V ZILINE

Co-funded by the
Erasmus+ Programme
of the European Union

MOBILE APPLICATION DEVELOPMENT

Úvod

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Co je mobilní zařízení

Hlavní rozdíly ?

- PROCESOR

Co je mobilní zařízení

Hlavní rozdíly ?

- PROCESOR
- Baterie

Co je mobilní zařízení

Hlavní rozdíly ?

- PROCESOR
- Baterie
- Senzory

Co je mobilní zařízení

Hlavní rozdíly ?

- PROCESOR
- Baterie
- Senzory
- Připojení

Co očekáváte od mobilních zařízení

- ??

Co očekáváte od mobilních zařízení

- ??
- Kolik zařízení máte?

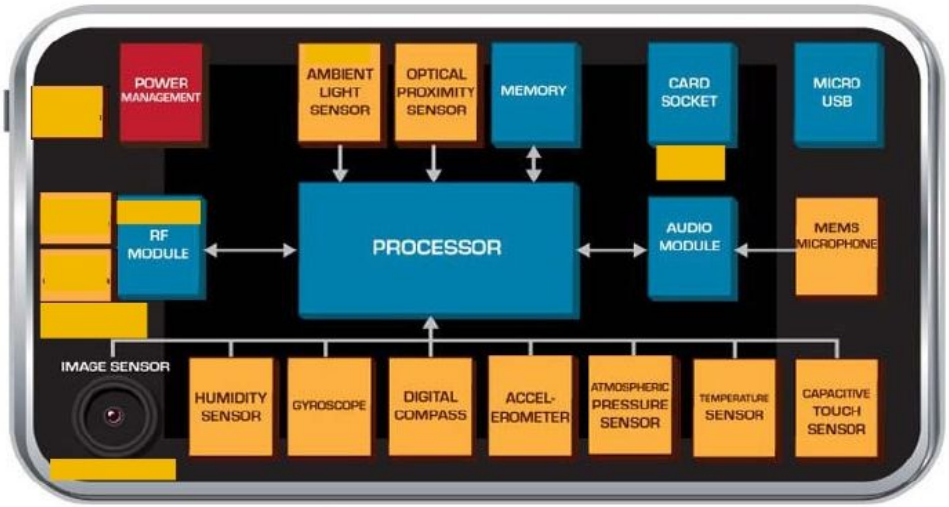
Co očekáváte od mobilních zařízení

- ??
- Kolik zařízení máte?
- V budoucnu - jedno zařízení mnoho aplikací

Co očekáváte od mobilních zařízení

- ??
- Kolik zařízení máte?
- V budoucnu - jedno zařízení mnoho aplikací
- ~~In future~~ ne TEĎ

Zařízení pro vývojáře



http://pl.scribd.com/doc/98309084/
Fusing-Sensors-Into-Mobile-OSes-Innovative-Use-Cases-Submitted-5-23-12

Hlavní prodejci

- Květen 2021 - gs.statcounter.com/vendor-market-share/mobile
- Samsung 27,84 %
- Apple 26,47 %
- Xioami 10,62 %
- Huawei 8,85 %
- Oppo 5,39 %

Mobilní operační systém

- Květen 2021 - `gs.statcounter.com/os-market-share/mobile/worldwide`
- Android 72,72 %
- iOS 26,47 %
- Samsung 0,4 %
- KaiOS 0,17 %
- Neznámý 0,17 %

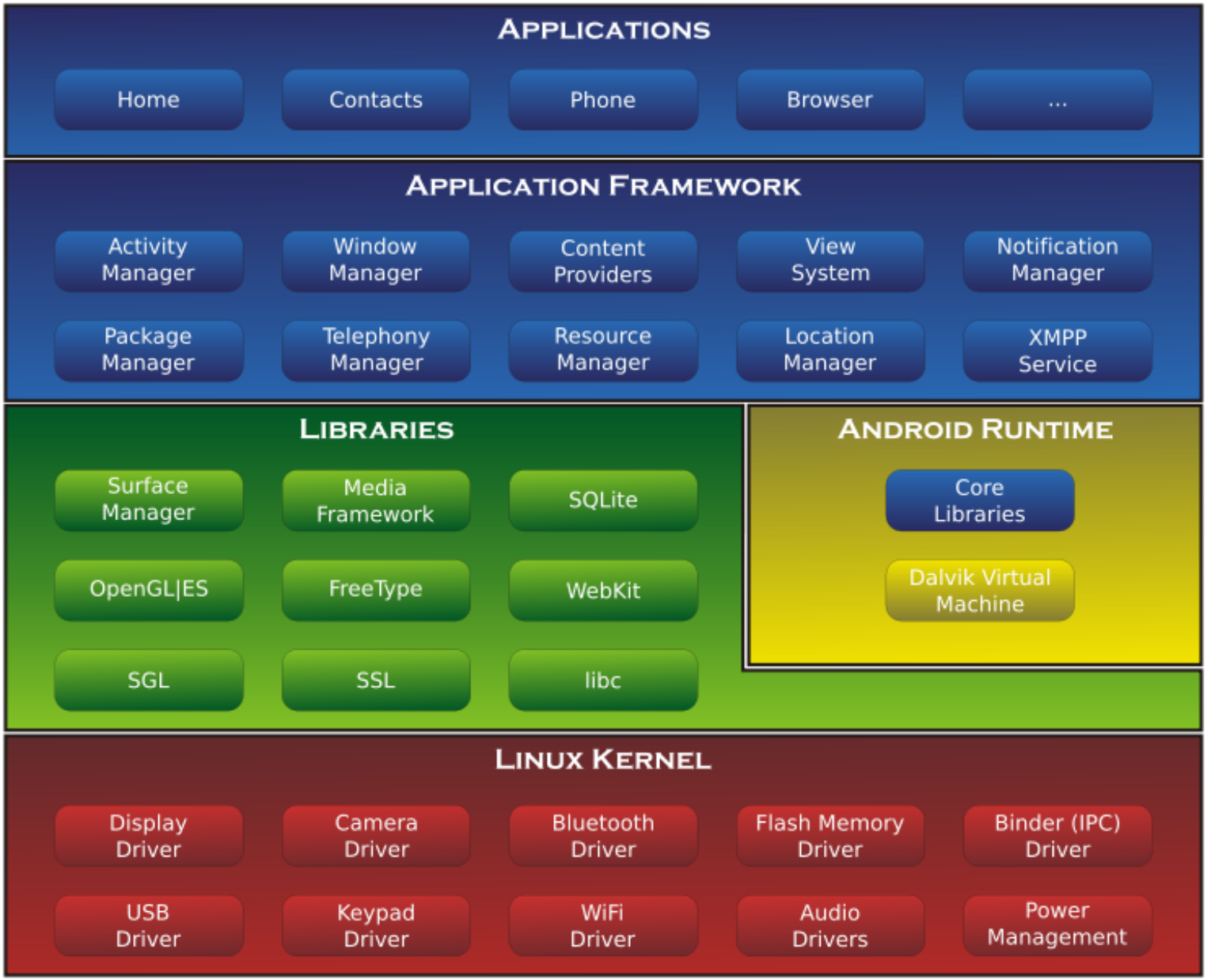
Co je Android?

- Zásobník softwaru s otevřeným zdrojovým kódem, který zahrnuje
  - Operační systém
  - Middleware
  - Klíčové mobilní aplikace (webový prohlížeč, PIM, SMS, e-mail ...).
  - Knihovny API pro psaní mobilních aplikací.
- Open-source vývojová platforma pro tvorbu mobilních aplikací
- Operační systém založený na Linuxu
- Obecně pro mobilní zařízení s dotykovou obrazovkou, jako jsou chytré telefony. a tablety

Kde je Android

- Telefony
- Tablety
- Televizory
- STB - set-top-box
- roboti
- budou v autech
- budou v zábavních systémech za letu v letadlech
- Android je všude !

Softwarový zásobník pro Android



Základy aplikace

- Aplikace pro Android jsou napsány v programovacím jazyce Java nebo Kotlin. - Obecně
- Nástroje Android SDK zkompilují kód (spolu s veškerými daty a prostředky do balíčku Android, archivního souboru s koncovkou .apk.
- Veškerý kód v jednom souboru .apk je považován za jednu aplikaci. a je to soubor, který zařízení s operačním systémem Android používají k instalaci aplikace.

Základy aplikace

- Aplikace pro Android jsou napsány v programovacím jazyce Java nebo Kotlin. - Obecně
- Nástroje Android SDK zkompilují kód (spolu s veškerými daty a prostředky do balíčku Android, archivního souboru s koncovkou .apk.
- Veškerý kód v jednom souboru .apk je považován za jednu aplikaci. a je to soubor, který zařízení s operačním systémem Android používají k instalaci aplikace.

Základy aplikace

- Aplikace pro Android jsou napsány v programovacím jazyce Java nebo Kotlin. - Obecně
- Nástroje Android SDK zkompilují kód (spolu s veškerými daty a prostředky do balíčku Android, archivního souboru s koncovkou .apk.
- Veškerý kód v jednom souboru .apk je považován za jednu aplikaci. a je to soubor, který zařízení s operačním systémem Android používají k instalaci aplikace.

Aplikace pro Android

- Aplikace Android žije ve vlastním bezpečnostním sandboxu,
- Andorid OS je víceuživatelský linuxový systém,
- Každá aplikace je jiný uživatel (ve výchozím nastavení).
- Každý proces má svůj vlastní virtuální počítač (VM).
- Ve výchozím nastavení běží každá aplikace ve vlastním linuxovém procesu.
- Systém Android spustí proces, když je třeba, aby některá z komponent aplikace byla spuštěna. spustit
- Vypne proces, když už není potřeba nebo když systém musí obnovit paměť pro jiné aplikace.

“Principle of least privilege”

- Každá aplikace má ve výchozím nastavení přístup pouze ke komponentám, které potřebuje ke své práci, a ne více.
- Aplikace nemůže přistupovat k částem systému, pro které není určena. uděleno oprávnění
- Otázka: Jak sdílet data s ostatními aplikacemi ?

Sdílení dat s jinými aplikacemi

- Sdílejí stejné ID uživatele Linuxu a v takovém případě mají vzájemný přístup k souborům.
- Vytvořit nebo použít zprostředkovatele obsahu
- Ukládat data na SDCard
- Důležité: Aplikace může požadovat oprávnění k přístupu k datům zařízení, jako např. uživatelských kontaktů, zpráv SMS, připojitelného úložiště (karty SD), fotoaparátu, Bluetooth a další

Komponenty aplikace

Čtyři různé typy aplikačních komponent. Každý typ slouží má odlišný účel a odlišný životní cyklus.

- Activities
- Services
- Content providers
- Broadcast receivers

Activity

- Představuje jednu obrazovku s uživatelským rozhraním.
- Více činností v jedné aplikaci
- Všechny aktivity mají vlastní životní cyklus
- Aktivita je implementována jako podtřída třídy Activity
- Např. aplikace Mail: čtení pošty, skládání pošty...

Service

- Služba běží na pozadí a provádí dlouhodobé operace nebo práci pro vzdálené procesy.
- Služba neposkytuje uživatelské rozhraní
- má vlastní životní cyklus
- Např. hudební aplikace: služba může na pozadí přehrávat hudbu, zatímco uživatel je v jiné aplikaci.
-

Content providers

- Zprostředkovatel obsahu spravuje sdílenou sadu dat aplikace
- Ostatní aplikace mohou data upravovat bez znalosti podrobné architektury
- Dobrou praxí je také používat poskytovatele obsahu jako interní nástroj.
- Zprostředkovatelé systémového obsahu ukládají informační řádky Kontakt, Fotografie, Video...

Broadcast receivers

- Mechanismus pro odesílání nebo přijímání zpráv vysílaných systémem Android a jinými aplikacemi systému Android
- Podobný návrhovému vzoru "publish-subscribe".
- Při odesílání broadcastové zprávy se ostatní aplikace musí přihlásit k odběru tohoto typu zprávy.
- Existuje mnoho systémových zpráv, např. systém pošle zprávu po dokončení startu systému, baterie je vybitá...

Intent

- Jedná se o samostatný mechanismus, který používáme ke spuštění tří ze čtyř typů komponent - aktivit, služeb a přijímačů vysílání.
- Se záměrem posílat informace o akcích a datech
- V závislosti na komponentě definujeme akce různě.

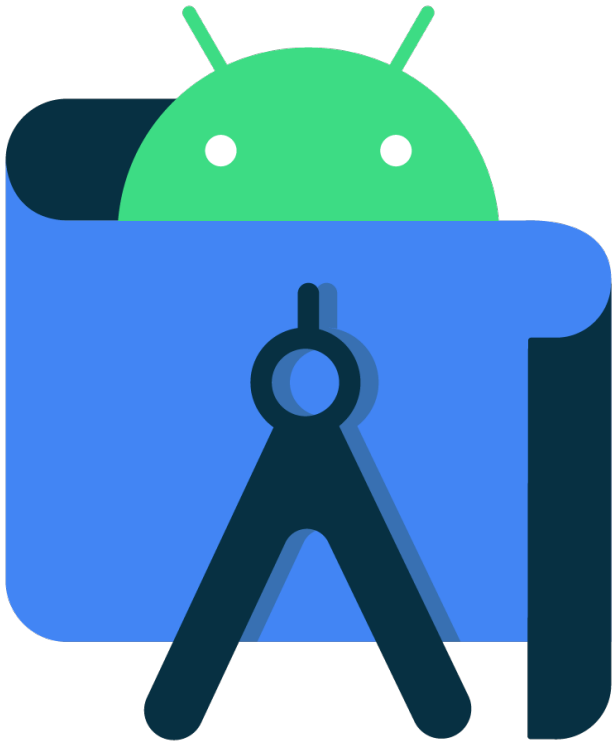
The Manifest File

- Všechny informace o kompomentu musí existovat v souboru *AndroidManifest.xml*.
- Zvlášť musíme zveřejnit informace o hlavní činnosti
- Musíme také zveřejnit informace o povoleních
- Aplikace vyžaduje
- Vyhlásit minimální úroveň API
- Deklarovat použité nebo požadované hardwarové a softwarové funkce (fotoaparát, služby bluetooth nebo vícedotykový displej atd.)
- Knihovny API (jiné než API rámce Android), například knihovna Google Maps.

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest ... >
3          <uses-permission android:name="android.permission. ..."/>
4      <application android:icon="@drawable/app_icon.png" ... >
5          <activity android:name="com.example.project.ExampleActivity"
6                    android:label="@string/example_label" ... >
7          </activity>
8          <service>
9          </service>
10         <receiver>
11         </receiver>
12         <provider>
13         </provider>
14         ...
15     </application>
16 </manifest>
```

Živý příklad - Hello World



Úvod do aplikace Android Studio

- https://www.youtube.com/watch?v=K2dodTXARqc
- https://www.youtube.com/user/androiddevelopers/

Řešení běžných problémů při vývoji systému Android

- http://www.vogella.com/articles/AndroidDevelopmentProblems/article.html
- http://d.android.com
- http://stackoverflow.com/questions/tagged/android

MOBILE APPLICATION DEVELOPMENT

Životní cyklus součástí

Innovative Open Source courses for Computer Science

30.05.2021

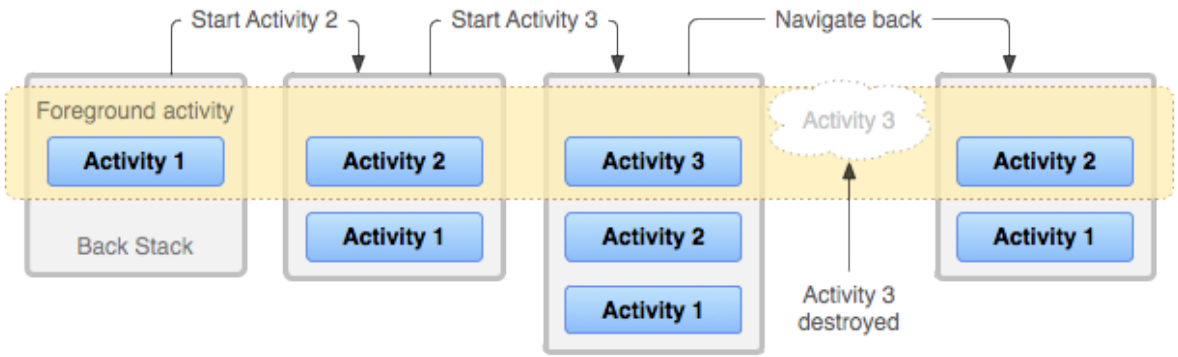
Funded by
the European Union

Activity

- Aktivita je komponent, ktorý poskytuje obrazovku s s ktorou môžu používatelia komunikovať, aby niečo urobili
- Každá činnosť má k dispozícii okno, do ktorého sa vykresľuje jej používateľské rozhranie.
- Okno zvyčajne vypĺňa obrazovku, niekedy môže byť menšie
- Aplikácia sa skladá z viacerých aktivít, ktoré sú voľne navzájom viazané.

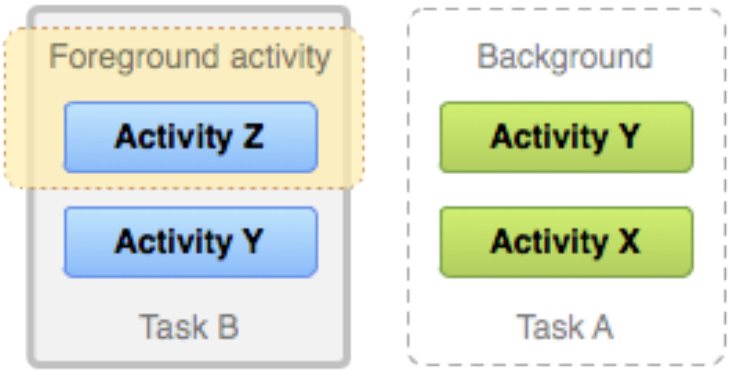
Multiply Activities - how arrange

- Každá činnosť môže potom spustiť inú činnosť, aby sa vykonala rôzne činnosti.
- Pri každom spustení novej činnosti sa predchádzajúca činnosť zastaví, ale systém zachováva aktivitu v zásobníku ("back stack").
- Keď sa spustí nová aktivita, presunie sa na zadný zásobník a prevezme zameranie používateľa.
- Zadný zásobník je základným mechanizmom zásobníka "posledný prichádza, prvý odchádza".



Násobiť úlohu - ako zariadiť

- Keď používateľ spustí aplikáciu prvýkrát alebo keď je aplikácia zničená, vytvorí sa nová úloha.
- Keď aplikácia existuje, úloha tejto aplikácie sa dostane do popredia



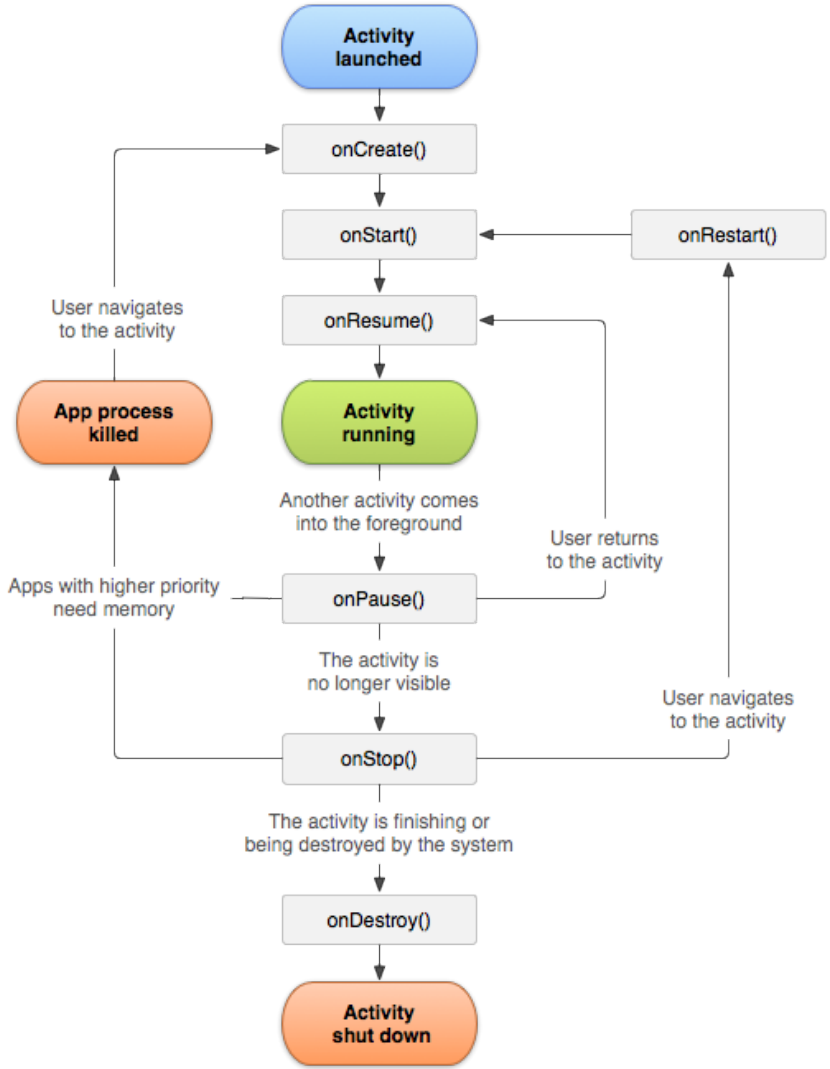
Vytvorenie aktivity

- Ak chcete vytvoriť aktivitu, musíte vytvoriť podtriedu Activity (alebo jej existujúca podtrieda)
- Aktivita má sedem metód spätného volania
- Musíme deklarovať iba jednu *onCreate()*
- Ostatné závisí od požiadaviek aplikácie
- Predvídateľnosť aktivity závisí od pochopenia životného cyklu aktivity

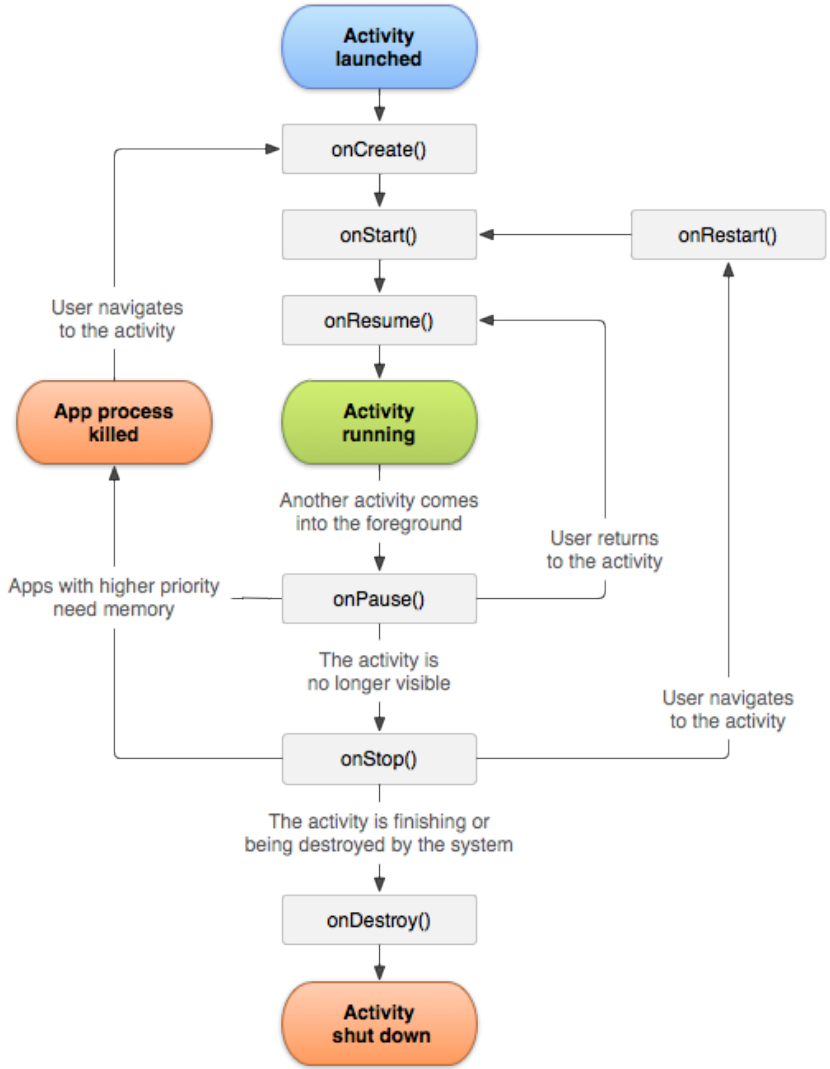
Implementácia spätných volaní životného cyklu - kostra

```
1  class NewActivity : AppCompatActivity(){
2      override fun onCreate(savedInstanceState: Bundle?){
3          super.onCreate(savedInstanceState)
4          setContentView(R.layout.activity_new)
5          // The activity is being created.
6      }
7      override fun onPause(){
8          super.onPause()
9          // Another activity is taking focus (this activity is about to be "paused").
10     }
11     override fun onRestart(){
12         super.onRestart()
13     }
14     override fun onResume(){
15         super.onResume()
16         // The activity has become visible (it is now "resumed").
17     }
18     override fun onStart(){
19         super.onStart()
20         // The activity is about to become visible.
21     }
22     override fun onStop(){
23         super.onStop()
24         // The activity is no longer visible (it is now "stopped")
25     }
26     override fun onDestroy(){
27         super.onDestroy()
28         // The activity is about to be destroyed.
29     }
30 }
```

Životný cyklus aktivity



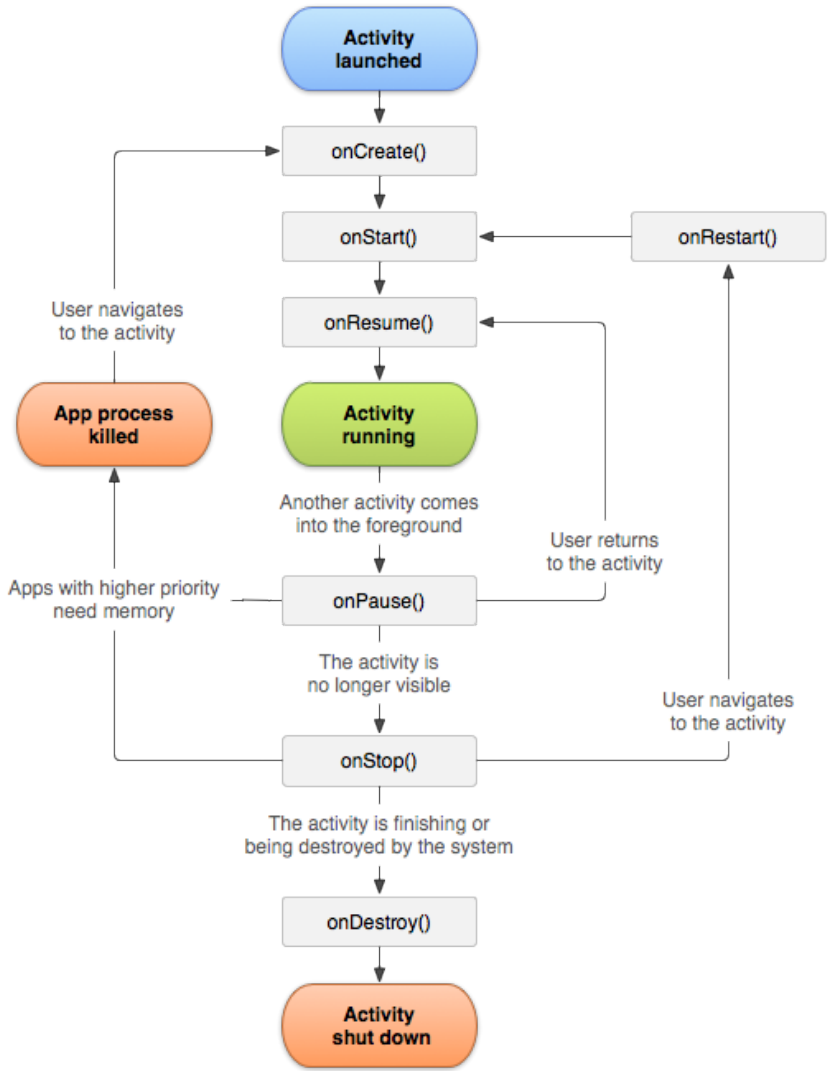
Životný cyklus aktivity



onCreate()

Volá sa pri prvom vytvorení aktivity. Mali by ste nastaviť - vytvoriť pohľady, zviazať údaje so zoznamami atď. Tejto metóde sa odovzdáva objekt Bundle, ktorý obsahuje predchádzajúce aktivity stav.
Vždy nasleduje po *onStart()*.

Životný cyklus aktivity



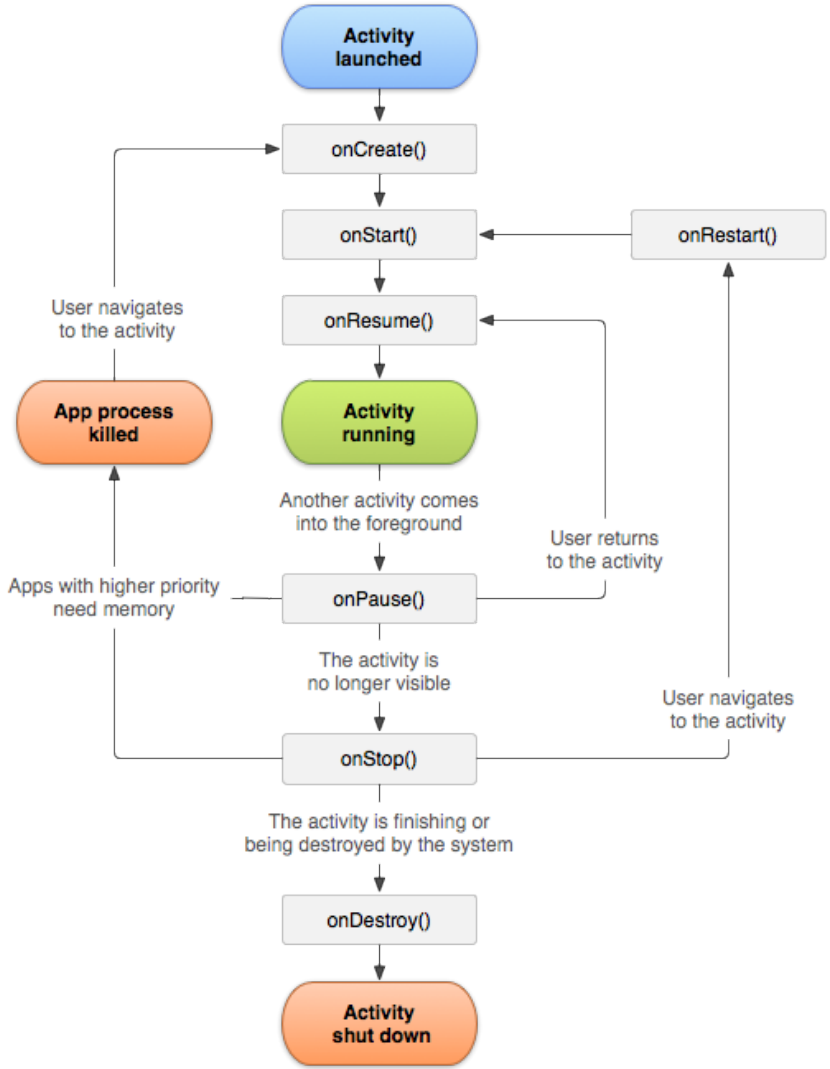
onRestart()

Volá sa po zastavení činnosti, tesne pred jej opätovným spustením.
Vždy nasleduje *onStart()*.

onStart()

Volá sa tesne pred tým, ako sa aktivita stane viditeľnou pre používateľa.
Nasleduje *onResume()*, ak sa aktivita dostane do popredia, alebo *onStop()*, ak sa stane skrytou.

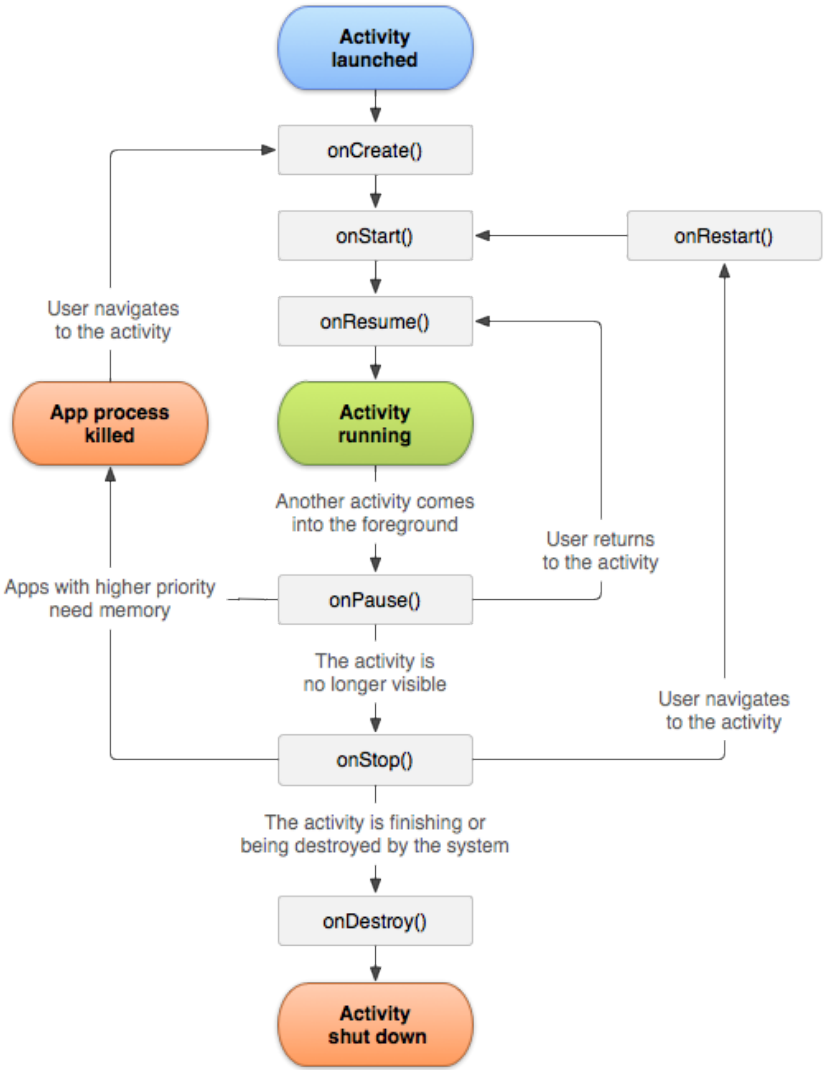
Životný cyklus aktivity



onResume()

Volá sa tesne predtým, ako aktivita začne komunikovať s používateľom. Pri tomto sa aktivita nachádza na vrchole zásobníka aktivít, pričom vstup od používateľa smeruje do nej. Vždy nasleduje *onPause()*.

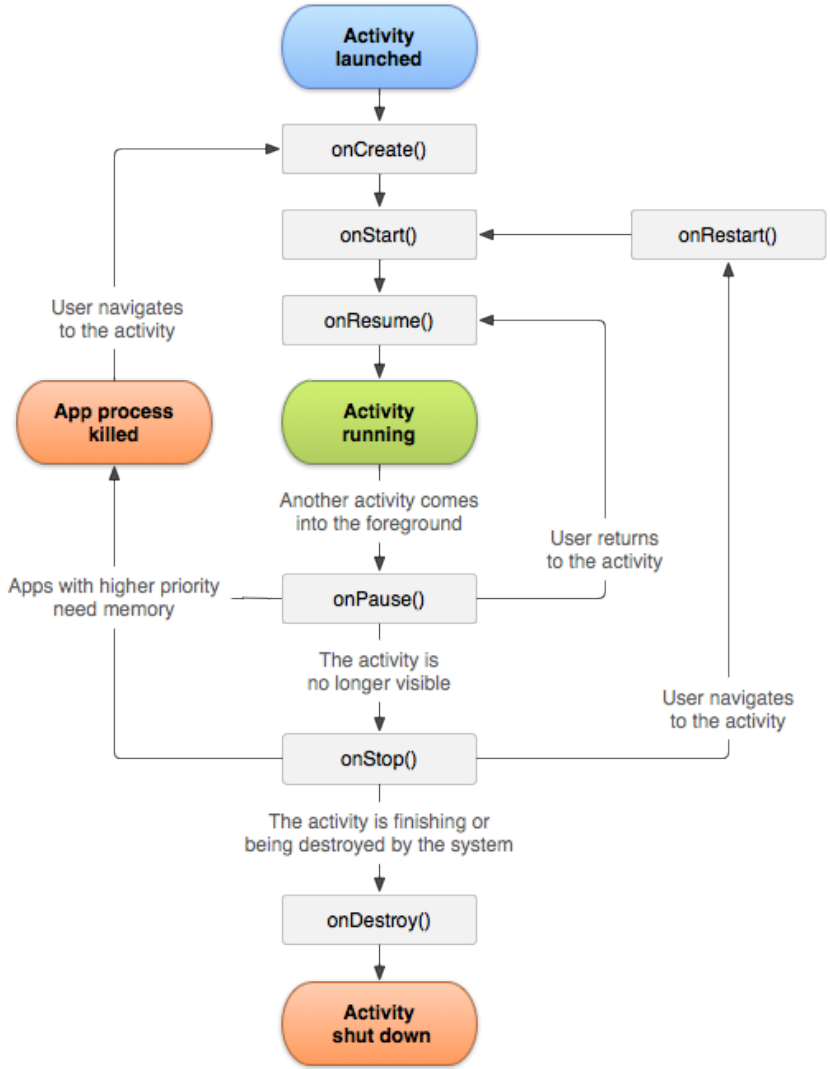
Životný cyklus aktivity



onPause()

Volá sa, keď sa systém chystá začať pokračovať v inej činnosti. Mali by ste: odovzdať neuložené zmeny do trvalých údajov, zastaviť animácie a iné veci, ktoré môžu spotrebovávať procesor, a podobne. Malo by to robiť čokoľvek veľmi rýchlo, ďalšia činnosť nebude pokračovať, kým sa nevráti. Nasleduje buď *onResume()*, ak sa aktivita vráti späť do dopredu, alebo *onStop()*, ak sa stane pre používateľa neviditeľnou.

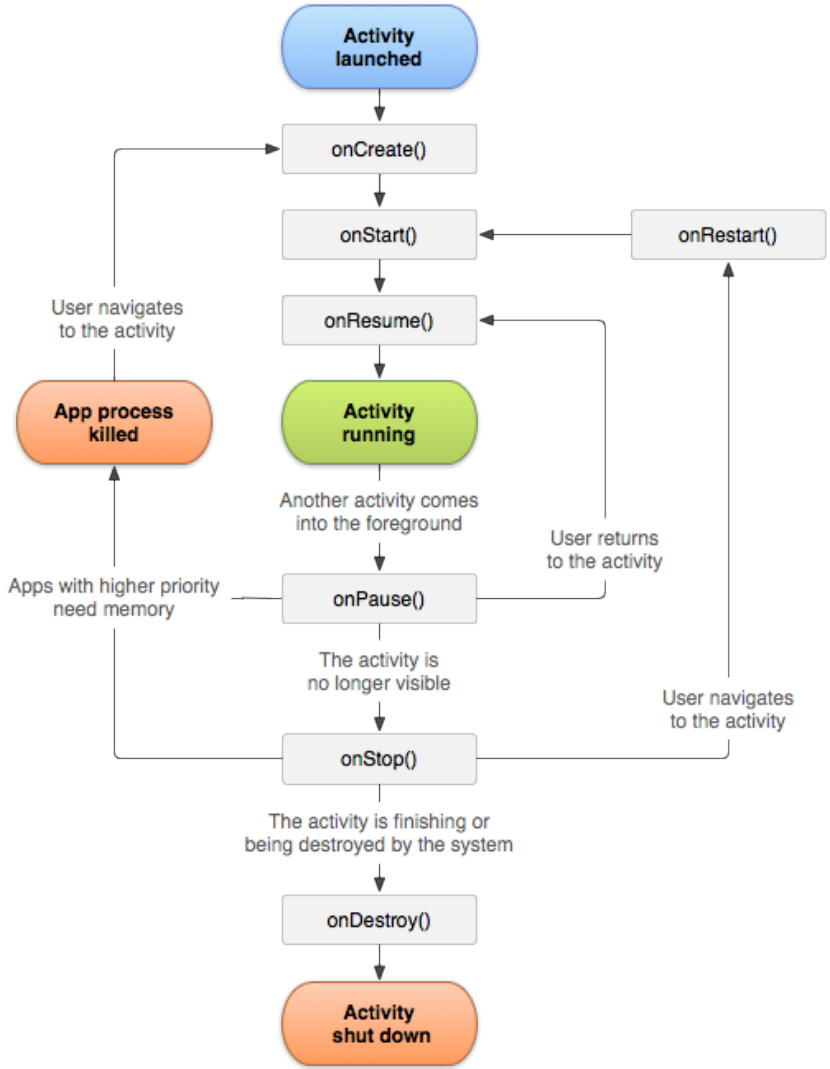
Životný cyklus aktivity



onStop()

Volá sa, keď činnosť už nie je pre používateľa viditeľná. To sa môže stať pretože je zničená alebo pretože iná aktivita (buď existujúca alebo nová) bola obnovená a zakrýva ju. Nasleduje buď *onRestart()*, ak sa činnosť vracia späť interakciu s používateľom, alebo po *onDestroy()*, ak sa táto činnosť vracia preč.

Životný cyklus aktivity



onDestroy()

Volá sa pred zničením činnosti. Toto je posledné volanie, ktoré aktivita prijme.

Viac informáciíViac informácií

- https://developer.android.com/guide/components/activities/activity-lifecycle

Uloženie stavu aktivity

- V predvolenom nastavení systém používa Bundle instance state na uloženie informácií o každom objekte View v rozložení aktivity, ale nie na uloženie všetkých informácií.
- Môžete však (a **should**)proaktívne zachovať stav vašich aktivít pomocou metódy *onSaveInstanceState()*.
- Keď sa vaša aktivita začne zastavovať, systém zavolá metódu onSaveInstanceState()
- Tieto metódy používajú dvojice kľúč-hodnota na uloženie stavu

```
1  override fun onSaveInstanceState(outState: Bundle?){
2      // Save the user's current game state
3      outState?.run{
4          putInt(STATE_SCORE, currentScore)
5          putInt(STATE_LEVEL, currentLevel)
6      }
7
8      // Always call the superclass so it can save the view hierarchy state
9      super.onSaveInstanceState(outState)
10 }
11
12 companion object{
13     val STATE_SCORE = "playerScore"
14     val STATE_LEVEL = "playerLevel"
15 }
```

Obnovenie stavu činnosti

- Môžete použiť *onCreate()* alebo *onRestoreInstanceState()*.
- Tieto metódy dostanú ten istý Bundle, ktorý obsahuje informácie o stave inštancie.

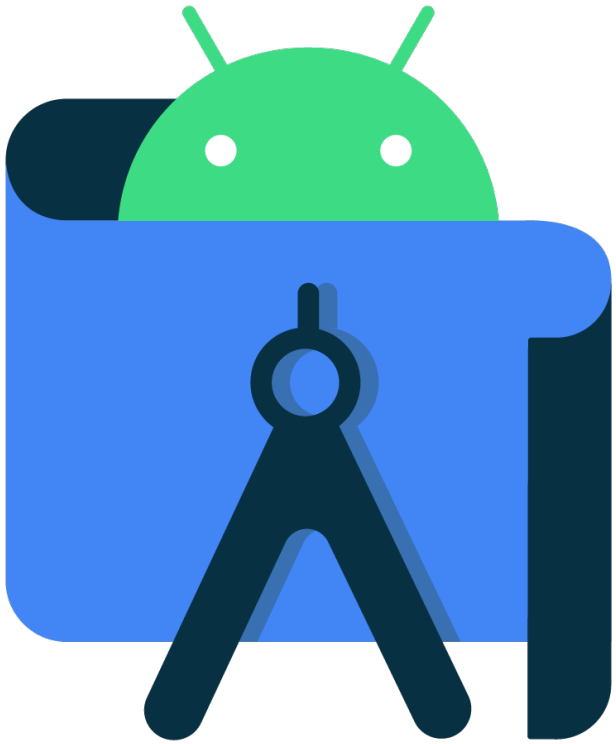
```
override fun onCreate(savedInstanceState: Bundle?){
    super.onCreate(savedInstanceState) // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null){
        with(savedInstanceState){
            // Restore value of members from saved state
            currentScore = getInt(STATE_SCORE)
            currentLevel = getInt(STATE_LEVEL)
        }
    } else{
        // Probably initialize members with default values for a new instance
    }
}
```

```
override fun onRestoreInstanceState(savedInstanceState: Bundle?){
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState)

    // Restore state members from saved instance
    savedInstanceState?.run{
        currentScore = getInt(STATE_SCORE)
        currentLevel = getInt(STATE_LEVEL)
    }
}
```

Živý príklad - Životný cyklus systému Android



Prechod medzi činnosťami

- Na spustenie aktivity môžeme použiť dve metódy *startActivity()* alebo *startActivityForResult()*
- Tieto metódy vyžadujú objekt Intent, ktorý obsahuje informácie o aktivite

Explicitné

```
1 val intent = Intent(this, OtherActivity::class.java)
2 startActivity(intent)
```

Implicitné

```
1 val intent = Intent(Intent.ACTION_SEND).apply{
2     putExtra(Intent.EXTRA_EMAIL, recipientArray)
3 }
4 startActivity(intent)
```

Spustenie činnosti a čakanie na výsledok [1/2]

- Keď potrebujeme získať výsledok, musíme použiť *startActivityForResult()*
- Impement tejto metódy musíme pridať kód vo vnútri dvoch aktivít

Prvá aktivita - Spustenie druhej aktivity

```
1  companion object{
2          const val REQUEST_CODE = 67 //declare request code
3      }
4      fun activityCall(){
5      val intent = Intent(this, OtherActivity::class.java)
6      startActivityForResult(intent,REQUEST_CODE)
7      }
```

Implement Receive methods

```
1      override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?){
2          super.onActivityResult(requestCode, resultCode, data)
3          // Check which request we're responding to
4          if (requestCode == REQUEST_CODE){
5              // Make sure the request was successful
6              if (resultCode == Activity.RESULT_OK){
7                  // Do something with the data here
8              }
9          }
10     }
```

Začiatok činnosti a čakanie na výsledok [2/2]

Odoslanie údajov z druhej aktivity

```
1     fun responseButton(view: View){
2         Log.i(TAG, "responseButton")
3         val returnIntent = Intent()
4         returnIntent.putExtra("result", "data from secondActivity")
5
6         setResult(RESULT_OK, returnIntent)
7         finish()
8     }
```

Intent

- Spustenie činnosti
- Spustenie služby
- Doručenie vysielania
- Explicitný zámer - špecifikácia, ktorá aplikácia bude spĺňať zámer
- Implicitný zámer - nepomenovať konkrétnu zložku, ale namiesto toho deklarovať všeobecnú činnosť, ktorá sa má vykonať

Vytvorenie zámeru

- Pri vytváraní objektu zámeru zadáme
- **Component name** - iba Explicit Intent
- **Action** - Reťazec, ktorý špecifikuje všeobecnú akciu, ktorá sa má vykonať, systémovú Akciu alebo vlastnú Akciu
- **Data** - objekt URI
- **Category** - dodatočné informácie o druhu komponentu, ktorý by mal spracovať zámer
- **Extras** - dvojice kľúč-hodnota, ktoré nesú dodatočné informácie potrebné na vykonanie požadovanej akcie

Štandardná systémová akcia

- ACTION_MAIN
- ACTION_VIEW
- ACTION_ATTACH_DATA
- ACTION_EDIT
- ACTION_PICK
- ACTION_CHOOSER
- ACTION_GET_CONTENT
- ACTION_DIAL
- ACTION_CALL
- ACTION_SEND
- ACTION_SENDTO
- ACTION_ANSWER
- ACTION_INSERT
- ACTION_DELETE
- ACTION_RUN
- ACTION_SYNC
- ACTION_PICK_ACTIVITY
- ACTION_SEARCH
- ACTION_WEB_SEARCH
- ACTION_FACTORY_TEST

Príklady

- ACTION _VIEW `content://contacts/people/1` – Zobrazenie informácie o osobe, ktorej identifikátor je “1”.
- ACTION_DIAL `content://contacts/people/1` – Zobrazenie telefónu s vyplnenou osobou.
- ACTION_EDIT `content://contacts/people/1` – Upraviť informácie o osobe, ktorej identifikátor je “1”.
- ACTION_VIEW `tel:123` – Zobrazenie telefónneho číselníka s daným s vyplneným číslom. Všimnite si, ako ACTION VIEW robí to, čo sa pre daný URI považuje za najrozumnejšie.
- ACTION_DIAL `tel:123` – Zobrazí telefónny číselník s daným vyplneným číslom.

Kategória

- Poskytuje dodatočné informácie o akcii, ktorá sa má vykonať.
- Napríklad CATEGORY _LAUNCHER znamená, že by sa mala zobraziť v spúšťači ako aplikácia najvyššej úrovne.
- CATEGORY_ALTERNATIVE znamená, že by mala byť zaradená do zoznamu alternatívnych akcií, ktoré môže používateľ vykonať na časti údajov.
- To znamená, že ak zahrniete kategórie CATEGORY_LAUNCHER a CATEGORY_ALTERNATIVE, potom budete riešiť iba komponenty so zámerom ktorý obsahuje zoznam oboch týchto kategórií.
- Činnosti budú veľmi často potrebovať podporovať kategóriu CATEGORY_DEFAULT, aby ich bolo možné nájsť Context.startActivity().
- DEFAULT CATEGORY sa vyžaduje pre všetky filtre - okrem tých, ktoré majú akciu MAIN a kategóriu LAUNCHER.

Štandardné kategórie

- CATEGORY_DEFAULT
- CATEGORY_BROWSABLE
- CATEGORY_TAB
- CATEGORY_ALTERNATIVE
- CATEGORY_SELECTED_ALTERNATIVE
- CATEGORY_LAUNCHER
- CATEGORY_INFO
- CATEGORY_APP_MARKET
- CATEGORY_HOME
- CATEGORY_PREFERENCE
- CATEGORY_TEST
- CATEGORY_CAR_DOCK
- CATEGORY_DESK_DOCK
- CATEGORY_LE_DESK_DOCK
- CATEGORY_HE_DESK_DOCK
- CATEGORY_CAR_MODE

Príklad kódu

Explicitný zámer

```
1 val fileDownloadIntent = Intent(this, FileDownloadService::class.java).apply{
2     data = Uri.parse(fileUrl)
3 }
4 startService(fileDownloadIntent)
```

```
1 val intent = Intent(this, OtherActivity::class.java)
2 startActivity(intent)
```

Implicitný zámer

```
1 val fileDownloadIntent = Intent(this, FileDownloadService::class.java).apply{
2     data = Uri.parse(fileUrl)
3 }
4 startService(fileDownloadIntent)
```

```
1
2 val sendIntent = Intent().apply{
3     action = Intent.ACTION_SEND
4     putExtra(Intent.EXTRA_TEXT, textMessage)
5     type = "text/plain"
6 }
7 // Try to invoke the intent.
8 try{
9     startActivity(sendIntent)
10 } catch (e: ActivityNotFoundException){
11     // Define what your app should do if no activity can handle the intent.
12 }
```

Vynucovanie výberu aplikácie

- Pomocou implicitných zámerov si používateľ môže vybrať, ktorú aplikáciu použije (ak je ich viac)
- Používateľ môže nastaviť predvolenú aplikáciu pre určitú činnosť
- Pomocou *createChooser()* zobrazíme choiceer a napr. pošleme dáta iným aplikáciám

```
1  //1. Define Intent
2  val sendIntent = Intent(Intent.ACTION_SEND)
3  // Always use string resources for UI text.
4  // This says something like "Share this photo with"
5  //2.Create title
6  val title: String = resources.getString(R.string.chooser_title)
7  //3. Create intent to show the chooser dialog
8  val chooser: Intent = Intent.createChooser(sendIntent, title)
9
10 //4. Verify the original intent will resolve to at least one activity
11 if (sendIntent.resolveActivity(packageManager) != null){
12     startActivity(chooser)
13 }
```

Prijímanie implicitného Intnetu

- Aby sme mohli prijímať implicitný zámer, musíme deklarovať jeden alebo viac filtrov zámeru pre každú z komponentov aplikácie
- Systém doručí implicitný zámer do vašej aplikačnej zložky len vtedy, ak zámer môže prejsť cez jeden z vašich filtrov zámerov.

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
```

Viac informácií

- https://developer.android.com/guide/components/activities/activity-lifecycle
- https://developer.android.com/guide/components/intents-filters

MOBILE APPLICATION DEVELOPMENT

Fragment - životní cyklus

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Fragment

- Představuje opakovaně použitelnou část uživatelského rozhraní aplikace.
- Fragment definuje a spravuje vlastní rozvržení
- Má vlastní životní cyklus
- Může zpracovávat vlastní vstupní události
- Fragment musí být hostován aktivitou nebo jiným fragmentem

Vytvoření fragmentu

- Nastavení prostředí
- Vytvoření třídy fragmentů
- Přidání fragmentu do aktivity

Přidat do projektu **build.gradle** informace o úložišti Google Maven

```
1  buildscript{
2      ...
3
4      repositories{
5          google()
6          ...
7      }
8  }
9
10 allprojects{
11     repositories{
12         google()
13         ...
14     }
15 }
```

Vytvoření fragmentu

- Nastavení prostředí
- Vytvoření třídy fragmentů
- Přidání fragmentu do aktivity

Přidání informací o knihovně AndroidX Fragment do aplikace **build.gradle**

```
dependencies{
    val fragment_version = "1.3.4"

    // Java language implementation
    implementation("androidx.fragment:fragment:$fragment_version")
    // Kotlin
    implementation("androidx.fragment:fragment-ktx:$fragment_version")
}
```

Vytvoření fragmentu

- Nastavení prostředí
- Vytvoření třídy fragmentů
- Přidání fragmentu do aktivity

Můžeme použít **Fragemnt, DialogFragment, PreferenceFragmentCompat**

```
1 class ExampleFragment : Fragment(R.layout.example_fragment)
```

Vytvoření fragmentu

- Nastavení prostředí
- Vytvoření třídy fragmentů
- Přidání fragmentu do aktivity

Definice v XML, *android:name* obsahující jednu třídu

```
1 <androidx.fragment.app.FragmentContainerView
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/fragment_container_view"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:name="com.example.ExampleFragment" />
```

Vytvoření fragmentu

- Nastavení prostředí
- Vytvoření třídy fragmentů
- Přidání fragmentu do aktivity

nebo (častěji) Definovat v kontejneru XML pro fragment

```
1 <androidx.fragment.app.FragmentContainerView
2     xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/fragment_container_view"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent" />
```

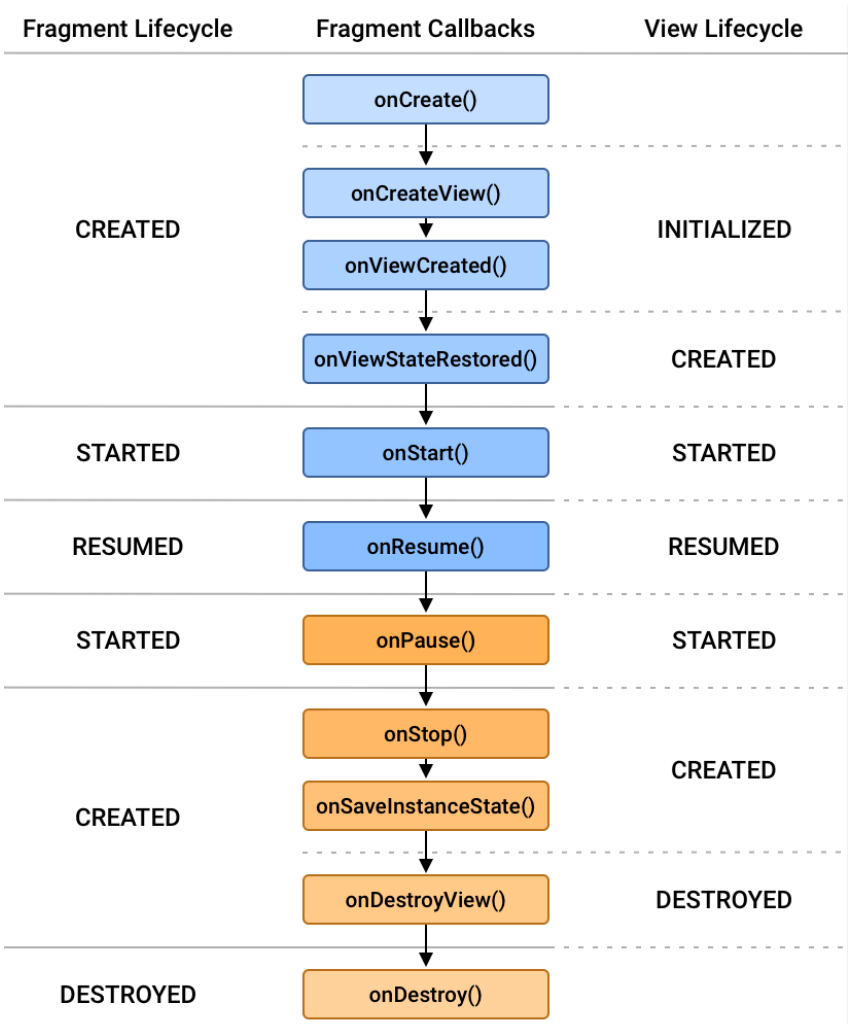
Přidání kódu do aktivity (onCreate())

```
1         supportFragmentManager.commit{
2                 setReorderingAllowed(true)
3                 add<ExampleFragment>(R.id.fragment_container_view)
```

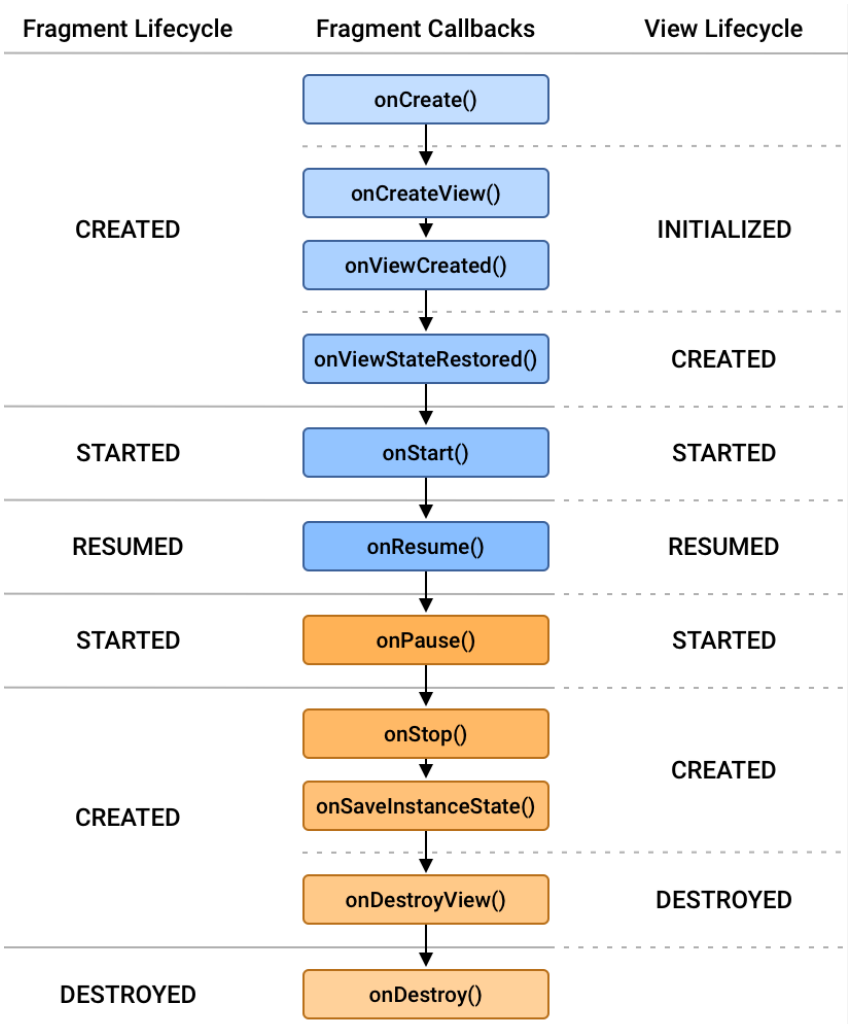
Životní cyklus fragmentu

- Každá instance **Fragment** má svůj vlastní životní cyklus.
- Životní cyklus pohledu se liší od životního cyklu fragmentu
- Stav fragmentu:
  - INITIALIZED
  - CREATED
  - STARTED
  - RESUMED
  - DESTROYED

Životní cyklus fragmentu



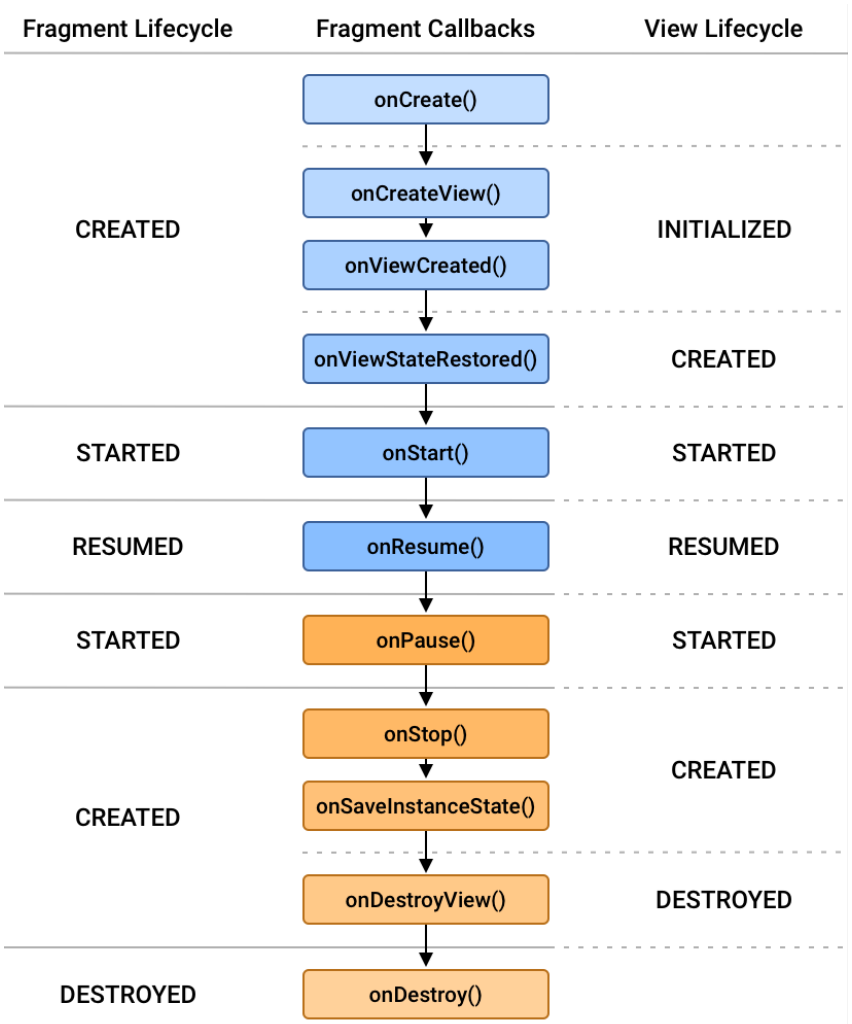
Životní cyklus fragmentu



CREATED

Byl přidán do FragmentManager-u a metoda onAttach() již byla zavolána. Životní cyklus pohledu fragmentu se vytvoří pouze tehdy, když váš Fragment poskytne platnou instanci pohledu. Můžete také přepsat funkci onCreateView(), abyste programově nafoukli nebo vytvořili zobrazení vašeho fragmentu.

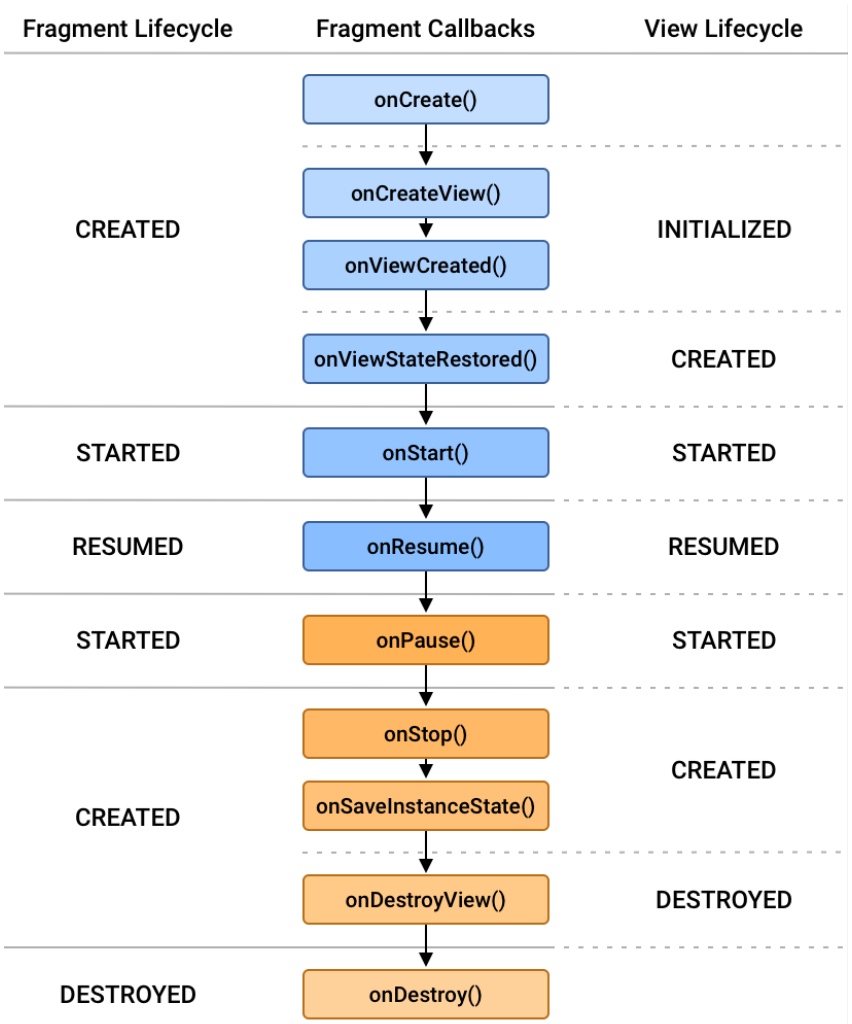
Životní cyklus fragmentu



STARTED

Tento stav zaručuje, že zobrazení fragmentu je k dispozici, pokud bylo vytvořeno, a že je bezpečné provést FragmentTransaction na podřízeném FragmentManager-u fragmentu.

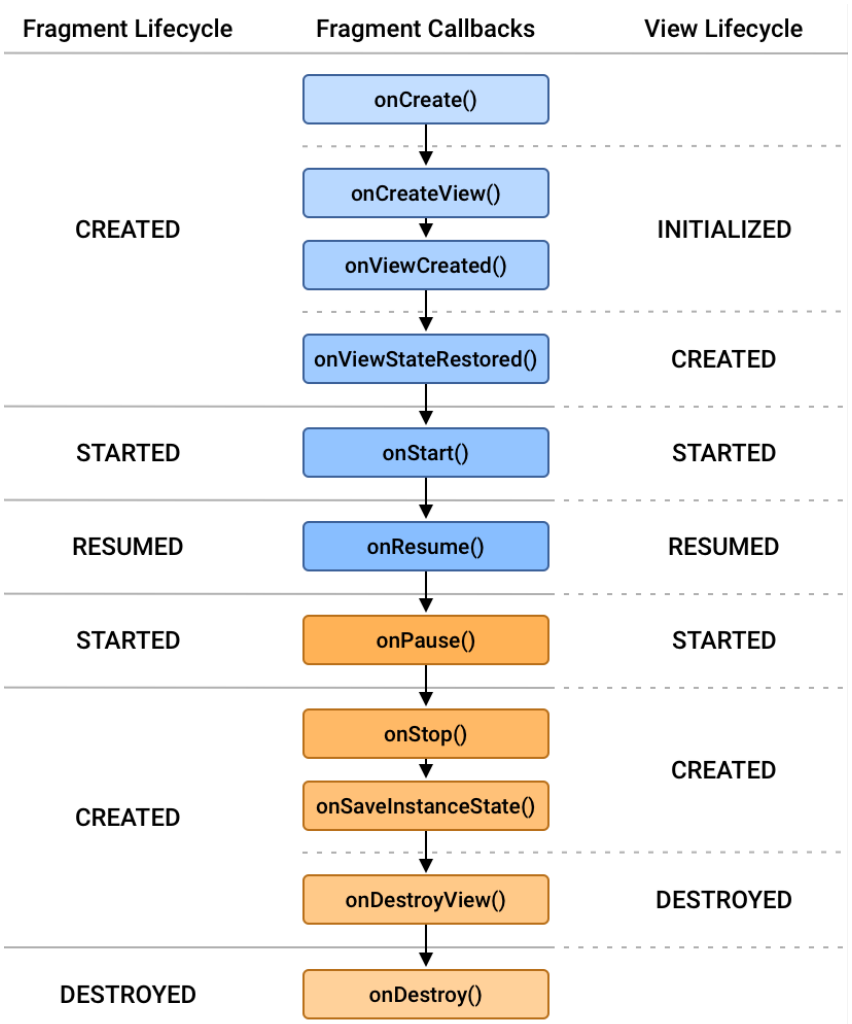
Životní cyklus fragmentu



RESUMED

Když je fragment viditelný, všechny efekty animátoru a přechodu jsou dokončeny a fragment je připraven k interakci s uživatelem.

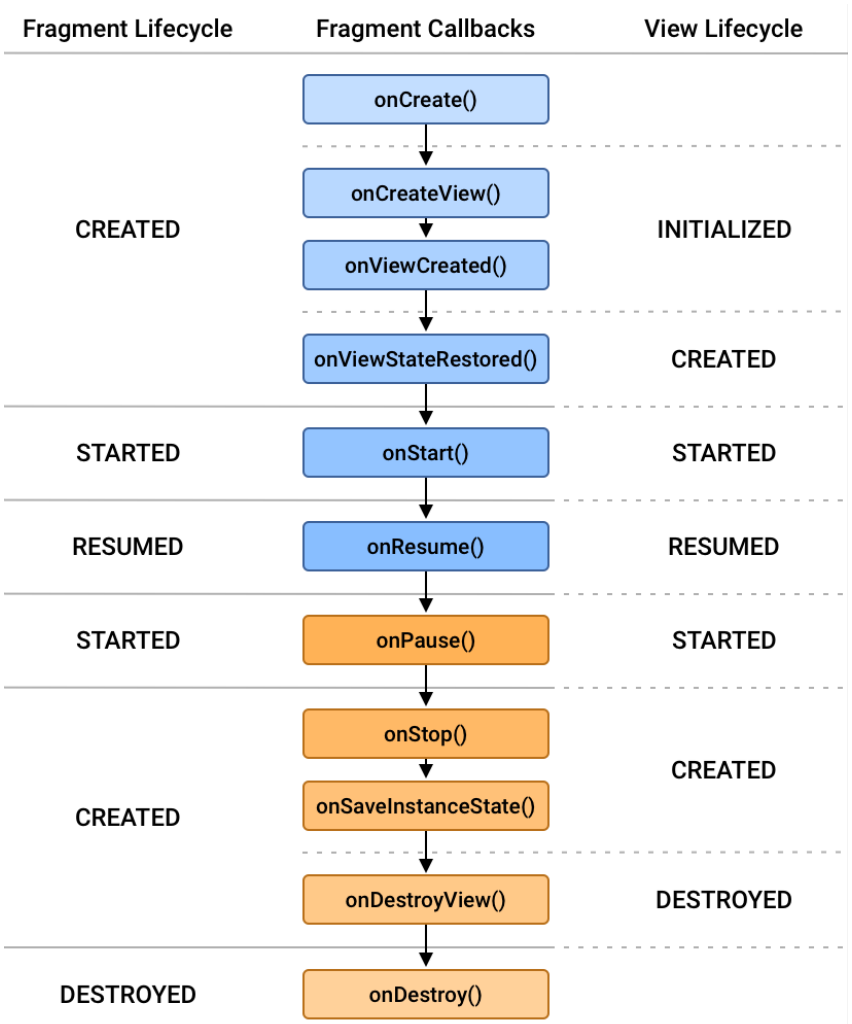
Životní cyklus fragmentu



STARTED

Jakmile uživatel začne fragment opouštět a dokud je fragment stále viditelný, životní cykly fragmentu a jeho zobrazení se přesunou zpět do stavu STARTED.

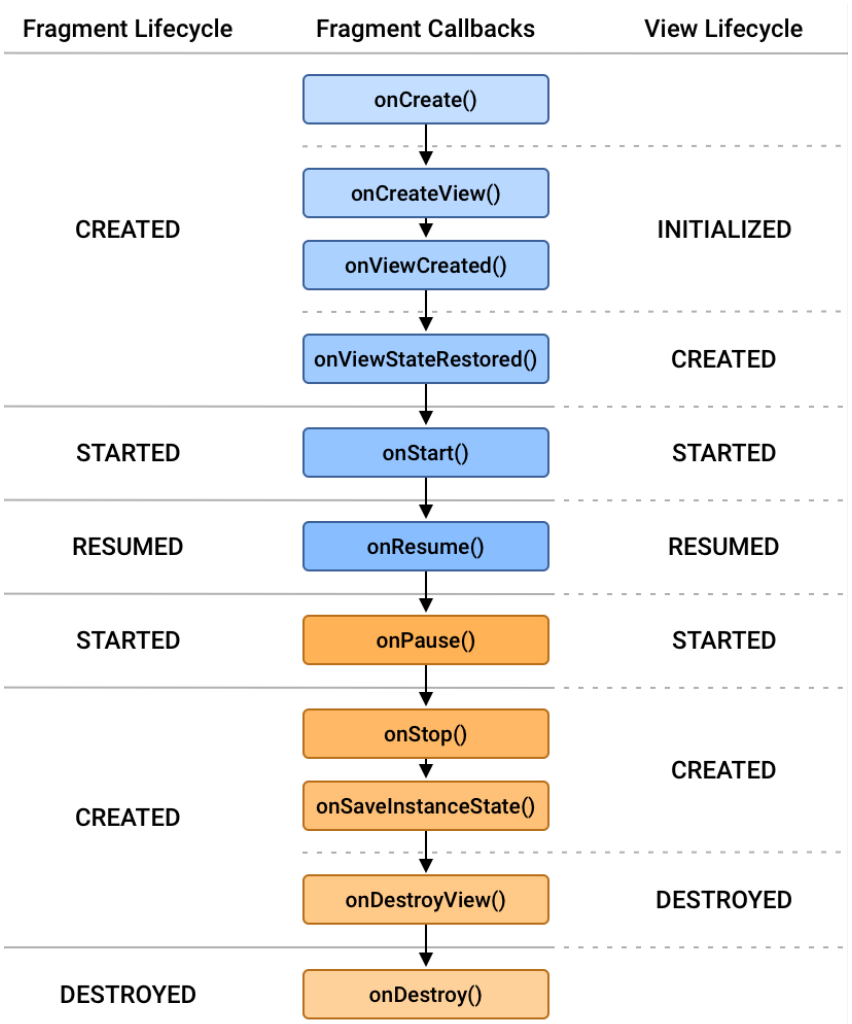
Životní cyklus fragmentu



CREATED

Jakmile fragment již není viditelný, životní cykly fragmentu a jeho zobrazení se přesunou do stavu STARTOVÁNO. Následuje buď *onResume()*, pokud se aktivita vrátí zpět do stavu dopředu, nebo *onStop()*, pokud se stane pro uživatele neviditelnou.

Životní cyklus fragmentu



DESTROYED

Fragment je odstraněn, nebo pokud je FragmentManager zničen

Metody životního cyklu - do obnoveného stavu (interakce s uživatelem).)

- *onAttach* - volá se, jakmile je fragment spojen s jeho činností.
- *onCreate* - volá se pro počáteční vytvoření fragmentu.
- *onCreateView* - vytvoří a vrátí hierarchii pohledů přidružených k fragmentu.
- *onActivityCreated* - sdělí fragmentu, že jeho aktivita dokončila svou *android.app.Activity-onCreate*.
- *onViewStateRestored* - sdělí fragmentu, že byl obnoven veškerý uložený stav jeho hierarchie pohledů.
- *onStart* - zviditelní fragment pro uživatele (na základě spuštění jeho obsahující aktivity).
- *onResume* - způsobí, že fragment začne komunikovat s uživatelem (na základě obnovení jeho obsahující aktivity).

- *onPause* - fragment již neinteraguje s uživatelem, protože jeho činnost je pozastavena nebo jej v činnosti modifikuje operace fragmentu.
- *onStop* - fragment již není pro uživatele viditelný, protože jeho činnost je zastavena nebo jej v činnosti modifikuje operace fragmentu.
- *onDestroyView* - umožňuje fragmentu vyčistit prostředky spojené s jeho pohledem.
- *onDestroy* - volá se za účelem konečného vyčištění stavu fragmentu.
- *onDetach* - volá se bezprostředně před tím, než fragment přestane být spojen se svou aktivitou.

Service

- Aplikační komponenta, která může provádět dlouhodobé operace na pozadí.
- Neposkytuje uživatelské rozhraní
- Rozšíření třídy Service
- Všechny služby musíte deklarovat v souboru manifestu aplikace.

Typy služeb

- Foreground
- Background
- Bound

- Služba v popředí provádí nějakou operaci, která je pro uživatele viditelná.
- Služby v popředí musí zobrazovat oznámení.
- Toto oznámení nelze zrušit, pokud není služba zastavena nebo odstraněna
- Pokračuje v běhu, i když uživatel s aplikací neinteraguje

Typy služeb

- Foreground
- Background
- Bound

- Služba na pozadí provádí operaci, kterou uživatel přímo nevnímá.
- např. kompaktní úložiště
- API 26 nebo vyšší - omezení spouštění služeb na pozadí, když samotná aplikace není v popředí, neměli byste přistupovat k informacím o poloze z pozadí

Typy služeb

- Foreground
- Background
- Bound
- Vázaná služba nabízí rozhraní klient-server, které umožňuje komponentám komunikovat se službou, posílat požadavky, přijímat výsledky, a to i napříč procesy pomocí meziprocesové komunikace (IPC).
- Funguje pouze do té doby, dokud je k ní vázána jiná aplikační komponenta.
- Ke službě se může vázat více komponent najednou, ale když se všechny odvážou, služba se zničí.

Více informací

- https://developer.android.com/guide/fragments
- https://developer.android.com/guide/fragments/lifecycle

MOBILE APPLICATION DEVELOPMENT

User Interface - Uživatelské rozhraní

Innovative Open Source courses for Computer Science

30.05.2021

Funded by the European Union

Uživatelské rozhraní

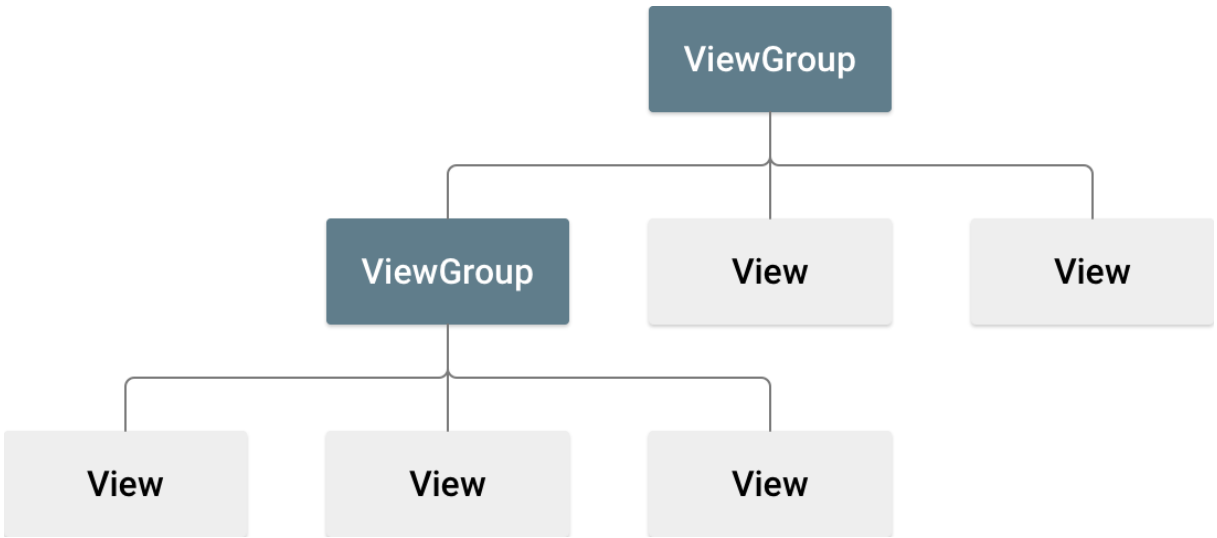
UI

Uživatelské rozhraní aplikace je vše, co uživatel vidí a s čím může interagovat. Systém Android poskytuje mnoho předpřipravených komponent uživatelského rozhraní, například **structured layout objects** a **UI controls**, které umožňují vytvořit grafické uživatelské rozhraní pro vaši aplikaci. Android poskytuje také další moduly uživatelského rozhraní pro speciální rozhraní, například **dialogs**, **notifications** a **menus**.

Layouts

Rozložení

Definuje strukturu uživatelského rozhraní v aplikaci, například v aktivitě. Všechny prvky rozvržení jsou vytvořeny pomocí hierarchie objektů View a ViewGroup.



Deklarovat rozložení

Rozložení můžete deklarovat dvěma způsoby:

- **Declare UI elements in XML.** - Systém Android poskytuje jednoduchý slovník XML, který odpovídá třídám a podtřídám View, jako jsou třídy pro widgety a rozvržení. K sestavení rozvržení XML pomocí rozhraní drag-and-drop můžete také použít editor rozvržení aplikace Android Studio.
- **Instantiate layout elements at runtime.** Vaše aplikace může programově vytvářet objekty View a ViewGroup (a manipulovat s jejich vlastnostmi).

Definice rozložení v ***XML*** soubor

- Deklarování uživatelského rozhraní v ***XML*** umožňuje oddělit prezentaci aplikace od kódu, který řídí její chování.
- ***View*** - se obvykle nazývá "widgety" a může být jednou z mnoha podtříd, například ***Button or TextView***.
- ***ViewGroup*** - obvykle se nazývají "layouty" a mohou být jedním z mnoha typů, které poskytují různou strukturu rozvržení, například ***LinearLayout or ConstraintLayout*** .
- Chcete-li rozvržení ladit za běhu, použijte nástroj Inspektor rozvržení. `https://developer.android.com/studio/debug/layout-inspector`

Příklad XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3                android:layout_width="match_parent"
4                android:layout_height="match_parent"
5                android:orientation="vertical" >
6      <TextView android:id="@+id/text"
7                android:layout_width="wrap_content"
8                android:layout_height="wrap_content"
9                android:text="Hello, I am a TextView" />
10     <Button android:id="@+id/button"
11             android:layout_width="wrap_content"
12             android:layout_height="wrap_content"
13             android:text="Hello, I am a Button" />
14 </LinearLayout>
```

- Každý soubor rozvržení musí obsahovat přesně jeden kořenový prvek, který musí být objektem View nebo ViewGroup (např. LinearLayout).
- Všechna rozvržení ukládáme do ***res/layout/***.
- Systém Android podporuje různé velikosti obrazovky

Různé velikosti obrazovek

```
1 res/layout/main_activity.xml                  # For handsets
2 res/layout-land/main_activity.xml             # For handsets in landscape
3 res/layout-sw600dp/main_activity.xml          # For 7 inch tablets
4 res/layout-sw600dp-land/main_activity.xml     # For 7 inch tablets in landscape
```

- Rozložení specifické pro obrazovku můžete poskytnout vytvořením dalších adresářů ***res/layout/*** - jednoho pro každou konfiguraci obrazovky, která vyžaduje jiné rozložení.
- Použijte kvalifikátor dostupné šířky (např. ***sw600dp*** - obrazovka s 600dp).
- Použijte kvalifikátory orientace (např. ***land or port*** - rozložení na výšku, resp. na šířku).

Typ rozvržení I

- Linear Layout - je skupina zobrazení, která zarovnává všechny děti v jednom směru, vertikálně nebo horizontálně.
- ~~Relative Layout~~ - je skupina zobrazení, která zobrazuje podřízená zobrazení v relativních pozicích.
- Constraint Layout -je zobrazení pro vytváření rozsáhlých a složitých rozvržení s plochou hierarchií zobrazení (bez vnořených skupin zobrazení). Je podobné RelativeLayout v tom, že všechna zobrazení jsou rozložena podle vztahů mezi sourozeneckými zobrazeními a nadřazeným rozložením, ale je flexibilnější než RelativeLayout.
- Table Layout - je zobrazení, které seskupuje pohledy do řádků a sloupců.
- Absolute Layout - umožňuje zadat přesné umístění jeho potomků.

Typ rozvržení II

- Frame Layout - je zástupný symbol na obrazovce, který můžete použít k zobrazení jednoho zobrazení.
- List View - ListView je skupina zobrazení, která zobrazuje seznam rolovatelných položek. (Rozložení s adaptérem)
- Grid View - GridView je skupina ViewGroup, která zobrazuje položky ve dvourozměrné rolovací mřížce. (Rozložení s adaptérem)
- `https://www.tutorialspoint.com/android/android_user_interface_layouts.htm`

Atributy rozvržení I

- *android* : *id* - Toto je ID, které jednoznačně identifikuje zobrazení.
- *android* : *layout_width* - Toto je šířka rozvržení.
- *android* : *layout_height* - Toto je výška rozvržení.
- *android* : *layout_marginTop* - Toto je prostor navíc na horní straně rozvržení.
- *android* : *layout_marginBottom* - Toto je prostor navíc na spodní straně rozložení.
- *android* : *layout_marginLeft* - Toto je prostor navíc na levé straně rozložení.
- *android* : *layout_marginRight* - Toto je prostor navíc na pravé straně rozložení.
- *android* : *layout_gravity* - Toto určuje, jak jsou umístěna podřízená zobrazení.

Atributy rozvržení II

- *android* : *layout_weight* - Toto určuje, kolik místa navíc v rozvržení má být přiděleno Zobrazení.
- *android* : *layout_x* - Toto určuje souřadnici x rozvržení.
- *android* : *layout_y* - Toto určuje y-ovou souřadnici rozložení.
- *android* : *layout_width* - Toto je šířka rozložení.
- *android* : *paddingLeft* - Toto je levý padding vyplněný pro rozložení.
- *android* : *paddingRight* - Toto je pravý padding vyplněný pro rozložení.
- *android* : *paddingTop* - Toto je horní výplň rozvržení.
- *android* : *paddingBottom* - Toto je spodní výplň rozvržení.

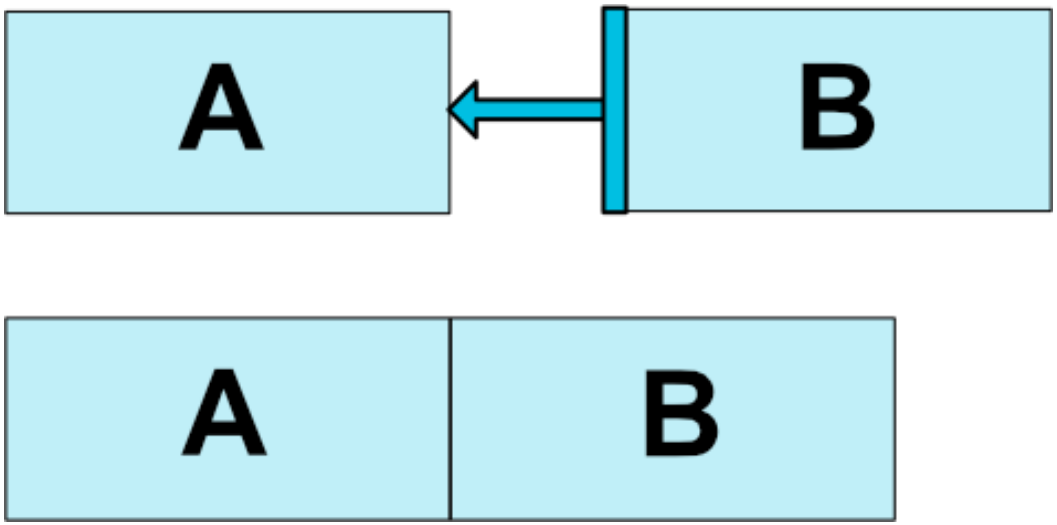
ConstraintLayout

- Rozložení ConstraintLayout je ***android.view.ViewGroup***, které umožňuje flexibilní umístění a velikost widgetů.
- V současné době existují různé typy omezení, které můžete použít:
  - Relativní pozicování
  - Okraje
  - Polohování na střed
  - Kruhové polohování
  - Chování při viditelnosti
  - Rozměrová omezení
  - Řetězce
  - Objekty virtuálních pomocníků
  -
- `https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout`

Relativní umístění

- Tato omezení umožňují umístit daný widget vzhledem k jinému widgetu.
- Widget můžete omezit na vodorovné a svislé ose:
  - Vodorovná osa: levá, pravá, počáteční a koncová strana.
  - Svislá osa: horní a dolní strana a základní čára textu
- Obecná koncepce spočívá v omezení dané strany widgetu na jinou stranu libovolného jiného widgetu.
- Všechny přebírají referenční id na jiný widget nebo na rodiče (který bude odkazovat na nadřazený kontejner, tj. na ConstraintLayout).

Relativní polohování - příklad



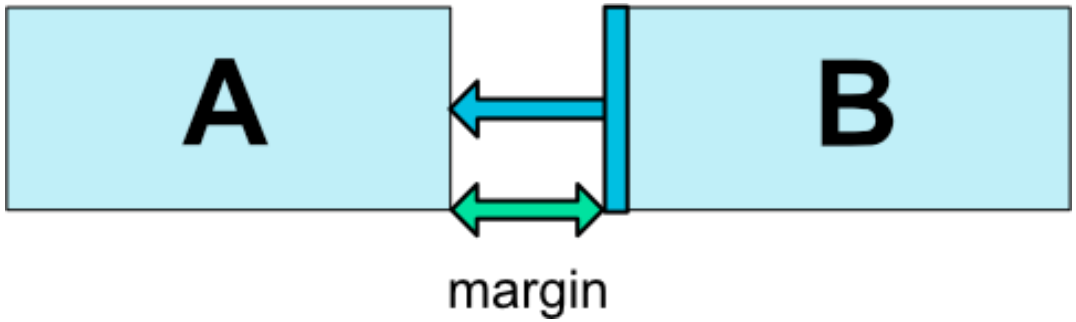

```
1 <Button android:id="@+id/buttonA" ... />
2          <Button android:id="@+id/buttonB" ...
3                  app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```

```
1 <Button android:id="@+id/buttonB" ...
2                  app:layout_constraintLeft_toLeftOf="parent" />
```

Dostupná omezení

- layout_constraintLeft_toLeftOf
- layout_constraintLeft_toRightOf
- layout_constraintRight_toLeftOf
- layout_constraintRight_toRightOf
- layout_constraintTop_toTopOf
- layout_constraintTop_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintBottom_toBottomOf
- layout_constraintBaseline_toBaselineOf
- layout_constraintStart_toEndOf
- layout_constraintStart_toStartOf
- layout_constraintEnd_toStartOf
- layout_constraintEnd_toEndOf

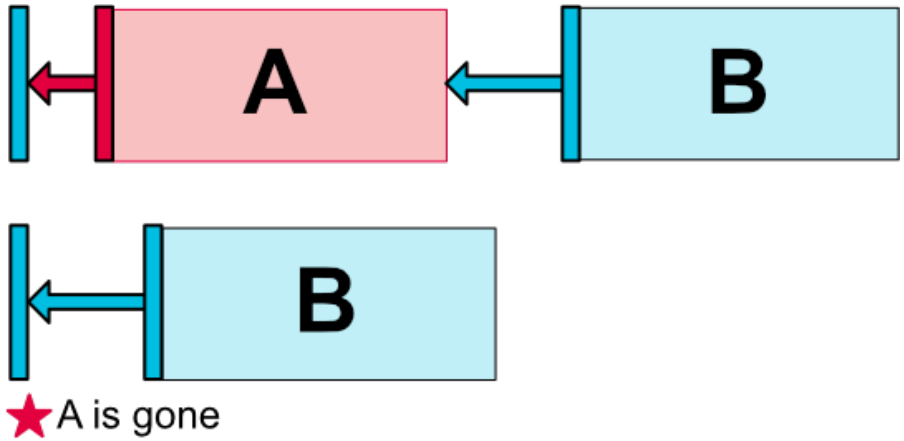
Marže



Pokud jsou nastaveny boční okraje, použijí se na odpovídající omezení a vynutí si okraj jako mezeru mezi cílovou a zdrojovou stranou.

- android:layout_marginStart
- android:layout_marginEnd
- android:layout_marginLeft
- android:layout_marginTop
- android:layout_marginRight
- android:layout_marginBottom

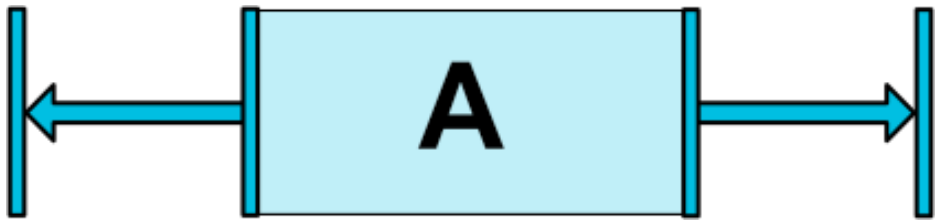
Chování při viditelnosti



ConstraintLayout má specifické zacházení s widgety, které jsou označeny jako ***View.GONE***. Widgety **GONE** se jako obvykle nezobrazí a nejsou součástí samotného rozvržení (tj. jejich skutečné rozměry se nezmění, pokud jsou označeny jako **GONE**). Z hlediska výpočtů rozvržení jsou však widgety GONE stále jeho součástí, s důležitým rozdílem:

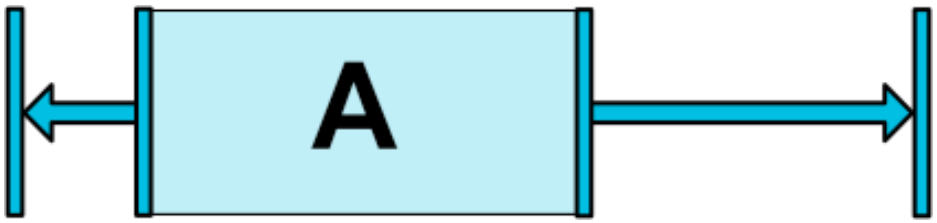
- Pro průchod rozvržením bude jejich rozměr považován za nulový (v podstatě budou vyřešeny do bodu).
- Pokud mají vazby na jiné widgety, budou stále respektovány, ale případné okraje budou jakoby rovny nule

Centrování polohy



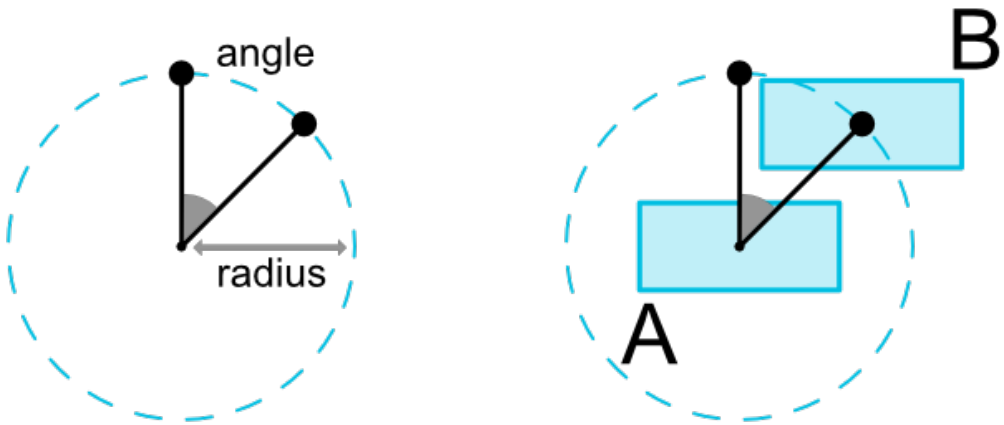

```
1 <androidx.constraintlayout.widget.ConstraintLayout ...>
2             <Button android:id="@+id/button" ...
3                 app:layout_constraintHorizontal_bias="0.3"
4                 app:layout_constraintLeft_toLeftOf="parent"
5                 app:layout_constraintRight_toRightOf="parent/>
6         </>
7
```

Centrování polohy s předsazením




```
1 <androidx.constraintlayout.widget.ConstraintLayout ...>
2             <Button android:id="@+id/button" ...
3                 app:layout_constraintLeft_toLeftOf="parent"
4                 app:layout_constraintRight_toRightOf="parent/>
5         </>
6
```

Kruhové polohování




```
1 <Button android:id="@+id/buttonA" ... />
2   <Button android:id="@+id/buttonB" ...
3       app:layout_constraintCircle="@+id/buttonA"
4       app:layout_constraintCircleRadius="100dp"
5       app:layout_constraintCircleAngle="45" />
```

Lze použít následující atributy:

- layout_constraintCircle : odkazuje na jiné id widgetu
- layout_constraintCircleRadius : vzdálenost od středu jiného widgetu
- layout_constraintCircleAngle : pod jakým úhlem má být widget umístěn (ve stupních, od 0 do 360)

Rozměrová omezení

Můžete definovat minimální a maximální velikost samotného rozvržení ConstraintLayout:

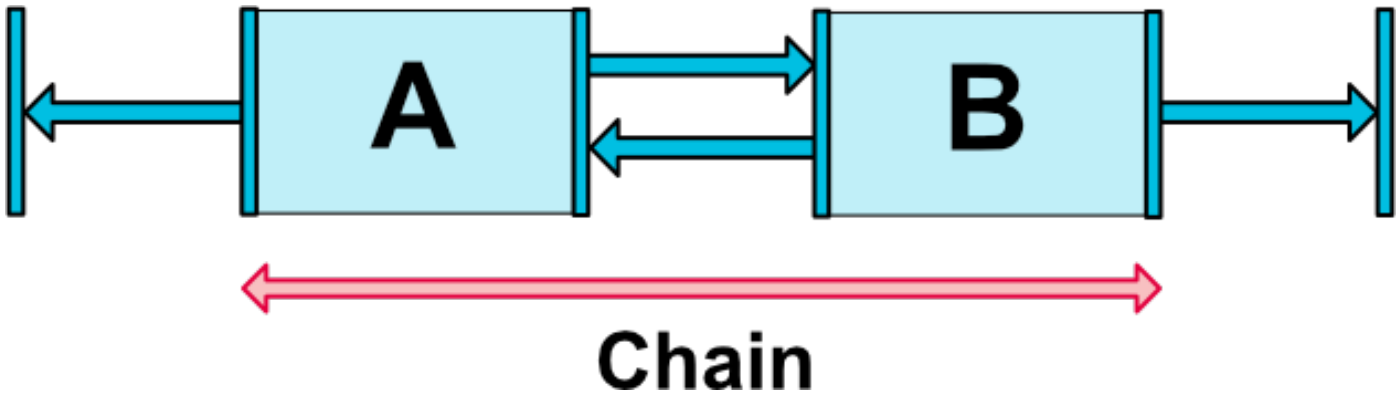
- android:minWidth nastaví minimální šířku rozložení
- android:minHeight nastavuje minimální výšku pro rozložení
- android:maxWidth nastavuje maximální šířku rozvržení
- android:maxHeight nastaví maximální výšku pro rozložení

Rozměr widgetů lze určit nastavením atributů android:layout_width a android:layout_height třemi různými způsoby:

- Pomocí konkrétního rozměru (buď doslovné hodnoty, jako je 123dp, nebo odkazu na Dimension).
- Pomocí WRAP_CONTENT, který požádá widget, aby si vypočítal svou vlastní velikost.
- Použití hodnoty 0dp, což je ekvivalent "MATCH_CONSTRAINT".

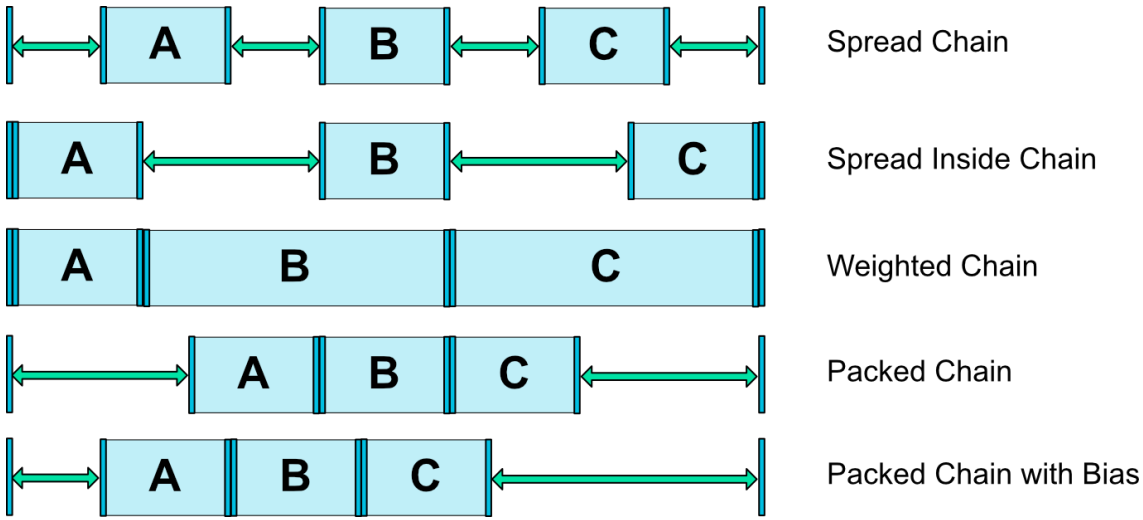
Řetězy

Řetězce poskytují chování podobné skupině v jedné ose (horizontálně nebo vertikálně). Ostatní osy lze omezit nezávisle.



Sada widgetů je považována za řetězec, pokud jsou spolu propojeny obousměrným spojením (na obrázku je zobrazen minimální řetězec se dvěma widgety).

Chain Style I



- CHAIN_SPREAD – prvky budou rozloženy (výchozí styl)
- Weighted chain – in CHAIN_SPREAD pokud jsou některé widgety nastaveny na MATCH_CONSTRAINT, rozdělí dostupný prostor.
- CHAIN_SPREAD_INSIDE – podobné, ale koncové body řetězce nebudou rozloženy.

Chain Style II

- CHAIN_PACKED – tPrvky řetězce budou na sebe nabaleny. Atribut horizontálního nebo vertikálního vychýlení podřízeného prvku pak ovlivní umístění zabalených prvků.

Více informací

https://developer.android.com/training/constraint-layout
https://constraintlayout.com/basics/setup.html
https://www.raywenderlich.com/155-android-listview-tutorial-with-kotlin

Tools

- https://material.io/resources
- https://romannurik.github.io/AndroidAssetStudio/
- https://material.io/color/
- https://www.img-bak.in/
- https://material.io/resizer/
- https://material.io/devices/

Material Design - Cíl

- Vytvořte vizuální jazyk, který spojuje klasické zásady dobrého designu s inovacemi a možnostmi technologie a vědy.
- Vytvořte jednotný základní systém, který umožní jednotný zážitek napříč platformami a velikostmi zařízení. Zásadní jsou mobilní premisy, ale dotyk, hlas, myš a klávesnice jsou prvotřídní vstupní metody.
- Material je návrhový systém vytvořený společností Google, který pomáhá týmům vytvářet vysoce kvalitní digitální zážitky pro systémy Android, iOS, Flutter a web.

Material Design - Komponenty

- Tyče aplikací
- Bunner
- Karta
- Plovoucí tlačítko
- Datové tabulky
- Dialogy
- Seznam, seznam obrázků
- Snackbary
- ToolTip
- ...

Témata materiálů

- Material Theming označuje přizpůsobení aplikace Material Designu tak, aby lépe odrážela značku vašeho produktu.
- Můžete předefinovat:
  - Barva
  - Typografii
  - Tvar, např. změnit velikost nebo rohy tlačítek.

MOBILE APPLICATION DEVELOPMENT

Senzory

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Sensors in mobile device

- Inerciální senzory : Gyroscope, Accelerometer, Magnetometer (e-Compass)
- Optické senzory: Proximity, Ambient Light, RGB Color, Image Sensors (Front/Rear)
- Dotykové senzory: Multi-Touch, Touchless Hover, Pressure Touch
- Senzory prostředí: Temperature, Humidity, Barometric Pressure, Gas (CO...)*
- Bezdrátové senzory/RF:GPS, WiFi, Cellular A-GPS, Bluetooth Low Energy, NFC
- Ostatní senzory:MEMS Microphones, Biometric/Fingerprint*, BioSensors*

MEMS Sensor
* - Senzory budoucnosti

Sensors

https://www.fierceelectronics.com/components/smartphone-sensor-evolution-rolls-rapidly-forward

PrzykĹ‚adowe aplikacje z uĹĽyciem pojedynczego czujnika

- Compass Apps
- Tilt Sensing
- Multi-Touch, Touchless Hover
- Ambient Light/Color or Proximity Sensing
- Ambient Temperature and Humidity Sensing
- ...

PrzykĹ‚adowe aplikacje z uĹĽyciem fuzji danych z czujnikĂłw

- Gesture UI Control (Motion, Proximity)
- Remote Control App (Motion, Multi-Touch, RF)
- Augmented Reality (Inertial, GPS, Image)
- Indoor Navigation and Positioning (Inertial, Pressure, WiFi)
- Context-Aware Mobile Services(VŠECHNY SENZORY !!)
- ...

Android - sensor

Platforma Android obsługuje trzy szerokie kategorie czujników:

- Czujniki ruchu - Czujniki te mierzą siły przyspieszenia oraz. siły obrotowe wzdłuż trzech osi. Kategoria ta obejmuje akcelerometry, czujniki grawitacyjne, żyroskopy i czujniki wektora obrotu.
- Czujniki środowiskowe - czujniki te mierzą różne różne parametry środowiskowe, takie jak temperatura i ciśnienie powietrza, oświetlenie i wilgotność. wilgotność. Kategoria ta obejmuje barometry, fotometry i termometry.
- Czujniki położenia - czujniki te mierzą fizyczne położenie urządzenia. urządzenia. Kategoria ta obejmuje czujniki orientacji i magnetometry.

Android - sensor

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in ¼ T .	Creating a compass.

http://developer.android.com/guide/topics/sensors/sensors_overview.html

Android - sensor

TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

http://developer.android.com/guide/topics/sensors/sensors_overview.html

Sensor Framework

- Určení, které senzory jsou v zařízení k dispozici.
- Určení schopností jednotlivých snímačů, jako je jejich maximální dosah, výrobce, požadavky na napájení a rozlišení.
- Získávání nezpracovaných dat ze senzorů a určení minimální rychlosti získávání dat ze senzorů.
- Registrace a zrušení registrace posluchačů událostí senzorů, kteří monitorují změny senzorů.

Sensor Framework

- Android sensor framework - přístup k nezpracovaným datům ze senzorů a jejich extrakce dat ze senzorů pomocí rámce senzorů Android.
- Framework senzoru je součástí balíčku *android.hardware* a obsahuje následující třídy a rozhraní. obsahuje následující třídy a rozhraní:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorManager

Pomocí této třídy můžeme vytvořit instanci služby senzoru. Tato třída poskytuje různé metody pro přístup k senzorům a jejich výpisu, registraci a odregistrování posluchačů událostí senzorů a získávání informací o orientaci. Tato třída také poskytuje řadu konstant senzorů, které se používají k hlášení. přesnosti snímače, nastavení rychlosti sběru dat a kalibraci snímačů.

Sensor Framework

- Android sensor framework - přístup k nezpracovaným datům ze senzorů a jejich extrakce dat ze senzorů pomocí rámce senzorů Android.
- Framework senzoru je součástí balíčku *android.hardware* a obsahuje následující třídy a rozhraní. obsahuje následující třídy a rozhraní:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

Sensor

Pomocí této třídy můžeme vytvořit instanci konkrétního senzoru. Tato třída poskytuje různé metody pro určení schopností senzoru.

Sensor Framework

- Android sensor framework - přístup k nezpracovaným datům ze senzorů a jejich extrakce dat ze senzorů pomocí rámce senzorů Android.
- Framework senzoru je součástí balíčku *android.hardware* a obsahuje následující třídy a rozhraní. obsahuje následující třídy a rozhraní:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorEvent

Systém používá tuto třídu k vytvoření objektu události senzoru, který poskytuje informace o události senzoru. Objekt události senzoru obsahuje následující údaje informace: nezpracovaná data senzoru, typ senzoru, který událost generoval, přesnost dat a časové razítko události.

Sensor Framework

- Android sensor framework - přístup k nezpracovaným datům ze senzorů a jejich extrakce dat ze senzorů pomocí rámce senzorů Android.
- Framework senzoru je součástí balíčku *android.hardware* a obsahuje následující třídy a rozhraní. obsahuje následující třídy a rozhraní:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorEventListener

Pomocí tohoto rozhraní můžete vytvořit dvě metody zpětného volání, které přijímají oznámení (události senzoru) při změně hodnot senzoru nebo při změně přesnosti senzoru.

Typowe kroki

- Identifikace snímače a jeho schopnosti
- Monitorování událostí snímače

Identyfikacja czujnikÄłw

Identifikace senzorů a schopností senzorů za běhu je užitečná, pokud vaše aplikace obsahuje funkce, které se spoléhají na určité typy senzorů nebo schopností. Můžete například chtít identifikovat všechny snímače, které jsou na zařízení přítomny, a zakázat všechny funkce aplikace, které se spoléhají na snímače, které nejsou přítomny. Podobně můžete chtít identifikovat všechny snímače určitého typu, abyste mohli vybrat implementaci snímače, která má pro vaši aplikaci optimální výkon.

Typowe kroki

- Identifikace snímače a jeho schopnosti
- Monitorování událostí snímače

Monitorowanie czujnikĂłw

Monitorování událostí snímače je způsob, jak získat surová data ze snímače. Událost senzoru nastane pokaždé, když senzor zjistí změnu parametrů, které měří. Událost senzoru poskytuje čtyři informace: název senzoru, který událost spustil, časovou značku události, přesnost události a nezpracovaná data senzoru, který událost spustil.

1. Získání odkazu na službu senzoru

```
^^Iprivate lateinit var sensorManager: SensorManager
...
^^IsensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
^^I
```

2. Zobrazení seznamu všech snímačů v zařízení

```
val deviceSensors: List<Sensor> = sensorManager.getSensorList(Sensor.TYPE_ALL)
^^I
```

2. Nebo použijte jinou konstantu místo TYPE_ALL such as TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, nebo TYPE_GRAVITY.

```
if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
    // Success! There's a magnetometer.
} else{
    // Failure! No magnetometer.
}
^^I
```

Capabilities

```
1  private lateinit var sensorManager: SensorManager
2  private var mSensor: Sensor? = null
3
4  ...
5
6  sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
7
8  if (sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
9      val gravSensors: List<Sensor> = sensorManager.getSensorList(Sensor.TYPE_GRAVITY)
10     // Use the version 3 gravity sensor.
11     mSensor = gravSensors.firstOrNull{ it.vendor.contains("Google LLC") && it.version
           == 3 }
12 }
13 if (mSensor == null){
14     // Use the accelerometer.
15     mSensor = if (sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
16         sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
17     } else{
18         // Sorry, there are no accelerometers on your device.
19         // You can't play this game.
20         null
21     }
22 }
```

UĹĽyteczne metody do pobrania weryfikacji wĹ‚asnoĹ›ci czujnikÄ‚Å‚w

- *getResolution(), getMaximumRange()*
- *getPower()*
- *getMinDelay()*

Monitorowanie zdarzeĹ„ z czujnikĂłw

Pro sledování nezpracovaných dat ze senzorů je třeba implementovat dvě metody zpětného volání, které jsou dostupné prostřednictvím rozhraní *SensorEventListener*.: onAccuracyChanged() and onSensorChanged()

Zmiany dokĹ‚adnoĹ›ci czujnikĂłw

```
1     override fun onAccuracyChanged(sensor: Sensor, accuracy: Int){
2         // Do something here if sensor accuracy changes.
3     }
4 ^^I
```

Czujnik zgĹ‚asza nowÄ… wartoĹ›Ä‡

```
1     override fun onSensorChanged(event: SensorEvent){
2         // The light sensor returns a single value.
3         // Many sensors return 3 values, one for each axis.
4         val lux = event.values[0]
5         // Do something with this sensor value.
6     }
7 ^^I
```

Monitorowanie zdarzeĹ„ z czujnikĂłw

```
1  class SensorActivity : Activity(), SensorEventListener{
2      private lateinit var sensorManager: SensorManager
3      private var mLight: Sensor? = null
4
5      public override fun onCreate(savedInstanceState: Bundle?){
6          super.onCreate(savedInstanceState)
7          setContentView(R.layout.main)
8
9          sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
10         mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)
11     }
12
13     override fun onAccuracyChanged(sensor: Sensor, accuracy: Int){
14         // Do something here if sensor accuracy changes.
15     }
16
17     override fun onSensorChanged(event: SensorEvent){
18         // The light sensor returns a single value.
19         // Many sensors return 3 values, one for each axis.
20         val lux = event.values[0]
21         // Do something with this sensor value.
22     }
23 ^^I
```

Monitorowanie zdarzeĹ„ czujnikĂłw - rejestracja

```
1
2       override fun onResume(){
3           super.onResume()
4           mLight?.also{ light ->
5               sensorManager.registerListener(this, light, SensorManager.
                    SENSOR_DELAY_NORMAL)
6           }
7       }
8
9       override fun onPause(){
10          super.onPause()
11          sensorManager.unregisterListener(this)
12      }
13  }
```

Google Play filters

```
1  <uses-feature android:name="android.hardware.sensor.accelerometer"
2                android:required="true" />
```

Najlepsze praktyki w zakresie uzyskiwania dostÄ™pu i korzystania z czujnikÅ‚w

- Sběr dat ze senzorů pouze v popředí
- Zrušení registrace posluchačů senzorů
- Testování pomocí emulátoru Androidu
- Neblokujte metodu onSensorChanged()
- Vyhněte se používání zastaralých metod nebo typů senzorů
- Ověřte senzory před jejich použitím
- Pečlivě vybírejte zpoždění senzorů

Gromadzenie danych z czujnikÅ‚w w tle

Na zařízeních se systémem Android 9 (úroveň API 28) nebo novějším:

- Senzory, které používají režim nepřetržitého hlášení, jako jsou akcelerometry a gyroskopy, nepřijímají události.
- Senzory, které používají režim hlášení při změně nebo jednorázové hlášení, nepřijímají události.

Najlepsze praktyki w zakresie uzyskiwania dostÄ™pu i korzystania z czujnikÅ‚w

- Sběr dat ze senzorů pouze v popředí
- Zrušení registrace posluchačů senzorů
- Testování pomocí emulátoru Androidu
- Neblokujte metodu onSensorChanged()
- Vyhněte se používání zastaralých metod nebo typů senzorů
- Ověřte senzory před jejich použitím
- Pečlivě vybírejte zpoždění senzorů

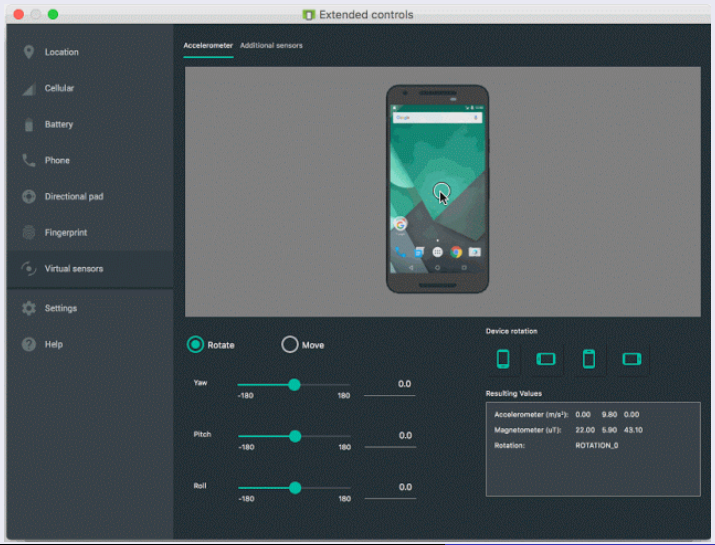
Wyrejestrowanie listenerÅ‚w czujnikÅ‚w

Nezapomeňte zrušit registraci posluchače senzoru, jakmile ukončíte jeho používání nebo když je činnost senzoru pozastavena. Pokud je posluchač senzoru zaregistrován a jeho činnost je pozastavena, bude senzor nadále získávat data a využívat prostředky baterie, pokud jeho registraci neodhlásíte.

Najlepsze praktyki w zakresie uzyskiwania dostÄ™pu i korzystania z czujnikÅ‚w

- Sběr dat ze senzorů pouze v popředí
- Zrušení registrace posluchačů senzorů
- Testování pomocí emulátoru Androidu
- Neblokujte metodu onSensorChanged()
- Vyhněte se používání zastaralých metod nebo typů senzorů
- Ověřte senzory před jejich použitím
- Pečlivě vybírejte zpoždění senzorů

Test with the Android Emulator



Najlepsze praktyki w zakresie uzyskiwania dostÄ™pu i korzystania z czujnikÅ‚w

- Sběr dat ze senzorů pouze v popředí
- Zrušení registrace posluchačů senzorů
- Testování pomocí emulátoru Androidu
- Neblokujte metodu onSensorChanged()
- Vyhněte se používání zastaralých metod nebo typů senzorů
- Ověřte senzory před jejich použitím
- Pečlivě vybírejte zpoždění senzorů

Nie blokujemy metody *onSensorChanged()*

- Data ze senzorů se mohou měnit velkou rychlostí - systém může metodu *onSensorChanged(SensorEvent)* volat poměrně často.
- Filtrování nebo redukce dat ze senzorů by se měly provádět mimo metodu onSensorChanged(SensorEvent)

Najlepsze praktyki w zakresie uzyskiwania dostÄ™pu i korzystania z czujnikÅ‚w

- Sběr dat ze senzorů pouze v popředí
- Zrušení registrace posluchačů senzorů
- Testování pomocí emulátoru Androidu
- Neblokujte metodu onSensorChanged()
- Vyhněte se používání zastaralých metod nebo typů senzorů
- Ověřte senzory před jejich použitím
- Pečlivě vybírejte zpoždění senzorů

OstroĹĽnie dobieraj opÅ‚Ĺşnienia czujnikÅ‚w

- Při registraci senzoru pomocí metody registerListener(),Ujistěte se, že jste zvolili rychlost přenosu dat, která je vhodná pro vaši aplikaci nebo případ použití.
- Umožnění systému odesílat další data, která nepotřebujete, plýtvá systémovými prostředky a spotřebovává energii baterie.

Motion Sensor

Senzory pohybu jsou užitečné pro monitorování **ruch urzÄ...dzenia, taki jak przechylanie, potrzÄ...sanie, obracanie lub koĹ‚ysanie.**
. Možné architektury senzorů se liší v závislosti na typu senzoru:

- Senzory gravitace, lineárního zrychlení, rotačního vektoru, významného pohybu, čítače kroků a detektoru kroků jsou buď hardwarové, nebo softwarové.
- Snímače akcelerometru a gyroskopu jsou vždy hardwarové.

`https://developer.android.com/guide/topics/sensors/sensors_motion`

Example - Motion Sensor

- Źródło + Opishttps://www.raywenderlich.com/10838302-sensors-tutorial-for-android-getting-started

Position sensors

Snímače polohy jsou užitečné pro určení fyzické polohy zařízení ve světovém referenčním systému. Například snímač geomagnetického pole lze použít ve spojení s akcelerometrem k určení polohy zařízení vzhledem k severnímu magnetickému pólu.

Výpočet orientace zařízení

```
1 private lateinit var sensorManager: SensorManager
2 ...
3 // Rotation matrix based on current readings from accelerometer and magnetometer.
4 val rotationMatrix = FloatArray(9)
5 SensorManager.getRotationMatrix(rotationMatrix, null, accelerometerReading,
      magnetometerReading)
6
7 // Express the updated rotation matrix as three orientation angles.
8 val orientationAngles = FloatArray(3)
9 SensorManager.getOrientation(rotationMatrix, orientationAngles)
```

Výpočet orientace zařízení

Systém vypočítává orientační úhly pomocí snímače geomagnetického pole zařízení v kombinaci s akcelerometrem zařízení. Pomocí těchto dvou hardwarových snímačů systém poskytuje údaje pro následující tři úhly orientace:

- **Azimuth (degrees of rotation about the -z axis)** Jedná se o úhel mezi aktuálním směrem kompasu zařízení a magnetickým severem. Pokud horní hrana zařízení směřuje k magnetickému severu, azimut je 0 stupňů; pokud horní hrana směřuje k jihu, azimut je 180 stupňů. Podobně pokud horní hrana směřuje na východ, azimut je 90 stupňů; pokud horní hrana směřuje na západ, azimut je 270 stupňů. .

Výpočet orientace zařízení

Systém vypočítává orientační úhly pomocí snímače geomagnetického pole zařízení v kombinaci s akcelerometrem zařízení. Pomocí těchto dvou hardwarových snímačů systém poskytuje údaje pro následující tři úhly orientace:

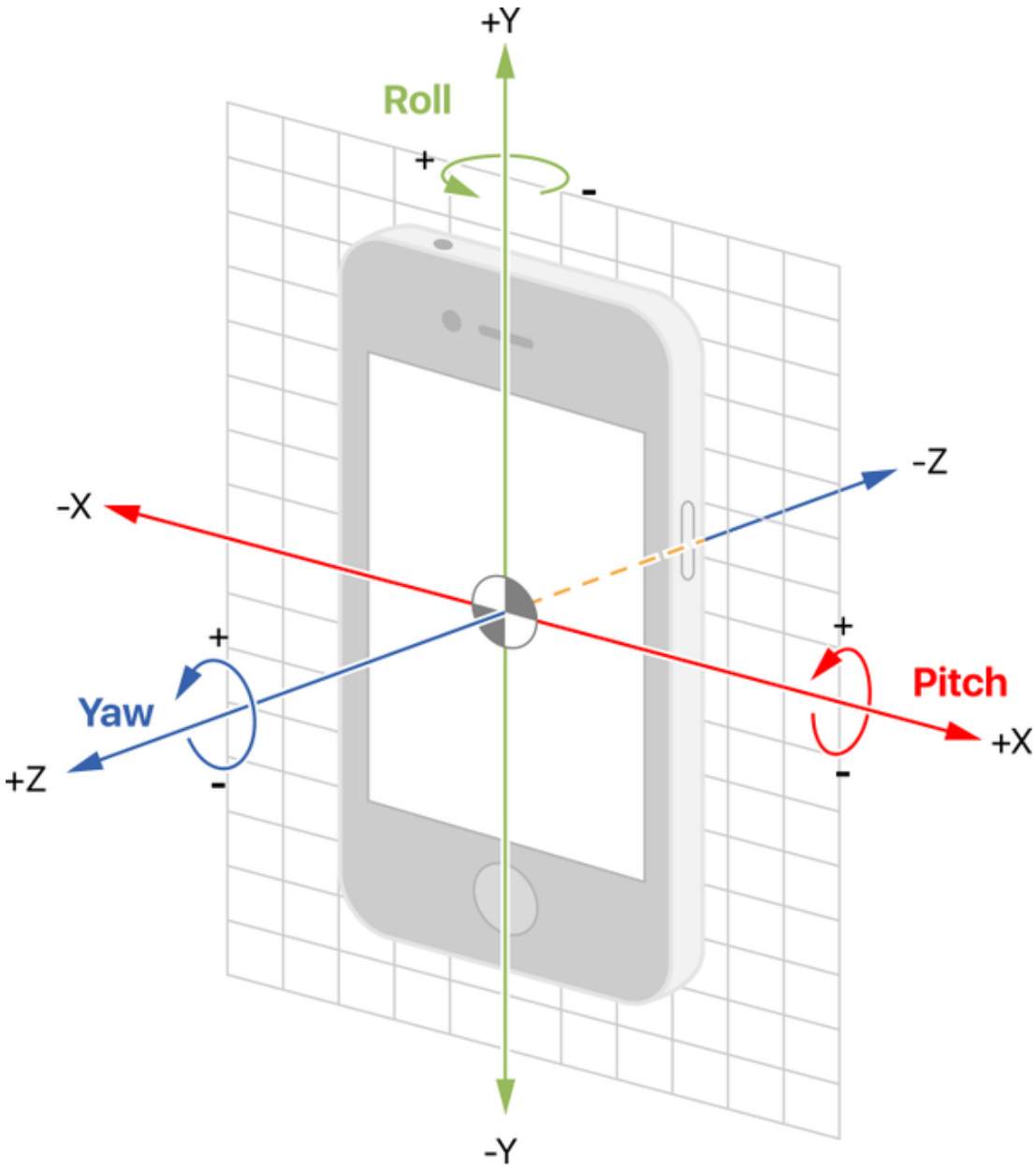
- **Pitch (degrees of rotation about the x axis)** Jedná se o úhel mezi rovinou rovnoběžnou s obrazovkou zařízení a rovinou rovnoběžnou se zemí. Pokud držíte zařízení rovnoběžně se zemí, přičemž spodní okraj zařízení je nejblíže k vám, a nakloníte horní okraj zařízení směrem k zemi, úhel sklonu se stane kladným. Naklonění v opačném směru - oddálení horního okraje zařízení od země - způsobí, že úhel náklonu bude záporný. Rozsah hodnot je -180 stupňů až 180 stupňů. .

Výpočet orientace zařízení

Systém vypočítává orientační úhly pomocí snímače geomagnetického pole zařízení v kombinaci s akcelerometrem zařízení. Pomocí těchto dvou hardwarových snímačů systém poskytuje údaje pro následující tři úhly orientace:

- **Roll (degrees of rotation about the y axis)** Jedná se o úhel mezi rovinou kolmou k obrazovce zařízení a rovinou kolmou k zemi. Pokud držíte zařízení rovnoběžně se zemí, přičemž spodní okraj zařízení je nejblíže k vám, a nakloníte levý okraj zařízení směrem k zemi, úhel sklonu se stane kladným. Naklonění v opačném směru - posunutí pravého okraje zařízení směrem k zemi - způsobí, že úhel sklonu bude záporný. Rozsah hodnot je od -90 stupňů do 90 stupňů.

Device Orientation



Environment sensors

- Pomocí těchto snímačů můžete sledovat relativní vlhkost, intenzitu okolního světla, okolní tlak a teplotu v blízkosti zařízení se systémem Android.
- Všechny čtyři snímače okolního prostředí jsou hardwarové a jsou k dispozici pouze v případě, že je výrobce zařízení zabudoval do zařízení.
- Senzory prostředí vracejí pro každou datovou událost jednu hodnotu senzoru.

Sensor	Sensor event data	Units of measure	Data description
TYPE_AMBIENT_TEMPERATURE	event.values[0]	°C	Ambient air temperature.
TYPE_LIGHT	event.values[0]	lx	Illuminance.
TYPE_PRESSURE	event.values[0]	hPa or mbar	Ambient air pressure.
TYPE_RELATIVE_HUMIDITY	event.values[0]	%	Ambient relative humidity.
TYPE_TEMPERATURE	event.values[0]	°C	Device temperature.[1]

Example

- AndroidManifest
- Service
- Callback methods
- Emulator

https://www.raywenderlich.com/10838302-sensors-tutorial-for-android-getting-started

MOBILE APPLICATION DEVELOPMENT

Lokalita

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Lokalita

Lokalita

Služba pro určení polohy zařízení a nepřímo i uživatele. V mobilních systémech je poloha jednou z jedinečných funkcí pro vytváření aplikací zohledňujících polohu.

- Umožňuje určit polohu zařízení.
- Obecná poloha
- Přesné určení polohy
- Služby Google Play - doporučená metoda určování polohy v systému Android

Příklady aplikací

- Informace specifické pro danou lokalitu (místní předpověď počasí, místní zprávy, koncentrace alergenů)
- Informace o blízkých zdrojích (banka, lékárna, hospoda)
- Interaktivní mapy a turistické informace
- Mobilní reklama závislá na poloze
- Řízení mobilních pracovníků

Google Play services

- Mechanismus pro určení polohy na základě dat od různých poskytovatelů, např. modulu GNSS (GPS), modulu WiFi nebo Bluetooth.
- Fúzovaný poskytovatel polohy
- Rychlejší určování polohy
- Snížení spotřeby energie
- Další funkce, např. geofencing, detekce aktivity.

Přidání knihovny služeb Google Play

```
1 apply plugin: 'com.android.application'
2
3 ...
4
5 dependencies{
6     implementation 'com.google.android.gms:play-services-location:21.0.0'
7 }
```

Steps required to define a location

- Definice typu umístění
- Žádost o povolení umístění zařízení
- Stahování polohy (poslední známá, cyklické stahování polohy)
- Použití polohy, např. zobrazení bodu na mapě

Povolení definice

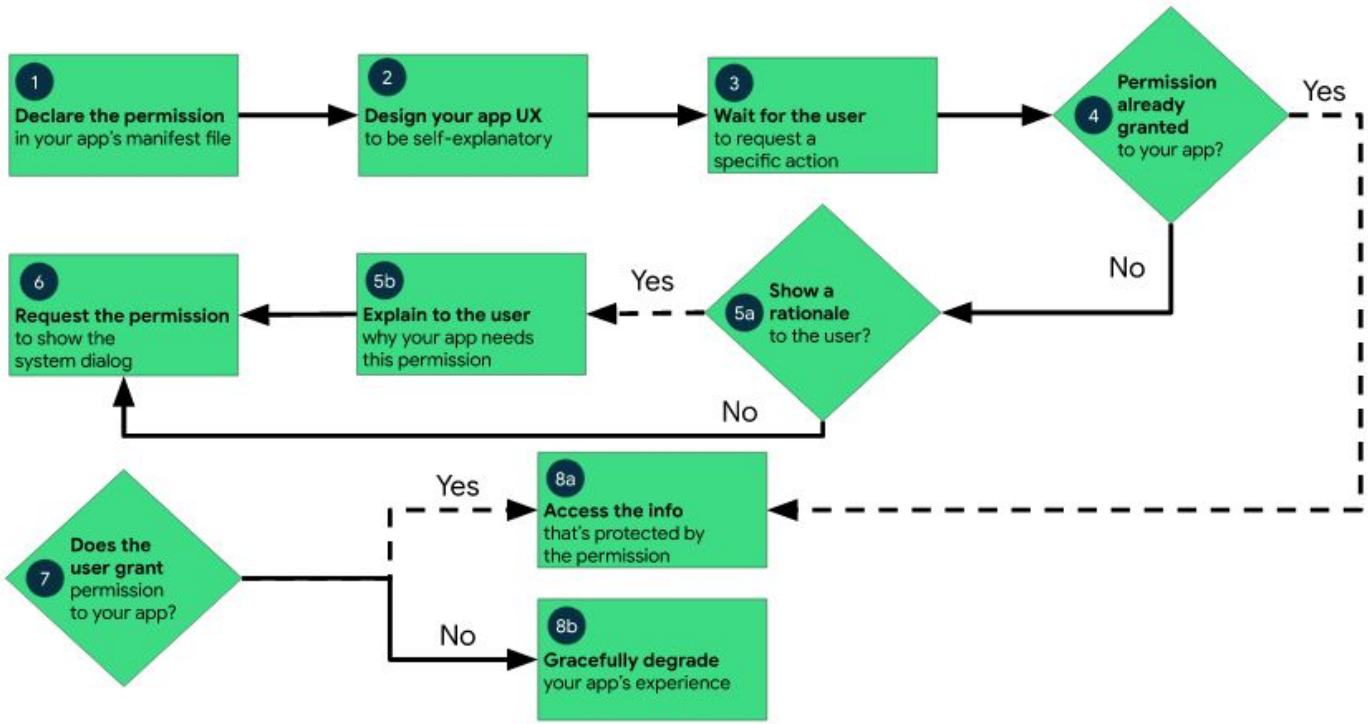
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
...
  <!-- Always include this permission -->
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

  <!-- Include only if your app benefits from precise location access. -->
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <!-- Recommended for Android 9 (API level 28) and lower. -->
  <!-- Required for Android 10 (API level 29) and higher. -->
...
  <!-- Required only when requesting background location access on
       Android 10 (API level 29) and higher. -->
  <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"
       />
...
  <application...>
    <service
      android:name="MyNavigationService"
      android:foregroundServiceType="location" ... >
    </service>
  </application>
...
</manifest>
```

Typy míst

- Obecné/hrubé umístění
- Přesná poloha
- Umístění v popředí
- Umístění v pozadí

Žádost o povolení - workflow



Příklad žádosti o povolení

```
1     private fun checkPermission()
2   {
3         if(ContextCompat.checkSelfPermission(this,
4                 Manifest.permission.ACCESS_FINE_LOCATION
5             ) != PackageManager.PERMISSION_GRANTED)
6             requestPermissionLauncher.launch(Manifest.permission.
                  ACCESS_FINE_LOCATION)
7     }
8     private val requestPermissionLauncher =
9         registerForActivityResult(
10            ActivityResultContracts.RequestPermission()
11        ){
12                isGranted: Boolean ->
13            if (isGranted){
14                Log.i("Permission: ", "Granted")
15            } else{
16                Log.i("Permission: ", "Denied")
17            }
18        }
```

Použití Fused Location Provider - poslední známá poloha

Definice objektu

```
1 private lateinit var fusedLocationClient: FusedLocationProviderClient
```

Inicializace

```
1 override fun onCreate(savedInstanceState: Bundle?){
2 ...
3         fusedLocationClient = LocationServices.getFusedLocationProviderClient(
              this)
4
5 }
```

Vyhledání poslední známé polohy

```
1         checkPermission()
2         fusedLocationClient.lastLocation
3             .addOnSuccessListener{ location : Location? ->
4                 val myPosition = location?.let{
5                     LatLng(it.latitude,it.longitude)
6                 }
7                 myPosition?.let{
8                     mMap.addMarker(MarkerOptions().position(myPosition).title("
                          My position"))
9                     mMap.moveCamera(CameraUpdateFactory.newLatLng(myPosition))
10                }
11            }
```

Průběžná poloha - I

- Definice objektu
- Inicializace
- Definice návratových metod
- Povolení a zakázání informací o aktualizaci polohy
- Volba obnovovací frekvence

Kontinuální umístění II

Definice objektů

```
1     private lateinit var locationRequest: LocationRequest
2     private lateinit var locationCallback: LocationCallback
```

Inicializace

```
1         fusedLocationClient = LocationServices.getFusedLocationProviderClient(
              this)
2         locationRequest = LocationRequest.Builder(Priority.
              PRIORITY_HIGH_ACCURACY,
3             500)
4             .build()
5
6         locationCallback = object : LocationCallback(){
7             override fun onLocationResult(locationResult: LocationResult){
8                 if (locationResult != null){
9                     super.onLocationResult(locationResult)
10                    locationResult.lastLocation?.let{
11
12                    //own code
13
14                    }
15
16                }
17            }
18        }
```

Průběžná poloha III

Aktivace informací o aktualizaci polohy

```
1            val addTask= fusedLocationClient.requestLocationUpdates(
                 locationRequest, locationCallback, Looper.myLooper())
2            addTask.addOnCompleteListener{task->
3                if (task.isSuccessful){
4                    Log.d("startStopRequestLocation", "Start loop Location
                         Callback.")
5                } else{
6                    Log.d("startStopRequestLocation", "Failed start  Location
                         Callback.")
7                }
8            }
```

Deaktivace informací o aktualizaci polohy

```
1            val removeTask = fusedLocationClient.removeLocationUpdates(
                 locationCallback)
2            removeTask.addOnCompleteListener{ task ->
3                if (task.isSuccessful){
4                    Log.d("startStopRequestLocation", "Location Callback removed
                         .")
5                } else{
6                    Log.d("startStopRequestLocation", "Failed to remove Location
                         Callback.")
7                }
8            }
```

MOBILE APPLICATION DEVELOPMENT

MVVM

Innovative Open Source courses for Computer Science

30.05.2021

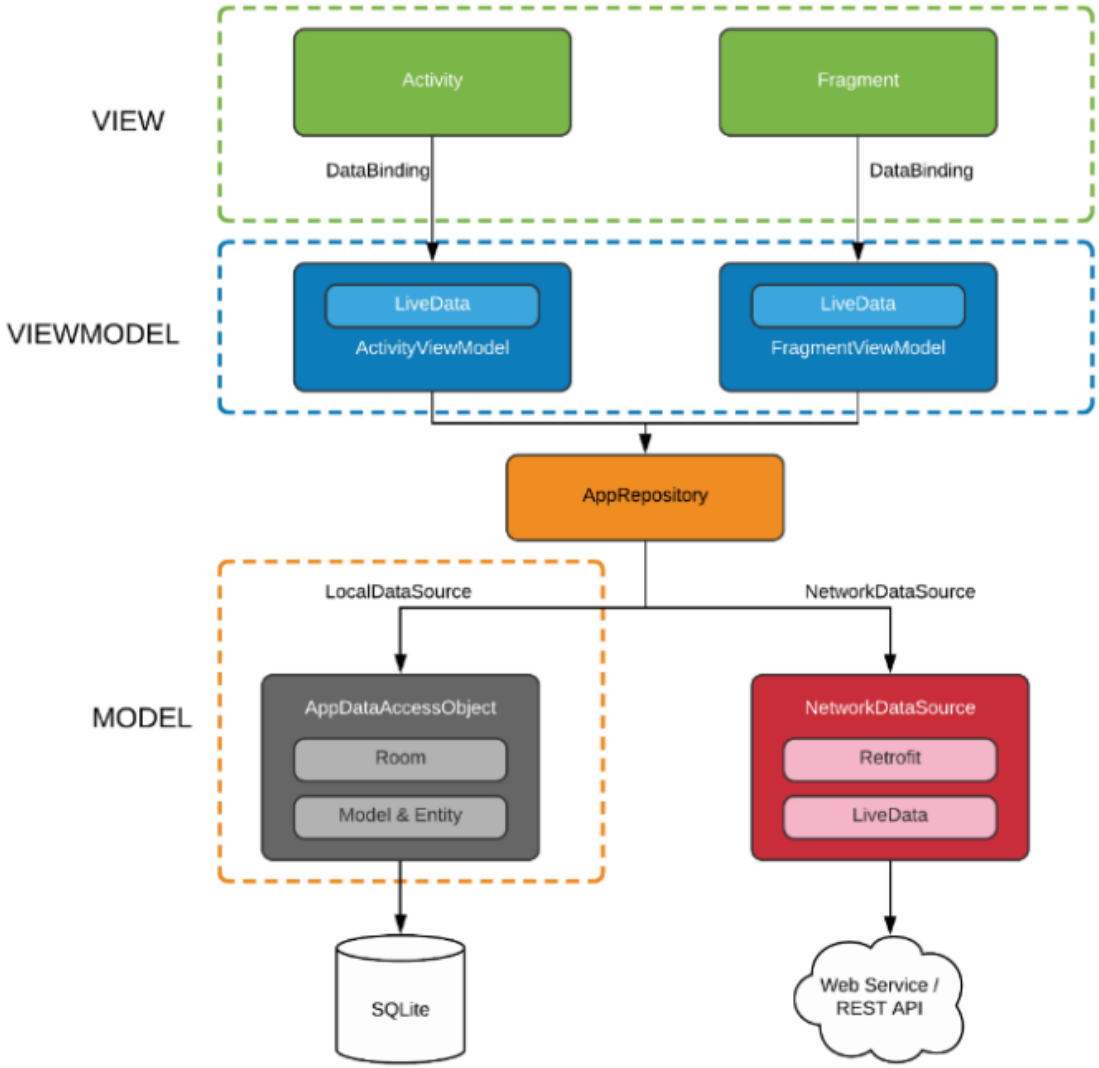
Funded by
the European Union

MVVM

Model — View — ViewModel je průmyslově uznávaný vzor softwarové architektury, který překonává všechny nevýhody návrhových vzorů MVP a MVC. MVVM navrhuje oddělit logiku prezentace dat (Views nebo UI) od hlavní části obchodní logiky aplikace.

- **Model** - Zde jsou uložena data aplikace. Nemůže přímo komunikovat s pohledem. Obecně se doporučuje zpřístupnit data ViewModelu prostřednictvím Observables.
- **View** - Představuje uživatelské rozhraní aplikace bez aplikační logiky. Sleduje ViewModel.
- **ViewModel** - Funguje jako spojovací článek mezi Modelem a Zobrazením. Je zodpovědný za transformaci dat z Modelu. Poskytuje datové toky do View. Používá také zpětná volání pro aktualizaci Zobrazení. Vyžádá si data z Modelu.

MVVM -2


VIEW
Activity
Fragment
DataBinding
DataBinding
VIEWMODEL
LiveData
ActivityViewModel
LiveData
FragmentViewModel
AppRepository
LocalDataSource
NetworkDataSource
MODEL
AppDataAccessObject
Room
Model & Entity
NetworkDataSource
Retrofit
LiveData
SQLite
Web Service /
REST API

Komponenty architektury

- Kolekce knihoven, které vám pomohou navrhovat robustní, testovatelné a udržovatelné aplikace.
- Slouží jako hlavní vodítko při budování páteře architektury našeho projektu.
- Pomocí komponent architektury Androidu se nemusíme příliš starat o správu životního cyklu aplikace ani o načítání dat do našeho uživatelského rozhraní

Komponenty

- ViewModel
- LiveData
- Lifecycle
- Extend LiveData

ViewModel

- Třída ViewModel je navržena tak, aby uchovávala a spravovala data související s uživatelským rozhraním s ohledem na životní cyklus.
- Díky tomu mohou data přežít změny konfigurace, například otočení obrazovky.
- Architecture Components poskytuje pomocnou třídu ViewModel pro řadič uživatelského rozhraní, který je zodpovědný za přípravu dat pro uživatelské rozhraní.
- Objekty ViewModelu jsou automaticky uchovávány při změnách konfigurace, takže data, která obsahují, jsou okamžitě k dispozici další instanci aktivity nebo fragmentu.

ViewModel benefity

- Umožňuje zachovat stav uživatelského rozhraní
- Umožňuje přístup k obchodní logice.

Perzistence

ViewModel umožňuje přežití jak prostřednictvím stavu, který ViewModel uchovává, tak prostřednictvím operací, které ViewModel spouští. Toto ukládání do mezipaměti znamená, že nemusíte znovu načítat data prostřednictvím běžných změn konfigurace, jako je například otočení obrazovky.

Přístup k podnikové logice

ViewModel je vhodným místem pro zpracování obchodní logiky ve vrstvě uživatelského rozhraní. ViewModel má také na starosti zpracování událostí a jejich delegování na další vrstvy hierarchie, když je třeba použít obchodní logiku pro změnu dat aplikace.

ViewModel - příklad

Definice ViewModelu

```
1  class MyViewModel : ViewModel(){
2      private val users: MutableLiveData<List<User>> by lazy{
3          MutableLiveData<List<User>>().also{
4              loadUsers()
5          }
6      }
7
8      fun getUsers(): LiveData<List<User>>{
9          return users
10     }
11
12     private fun loadUsers(){
13         // Do an asynchronous operation to fetch users.
14     }
15 }
```

Přístup k seznamu z aktivity je následující

```
1  override fun onCreate(savedInstanceState: Bundle?){
2
3          // Use the 'by viewModels()' Kotlin property delegate
4          // from the activity-ktx artifact
5          val model: MyViewModel by viewModels()
6          model.getUsers().observe(this, Observer<List<User>>{ users ->
7              // update UI
8          })
```

LiveData

- LiveData je třída pozorovatelného držitele dat.
- Na rozdíl od běžných pozorovatelných dat LiveData zohledňují životní cyklus, což znamená, že respektují životní cyklus ostatních komponent aplikace, jako jsou aktivity, fragmenty nebo služby.
- Toto povědomí zajišťuje, že LiveData aktualizuje pouze pozorovatele komponent aplikace, které jsou v aktivním stavu životního cyklu.
- Služba LiveData se řídí vzorem pozorovatele. LiveData upozorňuje objekty pozorovatelů, když se změní stav životního cyklu
- LiveData také automaticky předává existující hodnotu, pokud existuje, všem nově registrovaným objektům Observer.
- Ve spojení s funkcí automatického odstraňování je díky tomu LiveData velmi vhodný pro řešení změn konfigurace.

Typy v LiveData

- LiveData - ve výchozím nastavení jsou neměnná. Pomocí LiveData můžeme data pouze pozorovat a nemůžeme je nastavovat.
- MutableLiveData - jsou proměnlivá a jsou podtřídou LiveData. V MutableLiveData můžeme pozorovat a nastavovat hodnoty pomocí metod postValue() a setValue().
- MediatorLiveData - může pozorovat jiné objekty LiveData, například zdroje, a reagovat na jejich události onChange().

Prodloužení LiveData

- LiveData je komponenta, která si uvědomuje svůj životní cyklus, a proto vykonává své funkce podle stavu životního cyklu ostatních komponent aplikace.
- Pokud je stav životního cyklu pozorovatele aktivní, tj. buď STARTED, nebo RESUMED, pouze tehdy LiveData aktualizuje komponentu aplikace.

```
1  class StockLiveData(symbol: String) : LiveData<BigDecimal>(){
2      private val stockManager = StockManager(symbol)
3
4      private val listener ={ price: BigDecimal ->
5          value = price
6      }
7
8      override fun onActive(){
9          stockManager.requestPriceUpdates(listener)
10     }
11
12     override fun onInactive(){
13         stockManager.removeUpdates(listener)
14     }
15 }
```

Prodloužení LiveData - 2

- **onActive()** - metoda je volána, pokud má objekt LiveData aktivního pozorovatele. To znamená, že z této metody je třeba začít pozorovat aktualizace cen akcií.
- **onInactive()** - metoda se volá, když objekt LiveData nemá žádné aktivní pozorovatele. Protože žádný pozorovatel neposlouchá, není důvod, aby zůstával připojen.

Kroky k vytvoření aplikace s návrhovým paternem MVVM

- Přidání DataBindingu a implementací do souboru Gradle
- Vytvoření nové třídy pro Model
- Vytvoření nové třídy pro ViewModel
- Vylepšení třídy View
- Změna rozvržení

Konfigurace projektu

build.gradle(app)

```
1  android{
2      compileSdk 31
3
4      dataBinding{
5          enabled true
6      }
7
8
9      ...
10
11 dependencies{
12     //ViewModel
13     implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1'
14     implementation 'androidx.activity:activity-ktx:1.4.0'
15     //Lifecycle
16     implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.4.1"
17 ...
18 }
19 }
```

Model

```
1 data class SensorData(
2     var accX: Float,
3     var accY: Float,
4     var accZ: Float,
5     var gyroX: Float,
6     var gyroY: Float,
7     var gyroZ: Float,
8     var light: Float
9 )
```

ViewModel

```
class SensorViewModel(application: Application): AndroidViewModel(application){
  private val _sensor = SensorDataLiveData(application)
  private var _pauseReading = MutableLiveData<Boolean>()

  val sensor: LiveData<SensorData>
    get() = _sensor

  fun getPauseReading(): MutableLiveData<Boolean>{
    return _pauseReading
  }

  fun changeButtonStatus()
 {
    if(_pauseReading.value==true) _sensor.registerListeners()
    else _sensor.unregisterListeners()
    _pauseReading.value?.let{
      _pauseReading.value = !it
    }
  }
  init{
      _pauseReading = MutableLiveData(false)
  }
}
```

Změna kódu zobrazení

```
1     private lateinit var binding: ActivityMainBinding
2     private val sensorViewModel: SensorViewModel by viewModels()
3
4     override fun onCreate(savedInstanceState: Bundle?){
5         requestWindowFeature(Window.FEATURE_NO_TITLE)
6         requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
7
8         super.onCreate(savedInstanceState)
9         binding = ActivityMainBinding.inflate(layoutInflater)
10        setContentView(binding.root)
11        binding.sensorViewModel = sensorViewModel
12        binding.lifecycleOwner = this
13    }
```

Vylepšené rozvržení

```
1 <layout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:app="http://schemas.android.com/apk/res-auto"
3     xmlns:tools="http://schemas.android.com/tools">
4
5     <data>
6         <variable
7             name="sensorViewModel"
8             type="edu.zut.erasmus_plus.sensors.viewmodel.SensorViewModel" />
9     </data>
```

MOBILE APPLICATION DEVELOPMENT

Úložiště

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Možnosti úložiště systému Android

- Úložiště specifické pro aplikace
- Sdílené úložiště
- Předvolby
- Databáze

App-specific storage

Soubory určené pouze pro použití vaší aplikace ukládejte buď do vyhrazených adresářů v interním úložišti, nebo do různých vyhrazených adresářů v externím úložišti. Adresáře v interním úložišti používejte k ukládání citlivých informací, ke kterým by ostatní aplikace neměly mít přístup.

Možnosti úložiště systému Android

- Úložiště specifické pro aplikace
- Sdílené úložiště
- Předvolby
- Databáze

Sdílené úložiště

Ukládejte soubory, které vaše aplikace hodlá sdílet s jinými aplikacemi, včetně médií, dokumentů a dalších souborů.

Možnosti úložiště systému Android

- Úložiště specifické pro aplikace
- Sdílené úložiště
- Předvolby
- Databáze

Předvolby

Ukládejte soukromá, primitivní data v párech klíč-hodnota.

Možnosti úložiště systému Android

- Úložiště specifické pro aplikace
- Sdílené úložiště
- Předvolby
- Databáze

Databáze

Ukládání strukturovaných dat do soukromé databáze pomocí knihovny pro perzistenci Room.

Přehled úložišť

	Type of content	Access method	Permissions needed	Can other apps access?	Files removed on app uninstall?
App-specific files	Files meant for your app's use only	From internal storage, `getFilesDir()` or `getCacheDir()`	Never needed for internal storage	No, if files are in a directory within internal storage	Yes
		From external storage, `getExternalFilesDir()` or `getExternalCacheDir()`	Not needed for external storage when your app is used on devices that run Android 4.4 (API level 19) or higher	Yes, if files are in a directory within external storage	
Media	Shareable media files (images, audio files, videos)	`MediaStore` API	`READ_EXTERNAL_STORAGE` or `WRITE_EXTERNAL_STORAGE` when accessing other apps' files on Android 10 (API level 29) or higher	Yes, though the other app needs `READ_EXTERNAL_STORAGE` permission	No
			Permissions are required for all files on Android 9 (API level 28) or lower		
Documents and other files	Other types of shareable content, including downloaded files	Storage Access Framework	None	Yes, through the system file picker	No
App preferences	Key-value pairs	Jetpack Preferences library	None	No	Yes
Database	Structured data	Room persistence library	None	No	Yes

Který z nich si vybrat?

- Kolik místa potřebují vaše data?
- Jak spolehlivý musí být přístup k datům?
- Jaký druh dat potřebujete ukládat?
- Měla by být data pro vaši aplikaci soukromá?

Který z nich si vybrat?

- Kolik místa potřebují vaše data?
- Jak spolehlivý musí být přístup k datům?
- Jaký druh dat potřebujete ukládat?
- Měla by být data pro vaši aplikaci soukromá?

Kolik místa potřebují vaše data?

Interní úložiště má omezený prostor pro data specifická pro aplikaci. Pokud potřebujete uložit značné množství dat, použijte jiné typy úložišť.

Který z nich si vybrat?

- Kolik místa potřebují vaše data?
- Jak spolehlivý musí být přístup k datům?
- Jaký druh dat potřebujete ukládat?
- Měla by být data pro vaši aplikaci soukromá?

Jak spolehlivý musí být přístup k datům?

Pokud základní funkce vaší aplikace vyžadují určitá data, například při spuštění aplikace, umístěte je do interního adresáře nebo databáze. Soubory specifické pro aplikaci, které jsou uloženy v externím úložišti, nejsou vždy přístupné, protože některá zařízení umožňují uživatelům odebrat fyzické zařízení, které odpovídá externímu úložišti.

Který z nich si vybrat?

- Kolik místa potřebují vaše data?
- Jak spolehlivý musí být přístup k datům?
- Jaký druh dat potřebujete ukládat?
- Měla by být data pro vaši aplikaci soukromá?

Jaká data potřebujete ukládat?

Pokud máte data, která mají význam pouze pro vaši aplikaci, použijte úložiště specifické pro aplikaci. Pro mediální obsah, který lze sdílet, použijte sdílené úložiště, aby k němu měly přístup i ostatní aplikace. Pro strukturovaná data použijte buď předvolby (pro data typu klíč-hodnota), nebo databázi (pro data, která obsahují více než 2 sloupce).

Který z nich si vybrat?

- Kolik místa potřebují vaše data?
- Jak spolehlivý musí být přístup k datům?
- Jaký druh dat potřebujete ukládat?
- Měla by být data pro vaši aplikaci soukromá?

Měla by být data pro vaši aplikaci soukromá?

Při ukládání citlivých dat - dat, která by neměla být přístupná z žádné jiné aplikace - používejte interní úložiště, předvolby nebo databázi. Interní úložiště má navíc tu výhodu, že data jsou před uživateli skrytá.

Přístup k souborům specifickým pro aplikaci

- Adresáře interního úložiště - Tyto adresáře zahrnují jak vyhrazené místo pro ukládání trvalých souborů, tak další místo pro ukládání dat mezipaměti. Systém zabraňuje ostatním aplikacím v přístupu k těmto umístěním a v systému Android 10 (úroveň API 29) a vyšších jsou tato umístění šifrována. Díky těmto vlastnostem jsou tato umístění vhodným místem pro ukládání citlivých dat, ke kterým má přístup pouze samotná aplikace.
- Adresáře externího úložiště - Tyto adresáře zahrnují jak vyhrazené umístění pro ukládání trvalých souborů, tak další umístění pro ukládání dat mezipaměti. Ačkoli k těmto adresářům může přistupovat i jiná aplikace, pokud má příslušná oprávnění, soubory uložené v těchto adresářích jsou určeny pouze pro použití vaší aplikací. Pokud máte výslovně v úmyslu vytvářet soubory, ke kterým by měly mít přístup i jiné aplikace, měla by vaše aplikace tyto soubory ukládat raději do sdílené části externího úložiště.

Příklad

- `https://www.journaldev.com/9383/android-internal-storage-example-tutorial`
- `https://developer.android.com/training/data-storage/app-specific`
- `https://github.com/android/storage-samples`
- Chcete-li dále chránit soubory specifické pro aplikaci, použijte knihovnu Security, která je součástí balíku Android Jetpack, a zašifrujte tyto soubory v klidovém stavu. Šifrovací klíč je specifický pro vaši aplikaci.

Přístup k mediálním souborům ze sdíleného úložiště

- Mnoho aplikací umožňuje uživatelům přispívat a přistupovat k médiím, která jsou k dispozici na externím úložišti, aby se obohatil uživatelský zážitek.
- Framework poskytuje optimalizovaný index do sbírek médií, nazývaný úložiště médií, který umožňuje snadnější vyhledávání a aktualizaci těchto mediálních souborů.
- I po odinstalování aplikace zůstávají tyto soubory v zařízení uživatele.

ContentResolver

Pro interakci s abstrakcí úložiště médií použijte objekt ContentResolver, který načtete z kontextu aplikace:

```
1  val projection = arrayOf(media-database-columns-to-retrieve)
2  val selection = sql-where-clause-with-placeholder-variables
3  val selectionArgs = values-of-placeholder-variables
4  val sortOrder = sql-order-by-clause
5
6  applicationContext.contentResolver.query(
7      MediaStore.media-type.Media.EXTERNAL_CONTENT_URI,
8      projection,
9      selection,
10     selectionArgs,
11     sortOrder
12 )?.use{ cursor ->
13     while (cursor.moveToNext()){
14         // žPouijte sloupec ID z projekce k získání
15         // URI reprezentující samotnou mediální žpoloku.
16     }
17 }
```

Mediální soubory

Systém automaticky prohledá externí úložný svazek a přidá mediální soubory do následujících přesně definovaných kolekcí:

- **Images**, včetně fotografií a snímků obrazovky, které jsou uloženy v adresářích *DCIM/* a *Pictures/*. Systém tyto soubory přidá do tabulky *MediaStore.Images*.
- **Videos**, které jsou uloženy v adresářích *DCIM/*, *Movies/* a *Pictures/*. Systém přidá tyto soubory do tabulky *MediaStore.Video*.
- **Audio files**, které jsou uloženy v adresářích *Alarms/*, *Audiobooks/*, *Music/*, *Notifications/*, *Podcasts/*, *and Ringtones/*, a také zvukové seznamy skladeb, které jsou v adresářích *Music/ or Movies/*. Systém tyto soubory přidá do tabulky *MediaStore.Audio*.
- **Downloaded files**, které jsou uloženy v adresáři *Download/*. V zařízeních se systémem Android 10 (úroveň API 29) a vyšším jsou tyto soubory uloženy v tabulce *MediaStore.Downloads*. V systémech Android 9 (úroveň API

Předvolby aplikace

- Pokud máte relativně malou kolekci klíčových hodnot, které chcete uložit, měli byste použít funkci SharedPreferences.
- Objekt SharedPreferences ukazuje na soubor obsahující dvojice klíč-hodnota a poskytuje jednoduché metody pro jejich čtení a zápis.
- Každý soubor SharedPreferences je spravován frameworkem a může být soukromý nebo sdílený.

SharedPreferences

- This class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types.
- You can use SharedPreferences to save any primitive data: booleans, floats, ints, longs, and strings.
- Tato data zůstanou zachoHow using Jak používat vána napříč uživatelskými relacemi (i když vaše aplikace ukončena).
- *"SharedPreferences"* jsou uloženy jako soubory XML ve složce *shared_prefs*

SharePreferences ?

- 1. Získání předvoleb ze zadaného souboru

```
1 val sharedPref = activity?.getSharedPreferences(
2         getString(R.string.preference_file_key), Context.MODE_PRIVATE)
```

- 2. Přečtěte si

```
1 val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
2 val defaultValue = resources.getInteger(R.integer.saved_high_score_default_key)
3 val highScore = sharedPref.getInt(getString(R.string.saved_high_score_key),
      defaultValue)
```

- 3. Zapsat a použít (nebo odevzdat) změny

```
1 val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
2 with (sharedPref.edit()){
3     putInt(getString(R.string.saved_high_score_key), newHighScore)
4     apply() // commit() - synchronously
5 }
```

Nastavení

- Aplikace často obsahují nastavení, která umožňují uživatelům upravovat aplikace. funkce a chování.
- Nastavení je místo v aplikaci, kde uživatelé uvádějí své preference. jak se má aplikace chovat.
- Nastavení je v uživatelském rozhraní věnována malá pozornost, protože se v něm často nevyskytuje. potřeba.

How use settings

- 1. Vytvořte prostředek XML systému Android s názvem např. *preferences.xml* PreferenceScreen type.
- 2. Přidat do souboru PreferencesCategory, a jeden z nich *CheckBoxPreference, ListPreference, EditTextPreference*
- 3. Vytvořte třídu *MyPreferencesActivity*, která rozšiřuje *PreferenceActivity* Tato činnost načte *preference.xml* a umožní uživateli změnit hodnoty.
- 4. Přidejte do metody *onOptionsItemSelected()* kód pro spuštění PreferencesActivity

RoomDatabase

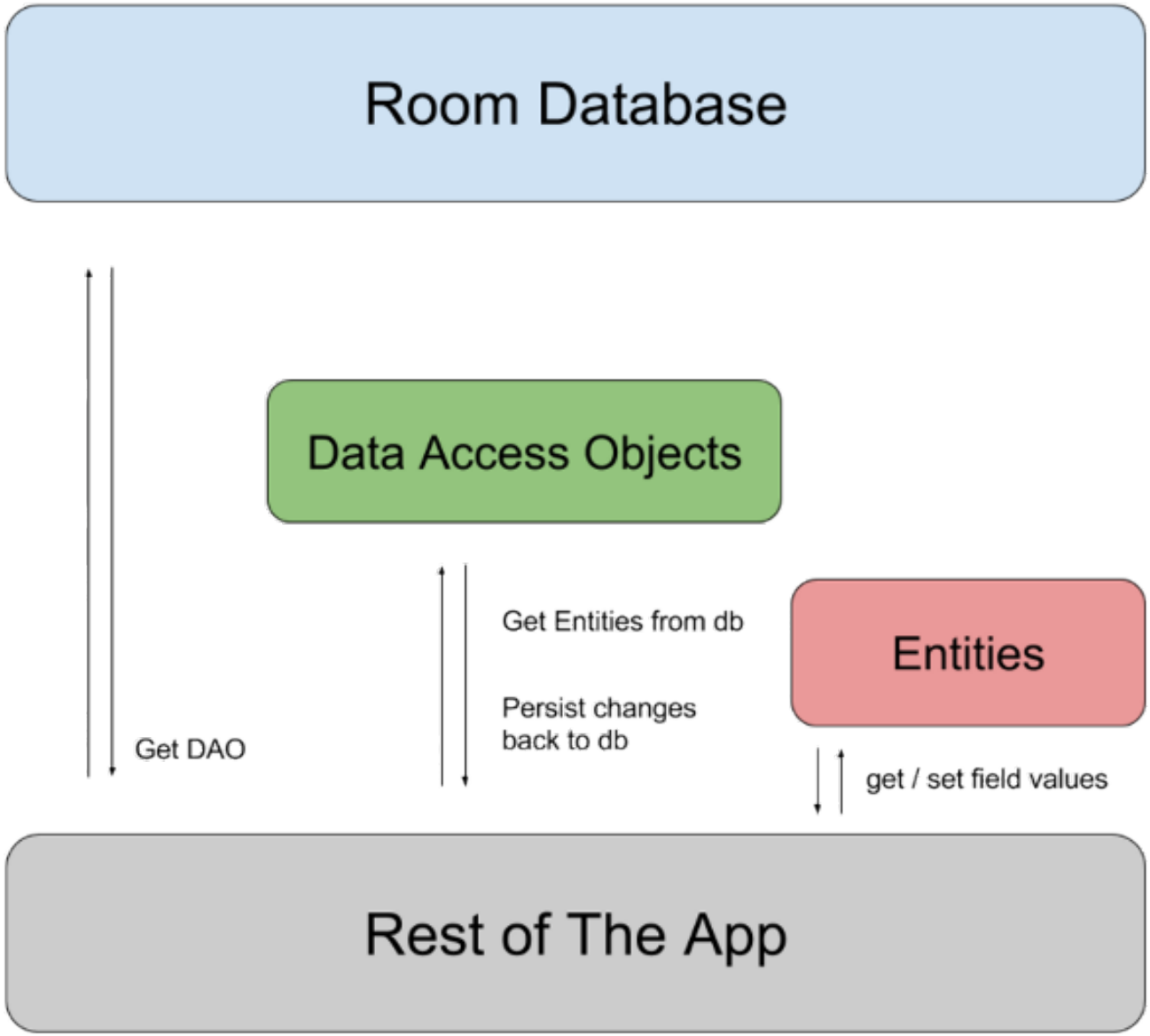
- Room poskytuje abstrakční vrstvu nad SQLite, která umožňuje plynulý přístup k databázi a zároveň využívá plný výkon SQLite.
- Aplikace, které zpracovávají netriviální množství strukturovaných dat, mohou mít z lokálního uchovávání těchto dat velký prospěch.
- Room takes care of caching data when device is offile
- Tato budoucnost způsobuje, že Room se doporučuje používat místo SQLite

RoomDatabase

Hlavní komponenty v Room

- Database - Obsahuje držitele databáze a slouží jako hlavní přístupový bod pro základní připojení k perzistovaným relačním datům vaší aplikace.
- Entity - Reprezentuje tabulku v rámci databáze
- DAO - Obsahuje metody používané pro přístup k databázi.

Room architecture diagram



Příklad kódu: Entity

```
1 @Entity
2 data class User(
3     @PrimaryKey val uid: Int,
4     @ColumnInfo(name = "first_name") val firstName: String?,
5     @ColumnInfo(name = "last_name") val lastName: String?
6 )
```

Příklad kódu: DAO

```
1  @Dao
2  interface UserDao{
3      @Query("SELECT * FROM user")
4      fun getAll(): List<User>
5
6      @Query("SELECT * FROM user WHERE uid IN (:userIds)")
7      fun loadAllByIds(userIds: IntArray): List<User>
8
9      @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
10            "last_name LIKE :last LIMIT 1")
11     fun findByName(first: String, last: String): User
12
13     @Insert
14     fun insertAll(vararg users: User)
15
16     @Delete
17     fun delete(user: User)
18 }
```

Příklad kódu: Database

```
1 @Database(entities = arrayOf(User::class), version = 1)
2 abstract class AppDatabase : RoomDatabase(){
3     abstract fun userDao(): UserDao
4 }
```

Živý příklad

Room Database Příklad