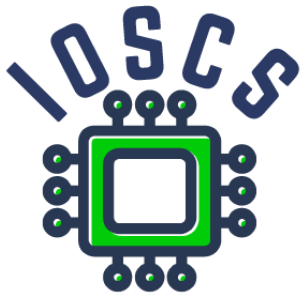


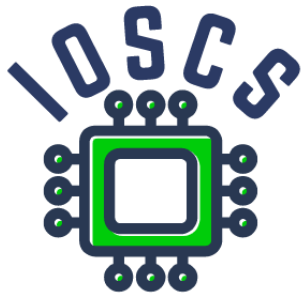
Project: Innovative Open Source Courses for Computer Science

Vývoj mobilných aplikácií
Prezentácie

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

What is mobile device

Hlavné rozdiely ?

- CPU

What is mobile device

Hlavné rozdiely ?

- CPU
- Batéria

What is mobile device

Hlavné rozdiely ?

- CPU
- Batéria
- Senzory

What is mobile device

Hlavné rozdiely ?

- CPU
- Batéria
- Senzory
- Pripojiteľnosť

Čo očakávate od mobilných zariadení

- ??

Čo očakávate od mobilných zariadení

- ??
- Koľko máte zariadení

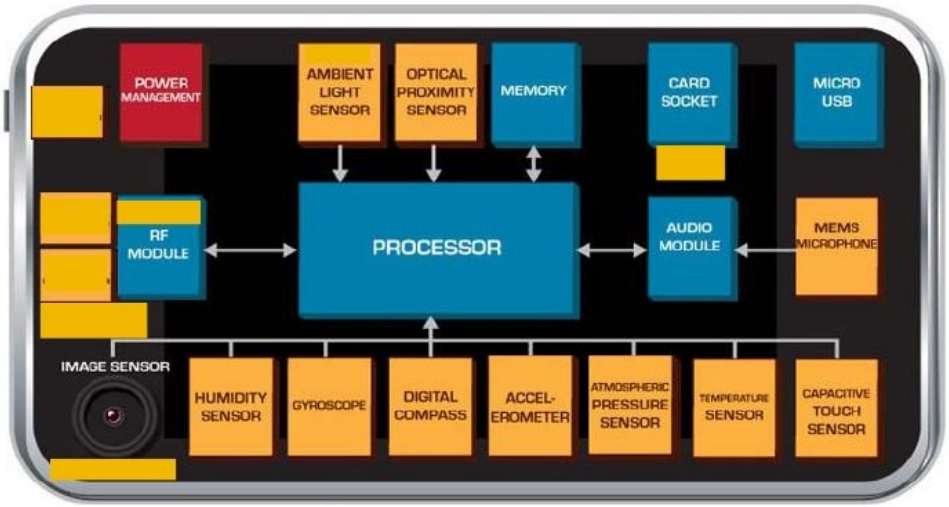
Čo očakávate od mobilných zariadení

- ??
- Koľko máte zariadení
- V budúcnosti - jedno zariadenie veľa aplikácií

Čo očakávate od mobilných zariadení

- ??
- Koľko máte zariadení
- V budúcnosti - jedno zariadenie veľa aplikácií
- ~~In future~~ nie TERAZ

Zariadenie pre vývojárov



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

Hlavní predajcovia

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

Mobilný operačný systém

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

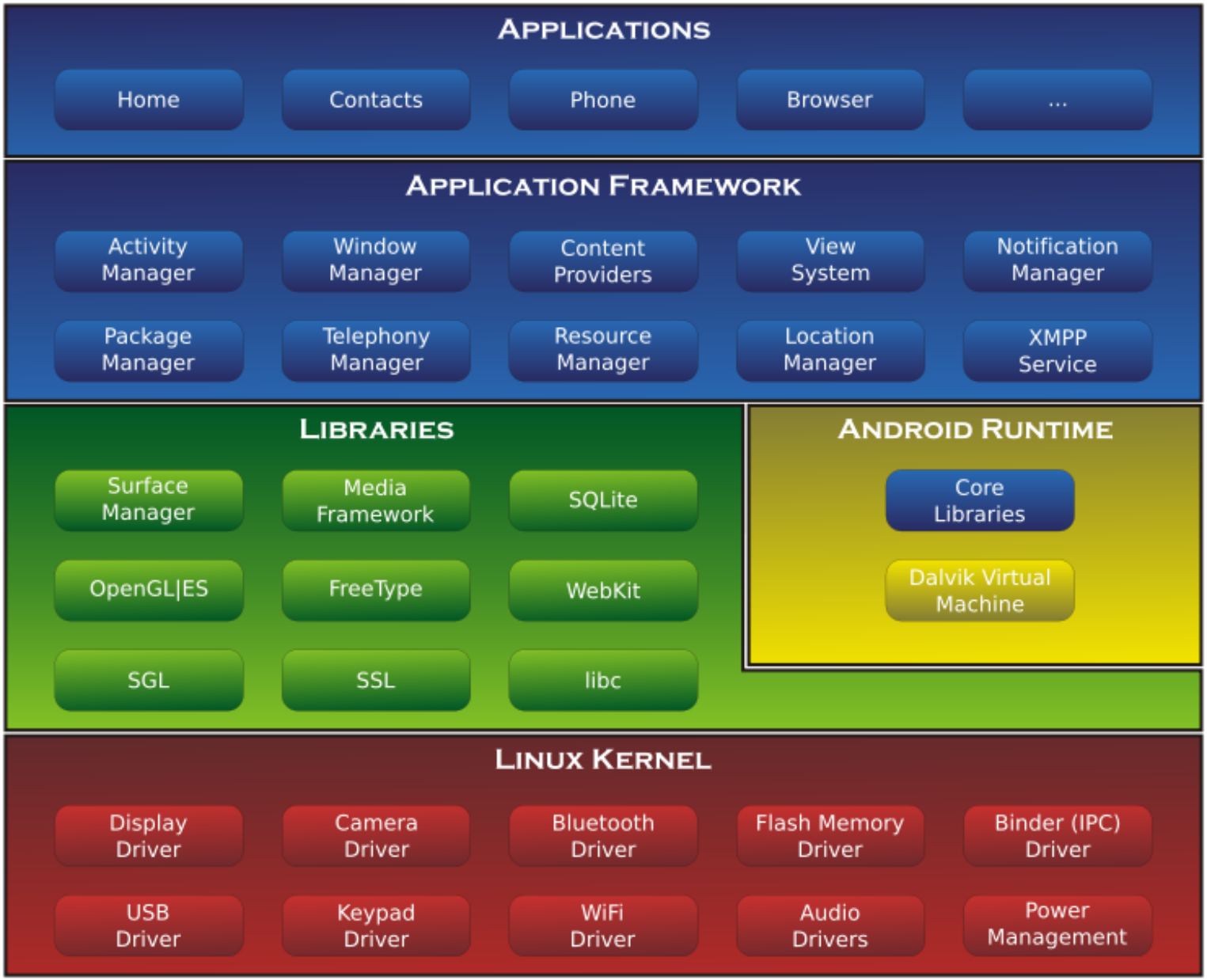
Čo je Android?

- Otvorený softvérový zásobník, ktorý zahŕňa
  - Operačný systém
  - Middleware
  - Kľúčové mobilné aplikácie (webový prehliadač, PIM, SMS, e-mail ...)
  - Knižnice API na písanie mobilných aplikácií
- Open-source vývojová platforma na tvorbu mobilných aplikácií
- Operačný systém založený na Linuxe
- Všeobecne pre mobilné zariadenia s dotykovou obrazovkou, ako sú smartfóny a tablety

Kde je Android

- Telefóny
- Tablety
- Televízory
- STB - set-top-box
- roboty
- budú v autách
- budú v zábavných systémoch v lietadlách
- Android je všade !

Softvérový zásobník pre Android



Application Fundamentals

- Aplikácie pre Android sú napísané v programovacom jazyku Java alebo Kotlin. - Všeobecne
- Nástroje Android SDK kompilujú kód (spolu so všetkými údajmi a zdrojmi) do balíka Android, archívneho súboru s príponou .apk
- Všetok kód v jednom súbore .apk sa považuje za jednu aplikáciu a je to súbor, ktorý zariadenia s operačným systémom Android používajú na inštaláciu aplikácie.

Application Fundamentals

- Aplikácie pre Android sú napísané v programovacom jazyku Java alebo Kotlin. - Všeobecne
- Nástroje Android SDK kompilujú kód (spolu so všetkými údajmi a zdrojmi) do balíka Android, archívneho súboru s príponou .apk
- Všetok kód v jednom súbore .apk sa považuje za jednu aplikáciu a je to súbor, ktorý zariadenia s operačným systémom Android používajú na inštaláciu aplikácie.

Application Fundamentals

- Aplikácie pre Android sú napísané v programovacom jazyku Java alebo Kotlin. - Všeobecne
- Nástroje Android SDK kompilujú kód (spolu so všetkými údajmi a zdrojmi) do balíka Android, archívneho súboru s príponou .apk
- Všetok kód v jednom súbore .apk sa považuje za jednu aplikáciu a je to súbor, ktorý zariadenia s operačným systémom Android používajú na inštaláciu aplikácie.

Aplikácia pre Android

- Aplikácia Android žije vo vlastnom bezpečnostnom sandboxe,
- Andorid OS je viacpoužívateľský systém Linux,
- Každá aplikácia je iný používateľ, (v predvolenom nastavení)
- Každý proces má svoj vlastný virtuálny stroj (VM)
- V predvolenom nastavení beží každá aplikácia vo vlastnom linuxovom procese.
- Systém Android spustí proces, keď je potrebné, aby sa niektorá z komponentov aplikácie vykonať
- Vypne proces, keď už nie je potrebný alebo keď systém musí obnoviť pamäť pre iné aplikácie.

“Zásada najmenších privilégií”

- Každá aplikácia má v predvolenom nastavení prístup len ku komponentom, ktoré ktoré potrebuje na svoju prácu, a nič viac.
- Aplikácia nemôže pristupovať k častiam systému, pre ktoré nie je udelené oprávnenie
- Otázka: Ako zdieľať údaje s inými aplikáciami ?

Zdieľanie údajov s inými aplikáciami

- Zdieľajú rovnaké ID používateľa Linuxu, v takom prípade majú vzájomný prístup k svojim súborom
- Vytvoriť alebo používať poskytovateľa obsahu
- Ukladať údaje na kartu SDCard
- Dôležité: Aplikácia môže požiadať o povolenie prístupu k údajom zariadenia, ako napr. k používateľským kontaktom, správam SMS, pripojiteľnému úložisku (karta SD), fotoaparátu, Bluetooth a ďalšie.

Komponenty aplikácie

Štyri rôzne typy aplikačných komponentov. Každý typ slúži má odlišný účel a odlišný životný cyklus.

- Činnosti
- Služby
- Poskytovatelia obsahu
- Príjemcovia vysielania

Activity

- Predstavuje jednu obrazovku s používateľským rozhraním
- Viacero činností v jednej aplikácii
- Všetky aktivity majú vlastný životný cyklus
- Aktivita je implementovaná ako podtrieda aktivity
- Napr. aplikácia Mail: čítanie pošty, zostavovanie pošty...

Service

- Služba beží na pozadí a vykonáva dlhodobé operácie alebo prácu pre vzdialené procesy
- Služba neposkytuje používateľské rozhranie
- má vlastný životný cyklus
- napr. hudobná aplikácia: služba môže prehrávať hudbu na pozadí, kým je používateľ v inej aplikácii
-

Content providers

- Poskytovateľ obsahu spravuje zdieľaný súbor aplikačných údajov
- Ostatné aplikácie môžu upravovať údaje bez znalosti podrobnej architektúry
- Dobrou praxou je tiež používať poskytovateľa obsahu ako interný
- Poskytovatelia systémového obsahu uchovávajú informačný riadok Kontakt, Fotografie, Video...

Broadcast receivers

- Mechanizmus na odosielanie alebo prijímanie vysielaných správ zo systému Android a iných aplikácií Android
- Podobne ako návrhový vzor "publish-subscribe"
- Pri odosielaní broadcastovej správy sa ostatné aplikácie musia prihlásiť na odber tohto typu správy
- Existuje mnoho systémových správ, napr. systém pošle správu po dokončení štartu systému, batéria je vybitá...

Intent

- Ide o samostatný mechanizmus, ktorý používame na prevádzku troch zo štyroch typov komponentov - aktivít, služieb a prijímačov vysielania
- So zámerom me posielať informáci e o akcii a údaje
- V závislosti od komponentu definujeme akcie rôzne

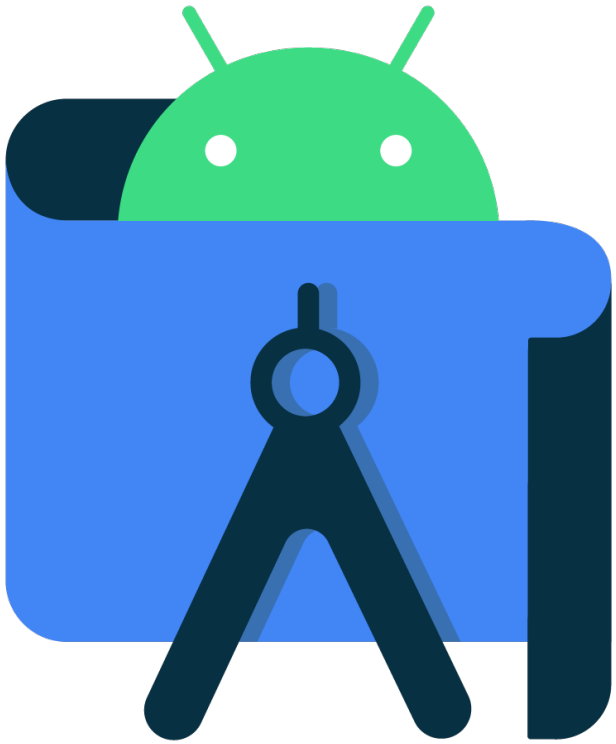
Súbor Manifest

- Všetky informácie o komponente musia existovať v súbore *AndroidManifest.xml*
- Osobitne musíme zverejniť informácie o hlavnej činnosti
- Musíme tiež zverejniť informácie o povolení
- Aplikácia vyžaduje
- Deklarovať minimálnu úroveň API
- Deklarovať použité alebo požadované hardvérové a softvérové funkcie (kamera, služby bluetooth alebo viacdotykový displej atď.)
- Knižnice API (iné ako API rámca Android), napríklad knižnica 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>
```

Príklad naživo - Hello World



Úvod do Android Studio

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

Riešenia bežných problémov pri vývoji systému Android

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

MOBILE APPLICATION DEVELOPMENT

Životný cyklus aktivity

Innovative Open Source courses for Computer Science

30.05.2021

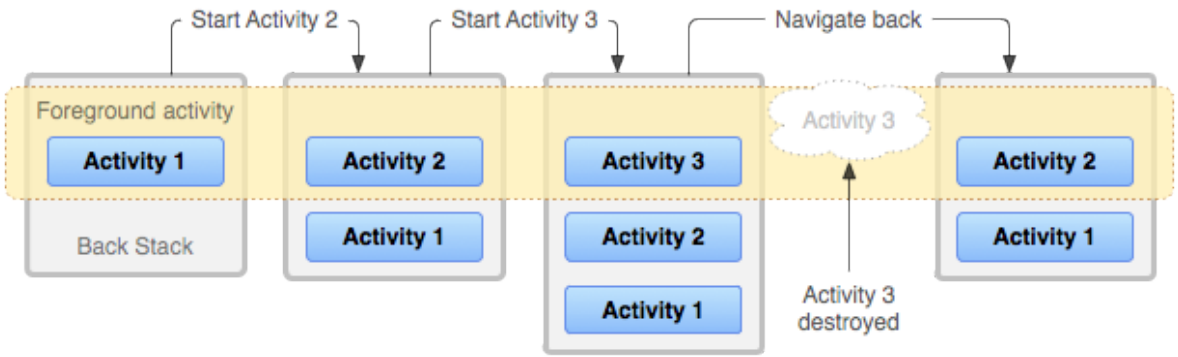
Funded by
the European Union

Activity

- Aktivita je komponent, ktorý poskytuje obrazovku s s ktorou môžu používatelia komunikovať, aby niečo vykonali.
- Každá aktivita 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.

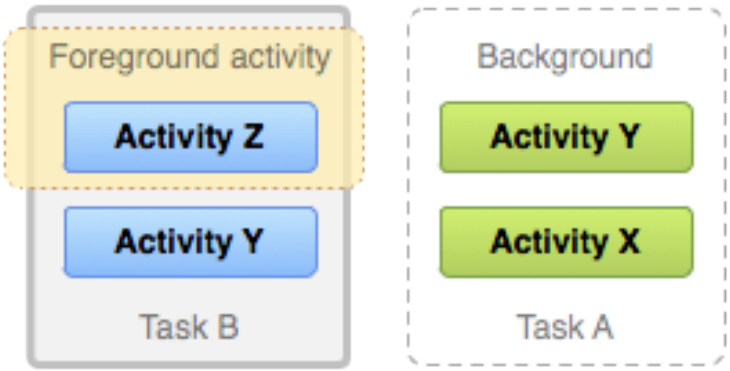
Násobiť aktivity - ako zariadiť

- 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



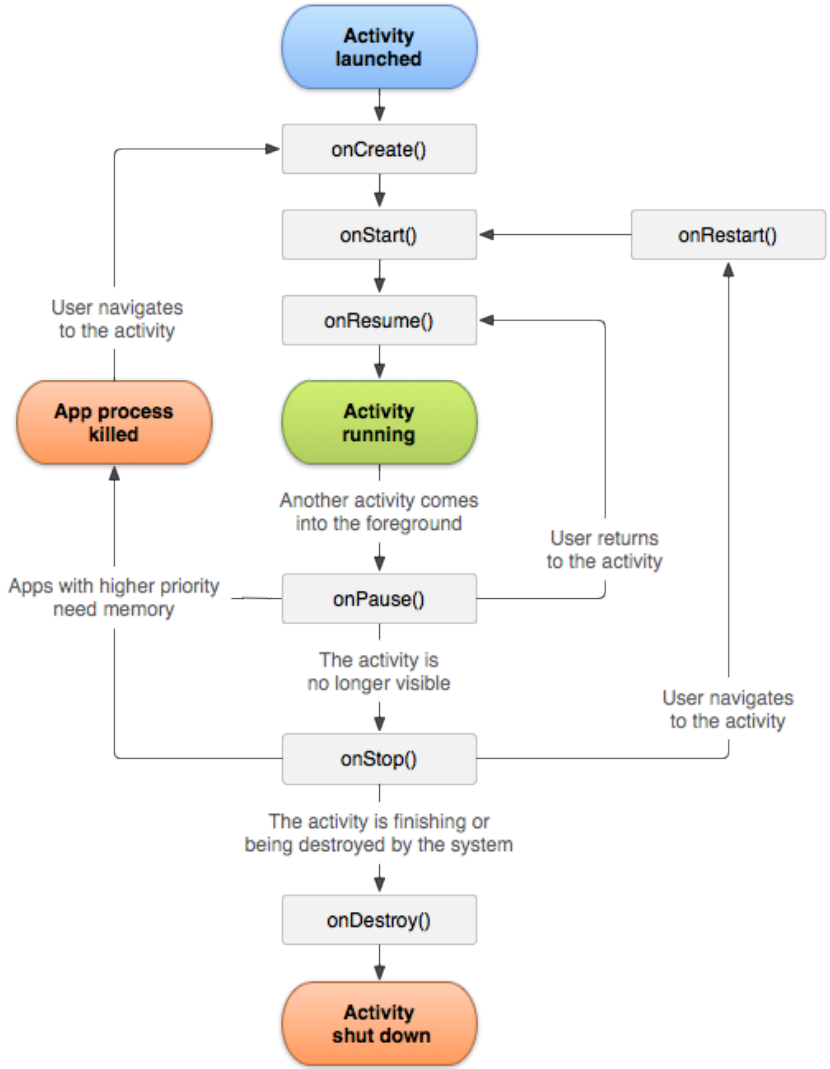
Vytvorenie aktivity

- 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

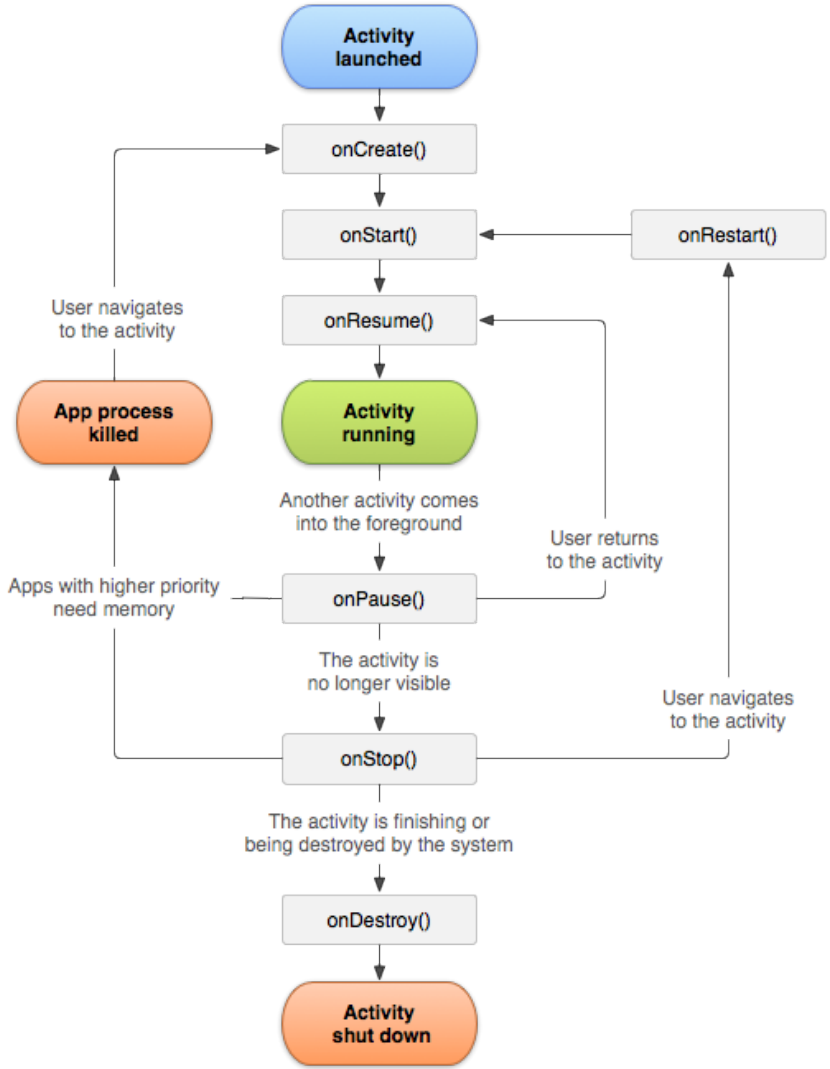
Implementácia spätných volaní životného cyklu - kostra

```
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          // Ďšalia čťinnos sa ďsústreuje (táto čťinnos bude "pozastavená").
10     }
11     override fun onRestart(){
12         super.onRestart()
13     }
14     override fun onResume(){
15         super.onResume()
16         // Aktivita sa stala ľviditenou (teraz je "obnovená").
17     }
18     override fun onStart(){
19         super.onStart()
20         // Aktivita sa čoskoro ľzviditení.
21     }
22     override fun onStop(){
23         super.onStop()
24         // Aktivita žu nie je ľviditená (je teraz "zastavená")
25     }
26     override fun onDestroy(){
27         super.onDestroy()
28         // Aktivita sa má čťznii.
29     }
30 }
```

Životný cyklus aktivity



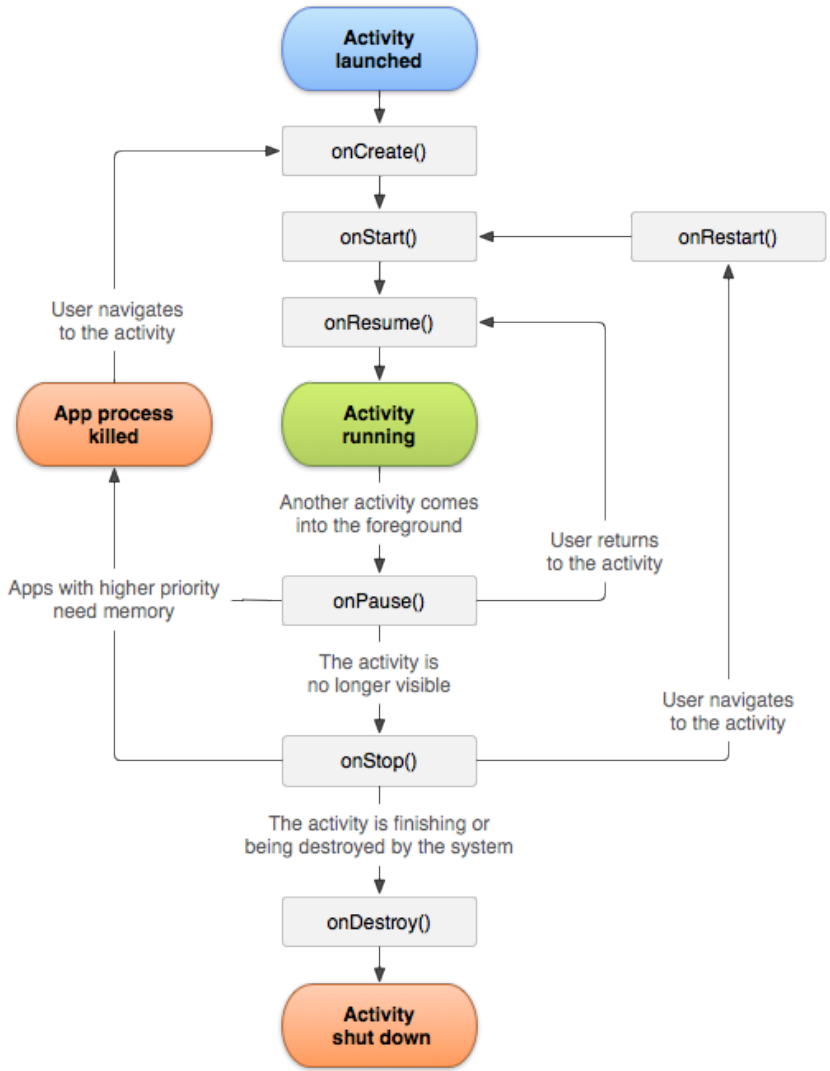
Životný cyklus aktivity



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



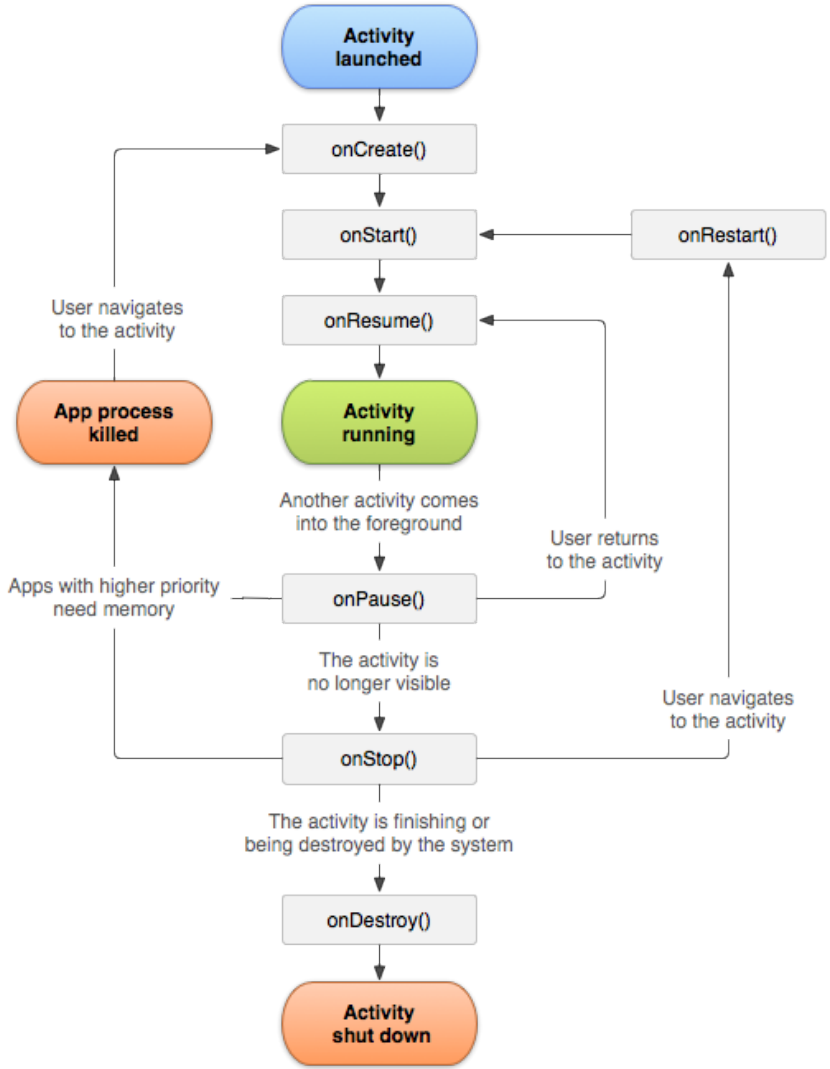
onRestart()

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.

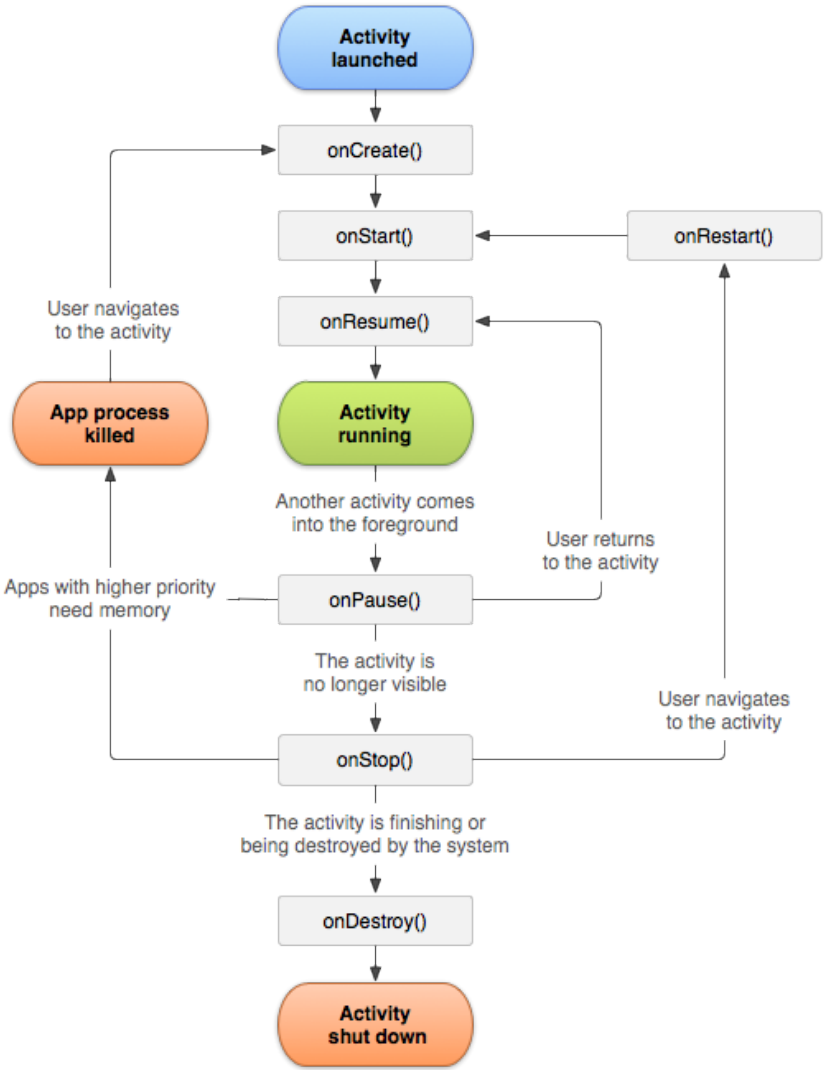
Životný cyklus aktivity



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()*.

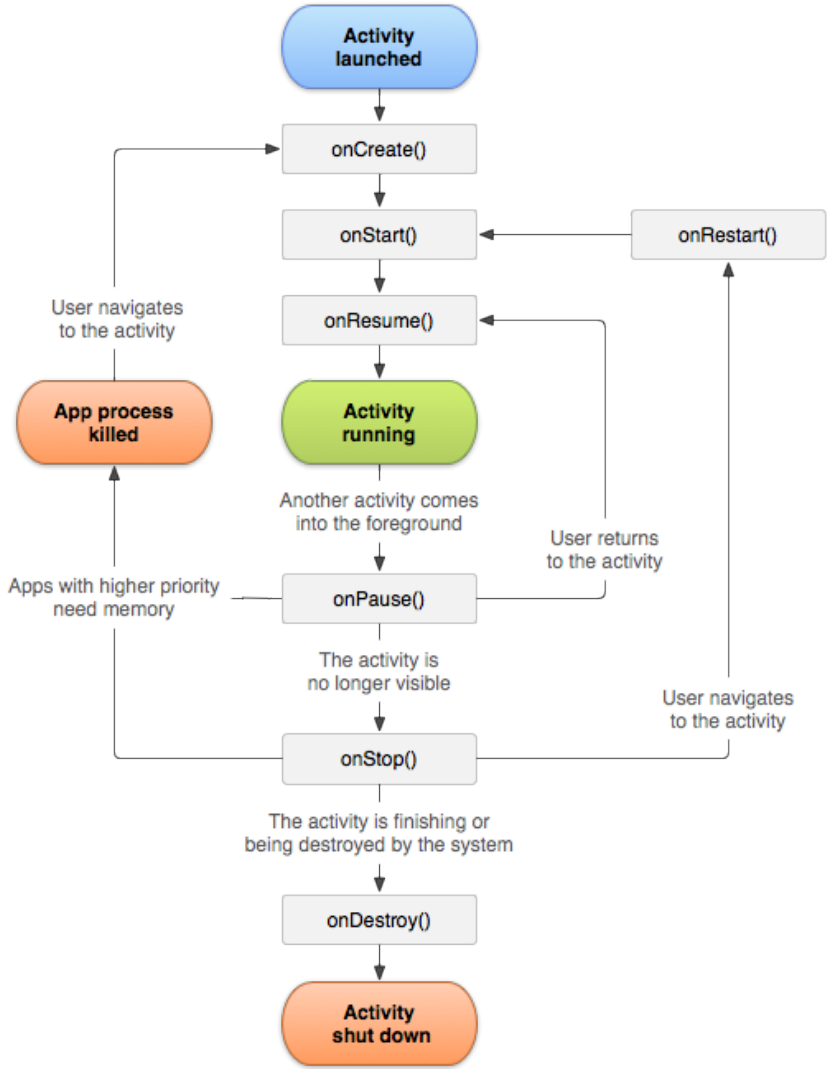
Životný cyklus aktivity



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.

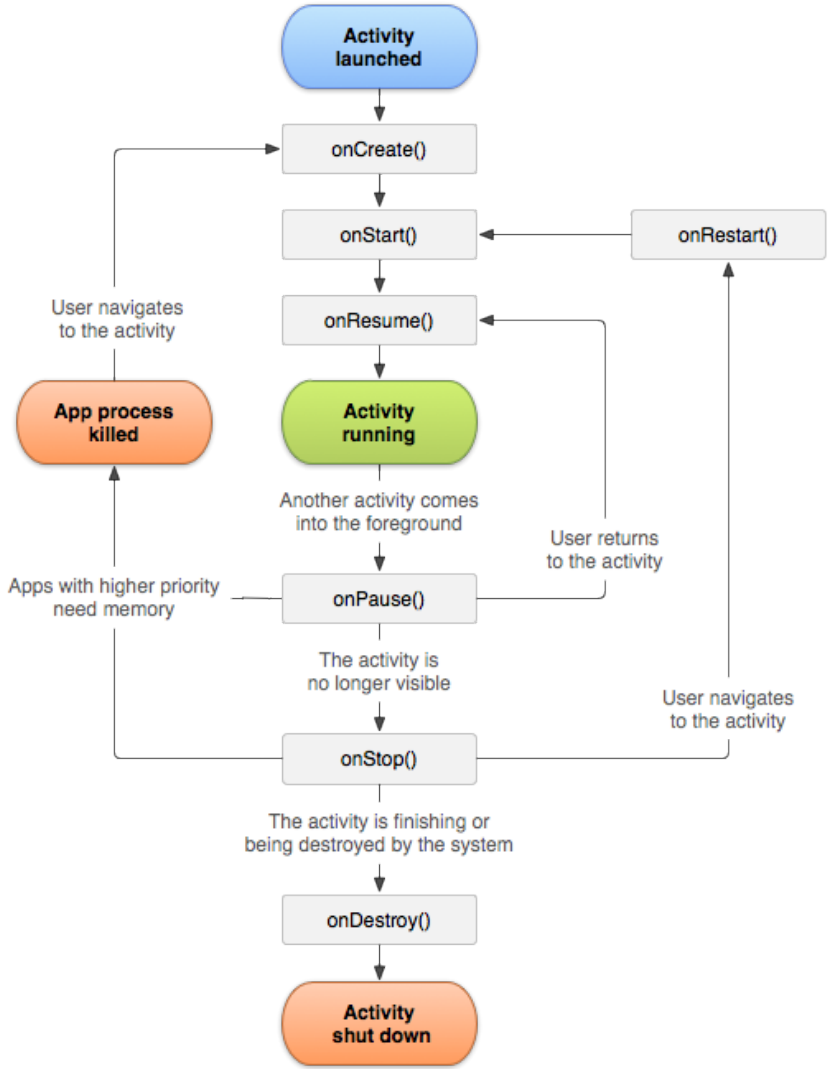
Životný cyklus aktivity



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 informácií

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

Uloženie stavu aktivity

- V predvolenom nastavení systém používa Bundle stav inštancie 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 dostávajú rovnaké Bundle ktorý obsahuje informácie o stave inštancie.

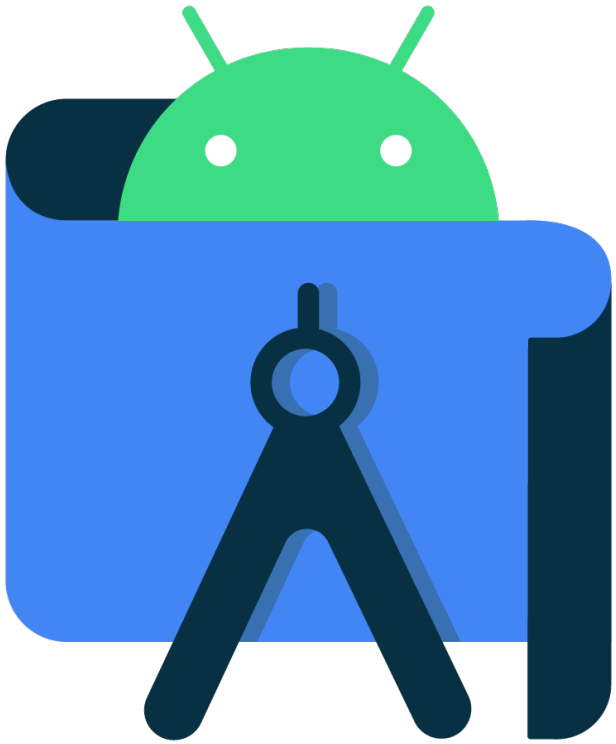
```
override fun onCreate(savedInstanceState: Bundle?){
    super.onCreate(savedInstanceState) // žVdy najprv volajte nadtriedu

    // Kontrola, či znovu vytvárame predtým čznienú šintanciu
    if (savedInstanceState != null){
        with(savedInstanceState){
            // Restore value of members from saved state
            currentScore = getInt(STATE_SCORE)
            currentLevel = getInt(STATE_LEVEL)
        }
    } else{
        // Pravdepodobne inicializácia členov s predvolenými hodnotami pre novú
            šintanciu
    }
}
```

```
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 Intent objekt, ktorý obsahuje informácie o Activity

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 aktivity 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 - Oheň druhá aktivita

```
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     }
```

Spustenie aktivity 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

Budovanie Intent

- Vytvorenie objektu zámeru, ktorý zadáme
- **Component name** - len výslovný zámer
- **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 s údajmi

- 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 akcia VIEW robí to, čo sa pre daný URI považuje za najrozumnejšie.
- ACTION_DIAL `tel:123` – Zobrazenie telefónneho číselníka s daným s vyplneným číslom.

Kategória

- Poskytuje dodatočné informácie o akcii, ktorá sa má vykonať.
- Napríklad KATEGÓRIA _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 kategória 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

Explicit Intent

```
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)
```

Implicit Intent

```
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 }
```

Vynútenie 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 }
```

Prijať implicitný intnet

- Aby sme mohli prijímať implicitné zámery, musíme deklarovať jeden alebo viac filtrov zámerov 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 - životný cyklus

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Fragment

- Predstavuje opakovane použiteľnú časť používateľského rozhrania aplikácie
- Fragment definuje a spravuje svoje vlastné rozloženie
- Má vlastný životný cyklus
- Môže spracovávať vlastné vstupné udalosti
- Fragment musí byť hostiteľom aktivity alebo iného fragmentu

Vytvorenie fragmentu

- Nastavenie prostredia
- Vytvorenie triedy fragmentov
- Pridanie fragmentu do aktivity

Pridať do projektu **build.gradle** informácie o úložisku Google Maven

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

Vytvorenie fragmentu

- Nastavenie prostredia
- Vytvorenie triedy fragmentov
- Pridanie fragmentu do aktivity

Pridať do aplikácie **build.gradle** informácie o knižnici AndroidX Fragment

```
dependencies{
    val fragment_version = "1.3.4"

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

Vytvorenie fragmentu

- Nastavenie prostredia
- Vytvorenie triedy fragmentov
- Pridanie fragmentu do aktivity

Môžeme použiť **Fragemnt, DialogFragment, PreferenceFragmentCompat**

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

Vytvorenie fragmentu

- Nastavenie prostredia
- Vytvorenie triedy fragmentov
- Pridanie fragmentu do aktivity

Define in XML, *android:name* containing a single class

```
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" />
```

Vytvorenie fragmentu

- Nastavenie prostredia
- Vytvorenie triedy fragmentov
- Pridanie fragmentu do aktivity

or (more offten) Define in XML container for fragment

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

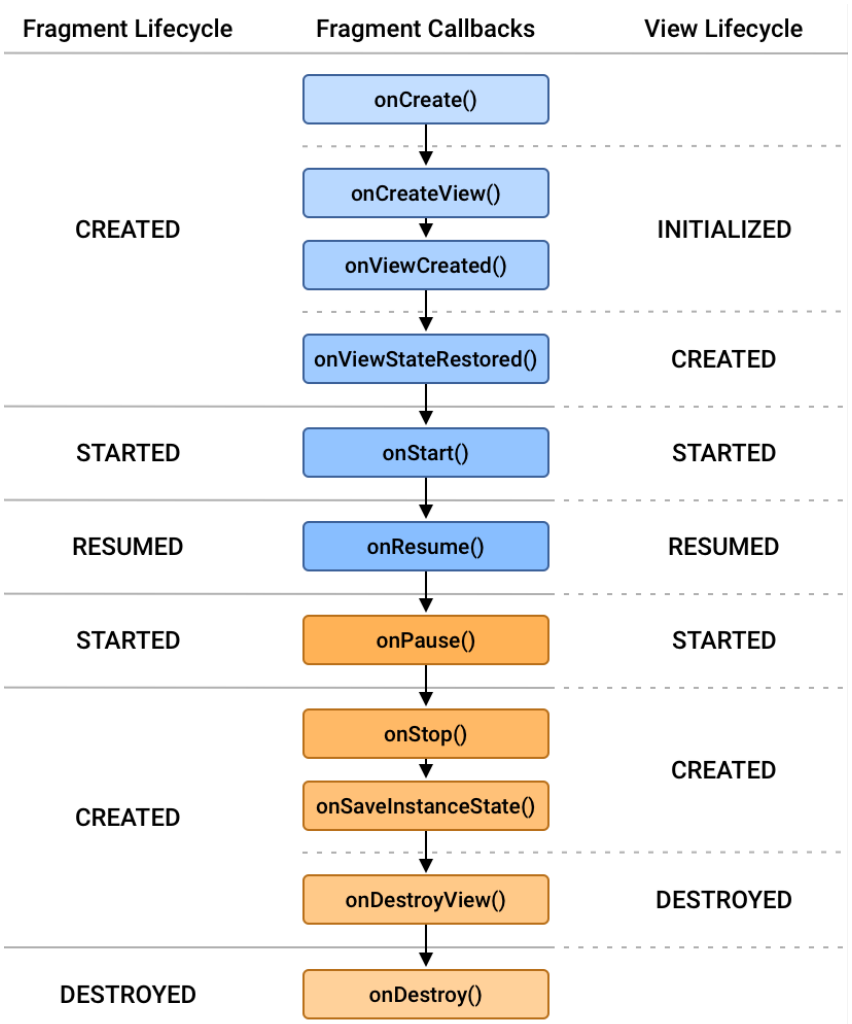
Add code to activity (onCreate())

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

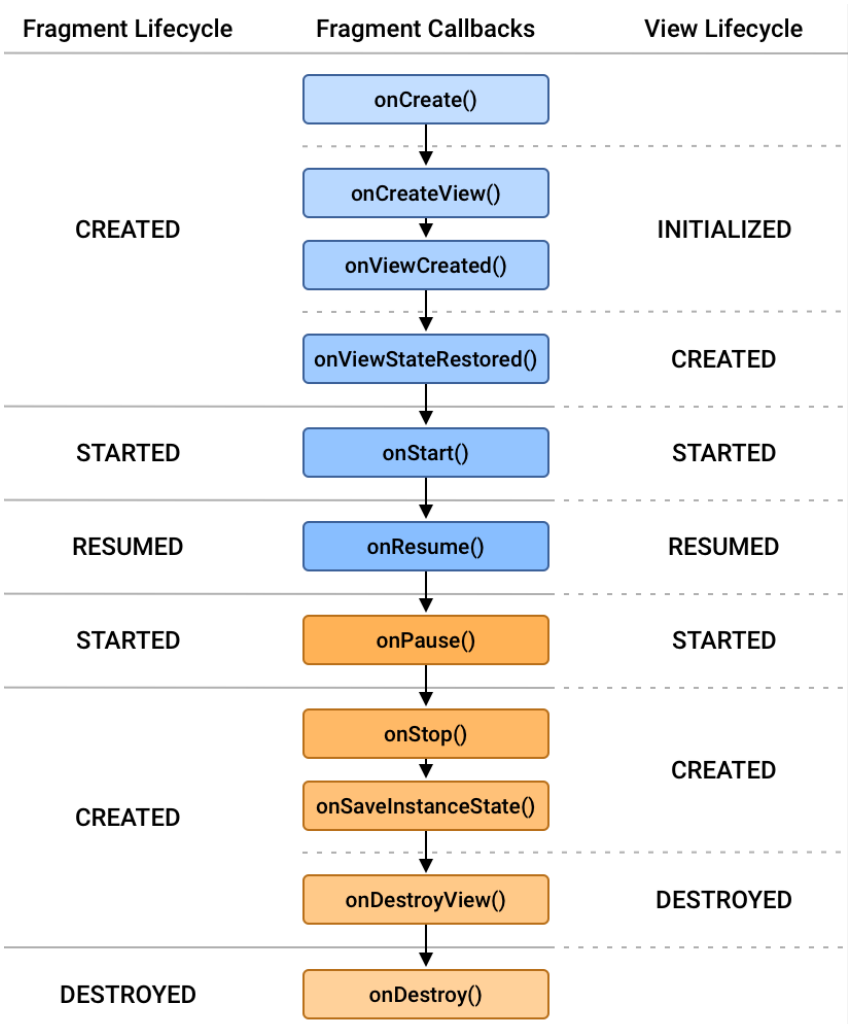
Fragment lifecycle

- Každá inštancia **Fragment** má svoj vlastný životný cyklus
- Životný cyklus pohľadu sa líši od životného cyklu fragmentu
- Stav fragmentu:
  - INITIALIZED
  - CREATED
  - STARTED
  - RESUMED
  - DESTROYED

Životný cyklus fragmentu



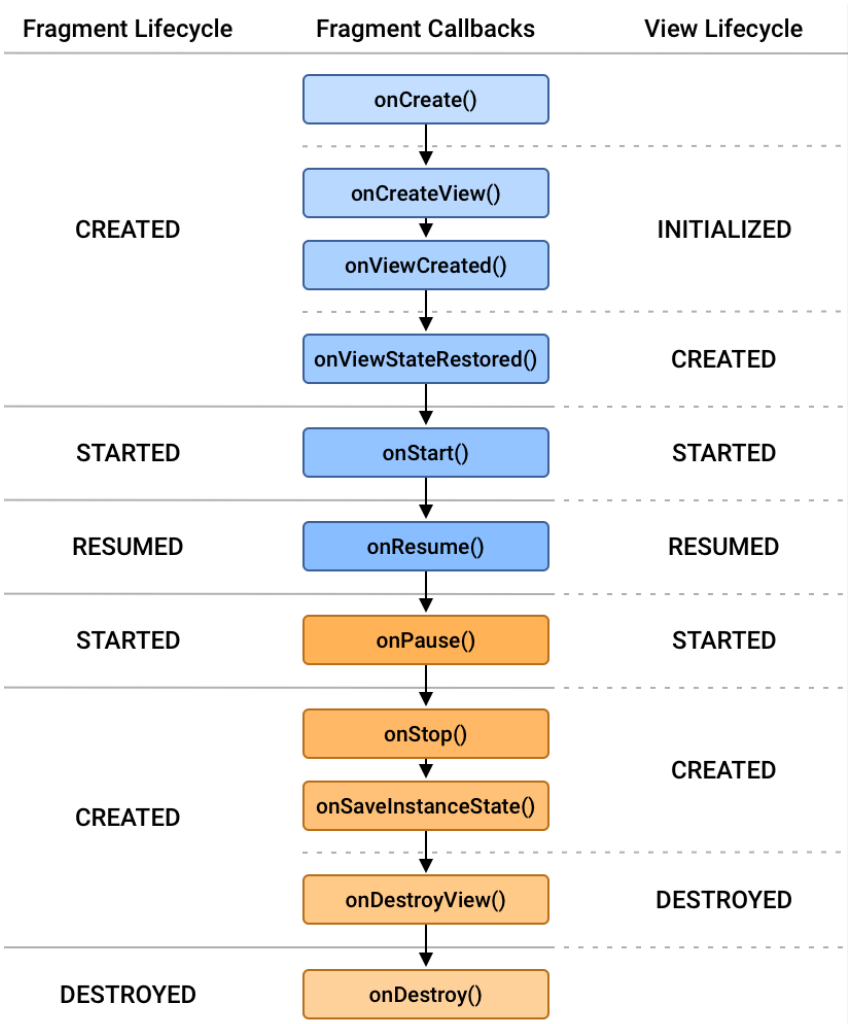
Životný cyklus fragmentu



CREATED

Bol pridaný do FragmentManager a metóda onAttach() už bola zavolaná.
Životný cyklus pohľadu fragmentu sa vytvorí len vtedy, keď váš Fragment poskytne platnú inštanciu View. Môžete tiež prekryť funkciu onCreateView(), aby ste programovo nafúkli alebo vytvorili pohľad vášho fragmentu.

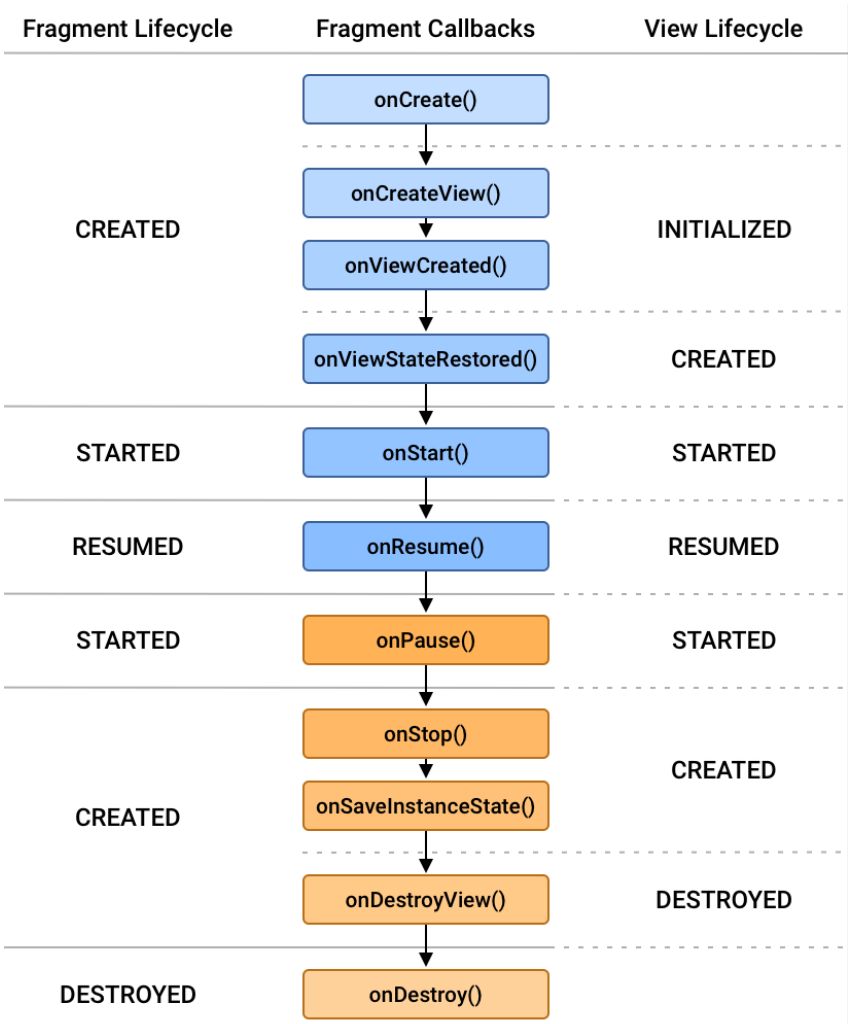
Životný cyklus fragmentu



STARTED

Tento stav zaručuje, že je k dispozícii zobrazenie fragmentu, ak bolo vytvorené, a že je bezpečné vykonať FragmentTransaction na podriadenom FragmentManager fragmentu.

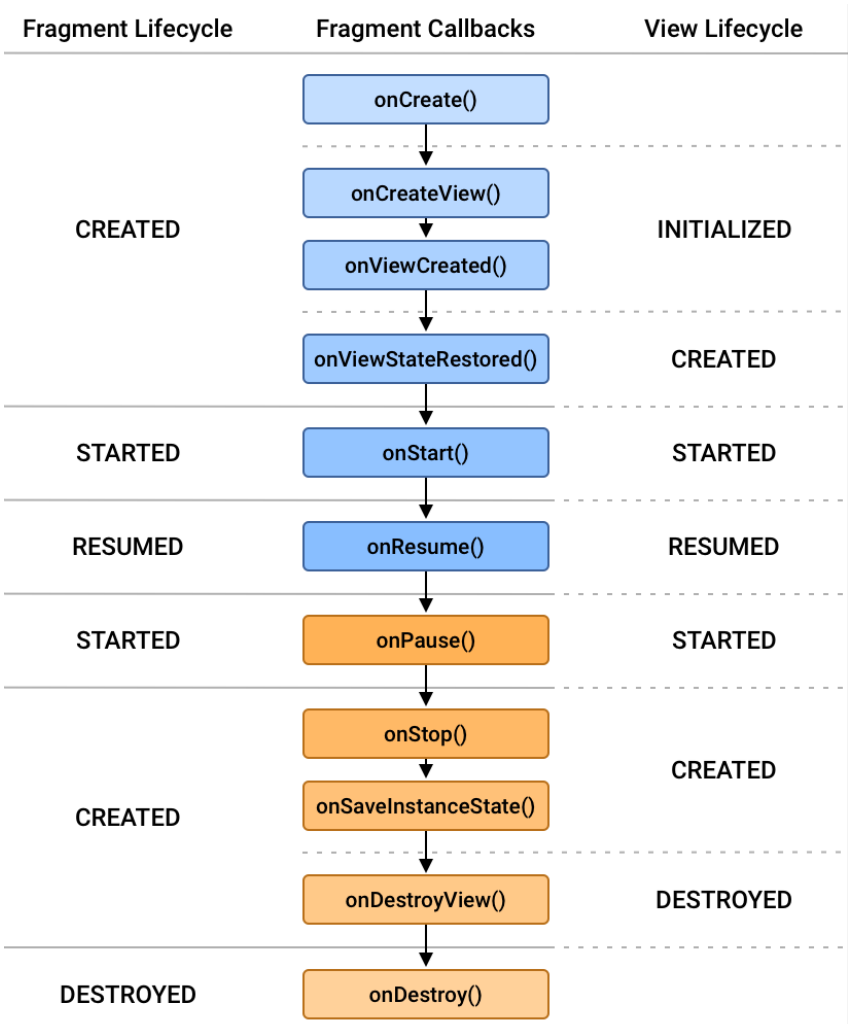
Životný cyklus fragmentu



RESUMED

Keď je fragment viditeľný, všetky efekty animátora a prechodu sú dokončené a fragment je pripravený na interakciu používateľa.

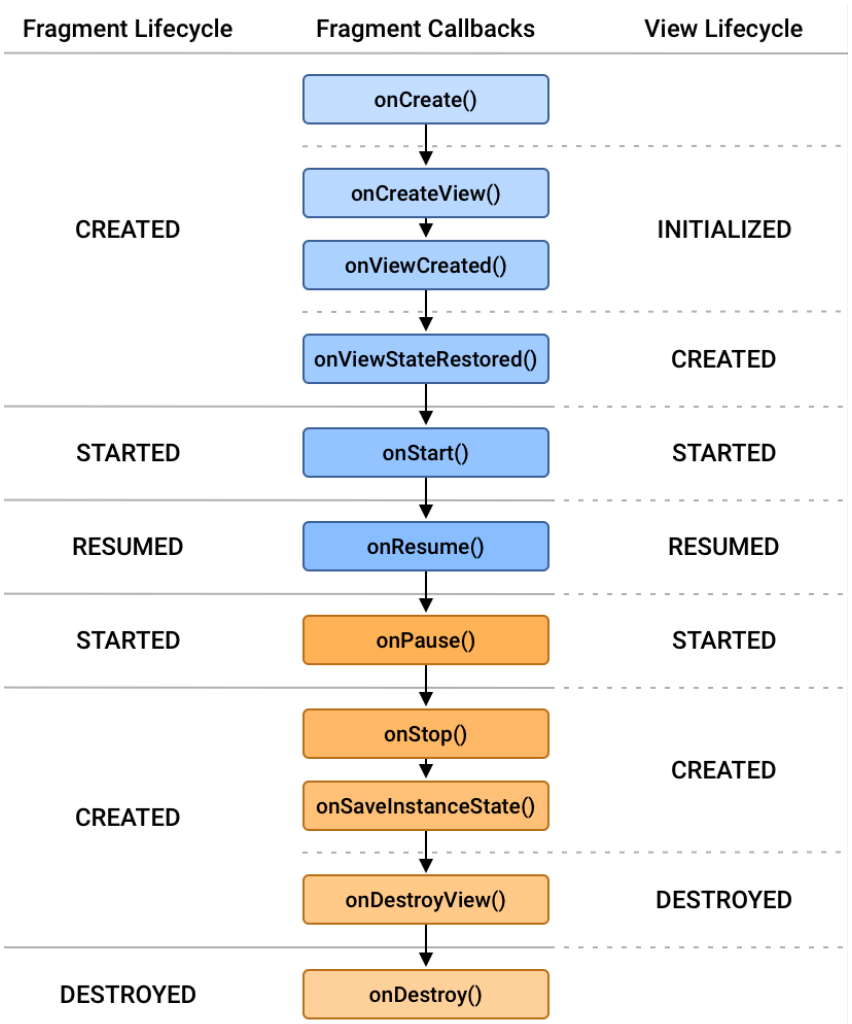
Životný cyklus fragmentu



STARTED

Keď používateľ začne opúšťať fragment a fragment je stále viditeľný, životné cykly pre fragment a jeho zobrazenie sa presunú späť do stavu STARTED

Životný cyklus fragmentu

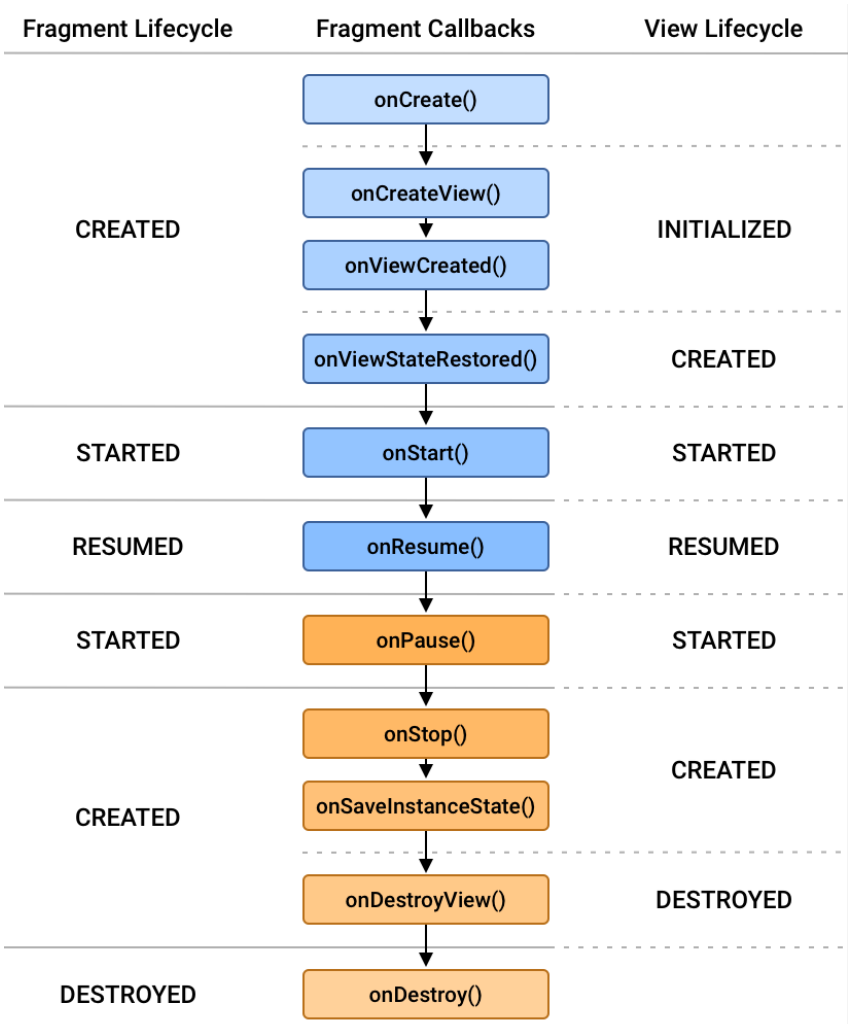


CREATED

Keď fragment už nie je viditeľný, životné cykly fragmentu a jeho zobrazenia sa presunú do stavu CREATED

Nasleduje buď *onResume()*, ak sa aktivita vráti späť do dopredu, alebo *onStop()*, ak sa stane pre používateľa neviditeľnou.

Životný cyklus fragmentu



DESTROYED

Fragment je odstránený, alebo ak je FragmentManager zničený

Lifecycle Methods - to resumed state (interacting with the user)

- *onAttach* - volá sa, keď je fragment spojený s jeho činnosťou.
- *onCreate* - volá sa na počiatočné vytvorenie fragmentu.
- *onCreateView* - vytvorí a vráti hierarchiu pohľadov spojenú s fragmentom.
- *onActivityCreated* - oznámi fragmentu, že jeho aktivita dokončila svoju *android.app.Activity-onCreate*.
- *onViewStateRestored* - oznamuje fragmentu, že všetky uložené stavy jeho hierarchie pohľadov boli obnovené.
- *onStart* - zviditeľní fragment pre používateľa (na základe spustenia jeho obsahujúcej aktivity).
- *onResume* - spôsobí, že fragment začne komunikovať s používateľom (na základe obnovenia jeho obsahujúcej činnosti).

Metódy životného cyklu

- *onPause* - fragment už nie je v interakcii s používateľom, pretože jeho činnosť je pozastavená alebo ho v činnosti modifikuje operácia fragmentu.
- *onStop* - fragment už nie je viditeľný pre používateľa buď preto, že jeho činnosť je zastavená, alebo ho v činnosti modifikuje operácia fragmentu.
- *onDestroyView* - umožňuje fragmentu vyčistiť prostriedky spojené s jeho zobrazením.
- *onDestroy* - volá sa na konečné vyčistenie stavu fragmentu.
- *onDetach* - volá sa bezprostredne pred tým, ako fragment už nie je spojený so svojou aktivitou.

Service

- Aplikačný komponent, ktorý môže vykonávať dlhodobé operácie na pozadí
- Neposkytuje používateľské rozhranie
- Rozšírenie triedy Service
- Všetky služby musíte deklarovať v súbore manifest aplikácie

Typy služieb

- Foreground
- Background
- Bound

- Služba na popredí vykonáva nejakú operáciu, ktorá je pre používateľa viditeľná
- Služby na popredí musia zobrazovať oznámenie
- Toto oznámenie nemožno zrušiť, pokiaľ služba nie je zastavená alebo odstránená
- Pokračuje v prevádzke aj vtedy, keď používateľ neinteraguje s aplikáciou

Typy služieb

- Foreground
- Background
- Bound

- Služba na pozadí vykonáva operáciu, ktorú si používateľ priamo nevšimne
- napr. kompaktné úložisko
- API 26 alebo vyššie - obmedzenia týkajúce sa spúšťania služieb na pozadí, keď samotná aplikácia nie je v popredí, nemali by ste pristupovať k informáciám o polohe z pozadia

Typy služieb

- Foreground
- Background
- Bound
- Viazaná služba ponúka rozhranie klient-server, ktoré umožňuje komponentom komunikovať so službou, posielať požiadavky, prijímať výsledky, a to dokonca naprieč procesmi pomocou medziprocesovej komunikácie (IPC
- Funguje len dovtedy, kým je na ňu naviazaná iná aplikačná zložka.
- Na službu sa môže viazať viacero komponentov naraz, ale keď sa všetky odviažu, služba sa zničí.

Viac informácií

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

MOBILE APPLICATION DEVELOPMENT

User Interface

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Používateľské rozhranie

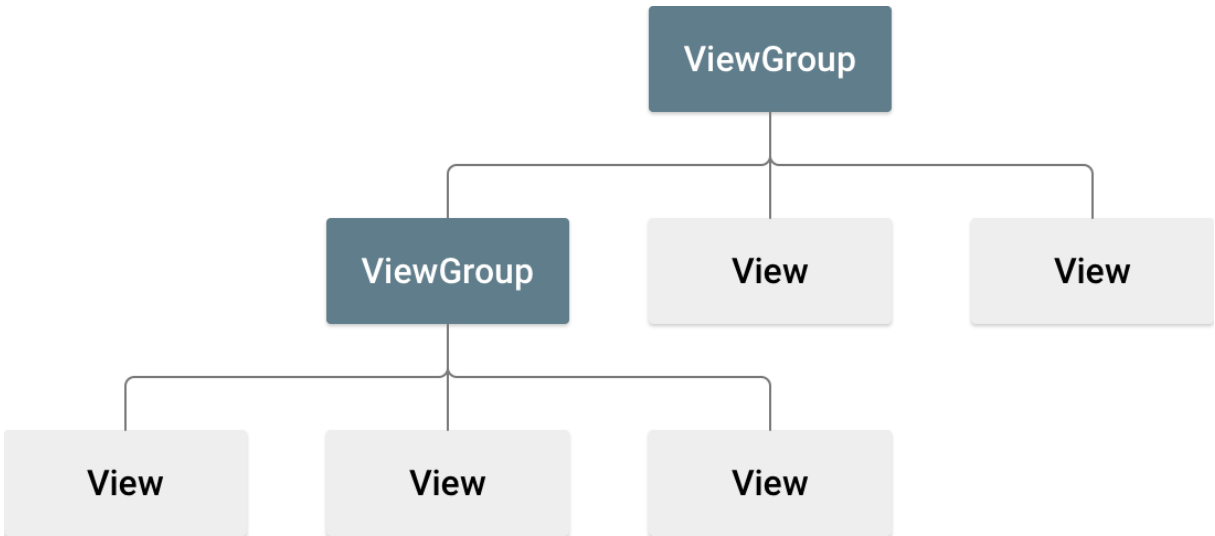
UI

Používateľské rozhranie vašej aplikácie je všetko, čo používateľ vidí a s čím môže interagovať. Systém Android poskytuje množstvo predpripravených komponentov používateľského rozhrania, napríklad **structured layout objects** a **UI controls**, ktoré vám umožňujú vytvoriť grafické používateľské rozhranie vašej aplikácie. Android poskytuje aj ďalšie moduly používateľského rozhrania pre špeciálne rozhrania, ako napríklad **dialogs**, **notifications** a **menus**.

Rozloženia

Layouts

Definuje štruktúru používateľského rozhrania v aplikácii, napríklad v aktivite. Všetky prvky rozloženia sú vytvorené pomocou hierarchie objektov View a ViewGroup.



Deklarovanie rozložení

Rozloženie môžete deklarovať dvoma spôsobmi:

- **Declare UI elements in XML.** - Android poskytuje jednoduchý slovník XML, ktorý zodpovedá triedam a podtriedam View, ako sú triedy pre widgety a rozloženia. Na vytvorenie rozvrhnutia XML pomocou rozhrania drag-and-drop môžete použiť aj editor rozvrhnutia Android Studio.
- **Instantiate layout elements at runtime.** Vaša aplikácia môže vytvárať objekty View a ViewGroup (a manipulovať s ich vlastnosťami) programovo.

Definovanie rozložení v ***XML*** súbor

- Deklarovanie používateľského rozhrania v ***XML*** umožňuje oddeliť prezentáciu aplikácie od kódu, ktorý riadi jej správanie.
- ***View*** - sa zvyčajne nazývajú "widgety" a môžu byť jednou z mnohých podtried, napríklad ***Button or TextView***
- ***ViewGroup*** - zvyčajne sa nazývajú "layouty" a môžu byť jedným z mnohých typov, ktoré poskytujú rôznu štruktúru rozloženia, napríklad ***LinearLayout or ConstraintLayout***
- Ak chcete ladiť rozloženie počas behu, použite nástroj Layout Inspector. `https://developer.android.com/studio/debug/layout-inspector`

Príklad 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 súbor rozloženia musí obsahovať presne jeden koreňový prvok, ktorý musí byť objektom View alebo ViewGroup (napr. LinearLayout)
- Všetky rozloženia ukladáme do ***res/layout/***
- Android podporuje rôzne veľkosti obrazovky

Rôzne veľkosti obrazovky

```
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
```

- Špecifické rozloženia obrazovky môžete poskytnúť vytvorením ďalších adresárov ***res/layout/*** - jeden pre každú konfiguráciu obrazovky, ktorá vyžaduje iné rozloženie
- Použite kvalifikátor dostupnej šírky (napr. ***sw600dp*** - obrazovka s 600dp)
- Použite kvalifikátory orientácie (napr. ***land or port*** - rozloženia na výšku, resp. na šírku)

Typ rozloženia I

- Linear Layout - je skupina zobrazení, ktorá zarovnáva všetky deti v jednom smere, vertikálne alebo horizontálne.
- ~~Relative Layout~~ - je skupina zobrazení, ktorá zobrazuje podriadené zobrazenia v relatívnych pozíciách.
- Constraint Layout -je pohľad na vytváranie veľkých a zložitých rozložení s plochou hierarchiou pohľadov (bez vnorených skupín pohľadov). Je podobné RelativeLayout v tom, že všetky pohľady sú rozložené podľa vzťahov medzi súrodeneckými pohľadmi a nadradeným rozložením, ale je flexibilnejšie ako RelativeLayout
- Table Layout - je pohľad, ktorý zoskupuje pohľady do riadkov a stĺpcov.
- Absolute Layout - umožňuje určiť presné umiestnenie jeho detí.

Typ rozloženia II

- Frame Layout - je zástupný symbol na obrazovke, ktorý môžete použiť na zobrazenie jedného zobrazenia.
- List View - ListView -je skupina zobrazení, ktorá zobrazuje zoznam rolovateľných položiek. (Rozloženia s adaptérom)
- Grid View - GridView je skupina ViewGroup, ktorá zobrazuje položky v dvojrozmernej rolovateľnej mriežke. (Rozloženia s adaptérom)
- `https://www.tutorialspoint.com/android/android_user_interface_layouts.htm`

Atribúty rozloženia I

- *android* : *id* - Toto je ID, ktoré jednoznačne identifikuje pohľad.
- *android* : *layout_width* - Toto je šírka rozloženia.
- *android* : *layout_height* - Toto je výška rozloženia
- *android* : *layout_marginTop* - Toto je dodatočné miesto na hornej strane rozloženia.
- *android* : *layout_marginBottom* - Toto je dodatočné miesto na spodnej strane rozloženia.
- *android* : *layout_marginLeft* - Toto je dodatočné miesto na ľavej strane rozloženia.
- *android* : *layout_marginRight* - Toto je dodatočné miesto na pravej strane rozloženia.
- *android* : *layout_gravity* - Toto určuje, ako sú umiestnené podriadené zobrazenia.

Atribúty rozloženia II

- *android* : *layout_weight* - Toto určuje, aká časť dodatočného priestoru v rozložení sa má prideliť Zobrazeniu.
- *android* : *layout_x* - Toto určuje x-ovú súradnicu rozloženia.
- *android* : *layout_y* - Toto určuje y-ovú súradnicu rozloženia.
- *android* : *layout_width* - Toto je šírka rozloženia.
- *android* : *paddingLeft* - Toto je ľavá výplň vyplnená pre rozloženie.
- *android* : *paddingRight* - Toto je pravý padding vyplnený pre rozloženie.
- *android* : *paddingTop* - Toto je horný padding vyplnený pre rozloženie.
- *android* : *paddingBottom* - Toto je spodná výplň vyplnená pre rozloženie.

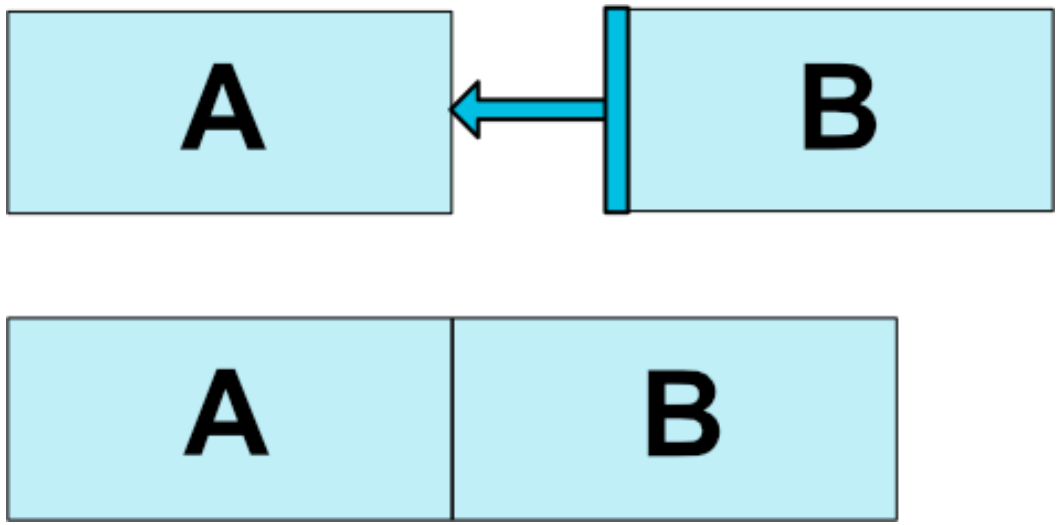
ConstraintLayout

- ConstraintLayout je ***android.view.ViewGroup***, ktorý umožňuje flexibilné umiestnenie a veľkosť widgetov.
- V súčasnosti existujú rôzne typy obmedzení, ktoré môžete použiť:
  - Relatívne umiestnenie
  - Okraje
  - Polohovanie na stred
  - Kruhové polohovanie
  - Správanie sa pri viditeľnosti
  - Rozmerové obmedzenia
  - Reťazce
  - Objekty virtuálnych pomocníkov
  - Optimalizátor
- `https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout`

Relatívne umiestnenie

- Tieto obmedzenia umožňujú umiestniť daný widget vzhľadom na iný widget.
- Widget môžete obmedziť na horizontálnej a vertikálnej osi:
  - Horizontálna os: ľavá, pravá, začiatočná a koncová strana
  - Vertikálna os: horná, dolná strana a základná čiara textu
- Všeobecná koncepcia spočíva v obmedzení danej strany widgetu na inú stranu akéhokoľvek iného widgetu.
- Všetky preberajú referenčné id na iný widget alebo na rodiča (ktorý bude odkazovať na rodičovský kontajner, t. j. ConstraintLayout)

Relatívne umiestnenie - príklad



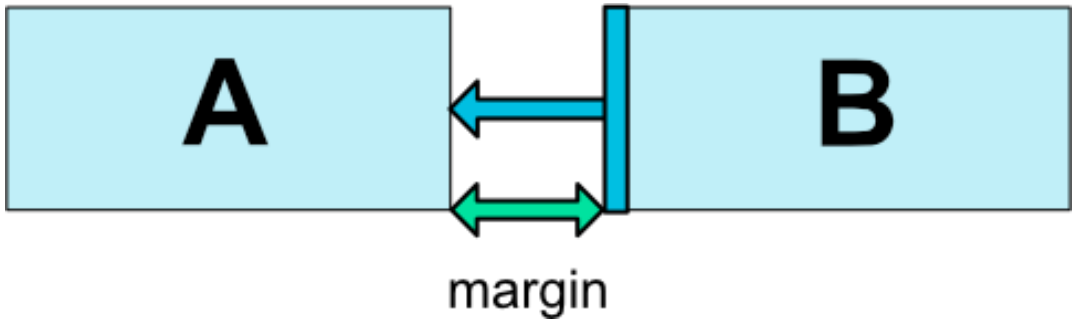

```
1 <Button android:id="@+id/buttonA" ... />
2          <Button android:id="@+id/buttonB" ...
3                  app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```

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

Available constraints

- 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

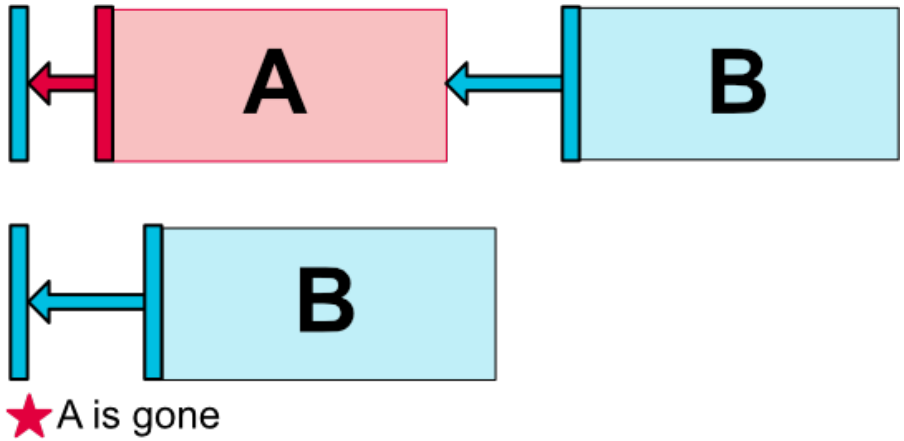
Marže



Ak sú nastavené bočné okraje, použijú sa na príslušné obmedzenia a vynútia si okraj ako medzeru medzi cieľovou a zdrojovou stranou.

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

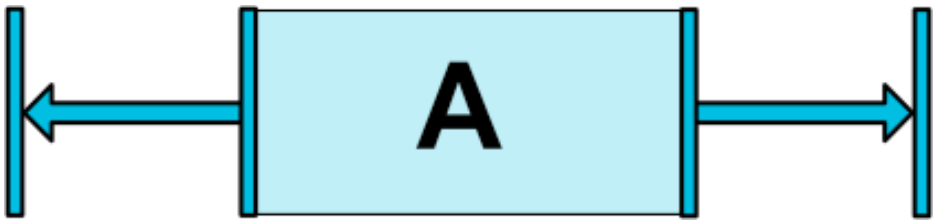
Správanie pri viditeľnosti



ConstraintLayout má špecifické spracovanie widgetov označených ako ***View.GONE***. Widgety **GONE** sa ako zvyčajne nezobrazia a nie sú súčasťou samotného rozloženia (t. j. ich skutočné rozmery sa nezmenia, ak sú označené ako **GONE**).
Z hľadiska výpočtov rozloženia sú však widgety GONE stále jeho súčasťou, s dôležitým rozdielom:

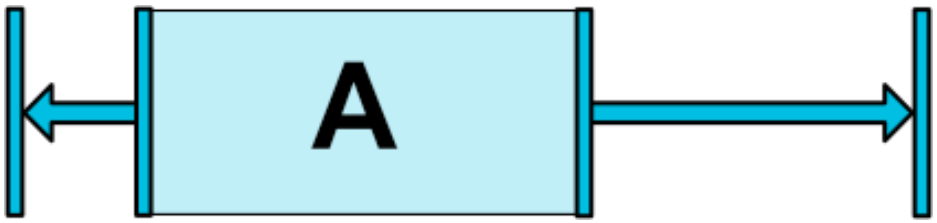
- Pre priechod rozložením sa ich rozmer bude považovať za nulový (v podstate budú vyriešené do bodu)
- Ak majú väzby na iné widgety, budú stále rešpektované, ale všetky okraje budú akoby rovné nule

Centrovanie polohy



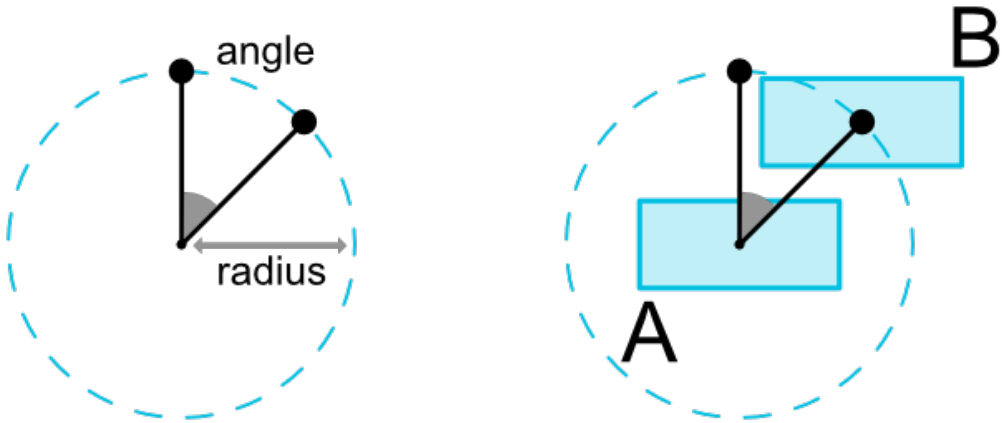

```
1 <androidx.constraintlayout.widget.ConstraintLayout ...>
2             <Button android:id="@+id/button" ...
3                 app:layout_constraintHorizontal_bias="0.3"
4                 app:layout_constraintLeft_toLeftOf="parent"
5                 app:layout_constraintRight_toRightOf="parent/>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

Centrovanie polohy so sklonmi




```
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 polohovanie




```
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" />
```

Možno použiť tieto atribúty:

- layout _constraintCircle : odkazuje na iný widget id
- layout_constraintCircleRadius : vzdialenosť od stredu iného widgetu
- layout_constraintCircleAngle : pod akým uhlom má byť widget umiestnený (v stupňoch, od 0 do 360)

Dimensions constraints

Môžete definovať minimálne a maximálne veľkosti pre samotný ConstraintLayout:

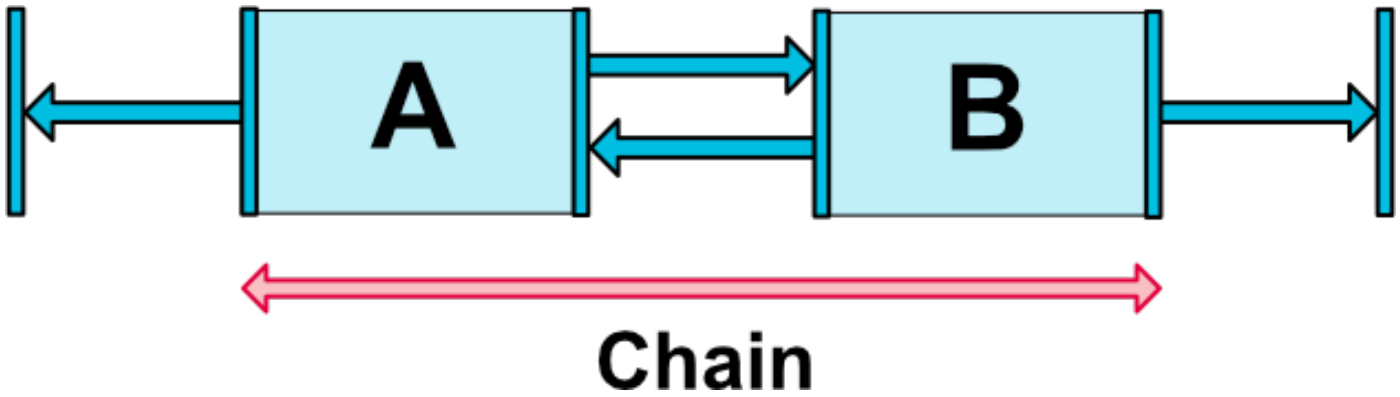
- android:minWidth nastaví minimálnu šírku rozloženia
- android:minHeight nastavuje minimálnu výšku pre rozloženie
- android:maxWidth nastavuje maximálnu šírku rozloženia
- android:maxHeight nastaviť maximálnu výšku pre rozloženie

Rozmery widgetov možno určiť nastavením atribútov android:layout_width a android:layout_height tromi rôznymi spôsobmi:

- Použitím konkrétneho rozmeru (buď doslovnej hodnoty, napríklad 123dp, alebo odkazu na Dimension)
- Použitie funkcie WRAP_CONTENT, ktorá požiada widget o výpočet vlastnej veľkosti
- Použitie hodnoty 0dp, čo je ekvivalent "MATCH_CONSTRAINT"

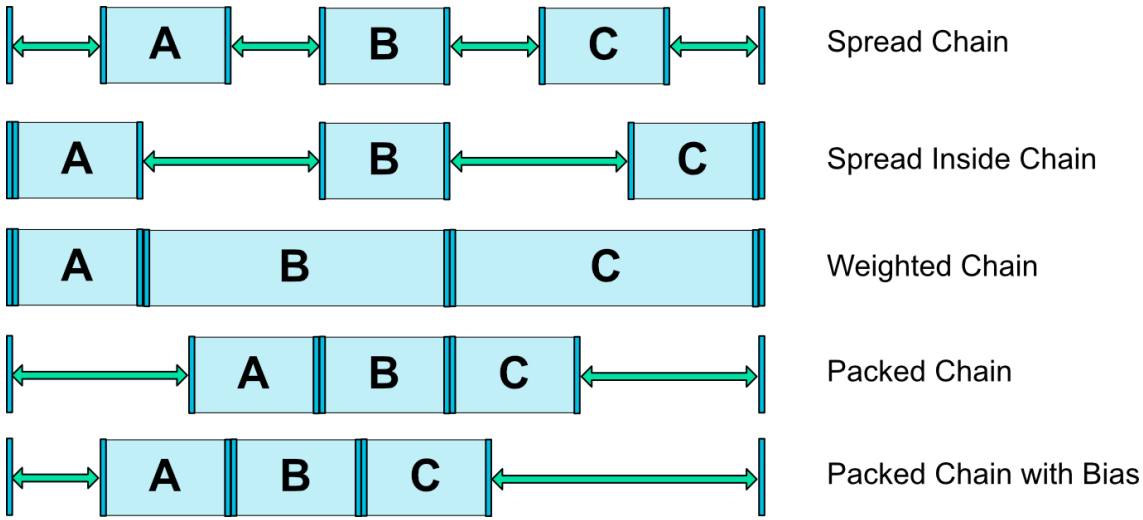
Reťaze

Reťazce poskytujú skupinové správanie v jednej osi (horizontálne alebo vertikálne). Ostatné osi môžu byť obmedzené nezávisle.



Súbor miniaplikácií sa považuje za reťaz, ak sú navzájom prepojené obojsmerným spojením (na obrázku je znázornená minimálna reťaz s dvoma miniaplikáciami).

Štýl reťaze I



- CHAIN_SPREAD – prvky budú rozložené (predvolený štýl)
- Vážený reťazec – v režime CHAIN_SPREAD, ak sú niektoré widgety nastavené na MATCH_CONSTRAINT, rozdelia dostupný priestor
- CHAIN_SPREAD_INSIDE – podobné, ale koncové body reťazca nebudú rozložené

Štýl reťaze II

- CHAIN_PACKED – prvky reťazca budú na seba nabalené. Atribút horizontálneho alebo vertikálneho vychýlenia potomka potom ovplyvní umiestnenie zabalených prvkov

Viac informácií

https://developer.android.com/training/constraint-layout

https://constraintlayout.com/basics/setup.html

https://www.raywenderlich.com/155-android-listview-tutorial-with-kotlin

Nástroje

- 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 - Cieľ

- Vytvorte vizuálny jazyk, ktorý syntetizuje klasické princípy dobrého dizajnu s inováciami a možnosťami technológie a vedy.
- Vytvorte jednotný základný systém, ktorý umožní jednotný zážitok na rôznych platformách a pri rôznych veľkostiach zariadení. Mobilné zásady sú základom, ale dotyk, hlas, myš a klávesnica sú prvotriedne metódy vstupu.
- Material je dizajnový systém vytvorený spoločnosťou Google, ktorý pomáha tímom vytvárať vysokokvalitné digitálne zážitky pre Android, iOS, Flutter a web.

Material Design - Komponenty

- Tyče aplikácií
- Bunner
- Karta
- Plávajúce tlačidlo
- Dátové tabuľky
- Dialógy
- Zoznam, zoznam obrázkov
- Snackbars
- ToolTip
- ...

Tématika materiálu

- Material Theming sa vzťahuje na prispôsobenie aplikácie Material Design tak, aby lepšie odrážala značku vášho produktu.
- Môžete si ho nadefinovať:
  - Farba
  - Typografia
  - Tvar, napr. zmeniť veľkosť alebo rohy tlačidiel.

MOBILE APPLICATION DEVELOPMENT

Senzory

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Senzory v mobilnom zariadení

- Inerciálne snímače: 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 prostredia: Temperature, Humidity, Barometric Pressure, Gas (CO...)*
- Bezdrôtové snímače/RF:GPS, WiFi, Cellular A-GPS, Bluetooth Low Energy, NFC
- Ostatné snímače:MEMS Microphones, Biometric/Fingerprint*, BioSensors*

MEMS Sensor

* - Senzory budúcnosti

Senzory

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

Príklady aplikácií využívajúcich jeden senzor

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

Príklady aplikácií využívajúcich fúziu údajov zo senzorov

- 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(VŠETKY SENZORY !!)
- ...

Android - senzor

Platforma Android podporuje tri široké kategórie snímačov:

- Senzory pohybu - Tieto senzory merajú sily zrýchlenia a. rotačné sily pozdĺž troch osí. Do tejto kategórie patria akcelerometre, gravitačné snímače, gyroskopy a snímače vektora rotácie.
- Senzory prostredia - tieto senzory merajú rôzne rôzne parametre prostredia, ako sú teplota a tlak vzduchu, osvetlenie a vlhkosť. vlhkosť. Do tejto kategórie patria barometre, fotometre a teplomery.
- Senzory polohy - tieto senzory merajú fyzickú polohu zariadenia. zariadenia. Do tejto kategórie patria snímače orientácie a magnetometre.

Android - senzor

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 - senzor

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

- Určenie, ktoré snímače sú v zariadení k dispozícii.
- Určenie možností jednotlivých snímačov, ako je ich maximálny rozsah, výrobca, požiadavky na napájanie a rozlíšenie.
- Získavanie nespracovaných údajov zo snímačov a určenie minimálnej rýchlosti získavania údajov zo snímačov.
- Registrácia a zrušenie registrácie poslucháčov udalostí senzora, ktorí monitorujú zmeny senzora.

Sensor Framework

- Rámec senzorov Android - prístup k nespracovaným údajom zo senzorov a ich extrakcia údajov zo senzorov pomocou rámca senzorov Android.
- Rámec senzorov je súčasťou balíka *android.hardware* a obsahuje nasledujúce triedy a rozhrania. obsahuje nasledujúce triedy a rozhrania:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorManager

Pomocou tejto triedy môžeme vytvoriť inštanciu služby senzora. Táto trieda poskytuje rôzne metódy na prístup k senzorom a ich zoznam, registráciu a odregistrovanie poslucháčov udalostí senzorov a získavanie informácií o orientácii. Táto trieda poskytuje aj niekoľko konštánt senzorov, ktoré sa používajú na hlásenie presnosti senzorov, nastavenie rýchlosti zberu údajov a kalibráciu

Sensor Framework

- Rámec senzorov Android - prístup k nespracovaným údajom zo senzorov a ich extrakcia údajov zo senzorov pomocou rámca senzorov Android.
- Rámec senzorov je súčasťou balíka *android.hardware* a obsahuje nasledujúce triedy a rozhrania. obsahuje nasledujúce triedy a rozhrania:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

Sensor

Pomocou tejto triedy môžeme vytvoriť inštanciu konkrétneho senzora. Táto trieda poskytuje rôzne metódy na určenie schopností senzora.

Sensor Framework

- Rámec senzorov Android - prístup k nespracovaným údajom zo senzorov a ich extrakcia údajov zo senzorov pomocou rámca senzorov Android.
- Rámec senzorov je súčasťou balíka *android.hardware* a obsahuje nasledujúce triedy a rozhrania. obsahuje nasledujúce triedy a rozhrania:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorEvent

Systém používa túto triedu na vytvorenie objektu udalosti senzora, ktorý poskytuje informácie o udalosti senzora. Objekt udalosti snímača obsahuje tieto údaje informácie: nespracované údaje snímača, typ snímača, ktorý udalosť vygeneroval, presnosť údajov a časovú značku udalosti.

Sensor Framework

- Rámec senzorov Android - prístup k nespracovaným údajom zo senzorov a ich extrakcia údajov zo senzorov pomocou rámca senzorov Android.
- Rámec senzorov je súčasťou balíka *android.hardware* a obsahuje nasledujúce triedy a rozhrania. obsahuje nasledujúce triedy a rozhrania:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorEventListener

Pomocou tohto rozhrania môžete vytvoriť dve metódy spätného volania, ktoré dostanú oznámenia (udalosti senzora) pri zmene hodnôt senzora alebo pri zmene presnosti senzora.

Typické kroky

- Identifikácia snímačov a možností snímačov
- Monitorovanie udalostí senzorov

Identifikácia snímača

Identifikácia snímačov a schopností snímačov počas behu je užitočná, ak má vaša aplikácia funkcie, ktoré sa spoliehajú na určité typy snímačov alebo schopností. Môžete napríklad chcieť identifikovať všetky snímače, ktoré sú prítomné na zariadení, a vypnúť všetky funkcie aplikácie, ktoré sa spoliehajú na snímače, ktoré nie sú prítomné. Podobne môžete chcieť identifikovať všetky snímače určitého typu, aby ste mohli vybrať implementáciu snímača, ktorá má optimálny výkon pre vašu aplikáciu.

Typické kroky

- Identifikácia snímačov a možností snímačov
- Monitorovanie udalostí senzorov

Monitorovanie senzorov

Monitorovanie udalostí senzora je spôsob získavania nespracovaných údajov zo senzora. Udalosť snímača nastane vždy, keď snímač zistí zmenu parametrov, ktoré meria. Udalosť senzora poskytuje štyri informácie: názov senzora, ktorý udalosť spustil, časovú značku udalosti, presnosť udalosti a nespracované údaje senzora, ktorý udalosť spustil.

Identifikácia snímačov a overenie ich schopností

1. Získanie odkazu na službu senzora

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

2. Zobrazenie zoznamu všetkých snímačov v zariadení

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

2. Alebo použite inú konštantu namiesto TYPE_ALL ako napr. TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, or TYPE_GRAVITY.

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

Schopnosti

```
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 }
```

Užitočné metódy preberania overovania snímačov

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

Monitorovanie udalostí senzorov

Na monitorovanie nespracovaných údajov zo senzorov je potrebné implementovať dve metódy spätného volania, ktoré sú dostupné prostredníctvom rozhrania *SensorEventListener*: onAccuracyChanged() and onSensorChanged()

Zmiany dokĹ‚adnoĹ›ci czujnikÄ‚Å‚w

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

Senzor hlási novú hodnotu

```
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
```

Monitorovanie udalostí senzorov

```
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
```

Monitorovanie udalostí snímača - záznam

```
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" />
```

Osvedčené postupy pre prístup k senzorom a ich používanie

- Zbieranie údajov zo senzorov len v popredí
- Zrušiť registráciu poslucháčov senzorov
- Otestujte s emulátorom Androidu.
- Neblokujte metódu onSensorChanged()
- Vyhnite sa používaniu zastaraných metód alebo typov senzorov.
- Pred použitím senzorov ich overte
- Starostlivo vyberajte odpory senzorov

Zber údajov zo snímačov na pozadí

Na zariadeniach so systémom Android 9 (úroveň API 28) alebo novším:

- Senzory, ktoré používajú režim nepretržitého hlásenia, ako sú akcelerometre a gyroskopy, neprijímajú udalosti.
- Snímače, ktoré používajú režim hlásenia pri zmene alebo jednorazové hlásenie, neprijímajú udalosti.

Osvedčené postupy pre prístup k senzorom a ich používanie

- Zbieranie údajov zo senzorov len v popredí
- Zrušiť registráciu poslucháčov senzorov
- Otestujte s emulátorom Androidu.
- Neblokujte metódu onSensorChanged()
- Vyhnite sa používaniu zastaraných metód alebo typov senzorov.
- Pred použitím senzorov ich overte
- Starostlivo vyberajte odpory senzorov

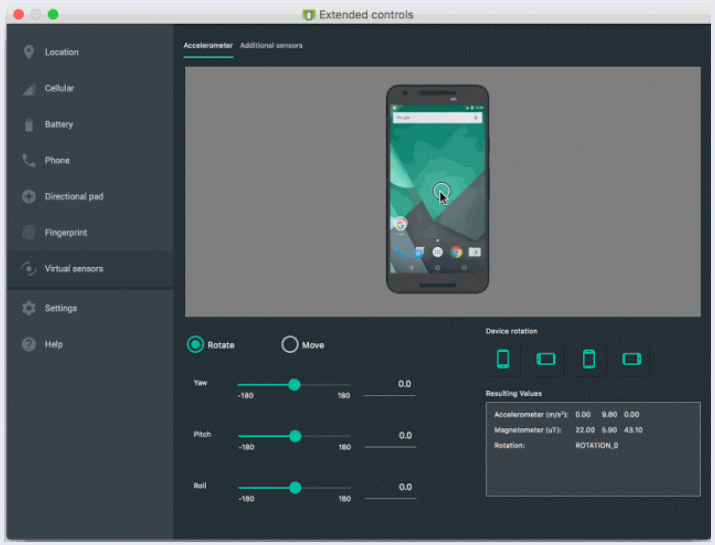
Zrušenie registrácie zoznamov snímačov

Nezabudnite zrušiť registráciu poslucháča snímača, keď ho skončíte používať alebo keď je činnosť snímača pozastavená. Ak je poslucháč senzora zaregistrovaný a jeho činnosť je pozastavená, senzor bude naďalej získavať údaje a využívať zdroje batérie, pokiaľ ho neodregistrujete.

Osvedčené postupy pre prístup k senzorom a ich používanie

- Zbieranie údajov zo senzorov len v popredí
- Zrušiť registráciu poslucháčov senzorov
- Otestujte s emulátorom Androidu.
- Neblokujte metódu onSensorChanged()
- Vyhnite sa používaniu zastaraných metód alebo typov senzorov.
- Pred použitím senzorov ich overte
- Starostlivo vyberajte odpory senzorov

Testovanie pomocou emulátora systému Android



Osvedčené postupy pre prístup k senzorom a ich používanie

- Zbieranie údajov zo senzorov len v popredí
- Zrušiť registráciu poslucháčov senzorov
- Otestujte s emulátorom Androidu.
- Neblokujte metódu onSensorChanged()
- Vyhnite sa používaniu zastaraných metód alebo typov senzorov.
- Pred použitím senzorov ich overte
- Starostlivo vyberajte odpory senzorov

Metódu neblokujeme *onSensorChanged()*

- Údaje zo senzorov sa môžu meniť vysokou rýchlosťou - systém môže metódu volať pomerne často *onSensorChanged(SensorEvent)*.
- Filtrovanie alebo redukcia údajov zo senzorov by sa mala vykonávať mimo metódy onSensorChanged(SensorEvent)

Osvedčené postupy pre prístup k senzorom a ich používanie

- Zbieranie údajov zo senzorov len v popredí
- Zrušiť registráciu poslucháčov senzorov
- Otestujte s emulátorom Androidu.
- Neblokujte metódu onSensorChanged()
- Vyhnite sa používaniu zastaraných metód alebo typov senzorov.
- Pred použitím senzorov ich overte
- Starostlivo vyberajte odpory senzorov

Starostlivo vyberte príslušenstvo snímača

- Pri registrácii senzora pomocou metódy registerListener() sa uistite, že ste zvolili rýchlosť prenosu údajov, ktorá je vhodná pre vašu aplikáciu alebo prípad použitia.
- Ak umožníte systému odosielať ďalšie údaje, ktoré nepotrebujete, plytváte systémovými zdrojmi a spotrebúvate energiu batérie.

Snímač pohybu

Senzory pohybu sú užitočné na monitorovanie **ruch urzÄ...dzenia, taki jak przechylanie, potrzÄ...sanie, obracanie lub koĹ‚ysanie.**

. Možné architektúry senzorov sa líšia v závislosti od typu senzora:

- Senzory gravitácie, lineárneho zrýchlenia, rotačného vektora, významného pohybu, počítadla krokov a detektora krokov sú buď hardvérové, alebo softvérové.
- Snímače akcelerometra a gyroskopu sú vždy hardvérové.

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

Príklad - Snímač pohybu

- Source + Descriptionhttps://www.raywenderlich.com/10838302-sensors-tutorial-for-android-getting-started

Snímače polohy

Snímače polohy sú užitočné na určenie fyzickej polohy zariadenia vo svetovom referenčnom systéme. Napríklad snímač geomagnetického poľa sa môže použiť v spojení s akcelerometrom na určenie polohy zariadenia vzhľadom na severný magnetický pól.

Vypočítať orientáciu zariadenia

```
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)
```

Vypočítať orientáciu zariadenia

Systém vypočíta orientačné uhly pomocou snímača geomagnetického poľa zariadenia v kombinácii s akcelerometrom zariadenia. Pomocou týchto dvoch hardvérových snímačov systém poskytuje údaje pre nasledujúce tri uhly orientácie:

- **Azimuth (degrees of rotation about the -z axis)** Toto je uhol medzi aktuálnym smerom kompasu zariadenia a magnetickým severom. Ak horný okraj zariadenia smeruje na magnetický sever, azimut je 0 stupňov; ak horný okraj smeruje na juh, azimut je 180 stupňov. Podobne, ak horná hrana smeruje na východ, azimut je 90 stupňov; ak horná hrana smeruje na západ, azimut je 270 stupňov.

Vypočítať orientáciu zariadenia

Systém vypočíta orientačné uhly pomocou snímača geomagnetického poľa zariadenia v kombinácii s akcelerometrom zariadenia. Pomocou týchto dvoch hardvérových snímačov systém poskytuje údaje pre nasledujúce tri uhly orientácie:

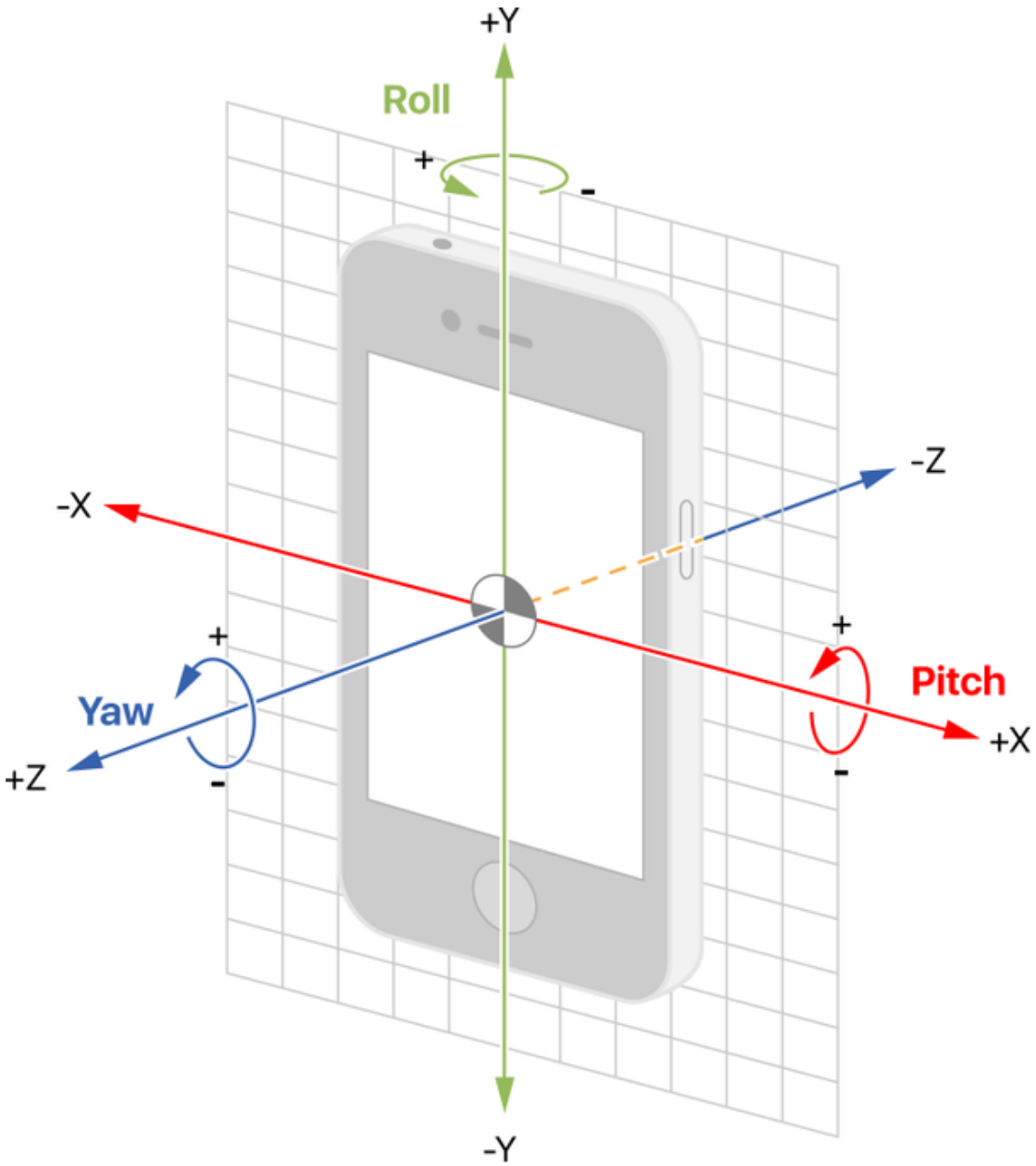
- **Pitch (degrees of rotation about the x axis)** Toto je uhol medzi rovinou rovnobežnou s obrazovkou zariadenia a rovinou rovnobežnou so zemou. Ak držíte zariadenie rovnobežne so zemou, pričom spodný okraj zariadenia je najbližšie k vám, a horný okraj zariadenia nakloníte smerom k zemi, uhol sklonu sa stane kladným. Naklonenie v opačnom smere - posunutie horného okraja zariadenia smerom od zeme - spôsobí, že uhol sklonu sa stane záporným. Rozsah hodnôt je od -180 stupňov do 180 stupňov.

Vypočítať orientáciu zariadenia

Systém vypočíta orientačné uhly pomocou snímača geomagnetického poľa zariadenia v kombinácii s akcelerometrom zariadenia. Pomocou týchto dvoch hardvérových snímačov systém poskytuje údaje pre nasledujúce tri uhly orientácie:

- **Roll (degrees of rotation about the y axis)** Toto je uhol medzi rovinou kolmou na obrazovku zariadenia a rovinou kolmou na zem. Ak držíte zariadenie rovnobežne so zemou, pričom spodný okraj zariadenia je najbližšie k vám, a ľavý okraj zariadenia nakloníte smerom k zemi, uhol sklonu sa stane kladným. Naklonenie v opačnom smere - posunutie pravého okraja zariadenia smerom k zemi - spôsobí, že uhol sklonu bude záporný. Rozsah hodnôt je od -90 stupňov do 90 stupňov.

Orientácia zariadenia



Senzory prostredia

- Pomocou týchto snímačov môžete monitorovať relatívnu vlhkosť, intenzitu okolitého svetla, tlak a teplotu okolia v blízkosti zariadenia so systémom Android.
- Všetky štyri snímače okolia sú hardvérové a sú k dispozícii len vtedy, ak ich výrobca zariadenia zabudoval do zariadenia.
- Senzory prostredia vracajú pre každú dátovú udalosť jednu hodnotu senzora

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]

Príklad

- 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 na určenie polohy zariadenia a nepriamo aj používateľa. V mobilných systémoch je poloha jednou z jedinečných funkcií na vytváranie aplikácií zohľadňujúcich polohu.

- Umožňuje určiť polohu zariadenia
- Všeobecná poloha
- Presná poloha
- Služby Google Play - odporúčaná metóda určovania polohy v systéme Android

Príklady aplikácií

- Informácie špecifické pre danú lokalitu (miestna predpoveď počasia, miestne správy/správy, koncentrácia alergénov)
- Informácie o blízkych zdrojoch (banka, lekáreň, pohostinstvo)
- Interaktívne mapy a turistické informácie
- Mobilná reklama závislá od miesta
- Riadenie mobilných pracovníkov

Google Play services

- Mechanizmus na určovanie polohy na základe údajov od rôznych poskytovateľov, napr. modul GNSS (GPS), modul WiFi alebo Bluetooth.
- Fúzovaný poskytovateľ polohy
- Rýchlejšie určovanie polohy
- Znížená spotreba energie
- Ďalšie možnosti, napr. geofencing, detekcia aktivity

Pridanie knižnice služieb 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 }
```

Kroky potrebné na definovanie miesta

- Definícia typu polohy
- Žiadosť o povolenie na umiestnenie zariadenia
- Sťahovanie polohy (posledné známe, cyklické sťahovanie polohy)
- Použitie polohy, napr. na zobrazenie bodu na mape

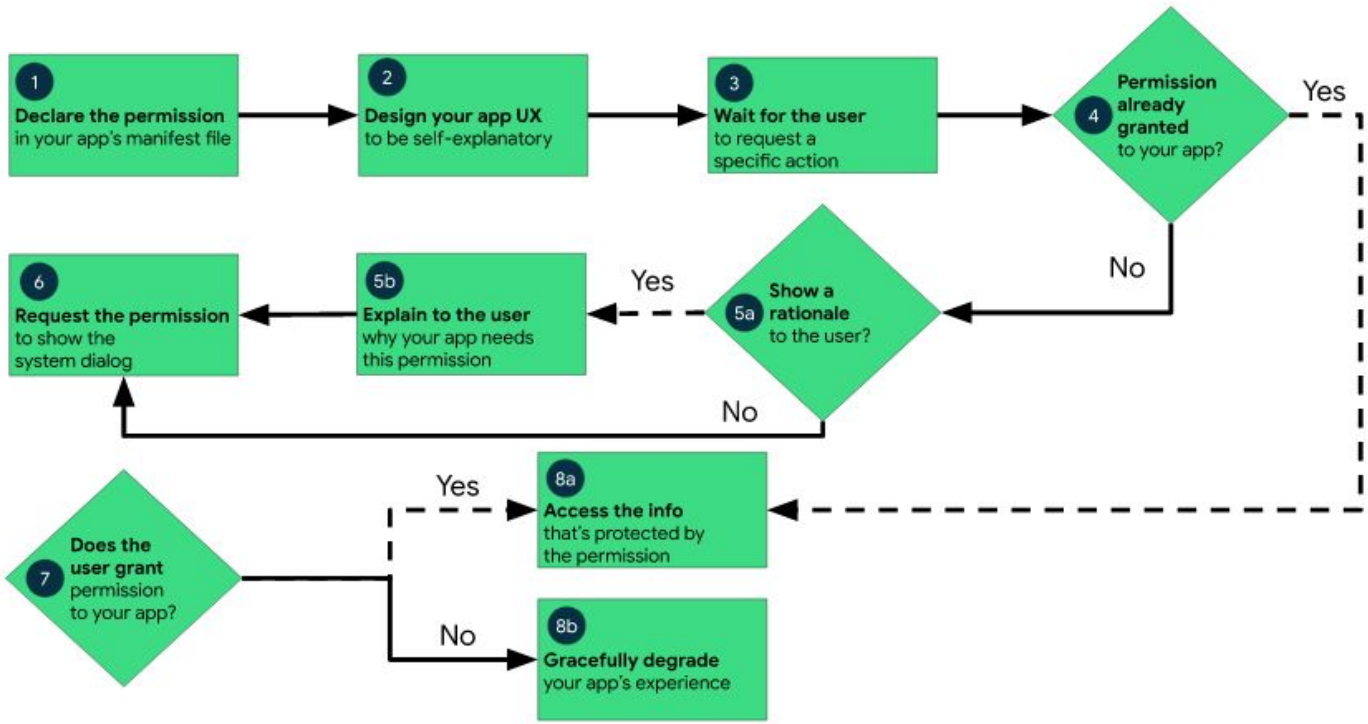
Definícia povolenia

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest ... >
3  ...
4    <!-- Always include this permission -->
5    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
6
7    <!-- Include only if your app benefits from precise location access. -->
8    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
9    <!-- Recommended for Android 9 (API level 28) and lower. -->
10   <!-- Required for Android 10 (API level 29) and higher. -->
11 ...
12   <!-- Required only when requesting background location access on
13        Android 10 (API level 29) and higher. -->
14   <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"
         />
15 ...
16   <application...>
17     <service
18       android:name="MyNavigationService"
19       android:foregroundServiceType="location" ... >
20     </service>
21   </application>
22 ...
23 </manifest>
```

Typy lokalít

- Všeobecná/hrubá poloha
- Presná poloha
- Umiestnenie v popredí
- Umiestnenie v pozadí

Žiadosť o povolenia - workflow



Príklad žiadosti o povolenie

```
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        }
```

Používanie služby Fused Location Provider - posledná známa poloha

Definícia objektu

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

Inicializácia

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

Vyhľadanie poslednej známej 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            }
```

Kontinuálne umiestnenie - I

- Definícia objektu
- Inicializácia
- Definovanie návratových metód
- Povolenie a zakázanie informácií o aktualizácii polohy
- Výber obnovovacej frekvencie

Kontinuálne umiestnenie II

Definícia objektov

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

Inicializácia

```
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        }
```

Priebežné umiestnenie III

Aktivácia informácií o aktualizácii 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             }
```

Deaktivácia informácií o aktualizácii 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

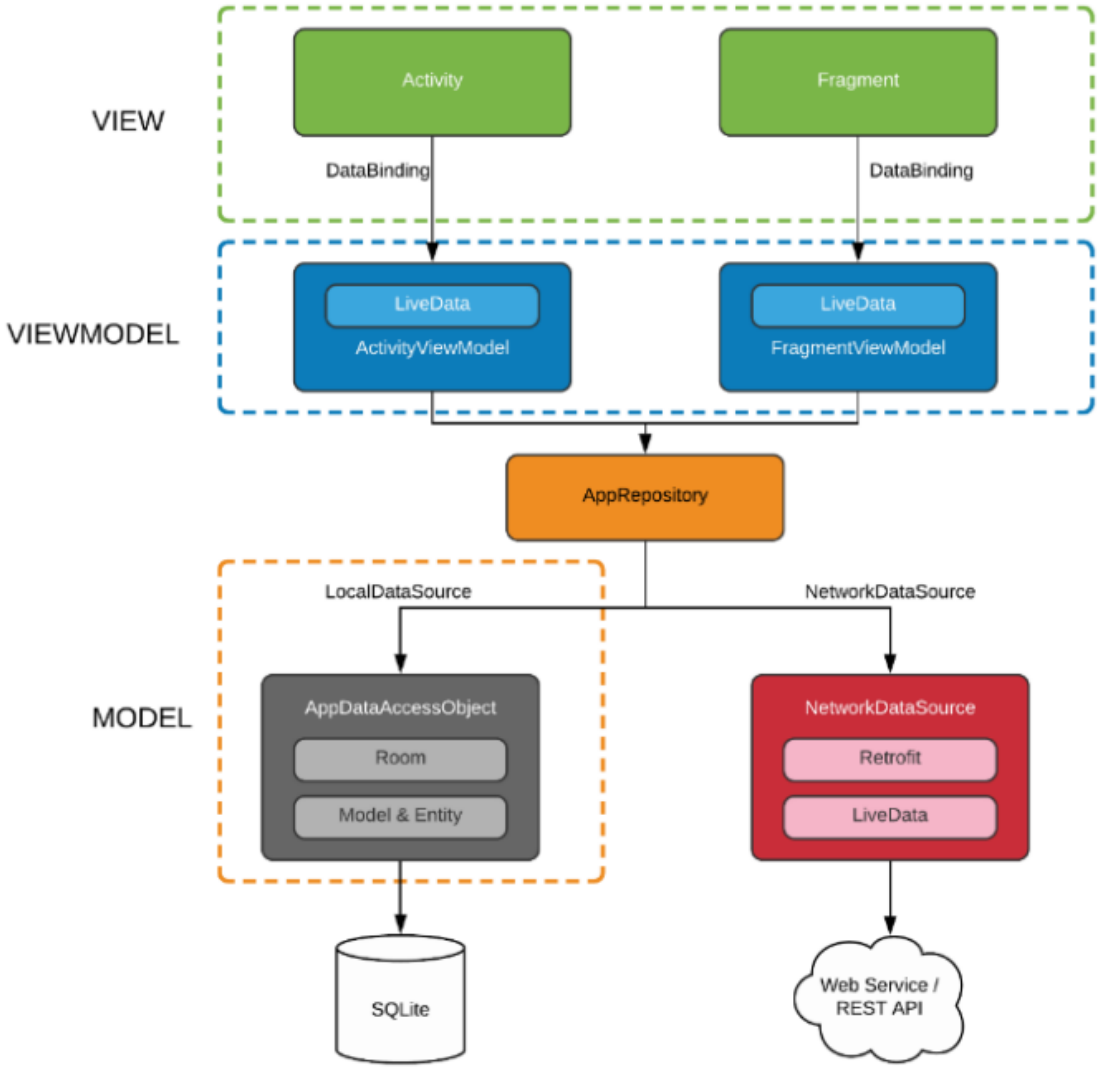
MVVM

MVVM

Model — View — ViewModel je priemyselne uznávaný vzor softvérovej architektúry, ktorý prekonáva všetky nevýhody návrhových vzorov MVP a MVC. MVVM navrhuje oddeliť logiku prezentácie údajov (Views alebo UI) od základnej obchodnej logiky aplikácie.

- **Model** - Tu sa uchovávajú údaje aplikácie. Nemôže priamo komunikovať so zobrazením. Vo všeobecnosti sa odporúča sprístupniť údaje ViewModelu prostredníctvom Observables.
- **View** - Predstavuje používateľské rozhranie aplikácie bez akejkoľvek aplikačnej logiky. Sleduje ViewModel.
- **ViewModel** - Funguje ako spojovací článok medzi Modelom a Zobrazením. Je zodpovedný za transformáciu údajov z Modelu. Poskytuje dátové toky do View. Taktiež využíva spätné volania na aktualizáciu View. Vyžiada si údaje 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

Architektonické komponenty

- Kolekcia knižníc, ktoré vám pomôžu navrhovať robustné, testovateľné a udržiavateľné aplikácie
- Slúžia ako hlavný návod pri budovaní chrbtice architektúry nášho projektu
- Pomocou komponentov architektúry Androidu sa nemusíme veľmi starať o správu životného cyklu aplikácie ani o načítanie údajov do nášho používateľského rozhrania

Komponenty

- ViewModel
- LiveData
- Lifecycle
- Extend LiveData

ViewModel

- Trieda ViewModel je navrhnutá tak, aby uchovávala a spravovala údaje súvisiace s používateľským rozhraním spôsobom, ktorý zohľadňuje životný cyklus.
- To umožňuje, aby údaje prežili zmeny konfigurácie, ako napríklad otočenie obrazovky.
- Architecture Components poskytuje pomocnú triedu ViewModel pre kontrolér používateľského rozhrania, ktorý je zodpovedný za prípravu údajov pre používateľské rozhranie.
- Objekty ViewModel sa automaticky uchovávajú počas zmien konfigurácie, takže údaje, ktoré uchovávajú, sú okamžite k dispozícii pre ďalšiu aktivitu alebo inštanciu fragmentu.

ViewModel výhody

- Umožňuje zachovať stav používateľského rozhrania
- Poskytuje prístup k obchodnej logike.

Perzistencia

ViewModel umožňuje prežitie prostredníctvom stavu, ktorý ViewModel uchováva, aj operácií, ktoré ViewModel spúšťa. Toto ukladanie do vyrovnávacej pamäte znamená, že nemusíte znovu načítavať údaje prostredníctvom bežných konfiguračných zmien, ako je napríklad otočenie obrazovky.

Prístup k biznis logike

ViewModel je vhodným miestom na spracovanie obchodnej logiky vo vrstve používateľského rozhrania. ViewModel má na starosti aj spracovanie udalostí a ich delegovanie na iné vrstvy hierarchie, keď je potrebné použiť obchodnú logiku na zmenu údajov aplikácie.

ViewModel - príklad

Definovanie 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 }
```

Prístup k zoznamu z aktivity je nasledovný

```
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 trieda pozorovateľného držiteľa údajov.
- Na rozdiel od bežných pozorovateľných tried LiveData si uvedomuje životný cyklus, čo znamená, že rešpektuje životný cyklus iných komponentov aplikácie, ako sú aktivity, fragmenty alebo služby.
- Toto povedomie zabezpečuje, že LiveData aktualizuje iba pozorovateľov komponentov aplikácie, ktoré sú v aktívnom stave životného cyklu.
- LiveData sa riadi vzorom pozorovateľa. LiveData upozorňuje objekty pozorovateľov, keď sa zmení stav životného cyklu
- LiveData tiež automaticky posúva existujúcu hodnotu, ak existuje, do všetkých nových registrovaných objektov Observer.
- V spojení s funkciou automatického odstraňovania je vďaka tomu LiveData veľmi výhodný na riešenie konfiguračných zmien.

Typy v LiveData

- LiveData - sú predvolene nemenné. Pomocou LiveData môžeme údaje len pozorovať a nemôžeme ich nastavovať.
- MutableLiveData - je mutabilný a je podtriedou LiveData. V MutableLiveData môžeme pozorovať a nastavovať hodnoty pomocou metód postValue() a setValue()
- MediatorLiveData - môže pozorovať iné objekty LiveData, napríklad zdroje, a reagovať na ich udalosti onChange().

Rozšírenie LiveData

- LiveData je komponent, ktorý si uvedomuje životný cyklus, a preto vykonáva svoje funkcie podľa stavu životného cyklu ostatných komponentov aplikácie.
- Ak je stav životného cyklu pozorovateľa aktívny, t. j. buď STARTED alebo RESUMED, iba vtedy LiveData aktualizuje aplikačnú zložku.

```
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 }
```

Rozšírenie LiveData - 2

- **onActive()** - metóda sa volá, keď má objekt LiveData aktívneho pozorovateľa. To znamená, že z tejto metódy musíte začať pozorovať aktualizácie cien akcií.
- **onInactive()** - metóda sa volá, keď objekt LiveData nemá žiadneho aktívneho pozorovateľa. Keďže žiadny pozorovateľ nepočúva, nie je dôvod zostať pripojený.

Kroky na vytvorenie aplikácie s návrhovým paternom MVVM

- Pridanie DataBinding a implementácií do súboru Gradle
- Vytvorenie novej triedy pre Model
- Vytvorenie novej triedy pre ViewModel
- Vylepšenie triedy View
- Zmena rozloženia

Konfigurácia 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)
  }
}
```

Zmena kódu zobrazenia

```
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 rozloženie

```
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

Úloisko

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Monosti ukladania v systéme Android

- Úloisko pecifické pre aplikáciu
- Zdieané úloisko
- Predvoby
- Databázy

Úloisko pecifické pre aplikáciu

Ukladajte súbory, ktoré sú urené len pre pouitie vaej aplikácie, bu do vyhradených adresárov v rámci zväzku interného úloiska, alebo do rôznych vyhradených adresárov v rámci externého úloiska. Adresáre v rámci interného úloiska pouívajte na ukladanie citlivých informácií, ku ktorým by iné aplikácie nemali ma prístup.

Monosti ukladania v systéme Android

- Úloisko pecifické pre aplikáciu
- Zdieané úloisko
- Predvoby
- Databázy

Zdieané úloisko

Ukladajte súbory, ktoré vaa aplikácia plánuje zdiea s inými aplikáciami, vrátane médií, dokumentov a iných súborov.

Monosti ukladania v systéme Android

- Úloisko pecifické pre aplikáciu
- Zdieané úloisko
- Predvoby
- Databázy

Predvoby

Ukladajte súkromné, primitívne údaje v pároch kú-hodnota.

Monosti ukladania v systéme Android

- Úloisko pecifické pre aplikáciu
- Zdieané úloisko
- Predvoby
- Databázy

Databázy

Uklada truktúrované údaje v súkromnej databáze pomocou kninice Room persistence.

Prehad úloísk

	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

Ktorý si vybra?

- Koko miesta potrebujú vae údaje?
- Aký spoahlivý musí by prístup k údajom?
- Aký druh údajov potrebujete uklada?
- Mali by by údaje pre vau aplikáciu súkromné?

Ktorý si vybra?

- Koko miesta potrebujú vae údaje?
- Aký spoahlivý musí by prístup k údajom?
- Aký druh údajov potrebujete uklada?
- Mali by by údaje pre vau aplikáciu súkromné?

Koko miesta potrebujú vae údaje?

Interné úloisko má obmedzený priestor pre údaje pecifické pre aplikáciu. Ak potrebujete uloi znané mnostvo údajov, pouite iné typy úloísk.

Ktorý si vybra?

- Koko miesta potrebujú vae údaje?
- Aký spoahlivý musí by prístup k údajom?
- Aký druh údajov potrebujete uklada?
- Mali by by údaje pre vau aplikáciu súkromné?

Aký spoahlivý musí by prístup k údajom?

Ak základná funkcia vaej aplikácie vyaduje urité údaje, napríklad pri spustení aplikácie, umiestnite údaje do adresára interného úloiska alebo databázy. Súbory pecifické pre aplikáciu, ktoré sú uloené v externom úloisku, nie sú vdy prístupné, pretoe niektoré zariadenia umoujú pouívateom odstráni fyzické zariadenie, ktoré zodpovedá externému úloisku.

Ktorý si vybra?

- Koko miesta potrebujú vae údaje?
- Aký spoahlivý musí by prístup k údajom?
- Aký druh údajov potrebujete uklada?
- Mali by by údaje pre vau aplikáciu súkromné?

Aké údaje potrebujete uloi?

Ak máte údaje, ktoré majú význam len pre vau aplikáciu, pouite úloisko pecifické pre aplikáciu. V prípade mediálneho obsahu, ktorý je moné zdiea, pouite zdieané úloisko, aby k nemu mali prístup aj iné aplikácie. Pre truktúrované údaje pouite bu preferencie (pre údaje typu kú-hodnota), alebo databázu (pre údaje, ktoré obsahujú viac ako 2 stpce).

Ktorý si vybra?

- Koko miesta potrebujú vae údaje?
- Aký spoahlivý musí by prístup k údajom?
- Aký druh údajov potrebujete uklada?
- Mali by by údaje pre vau aplikáciu súkromné?

Mali by by údaje súkromné pre vau aplikáciu?

Pri ukladaní citlivých údajov - údajov, ktoré by nemali by prístupné zo iadnej inej aplikácie - pouite interné úloisko, preferencie alebo databázu. Interné úloisko má navye tú výhodu, e údaje sú pred pouívatemi skryté.

Prístup k súborom pecifickým pre aplikáciu

- Adresáre interného úloiska - Tieto adresáre obsahujú vyhradené miesto na ukladanie trvalých súborov a alie miesto na ukladanie údajov vyrovnávacej pamäte. Systém zabrauje iným aplikáciám v prístupe k týmto umiestneniam a v systéme Android 10 (úrove API 29) a vyích sú tieto umiestnenia ifrované. Vaka týmto vlastnostiam sú tieto umiestnenia vhodným miestom na ukladanie citlivých údajov, ku ktorým má prístup len samotná aplikácia.
- Adresáre externého úloiska - Tieto adresáre obsahujú vyhradené miesto na ukladanie trvalých súborov a alie miesto na ukladanie údajov vyrovnávacej pamäte. Hoci je moné, aby k týmto adresárom pristupovala aj iná aplikácia, ak má prísluné oprávnenia, súbory uloené v týchto adresároch sú urené len pre vau aplikáciu. Ak máte peciálny zámer vytvára súbory, ku ktorým by mali ma prístup aj iné aplikácie, vaa aplikácia by mala tieto súbory uklada namiesto toho do zdieanej asti externého úloiska.

Príklad

- `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`
- Ak chcete alej chráni súbory pecifické pre aplikáciu, pouite na ifrovanie týchto súborov v pokoji knini cu Security, ktorá je súasou balíka Android Jetpack. ifrovací kú je pecifický pre vau aplikáciu.

Prístup k mediálnym súborom zo zdieaného úloiska

- Mnohé aplikácie umoujú pouívateom prispieva a pristupova k médiám, ktoré sú k dispozícii na externom úlonom zväzku, aby poskytli bohatí pouívateský záitok.
- Rámec poskytuje optimalizovaný index do zbierok médií, nazývaný úloisko médií, ktorý umouje jednoduchie naítanie a aktualizáciu týchto mediálnych súborov.
- Aj po odintalovaní aplikácie zostávajú tieto súbory v zariadení pouívatea.

ContentResolver

Na interakciu s abstrakciou úloiska médií pouite objekt ContentResolver, ktorý naítate z kontextu aplikácie:

```
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         // Pouite stpec ID z projekcie na získanie
15         // URI reprezentujúci samotnú mediálnu poloku.
16     }
17 }
```

Mediálne súbory

Systém automaticky skenuje externý pamäový zväzok a pridáva mediálne súbory do nasledujúcich presne definovaných kolekcií:

- **Images** vrátane fotografií a snímok obrazovky, ktoré sú uloené v adresároch *DCIM/* a *Pictures/*. Systém pridá tieto súbory do tabuky *MediaStore.Images*.
- **Videos**, ktoré sú uloené v adresároch *DCIM/*, *Movies/* a *Pictures/*. Systém pridá tieto súbory do tabuky *MediaStore.Video*.
- **Audio files**, ktoré sú uloené v adresároch *Alarms/*, *Audiobooks/*, *Music/*, *Notifications/*, *Podcasts/*, *and Ringtones/*, ako aj zvukové zoznamy skladieb, ktoré sú v adresároch *Music/ or Movies/*. Systém pridá tieto súbory do tabuky *MediaStore.Audio*.
- **Downloaded files**, ktoré sú uloené v adresári *Download/*. V zariadeniach so systémom Android 10 (úrove API 29) a vyou sú tieto súbory uloené v tabuke *MediaStore.Downloads*. Táto tabuka nie je k dispozícii v systéme Android 9 (úrove API 28)

Predvoby aplikácie

- Ak máte relatívne malú kolekciu hodnôt kúov, ktoré chcete uloi, mali by ste poui funkciu SharedPreferences
- Objekt SharedPreferences ukazuje na súbor obsahujúci páry kú-hodnota a poskytuje jednoduché metódy na ich ítanie a zápis
- Kadý súbor SharedPreferences spravuje rámec a môe by súkromný alebo zdieaný.

SharedPreferences

- Táto trieda poskytuje veobecný rámec, ktorý umouje uklada a získava perzistentné dvojice kú-hodnota primitívnych dátových typov.
- Na uloenie ubovoných primitívnych údajov môete poui funkciu SharedPreferences: booleans, floats, ints, longs, and strings.
- Tieto údaje pretrvajú naprie reláciami pouívatea (aj ke vaa aplikácia je ukonená).
- *"SharedPreferences"* sú uloené ako súbory XML v prieinku *shared_prefs*

Ako pouíva SharePreferences ?

- 1. Získanie predvolieb zo zadaného súboru

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

- 2. Preítajte 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. Zápis a aplikácia (alebo záväzok) zmien

```
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 }
```

Nastavenia

- Aplikácie asto obsahujú nastavenia, ktoré umoujú pouívateom upravova aplikácie funkcie a správanie.
- Nastavenia sú miestom v aplikácii, kde pouívatelia uvádzajú svoje preferencie. ako by sa mala vaa aplikácia správa.
- Nastavenia majú v pouívateskom rozhraní nízku dôleitos, pretoe nie sú asto potrebné.

Ako pouíva nastavenia

- 1. Vytvorte zdroj XML systému Android s názvom napr. *preferences.xml* typu PreferenceScreen.
- 2. Pridajte do súboru PreferencesCategory a jednu z týchto moností *CheckBoxPreference*, *ListPreference*, *EditTextPreference*
- 3. Vytvorenie triedy *MyPreferencesActivity*, ktorá roziruje *PreferenceActivity* Táto innos naíta *preference.xml* a umoní pouívateovi zmeni hodnoty.
- 4. Do metódy *onOptionsItemSelected()* pridajte kód na spustenie PreferencesActivity

RoomDatabase

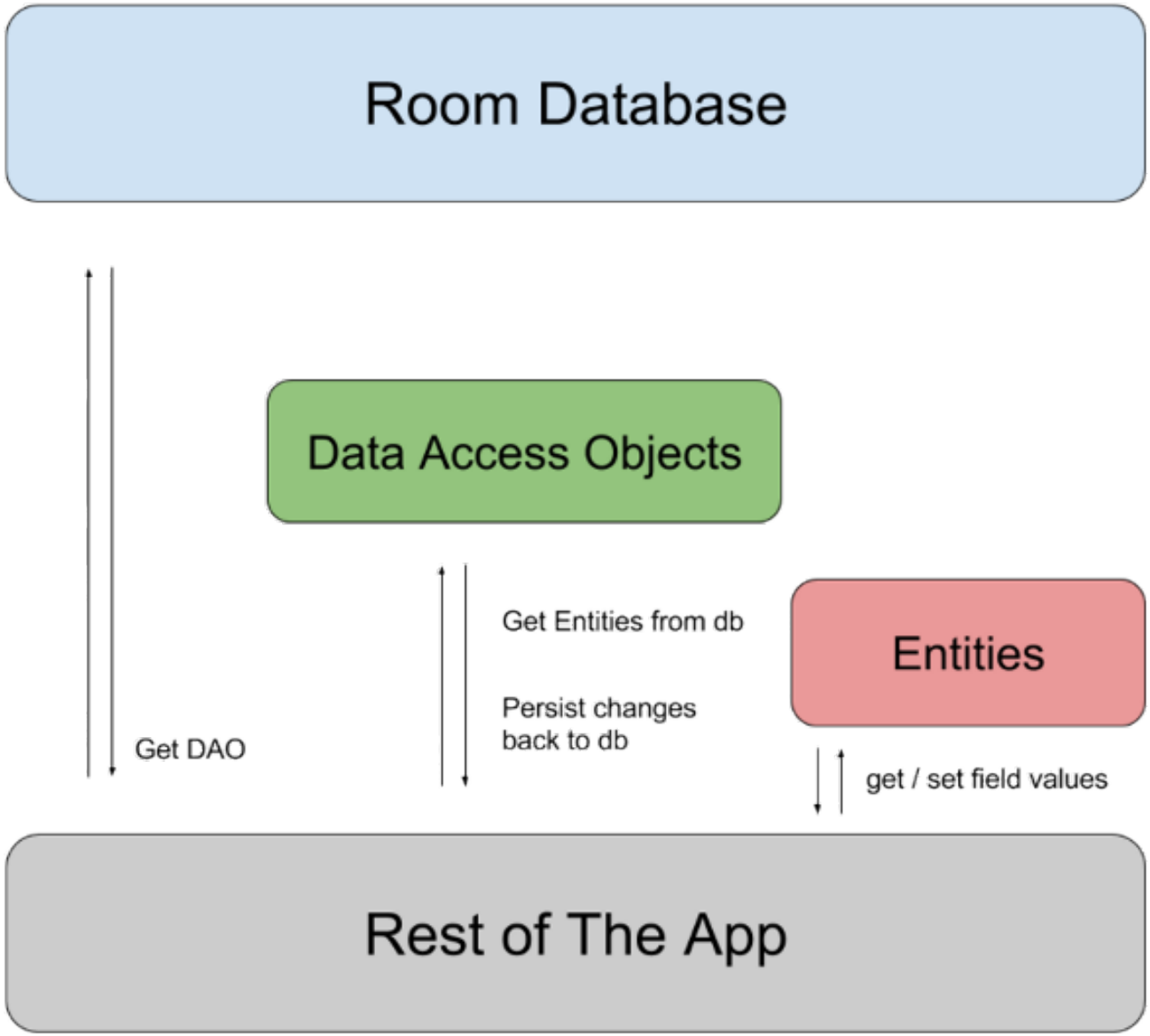
- Room poskytuje abstraknú vrstvu nad SQLite, ktorá umouje plynulý prístup k databáze a zárove vyuíva plný výkon SQLite.
- Aplikácie, ktoré spracúvajú netriviálne mnostvo truktúrovaných údajov, môu ma veký úitok z lokálneho uchovávania týchto údajov.
- Room sa stará o ukladanie dát do vyrovnávacej pamäte, ke je zariadenie off-line
- Táto budúcnos spôsobuje, e room sa odporúa pouíva namiesto SQLite

RoomDatabase

Hlavné komponenty v systéme Room

- Database - Obsahuje dritea databázy a slúi ako hlavný prístupový bod pre základné pripojenie k perzistovaným relaným údajom vaej aplikácie.
- Entity - Reprezentuje tabuku v rámci databázy
- DAO - Obsahuje metódy pouívané na prístup k databáze

Schéma architektúry Room



Príklad 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 )
```

Príklad 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 }
```

Príklad kódu: Database

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

Príklad naivo

Room Database Príklad