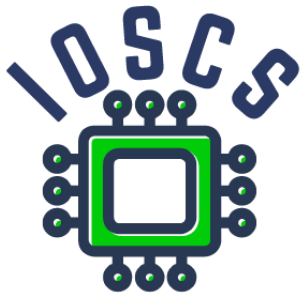


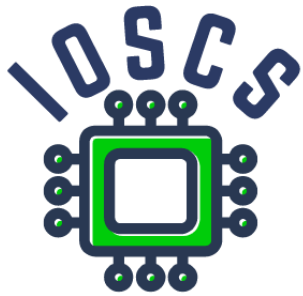
Project: Innovative Open Source Courses for Computer Science

Projektowanie aplikacji mobilnych
Prezentacje

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

Wprowadzenie

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Co to jest urządzenie mobilne

Co to jest urządzenie mobilne

Główne różnice ?

- PROCESOR
- Bateria

Co to jest urządzenie mobilne

Główne różnice ?

- PROCESOR
- Bateria
- Czujniki

Co to jest urządzenie mobilne

Główne różnice ?

- PROCESOR
- Bateria
- Czujniki
- Łączność

Czego oczekujesz od urządzeń mobilnych

- ??

Czego oczekujesz od urządzeń mobilnych

- ??
- Ile masz urządzeń?

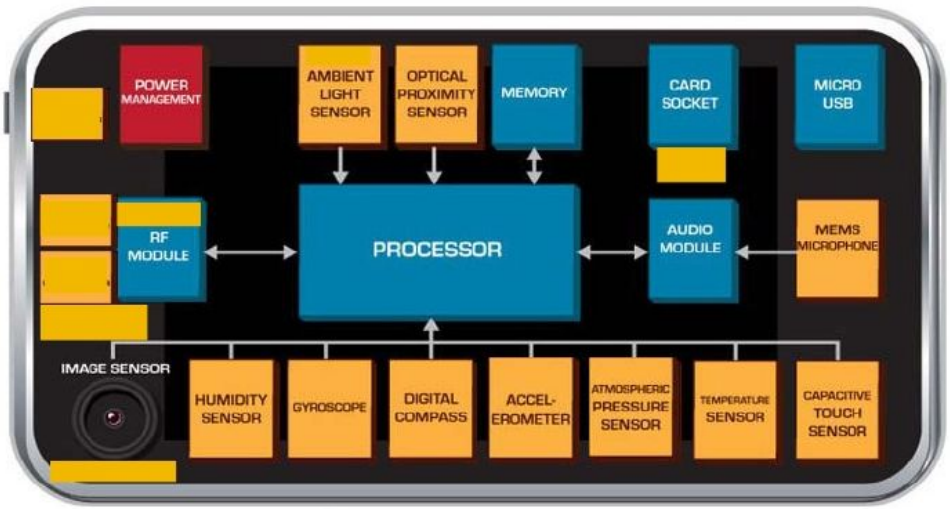
Czego oczekujesz od urządzeń mobilnych

- ??
- Ile masz urządzeń?
- W przyszłości - jedno urządzenie wiele aplikacji

Czego oczekujesz od urządzeń mobilnych

- ??
- Ile masz urządzeń?
- W przyszłości - jedno urządzenie wiele aplikacji
- ~~In future~~ nie TERAZ

Czym jest Urządzenie mobilne dla programistów



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

Główni producenci

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

Systemy operacyjne (mobilne)

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

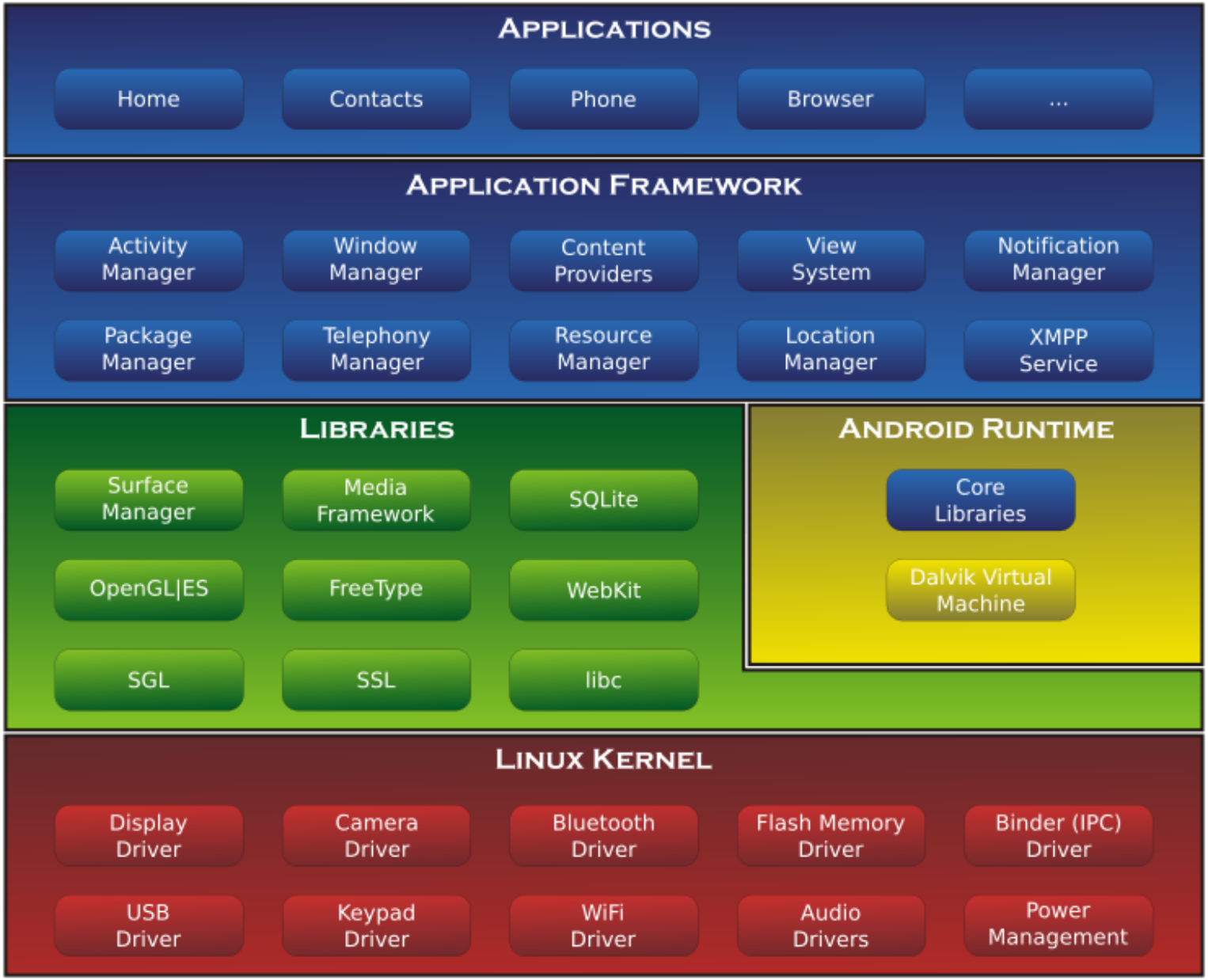
Co to jest Android?

- Stos oprogramowania open source, który obejmuje.
  - System operacyjny
  - Middleware
  - Kluczowe aplikacje mobilne (przeglądarka internetowa, PIM, SMS, Email...)
  - Biblioteki API do pisania aplikacji mobilnych
- Platforma deweloperska typu open-source do tworzenia aplikacji mobilnych aplikacji
- System operacyjny oparty na Linuksie
- Generalnie dla urządzeń mobilnych z ekranem dotykowym, takich jak smartfony i tablety

Gdzie jest Android

- Telefony
- Tablety
- Telewizory
- STB - set-top-box
- roboty
- będą w samochodach
- będą w systemach rozrywki pokładowej w samolotach
- Android jest wszędzie !

Stos Android Software



Podstawy aplikacji

- Aplikacje na Androida są pisane w języku programowania Java lub Kotlin. - Ogólnie
- Narzędzia SDK Androida kompilują kod (wraz z danymi i zasobami) do pakietu Android, pliku archiwum z rozszerzeniem .apk. i plikami zasobów) do pakietu Android, pliku archiwum z rozszerzeniem .apk.
- Cały kod w pojedynczym pliku .apk jest uważany za jedną aplikację i jest to plik, którego urządzenia z systemem Android używają do zainstalowania aplikacji.

Podstawy aplikacji

- Aplikacje na Androida są pisane w języku programowania Java lub Kotlin. - Ogólnie
- Narzędzia SDK Androida kompilują kod (wraz z danymi i zasobami) do pakietu Android, pliku archiwum z rozszerzeniem .apk. i plikami zasobów) do pakietu Android, pliku archiwum z rozszerzeniem .apk.
- Cały kod w pojedynczym pliku .apk jest uważany za jedną aplikację i jest to plik, którego urządzenia z systemem Android używają do zainstalowania aplikacji.

Podstawy aplikacji

- Aplikacje na Androida są pisane w języku programowania Java lub Kotlin. - Ogólnie
- Narzędzia SDK Androida kompilują kod (wraz z danymi i zasobami) do pakietu Android, pliku archiwum z rozszerzeniem .apk. i plikami zasobów) do pakietu Android, pliku archiwum z rozszerzeniem .apk.
- Cały kod w pojedynczym pliku .apk jest uważany za jedną aplikację i jest to plik, którego urządzenia z systemem Android używają do zainstalowania aplikacji.

Aplikacja Android

- Aplikacja Android żyje w swojej własnej "piaskownicy bezpieczeństwa",
- Andorid OS to wieloużytkownikowy system Linux,
- Każda aplikacja jest innym użytkownikiem, (domyślnie)
- Każdy proces ma swoją własną maszynę wirtualną (VM)
- Domyślnie, każda aplikacja działa w swoim własnym procesie Linux.
- Android uruchamia proces, gdy którykolwiek z komponentów aplikacji musi zostać wykonać
- Wyłącza proces, gdy nie jest on już potrzebny lub gdy system musi odzyskać pamięć dla innych aplikacji.

“Principle of least privilege”

- Każda aplikacja, domyślnie, ma dostęp tylko do tych komponentów, których wymaga do wykonania swojej pracy i nie więcej.
- Aplikacja nie może uzyskać dostępu do części systemu, do których nie ma nie otrzymała uprawnień
- Pytanie: Jak współdzielić dane z innymi aplikacjami ?

Współdzielenie danych z innymi aplikacjami

- Współdzielą ten sam identyfikator użytkownika systemu Linux, w którym to przypadku są w stanie uzyskać dostęp do plików drugiej osoby
- Tworzenie lub używanie Content Provider'a
- Przechowywanie danych na karcie SDCard
- Ważne: Aplikacja może poprosić o pozwolenie na dostęp do danych urządzenia, takich jak kontakty użytkownika, wiadomości SMS, pamięć masowa (karta SD), aparat fotograficzny, Bluetooth, i innych.

Komponenty aplikacji

Cztery różne typy komponentów aplikacji. Każdy typ służy odrębnemu celowi i ma odrębny cykl życia.

- Activities
- Services
- Content providers
- Broadcast receivers

Activity

- Reprezentuje pojedynczy ekran z interfejsem użytkownika
- Wiele aktywności w jednej aplikacji
- Wszystkie aktywności mają swój własny cykl życia
- Aktywność jest zaimplementowana jako podklasa Activity
- np. aplikacja Mail: czytanie poczty, komponowanie poczty...

Service

- Usługa działa w tle w celu wykonywania długo trwających operacji lub wykonywania pracy dla zdalnych procesów
- usługa nie udostępnia interfejsu użytkownika
- posiada własny cykl życia
- np. aplikacja muzyczna: usługa może odtwarzać muzykę w tle, podczas gdy użytkownik znajduje się w innej aplikacji
-

Content providers

- Content Provider zarządza współdzielonym zestawem danych aplikacji
- Inne aplikacje mogą modyfikować dane, bez znajomości szczegółowej architektury
- Dobrą praktyką jest również używanie dostawcy treści jako wewnętrzne API.
- Systemowe Content Providery przechowują informacje takie jak: Kontakt, Zdjęcia, Wideo...

Broadcast receivers

- Mechanizm do wysyłania lub odbierania wiadomości rozgłoszeniowych z systemu Android i innych aplikacji Android
- Podobny do wzorca projektowego "publish-subscribe"
- Kiedy wysyłasz wiadomość rozgłoszeniową inne aplikacje muszą subskrybować ten typ wiadomości
- Istnieje wiele wiadomości systemowych, np. system wysyła wiadomość po zakończeniu rozruchu, bateria jest na wyczerpaniu...

Intent

- Jest to osobny mechanizm, którego używamy do uruchamiania trzech z czterech typów komponentów - działań, usług i odbiorników transmisji.
- Z intencją wysłania informacji o akcji i danych
- W zależności od komponentu, akcje definiujemy w różny sposób

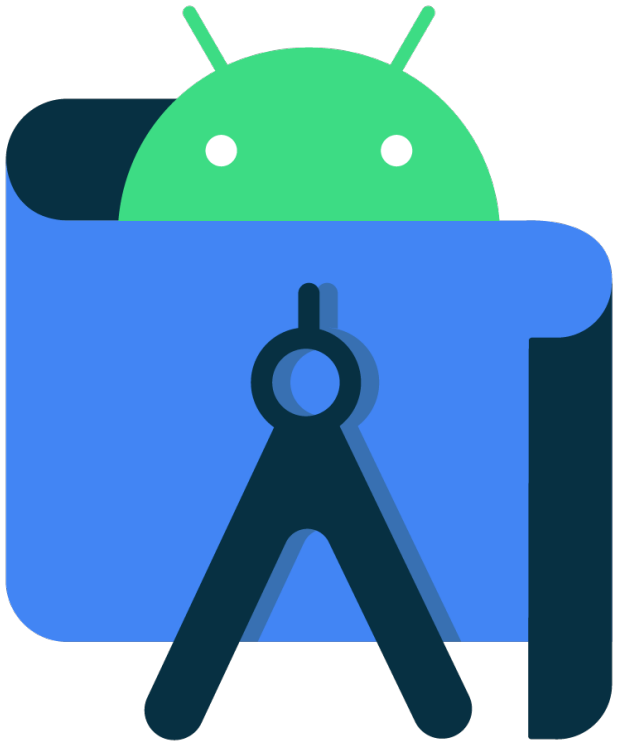
Manifest File

- Wszystkie informacje o firmie muszą istnieć w pliku *AndroidManifest.xml*.
- W szczególności musimy publikować informacje o głównej działalności
- Musimy również publikować informacje o zezwoleniu
- Aplikacja wymaga
- Zadeklarować minimalny poziom API
- Deklaracja używanych lub wymaganych funkcji sprzętowych i programowych (kamera, usługi bluetooth, ekran multitouch itp.)
- Biblioteki API (inne niż API frameworka Android), takie jak biblioteka Google Maps.

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest ... >
3  ^^I<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>
17 ^^I
```

Przykład na żywo - Hello World



Wprowadzenie do programu Android Studio

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

Rozwiązania typowych problemów związanych z tworzeniem aplikacji na Androida

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

MOBILE APPLICATION DEVELOPMENT

Cykl życia komponentów

Innovative Open Source courses for Computer Science

30.05.2021

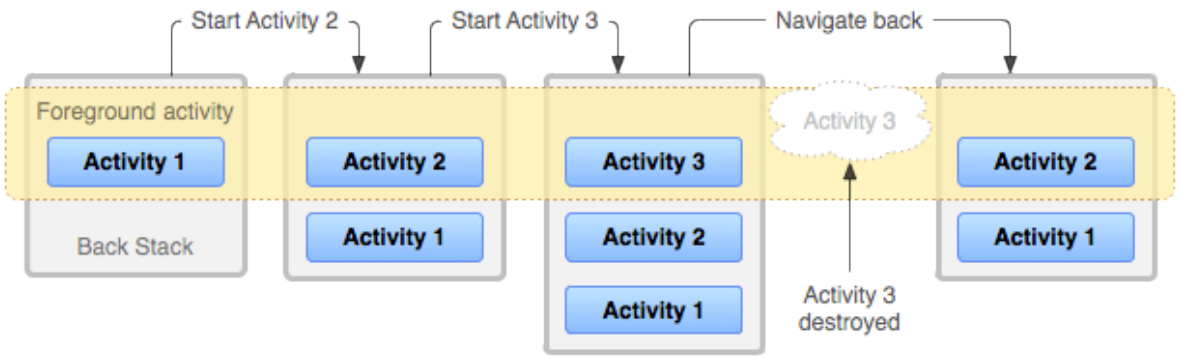
Funded by
the European Union

Activity

- Activity jest komponentem obsługującym ekran, definiuje interfejs użytkownika, z którym użytkownicy mogą wchodzić w interakcję
- Każda aktywność otrzymuje okno, w którym rysuje swój interfejs użytkownika.
- Okno wypełnia cały ekran, czasami może być mniejsze.
- Aplikacja może się składać z wielu aktywności, które są luźno powiązane ze sobą.

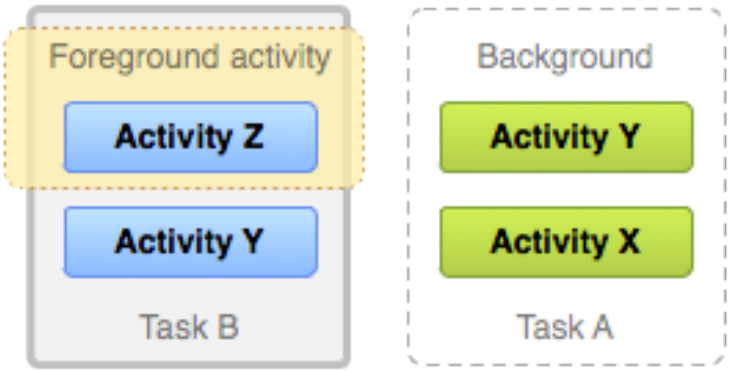
Wiele aktywności - jak zorganizować

- Każda aktywność może uruchomić inną aktywność
- Za każdym razem, gdy rozpoczyna się nowa aktywność, poprzednia aktywność jest zatrzymywana, a system zachowuje aktywność na stosie (tzw "back stack").
- Gdy rozpoczyna się nowa aktywność, zostaje ona odłożona na "back stack"
- "Back stack" jest mechanizmem stosu typu LIFO



Multiply Task - jak zorganizować

- Gdy użytkownik uruchamia aplikację po raz pierwszy lub gdy aplikacja zostanie zniszczona, tworzone jest nowe zadanie.
- Gdy aplikacja istnieje, zadanie tej aplikacji pojawia się na pierwszym planie.



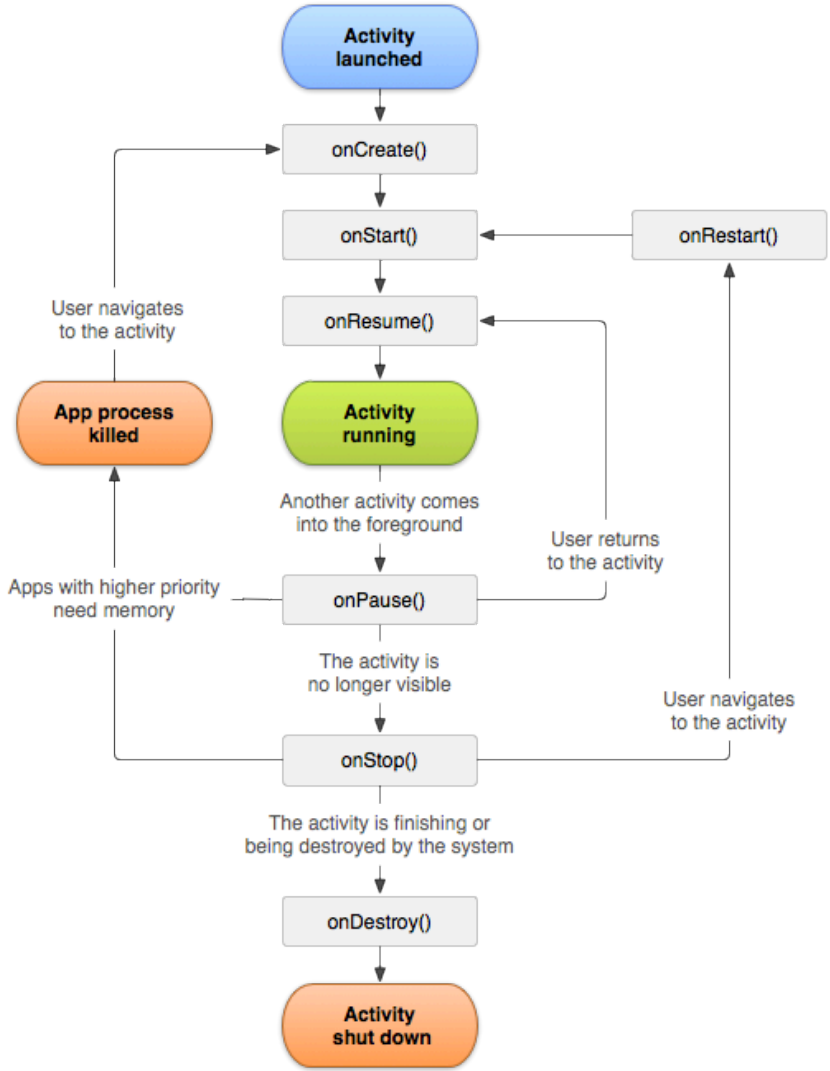
Tworzenie Aktywności

- Aby utworzyć aktywność, musisz utworzyć podklasę Activity (lub jej podklasę).
- Aktywność posiada siedem metod wywołania zwrotnego (Callback)
- Musimy zadeklarować tylko jedną metodę *onCreate()*
- Inne zależą od wymagań aplikacji
- Przewidywalność i poprawne działanie aktywności zależy od prawidłowego zrozumienia cyklu życia aktywności

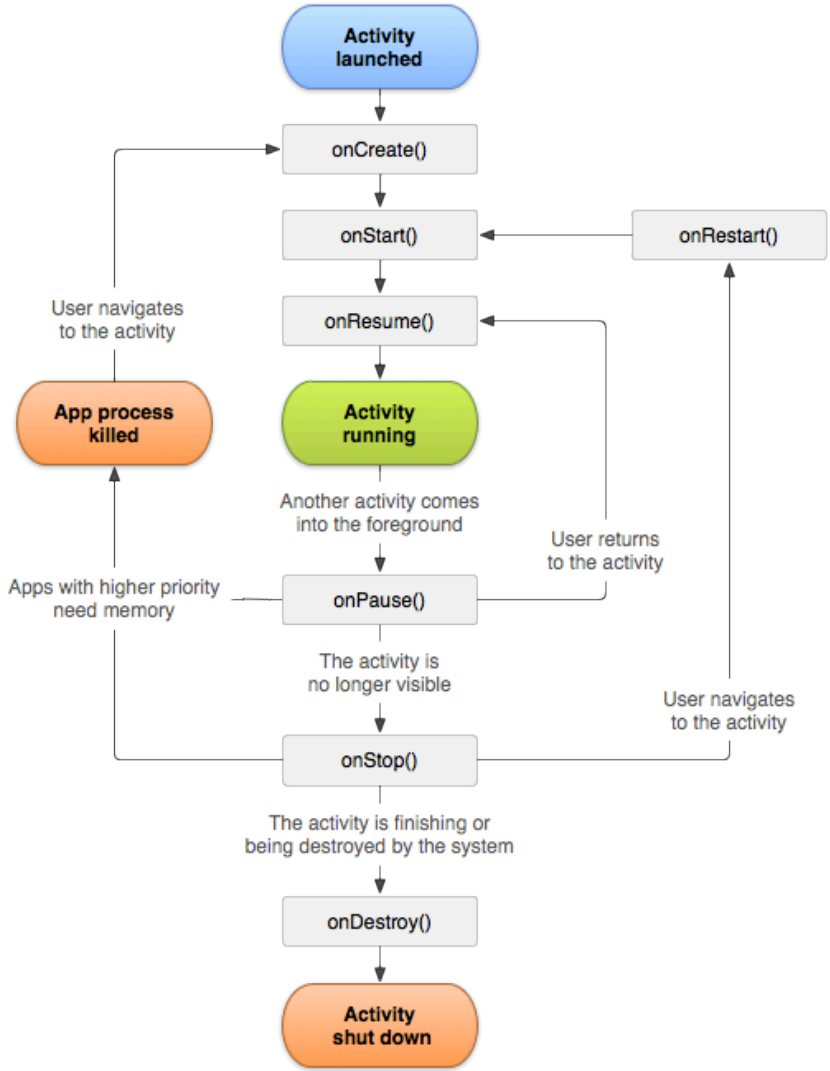
Implementacja cyklu życia - szkielet

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

Activity - Cykl życia



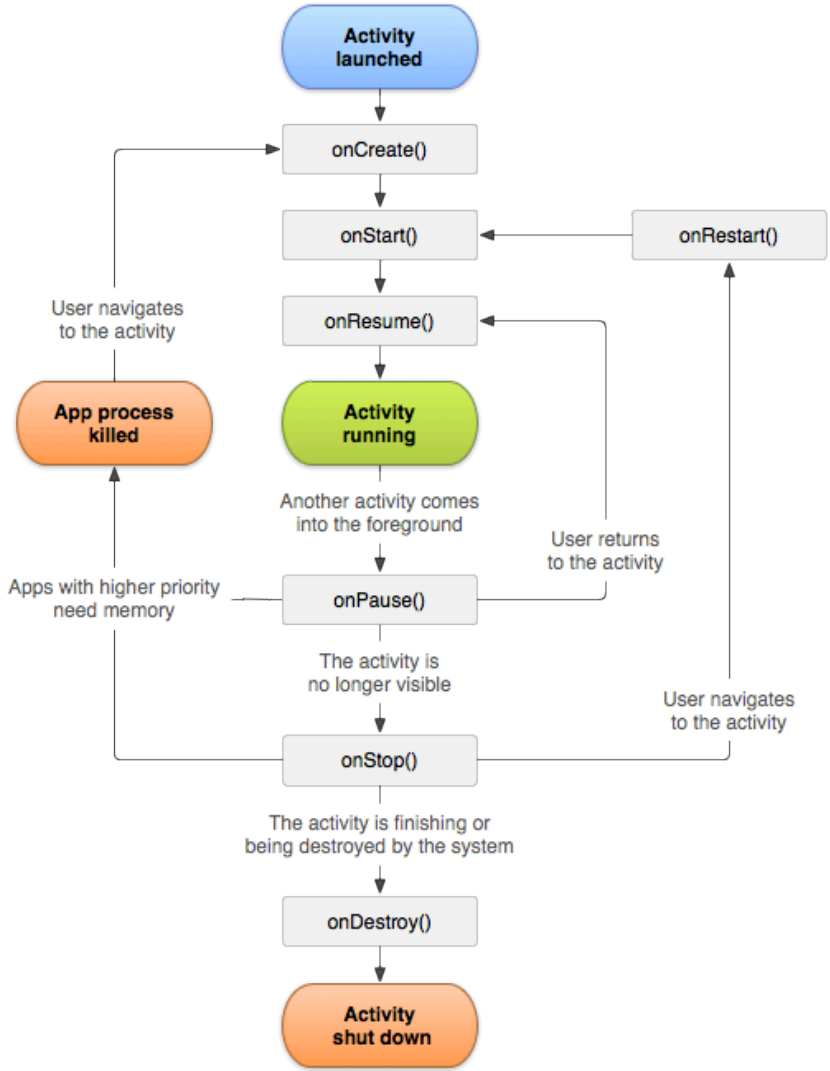
Activity - Cykl życia



onCreate()

Wywoływana przy pierwszym utworzeniu aktywności. Musimy ją zaimplementować, tworzone są widoki, wiążemy dane z listami itd. Metodzie tej przekazywany jest obiekt Bundle zawierający poprzedni stan aktywności stan. Zawsze po niej następuje *onStart()*.

Activity - Cykl życia



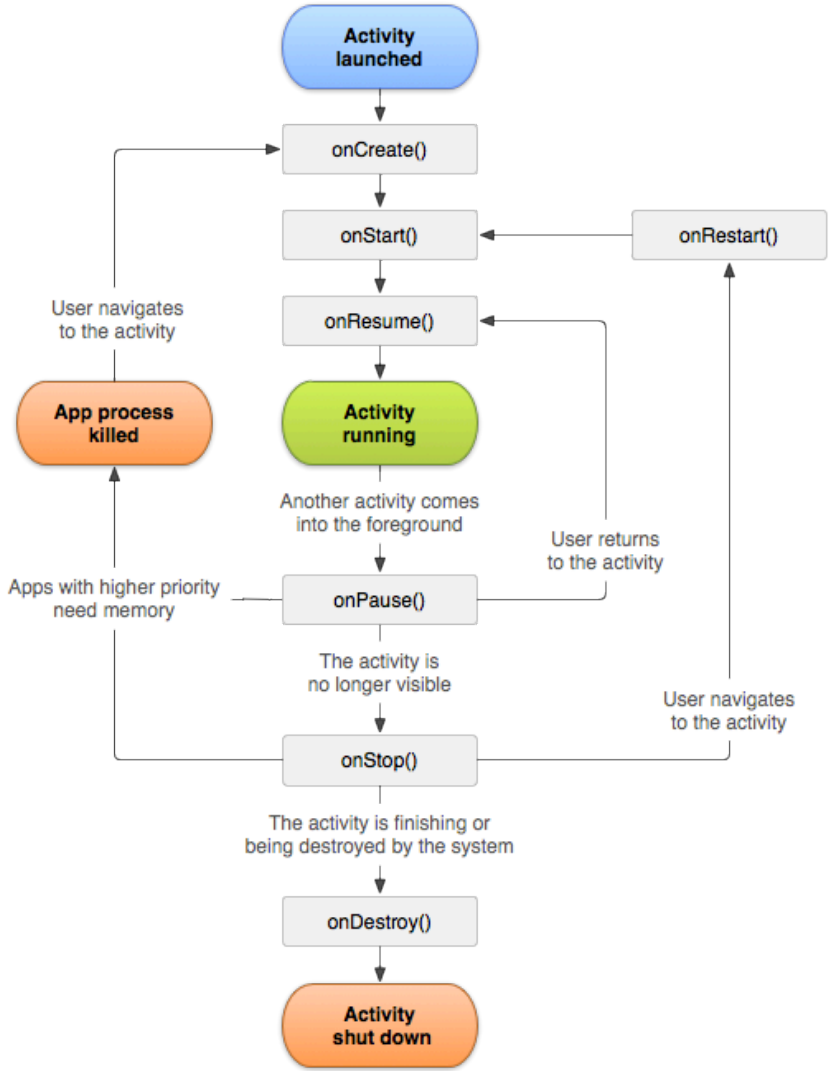
onRestart()

Wywoływane po zatrzymaniu aktywności, tuż przed jej ponownym uruchomieniem. Zawsze następuje po *onStop()*.

onStart()

Wywoływane tuż przed tym, jak aktywność stanie się widoczna dla użytkownika.
Następne jest *onResume()*

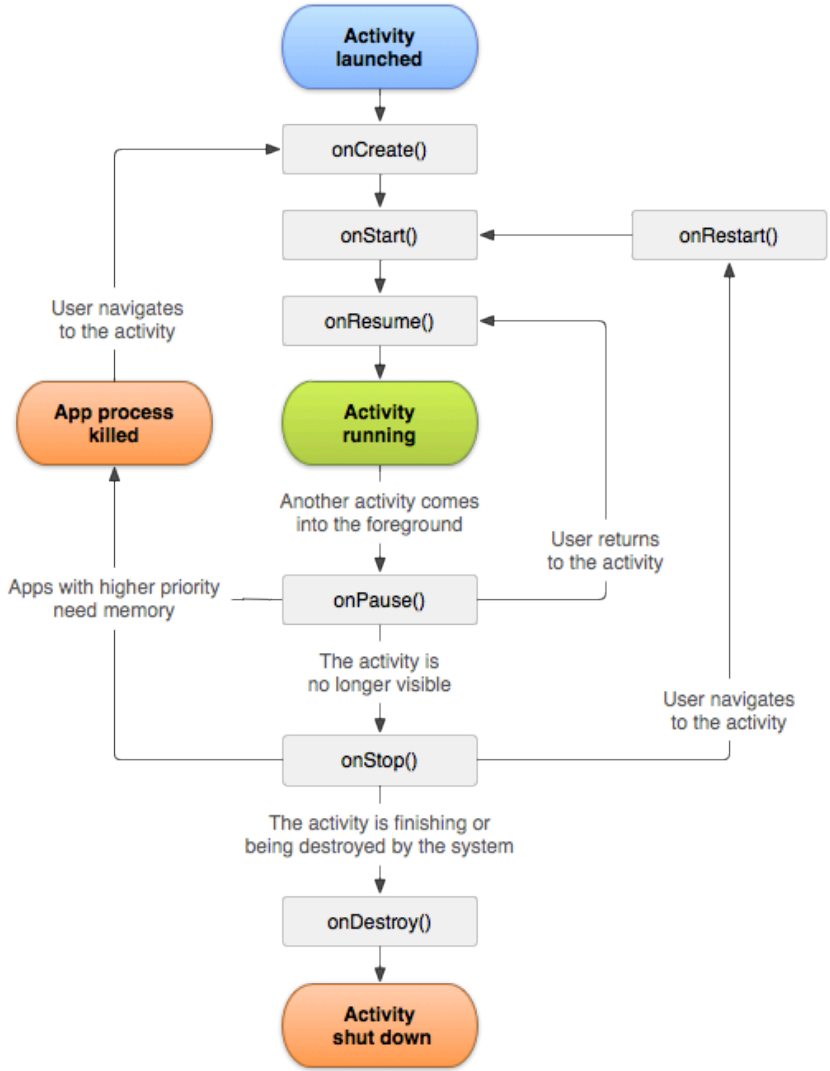
Activity - Cykl życia



onResume()

Wywoływane tuż przed rozpoczęciem interakcji aktywności z użytkownikiem. W tym momencie aktywność znajduje się na szczycie stosu aktywności, a dane wejściowe użytkownika trafia do niej.

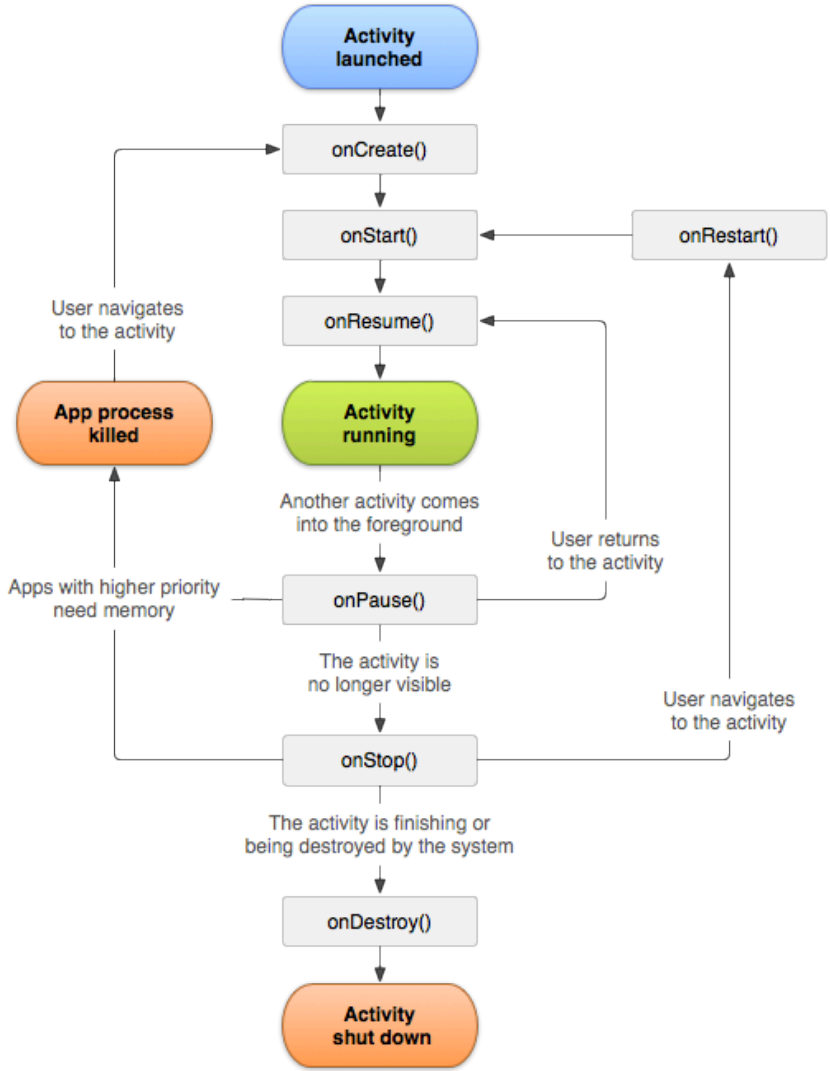
Activity - Cykl życia



onPause()

Wywoływane, gdy system ma zamiar rozpocząć wznawianie innej czynności. Powinno się: zatwierdzić niezapisane zmiany w trwałych danych, zatrzymać animacje i inne rzeczy, które mogą zużywać CPU, i tak dalej. Powinno się zrobić wszystko bardzo szybko, następna aktywność nie zostanie nie zostanie wznowiona, dopóki metoda nie zakończy działania. Następne jest *onResume()* jeśli aktywność wraca z powrotem na pierwszy plan, lub *onStop()*, jeśli aktywność staje się niewidoczna

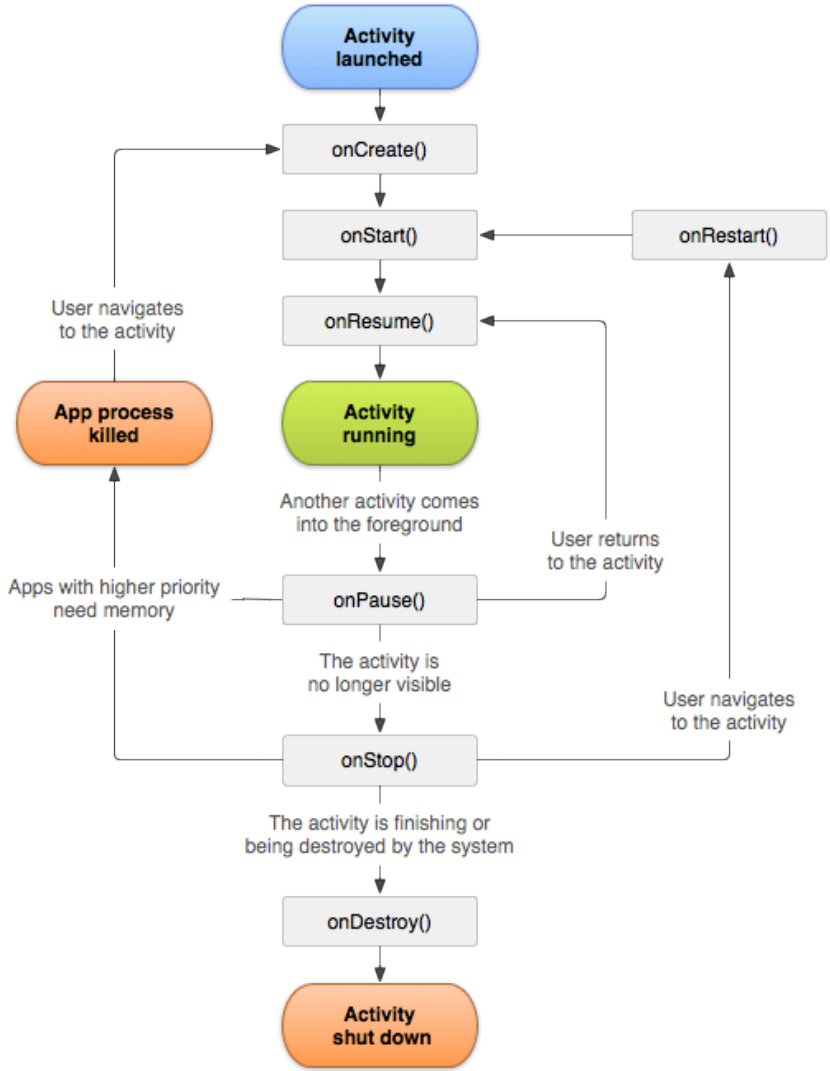
Activity - Cykl życia



onStop()

Wywoływane, gdy aktywność nie jest już widoczna dla użytkownika. Może to nastąpić ponieważ jest ona niszczona lub inna aktywność (istniejąca lub nowa) została wznowiona i ją zasłania.
Następną metodą jest *onRestart()*, jeśli aktywność powraca do interakcji z użytkownikiem, lub przez *onDestroy()*, jeżeli ta czynność kończy działanie.

Activity - Cykl życia



onDestroy()

Wywoływane przed zniszczeniem aktywności. Jest to ostatnia metoda.

Więcej informacji

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

Saving Activity State

- Domyślnie system używa stanu instancji Bundle do zapisania informacji o każdym obiekcie View w twoim układzie aktywności, ale nie zapisuje wszystkich informacji.
- Jednak można (i **powinno się**)proaktywnie zachować stan aktywności używając metody *onSaveInstanceState()*.
- Gdy twoja aktywność zaczyna się zatrzymywać, system wywołuje metodę onSaveInstanceState()
- Metody te używają par klucz-wartość do zapisania stanu.

```
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 }^^I
```

Przywracanie stanu aktywności

- Można użyć *onCreate()* lub *onRestoreInstanceState()*.
- Metody te otrzymują ten sam Bundle, który zawiera informacje o stanie instancji.

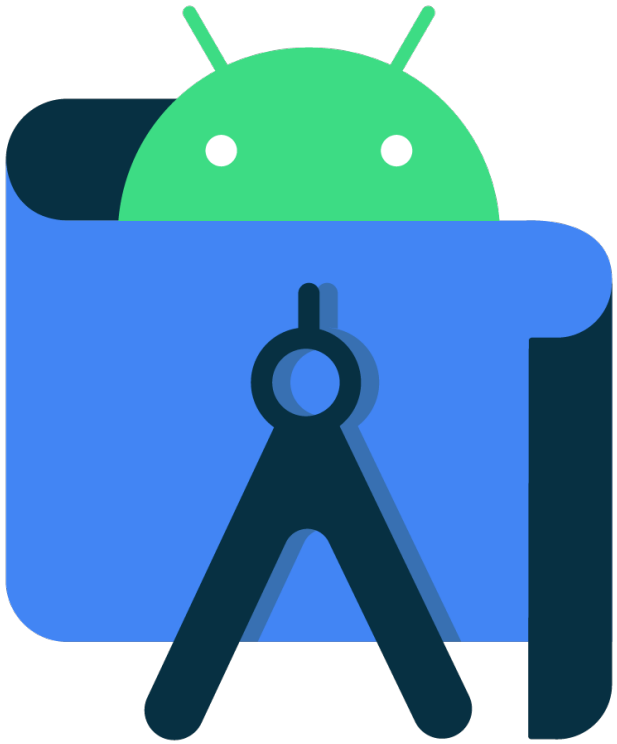
```
override fun onCreate(savedInstanceState: Bundle?){
    super.onCreate(savedInstanceState) // Always call the superclass first

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

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

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

Przykład - Android Cykl życia



Przejście między działaniami

- Aby rozpocząć aktywność możemy użyć dwóch metod *startActivity()* lub *startActivityForResult()*
- Metody te wymagają obiektu Intent, który zawiera informacje o aktywności

Explicit

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

Implicit

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

Rozpoczęcie aktywności i oczekiwanie na wynik [1/2]

- Gdy potrzebujemy otrzymać wynik musimy użyć *startActivityForResult()*
- Wdrażając te metody musimy dodać kod wewnątrz dwóch aktywności

First Activity - Fire second Activity

```
companion object{
        const val REQUEST_CODE = 67 //declare request code
    }
    fun activityCall(){
    val intent = Intent(this, OtherActivity::class.java)
    startActivityForResult(intent,REQUEST_CODE)
    }
^^I
```

Implement Receive methods

```
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?){
        super.onActivityResult(requestCode, resultCode, data)
        // Check which request we're responding to
        if (requestCode == REQUEST_CODE){
            // Make sure the request was successful
            if (resultCode == Activity.RESULT_OK){
                // Do something with the data here
            }
        }
    }
^^I
```

Uruchomienie aktywności i oczekiwanie na wynik [2/2]

Wyślij dane z drugiej aktywności

```
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     }^^I
```

Klasa Intent

Pozwala na:

- Uruchamianie aktywności
- Uruchamianie serwisu
- Użycie Braodcast Reciver-ów
- Intencja jawna - określa, która aplikacja spełni tę intencję
- Intencja niejawna - nie wymieniają nazwy konkretnego komponentu, lecz deklarują ogólną akcję do wykonania

Tworzenie intencji

- Tworząc obiekt *Intent* określamy
- **Component name** - tylko Jawna Intencja (Explicit Intent)
- **Action** - Ciąg znaków określający akcję generyczną do wykonania, akcję systemową lub własną
- **Data** - Obiekt URI
- **Category** - dodatkowe informacje o rodzaju komponentu, który powinien obsłużyć intencję
- **Extras** - pary klucz-wartość, które przenoszą dodatkowe informacje wymagane do wykonania żądanej akcji

Standard System Action

- 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

Przykłady z danymi

- ACTION_VIEW `content://contacts/people/1` – Wyświetl informacje o osobie, której identyfikator to “1”.
- ACTION_DIAL `content://contacts/people/1` – Wyświetlić telefon dialera z wpisaną osobą.
- ACTION_EDIT `content://contacts/people/1` – Edytuj informacje o osobie, której identyfikatorem jest “1”.
- ACTION_VIEW `tel:123` – Wyświetla dialer telefoniczny z podanym wypełnionym numerem. Zauważ, że akcja VIEW robi to, co jest uważane za najbardziej rozsądną rzecz dla danego URI.
- ACTION_DIAL `tel:123` – Wyświetla dialer telefoniczny z podanym wypełnionym numerem.

Category

- Podaje dodatkowe informacje na temat akcji do wykonania.
- Na przykład, CATEGORY_LAUNCHER oznacza, że powinna pojawić się w Launcherze jako aplikacja najwyższego poziomu.
- CATEGORY_ALTERNATIVE oznacza, że powinno być ono uwzględnione w liście alternatywnych działań, które użytkownik może wykonać na danym elemencie danych.
- Oznacza to, że jeśli uwzględni się kategorie CATEGORY_LAUNCHER i CATEGORY_ALTERNATIVE, wtedy będziesz rozwiązywał tylko komponenty z intencją która wymienia obie te kategorie.
- Działania bardzo często będą musiały wspierać CATEGORY_DEFAULT tak, że można je znaleźć przez Context.startActivity().
- DEFAULT category jest wymagany dla wszystkich filtrów - z wyjątkiem tych z MAIN action and LAUNCHER category.

Standard Categories

- 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

Przykład

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 }
13 ^^I
```

Wymuszanie wyboru aplikacji

- Using Implicit Intents user can select which app use (if more than one)
- User can seting default app for certain action
- Using *createChooser()* we show the chooser, and e.g. send data to other apps

```
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 }
```

Otrzymywanie niejawnych intencji

- Aby otrzymywać niejawne intencje musimy zadeklarować jeden lub więcej filtrów intencji dla każdego z komponentów aplikacji.
- System dostarcza niejawną intencję do twojego komponentu aplikacji tylko wtedy, gdy intencja może przejść przez jeden z twoich filtrów intencji.

```
<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"/>
```

Więcej informacji

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

MOBILE APPLICATION DEVELOPMENT

Fragment - Cykl życia

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Fragment

- Reprezentuje część UI aplikacji, którą można ponownie wykorzystać.
- Fragment definiuje i zarządza swoim własnym layoutem
- Posiada swój własny cykl życia
- Może obsługiwać własne zdarzenia wejściowe
- Fragment musi podrzędny w stustunku do inngeo framgentu lub innej aktywności

Tworzenie fragmentu

- Ustawienie środowiska, dodanie potrzenych bibliotek
- Stworzenie klasy fragment
- Dodanie stworzonej kalsy fragment do aktywności

Dodanie do projektu **build.gradle** informacji o repozytorium Google Maveny

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

Tworzenie fragmentu

- Ustawienie środowiska, dodanie potrzenych bibliotek
- Stworzenie klasy fragment
- Dodanie stworzonej kalsy fragment do aktywności

Dodaj do aplikacji **build.gradle** informacje o bibliotece Fragmenty AndroidX

```
dependencies{
    val fragment_version = "1.3.4"

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

Tworzenie fragmentu

- Ustawienie środowiska, dodanie potrzenych bibliotek
- Stworzenie klasy fragment
- Dodanie stworzonej kalsy fragment do aktywności

Możemy użyć **Fragemnt, DialogFragment, PreferenceFragmentCompat**

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

Tworzenie fragmentu

- Ustawienie środowiska, dodanie potrzenych bibliotek
- Stworzenie klasy fragment
- Dodanie stworzonej kalsy fragment do aktywności

Zdefiniuj w XML, *android:name* zawierający pojedynczą klasę

```
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" />
```

Tworzenie fragmentu

- Ustawienie środowiska, dodanie potrzenych bibliotek
- Stworzenie klasy fragment
- Dodanie stworzonej kalsy fragment do aktywności

lub (częściej) Zdefiniuj w XML kontener dla fragmentu

```
<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" />
```

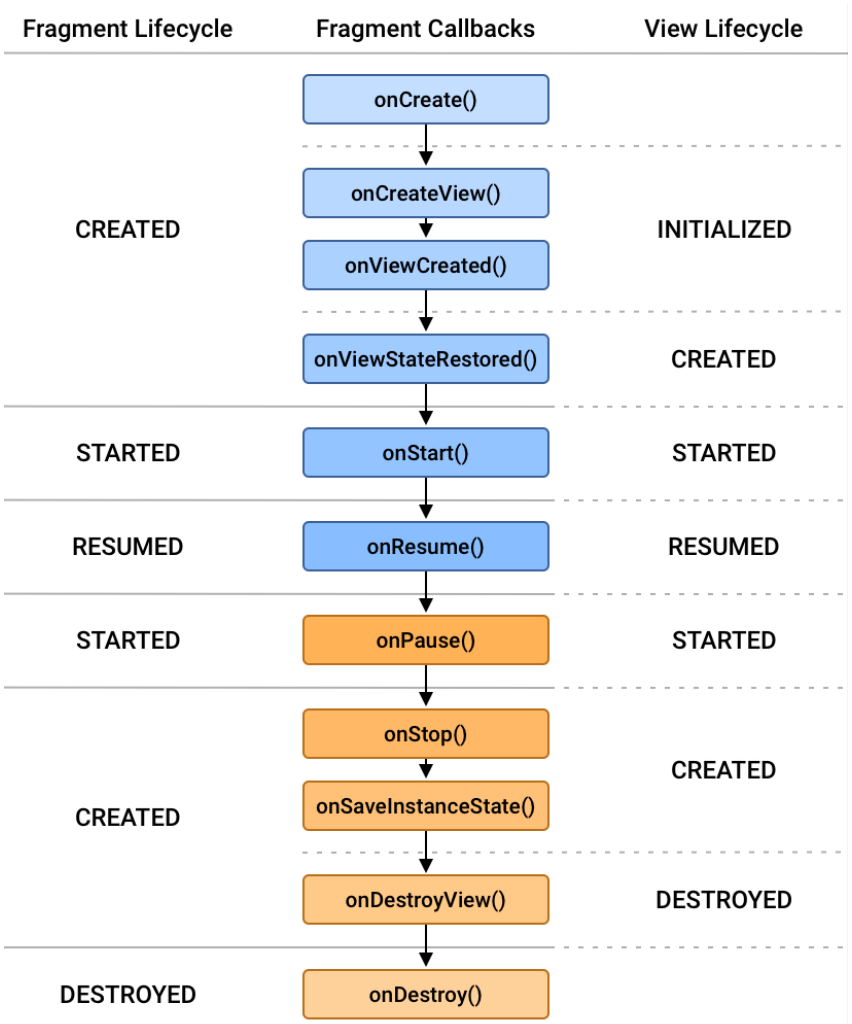
Dodaj kod do aktywności (onCreate())

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

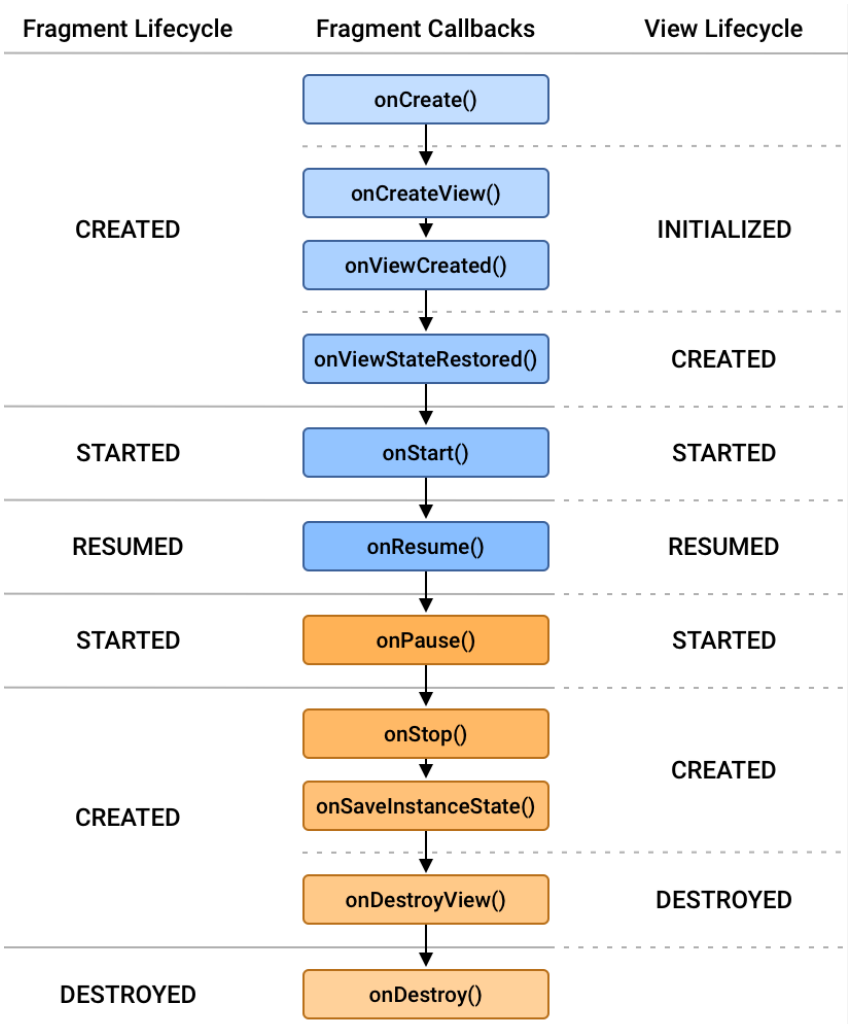
Fragment lifecycle

- Instancja **Fragment**-u ma swój własny cykl życia
- Cykl życia widoku jest inny niż cykl życia fragmentu
- Stan fragmentu:
  - INITIALIZED
  - CREATED
  - STARTED
  - RESUMED
  - DESTROYED

Fragment Lifecycle



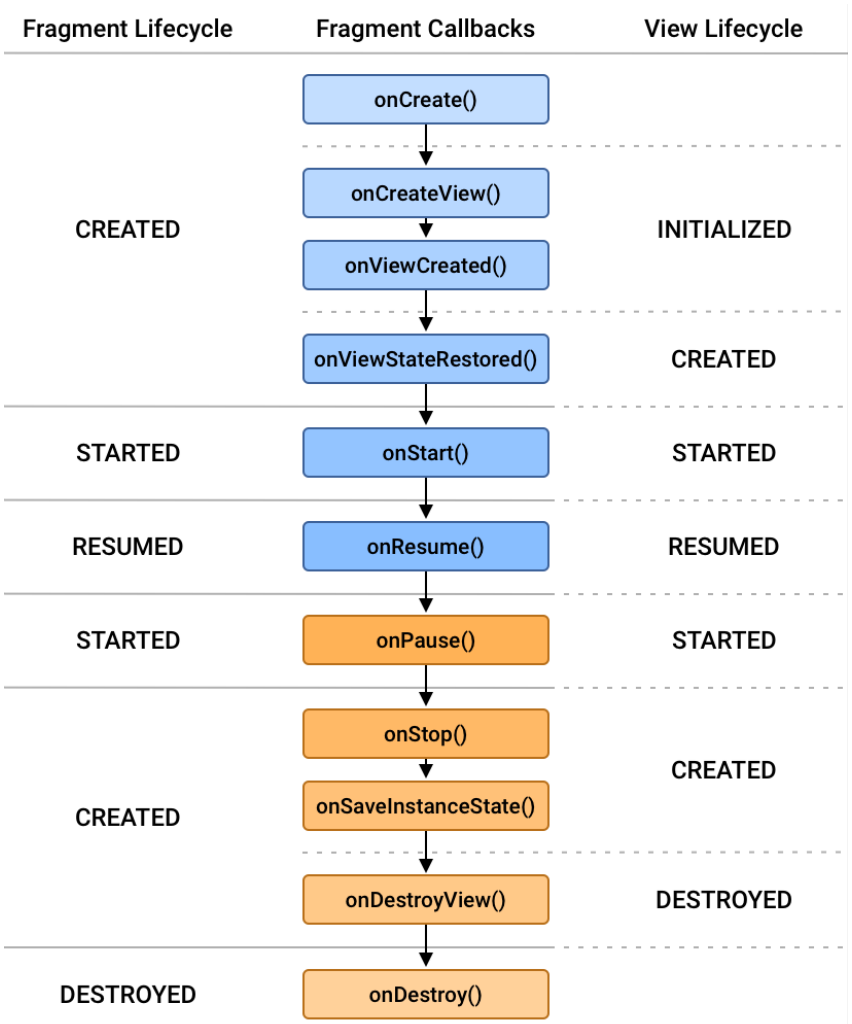
Fragment Lifecycle



CREATED

Został on dodany do FragmentManager i metoda onAttach() została już wywołana. Cykl życia widoku fragmentu jest tworzony tylko wtedy, gdy twój fragment dostarcza poprawną instancję widoku. Możesz również nadpisać onCreateView() aby programowo utworzyć widok fragmentu.

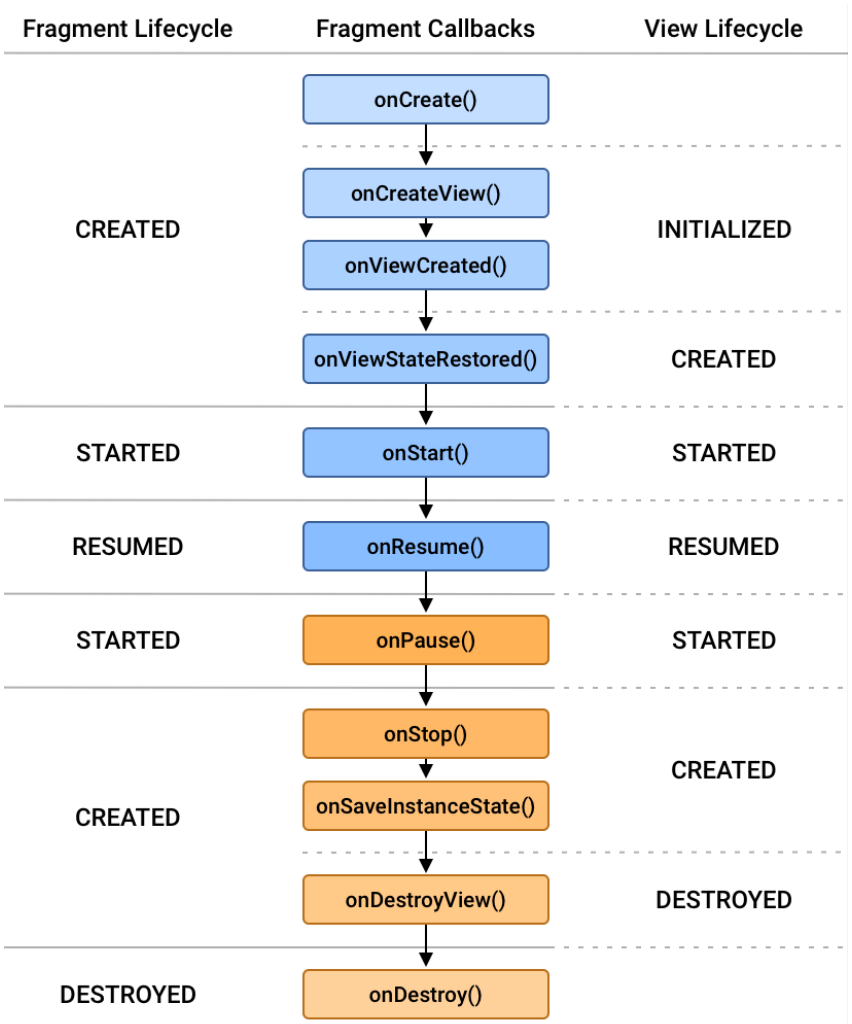
Fragment Lifecycle



STARTED

Ten stan gwarantuje, że widok fragmentu jest dostępny, jeśli został utworzony, i że można bezpiecznie wykonać FragmentTransaction na FragmentManager.

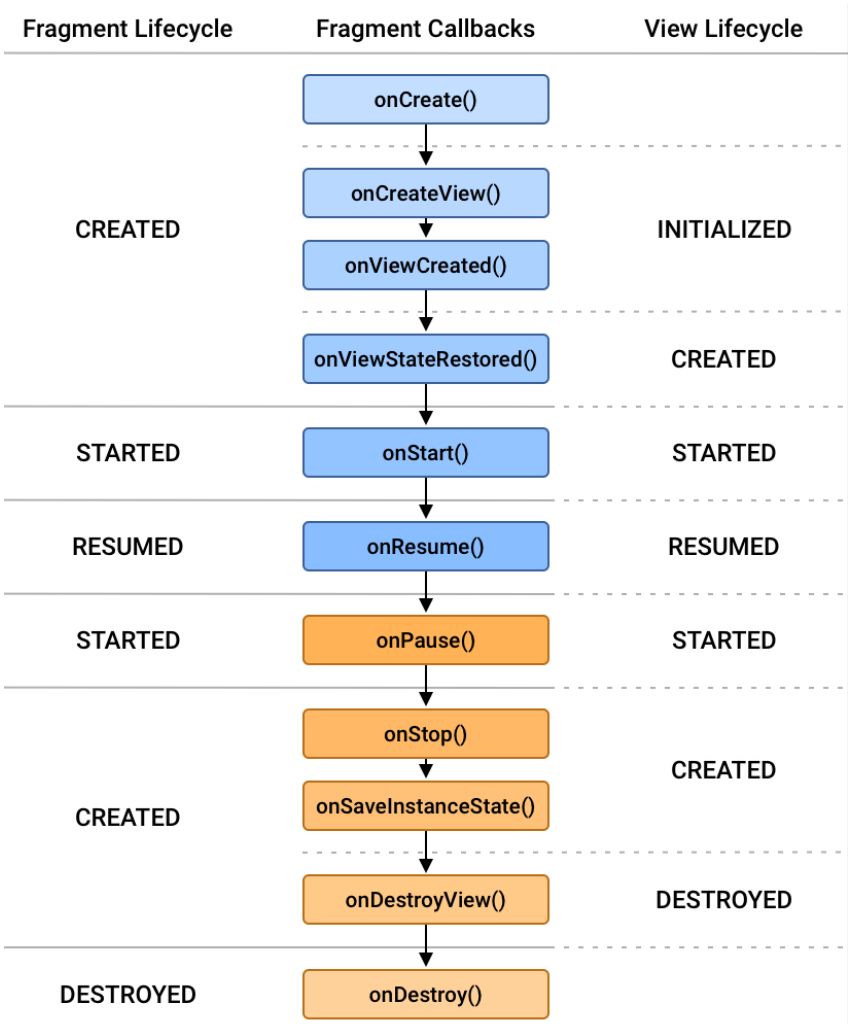
Fragment Lifecycle



RESUMED

Gdy fragment jest widoczny, wszystkie efekty (animator, przejścia) zostały zakończone, a fragment jest gotowy do interakcji z użytkownikiem.

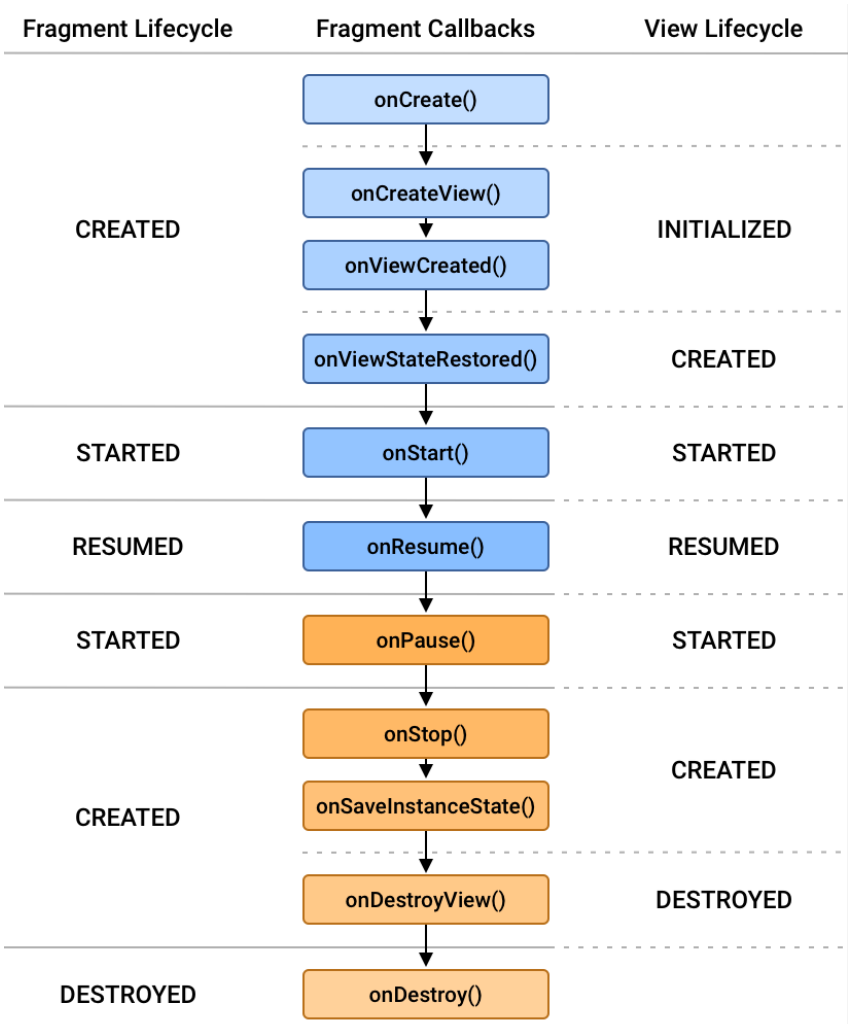
Fragment Lifecycle



STARTED

Gdy użytkownik zaczyna opuszczać fragment, a fragment jest nadal widoczny, cykle życia dla fragmentu i jego widoku są cofane do stanu STARTED

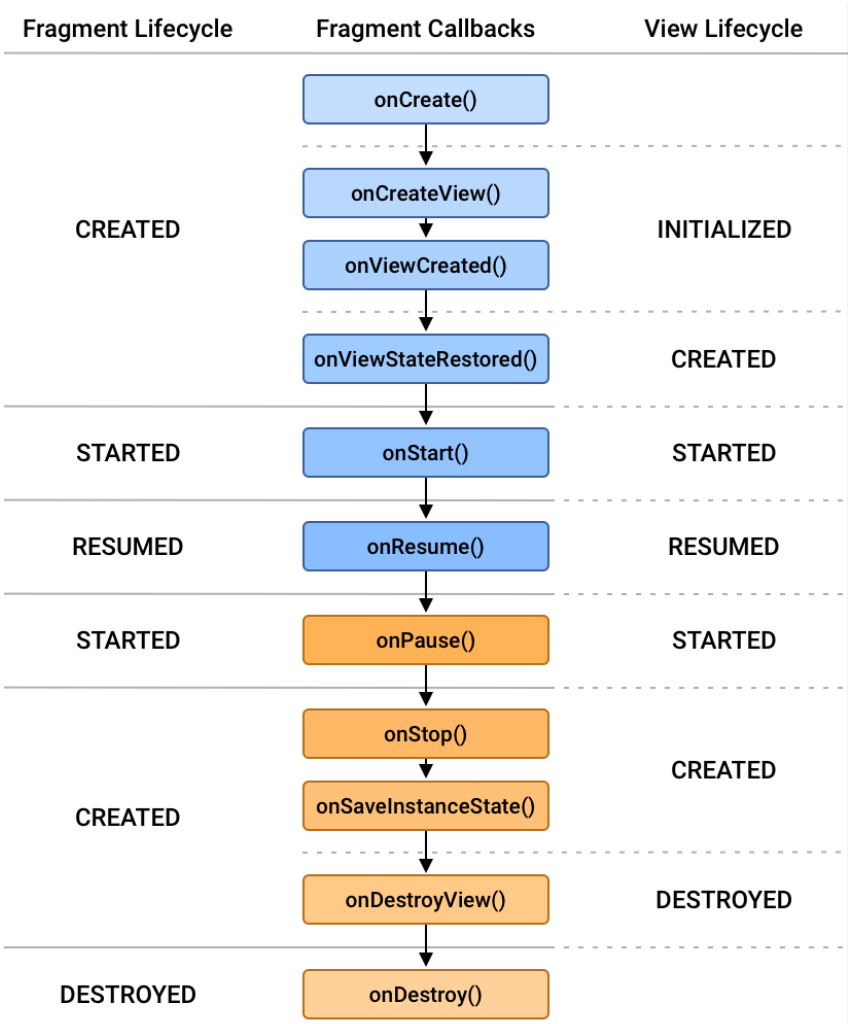
Fragment Lifecycle



CREATED

Gdy fragment nie jest już widoczny, cykle życia dla fragmentu i jego widoku są przenoszone do stanu CREATED Następuje albo przez *onResume()*, jeśli aktywność wraca na przód, albo przez *onStop()*, jeśli staje się niewidoczna dla użytkownika.

Fragment Lifecycle



DESTROYED

Fragment jest usuwany, lub jeśli FragmentManager jest niszczony.

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

- *onAttach* - wywoływana po skojarzeniu fragmentu z jego aktywnością.
- *onCreate* - wywoływana, aby wykonać wstępne tworzenie fragmentu.
- *onCreateView* - tworzy i zwraca hierarchię widoku powiązaną z danym fragmentem.
- *onActivityCreated* - informuje fragment, że jego aktywność została zakończona *android.app.Activity-onCreate*.
- *onViewStateRestored* - informuje fragment, że cały zapisany stan jego hierarchii widoku został przywrócony.
- *onStart* - powoduje, że fragment staje się widoczny dla użytkownika (na podstawie uruchomienia aktywności zawierającej fragment).
- *onResume* - powoduje, że fragment rozpoczyna interakcję z użytkownikiem (na podstawie wznowienia zawierającej go aktywności).

- *onPause* - fragment nie wchodzi już w interakcję z użytkownikiem, ponieważ jego aktywność jest wstrzymywana lub operacja fragmentu modyfikuje go w aktywności.
- *onStop* - fragment nie jest już widoczny dla użytkownika, ponieważ jego aktywność jest wstrzymana lub operacja fragmentu modyfikuje go w aktywności.
- *onDestroyView* - pozwala fragmentowi wyczyścić zasoby związane z jego widokiem.
- *onDestroy* - wywoływany w celu ostatecznego wyczyszczenia stanu fragmentu.
- *onDetach* - wywoływany bezpośrednio przed tym, jak fragment przestanie być związany ze swoją aktywnością.

Service

- Komponent aplikacji, który może wykonywać długo trwające operacje w tle.
- Nie dostarcza interfejsu użytkownika
- Rozszerzanie klasy Service
- Musisz zadeklarować wszystkie usługi w pliku manifest.xml swojej aplikacji.

Types of Services

- Foreground
- Background
- Bound

- Usługa pierwszoplanowa wykonuje jakąś operację, która jest zauważalna dla użytkownika.
- Usługi pierwszoplanowe muszą wyświetlać powiadomienie.
- Powiadomienie to nie może zostać odrzucone, chyba że usługa zostanie zatrzymana lub usunięta.
- Kontynuowanie działania nawet wtedy, gdy użytkownik nie wchodzi w interakcję z aplikacją

Types of Services

- Foreground
- Background
- Bound

- Usługa w tle wykonuje operację, która nie jest bezpośrednio zauważana przez użytkownika.
- np. kompaktowanie danych
- API 26 lub wyższe - ograniczenia dotyczące uruchamiania usług w tle, gdy sama aplikacja nie jest na pierwszym planie, nie należy uzyskiwać dostępu do informacji o lokalizacji z drugiego planu (background)

Types of Services

- Foreground
- Background
- Bound
- Usługa powiązana oferuje interfejs klient-serwer, który pozwala komponentom na interakcję z usługą, wysyłanie żądań, odbieranie wyników, a nawet robienie tego pomiędzy procesami za pomocą komunikacji międzyprocesowej (IPC).
- Działa tylko tak długo, jak długo inny komponent aplikacji jest z nią powiązany.
- Wiele komponentów może jednocześnie wiązać się z usługą, ale gdy wszystkie z nich się rozłączą, usługa jest niszczona.

More information

- 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

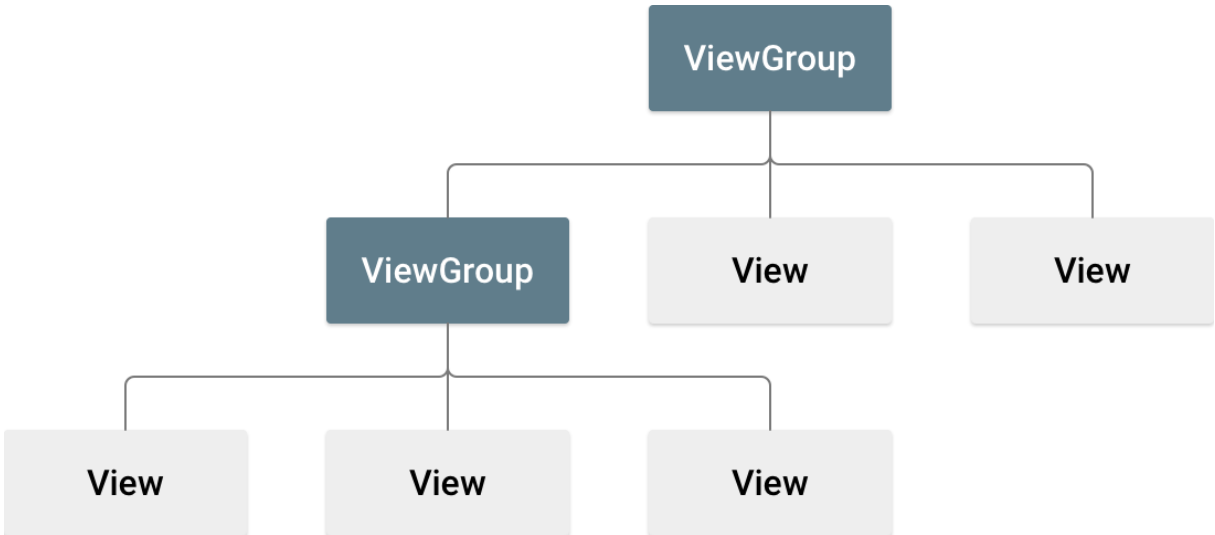
User Interface

UI

Interfejs użytkownika Twojej aplikacji to wszystko, co użytkownik może zobaczyć i z czym może wejść w interakcję. Android dostarcza wiele gotowych komponentów UI, takich jak **structured layout objects** i **UI controls**, które pozwalają na zbudowanie graficznego interfejsu użytkownika dla Twojej aplikacji. Android dostarcza również inne moduły UI dla specjalnych interfejsów, takie jak **dialogs**, **notifications** i **menus**.

Layouts

Określa strukturę interfejsu użytkownika w Twojej aplikacji, np. w aktywności. Wszystkie elementy w layoucie są budowane przy użyciu hierarchii obiektów View i ViewGroup.



Deklarowanie układów

Układ można zadeklarować na dwa sposoby:

- **Declare UI elements in XML.** - Android udostępnia proste słownictwo XML, które odpowiada klasom View i podklasom, takim jak te dla widżetów i układów. Możesz również użyć edytora układów Android Studio do zbudowania układu XML za pomocą mechanizmu "przeciągnij i upuść".
- **Instantiate layout elements at runtime.** Twoja aplikacja może tworzyć obiekty View i ViewGroup (i manipulować ich właściwościami) programowo.

Definiowanie układów w*XML*

- Deklarowanie UI w ***XML*** pozwala oddzielić prezentację aplikacji od kodu sterującego jej zachowaniem.
- ***View*** - zwykle nazywane "widżetami" i mogą być jedną z wielu podklas, takich jak ***Button or TextView***
- ***ViewGroup*** - zwykle nazywane "układami" mogą być jednym z wielu typów, które zapewniają inną strukturę układu, takich jak ***LinearLayout or ConstraintLayout***
- Do debugowania układu w czasie pracy należy użyć narzędzia Layout Inspector. `https://developer.android.com/studio/debug/layout-inspector`

Przykład 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>
```

- Każdy plik layoutu musi zawierać dokładnie jeden element root, który musi być obiektem View lub ViewGroup (np. LinearLayout)
- Wszystkie układy przechowujemy w ***res/layout/***
- Android wspiera różne rozmiary ekranów

Różne rozmiary ekranu

```
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
```

- Możesz zapewnić układy specyficzne dla ekranu, tworząc dodatkowe katalogi ***res/layout/*** - jeden dla każdej konfiguracji ekranu, która wymaga innego układu.
- Użyj kwalifikatora dostępnej szerokości (np. ***sw600dp*** - ekran 600dp)
- Użyj kwalifikatorów orientacji (np. ***land or port*** - układy odpowiednio pionowe i poziome)

Typ układu I

- Linear Layout - jest grupą widoków, która wyrównuje wszystkie dzieci w jednym kierunku, pionowo lub poziomo.
- ~~Relative Layout~~ - jest grupą widoków, która wyświetla widoki dzieci w pozycjach względnych.
- Constraint Layout - jest widokiem do tworzenia dużych i złożonych układów z płaską hierarchią widoków (bez zagnieżdżonych grup widoków). Jest podobny do RelativeLayout w tym, że wszystkie widoki są ułożone zgodnie z relacjami pomiędzy widokami rodzica i rodzeństwa, ale jest bardziej elastyczny niż RelativeLayout
- Table Layout - jest widokiem, który grupuje widoki w wiersze i kolumny.
- Absolute Layout - umożliwia określenie dokładnego położenia jego dzieci.

Typ układu II

- Frame Layout - jest miejscem na ekranie, które można wykorzystać do wyświetlenia pojedynczego widoku.
- List View - ListView jest grupą widoków, która wyświetla listę przewijanych elementów. (Układy z adapterem)
- Grid View - GridView jest grupą widoków, która wyświetla elementy w dwuwymiarowej, przewijanej siatce. (Układy z adapterem)
- `https://www.tutorialspoint.com/android/android_user_interface_layouts.htm`

Atrybuty układu - wspólne I

- *android* : *id* - Jest to identyfikator, który jednoznacznie identyfikuje widok.
- *android* : *layout_width* - Jest to szerokość layoutu.
- *android* : *layout_height* - Wysokość układu.
- *android* : *layout_marginTop* - Jest to dodatkowa przestrzeń po górnej stronie układu.
- *android* : *layout_marginBottom* - Jest to dodatkowa przestrzeń po dolnej stronie układu.
- *android* : *layout_marginLeft* - To jest dodatkowe miejsce po lewej stronie układu.
- *android* : *layout_marginRight* - To jest dodatkowe miejsce po prawej stronie układu.
- *android* : *layout_gravity* - Określa sposób pozycjonowania widoków podrzędnych.

Atrybuty układu - wspólne II

- *android* : *layout_weight* - Określa ile z dodatkowego miejsca w layoucie powinno być przeznaczone dla widoku.
- *android* : *layout_x* - Określa współrzędną x układu.
- *android* : *layout_y* - Określa współrzędną y układu.
- *android* : *layout_width* - Szerokość układu.
- *android* : *paddingLeft* - Wypełnienie lewego obramowania dla układu.
- *android* : *paddingRight* - Wypełnienie prawej strony układu.
- *android* : *paddingTop* - To jest wypełnienie górne dla układu.
- *android* : *paddingBottom* - To jest wypełnienie dolne dla układu.

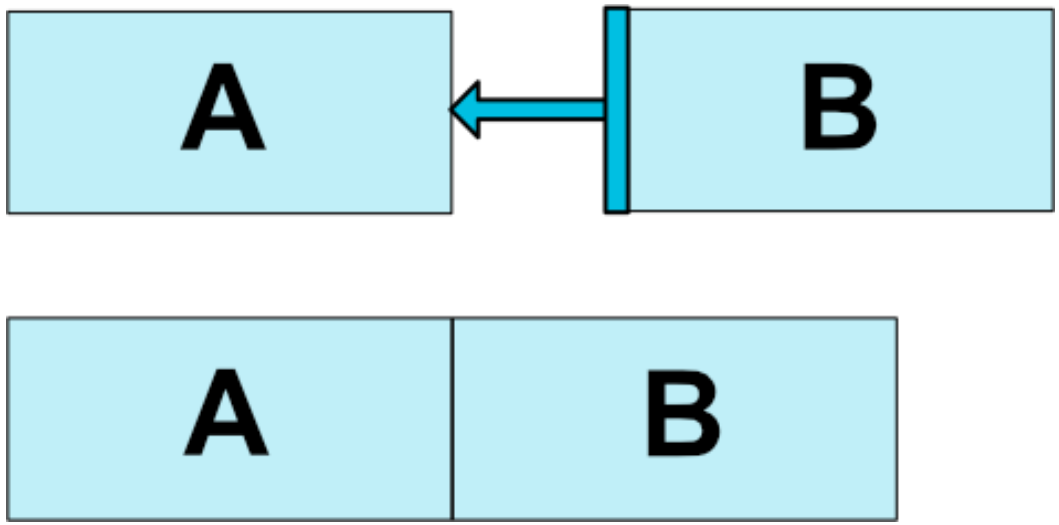
ConstraintLayout

- A ConstraintLayout jest ***android.view.ViewGroup***, który pozwala na elastyczne pozycjonowanie i określanie rozmiaru widżetów.
- Obecnie istnieją różne typy ograniczeń, których można używać:
  - Pozycjonowanie względne
  - Marginesy
  - Pozycjonowanie środkowe
  - Pozycjonowanie kołowe
  - Zachowanie widoczności
  - Ograniczenia wymiarów
  - Łańcuchy
  - Obiekty wirtualnych pomocników
- `https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout`

Pozycjonowanie względne

- Ograniczenia te pozwalają na pozycjonowanie danego widżetu względem innego.
- Możesz ograniczyć widżet na osi poziomej i pionowej:
  - Oś pozioma: lewa, prawa, strona początkowa i końcowa
  - Oś pionowa: góra, dół i linia bazowa tekstu
- Ogólna koncepcja polega na ograniczeniu danego boku widżetu do innego boku dowolnego innego widżetu.
- Wszystkie one pobierają identyfikator odniesienia do innego widżetu lub rodzica (który będzie odwoływał się do kontenera nadrzędnego, tj. ConstraintLayout).

Pozycjonowanie względne - przykład



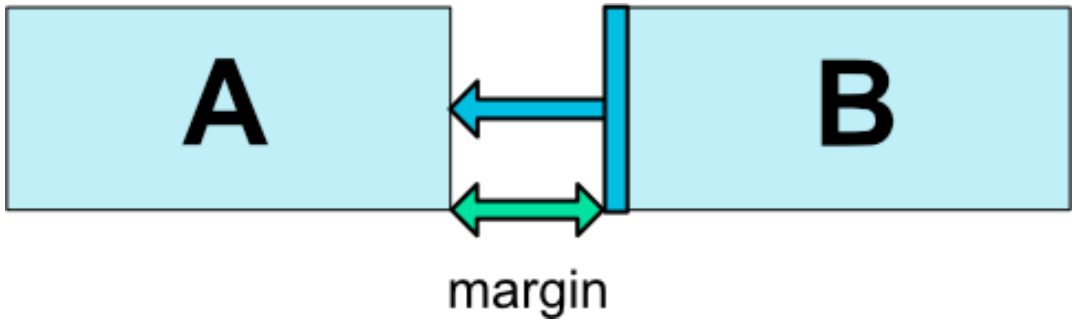

```
1 <Button android:id="@+id/buttonA" ... />
2          <Button android:id="@+id/buttonB" ...
3                  app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```

```
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

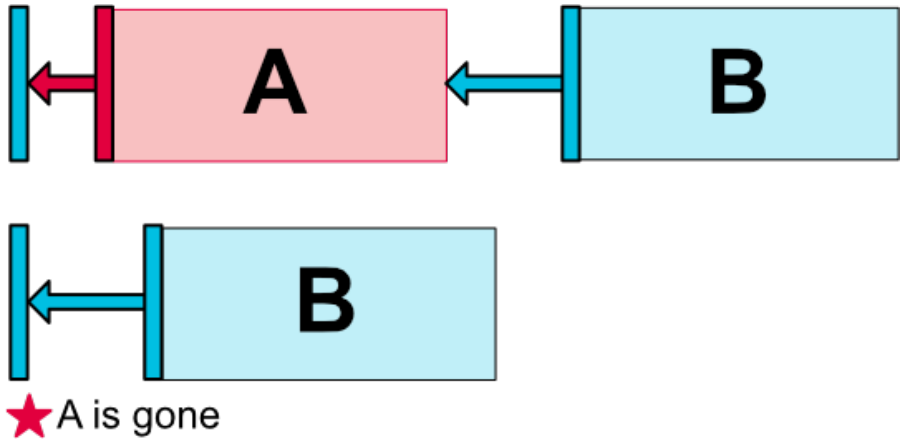
Marginesy



Jeśli marginesy boczne są ustawione, zostaną one zastosowane do odpowiednich ograniczeń, wymuszając margines jako przestrzeń pomiędzy stroną docelową a źródłową.

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

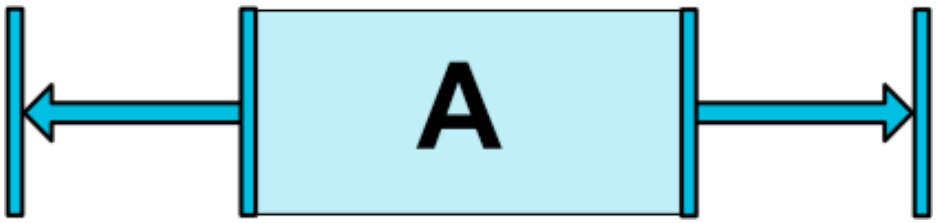
Zachowanie widoczności



ConstraintLayout ma specyficzną obsługę widżetów oznaczonych jako ***View.GONE***. **GONE** widżety, jak zwykle, nie będą wyświetlane i nie są częścią samego układu (tj. ich rzeczywiste wymiary nie zostaną zmienione, jeśli są oznaczone jako **GONE**). Ale pod względem obliczeń layoutu widżety GONE nadal są jego częścią, z ważnym rozróżnieniem:

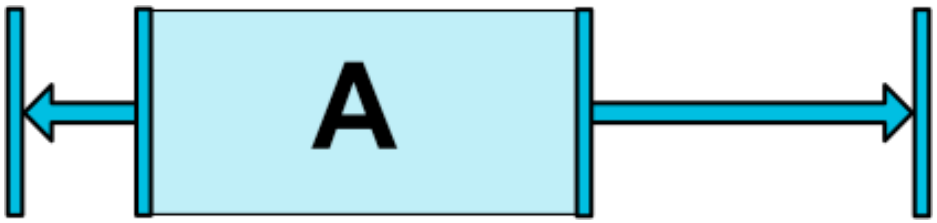
- Dla przebiegu layoutu ich wymiar będzie traktowany jako zerowy (w zasadzie zostaną one rozwiązane do punktu)
- Jeśli mają ograniczenia do innych widżetów, będą one nadal respektowane, ale wszelkie marginesy będą tak jakby równe zeru

Pozycjonowanie centralne



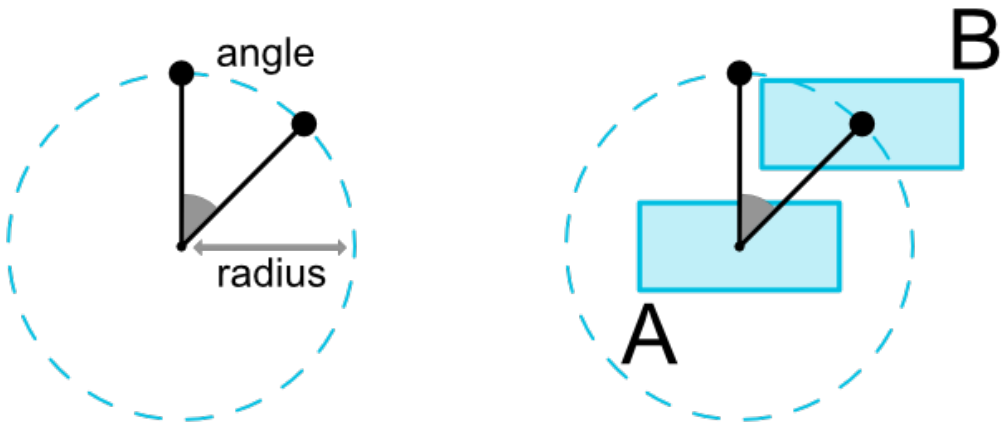

```
1 <androidx.constraintlayout.widget.ConstraintLayout ...>
2             <Button android:id="@+id/button" ...
3                 app:layout_constraintHorizontal_bias="0.3"
4                 app:layout_constraintLeft_toLeftOf="parent"
5                 app:layout_constraintRight_toRightOf="parent/>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

Pozycjonowanie centralne z ”wagą”




```
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
```

Pozycjonowanie kołowe




```
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" />
```

Mogą być użyte następujące atrybuty:

- layout_constraintCircle : odwołuje się do id innego widżetu
- layout_constraintCircleRadius : odległość do środka innego widżetu
- layout_constraintCircleAngle : pod jakim kątem powinien znajdować się widżet (w stopniach, od 0 do 360)

Ograniczenia wymiarowe

Możesz zdefiniować minimalne i maksymalne rozmiary dla samego ConstraintLayout:

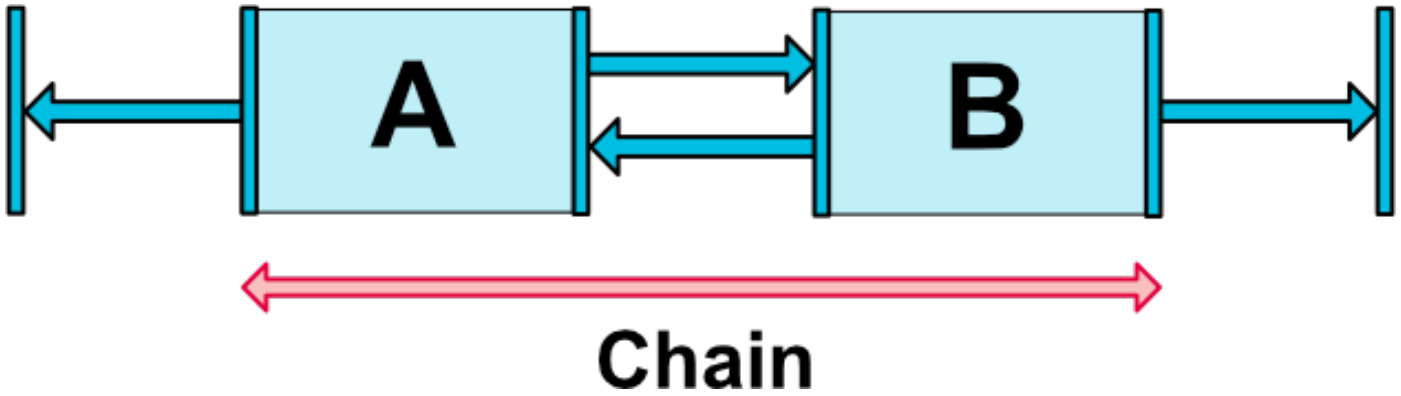
- android:minWidth ustawia minimalną szerokość dla layoutu.
- android:minHeight ustawia minimalną wysokość dla layoutu
- android:maxWidth ustawia maksymalną szerokość dla layoutu
- android:maxHeight ustawia maksymalną wysokość dla layoutu

Wymiar widżetów można określić poprzez ustawienie atrybutów android:layout_width i android:layout_height na 3 różne sposoby:

- Używając określonego wymiaru (albo literalnej wartości, takiej jak 123dp, albo referencji Dimension)
- Używając WRAP_CONTENT, który poprosi widżet o obliczenie jego własnego rozmiaru
- Użycie 0dp, co jest odpowiednikiem "MATCH_CONSTRAINT"

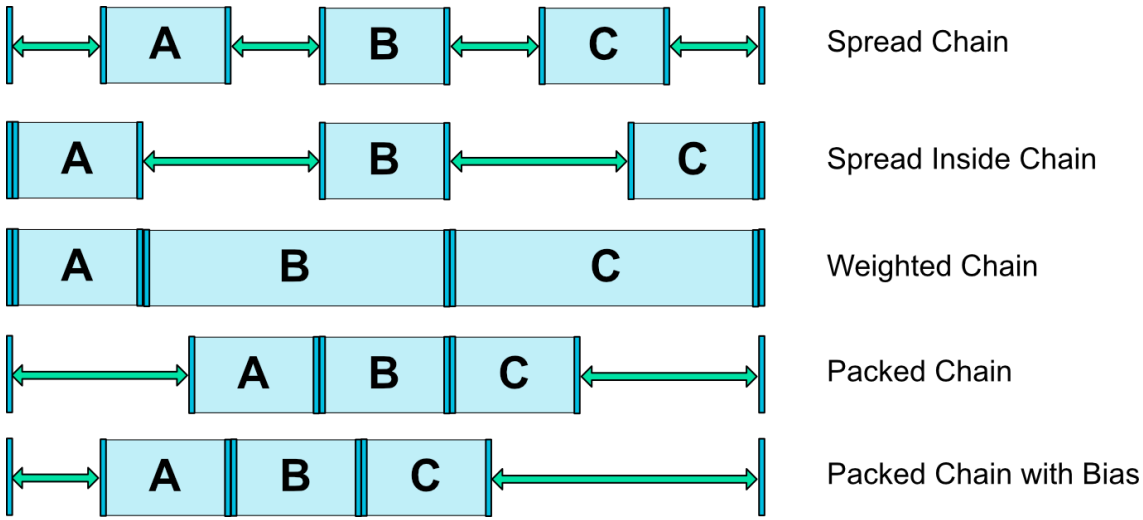
Łańcuchy

Łańcuchy zapewniają zachowanie podobne do grupy w jednej osi (poziomo lub pionowo). Pozostałe osie mogą być ograniczane niezależnie.



Zbiór widżetów uważamy za łańcuch, jeśli są one ze sobą połączone dwukierunkowym połączeniem (na rysunku, przedstawiającym minimalny łańcuch, z dwoma widżetami).

Styl łańcuchowy I



- CHAIN_SPREAD – elementy będą rozłożone (styl domyślny)
- Łańcuch ważony – w trybie CHAIN_SPREAD, jeśli niektóre widżety są ustawione na MATCH_CONSTRAINT, podzielą one dostępne miejsce.
- CHAIN_SPREAD_INSIDE – podobne, ale punkty końcowe łańcucha nie będą rozłożone.

Styl łańcuchowy II

- CHAIN_PACKED – elementy łańcucha będą upakowane razem. Poziomy lub pionowy atrybut "bias" elementu podrzędnego będzie miał wpływ na pozycjonowanie spakowanych elementów.

Więcej informacji

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

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

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

Tools

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

Material Design - Goal

- Stworzyć język wizualny, który syntetyzuje klasyczne zasady dobrego projektowania z innowacyjnością i możliwościami technologii i nauki.
- Opracowanie jednego systemu bazowego, który pozwala na ujednolicenie doświadczeń na różnych platformach i urządzeniach. Zasady mobilności są fundamentalne, ale dotyk, głos, mysz i klawiatura są pierwszorzędnymi metodami wprowadzania danych.
- Material to system projektowania stworzony przez Google, aby pomóc zespołom w tworzeniu wysokiej jakości cyfrowych doświadczeń dla systemów Android, iOS, Flutter i sieci.

Material Design - Komponenty

- App bars
- Bunner
- Card
- Floataing Button
- Data Tables
- Dialogs
- List, Image List
- Snackbars
- ToolTip
- ...

Material Theming

- Material Theming odnosi się do dostosowywania aplikacji Material Design, aby lepiej odzwierciedlała markę Twojego produktu.
- Możesz na nowo zdefiniować:
  - Kolor
  - Typografia
  - Kształt np. zmienić rozmiar lub narożniki przycisków.

MOBILE APPLICATION DEVELOPMENT

Sensors

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Czujniki w urządzeniach mobilnych

- Czujniki inercyjne: Gyroscope, Accelerometer, Magnetometer (e-Compass)
- Czujniki optyczne: Proximity, Ambient Light, RGB Color, Image Sensors (Front/Rear)
- Czujniki dotyku: Multi-Touch, Touchless Hover, Pressure Touch
- Czujniki środowiskowe: Temperature, Humidity, Barometric Pressure, Gas (CO...)*
- Czujniki bezprzewodowe/RF:GPS, WiFi, Cellular A-GPS, Bluetooth Low Energy, NFC
- Inne czujniki:MEMS Microphones, Biometric/Fingerprint*, BioSensors*

MEMS Sensor

* - Czujniki przyszłościowe

Sensory

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

Przykładowe aplikacje z użyciem pojedynczego czujnika

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

Przykładowe aplikacje z użyciem fuzji danych z czujników

- 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(WSZYSTKIE CZUJNIKI !!)
- ...

Android - sensory

Platforma Android obsługuje trzy szerokie kategorie czujników:

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

Android - sensory

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 - sensory

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

- Określenie, które czujniki są dostępne w urządzeniu.
- Określanie możliwości poszczególnych czujników, takich jak ich maksymalny zasięg, producent, wymagania dotyczące zasilania i rozdzielczość.
- Pozyskiwanie nieprzetworzonych danych z czujników i określenie minimalnej szybkości pozyskiwania danych z czujników.
- Rejestrowanie i wyrejestrowywanie listenerów od zdarzeń czujnika, które monitorują zmiany w czujniku.

Sensor Framework

- Android sensor framework - dostęp i pozyskiwanie surowych danych z czujników danych z czujników za pomocą frameworka czujników Androida.
- Szkielet sensora jest częścią pakietu *android.hardware* i zawiera następujące klasy i interfejsy zawiera następujące klasy i interfejsy:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorManager

Możemy użyć tej klasy do stworzenia instancji usługi czujnika. Ta klasa udostępnia różne metody dla dostępu i listowania czujników, rejestrowania i wyrejestrowywania słuchaczy zdarzeń czujników oraz pozyskiwania informacji o orientacji. Klasa ta dostarcza również kilka stałych czujnika, które są używane do raportowania dokładności czujnika, ustawiania szybkości zbierania danych i kalibrowania czujników

Sensor Framework

- Android sensor framework - dostęp i pozyskiwanie surowych danych z czujników danych z czujników za pomocą frameworka czujników Androida.
- Szkielet sensora jest częścią pakietu *android.hardware* i zawiera następujące klasy i interfejsy zawiera następujące klasy i interfejsy:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

Sensor

Możemy użyć tej klasy do stworzenia instancji konkretnego czujnika. Ta klasa udostępnia różne metody, które pozwalają określić możliwości czujnika.

Sensor Framework

- Android sensor framework - dostęp i pozyskiwanie surowych danych z czujników danych z czujników za pomocą frameworka czujników Androida.
- Szkielet sensora jest częścią pakietu *android.hardware* i zawiera następujące klasy i interfejsy zawiera następujące klasy i interfejsy:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorEvent

System wykorzystuje tę klasę do tworzenia obiektu zdarzenia czujnika, który dostarcza informacje o zdarzeniu czujnika. Obiekt zdarzenia czujnika zawiera następujące informacje informacje: surowe dane z czujnika, typ czujnika, który wygenerował zdarzenie, dokładność danych i znacznik czasu dla zdarzenia.

Sensor Framework

- Android sensor framework - dostęp i pozyskiwanie surowych danych z czujników danych z czujników za pomocą frameworka czujników Androida.
- Szkielet sensora jest częścią pakietu *android.hardware* i zawiera następujące klasy i interfejsy zawiera następujące klasy i interfejsy:
  - SensorManager
  - Sensor
  - SensorEvent
  - SensorEventListener

SensorEventListener

Możesz użyć tego interfejsu do stworzenia dwóch metod wywołania zwrotnego, które otrzymują powiadomienia (zdarzenia czujnika), gdy zmienią się wartości czujnika lub gdy zmieni się dokładność czujnika.

Typowe kroki

- Identyfikacja czujników i możliwości czujników
- Monitorowanie zdarzeń z czujników

Identyfikacja czujników

Identyfikacja czujników i możliwości czujników w czasie pracy jest przydatna, jeśli aplikacja posiada funkcje, które polegają na określonych typach czujników lub możliwościach. Na przykład, możesz chcieć zidentyfikować wszystkie czujniki, które są obecne na urządzeniu i wyłączyć wszystkie funkcje aplikacji, które opierają się na czujnikach, które nie są obecne. Podobnie, możesz chcieć zidentyfikować wszystkie czujniki danego typu, aby móc wybrać implementację czujnika, która ma optymalną wydajność dla Twojej aplikacji.

Typowe kroki

- Identyfikacja czujników i możliwości czujników
- Monitorowanie zdarzeń z czujników

Monitorowanie czujników

Monitorowanie zdarzeń czujnika to sposób pozyskiwania surowych danych z czujnika. Zdarzenie czujnika występuje za każdym razem, gdy czujnik wykryje zmianę w mierzonych przez siebie parametrach. Zdarzenie czujnika dostarcza czterech informacji: nazwę czujnika, który wywołał zdarzenie, znacznik czasowy zdarzenia, dokładność zdarzenia oraz surowe dane czujnika, który wywołał zdarzenie.

Identyfikacja czujników i weryfikacja ich możliwości

1. Uzyskaj odniesienie do usługi czujnika

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

2. Wyświetlanie listy wszystkich czujników na urządzeniu

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

2. Lub użyj innej stałej zamiast TYPE_ALL such as 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
```

Możliwości

```
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 }
```

Użyteczne metody do pobrania weryfikacji własności czujników

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

Monitorowanie zdarzeń z czujników

Aby monitorować surowe dane z czujników należy zaimplementować dwie metody wywołania zwrotnego, które są dostępne poprzez interfejs *SensorEventListener*.: onAccuracyChanged() and onSensorChanged()

Zmiany dokładności czujników

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

Czujnik zgłasza nową wartość

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

Monitorowanie zdarzeń z czujników

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

Monitorowanie zdarzeń czujników - rejestracja

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

Google Play filters

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

Najlepsze praktyki w zakresie uzyskiwania dostępu i korzystania z czujników

- Zbieranie danych z czujników tylko na pierwszym planie
- Wyrejestruj nasłuchiwaczy czujników
- Testuj za pomocą emulatora Androida
- Nie blokuj metody onSensorChanged()
- Unikaj używania przestarzałych metod lub typów czujników
- Weryfikuj czujniki zanim ich użyjesz
- Starannie dobieraj opóźnienia czujników

Pobieranie danych z czujników działających w tle

Na urządzeniach z systemem Android 9 (poziom API 28) lub nowszym:

- Czujniki, które używają trybu ciągłego raportowania, takie jak akcelerometry i żyroskopy, nie otrzymują zdarzeń.
- Czujniki, które używają trybów raportowania on-change lub one-shot nie otrzymują zdarzeń.

Najlepsze praktyki w zakresie uzyskiwania dostępu i korzystania z czujników

- Zbieranie danych z czujników tylko na pierwszym planie
- Wyrejestruj nasłuchiwaczy czujników
- Testuj za pomocą emulatora Androida
- Nie blokuj metody onSensorChanged()
- Unikaj używania przestarzałych metod lub typów czujników
- Weryfikuj czujniki zanim ich użyjesz
- Starannie dobieraj opóźnienia czujników

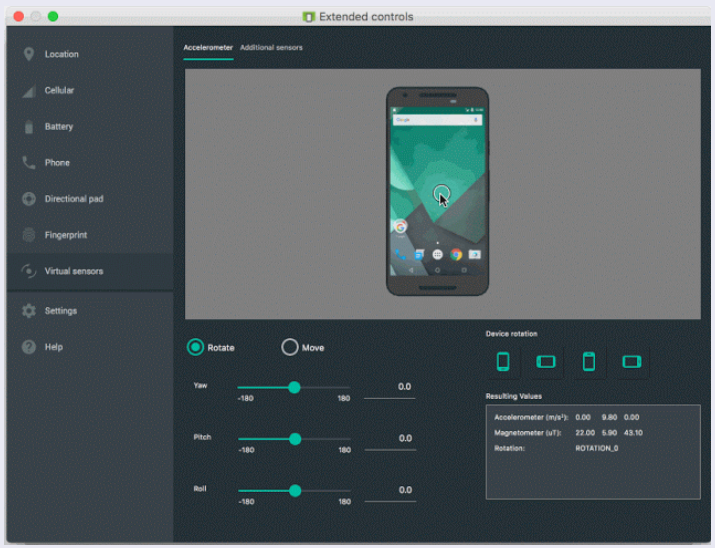
Wyrejestrowanie listenerów czujników

Pamiętaj, aby wyrejestrować listener czujnika, gdy skończysz go używać lub gdy aktywność czujnika zostanie wstrzymana. Jeśli listener czujnika jest zarejestrowany, a jego aktywność jest wstrzymana, czujnik będzie nadal pozyskiwał dane i wykorzystywał zasoby baterii, chyba że go wyrejestrujesz.

Najlepsze praktyki w zakresie uzyskiwania dostępu i korzystania z czujników

- Zbieranie danych z czujników tylko na pierwszym planie
- Wyrejestruj nasłuchiwaczy czujników
- Testuj za pomocą emulatora Androida
- Nie blokuj metody onSensorChanged()
- Unikaj używania przestarzałych metod lub typów czujników
- Weryfikuj czujniki zanim ich użyjesz
- Starannie dobieraj opóźnienia czujników

Test with the Android Emulator



Najlepsze praktyki w zakresie uzyskiwania dostępu i korzystania z czujników

- Zbieranie danych z czujników tylko na pierwszym planie
- Wyrejestruj nasłuchiwaczy czujników
- Testuj za pomocą emulatora Androida
- Nie blokuj metody onSensorChanged()
- Unikaj używania przestarzałych metod lub typów czujników
- Weryfikuj czujniki zanim ich użyjesz
- Starannie dobieraj opóźnienia czujników

Nie blokujemy metody *onSensorChanged()*

- Dane z czujników mogą się zmieniać z dużą szybkością - system może dość często wywoływać metodę *onSensorChanged(SensorEvent)*.
- Filtrowanie lub redukcja danych z czujników, powinno się wykonać poza metodą onSensorChanged(SensorEvent)

Najlepsze praktyki w zakresie uzyskiwania dostępu i korzystania z czujników

- Zbieranie danych z czujników tylko na pierwszym planie
- Wyrejestruj nasłuchiwaczy czujników
- Testuj za pomocą emulatora Androida
- Nie blokuj metody onSensorChanged()
- Unikaj używania przestarzałych metod lub typów czujników
- Weryfikuj czujniki zanim ich użyjesz
- Starannie dobieraj opóźnienia czujników

Ostrożnie dobieraj opóźnienia czujników

- Kiedy rejestrujesz czujnik za pomocą metody registerListener(), upewnij się, że wybrałeś szybkość dostarczania danych odpowiednią dla twojej aplikacji lub przypadku użycia.
- Pozwalanie systemowi na wysyłanie dodatkowych danych, których nie potrzebujesz, marnuje zasoby systemowe i zużywa energię baterii.

Czujniki ruchu

Czujniki ruchu są przydatne do monitorowania **ruch urządzenia, taki jak przechylanie, potrząsanie, obracanie lub kołysanie.** Możliwe architektury czujników różnią się w zależności od typu czujnika:

- Czujniki grawitacji, przyspieszenia liniowego, wektora obrotu, ruchu znaczącego, licznika kroków i detektora kroków są albo sprzętowe, albo programowe.
- Czujniki akcelerometru i żyroskopu są zawsze sprzętowe.

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

Przykład - Czujniki ruchu

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

Czujniki położenia

Czujniki położenia są przydatne do określania fizycznego położenia urządzenia w światowym układzie odniesienia. Na przykład, można użyć czujnika pola geomagnetycznego w połączeniu z akcelerometrem do określenia położenia urządzenia względem magnetycznego bieguna północnego.

Oblicz orientację urządzenia

```
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)
```

Oblicz orientację urządzenia

System oblicza kąty orientacji, wykorzystując czujnik pola geomagnetycznego urządzenia w połączeniu z akcelerometrem urządzenia. Korzystając z tych dwóch czujników sprzętowych, system dostarcza danych dla następujących trzech kątów orientacji:

- **Azimuth (degrees of rotation about the -z axis)** Jest to kąt pomiędzy aktualnym kierunkiem kompasu urządzenia a północą magnetyczną. Jeśli górna krawędź urządzenia jest skierowana na północ magnetyczną, azymut wynosi 0 stopni; jeśli górna krawędź jest skierowana na południe, azymut wynosi 180 stopni. Podobnie, jeśli górna krawędź jest skierowana na wschód, azymut wynosi 90 stopni, a jeśli górna krawędź jest skierowana na zachód, azymut wynosi 270 stopni.

Oblicz orientację urządzenia

System oblicza kąty orientacji, wykorzystując czujnik pola geomagnetycznego urządzenia w połączeniu z akcelerometrem urządzenia. Korzystając z tych dwóch czujników sprzętowych, system dostarcza danych dla następujących trzech kątów orientacji:

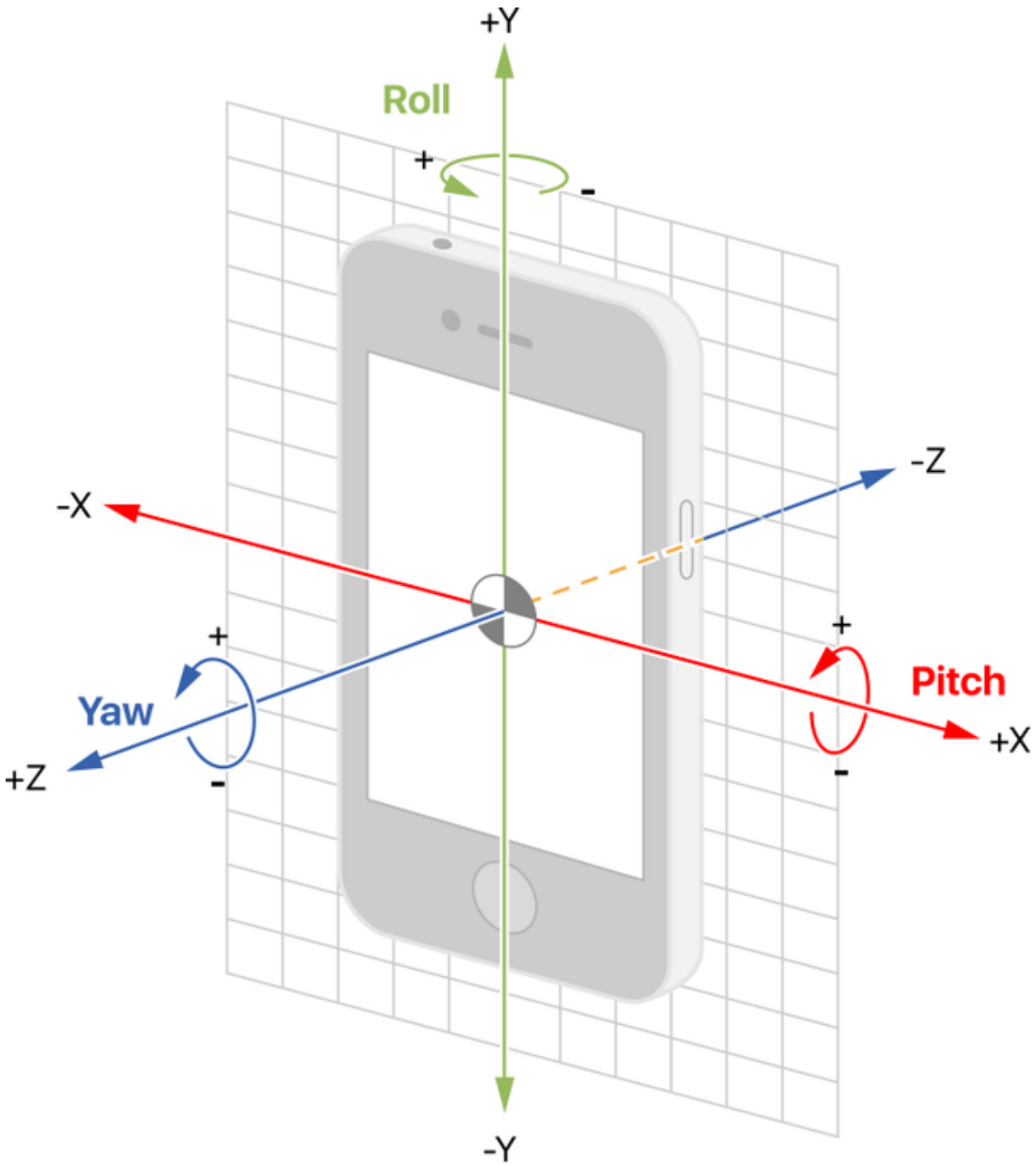
- **Pitch (degrees of rotation about the x axis)** Jest to kąt między płaszczyzną równoległą do ekranu urządzenia a płaszczyzną równoległą do podłoża. Jeśli trzymasz urządzenie równolegle do podłoża, a jego dolna krawędź znajduje się najbliżej Ciebie, i przechylasz górną krawędź urządzenia w kierunku podłoża, kąt nachylenia staje się dodatni. Przechylenie w przeciwnym kierunku - odsunięcie górnej krawędzi urządzenia od podłoża - powoduje, że kąt nachylenia staje się ujemny. Zakres wartości wynosi od -180 stopni do 180 stopni.

Oblicz orientację urządzenia

System oblicza kąty orientacji, wykorzystując czujnik pola geomagnetycznego urządzenia w połączeniu z akcelerometrem urządzenia. Korzystając z tych dwóch czujników sprzętowych, system dostarcza danych dla następujących trzech kątów orientacji:

- **Roll (degrees of rotation about the y axis)** Jest to kąt między płaszczyzną prostopadłą do ekranu urządzenia a płaszczyzną prostopadłą do podłoża. Jeśli trzymasz urządzenie równolegle do podłoża, a jego dolna krawędź znajduje się najbliżej Ciebie, i przechylasz lewą krawędź urządzenia w kierunku podłoża, kąt przechylenia staje się dodatni. Przechylenie w przeciwnym kierunku - przesunięcie prawej krawędzi urządzenia w kierunku podłoża - powoduje, że kąt przechylenia staje się ujemny. Zakres wartości wynosi od -90 stopni do 90 stopni.

Orientacja urządzenia



Czujniki środowiska

- Możesz użyć tych czujników do monitorowania względnej wilgotności otoczenia, natężenia oświetlenia, ciśnienia otoczenia i temperatury otoczenia w pobliżu urządzenia z systemem Android.
- Wszystkie cztery czujniki otoczenia są oparte na sprzęcie i są dostępne tylko wtedy, gdy producent urządzenia wbudował je w urządzenie.
- czujniki środowiskowe zwracają pojedynczą wartość czujnika dla każdego zdarzenia danych

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]

Przykłady

- AndroidManifest
- Service
- Callback methods
- Emulator

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

MOBILE APPLICATION DEVELOPMENT

Lokalizacja

Innovative Open Source courses for Computer Science

30.05.2021

Funded by
the European Union

Lokalizacja

Lokalizacja

Usługa pozwalająca na określenie pozycji urządzenia, a pośrednio użytkownika. W systemach mobilnych lokalizacja jest jednym z unikalnych cech, dzięki którym można tworzyć aplikacje świadome lokalizacji.

- Pozwala na określenie położenia urządzenia
- Lokalizacja ogólna
- Lokalizacja dokładna
- Google Play services - zalecana metoda lokalizacyjna w systemie Android

Przykłady aplikacji

- Informacja dostosowana do lokalizacji użytkownika (prognoza pogody w danym miejscu, wiadomości lokalne/komunikaty,stężenie alergenów)
- Informacja o zasobach znajdujących się w pobliżu (bank, apteka, pub)
- Interaktywne mapy i informacja turystyczna
- Reklamy mobilne zależne od położenia użytkownika
- Zarządzanie mobilnymi pracownikami

Google Play services

- Mechanizm określania pozycji na podstawie danych pochodzących od różnych dostawców, np. modułu GNSS (GPS), modułu WiFi czy Bluetooth.
- Fused Location Provider
- Szybsze określenie pozycji
- Mniejsze zużycia energii
- Dodatkowe możliwości np. geofencing, detekcja aktywności

Dodanie biblioteki Google Play services

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

Kroki wymagane do określenia pozycji

- Definicja typu lokalizacji
- Żądanie pozwolenia na lokalizację urządzenia
- Pobieranie lokalizacji (ostatnia znana, cykliczne pobieranie lokalizacji)
- Wykorzystanie lokalizacji, np. w celu pokazania punktu na mapie

Zdefiniowanie uprawnień

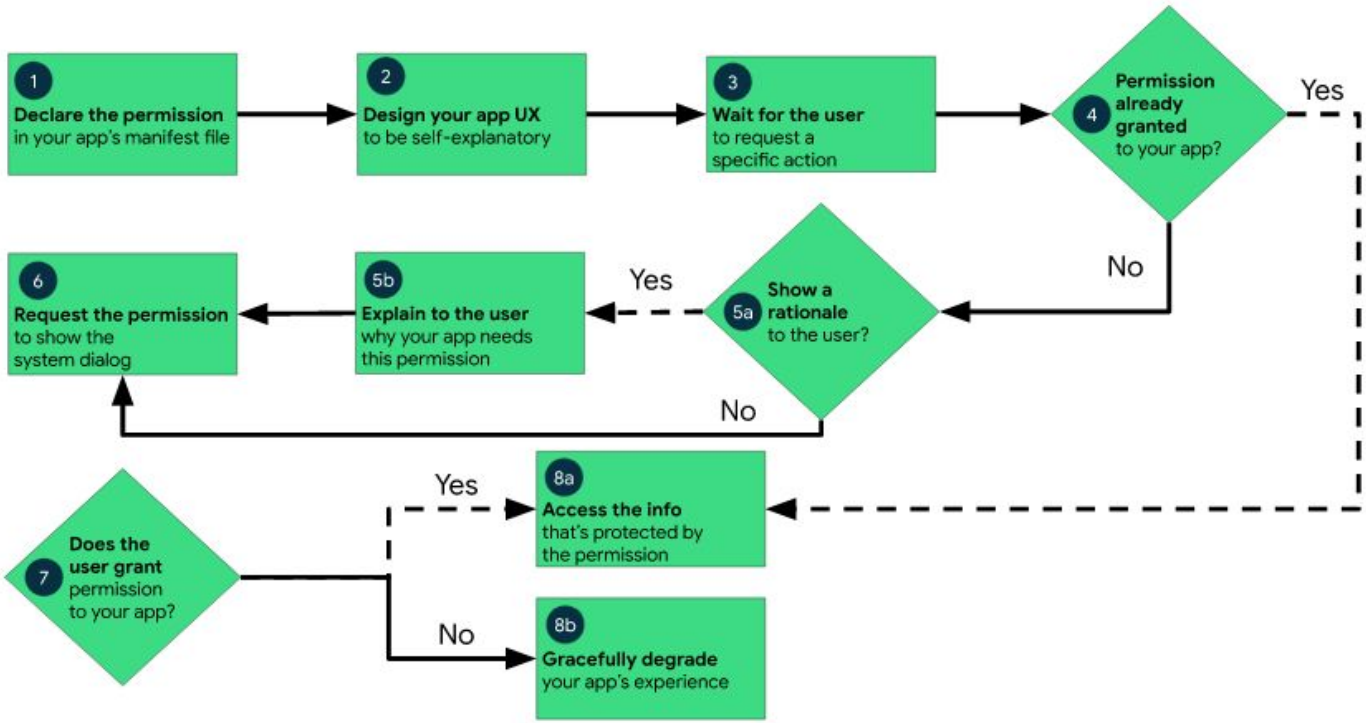
```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
...
  <!-- Always include this permission -->
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

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

Typy lokalizacji

- Lokalizacja ogólna
- Lokalizacja dokładna
- Foregrand location
- Background location

Żadanie uprawnień - workflow



Przykładowy kod pytający o pozwolenie na uprawnienia

```
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        }
```

Wykorzystanie Fused Location Provider - ostatnio znana pozycja

Definicja obiektu

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

Inicjalizacja

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

Pobranie ostatniej znanej lokalizacji

```
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            }
```

Lokalizacja ciągła - I

- Definicja obiektów
- Inicjalizacja
- Zdefiniowanie metod zwrotnych
- Włączenie i wyłączenie informacji o aktualizacji lokalizacji
- Dobór częstotliwości odświeżania

Lokalizacja ciągła II

Definicja obiektów

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

Inicjalizacja

```
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        }
```

Lokalizacja ciągła III

Włączenie informacji o aktualizacji lokalizacji

```
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            }
```

Wyłączenie informacji o aktualizacji lokalizacji

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

MOBILE APPLICATION DEVELOPMENT

MVVM

Innovative Open Source courses for Computer Science

30.05.2021

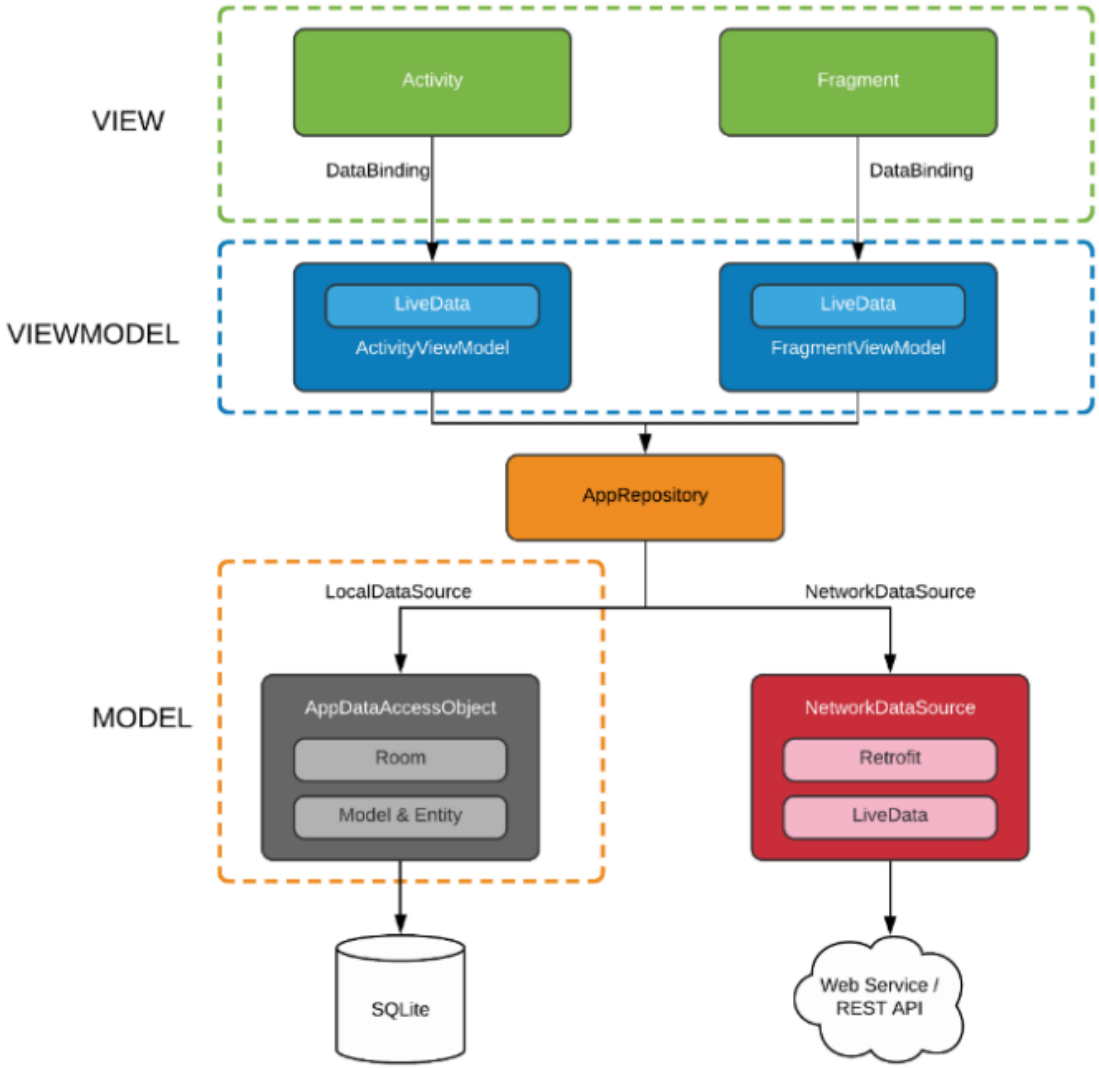
Funded by
the European Union

MVVM

Model — View — ViewModel jest uznanym w branży wzorcem architektonicznym oprogramowania, który przezwycięża wady wzorców projektowych MVP i MVC. MVVM sugeruje oddzielenie logiki prezentacji danych (Widoki lub UI) od głównej części logiki biznesowej aplikacji.

- **Model** - Przechowuje dane aplikacji. Nie może bezpośrednio rozmawiać z Widokiem. Generalnie zaleca się eksponowanie danych do ViewModelu poprzez Observables.
- **View** - Reprezentuje UI aplikacji pozbawionej jakiejkolwiek Logiki Aplikacji. Obserwuje ViewModel.
- **ViewModel** - Pełni rolę łącznika pomiędzy Modelem a Widokiem. Jest odpowiedzialny za przekształcanie danych z Modelu. Dostarcza strumienie danych do Widoku. Używa również callbacków do aktualizacji Widoku. Będzie prosił o dane z Modelu.

MVVM -2


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

Komponenty architektury

- Kolekcja bibliotek, które pomagają w projektowaniu solidnych, testowalnych i łatwych w utrzymaniu aplikacji
- Służą jako główna wytyczna w budowaniu szkieletu architektury naszego projektu
- Używając komponentów architektury Androida nie musimy się zbytnio martwić o zarządzanie cyklem życia aplikacji czy ładowanie danych do naszego UI

Komponenty

- ViewModel
- LiveData
- Lifecycle
- Extend LiveData

ViewModel

- Klasa ViewModel jest przeznaczona do przechowywania i zarządzania danymi związanymi z UI w sposób świadomy cyklu życia.
- Dzięki temu dane mogą przetrwać zmiany konfiguracji, takie jak obrót ekranu.
- Architecture Components dostarcza klasę pomocniczą ViewModel dla kontrolera UI, który jest odpowiedzialny za przygotowanie danych dla UI.
- Obiekty ViewModel są automatycznie zachowywane podczas zmian konfiguracji, dzięki czemu przechowywane przez nie dane są natychmiast dostępne dla następnej aktywności lub instancji fragmentu.

ViewModel - korzyści

- Pozwala na utrzymywanie stanu UI
- Zapewnia dostęp do logiki biznesowej.

Trwałość

ViewModel pozwala na przetrwanie zarówno przez stan, który przechowuje ViewModel, jak i operacje, które wyzwala ViewModel. To buforowanie oznacza, że nie trzeba ponownie pobierać danych przez częste zmiany konfiguracji, takie jak obrót ekranu.

Dostęp do logiki biznesowej

ViewModel jest właściwym miejscem do obsługi logiki biznesowej w warstwie UI. ViewModel odpowiada również za obsługę zdarzeń i delegowanie ich do innych warstw hierarchii, gdy trzeba zastosować logikę biznesową do modyfikacji danych aplikacji.

ViewModel - przykład

Definiowanie 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 }
```

Dostęp do listy z aktywności w następujący sposób

```
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 jest klasą obserwowalnych uchwytów danych.
- W przeciwieństwie do zwykłej obserwowalnej klasy, LiveData jest świadoma cyklu życia, co oznacza, że respektuje cykl życia innych komponentów aplikacji, takich jak aktywności, fragmenty czy usługi.
- Ta świadomość sprawia, że LiveData aktualizuje tylko te obserwatory komponentów aplikacji, które są w aktywnym stanie cyklu życia.
- LiveData podąża za wzorcem obserwatora. LiveData powiadamia obiekty obserwatorów, gdy zmienia się stan cyklu życia.
- LiveData również automatycznie przesyła istniejącą wartość, jeśli istnieje, do wszelkich nowych zarejestrowanych obiektów Observer.
- W połączeniu z funkcją automatycznego usuwania, czyni to LiveData bardzo wygodną w radzeniu sobie ze zmianami konfiguracji.

Typy LiveData

- LiveData - jest domyślnie niezmienna. Korzystając z LiveData możemy jedynie obserwować dane i nie możemy ich ustawiać.
- MutableLiveData - jest mutowalna i jest podklasą LiveData. W MutableLiveData możemy obserwować i ustawiać wartości za pomocą metod postValue() i setValue()
- MediatorLiveData - może obserwować inne obiekty LiveData takie jak źródła i reagować na ich zdarzenia onChange().

Rozszerzenie klasy LiveData

- LiveData is a lifecycle-aware component and thus it performs its functions according to the lifecycle state of other application components.
- If the observer's lifecycle state is active i.e., either STARTED or RESUMED, only then LiveData updates the app component.

```
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 }
```

Rozszerzenie klasy LiveData - 2

- **onActive()** - method is called when the LiveData object has an active observer. This means you need to start observing the stock price updates from this method.
- **onInactive()** - method is called when the LiveData object doesn't have any active observers. Since no observers are listening, there is no reason to stay connected.

Kroki do budowy aplikacji z wykorzystaniem wzorca projektowego MVVM

- Dodawanie DataBinding i implementacji w twoim pliku Gradle
- Utwórz nową klasę dla Modelu
- Utwórz nową klasę dla ViewModel
- Ulepszenie klasy View
- Zmiana układu graficznego

Configure project

build.gradle(app)

```
android{
    compileSdk 31

    dataBinding{
        enabled true
    }


    ...

dependencies{
    //ViewModel
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1'
    implementation 'androidx.activity:activity-ktx:1.4.0'
    //Lifecycle
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.4.1"
...
}
}
```

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)
  }
}
```

Zmień kod widoku

```
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    }
```

Zmiana układu

```
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

Storage

Radosław Maciaszczyk

Katedra Architektury Komputerów i Telekomunikacji

Szczecin, 7 sierpnia 2021

Opcje pamięci masowej w systemie Android

- Pamięć masowa specyficzna dla aplikacji
- Współdzielona pamięć masowa
- Preferencje
- Bazy danych

Pamięć masowa specyficzna dla aplikacji

Przechowuj pliki, które są przeznaczone tylko do użytku Twojej aplikacji, w dedykowanych katalogach w wewnętrznej pamięci masowej, albo w innych dedykowanych katalogach w zewnętrznej pamięci masowej. Używaj katalogów w wewnętrznej pamięci masowej do zapisywania poufnych informacji, do których inne aplikacje nie powinny mieć dostępu.

Opcje pamięci masowej w systemie Android

- Pamięć masowa specyficzna dla aplikacji
- Współdzielona pamięć masowa
- Preferencje
- Bazy danych

Współdzielona pamięć masowa

Przechowuj pliki, które Twoja aplikacja ma zamiar udostępniać innym aplikacjom, w tym media, dokumenty i inne pliki.

Opcje pamięci masowej w systemie Android

- Pamięć masowa specyficzna dla aplikacji
- Współdzielona pamięć masowa
- Preferencje
- Bazy danych

Preferencje

Przechowuj prywatne, proste dane w parach klucz-wartość.

Opcje pamięci masowej w systemie Android

- Pamięć masowa specyficzna dla aplikacji
- Współdzielona pamięć masowa
- Preferencje
- Bazy danych

Bazy danych

Przechowuj ustrukturyzowane dane w prywatnej bazie danych za pomocą biblioteki Room.

Storage - możliwości

	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

Który z nich wybrać?

- Jak dużo miejsca wymagają Twoje dane?
- Jak niezawodny musi być dostęp do danych?
- Jakiego rodzaju dane potrzebujesz przechowywać?
- Czy dane powinny być prywatne dla Twojej aplikacji?

Który z nich wybrać?

- Jak dużo miejsca wymagają Twoje dane?
- Jak niezawodny musi być dostęp do danych?
- Jakiego rodzaju dane potrzebujesz przechowywać?
- Czy dane powinny być prywatne dla Twojej aplikacji?

Jak dużo miejsca potrzebują Twoje dane?

Wewnętrzna pamięć masowa ma ograniczoną ilość miejsca na dane związane z aplikacjami. Jeśli chcesz zapisać dużą ilość danych, użyj innych rodzajów pamięci masowej.

Który z nich wybrać?

- Jak dużo miejsca wymagają Twoje dane?
- Jak niezawodny musi być dostęp do danych?
- Jakiego rodzaju dane potrzebujesz przechowywać?
- Czy dane powinny być prywatne dla Twojej aplikacji?

Jak niezawodny musi być dostęp do danych?

Jeśli podstawowa funkcjonalność Twojej aplikacji wymaga pewnych danych, np. podczas uruchamiania aplikacji, umieść te dane w wewnętrznym katalogu pamięci masowej lub w bazie danych. Pliki specyficzne dla aplikacji, które są przechowywane w pamięci zewnętrznej, nie zawsze są dostępne, ponieważ niektóre urządzenia pozwalają użytkownikom na usunięcie fizycznego urządzenia, które odpowiada pamięci zewnętrznej.

Który z nich wybrać?

- Jak dużo miejsca wymagają Twoje dane?
- Jak niezawodny musi być dostęp do danych?
- Jakiego rodzaju dane potrzebujesz przechowywać?
- Czy dane powinny być prywatne dla Twojej aplikacji?

Jakiego rodzaju dane trzeba przechowywać?

Jeśli masz dane, które mają znaczenie tylko dla Twojej aplikacji, użyj pamięci masowej specyficznej dla aplikacji. W przypadku treści multimedialnych, które można udostępniać, użyj współdzielonego magazynu, aby inne aplikacje mogły uzyskać do nich dostęp. W przypadku danych strukturalnych użyj preferencji (dla danych kluczowych) lub bazy danych (dla danych zawierających więcej niż 2 kolumny).

Który z nich wybrać?

- Jak dużo miejsca wymagają Twoje dane?
- Jak niezawodny musi być dostęp do danych?
- Jakiego rodzaju dane potrzebujesz przechowywać?
- Czy dane powinny być prywatne dla Twojej aplikacji?

Czy dane powinny być prywatne dla Twojej aplikacji?

Podczas przechowywania wrażliwych danych - danych, które nie powinny być dostępne z żadnej innej aplikacji - używaj pamięci wewnętrznej, preferencji lub bazy danych. Wewnętrzna pamięć masowa ma tę dodatkową zaletę, że dane są ukryte przed użytkownikami.

Dostęp do plików specyficznych dla aplikacji

- Wewnętrzne katalogi pamięci masowej - Katalogi te obejmują zarówno dedykowaną lokalizację do przechowywania trwałych plików, jak i inną lokalizację do przechowywania danych pamięci podręcznej. System uniemożliwia innym aplikacjom dostęp do tych lokalizacji, a w systemie Android 10 (poziom API 29) i nowszych, lokalizacje te są szyfrowane. Te cechy sprawiają, że lokalizacje te są dobrym miejscem do przechowywania wrażliwych danych, do których dostęp ma tylko sama aplikacja.
- Zewnętrzne katalogi pamięci masowej - Te katalogi zawierają zarówno dedykowaną lokalizację do przechowywania trwałych plików, jak i inną lokalizację do przechowywania danych pamięci podręcznej. Chociaż inna aplikacja może uzyskać dostęp do tych katalogów, jeśli ma odpowiednie uprawnienia, pliki przechowywane w tych katalogach są przeznaczone tylko dla twojej aplikacji. Jeśli zamierzasz tworzyć pliki, do których inne aplikacje powinny mieć dostęp, Twoja aplikacja powinna

Przykłady

- `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`
- Aby dodatkowo zabezpieczyć pliki specyficzne dla aplikacji, użyj biblioteki Security, która jest częścią Android Jetpack, aby zaszyfrować te pliki w spoczynku. Klucz szyfrowania jest specyficzny dla twojej aplikacji.

Dostęp do plików multimedialnych z udostępnionej pamięci masowej

- Aby zapewnić bardziej wzbogacone doświadczenie użytkownika, wiele aplikacji pozwala użytkownikom na wnoszenie i dostęp do mediów, które są dostępne na zewnętrznym woluminie pamięci masowej.
- Framework zapewnia zoptymalizowany indeks do kolekcji mediów, zwany sklepem medialnym, który pozwala na łatwiejsze pobieranie i aktualizowanie tych plików multimedialnych.
- Nawet po odinstalowaniu aplikacji, pliki te pozostają na urządzeniu użytkownika.

ContentResolver

Aby wejść w interakcję z abstrakcją media store, użyj obiektu ContentResolver:

```
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         // Use an ID column from the projection to get
15         // a URI representing the media item itself.
16     }
17 }
```

Pliki multimedialne

System automatycznie skanuje zewnętrzny wolumen pamięci masowej i dodaje pliki multimedialne do następujących dobrze zdefiniowanych kolekcji:

- **Images**, w tym zdjęcia i zrzuty ekranu, które są przechowywane w katalogach *DCIM/* i *Pictures/*. System dodaje te pliki do tablicy *MediaStore.Images*.
- **Videos**, które są przechowywane w katalogach *DCIM/*, *Movies/* i *Pictures/*. System dodaje te pliki do tabeli *MediaStore.Video*.
- **Audio files**, które są przechowywane w katalogach *Alarms/*, *Audiobooks/*, *Music/*, *Notifications/*, *Podcasts/*, *and Ringtones/*, jak również listy odtwarzania audio, które znajdują się w katalogach *Music/ or Movies/*. System dodaje te pliki do tabeli *MediaStore.Audio*.
- **Downloaded files**, które są przechowywane w katalogu *Download/*. Na urządzeniach z systemem Android 10 (poziom API 29) i nowszym, pliki te są przechowywane w tabeli

Preferencje aplikacji

- Jeśli masz stosunkowo niewielką kolekcję wartości kluczowych, które chciałbyś zapisać, powinieneś użyć obiektu SharedPreferences.
- Obiekt SharedPreferences wskazuje na plik zawierający pary klucz-wartość i udostępnia proste metody do ich odczytu i zapisu.
- Każdy plik SharedPreferences jest zarządzany przez framework i może być prywatny lub współdzielony.

SharedPreferences

- Klasa ta zapewnia ogólne ramy, które pozwalają na zapisywanie i pobieranie trwałych par klucz-wartość prymitywnych typów danych.
- Możesz użyć SharedPreferences do zapisania dowolnych prymitywnych danych: booleans, float, ints, longs i strings.
- Dane te będą zachowane przez wszystkie sesje użytkownika (nawet jeśli Twoja aplikacja zostanie zabita).
- *"SharedPreferences"* są zapisywane jako pliki XML w folderze *shared_prefs*.

How using SharePreferences ?

- 1. Pobierz preferencje z określonego pliku

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

- 2. Read

```
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. Write and Apply (or Commit)changes

```
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 }
```

Ustawienia

- Aplikacje często zawierają ustawienia, które pozwalają użytkownikom modyfikować funkcje i zachowania aplikacji.
- Ustawienia to miejsce w Twojej aplikacji, gdzie użytkownicy wskazują swoje preferencje dotyczące jak powinna zachowywać się Twoja aplikacja.
- Ustawienia mają niski priorytet w projektowaniu, ponieważ nie są często używane.

Jak używać

- 1. Utwórz zasób XML systemu Android o nazwie np. *preferences.xml* PreferenceScreen.xml
- 2. Dodaj do pliku PreferencesCategory, oraz jeden z *CheckBoxPreference, ListPreference, EditTextPreference*
- 3. Utwórz klasę *MyPreferencesActivity*, która rozszerza *PreferenceActivity* Ta aktywność wczytuje *preference.xml* plik i pozwala użytkownikowi na zmianę wartości.
- 4. Dodaj do metody *onOptionsItemSelected()* kod uruchamiający PreferencesActivity

RoomDatabase

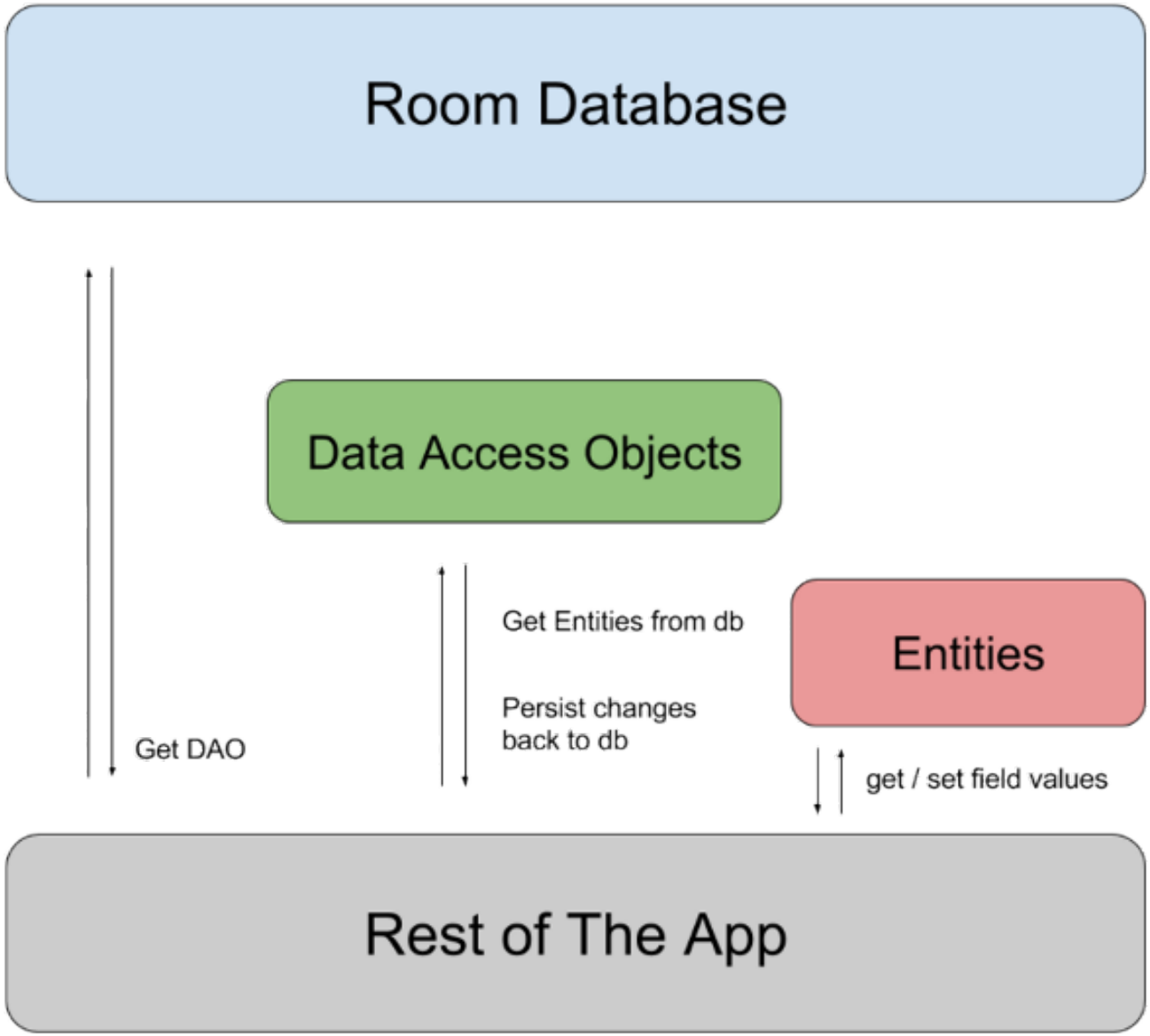
- Room zapewnia warstwę abstrakcji nad SQLite, aby umożliwić płynny dostęp do bazy danych, jednocześnie wykorzystując pełną moc SQLite.
- Aplikacje operujące na nietrywialnych ilościach danych strukturalnych mogą odnieść duże korzyści z przechowywania tych danych lokalnie.
- Room zajmuje się buforowaniem danych, gdy urządzenie jest offile
- Ta funkcjonaność powoduje, że Room jest zalecany do użycia zamiast SQLite

RoomDatabase

Główne komponenty w Room

- Database - Zawiera uchwyt bazy danych i służy jako główny punkt dostępu do bazowego połączenia z persystentnymi, relacyjnymi danymi Twojej aplikacji.
- Entity - Reprezentuje tabelę w bazie danych
- DAO - zawiera metody używane do dostępu do bazy danych

Schemat architektury Room



Example code 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 )
```

Example code 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 }
```

Przykładowa instancja kodu Database

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

Live Example

Room Database Example