Tvorba mobilních aplikací

6. seminář

Radek Janoštík

Univerzita Palackého v Olomouci

22. 3. 2022

Opakování

- Definice vzhledů bychom neměli opakovat
  - Definice barev v `/res/values/colors.xml`
  - Definice stylů v `/res/values/styles.xml`
  - Globální styly = témata – `/res/values/themes.xml`
- Zobrazované řetězce by neměly být v aplikaci „napevno“
  - Pojmenování řetězců a přesunutí obsahu do `/res/values/strings.xml`
  - Podpora jazykových mutací
- Zdroje můžeme přizpůsobovat podle hustoty pixelů, orientace displeje, jazyka, ...
  - Konvence pojmenování adresářů `adresář-kategorie`
  - Např.: `values-cs`, `layout-land`, `drawable-xdpi`, ...
- Při zobrazování seznamů položek používáme `RecyclerView`
  - „Na pozadí“ máme kolekci položek
  - Nadefinujeme, jak má vypadat jedna položka
  - Definujeme `Adapter`, který mapuje položky na View prvky

Toolbar menu (1 / 3)

- Položky v toolbar menu můžeme definovat v `/res/menu/nazev_aktivity.xml`
- Dokumentace: `https://developer.android.com/guide/topics/resources/menu-resource`

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_add"
        android:icon="@drawable/ic_baseline_add_box_24"
         android:title ="@string/add"
        app:showAsAction="always"/>
    <item android:id="@+id/action_settings"
         android:title ="@string/settings"
        app:showAsAction="never"/>
</menu>
```

Toolbar menu (2 / 3)

- Význam položek
  - `android:id` – id pro odkazování z kódu (definice akce, . . .)
  - `android:icon` – ikona
  - `android:title` – zobrazovaný text
  - `android:showAsAction` – definice zobrazení – jen ikona, text, text pouze pokud je míst, pouze v menu
- Pro zobrazení musíme změnit téma na `NoActionBar` a přidat toolbar do layoutu aktivity

```
<android.support.v7.widget.Toolbar
        android:id="@+id/main_toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary"
        android:elevation="4dp"
        android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Toolbar menu (3 / 3)

- Potřebujeme ještě dodefinovat zobrazení menu:
  - Přidání setSupportActionBar(findViewById(R.id.main_toolbar)) do metody onCreate()
  - Definování, jaký layout pro menu se má použít:

```
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater. inflate (R.menu.activity_main, menu)
        return true
    }
```

- Reakce na klik:

```
override fun onOptionsItemSelected(item: MenuItem) = when (item.itemId) {
    R.id. action_settings  -> { Log.w("warn","klik na nastaveni")
                                true
    }
    R.id.action_add -> {Log.w("warn","klik na pridani")
        true
    }
    else -> super.onOptionsItemSelected(item)
}
```

Změna titulku a ikony aplikace

- Obojí můžeme změnit v `AndroidManifest.xml`
- Titulek – atribut `android:label` – řetězec nebo odkaz na něj
- Ikona – atribut `android:icon` – odkaz do složky `drawable` nebo `mipmap`
  - *mipmap* $\approx$ drawable používající jiný přepočet hustoty pixelů – vhodnější pro launcher
  - Doporučuji použít nějaké generátory ikon (web, aplikace)
  - atribut `android:roundIcon` pro kulaté ikony
  - Adaptivní ikony – různé masky, animace, od 7.1 – API 25 `https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive`

Ukládání dat

- Prozatím naše aplikace neuměla nikam uložit svá data
- Po restartu aktivit se vše smazalo
- Máme k dispozici několik způsobů jak ukládat trvalá data
  - *SharedPreferences* – určeno pro jednoduchá data
  - *Statické soubory – assets* – jen pro čtení, zabaleno v APK
  - *Aplikační soubory* – „Filesystém“ pouze pro aplikaci
  - *Relační databáze* – SQLite
  - *Media* – Je potřeba speciální oprávnění
- Více na: `https://developer.android.com/training/data-storage`
  - Zvláště úvodní tabulka (viditelnost a „trvanlivost“ uložených dat)

SharedPreferences

- Slouží k ukládání dvojic tvaru `Klíč:Hodnota`
- Klíče – pouze řetězce
- Hodnoty
  - Primitivní typy `Int, Float, Boolean, Long`
  - Řetězce (`String`)
  - Množina řetězců: `Set<String>`
- Uloženo v souboru ve vnitřní paměti
  - Nemáme k němu přímý přístup, pouze přes rozhraní systému

SharedPreferences – uložení dat

- Třídu `SharedPreferences` získáme pomocí getPreferences(Context.MODE_PRIVATE)
- Můžeme mít i více „souborů“ pro ukládání – getSharedPreferences(getString(R.string.FileName), Context.MODE_PRIVATE)
  - Pokud neexistuje, tak se vytvoří nové
- Uložení:

```
val sharedPref = getPreferences(Context.MODE_PRIVATE) ?: return
with (sharedPref.edit()) {
    putString(getString(R.string.storedKey), textbox.text.toString())
    apply()
}
```

- Metody `putString, putInt, putStringSet,...`
- Nezapomenout zavolat metodu `apply()` pro uložení

SharedPreferences – načtení dat

- Načtení je obdobné:

  ```
  val sharedPref = getPreferences(Context.MODE_PRIVATE) ?: return
  val storedText = sharedPref.getString(getString(R.string.storedKey), "")
  ```

  Metody `getString, getInt, getStringSet,...`

- Jako druhý argument dávám hodnotu, která se vrátí v případě, že klíč nebyl nalezen

Statické soubory aplikace – assets

- Soubory, které jsou zabalené „natvrdo“ v `apk` souboru
- Uložené v adresáři `/src/main/assets`
- Jsou pouze ke čtení
- Vhodné pro konfigurační soubory, obrázky, zvuky, ...
- Dostupné přes `assets.open()`. Např.:

```
val text = assets.open("test.txt").bufferedReader().readText()
```

Aplikační soubory

- Systém poskytuje úložiště jen pro aplikaci
- Přístupné pomocí klasického Java I/O
- Při odinstalování je úložiště smazáno
  - Ne pro data, které chtějí uživatelé zachovat
- Dvě úložiště
  - Pro „trvalé soubory“ – `context.filesDir`
  - Pro dočasné (cache) soubory – `context.cacheDir`
- Přístup ke klasickému souboru

```
val file = File(this.filesDir, "soubor")
```

- Procházení souborů `var files : Array<String> = context.fileList()`
- Můžeme zvolit i externí úložiště (více v dokumentaci)

Relační databáze SQLite

- Oblíbená, odlehčená databáze pro relační data
- Obsaženo v jednom souboru, není potřeba DB server
- Podobné známým plnohodnotným databázím (PostgreSQL, MySQL, MSSQL, . . .)
- Android podporuje SQLite nativně, nepotřebujeme další knihovny
- Přímý přístup se již nedoporučuje
  `https://developer.android.com/training/data-storage/sqlite`
- ⇒ Použití knihovny *Room* pro objektově relační mapování
  - Nemusíme psát tolik zbytečného kódu

ORM v kostce

- Při „klasickém“ přístupu k SQL databázím musíme psát mnoho zdlouhavého kódu
  - Určovaní, který sloupec se dá do které proměnné
  - Spousta „polo-magických konstant“
  - Velký prostor pro chyby
  - Špatné ošetřování chybových stavů
- Idea: „Co kdyby někdo za nás určoval, jak se databázové objekty přemění na programové“
- Objektově Relační Mapování takto funguje
- Často je možné, že nejdříve píšeme objekty v kódu a z nich se vytvoří databáze

Knihovna Room

- ORM knihovna součástí Android Jetpack
  `https://developer.android.com/jetpack`
- V čase kompilace kontroluje SQL dotazy
- Podpora asynchronních operací (více příště)
- „Konfigurace“ pomocí anotací
- Podpora migrací databázového modelu
- Do `build.gradle` modulu přidáme do `dependecies`

```
def room_version = "2.4.2"
implementation "androidx.room:room-runtime:$room_version"
kapt "androidx.room:room-compiler:$room_version"
implementation "androidx.room:room-ktx:$room_version"
```

- Do `build.gradle` modulu přidáme do `plugins`

```
id ' kotlin –kapt'
```

Room – struktura

- Pro přístup k datům definuje Room několik vrstev
- Databázová třída – Room Database
  - Zajišťuje přístup k SQLite
  - Obsahuje definici „tabulek“
  - Řeší migrace mezi verzemi databáze
  - Definujeme pouze abstraktní třídu
- Databázové entity – nosiče dat
  - Jednotlivé datové třídy
  - Definice vlastností
- *Data Access Objects(DAO)*
  - Poskytují metody na získání, modifikaci, vytvoření dat
  - Definujeme pouze rozhraní, implementaci doplní Room

Room – databázové entity

- Vytvoříme objekty, které budeme chtít ukládat do databáze

```
@Entity(tableName = "notes")
data class Note(@PrimaryKey(autoGenerate = true) val id:Long,
                @ColumnInfo(name="note_title") val title: String,
                val text : String,
                val archived: Boolean) {
}
```

- *Anotace* jsou uvedeny za zavináčem
  - *Entity* – deklarujeme, že daný objekt bude uložen do DB tabulky `notes`
  - *PrimaryKey* – vlastnost bude primárním klíčem (unikátnost), hodnota bude přidělena
  - *ColmnInfo* – můžeme modifikovat informace o sloupci
- Můžeme definovat vazby mezi objekty (1:1, 1:N, M:N) `https://developer.android.com/training/data-storage/room/relationships`

Room – DAO (1 / 2)

- Pro každou vytvořenou *entitu* vytvoříme DAO
- Pomoci DAO budeme provádět veškeré databázové operace

```
@Dao
interface NoteDao {
    @Query("SELECT * FROM notes")
     fun getAll(): List<Note>
    @Query("SELECT * FROM notes WHERE note_title LIKE :title LIMIT 1")
     fun findByTitle( title : String): Note
    @Insert
     fun insert(note: Note): Long
    @Delete
     fun delete(note: Note)
    @Update
     fun update(notes: List<Note>) : Int
}
```

- Anotace `@Dao` označuje náš objekt jako DAO objekt pro Room

Room – DAO (2 / 2)

- `@Query` metody zpřístupňují SQL dotazy jako metody
  - Dotazy jsou kontrolovány v čase kompilace
  - Parametry (např. pro filtraci) odkazovány z SQL pomocí dvojtečky
  - Podpora dotazů se *spojením (JOIN)*, return jako Multimapa
- `@Insert` metody umožňují vkládat entity
  - Parametrem musí být *room entita* nebo jejich kolekce
  - Pokud je parametrem jedna entita, může vracet `long` (rowId)
  - Je-li parametrem více entit může vracet kolekci `long`
- `@Update` metody umožňují upravovat stávající entity
  - Parametry podobně jako Insert
  - Primární klíč je použit pro „spárování“ entity a jejich update
  - Mohou vracet `int` – počet upravených řádků
- `@Delete` metody umožňují mazat entity
  - Parametry podobně jako Insert
  - Primární klíč je použit pro nalezení entity ke smazání
  - Mohou vracet `int` – počet smazaných řádků

Room – databáze

- Nadefinujeme abstraktní třídu pro naší databázi
- Agreguje použitelné DAO objekty
- Umožňuje správu migrací `https://developer.android.com/training/data-storage/room/migrating-db-versions`

```
@Database(entities = [Note::class], version = 1)
abstract class AppDb : RoomDatabase() {
    abstract fun noteDao(): NoteDao
}
```

- V parametru anotace udáváme seznam ukládaných entit
- Room na základě anotací vytvoří implementaci

Room – databáze – jedináček

- Potřebujeme zajistit, aby existovala pouze jedna instance databáze
- ⇒ Použijeme návrhový vzor jedináček, upravíme definici

```
@Database(entities = [Note::class], version = 1)
abstract class AppDb : RoomDatabase() {
    abstract fun noteDao(): NoteDao
    companion object {
        private var instance: AppDb? = null
        fun getDatabase(context: Context) : AppDb {
            if (instance!=null) {
                return instance as AppDb
            } else {
                instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDb::class.java,
                    "MojeDatabase"
                ).allowMainThreadQueries().build()
                return instance as AppDb
            }
```

Room – použití (1 / 2)

- Pro vložení záznamu musíme získat databázi z jedináčka, vytvořit entitu a pomocí DAO ji vložit

```
val db = AppDb.getDatabase(applicationContext) // ziskame DB
val dao = db.noteDao() // ziskame DAO
val n = Note(0, "titulek", "text", false) // Vytvorime entitu
dao.insert(n) // vlozime do tabulky
```

- Pro update musíme mít entitu s existujícím primárním klíčem

```
val db = AppDb.getDatabase(applicationContext) // ziskame DB
val dao = db.noteDao() // ziskame DAO
val note = dao.findByTitle("titulek")
note. title  = "Zmena"
dao.update(note)
```

- Pozor na `null`

Room – použití

- Vymazání je obdobné:

```
val db = AppDb.getDatabase(applicationContext) // ziskame DB
val dao = db.noteDao() // ziskame DAO
val note = dao.findByTitle("Zmena")
dao.delete(note)
```

- Pozor na `null`
- Čtení obdobně pomocí `Query` metod. Např.:

```
val notes = dao.getAll()
```

- Při „bouřlivém vývoji“ jsou migrace spíše na škodu
  - Výmaz dat aplikace v nastavení
  - Nebo povýšit verzi a přidat `.fallbackToDestructiveMigration()` databaseBuilderu

Úkol

1. Přidejte poznámkám z předchozího úkolu `id: Long` a `archived: Boolean`
2. Přidejte aktivitu pro editaci poznámky a umožněte po kliku poznámku editovat
3. Přidejte aktivitu `Nastavení` kde si uživatel může pomocí switche zvolit, zda zobrazovat archivované poznámky
4. Změna nastavení bude persistentní (Hint: *SharedPreferences*)
5. Hlavní aktivita bude vizuálně odlišovat archivované a aktivní poznámky (např. šedé pozadí
   - Hint: onBindViewHolder
6. Předělejte úložiště poznámek do SQLite databáze za pomocí knihovny Room
   - Hint: Při `onCreate, po přidání, po editaci` načíst aktuální data z databáze do „našeho“ **val** notes: MutableList<Note>