Tvorba mobilních aplikací

8. seminář

Radek Janoštík

Univerzita Palackého v Olomouci

5. 4. 2022

Opakování

- Je rozumné mít v architektuře aplikace pořádek
  - Mít rozdělené „kdo má odpovědnost z jakou funkcionalitu“
- Existuje řada zaběhnutých vzorů pro architekturu aplikace
- Pro Android je doporučovaná architektura MVVM
  - Datová vrstva – data + logika, delší životnost, persistentní stav
  - UI vrstva – UI elemety a „state holders“
- Máme k dispozici třídu `ViewModel`
  - Uchovává vnitřní stav UI
  - Bere v potaz životní cyklus komponent
  - Data z ViewModelu se *bindují* na jednotlivé UI prvky

Práce se sítí

- Pro práci se sítí (Internetem) je potřeba udělit oprávnění
- Do elementu `<manifest>` v `AndroidManifest.xml` přidáme

  ```
  <uses–permission android:name="android.permission.INTERNET" />
  ```

- Systém neumožní pracovat „se sítí“ v UI vlákně
  - Veškerou komunikace budeme provádět v modelu (v nějakém repositáři)
  - Spouštět ji budeme asynchronně pomocí *korutin*

Prostředky pro práci se sítí

- `HttpUrlConnection` – základní třída v SDK
  - Dostatečná funkcionalita, „surovější“
  - Podpora SSL, cookies, autentizace, cachování
- WebView
  - Zobrazení webových stránek v aplikaci (UI prvek, ovládatelný z kódu)
- DownloadManager
  - Systémová *služba* pro stahování objemných souborů
  - Podpora notifikací, callbacky, . . .
- Knihovny třetích stran
  - Retrofit – doporučuji pro webové API
  - Google Volley
  - . . . (dle gusta)

Stažení poznámek z internetu

- Pojďme do našeho projektu přidat získávání poznámek z internetu
- Přidáme do našeho `NoteRepository` funkci **fun** getNotesFromWeb(): **String**

```
fun getStringNotesFromWeb() : String {
    val urlConnection =
        URL("https://apollo.inf.upol.cz/~janostik/tmap/notes.php")
                                        .openConnection() as HttpURLConnection
    val data = urlConnection.inputStream.bufferedReader().readText()
    Log.w("warn", data)
    urlConnection.disconnect()
    return data
}
```

- Tím získáme řetězec obsahující *json* s nějakými daty
  - Chtělo by z něj získat „naše“ `Note`

Intermezzo: JSON

- JavaScript Object Notation(JSON)
- „Neoficiální datový jazyk pro webové služby“ – dvě struktury

1. *Objekt* – kolekce párů „klíč–hodnota“
   - Bývá uvozen složenými závorkami {}
   - Klíče – řetězce v uvozovkách, dvojtečka, hodnoty
   - Odděleny čárkami

```
{"id": 0, "title": "Titulek Poznamky", "text": "Telo poznamky", "archived": false}
```

2. *seznam/pole* objektů/hodnot
   - Uvozen hranatými závorkami []
   - Objekty/hodnoty odděleny čárkami

```
[
{"id": 0, "title": "Titulek Poznamky", "text": "Telo poznamky", "archived": false},
{"id": 1, "title": "Kup chleba", "text": "V Albertu, je v akci", "archived": true},
{"id": 2, "title": "Zjisteni", "text": "P=NP, nikomu to nerikej", "archived": false}
]
```

Deserializace

- JSON určitě nebudeme parsovat ručně – použijeme deserializaci
- Do `build.gradle` projektu (top-level) přidáme do `dependencies`

  classpath `"org.jetbrains.kotlin:kotlin-serialization:1.5.20"`

- Do `build.gradle` modulu přidáme
  - Do `plugins`: id ' kotlinx – serialization '
  - Do `dependencies`:
    implementation `"org.jetbrains.kotlinx:kotlinx-serialization-json:1.2.2"`
- Označíme naši třídu s poznámkou anotací @Serializable

Deserializace poznámek

- Nyní již můžeme upravit naši metodu, aby vracela seznam poznámek

```
fun getNotesFromWeb() : List<Note> {
        val data = getStringNotesFromWeb()
        val dataDecoded = Json.decodeFromString<List<Note>>(data)
        Log.w("warn", dataDecoded.toString())
        return dataDecoded
    }
```

- URL adresa v přechozí metě by měla být v Resources

Services – služby

- Korutiny jsou dobrým nástroj pro krátkodobé akce na pozadí
  - Jsou závislé na životním cyklu aktivity
- *Services (služby)* – běh déle trvajících operací na pozadí
- Nemají uživatelské rozhraní
- Např.: Přehrávání hudby, stahování dat, sledování polohy, ...
- Mohou být spuštěny aktivitami, mohou s nimi komunikovat (zobrazovat stav)
- Dva typy
  - Started services – systém je nechá běžet, dokud nedokončí svoji práci (dva přístupy – informovanost uživatele vs. služba na pozadí, o které neví)
  - Bound services – spuštěny jako závislost jiné aplikace (poskytují API, službu)
- Implementace vlastní služby rozšířením třídy `Service`

Vytvoření služby

- Vytvoříme vlastní třídu, která bude dědit z `Service`
  - Implementuje metody:

```
override fun onBind(p0: Intent?): IBinder? {
    return null;
}
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    // vytvoreni vlakna a kod prace na pozadi
    return Service.START_NOT_STICKY
}
```

- `onStartCommand` – vykoná se, při spuštění služby
  - Běží v UI vlákně → musíme vytvořit vlastni
  - Musí vracet jednu z konstant (jak se budou chovat při zabití):
    - `START_STICKY` – služba bude obnovena jen pokud jí je doručován nějaký intent
    - `Service.START_NOT_STICKY` – bude obnovena s `null` intentem nebo jako výše
    - `START_REDELIVER_INTENT` – bude obnovena s posledním intentem nebo s čekajícími

Vytvoření služby

- Vytvoříme vlastní třídu, která bude dědit z `Service`
  - Implementuje metody:

```
override fun onBind(p0: Intent?): IBinder? {
    return null;
}
override fun onStartCommand(intent: Intent?, flags: Int, startId : Int): Int {
    // vytvoreni vlakna a kod prace na pozadi
    return Service.START_NOT_STICKY
}
```

- `onStartCommand` – vykoná se, při spuštění služby
  - Běží v UI vlákně → musíme vytvořit vlastni
  - Musí vracet jednu z konstant (jak se budou chovat při zabití):
    - ★ `START_STICKY` – služba bude obnovena jen pokud jí je doručován nějaký intent
    - ★ `Service.START_NOT_STICKY` – bude obnovena s `null` intentem nebo jako výše
    - ★ `START_REDELIVER_INTENT` – bude obnovena s posledním intentem nebo s čekajícími
- Musíme přidat službu do `AndroidManifest.xml` do elementu `application`
  <service android:name=".services.MyService"/>

Spuštění služby

- Spouštění probíhá pomocí `Intentu`

```
val i = Intent(app.applicationContext, MyService::class.java)
i.putExtra("bool", true)
app.applicationContext.startService(i)
```

- Definujeme, jakou třídu služby chceme spustit
- Můžeme předat data pomocí metod `putExtra()` jako aktivitám
- Metoda `override fun onStartCommand()` vykonána v UI vlákně
  - Vytvořit nové vlákno

```
override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    val app = application as MyApplication
    val thread = Thread { doSomeWork()
     doAnotherWork
     stopSelf()
    }
  }
  thread.start()
  return Service.START_NOT_STICKY
}
```

Ukončení služby

- Službu ukončíme (téměř) úplně stejně, jako jsme ji vytvořili
- Zavoláme metodu `stopService(intent)`
  - Intent definuje službu, kterou ukončujeme
  - Data v intentu nehrají žádnou roli

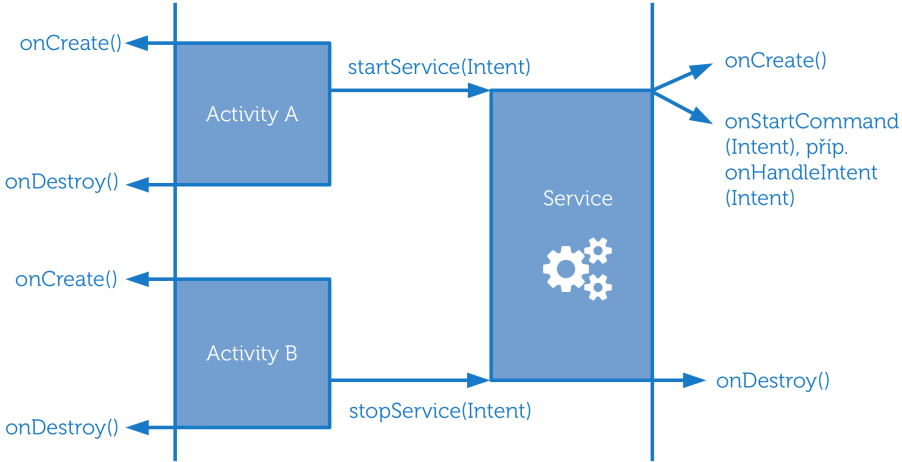
```
val i = Intent(app.applicationContext, MyService::class.java)
app.applicationContext.stopService(i)
```

- Pokud jí běží nějaké vlákno, automaticky se neukoční (nikdo o něm neví)

```
override fun onDestroy() {
        thread.interrupt()
        super.onDestroy()
    }
```

- Služba by se měla sama ukončit po dokončení své práce
  - Metoda `stopSelf()`

Spolupráce aktivity a služby



Obrázek: Autor: Lukáš Novák

Předávání dat

- Zatím umíme předat data z aktivit do služeb
  - Pomocí intentu
- Druhý směr je složitější
  - Proč?
- Spouštěná služba nemá žádnou vazbu na aktivitu která ji spustila
  - I kdyby ji měla, aktivita již může být dávno mrtvá
- Máme několik možností
  - Broadcast receivers
  - Notifikace
  - Knihovny třetích stran (EventBus, . . .)

Broadcast receivers

- V systému mohou nastat *události* (např. přijetí SMS, vybití baterie, hovor, . . .)
- Princip *publish/subscribe* – aplikace mohou vyvolat událost nebo se přihlásit k jejímu příjmu
- Nastane-li událost je vysílána(broadcast) a mohou ji zachytit Broadcast receivers
- Nemají UI, mohou vyvolat notifikaci do status baru
- Další vstupní bod do aplikace
- Vzniká rozšířením třídy `BroadcastReceiver`
- Vysílání je doručeno jako `Intent` (záměr)
- Tento mechanismus lze použít pro komunikaci mezi službou a aktivitou

Přihlášení k odběru událostí (1 / 2)

- Vytvoříme anonymní implementaci abstraktní třídy `BroadcastReceiver`

```
private val receiver = object : BroadcastReceiver() {
    override fun onReceive(ctx: Context, intent: Intent) {
    Log.w("Warn", "Obdrzeno")
    when(intent.action) {
      Intent.ACTION_TIME_TICK -> Log.w("Warn","tik hodin")
      MyService.WORK_COMPLETE -> {Log.w("warn", "moje zprava")
                    // get data
                }
            }
        }
    }
```

- Při obdržení události se spustí metoda `onReceive`
- Spuštěna v hlavním vlákně
- „životnost“ pouze po dobu vykonávání `onReceive` (nevytvářet vlákna)
- Data nalezneme (opět) v intentu

Přihlášení k odběru události (2 / 2)

- K odběru událostí se přihlašujeme v metodě `onResume()`
- Pomocí `IntentFilter` definujeme, na které události se má reagovat

```
override fun onResume() {
    super.onResume()
    val filter = IntentFilter()
    filter.addAction(MyService.WORK_COMPLETE)
    filter.addAction(Intent.ACTION_TIME_TICK)
    registerReceiver(receiver, filter)
}
```

- V metodě `onPause()` se musíme z odběru odhlásit

```
override fun onPause() {
    super.onPause()
    unregisterReceiver(receiver)
}
```

Lokální události

- Citlivá data aplikace by ji neměly opustit
- Nebudeme je tedy vysílat „globálně“

Lokální události

- Citlivá data aplikace by ji neměly opustit
- Nebudeme je tedy vysílat „globálně“
- Třída `LocalBroadcastManager`
- Registrace:
  LocalBroadcastManager.getInstance(**this**).registerReceiver(receiver, filter)
- Odhlášení: LocalBroadcastManager.getInstance(**this**).unregisterReceiver(receiver)

Zaslání události

- Vytvoříme `Intent` s danou akcí

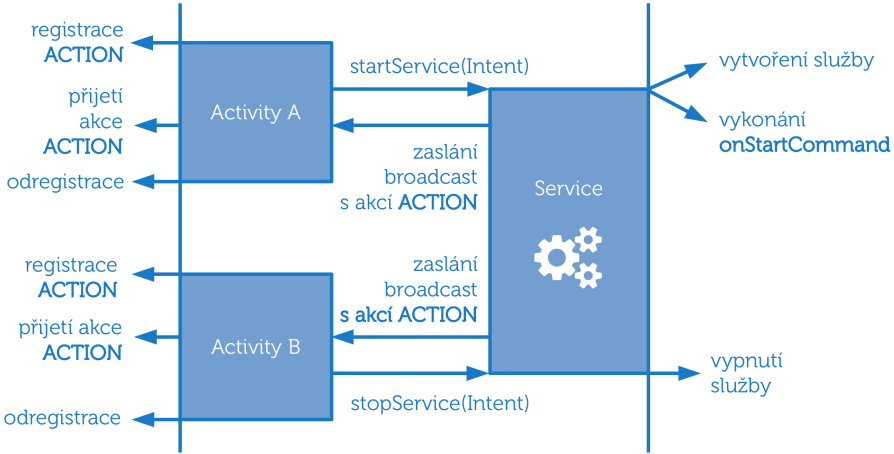
```
val int = Intent(WORK_COMPLETE)
int.putExtra(DATA,notes)
```

- A odešeleme
  - Lokální vysílání:

    LocalBroadcastManager.getInstance(**this**).sendBroadcast(int)
  - Globální vysílání:

    sendBroadcast(int)

Spolupráce aktivit a služeb



Obrázek: Autor: Lukáš Novák

Notifikace

- = zobrazení upozornění uživateli
  - o průběhu práce na pozadí
  - o dokončení nějaké činnosti
  - události potřebující uživatelovu pozornost (zpráva, alarm, . . .)

Notifikace

- = zobrazení upozornění uživateli
  - o průběhu práce na pozadí
  - o dokončení nějaké činnosti
  - události potřebující uživatelovu pozornost (zpráva, alarm, . . .)
- Variabilita možností zobrazení (obrázky, ovládací tlačítka, množství textu, . . .)

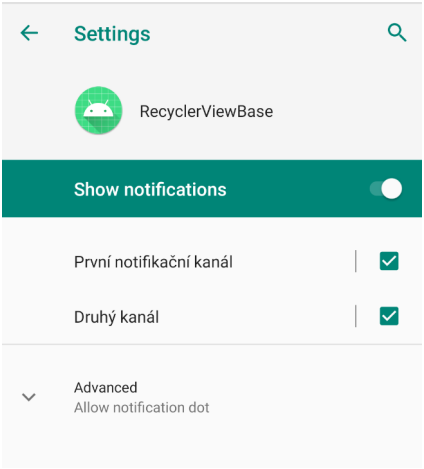
Notifikace

- = zobrazení upozornění uživateli
  - o průběhu práce na pozadí
  - o dokončení nějaké činnosti
  - události potřebující uživatelovu pozornost (zpráva, alarm, . . .)
- Variabilita možností zobrazení (obrázky, ovládací tlačítka, množství textu, . . .)
- Notifikace jsou vysílány v tzv. *kanálech* (od verze 8.0)
- Notifikační kanály mají jméno, popise, prioritu
  - Sdružují notifikace podobného typu
  - Uživatel může v nastavení aplikace vypínat notifikační kanály zvlášť

Notifikace

- = zobrazení upozornění uživateli
  - o průběhu práce na pozadí
  - o dokončení nějaké činnosti
  - události potřebující uživatelovu pozornost (zpráva, alarm, . . .)
- Variabilita možností zobrazení (obrázky, ovládací tlačítka, množství textu, . . .)
- Notifikace jsou vysílány v tzv. *kanálech* (od verze 8.0)
- Notifikační kanály mají jméno, popise, prioritu
  - Sdružují notifikace podobného typu
  - Uživatel může v nastavení aplikace vypínat notifikační kanály zvlášť
- Na starších verzích < 8 udáváme prioritu u každé notifikace zvlášť
  - Tabulka: `https://developer.android.com/training/notify-user/channels#importance`
- Kanál musí být založen před prvním odesláním notifikace

Nastavení notifiakčních kanálů



Obrázek: Notifikační kanály lze vypínat v nastavení aplikace

Vytvoření notifikačního kanálu

- Do metody `onCreate()` aplikace přidáme volání `createNotificationChannels()`:

```
private fun createNotificationChannels() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        val name = getString(R.string.firstChannel)
        val descriptionText = getString(R.string.channelDescription)
        val importance = NotificationManager.IMPORTANCE_DEFAULT
        val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
            description = descriptionText
    }
    val notificationManager: NotificationManager =
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
        notificationManager.createNotificationChannel(channel)
    }
}
```

Vytvoření notifikace (1 / 2)

- Notifikaci vytvoříme pomocí `NotificationCompat.Builder`

```
var builder = NotificationCompat.Builder(app.applicationContext,
    MyApplication.CHANNEL_ID)
    .setSmallIcon(R.drawable.ic_baseline_baby_changing_station_24)
    .setContentTitle("Moje prvni notifikace")
    .setContentText("Trochu delsi text nez titulek")
    .setStyle(NotificationCompat.BigTextStyle()
        .bigText("A toto je mnohem delsi text, ktery se uzivateli
            zobrazi pri kliku na sipecku"))
    . setPriority (NotificationCompat.PRIORITY_DEFAULT)
    .setContentIntent(pendingIntent)
    .setAutoCancel(true)
```

- Pro přehled různých vzhledů a funkcí doporučují projít dokumentaci

Vytvoření notifikace (2 / 2)

- Builder umí přidat *PendingIntent*, který obaluje *Intent*, který se provede při kliku na notifikaci
  - Přidává context, `request code` a *flag* jaké má mít *PendingIntent* vlastnosti

```
val intent = Intent(app.applicationContext, MainActivity::class.java)
val pendingIntent: PendingIntent =
    PendingIntent.getActivity(app.applicationContext, 0, intent,
    PendingIntent.FLAG_IMMUTABLE)
```

Vytvoření notifikace (2 / 2)

- Builder umí přidat *PendingIntent*, který obaluje *Intent*, který se provede při kliku na notifikaci
  - Přidává context, `request code` a *flag* jaké má mít *PendingIntent* vlastnosti

```
val intent = Intent(app.applicationContext, MainActivity::class.java)
val pendingIntent: PendingIntent =
    PendingIntent.getActivity(app.applicationContext, 0, intent,
    PendingIntent.FLAG_IMMUTABLE)
```

- Notifikaci vyvoláme s unikátním ID

```
with(NotificationManagerCompat.from(app.applicationContext)) {
            notify(42, builder.build())
}
```

- Na ID se můžeme odkazovat při pozdější editaci/smazání existující notifkace

Modifikace notifikace

- Modifikaci lze provést sestavením upravené notifikace
- Použití funkce `notify()` se stejným ID
  - Exitující notifikace se modifikuje
  - Pokud neexistuje, vznikne nová
- Zrušení notifikace:
  - Zavoláním metody cancel(id)

```
with(NotificationManagerCompat.from(app.applicationContext)) {
    cancel(42)
}
```

  - Případně cancelAll()
  - Vypršení času (při použití setTimeoutAfter()
  - Uživatelem (klik, swipe)

Úkol

1. Do nastavení vaší aplikace přidejte další switch: „Stahovat poznámky z internetu“
2. Dodělejte zobrazení poznámek z internetu
   - Při gestu „aktualizovat“(swipe shora) dojde k načtení poznámek z internetu
   - Při načítání se objeví „točící se kolečko“
   - Celé se to povede pouze pokud je v nastavení povoleno, dojde ke spojení poznámek z DB a z internetu
   - Rada: SwipeRefreshLayout, binding, korutiny
3. Vytvořte službu, která bude pravidelně načítat nové poznámky z webu
   - K načítání bude docházet pouze při povolení v nastavení
4. Vyzkoušejte si vytvoření notifikace (např. při stažení nového seznamu službou)