LUKÁŠ NOVÁK

KMI / TMA

Tvorba mobilních aplikací

8. SEMINÁŘ | 16.11.2017

ZS 2017/2018 | ČTVRTEK 13:15-15:45

OBSAH SEMINÁŘE

Komunikace napříč aplikací,
Práce na pozadí II.,
notifikace

PRÁCE NA POZADÍ II.

Něco lepšího než AsyncTask?

- pro flexibilnější práci na pozadí využijeme tzv. služby, které zajišťuje třída **Service**
- používají se pro práci na pozadí, která není závislá na aktivitě, např. stahování souboru, přehrávání hudby bez aktivity, aj.
- mají svůj životní cyklus podobně jako aktivity

PRÁCE NA POZADÍ II.

Vytvoření služby

- službu vytvoříme zděděním třídy **Service**
- implementujeme **onStartCommand**, která se vykoná, pokud službu spustíme
- je třeba implementovat **onBind**, většinou může vracet **null**

PRÁCE NA POZADÍ II.

SPUŠTĚNÍ SLUŽBY

- **onStartCommand** musí vracet jednu z konstant, které určují, jak se služba bude chovat, pokud je proces aplikace ukončen při běhu služby
  - **Service.START_STICKY**
  - **Service.START_REDELIVER_INTENT**
  - **Service.START_NOT_STICKY**

PRÁCE NA POZADÍ II.

Vytvoření služby

```
public class MyService extends Service {

    @Override
    public IBinder onBind(Intent intent) {

        return null;

    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        // do background work

        return Service.START_NOT_STICKY;

    }

}
```

PRÁCE NA POZADÍ II.

Definice služby v manifestu

- službu musíme definovat v manifestu
  atribut android:name ukazuje na
  cestu k potomku třídy **Service**

```
<service
    android:name=".work.MyService" />
```

PRÁCE NA POZADÍ II.

SPUŠTĚNÍ SLUŽBY

- zavoláme **startService(intent)**, kde intent je instance **Intentu**, který definuje jakou třídu chceme spustit, případně ji předává argumenty (pomocí metod **put***)
- vykonání je asynchronní, služba je spuštěna na pozadí, nicméně **onStartCommand** je vykonáno v UI vlákně, doporučuje se ve službě vytvořit nové vlákno

PRÁCE NA POZADÍ II.

Ukončení služby

- odkudkoli můžeme zavolat metodu **stopService(intent)**, kde intent definuje třídu služby
- ze služby můžeme zavolat metodu **stopSelf**

```
Intent i = new Intent(this, MyService.class);
i.putExtra(MyService.EXTRA_URL, URL);
startService(i);
```

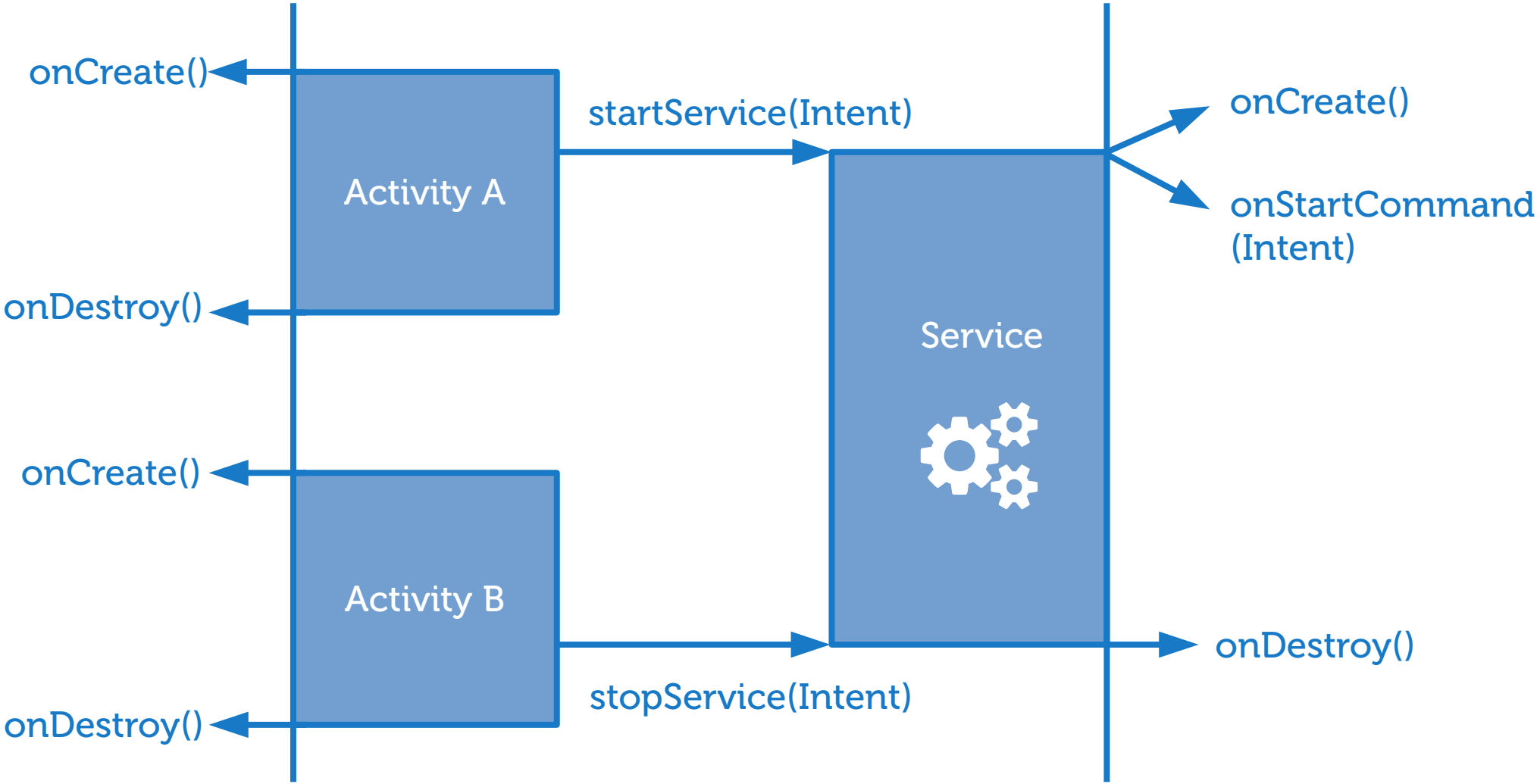
PRÁCE NA POZADÍ II.

PŘEDPŘIPRAVENÁ SLUŽBA

- pro jednodušší použití služeb je k dispozici potomek **Service** = **IntentService**
- používá se pro krátkodobé práce na pozadí, služba se po vykonání sama ukončí
- navíc nabízí metodu **onHandleIntent**, která pracuje v novém vlákně a zpracovává intent předaný v **startService**
- v konstruktoru bez parametrů je nutné zavolat konstruktor předka se jménem služby

PRÁCE NA POZADÍ II.

Spolupráce aktivity a služby



PRÁCE NA POZADÍ II.

OBOUSMĚRNÁ KOMUNIKACE

- pomocí **onStartCommand** nebo **onHandleIntent** předáme data z aktivity do služby
- pokud chceme předat data ze služby do aktivity (např. výsledek), můžeme využít:
  - BroadcastReceiver
  - EventBus, ... (3rd knihovny)
  - notifikace

NASLOUCHÁNÍ UDÁLOSTEM

BroadcastReceiver

- používáme, pokud chceme zpracovávat zajímavé události ze strany systému, např. změna hladiny baterie, změna času, ...
- můžeme také využít pro naše vlastní události z jiné komponenty (**Service**) nebo události z jiné aplikace

NASLOUCHÁNÍ UDÁLOSTEM

Definice IntentFilter

- vytvoříme **IntentFilter**, kterému definujeme, akce typu **String**, kterým chceme naslouchat
- systémové akce např.
  - **Intent.ACTION_TIME_TICK**,
  - **Intent.ACTION_...** (a další)
- můžeme si definovat vlastní události, např. String "my_event"

```
IntentFilter filter = new IntentFilter();
filter.addAction(MyService.ACTION_DONE); // custom action
filter.addAction(Intent.ACTION_TIME_TICK); // system action
```

NASLOUCHÁNÍ UDÁLOSTEM

Vytvoření BroadcastReceiveru

- **BroadcastReceiver** je objekt, který bude události zpracovávat
- vytvoříme instanci jako členskou proměnnou, ve které je nutné implementovat metodu **onReceive**, která nám předá **Context** a přijatý **Intent**

NASLOUCHÁNÍ UDÁLOSTEM

Registrace globálních událostí

- **BroadcastReceiver** zaregistrujeme v **onResume** pomocí **registerReceiver**
- nesmíme zapomenout na odregistrování v **onPause** pomocí **unregisterReceiver**
- mezi **onResume** a **onPause** tak dostáváme globální události ze systému, ale i z jiných aplikací

NASLOUCHÁNÍ UDÁLOSTEM

Registrace lokálních událostí

- pokud nás zajímají pouze „naše“ lokální akce, potom použijeme pro registraci a odregistraci **LocalBroadcastManager**

```
LocalBroadcastManager.getInstance(this)
        .registerReceiver(mReceiver, filter);

LocalBroadcastManager.getInstance(this)
    .unregisterReceiver(mReceiver);
```

NASLOUCHÁNÍ UDÁLOSTEM

```
private BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // do something
    }
};

@Override
protected void onResume() {

    super.onResume();

    IntentFilter filter = new IntentFilter();
    filter.addAction(MyService.ACTION_DONE);

    LocalBroadcastManager.getInstance(this)
        .registerReceiver(mReceiver, filter);

}

@Override
protected void onPause() {

    super.onPause();

    LocalBroadcastManager.getInstance(this)
        .unregisterReceiver(mReceiver);

}
```

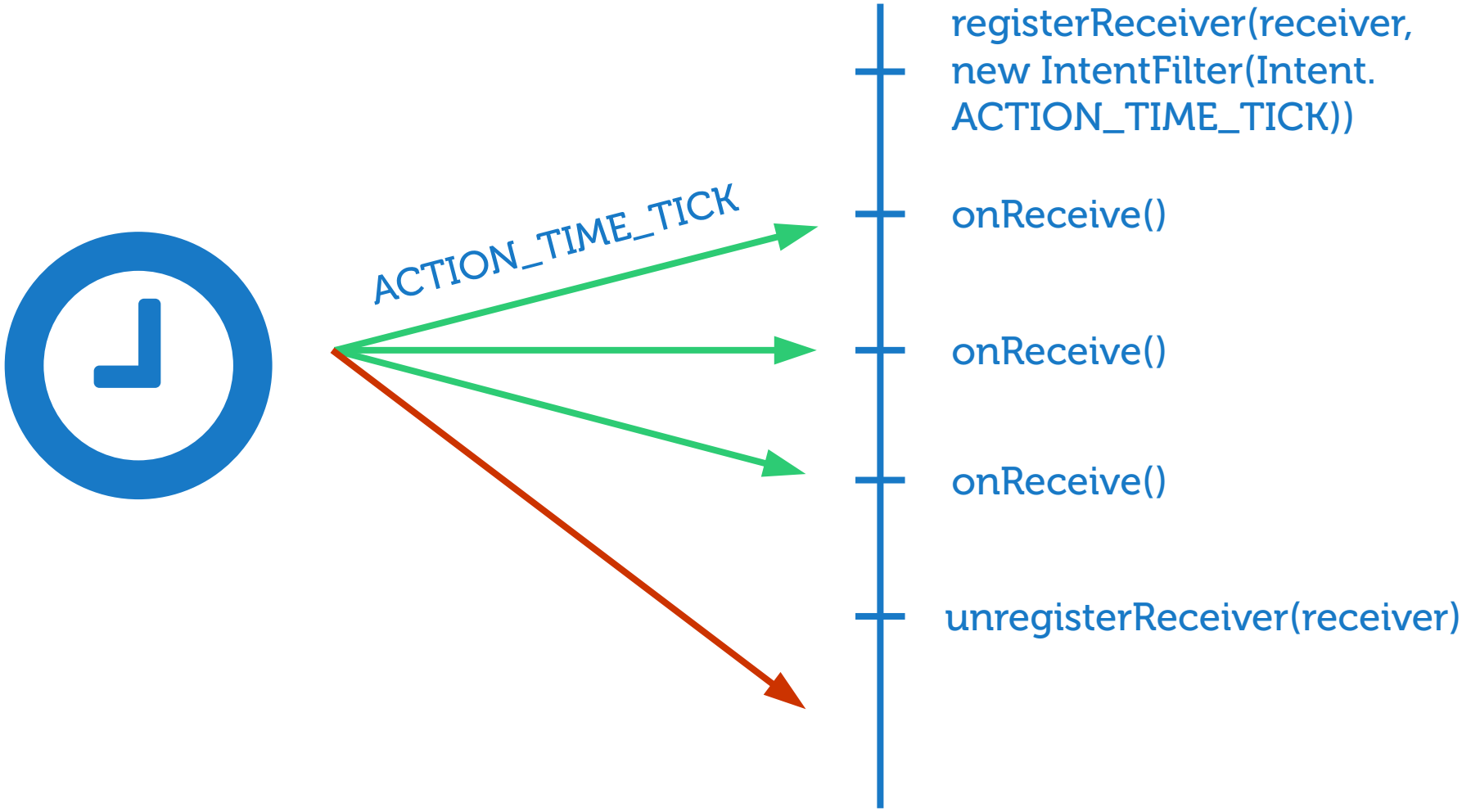
NASLOUCHÁNÍ UDÁLOSTEM

Práce s BroadcastReceiverem

- **BroadcastReceiver** žije pouze po dobu vykonání **onReceive**, nemůžeme tedy např. spustit práci na jiném vlákně
- **onReceive** je zavolána na UI vlákně
- v přijatém **Intentu** některých událostí nalezneme zajímavá data, např. hladinu baterie při **ACTION_BATTERY_CHANGED**

NASLOUCHÁNÍ UDÁLOSTEM

Práce s BroadcastReceiverem



NASLOUCHÁNÍ UDÁLOSTEM

ZASÍLÁNÍ UDÁLOSTÍ

- pro zaslání události vytvoříme prázdný **Intent**, kterému nastavíme akci, které naslouchám
- můžeme předat data zpět pomocí přidání „extra“ dat

```
Intent i = new Intent();
i.setAction(MY_ACTION);
i.putExtra(DATA, data);
```

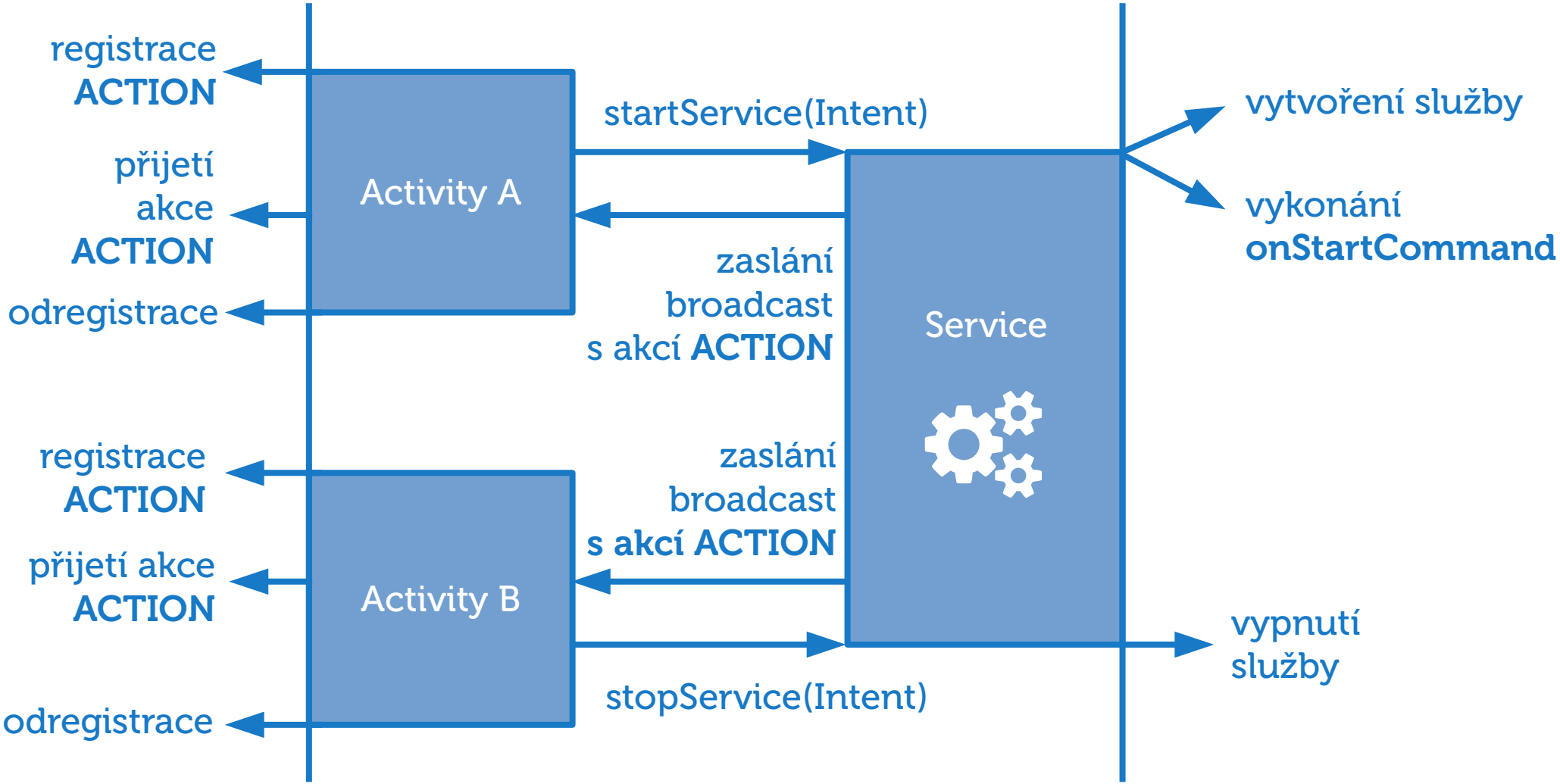
NASLOUCHÁNÍ UDÁLOSTEM

Zasílání událostí

- zaslání události:
  - globální události
    - **sendBroadcast(Intent i)**
  - lokální události
    - **LocalBroadcastManager.**
      getInstance(Context context)
      .sendBroadcast(Intent i)

PRÁCE NA POZADÍ II.

Spolupráce aktivity a služby



NOTIFIKACE VE STATUS BARU

Notification

- zobrazení upozornění často o průběhu nebo výsledku práce na pozadí
- mnoho možností vzhledu a funkcí notifikací, např. notifikace s ovládaním hudebního přehrávače
- použití ve spojení s push notifikacemi ze vzdáleného serveru

NOTIFIKACE VE STATUS BARU

Vytvoření notifikace

- vytvoření instance builderu a pomocí **set*** metod nastavení vlastností notifikace

```
Notification notification = builder
    .setContentTitle("Title")
    .setContentText("Content")
    .setAutoCancel(true)
    .setSmallIcon(R.drawable.ic_file_download_white_18dp)
    .setContentIntent(pi)
    .build();
```

NOTIFIKACE VE STATUS BARU

NASTAVENÍ NOTIFIKACE

- **builder.setContentTitle(...)**
  - titulek notifikace, String nebo stringRes
- **builder.setContentText(...)**
  - text notifikace, String nebo stringRes
- **builder.setSmallIcon(...)**
  - malá ikona ve status baru, drawableRes

NOTIFIKACE VE STATUS BARU

NASTAVENÍ NOTIFIKACE

- **builder.setContentIntent(...)**
  - akce, která se provede po kliku na notifikaci
  - přijímá **PendingIntent**
    - ten vytvoříme z klasického **Intent**
      - **PendingIntent.getActivity (this, 0, intent, 0)**

NOTIFIKACE VE STATUS BARU

NASTAVENÍ NOTIFIKACE

- **builder.setAutoCancel**
  - booleovská hodnota, pokud je true, potom notifikace po kliku provede akci a zmizí
- **builder.setDefaults**
  - nastavení upozornění, jedna z konstant
  - Notification.DEFAULT_LIGHTS, DEFAULT_
  - SOUND, DEFAULT_VIBRATE, DEFAULT_ALL

NOTIFIKACE VE STATUS BARU

NASTAVENÍ NOTIFIKACE

- **builder.setProgress** a mnoho dalšího, viz dokumentace
- po nastavení všech vlastností nutné zavolat metodu **build** pro získání instance **Notification**

NOTIFIKACE VE STATUS BARU

NASTAVENÍ NOTIFIKACE

- pro zobrazení notifikace je nutné získat instanci systémové služby **NotificationManager**

```
NotificationManager mgr = (NotificationManager)
    getSystemService(NOTIFICATION_SERVICE);
```

- a notifikaci zobrazíme pomocí **notify**
- typ notifikace definujeme identifikátorem typu **int**

```
mgr.notify(NOTIFICATION_ID, notification);
```

NOTIFIKACE VE STATUS BARU

NÁSLEDNÁ PRÁCE S NOTIFIKACÍ

- identifikátor slouží pro
  - zrušení notifikace pomocí metody **cancel**
  - pro zrušení všech notifikací slouží metoda **cancelAll**
  - nahrazení notifikace zobrazením jiné notifikace se stejným identifikátorem

ÚKOL 8. SEMINÁŘE

1) přepsat stahování TODO položek z **AsyncTask** s použitím **Service**, tj. spuštění služby, stáhnutí na pozadí a zaslání události o dokončení do aktivity
2) při začátku stahování zobrazit notifikaci s progress barem
3) po úspěchu zobrazit notifikaci, která nahradí notifikaci stahování
4) při chybě (např. vypnutém internetu) zobrazit notifikaci, která nahradí notifikaci stahování

Tipy pro řešení:

1) ve službě nemáte přístup k aktivitě, služba by tedy neměla znát implementaci aktivity, ale měla by získat vstupní parametry, zpracovat akci a informovat všechny příjemce o provedení akce nezávisle na tom, kdo události naslouchá
2) služba by se měla starat o vše spojené se stahováním, tedy i zobrazení notifikací
3) čisté řešení je, že aktivita po přijetí akce pouze aktualizuje seznam

OTÁZKY

Ptejte se!