9. PRÁCE NA POZADÍ

BI-AND

9. přednáška

- Práce na pozadí
- Service
- Notifications

Práce na pozadí

- Jeden z hlavních rozdílů oproti iOS je možnost provádět téměř libovolné úkony na pozadí

- Vlákna
  - Používají se ke spouštění kódu na pozadí, zatímco uživatel interaguje s UI (načítání stránek ve webovém prohlížeči, náročné výpočty)

- Service
  - Používají se pro pravidelné a „nepřetržité“ provádění akci, které nevyžadují UI (např. hudební přehrávač, automatické stahování…)

Práce na pozadí

- Aplikace musí vždy odpovědět do pěti sekund, jinak Android nabídne uživateli, aby ji ukončil
- Activity, Service a Broadcast Receivers běží v hlavním (UI) vlákně aplikace, kde je aplikován limit pěti (resp. deseti) sekund
- Pro vytváření vláken slouží `Thread` nebo `AsyncTask`

Thread

- Jednoduchý způsob, jak něco provést na pozadí
- Stejné chování jako u Java SE

```
1.  new Thread(){
2.      public void run() {
3.              ...
4.      }
5.  }.start();
```

Handler

- Pouze hlavní vlákno může přistupovat k UI
- Pro spuštění kódu z jiného vlákna ve hlavním vlákně slouží `Handler`
  - Funguje na bázi odesílání a přijímání zpráv

- Přijímání zpráv v hlavním vlákně:

```
1.  private Handler mHandler = new Handler() {
2.      //nasledujici kod se spusti v hlavnim (GUI) vlakne
3.      public void handleMessage(Message msg) {
4.              Bundle bundle = msg.getData();
5.              myTextView.setText(bundle.getString("text"));
6.      };
7.  };
```

Handler

- Odeslání zprávy z jiného vlákna
  - Musíme mít referenci na vytvořený Handler
  - Metoda `mHandler.obtainMessage()`
    - Efektivnější než vytváření nové zprávy
  - Lze předat dva int argumenty (arg1 a arg2) u `Message` bez nutnosti vytvářet Bundle

```
1.  new Thread() {
2.     public void run() {
3.        ...
4.        Message msg = mHandler.obtainMessage();
5.        Bundle b = new Bundle();
6.        b.putString("text", "novy text");
7.        msg.setData(b);
8.        mHandler.sendMessage(msg);
9.     }
10. }.start();
```

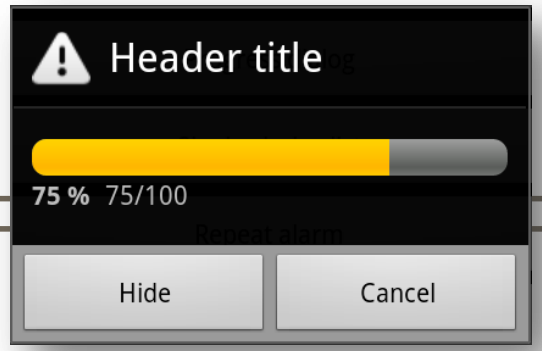
ProgressDialog

```
1.protected Dialog onCreateDialog(int id) {
2.  …
3.   case DIALOG_PROGRESS:
4.      mProgressDialog = new ProgressDialog(AlertDialogSamples.this);

5.      mProgressDialog.setIcon(R.drawable.alert_dialog_icon);
6.      mProgressDialog.setTitle(R.string.select_dialog);
7.      mProgressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
8.      mProgressDialog.setMax(MAX_PROGRESS);

9.      mProgressDialog.setButton(DialogInterface.BUTTON_POSITIVE,
10.         getText(R.string.alert_dialog_hide), new DialogInterface.OnClickListener() {
11.               public void onClick(DialogInterface dialog, int whichButton) {
12.               }
13.      });

14.      mProgressDialog.setButton(DialogInterface.BUTTON_NEGATIVE,
15.         getText(R.string.alert_dialog_cancel), new DialogInterface.OnClickListener() {
16.               public void onClick(DialogInterface dialog, int whichButton) {
17.               }
18.      });
19.      return mProgressDialog;
20.  …
21.}
```

ProgressDialog

```
public void onClick(View v) {
      showDialog(DIALOG_PROGRESS);
      mProgress = 0;
      mProgressDialog.setProgress(0);
      mProgressHandler.sendEmptyMessage(0);
}
```

Dialog je možné ovládat přes Handler




```
mProgressHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        if (mProgress >= MAX_PROGRESS) {
            mProgressDialog.dismiss();
        } else {
            mProgress++;
            mProgressDialog.incrementProgressBy(1);
            mProgressHandler.sendEmptyMessageDelayed(0, 100);
        }
    }
};
```

Na zprávu Handler zareaguje zvýšením čítače pokroku

AsyncTask

- Zjednodušuje volání metod v hlavním vlákně
- Kombinace metod, z nichž jedna běží ve vedlejším vlákně a průběžné výsledky se spouští v hlavním
- `AsyncTask<Params, Progress, Result>` je definovaný třemi generickými typy:
  - Params – typ parametru nutný pro vykonání operace
  - Progress – typ parametru použitý při „zobrazování“ pokroku
  - Result – typ výsledku

AsyncTask

- `AsyncTask<Params, Progress, Result>` má 4 základní metody:
  - `void onPreExecute()` – provede kód v hlavním vlákně těsně před tím, než se spustí vedlejší
  - `result doInBackground(Params... params)` – zde se spustí kód na pozadí
  - `void onPostExecute(Result result)` – spustí se v hlavním vlákně, jakmile se `doInBackground` ukončí
  - `void onProgressUpdate(Progress... values)` – spustí se když, se v `doInBackground` zavolá `publishProgress(progress);`
- Spustí se zavoláním `asyncTask.execute(params);`
  - Musí provést hlavní vlákno

Zrušení AsyncTask

- Může být zrušen kdykoliv pomocí `cancel()`
- Metoda `isCancelled()` začne vracet `true`
- Místo `onPostExecute()` je zavolána metoda `onCancelled()`
- Toto zavolání proběhne až po skončení `doInBackground()`
- Pokud je to možné, kontrolovat (např. ve smyčce) v `doInBackground()`, jestli nebyl AsyncTask zrušen

AsyncTask

```
1. myTask = new AsyncTask<String, Integer, Double>() {
2.     @Override
3.     protected void onPreExecute() {
4.           myTextView.setText("Loading...");
5.           super.onPreExecute();
6.     }
7.     @Override
8.     protected Double doInBackground(String... params) {
9.           Double result;
10.          int progress;
11.          ...
12.          publishProgress(progress);//progress se preda do onProgressUpdate
13.          ...
14.          return result;//preda se do onPostExecute
15.    }
16.    @Override
17.    protected void onProgressUpdate(Integer... values) {
18.          myTextView.setText("Running, progress= " + values[0]);
19.          super.onProgressUpdate(values);
20.    }
21.    @Override
22.    protected void onPostExecute(Double result) {
23.          myTextView.setText("Finished, result= " + result);
24.          super.onPostExecute(result);
25.    }
26.};
27.myTask.execute("param");//param se preda do doInBackground
```

Použití např. pro načítání obrázků do ListView/ViewPagger

Změna konfigurace Activity

- Activita, která vytvořila vlákno, prochází životním cyklem a díky změně konfigurace může dojít k jejímu zničení a znovu vytvoření

- Možná řešení
  - Zakázat změnu konfigurace pomocí AndroidManifestu a případně přetížení metody `onConfigurationChanged()` NEPOUŽÍVAT!!!
  - Použít metodu `onRetainNonConfigurationInstance()` a `getLastNonConfigurationInstance()`

Změna konfigurace Activity

- Do AsyncTask přidáme metody pro získání aktuálního contextu activity
  - Attach – předání nového contextu
  - Detach – znevalidnění starého contextu

Co se stane, když AsyncTask skončí během změny konfigurace?

- AsyncTask je uložen pomocí metody `onRetainNonConfigurationInstance()`
- V onCreate zkontrolujeme pomocí `getLastNonConfigurationInstance()`, zdali již AsyncTask neběží a předáme mu nový context activity

Service

- Vhodné, pokud potřebujeme provést úkon nezávislý na UI
  - Přehrávání hudby
  - Stahování velkého objemu dat
  - Pravidelně se opakující akce
  - Spolupráce s Web Service
  - Spolupráce s App Widgety

Kdy použít Thread nebo Service?

- Běží v hlavním vlákně
  - Ve většině případů je nutné vytvořit další vlákno

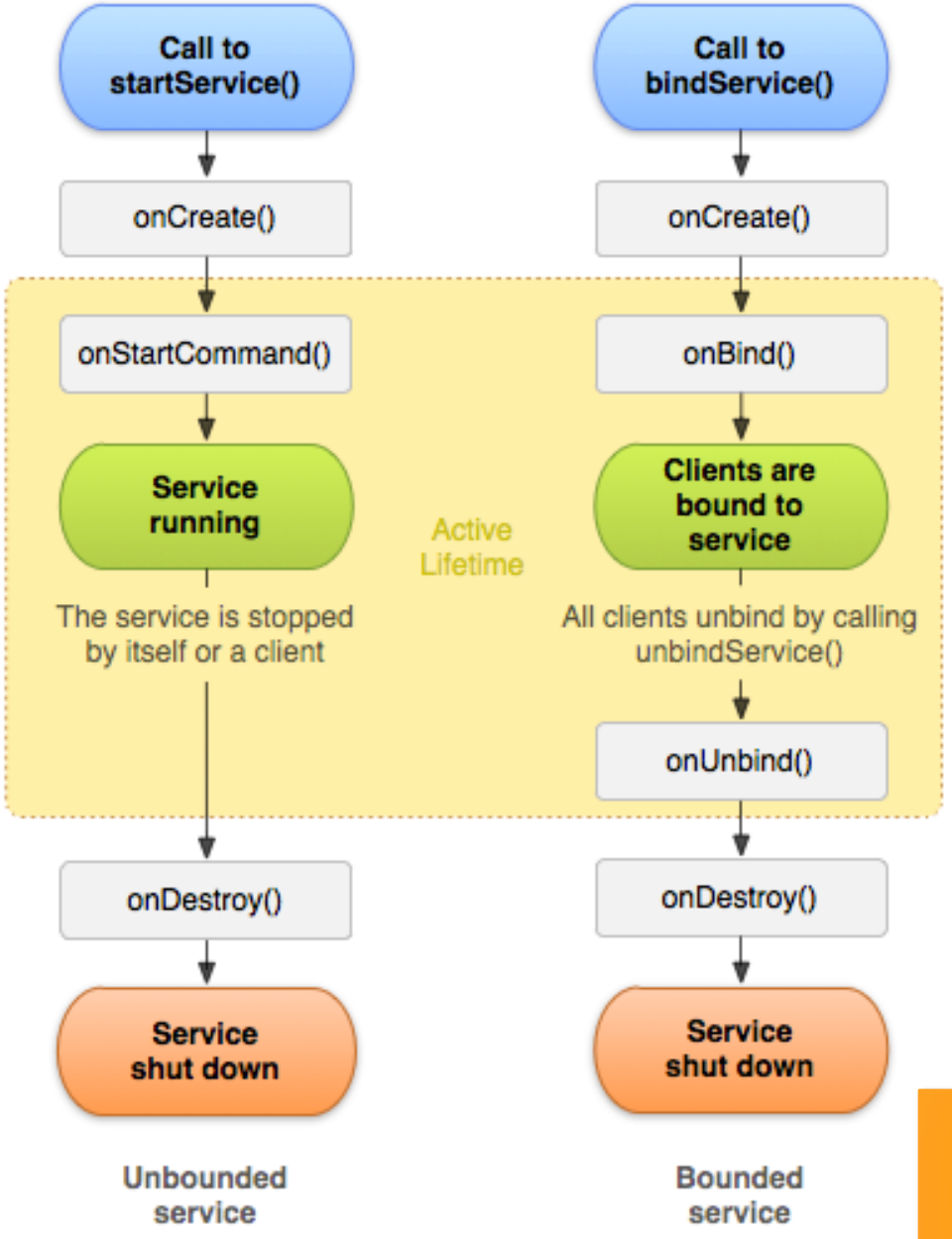
- Není vytvořena ve vlastním procesu
  - Lze změnit v manifestu

Service

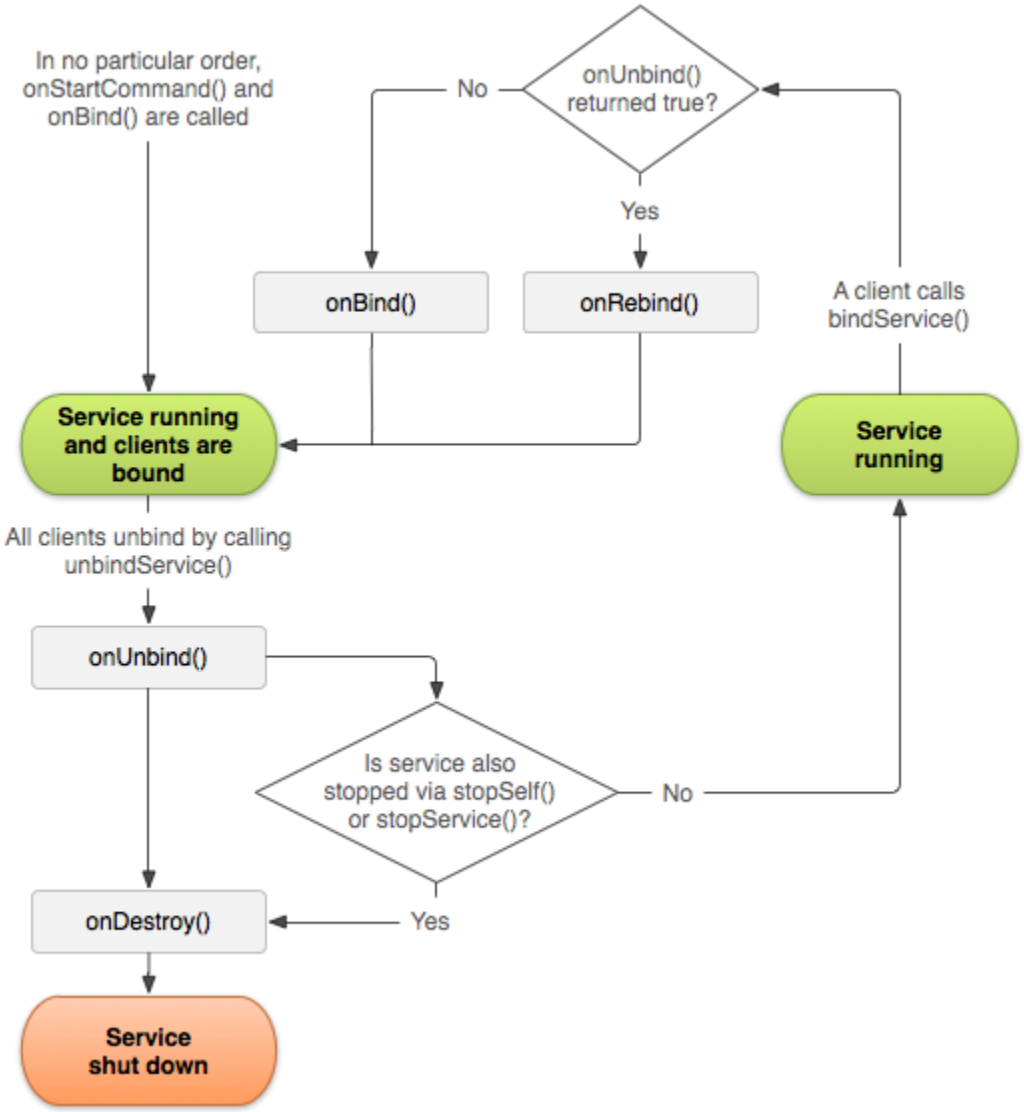
- Méně náchylné k násilnému ukončení systémem než activity na pozadí
- Pokud jsou přeci jen ukončeny, systém je po získání potřebných prostředků restartuje
- Jsou spouštěny, zastavovány a kontrolovány z jiných komponent aplikace (např. jiné Service, Activity nebo BroadcastReceiveru)
- Je nutné je registrovat v AndroidManifestu

Životní cyklus Service

- `onCreate()` – zavoláno, když je Service vytvořena jakýkoliv způsobem
- `onStartCommand()` – zavoláno, když je vyvolána metoda `startService()`
- `onBind()` – zavoláno, pokud klient zavolá bindService()
- `onDestroy()` – zavoláno, když má být Service odstraněna
  - Není zaručeno, že tato metoda bude zavolána



Lze bindovat started/unbouded Service?


In no particular order, onStartCommand() and onBind() are called
No
onUnbind() returned true?
Yes
onBind()
onRebind()
A client calls bindService()
Service running and clients are bound
Service running
All clients unbind by calling unbindService()
onUnbind()
Is service also stopped via stopSelf() or stopService()?
No
Yes
onDestroy()
Service shut down

Service

MyService.java

```
1. public class MyService extends Service{
2.     @Override
3.     public int onStartCommand(Intent intent, int flags, int startId) {
4.         //provedeni potrebneho ukonu
5.         return START_STICKY;
6.     }
7.     @Override
8.     public IBinder onBind(Intent intent) {
9.         return null;
10.    }
11. }
```

MainActivity.java

```
1. startService(new Intent(this, MyService.class));
2. stopService(new Intent(this, MyService.class));
```

AndroidManifest

```
1. <service android:enabled="true" android:name=".MyService" />
```

Service – onStartCommand()

- `onStartCommand()` vrací konstanty, které určují, jak se bude služba chovat, když ji systém ukončí, kvůli nedostatku prostředků
  - Zavedeno od Android 2.0
  - Dříve používaná metoda `onStart()` je deprecated

- *`START_STICKY`*
  - Standardní chování
  - Služba se vždy restartuje, ale Intent v `onStartCommand()` bude `null`
  - Typické pro služby, které jsou explicitně spouštěny a zastavovány pomocí `startService()` a `stopService()`

Service – onStartCommand()

- *START_NOT_STICKY*
  - Služba se restartuje, pouze pokud čekají na vyřízení další požadavky. Jinak zůstane služba zastavena
  - Typické pro služby, které se spouští periodicky a po skončení požadovaného úkonu se samy ukončí pomocí `stopSelf()`

- *START_REDELIVER_INTENT*
  - Podobné *START_NOT_STICKY*
  - Služba se restartuje, pouze pokud čekají na vyřízení další požadavky nebo pokud před ukončením nedokončila služba svoji práci. Byla ukončena před zavoláním `stopSelf`()
  - Po restartu je předán do `onStartCommand()` původní Intent
  - Typické pro služby, kdy je potřeba zajistit vykonání operace

Service – běh

- Na rozdíl od Activity má daná služba vždy maximálně jednu instanci
- Po opětovném zavolání `startService()` se pouze znovu zavolá `onStartCommand()` v již běžící službě
- Service zůstává spuštěna dokud není explicitně zastavena z jiné komponenty pomocí `stopService()`, nezavolá `stopSelf()` nebo není ukončena systémem

Service – flags

- Pro zjištění, jak se Service spustila, slouží atribut `flags`, který může nabývat těchto hodnot:

- *`START_FLAG_REDELIVERY`* – indikuje, že Intent je znovu doručením předchozího Intentu (*`START_REDELIVER_INTENT`*), ale Service byla ukončena před zavoláním `stopSelf()`

- *`START_FLAG_RETRY`* – indikuje, že Service byla restartována systémem (kvůli nedostatku prostředků) a měla nastaveno *`START_STICKY`*

Provázání z Activity do Service

- `startService()`
  - Jako parametr je Intent s potřebnými Extras
  - Volání je asynchronní
  - Je zavolána metoda `onCreate()`
  - Intent je předán do metody `onStartCommand()`
  - Služba běží dokud není (explicitně) zastavena

Provázání z Activity do Service

- `bindService()`
  - Způsob, jak vystavit určité API služby ostatním komponentám (klientům)
  - Volání je asynchronní
  - Služba vrací tzv. `IBinder`, který reprezentuje určitý objekt ze služby
  - Současně je potřeba vytvořit `ServiceConnection` v daném klientovi
  - Pokud je jako parametr předáno *`BIND_AUTO_CREATE`*, je vytvořena nová služba, jinak se vrací `false`
    - Neprovede se však operace `onStartCommand()`

Provázání z Activity do Service

- V dané službě vytvoříme třídu která dědí od Binder a vrací odkaz na danou instanci služby
- Odkaz na instanci této třídy vrátíme v `onBind`

```
1.  private final IBinder binder = new MyBinder();
2.  public class MyBinder extends Binder {
3.      MyService getService() {
4.              return MyService.this;
5.      }
6.  }
7.  @Override
8.  public IBinder onBind(Intent intent) {
9.      return binder;
10. }
```

Provázání z Activity do Service

- V activitě vytvoříme třídu dědící od `ServiceConnection`

```
1.   // odkaz na sluzbu
2.   private MyService service;
3.   // ma na starosti spojeni mezi sluzbou a activitou
4.   private ServiceConnection mConnection = new ServiceConnection() {
5.       // spusti se jakmile se povede navazat spojeni
6.       public void onServiceConnected(ComponentName className, IBinder service) {
7.            //ziskame instanci nasi sluzby
8.            service = ((MyService.MyBinder) service).getService();
9.       }
10.      // spusti se jakmile se spojeni ukonci
11.      public void onServiceDisconnected(ComponentName className) {
12.           service = null;
13.      }
14.  };
```

- Její instanci poté předáme do `bindService()`

```
1. Intent i = new Intent(this, MyService.class);
2. bindService(i, mConnection, BIND_AUTO_CREATE);
```

Provázání ze Service do Activity

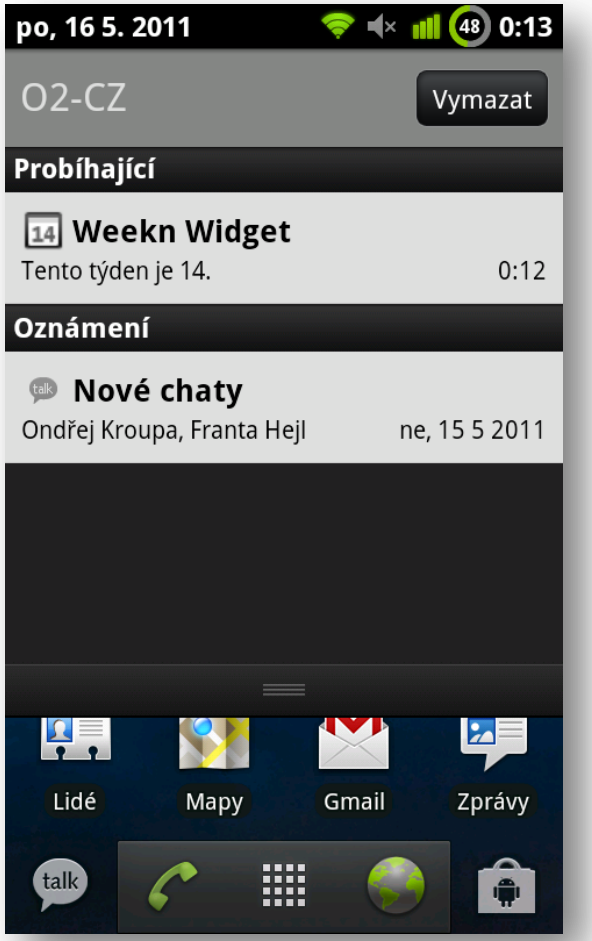
- Callback object nebo Listener
  - Klient poskytne objekt, přes který bude služba komunikovat
  - Pozor na memory leaky a ošetření životního cyklu
- Broadcast Intents
  - Využití Broadcast Receiveru pro zachycení Intentu
- Pending Results
  - Zapouzdření Intentu a vykonání příslušné akce
  - Viz. dále

Provázání ze Service do Activity

- Messenger
  - Zasílání zpráv do Handleru Activity
  - Messenger je Parcelable
  - Viz. 10. přednáška
- Notification
  - Zprávy v notifikační oblasti
  - Viz. dále
- AIDL
  - Běžně služby využívá pouze aplikace, s níž jsou publikovány
  - V opačném případě komunikace probíhá přes IPC s tzv. Remote Service
  - Nemusí vůbec dojít k bindingu
  - Viz. 7. přednáška

Notification

- Slouží k upozornění uživatele na nějakou událost bez použití Activity
- Indikují také probíhající služby (služby, které mají zvýšenou prioritu)
- Mohou mít tyto funkce:
  - Zobrazení ikonky ve Statusbaru
  - Zobrazení podrobnějších informací a spouštění Intentu po rozbalení Statusbaru
  - Blikání notifikačních LED diod, zvukové upozornění a další HW záležitosti



Notification Manager

- Systémová služba starající se o notifikace

- Umožňuje vytvářet nové notifikace, upravovat staré nebo odstranit již nepotřebné

```
1. String nsName = Context.NOTIFICATION_SERVICE;
2. NotificationManager notificationManager;
3. notificationManager = (NotificationManager)getSystemService(nsName);
```

Notification – vytvoření

- Vytvoření notifikace a nastavení základních parametrů

```
1.  // Vybereme ikonku, ktera se bude zobrazovat u notifikace
2.  int icon = R.drawable.icon;
3.  // Text ktery se objevi pri activace notifikace
4.  String tickerText = "Notifikace";
5.  // Cas, podle ktereho se urcuje poradi notifikaci
6.  long when = System.currentTimeMillis();
7.  Notification notification = new Notification(icon, tickerText, when);
8.  //Nastavime pocet, ktery bude zobrazen pres ikonku notifikace
9.  notification.number=10;
```

Notification – rozšiřující informace

- Nastavení defaultního View po rozbalení Statusbaru a Intentu, který se spustí po kliku na notifikaci

```
1.  Context context = this;
2.  // Text ktery se zobrazi po roztahnuti statusbaru
3.  String expandedText = "Podrobny text notifikace";
4.  // Titulek notifikace po roztahnuti statusbaru
5.  String expandedTitle = "Titulek notifikace";
6.  // Intent ktery spusti activitu po kliknuti na notifikaci
7.  Intent intent = new Intent(this, MainActivity.class);
8.  PendingIntent pi = PendingIntent.getActivity(context, 0, intent, 0);
9.  //Nastavime defaultni View
10. notification.setLatestEventInfo(context, expandedTitle, expandedText, pi);
```

- Zobrazení notifikace

```
1.  int id=0;//jedinecne id notifikace v ramci aplikace
2.  notificationManager.notify(id, notification);
```

PendingIntent

- Zapouzdří Intent, který pak může být použit systémem nebo jinými aplikacemi

- Existují 3 typy podle toho, co se následně provede:
  - Spustí activitu (jako zavolání `Context.startActivity(Intent)`)
  - Spustí službu (jako zavolání `Context.startService(Intent)`)
  - Vyšle broadcast (jako zavolání `Context.sendBroadcast()`)

- Obdrží se zavoláním patřičné metody

```
1. pi = PendingIntent.getActivity(context, requestCode, intent, flags);
2. pi = PendingIntent.getService(context, requestCode, intent, flags);
3. pi = PendingIntent.getBroadcast(context, requestCode, intent, flags);
```

Notification – další parametry

- `notification.contentView` – nastavení vlastního layoutu notifikace po rozbalení statusbaru
- `notification.defaults` – možnosti upozornění (*`DEFAULT_LIGHTS`*|*`DEFAULT_SOUND`*|*`DEFAULT_VIBRATE`*)
- `notification.flags` – upřesnění notifikace (viz dokumentace)
- `notification.sound` – Uri souboru pro zvukové upozornění
- `notification.vibrate` – Pole s hodnotame, v jaké frekvenci bude telefon vibrovat
- `notification.ledARGB` – Barva LED diody

Notification – startForeground()

- Ve výchozím nastavení služba běží na pozadí
- Pokud potřebujeme zvýšit prioritu služby je potřeba službu dostat na popředí
- Je téměř zaručeno, že systém tuto službu předčasně neukončí
- Není použit `NotificationManager`, ale speciální metoda, která automaticky po skončení služby odstraní notifikaci

```
1. startForeground(id, notification);
```

Priority procesu

- Pět různých úrovní (priorit) pro proces. Určuje „důležitost“ procesu pro uživatele a ukončení při nedostatku paměti
  - Níže daný výpis je částečně zjednodušen
- Foreground process – např.
  - Activita na popředí (její `onResume()` metoda byla zavolána)
  - Service, která zavolala `startOnForeground()`
  - Service, která je využívána (bounded) activitou na popředí
  - Brodcast Receiver zpracovávající metodu `onReceive()`
- Visible process – stále viditelný uživatelem např.
  - Activita, jejíž `onPause()` metoda byla zavolána
  - Service, která je využívána (bounded) visible activitou

Priority procesu

- Service process
  - Service, která byla nastartována pomocí `startService()` a nespadá do výše uvedených kategorií

Existují další pravidla např. podle provázanosti komponent

- Background process
  - Activita, jejíž `onStop()` metoda byla zavolána
  - Proces neovlivňuje to, co uživatel dělá nebo vidí
  - Pro ukončení se používá LRU algoritmus
  - Důležité správně implementovat životní cyklus

- Empty process
  - Neobsahuje žádné aktivní komponenty
  - Používán pro cachování a zrychlení startup time

Další zdroje

- http://android-developers.blogspot.com/2010/02/service-api-changes-starting-with.html
- http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html