3. ŽIVOTNÍ CYKLUS ACTIVITY

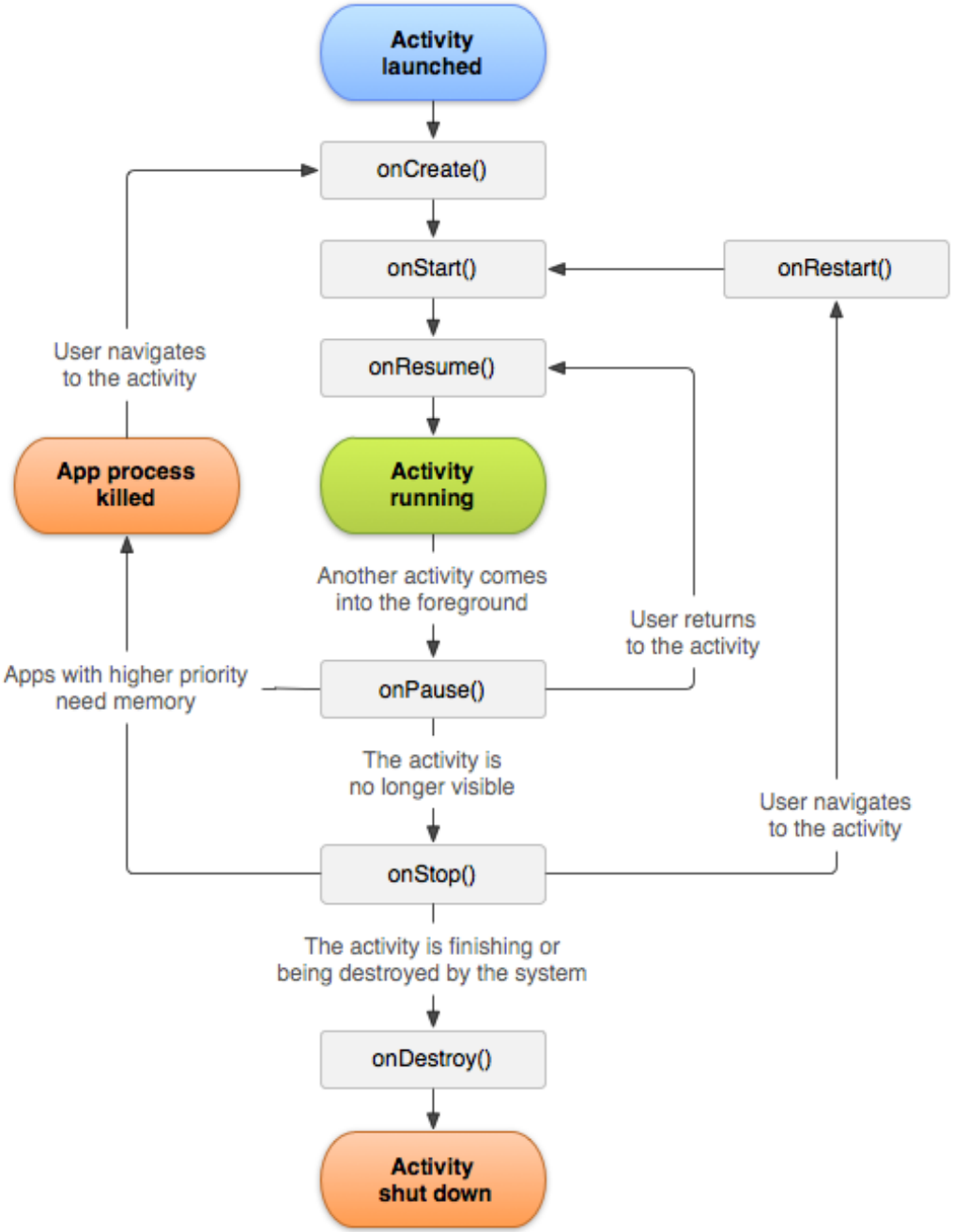
BI-AND

Obsah

- Životní cyklus activity
- Context
- Intent
- Spouštění activit
- Interakce s uživatelem
  - Toast

Stavy activity

- Active
  - Běží, je viditelná, má user focus
- Paused
  - Částečně viditelná, Activity objekt zůstává v paměti
  - Může být ukončena
- Stopped
  - Úplně překryta jinou activitou
  - Přesunuta do pozadí (background)
  - Stále naživu, Activity objekt zůstává v paměti
- Destroyed


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

onCreate(Bundle savedInstanceState) {…}

- První nastartování activity
- Activita běžela, ale byla překryta jinou activitou (příp. aplikací). Uživatel se opět vrátil k původní activitě
  - OS potřeboval paměť (původní proces mohl být zabit)
- Activita běží a je vynucena změna Resources
  - Otočení displeje
  - Zasunutí nebo vysunutí
    - HW klávesnice
    - Z dokovací stanice
  - Změna jazyka

onCreate(Bundle savedInstanceState) {…}

- Zavedení layoutu a inicializace widgetů
  - Nastavení proměnných
  - Nastavení UI
- Jako parametr je `Bundle` – objekt s uloženým předchozím stavem activity (pokud nějaký je)
- Získání dat z volajícího Intentu (pokud je potřeba)
- Následuje `onStart()`

```
1.  @Override
2.  public void onCreate(Bundle savedInstanceState) {
3.        super.onCreate(savedInstanceState);
4.        setContentView(R.layout.main);
5.  }
```

onStart() {…}

- Zavolá se, když se activita stává viditelná uživatelem (po metodě `onRestart()` nebo `onCreate()`)

- Následuje:
  - `onResume()` – pokud se activita dostane do popředí

```
1.  @Override
2.  protected void onStart() {
3.    super.onStart();
4.  }
```

onResume() {…}

- Zavolá se, pokud je activita na popředí
- V tento moment je activita na vrcholu zásobníku
- Activita nyní přijímá vstup od uživatele (má user focus)
- Vždy následuje `onPause()`

```
1.  @Override
2.  protected void onResume() {
3.    super.onResume();
4.  }
```

onRestart() {…}

- Zavolá se, poté, co byla activita zastavená a před tím, než se znovu spustí

- Vždy následuje `onStart()`

```
1.  @Override
2.  protected void onRestart() {
3.    super.onRestart();
4.  }
```

onPause() {…}

- Zavolá se, pokud activita přestane být na popředí
  - Při přechodu na jinou activitu
  - Při zobrazení dialogu
  - Při stisku tlačítka HOME

Dialog vs. dialog

- Po skončení této metody může být proces zabit

- Následuje:
  - `onResume()` – pokud se activita vrátí do viditelného stavu
  - `onStop()` – pokud activita není již viditelná
  - Zabití procesu – pokud systém potřebuje prostředky

onPause() {…}

- Doporučená implementace:
  - Uložení neobnovitelných dat
  - Ukončení všech animací a operací náročných na CPU
  - Přerušení sledování GPS a jiných senzorů
  - Uvolnění různých zámků (screenLock, wakeLock…)

- Operace musí (by měly) být časově nenáročné

```
1.  @Override
2.  protected void onPause() {
3.    super.onPause();
4.    wakeLock.release();
5.    locationManager.removeUpdates(locationListener);
6.    saveChanges();
7.  }
```

onStop() {…}

- Zavolá se, pokud activita již není viditelná
- Activita je na pozadí

Nemusí být nikdy zavolána!

- Následuje:
  - `onRestart()` – Pokud se activita vrátí do viditelného stavu
  - `onDestroy()` – Pokud se activita ukončí
  - Zabití procesu – pokud systém potřebuje prostředky

```
1.  @Override
2.  protected void onStop() {
3.    super.onStop();
4.  }
```

onDestroy() {…}

- Poslední metoda, která se zavolá před tím než je activita ukončena

Nemusí být nikdy zavolána!

- Následuje ukončení activity

```
1.  @Override
2.  protected void onDestroy() {
3.    super.onDestroy();
4.  }
```

Ukládání stavu activity

- `onSaveInstanceState(Bundle outState)`
  - Ukládání stavu activity (proměnné, nastavení widgetů…)
  - Zavoláno předtím, než je **activita** zničena
  - Nezaměňovat s metodou `onPause()`, která je volána vždy
  - Defaultní implementace se stará o většinu UI elementů
    - Musí být nastaveno ID widgetu (např. uložení obsahu `EditText`)
    - Nezahrnuje např. jestli je `Button` enabled nebo disabled
  - Obsah proměnných se ukládá do Bundle `outState`

Ukládání stavu activity

- Obnovení stavu během
  - `onCreate(Bundle savedInstanceState)`
  - `onRestoreInstanceState(Bundle savedInstanceState)`
- Dodán stejný Bundle pro obě metody
- Většinou vše zpracováno v `onCreate()`
- Zavoláno po `onStart()`
- Stav není možné obnovit, pokud byl proces ukončen

Další metody životního cyklu

- `onPostCreate()`
  - Zavoláno po `onStart()` a `onRestoreInstanceState()`
  - „Start-up“ / konfigurace Activity byla dokončena
- `onPostResume()`
  - Zavoláno po `onResume()`
  - Většinou používáno pouze OS a nebývá implementováno vývojářem
- `onUserLeaveHint()` a `onUserInteraction()`
  - Pomáhá rozlišit různé akce uživatete a podle toho např. upravit notifikace

Bundle

Vysoká náročnost zpracování serializace

- Třída sloužící k mapování jakýchkoli *Serializable* nebo *Parcelable* objektů
- Mapuje se pomocí *String* -ových klíčů

Pracnější sepsání implementace Parcelable

Vkládání:

```
1.   Bundle bundle = new Bundle();
2.   bundle.putString("klíč", "hodnota");
3.   bundle.putInt("číslo", 1);
4.   bundle.putSerializable("cokoli serializovatelného", new Integer(3));
```

Pro klíč používat konstanty!

Získávání:

```
1.   String s = bundle.getString("klíč"); //nebo (String)bundle.get("klíč");
2.   int i = bundle.getInt("číslo");
3.   Integer integer = (Integer) bundle.get("cokoli serializovatelného");
```

Ukládání stavu activity

- Uložení proměnné `count`:

```
1.  @Override
2.  protected void onSaveInstanceState(Bundle outState) {
3.        super.onSaveInstanceState(outState);
4.        outState.putInt("count", count);
5.  }
```

- Obnovení:

```
1.  @Override
2.  protected void onCreate(Bundle savedInstanceState) {
3.    super.onCreate(savedInstanceState);
4.    if(savedInstanceState!=null){
5.          count = savedInstanceState.getInt("count");
6.    }
7.    ...
8.  }
```

Spouštění activit

- Activity se spouštějí pomocí `startActivity(intent);` nebo `startActivityForResult(intent, requestCode);`
- Historie activit se ukládá do zásobníku
- Pomocí Intentu se definuje, jaká activita se jak spustí

```
1. Context context = this;
2. Intent i = new Intent(context, NewActivity.class);
3. startActivity(i);
```

Context

- Abstraktní třída, od které dědí komponenty aplikace
- Poskytuje základní funkčnost společnou pro všechny komponenty
  - Přístup ke zdrojům
  - Přístup k třídám
  - Spouštění activit a služeb
  - Vysílání a odchytávání Intentů
  - …

Intent

- Prostředek pro komunikaci mezi komponentami
- Může obsahovat:
  - Jednoznačný identifikátor komponenty
  - Extras
  - Category
  - Data
  - Flags
- Více v 8. přednášce

Intent – jednoznačný identifikátor

- Slouží pro specifikování, jaká konkrétní třída se má po odeslání Intentu spustit

- Metoda má 2 parametry:
  - Context – odkaz na aktuální komponentu
  - Class – třída, která se má po odeslání intentu spustit

  - Např:

```
intent.setClass(MainActivity.this, NewActivity.class);
```

Intent – Extras

- Slouží pro předávání hodnot mezi komponentami

- Používá se stejné mapování jako u Bundle
  - Lze předat jakýkoli *Serializable* nebo *Parcelable* objekt
  - Mapuje se pomocí Stringů

```
intent.putExtra("count", count);
```

Intent – Flags

- Určuje, jakým způsobem se spustí nová activita (příp. jak bude umístěna na zásobník)

- `Intent.FLAG_ACTIVITY_NO_ANIMATION;`
  - Vypne animace při spouštění dané activity

- `Intent.FLAG_ACTIVITY_NO_HISTORY;`
  - Spouštěná activita se neuloží do zásobníku
  - Jakmile z ní uživatel odejde, activita se ukončí

Intent – Flags

- `Intent.FLAG_ACTIVITY_CLEAR_TOP;`
  - Pokud již je spouštěná activita v zásobníku, obnoví se a všechny activity nad ní budou ukončeny
- `Intent.FLAG_ACTIVITY_REORDER_TO_FRONT;`
  - Pokud již je spouštěná activita v zásobníku, obnoví se a přesune se navrch zásobníku
- `Intent.FLAG_ACTIVITY_SINGLE_TOP;`
  - Pokud je spouštěná activita na vrcholu zásobníku, obnoví se a nespustí se nová

Skok na domovskou activitu

- Určení domovské activity a vyčištění zásobníku

```
1. Intent intent = new Intent(this,NewActivity.class);
2. Intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
3. startActivity(intent);
4. finish();
```

- Navigace v rámci aplikace prošla výraznou změnou od verze 3.0
  - Některé informace mohou být deprecated (více. v 11. přednášce)

Launchmode

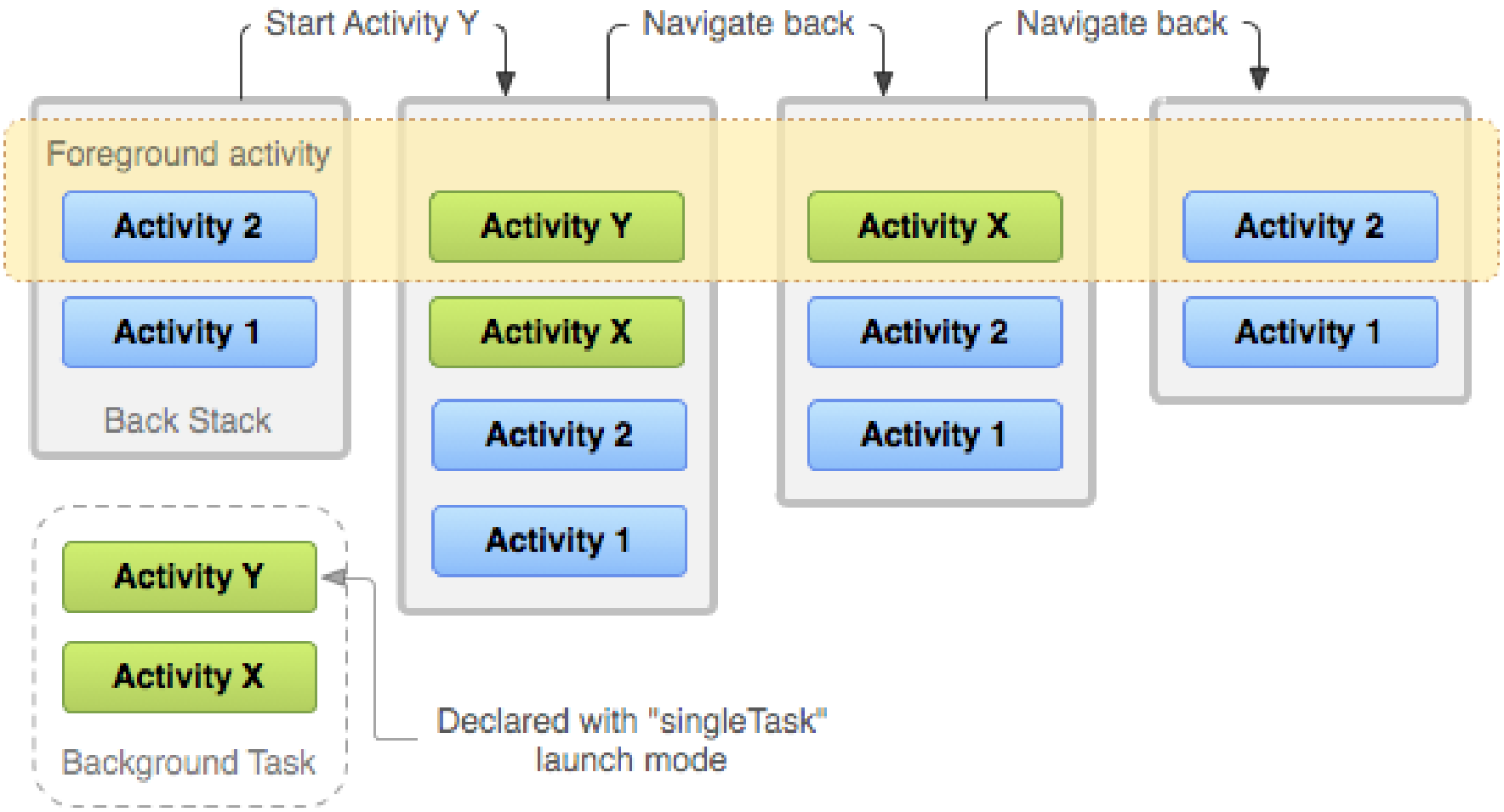
- Jak se spustí daná activita se také dá určit v *AndroidManifest.xml* atributem `android:launchMode`
- Učitá analogie ke konstantám *`FLAG_ACTIVITY_*`* v Intentu
- Může nabývat těchto hodnot: `standard`, `singleTop`, `singleTask` a `singleInstance`

Launchmode

Back Stack vs. Task

- `standard` – vždy vytvořena nová instance
- `singleTop` – vždy vytvořena nová instance, pokud není již instance activity na vrchu zásobníku
- `singleTask` – vytvořen nový task, pokud instance activity již existuje, dojde k přesměrování na ní (např. webový prohlížeč)
- `singleInstance` – stejné jako `singleTask`, ale v rámci tasku je vždy jenom jedna activita

Vždy funguje Back button nehledě na lauchmode


Start Activity Y
Navigate back
Navigate back
Foreground activity
Activity 2
Activity 1
Back Stack
Activity Y
Activity X
Activity 2
Activity 1
Activity X
Activity 2
Activity 1
Activity 2
Activity 1
Activity Y
Activity X
Background Task
Declared with "singleTask" launch mode

Activita očekávající výsledek

- Pomocí `startActivityForResult(intent, requestCode)` spustíme activitu, od které očekáváme, že nám po ukončení vrátí výsledek

- Nastavením `requestCode` umožníme vyfiltrování odpovědí při zpracování výsledků

```
1. Context context = this;
2. Intent i = new Intent(context, NewActivity.class);
3. i.setFlags(Intent.FLAG_ACTIVITY_NO_ANIMATION |
       Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
4. i.putExtra("cislo", 10);
5. startActivityForResult(i, 1);
```

Activita poskytující výsledek

- Pomocí `getIntent()` získáme Intent, který spustil activitu
- Pomocí `setResult(resultCode, data)` nastavíme výsledek, který se vrátí volající activitě
- Příkazem `finish()` ukončíme aktuální activitu

```
1.  Intent i = getIntent();
2.  int cislo = i.getIntExtra("cislo", 0);
3.
4.  Intent result = new Intent();
5.  result.putExtra("vysledek", cislo+10);
6.  setResult(RESULT_OK, result);
7.  finish();
```

Activita očekávající výsledek

- Metoda `onActivityResult` se zavolá když se ukončí activita spuštěná pomocí `startActivityForResult`
- `requestCode` je stejný jako při spouštění activity
- `resultCode` je číslo nastavené pomocí `setResult` v již ukončené aktivitě
- `data` – Intent s daty vrácenými activitou

```
1.   @Override
2.   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
3.      if(requestCode == 1 && resultCode==RESULT_OK){
4.            int vysledek=data.getIntExtra("vysledek", -1);
5.            ...
6.      }
7.      super.onActivityResult(requestCode, resultCode, data);
8.   }
```

Toast – zpráva pro uživatele

- Zobrazí se nad activitou
- Automaticky se zobrazuje a mizí
  - Nelze ji programově schovat ani nijak ovlivňovat při zobrazování
- Zobrazovat pouze dodatečné (krátké) informace
- Uživatel si nemusí Toastu všimnout
  - Bude tímto ovlivněna práce s aplikací?

Toast – zpráva pro uživatele

- Trvání zobrazení lze nastavit pouze na hodnoty
  - `Toast.LENGTH_SHORT`
  - `Toast.LENGTH_LONG`

- Pozicovatelnost – nastavuje se pomocí metody `setGravity(Gravity)`

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Další zdroje

- http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html
- http://developer.android.com/guide/practices/ui_guidelines/activity_task_design.html (depracated)

- Guidelines pro vývoj aplikací
- http://developer.android.com/guide/practices/design/performance.html