LUKÁŠ NOVÁK

KMI / TMA

Tvorba mobilních aplikací

9. seminář | 22.11.2016

ZS 2016/2017 | Středa 13:15-15:45

OBSAH SEMINÁŘE

Fragmenty a to,
co se nevlezlo jinam

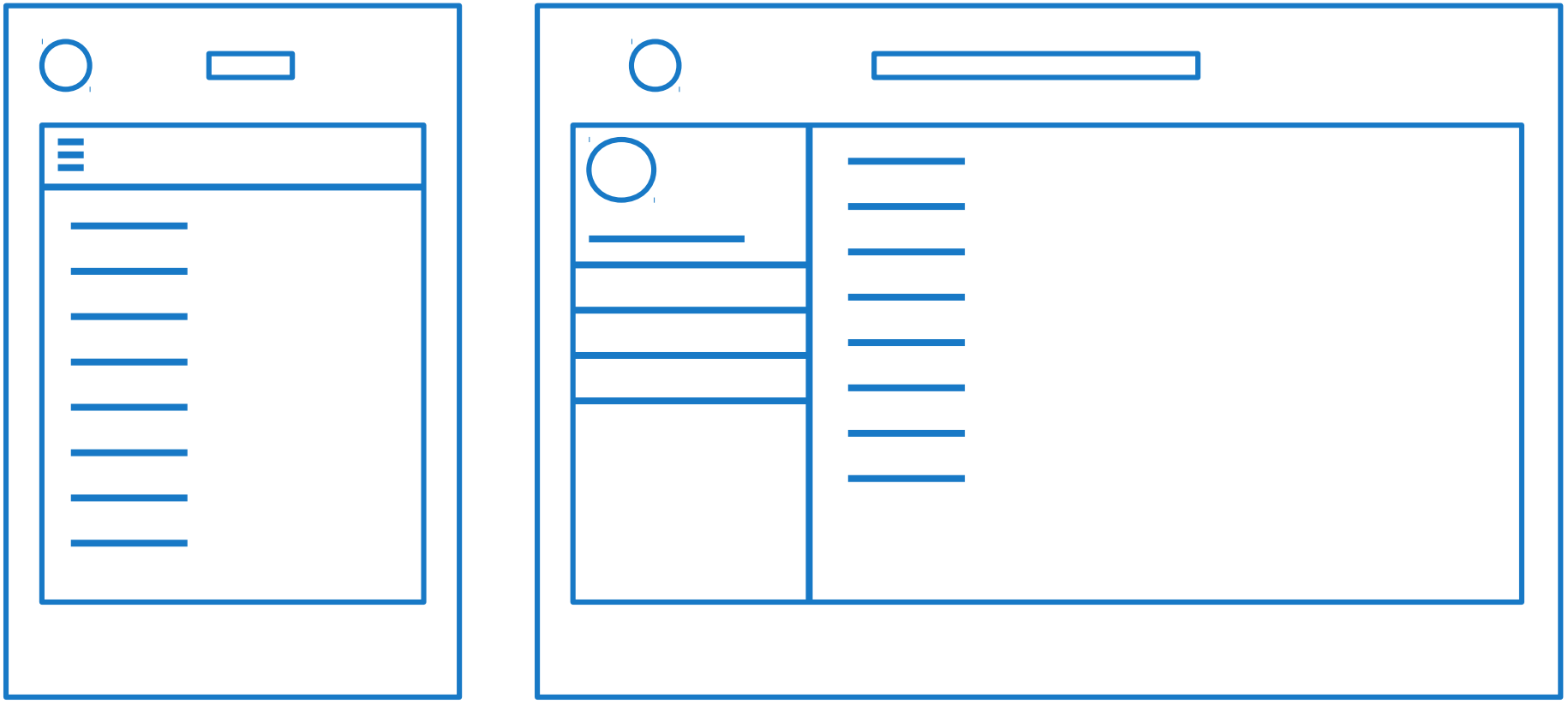
VĚTŠÍ DISPLEJE

JE TŘEBA NĚCO DĚLAT?

- většina aplikací, které nejsou uzpůsobeny pro různé velikosti displejů, nevypadá na velkých displejích dobře
- UI prvky se roztáhnou a UI vypadá najednou prázdně

VĚTŠÍ DISPLEJE

JE TŘEBA NĚCO DĚLAT?



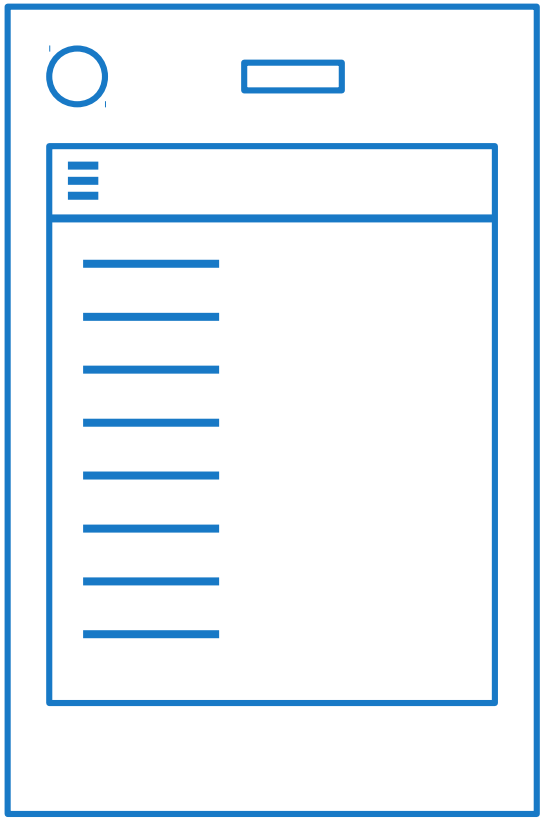
VĚTŠÍ DISPLEJE

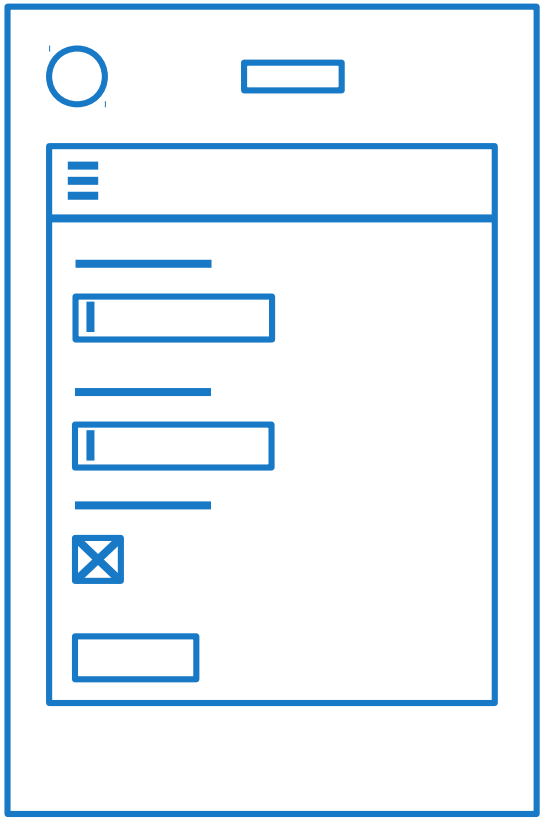
JE TŘEBA NĚCO DĚLAT?

- často je i lepší, pokud se aplikace chová na tabletu jinak než na telefonu
- jeden z přístupů je ten, že budeme brát displej tabletu jako více menších obrazovek
- spousta aplikací implementuje tzv. master-detail pattern

VĚTŠÍ DISPLEJE

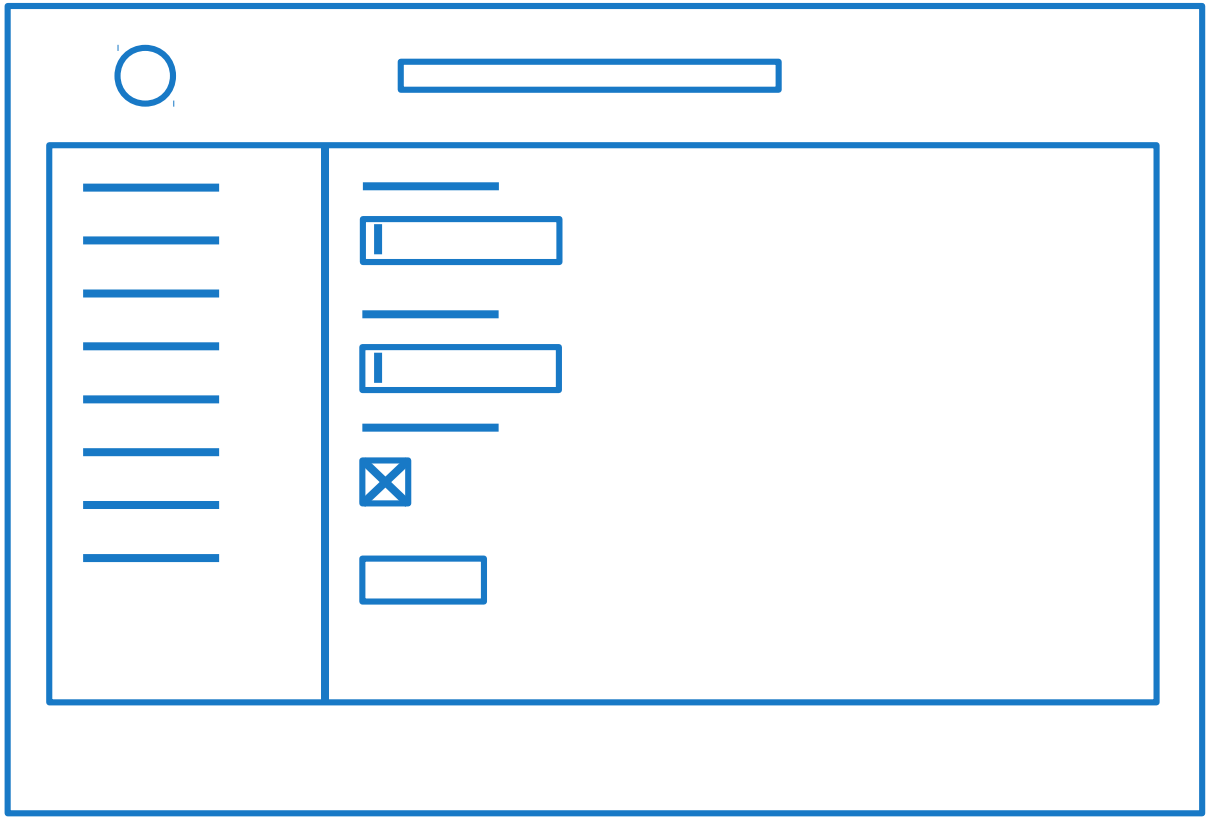
JE TŘEBA NĚCO DĚLAT?





VĚTŠÍ DISPLEJE

JE TŘEBA NĚCO DĚLAT?



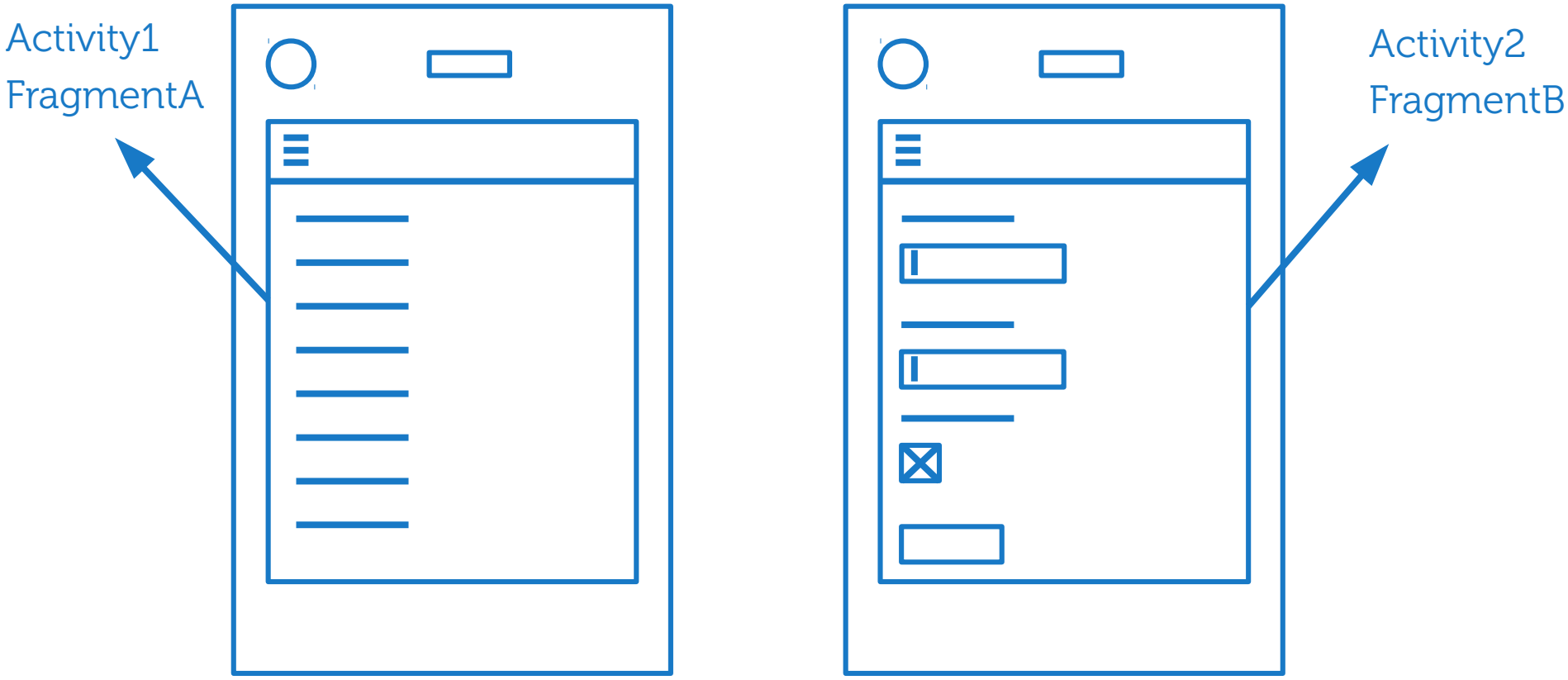
FRAGMENTY

ŘEŠENÍ?

- od Android 3.0 je k dispozici Fragment API, pomocí kterého je možné implementovat takové chování
- fragmenty jsou znovupoužitelné části UI, které je možné vkládat do aktivit
- najednou neplatí 1 aktivita = 1 obrazovka
- v extrémním případě může aplikace obsahovat pouze 1 aktivitu + X fragmentů

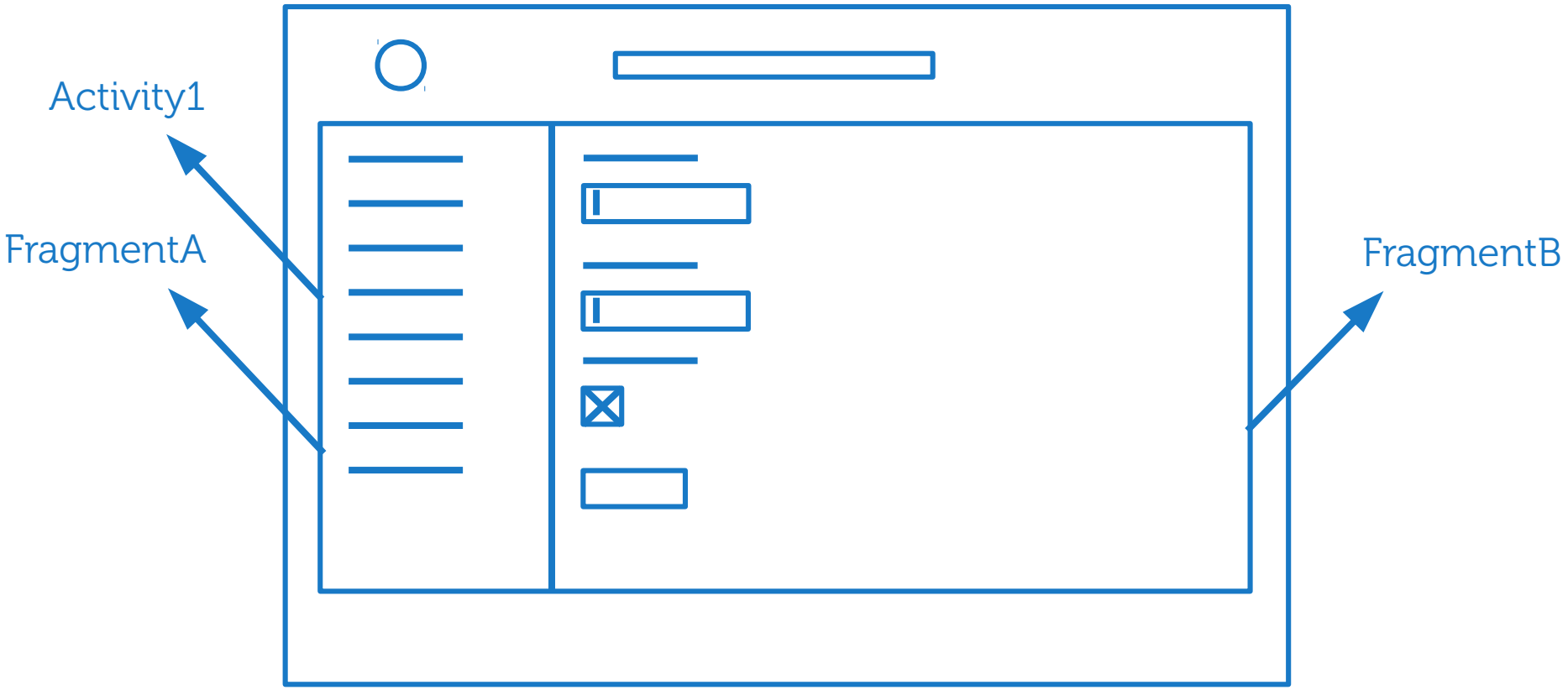
FRAGMENTY

ŘEŠENÍ?



FRAGMENTY

ŘEŠENÍ?



FRAGMENTY

LOW-LEVEL

- fragmenty jsou velmi flexibilní, master-detail pattern je jen jeden z mnoha použití fragmentů
- s flexibilitou fragmentů však roste náročnost na implementaci funkčního UI
- implementace často závisí na správné architektuře a oddělení UI od logiky

FRAGMENTY

DEFINICE FRAGMENTŮ

- fragmenty je možné definovat přímo v layoutu nebo je z aktivity načítat dynamicky podle situace
- dynamické UI s fragmenty spočívá ve vytvoření layoutu, který má **FrameLayout** na místech, kam přijde fragment

FRAGMENTY

DYNAMICKÉ PŘIDÁVÁNÍ FRAGMENTŮ

- dynamické přidávání zajišťuje **FragmentManager**
- změny v UI s fragmenty probíhají
- v transakcích, lze provést více změn v jeden okamžik

```
getSupportFragmentManager().beginTransaction()
    .add(R.id.master, fragment)
    .commit();
```

FRAGMENTY

Stručný popis master-detail

- vytvoříme dva layouty pro hlavní aktivitu:
  - na výšku 1x **FrameLayout**
  - na šířku 2x **FrameLayout** vedle sebe
- v aktivitě zjistíme, kolik máme k dispozici fragmentů a podle toho načteme buď pouze **master fragment** nebo i **detail fragment**
- logiku pro zobrazení master/detail zajišťují fragmenty

FRAGMENTY

Stručný popis master-detail

- pro zobrazení jiného **detail fragment** z **master fragmentu** potřebujeme mít přístup z **master fragmentu** do aktivity
  - kontrakt ve formě rozhraní, které zajišťuje provedení akce na základě layoutu, tj. zobrazení detailu v **detail fragmentu** nebo spuštění aktivity s **detail fragmentem**

VLASTNÍ ADAPTÉR

KDYŽ NÁM TOSTRING() NESTAČÍ

VLASTNÍ ADAPTÉR

KDYŽ NÁM TOSTRING() NESTAČÍ

- prostředníka mezi daty a seznamem zajišťuje adaptér, v našem případě to byl jednoduchý **ArrayAdapter**, který na místě v jednotlivých řádcích seznamu zobrazoval výstup metody **toString()**
- to však často nestačí, např. když chceme mít v řádku složitější layout

VLASTNÍ ADAPTÉR

KDYŽ NÁM TOSTRING() NESTAČÍ

- řešení je vytvořit potomka **ArrayAdapter**, který bude mít kolekci v členské proměnné
- pro základní použití přepíšeme metody:
  - **getCount**: vrací počet položek v kolekci
  - **getItem**: vrátí položku na předané pozici
  - **getView**: vrátí View pro danou pozici
- konstruktoru na místě layoutu předáme 0, protože vytvoření layout si řešíme sami v **getView**

VLASTNÍ ADAPTÉR

KDYŽ NÁM TOSTRING() NESTAČÍ

- implementace **getView**:
  - získáme si **LayoutInflater** a načteme **View** z layoutu

```
View view = LayoutInflater.from(getContext())
    .inflate(R.layout.layout_item, parent, false);
```

  - získáme UI prvky z layout (**findViewById**) a upravíme je dle položky, kterou získáme pomocí přepsané **getItem**

VLASTNÍ ADAPTÉR

KDYŽ NÁM TOSTRING() NESTAČÍ

- implementace **getView**:
  - v produkčních aplikacích je potřeba použít **ViewHolder** pattern, který zajistí plynulé scrollování seznamu při složitých layoutech a mnoha položkách
  - v současné době se doporučuje používat **RecyclerView**:
    - **ListView** + **Adapter** + něco navíc

KNIHOVNY

JAK NEVYNALÉZAT KOLO

KNIHOVNY

JAK NEVYNALÉZAT KOLO

- nástroj **Gradle** pro sestavení aplikace dokáže získat knihovny třetích stran z centrálního repozitáře **jCenter**
- v **build.gradle** modulu stačí do sekce **dependencies** přidat položku **compile**, např. pro **GSON**:

```
compile 'com.google.code.gson:gson:2.8.0'
```

- a následně v toolbaru „**Sync with Gradle**“

ÚKOL 9. SEMINÁŘE

1) přidat do seznamu ikonku znázorňující provedení úkolu
2) přidat GSON knihovnu pro práci s JSON a provést pomocí ní deserializaci TODO položek z URL namísto ručního získání vlastností

Tipy pro řešení:

1) objekt todo položky musí mít stejné jména členských proměnných, v opačném případě je nutné k proměnným přidat anotaci @SerializedName("?"}, kde ? je jméno vlastnosti v JSON
2) dokumentace k použití GSON je na https://sites.google.com/site/gson/gson-user-guide (hledejte **Object Examples – deserialization** nebo ještě lépe přímo **Array Examples – deserialization**)

OTÁZKY

Ptejte se!