

# POEC Android



Pierre Duchemin ( [@pierreduchemin](#) )

## Gagner du temps (2)

- Analyser un apk : Build -> Analyse APK...
  - Alternative : JADX
- Annotations : @Nullable, @NonNull, @ColorRes, @StringRes...
- Tools: pour la preview

```
android:text="Hello World!"  
tools:text="Hello World!"
```

# Les fuites de mémoire

C'est un bug qui fait qu'un processus ne libère pas de la mémoire qui lui a été allouée et dont il n'a plus besoin.

S'il ne reste plus de mémoire utilisable : `OutOfMemoryError`

## Causes

- Quand on place en static des Views

```
private static TextView textView;
```

- ... ou des Contexts

```
private static Context context;
```

<https://riggaroo.co.za/fixing-memory-leaks-in-android-outofmemoryerror/>

# Thread

```
final Handler handler = new Handler();
handler.post(new Runnable() {
    @Override
    public void run() {
        Toast.makeText(MainActivity.this,
            "Je suis dans un thread",
            Toast.LENGTH_LONG).show();
        handler.postDelayed(this, 10000);
    }
});
```

# Images et résolutions

- Le 9-patch est un procédé permettant d'étirer les images
  - Pas de problème de résolution
  - Gain de place
  - Ne s'applique qu'à peu de cas

What you want

Asset you'll need to provide

**Yes, I'm in**



## Exercice 5

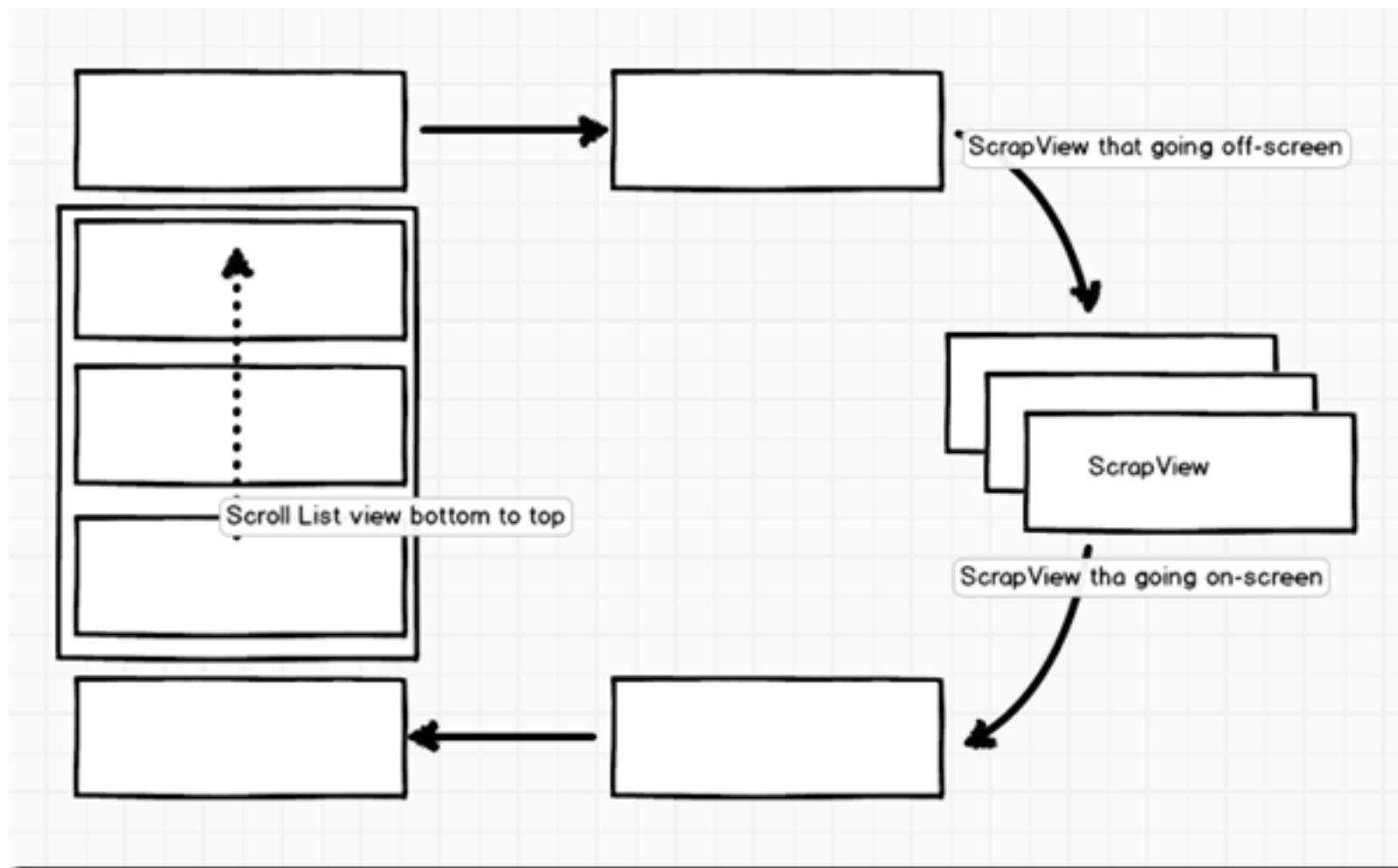
- Faire un chronomètre (précision : secondes)
- Utiliser des 9-patches pour faire les boutons

# Gérer des listes

- **Adapter : groupe d'éléments.** Permet de faire des listes, des grilles... Un adapter doit **renvoyer l'IHM d'un élément pour un indice donné**
- **ViewHolder :** stocke les références aux vues pour simplifier le code des adapters

# Gérer des listes

- L'inflating (l'analyse du xml pour le rendre utilisable en java) est long
- Recycling :





# Gérer des listes

Recycling géré manuellement :

```
if (convertView == null) {  
    holder = new ViewHolder();  
    LayoutInflater inflater = (LayoutInflater)context.get  
    convertView = inflater.inflate(R.layout.beer_list_item,  
    holder.tvText = (TextView) convertView.findViewById(R  
        convertView.setTag(holder);  
} else {  
    holder = (ViewHolder) convertView.getTag();  
}
```

# Gérer des listes

- **RecyclerView** : abstraction d'une liste
- Requiert un **LayoutManager** qui détermine la mise en page (GridLayoutManager, LinearLayoutManager...)
- Requiert l'ajout de :

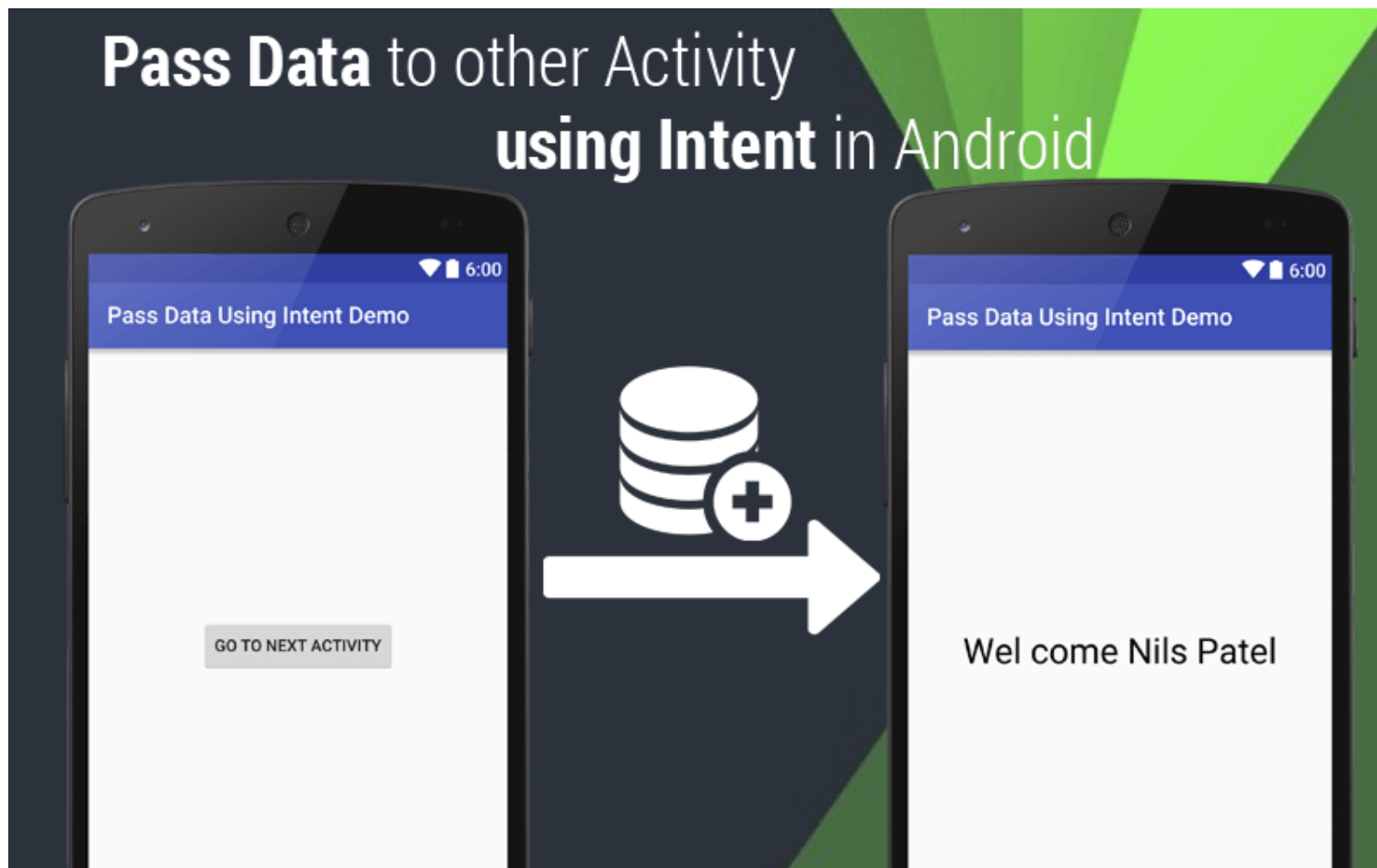
```
compile 'com.android.support:appcompat-v7:25.2.0'
```

## Exercice 6

- Créer une liste
- Reprendre le projet et gérer une grille

# Rappel : les Intents

Les **Intent** des relais entre des **activity** ou des **applications**. Ils permettent de passer des données d'une entité à l'autre. Ils peuvent également être porteurs de "flags", des marqueurs indiquant une action particulière à réaliser.



# Rappel : les Intents

- Envoi :

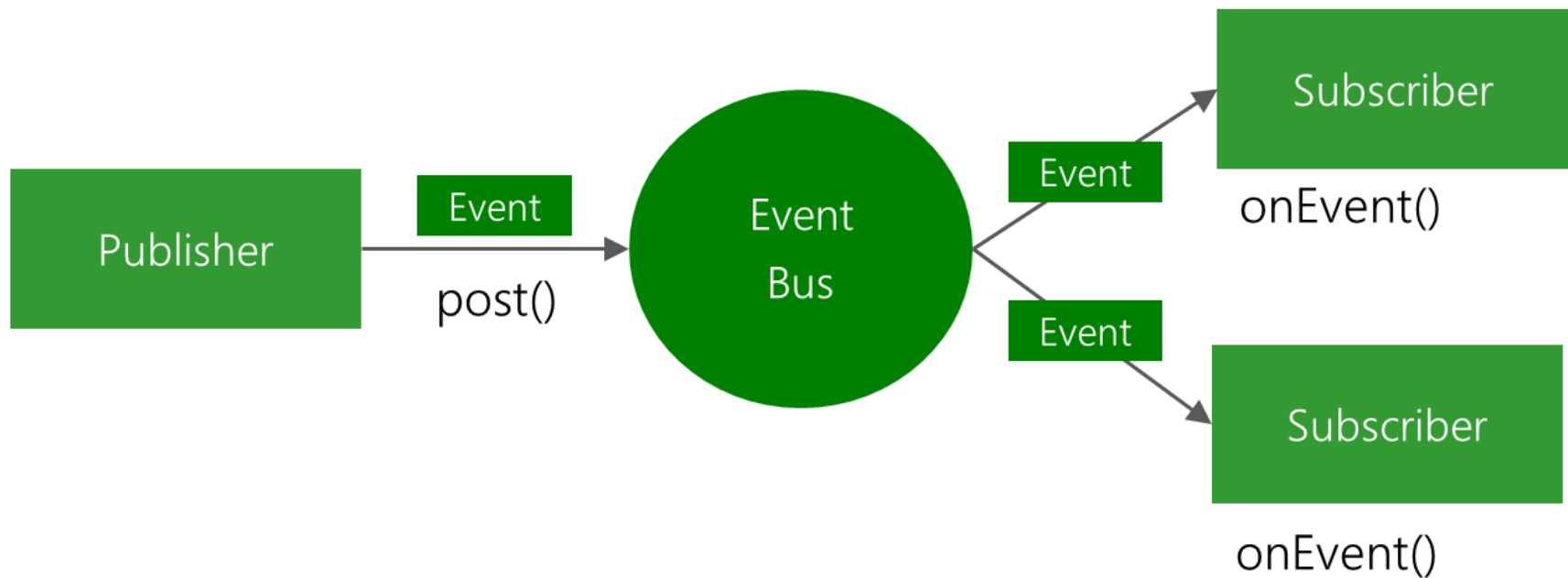
```
Intent intent = new Intent(AccountManagerActivity.this,  
    LoginActivity.class);  
intent.putExtra("account:email", item.emailAccount);  
startActivity(intent);
```

- Récupération :

```
extraEmail = getIntent().getExtras()  
    .getString("account:email");
```

# EventBus

- Permet de simplifier les IPC



```
compile 'org.greenrobot:eventbus:3.0.0'
```

# EventBus

- Définir un événement :

```
public class MessageEvent { /* Additional fields if needed
```

- Enregistrer les subscribers (onCreate ou constructeur) :

```
EventBus.getDefault().register(this);
```

# EventBus

- Désinscrire les subscribers avec :

```
EventBus.getDefault().unregister(this);
```

- Réagir aux événements :

```
@Subscribe  
public void onEvent(AnyEventType event) {/* Do something
```

- Envoyer un événement :

```
EventBus.getDefault().post(event);
```



# Exercice 7

- Utilisation d'un eventbus
  - 1 - Faire une IHM avec 1 Button
  - 2 - Quand l'application s'ouvre, un événement est posté sur l'eventbus par défaut
  - 3 - L'événement affiche un message : "Message déclenché depuis l'EventBus !"
  - 4 - Quand on quitte l'application, la mémoire est correctement vidée