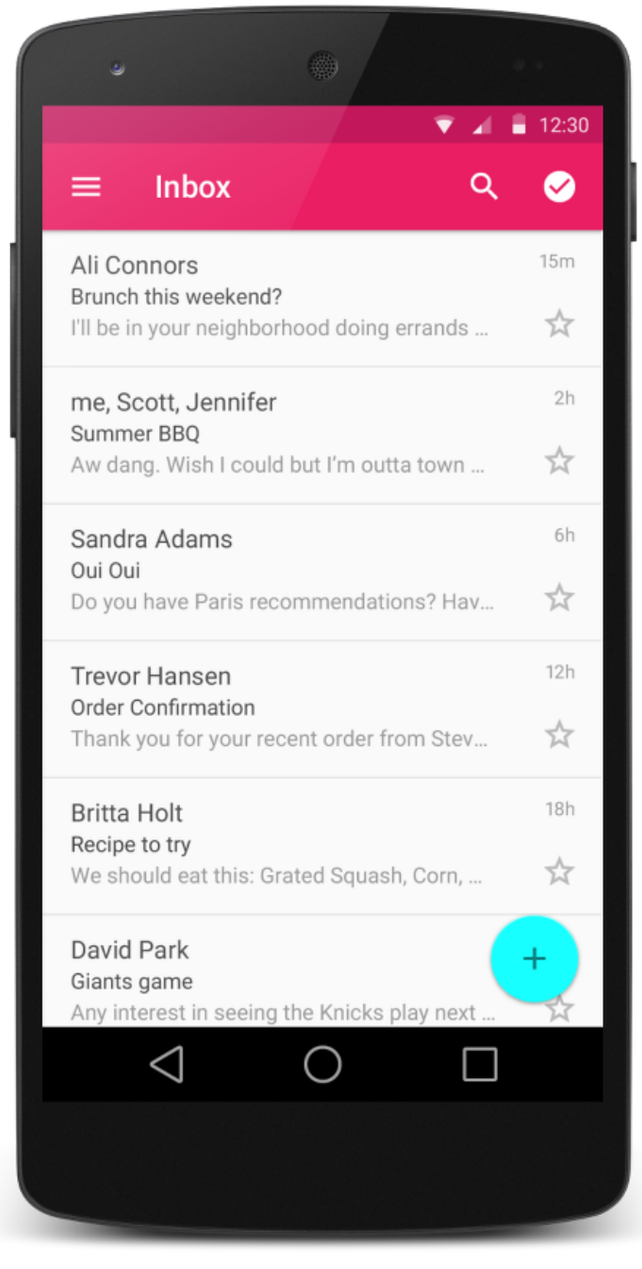
# MOBG56 - Développement mobile

## RecyclerView

# RecyclerView

- RecyclerView : liste scrollable de Views
- Très répandus dans les applications mobiles
- Widget le plus personnalisable des Widget list
- Le plus complexe aussi
- Classe dérivée de ViewGroup : contient plusieurs Views enfant
- Les Views item basé sur un layout xml

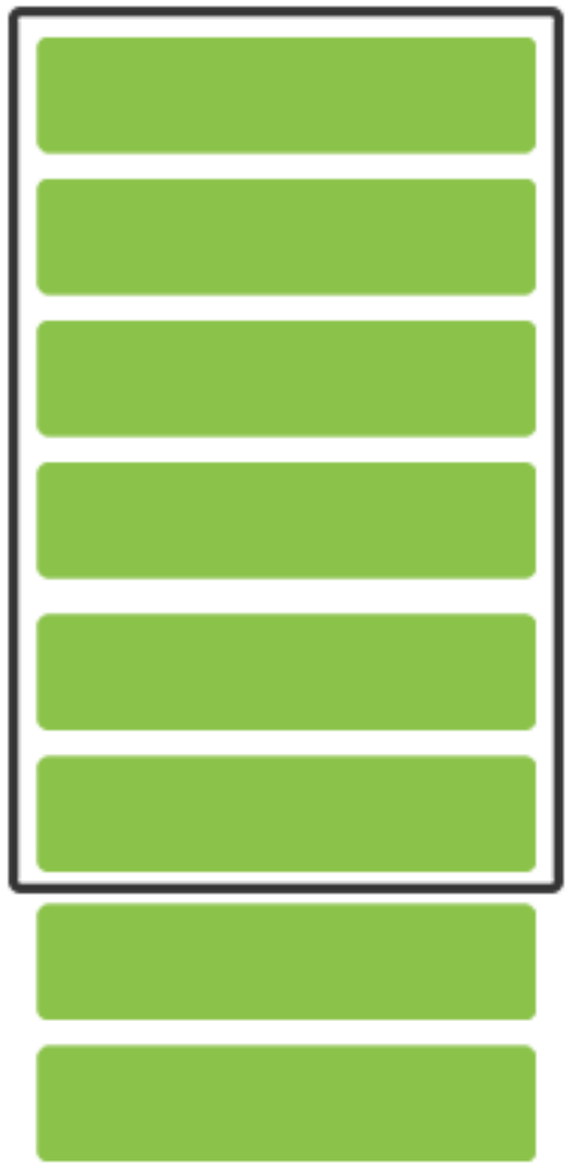


# RecyclerView

- Comment faire pour une liste de 10.000 éléments ? Impossible à garder en mémoire

- Un item n’a besoin d’une View que lorsqu’il est affiché à l’écran

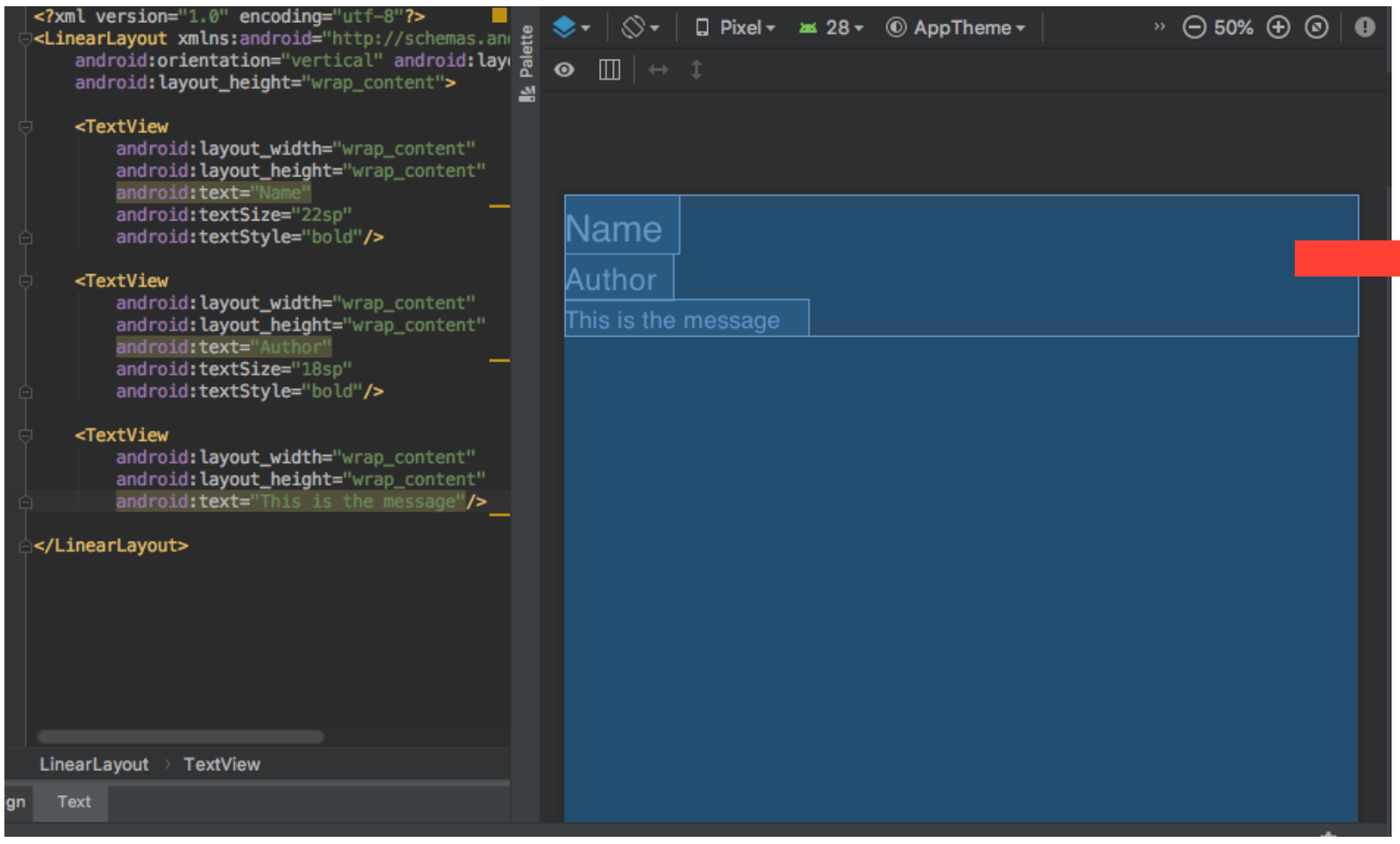
- Solution : RecyclerView va créer un nombre minimum de Views pour remplir l’écran et les réutiliser (recycler) petit à petit quand l’utilisateur scrolle

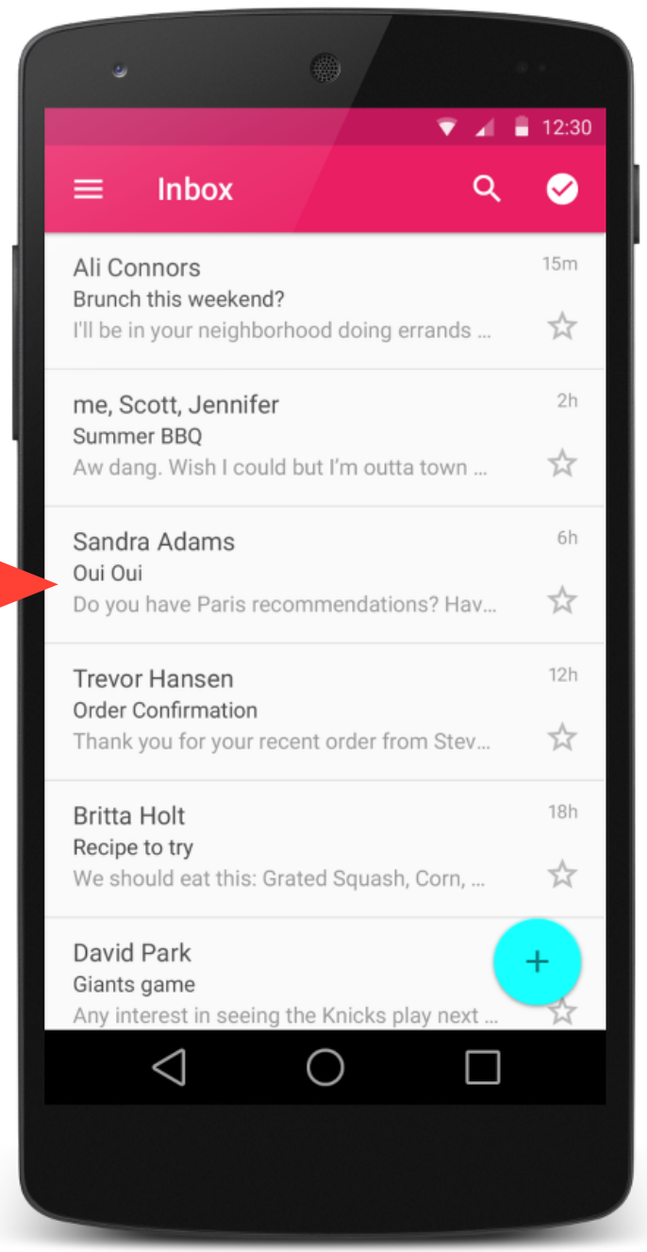


# Layout des items

- Les items du RecyclerView sont basés sur un layout xml similaire aux layouts des activités.
- Un seul layout servira de modèle pour tous les items.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.an
    android:orientation="vertical" android:lay
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Name"
        android:textSize="22sp"
        android:textStyle="bold"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Author"
        android:textSize="18sp"
        android:textStyle="bold"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is the message"/>

</LinearLayout>
```



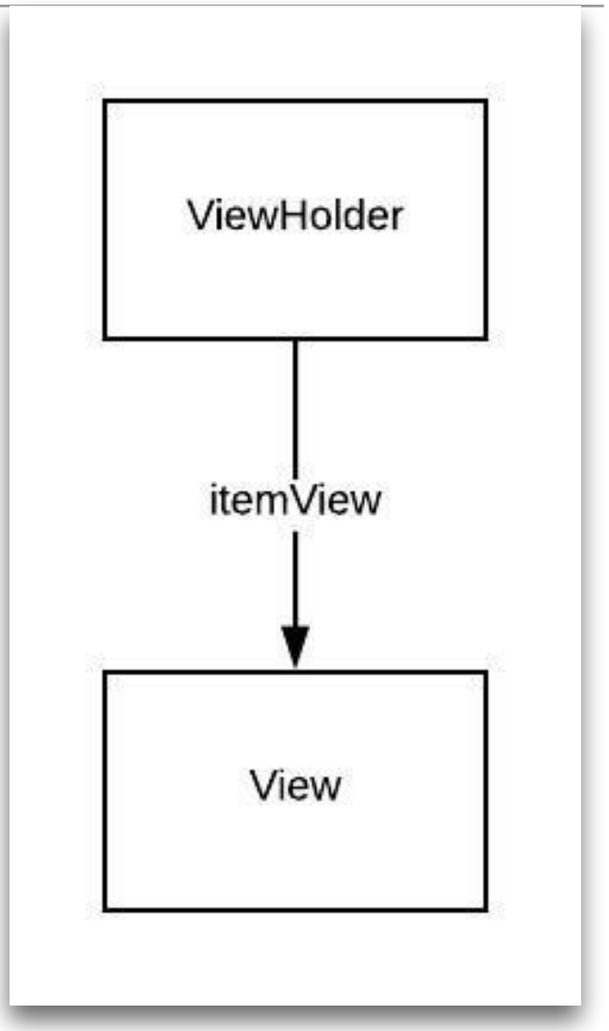



# ViewHolder

- **ViewHolder** est le composant qui va contenir la View de l'Item

- Il contient également sa position dans la liste

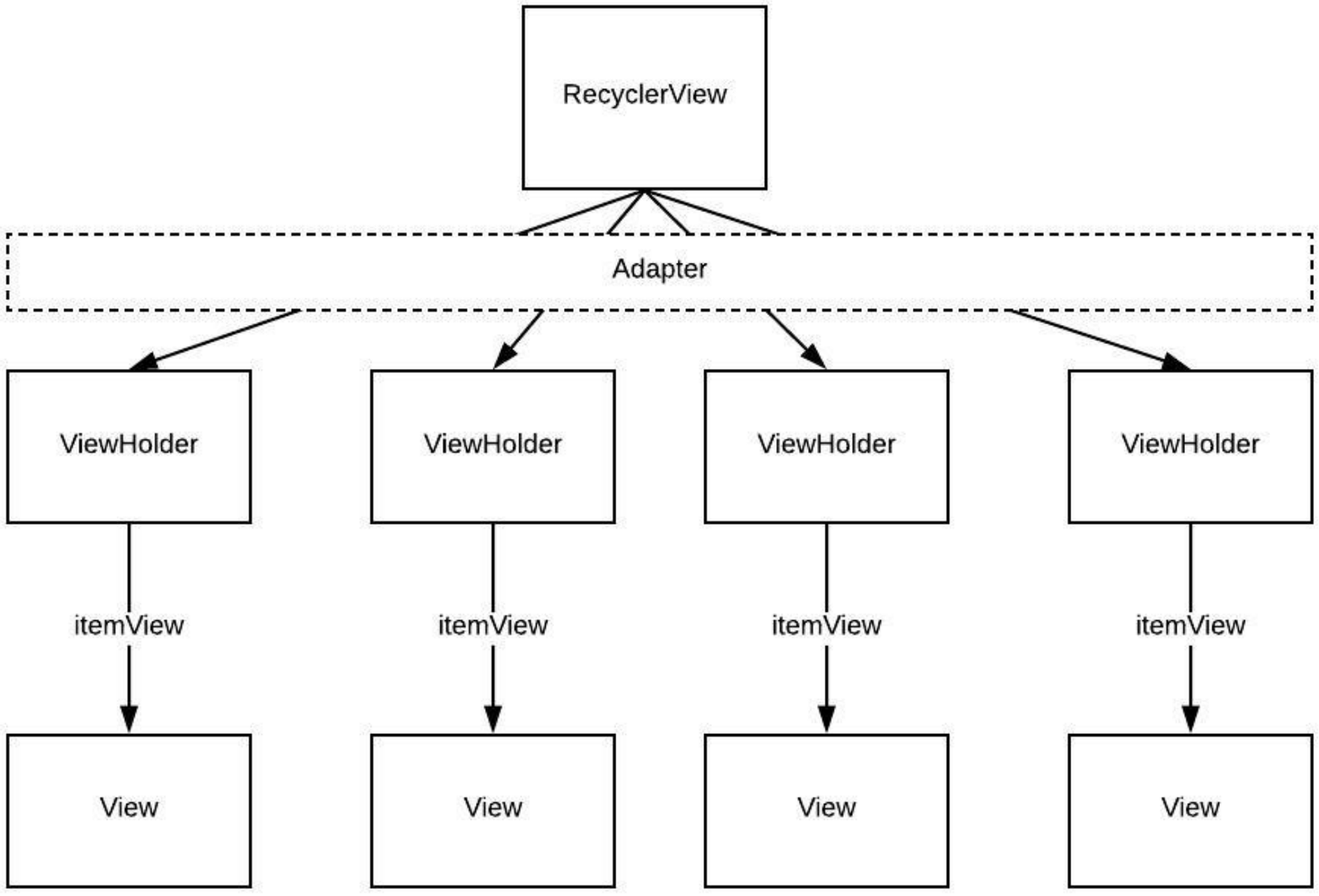
- Déclaration d'un ViewHolder :




```
class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)
{

}
```

# ViewHolder

- intermédiaire du RecyclerView : **Adapter**



# Adapter

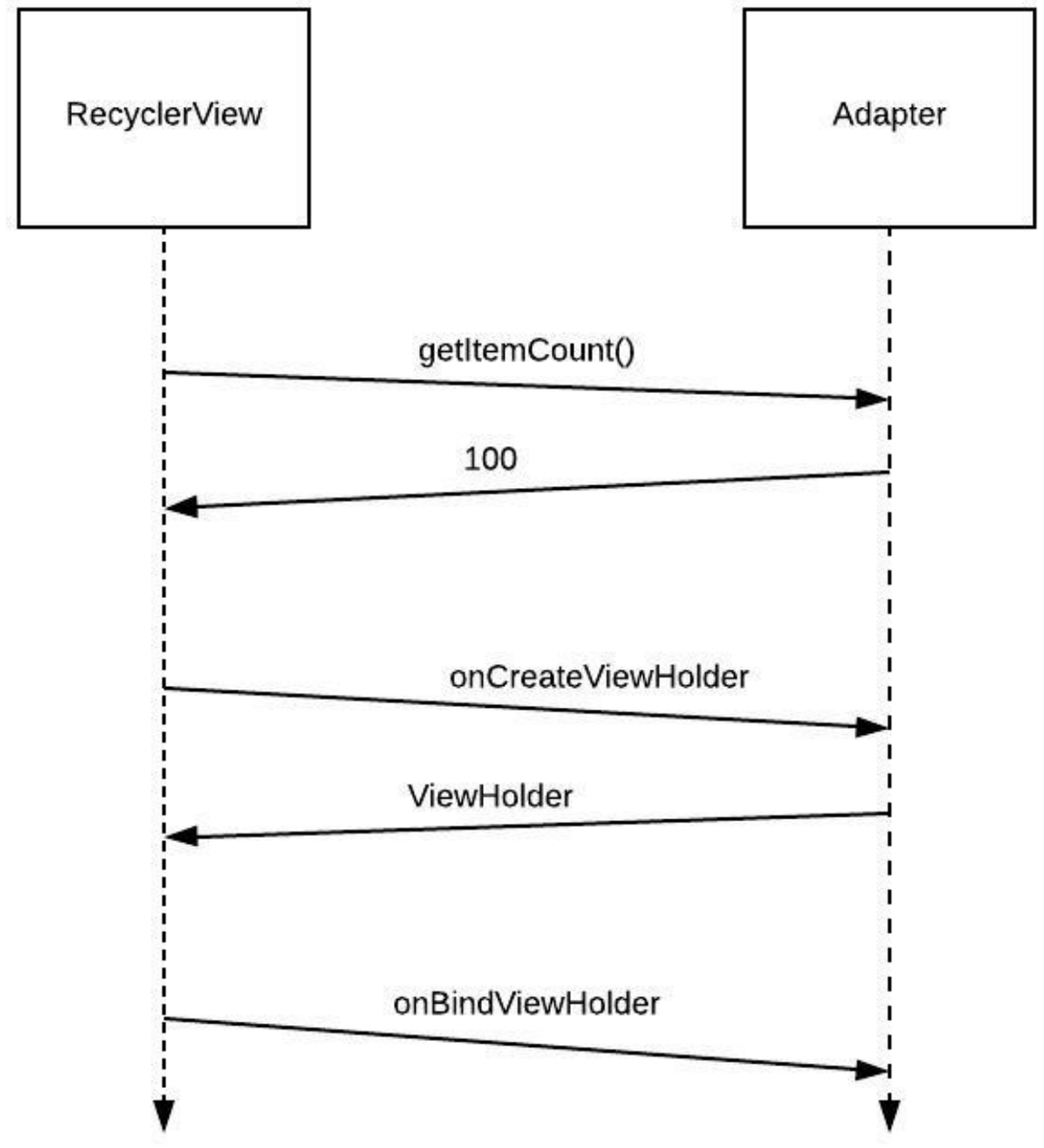
- **Adapter** : intermédiaire entre le Recycler View et les données/viewHolder
  - Crée les ViewHolders
  - Responsable de fournir les données au ViewHolders
- RecyclerView a besoin d'afficher un item ? -> démarre une conversation avec son Adapter :
  - Combien d'items la liste contient-elle ?
  - Je dois afficher 8 items, créée-moi 12 ViewHolders
  - Rempli-les avec les données du modèle
- Le développeur répond a ces questions dans des callbacks de l'Adapter appelées par le RecyclerView

# Adapter

- **getItemCount** :
  combien d'items dans la liste ?

- **onCreateViewHolder** :
  crée un ViewHolder

- **onBindViewHolder** :
  rempli le ViewHolder avec des données



# Adapter

- Déclarer un Adapter :

```
class MyAdapter : RecyclerView.Adapter<MyAdapter.MyViewHolder>(){

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
                                                                MyViewHolder {
       ...
    }

    override fun getItemCount(): Int {
       ...
    }

    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
       ...
    }

    inner class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){

    }
}
```

# Adapter

1. - **onCreateViewHolder** : appelé par le RecyclerView quand il a besoin d'une nouvelle View

   - Crée un nouveau ViewHolder à partir d'une view et d'un layout

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
    // get a layoutInflater from the context attached to the parent view
    val layoutInflater = LayoutInflater.from(parent.context)

    // inflate the layout "item" in a view
    val myView = layoutInflater.inflate(R.layout.item, parent, false)

    // create a new ViewHolder with the view planetView
    return MyViewHolder(myView)
}
```

- Cette méthode est normalement appelée lorsque le RecyclerView est initialisé.

# Adapter

2. - **OnBindViewHolder** : appelé par le RecyclerView quand il doit charger des données dans un item.

   - Fait le lien entre les données (modèle) et le ViewHolder.  L'argument **position** représente la position de l'item dans la liste.

```
// some data coming from Model ...
var modelData : List<String> = ...

override fun onBindViewHolder(holder: MyViewHolder, position: Int){
    // get the corresponding data in the model
    val dataToDisplay = modelData[position]

    // update UI of the item
    val itemViewGroup = holder.itemView as ViewGroup
    val textView : TextView = itemViewGroup.findViewById(R.id.text_view)
    textView.text = dataToDisplay
}
```

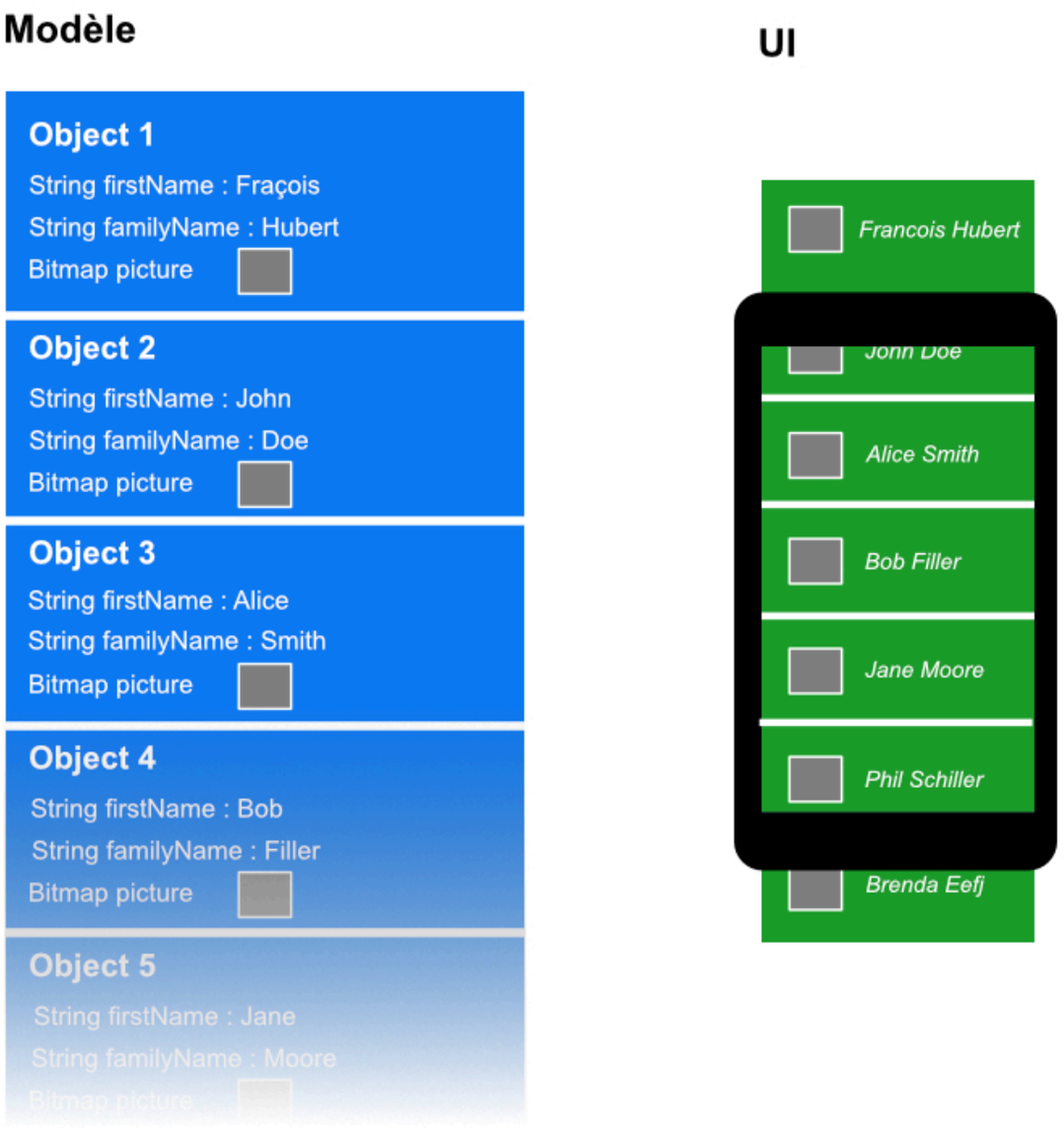
# Adapter

3. **getItemCount**, appelé par le RecyclerView pour connaître le nombre total d'items

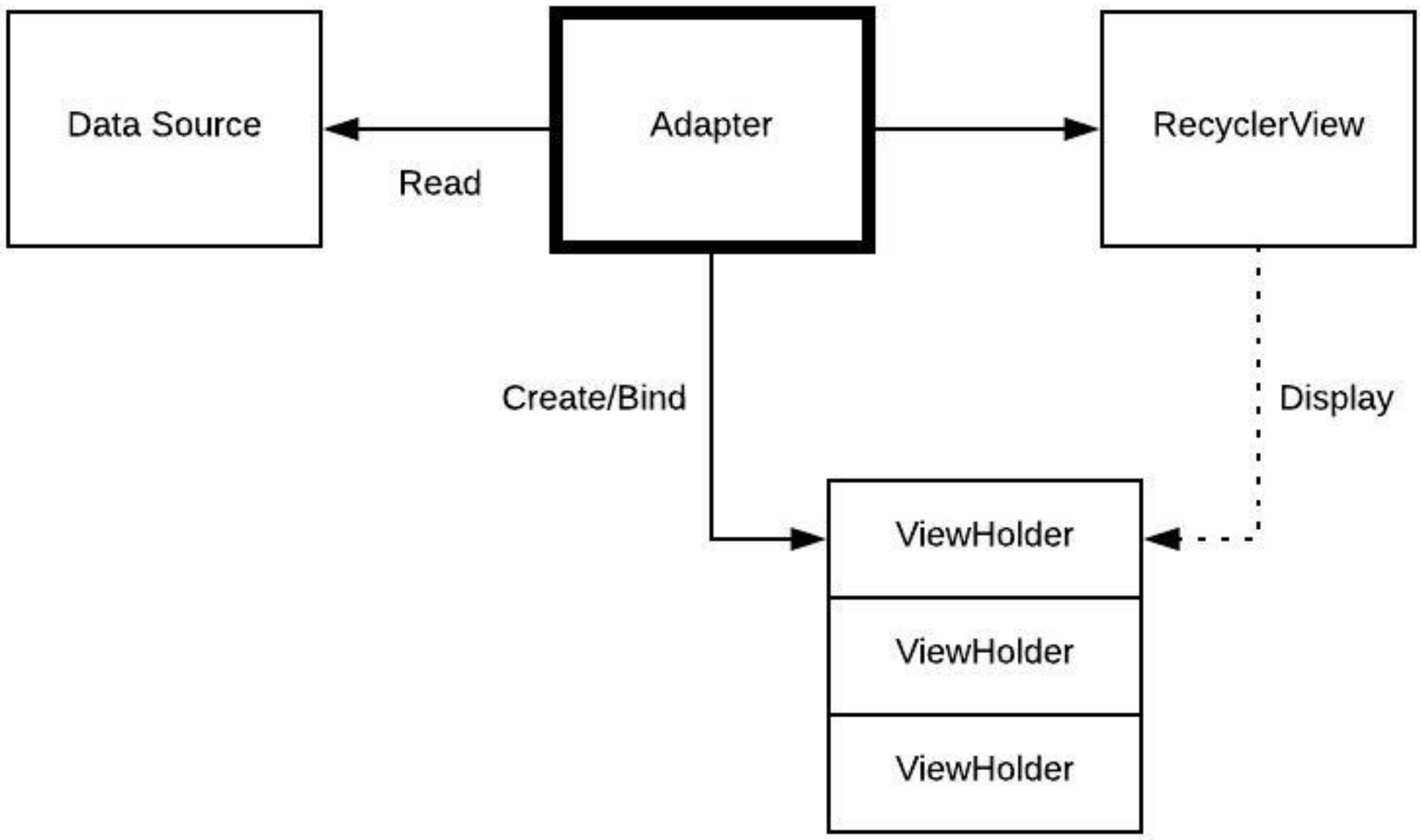
```
override fun getItemCount(): Int {
   return modelData.size
}
```

# Modèle

- Modèle (data-source) :
  - tableau en mémoire (List<>)
  - une base de donnée
  - contenu généré à la volée
  - dépend entièrement du dévelopeur.



# Récapitulatif



# Layout Manager

- Le RecyclerView peut être présenté selon plusieurs layouts :
  - liste linéaire verticale
  - liste linéaire horizontale
  - grille
  - forme personnalisée
- Ces formes sont définies par un objet **LayoutManager**. Pour afficher un Recycler View, il faut passer une instance LayoutManager au RecyclerView.
- Android fourni des LayoutManagers pour les layouts standards. En cas de besoin, un LayoutManager peut être créé sur mesure (c'est rarement le cas).

```
val layoutManager = LinearLayoutManager(context)
recyclerView.layoutManager = layoutManager
```

# Modification des items

- RecyclerView doit être notifié de tout changement dans les données à afficher. Ceci se fait grâce aux méthodes RecyclerView.notify…
- **NotifyDataSetChanged** recharge TOUT le RecyclerView.  A utiliser avec parcimonie
- **NotifyItemChanged** recharge un item précis

| | |
|---|---|
| `final void` | `notifyDataSetChanged()`<br>Notify any registered observers that the data set has changed. |
| `final void` | `notifyItemChanged(int position, Object payload)`<br>Notify any registered observers that the item at `position` has changed with an optional payload object. |
| `final void` | `notifyItemChanged(int position)`<br>Notify any registered observers that the item at `position` has changed. |
| `final void` | `notifyItemInserted(int position)`<br>Notify any registered observers that the item reflected at `position` has been newly inserted. |
| `final void` | `notifyItemMoved(int fromPosition, int toPosition)`<br>Notify any registered observers that the item reflected at `fromPosition` has been moved to `toPosition`. |
| `final` | `notifyItemRangeChanged(int positionStart, int itemCount, Object payload)` |

# Récapitulatif

1. Créer un layout de l'item

2. Implémenter une classe Adapter

3. Implémenter un ViewHolder (inner class de l'adapter)

4. Redéfinir les méthodes onCreateViewHolder, getItemCount, OnBindViewHolder

5. Créer un layoutManager et l'ajouter au RecyclerView