# Programmation et interfaces Android

Interfaces utilisateurs ( avancées )

# Sommaire

# Les Fragments ( Principes )

- Un fragment est une partie ( ou un ... fragment :p ), réutilisable de notre interface graphique.

# Les Fragments ( Principes )

- Un fragment est une partie ( ou un ... fragment :p ), réutilisable de notre interface graphique.
- Notre activité va donc être découpée en un ou plusieurs fragments.

# Les Fragments ( Principes )

- Un fragment est une partie ( ou un ... fragment :p ), réutilisable de notre interface graphique.
- Notre activité va donc être découpée en un ou plusieurs fragments.
- Cela rendra donc notre UI, modulable et flexible.

# Les Fragments ( Principes )

Quelques règles élémentaires :

- Les fragments ne sont pas des activités. Cependant ils ne sont pas indépendant ( ils ont besoin d'être placé dans une activity )

# Les Fragments ( Principes )

Quelques règles élémentaires :

- Les fragments ne sont pas des activités. Cependant ils ne sont pas indépendant ( ils ont besoin d'être placé dans une activity )
- Les fragments se gère comme les activity : ils possèdent un cycle de vie similaire et ont besoin d'un layout

# Les Fragments ( Principes )

Quelques règles élémentaires :

- Les fragments ne sont pas des activités. Cependant ils ne sont pas indépendant ( ils ont besoin d'être placé dans une activity )
- Les fragments se gère comme les activity : ils possèdent un cycle de vie similaire et ont besoin d'un layout
- Les fragments sont lié à l'acitivity, leurs cycle de vie aussi. Détruire une activity c'est aussi détruire les fragments

# Les Fragments ( Principes )

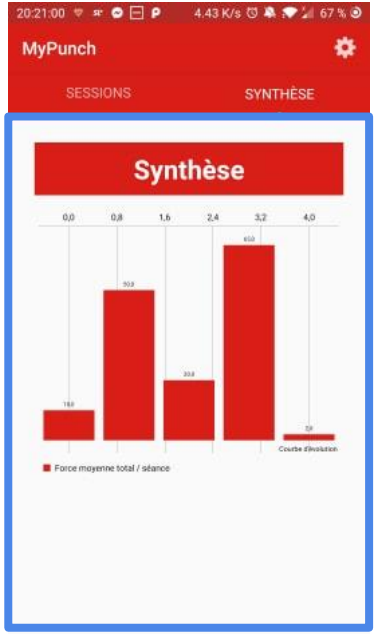
## Quelques règles élémentaires :

- Les fragments ne sont pas des activités. Cependant ils ne sont pas indépendant ( ils ont besoin d'être placé dans une activity )
- Les fragments se gère comme les activity : ils possèdent un cycle de vie similaire et ont besoin d'un layout
- Les fragments sont lié à l'acitivity, leurs cycle de vie aussi. Détruire une activity c'est aussi détruire les fragments
- Dans une architecture MVC, l'activity joue le rôle de contrôleur "global" ( getsion des communications et de l'affichage des fragments, de la navigation, etc ... ) tandis que les fragment joue le rôle de contrôleur "local" ( gestion des vues, des modèles de données, des appels réseaux, etc ... )

# Les Fragments ( Principes )

## Un exemple de fragment :

Ici le fragment synthèse est active mais l'activity possède aussi un fragment sessions

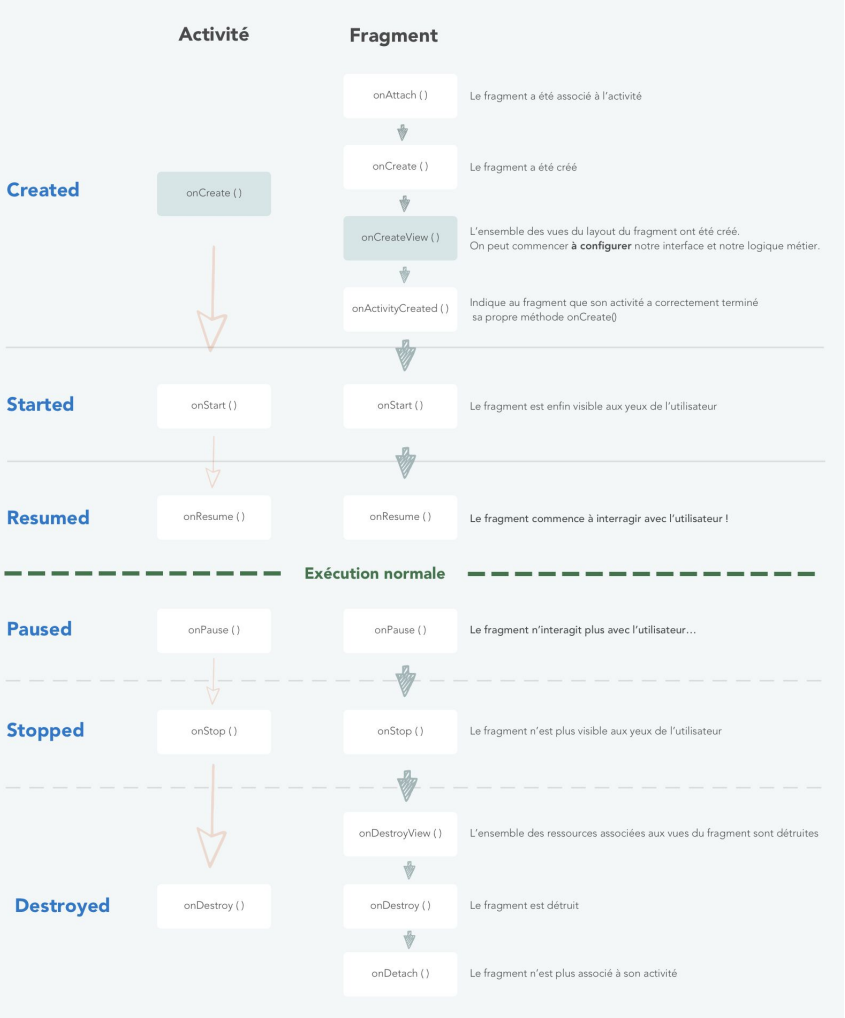


# Les Fragments ( cycle de vie )

Comme pour les activity, le fragment possède un cycle de vie et celui-ci est lié à celui de l'activity,

Il possède donc tous les état du cycle de vie d'une activty + d'autres etats propre à son cycle à lui

Le voici :


Activité
Fragment
Created
onCreate ( )
onAttach ( )
Le fragment a été associé à l'activité
onCreate ( )
Le fragment a été créé
onCreateView ( )
L'ensemble des vues du layout du fragment ont été créé.
On peut commencer à configurer notre interface et notre logique métier.
onActivityCreated ( )
Indique au fragment que son activité a correctement terminé sa propre méthode onCreate()
Started
onStart ( )
onStart ( )
Le fragment est enfin visible aux yeux de l'utilisateur
Resumed
onResume ( )
onResume ( )
Le fragment commence à interragir avec l'utilisateur !
Exécution normale
Paused
onPause ( )
onPause ( )
Le fragment n'interagit plus avec l'utilisateur...
Stopped
onStop ( )
onStop ( )
Le fragment n'est plus visible aux yeux de l'utilisateur
onDestroyView ( )
L'ensemble des ressources associées aux vues du fragment sont détruites
Destroyed
onDestroy ( )
onDestroy ( )
Le fragment est détruit
onDetach ( )
Le fragment n'est plus associé à son activité

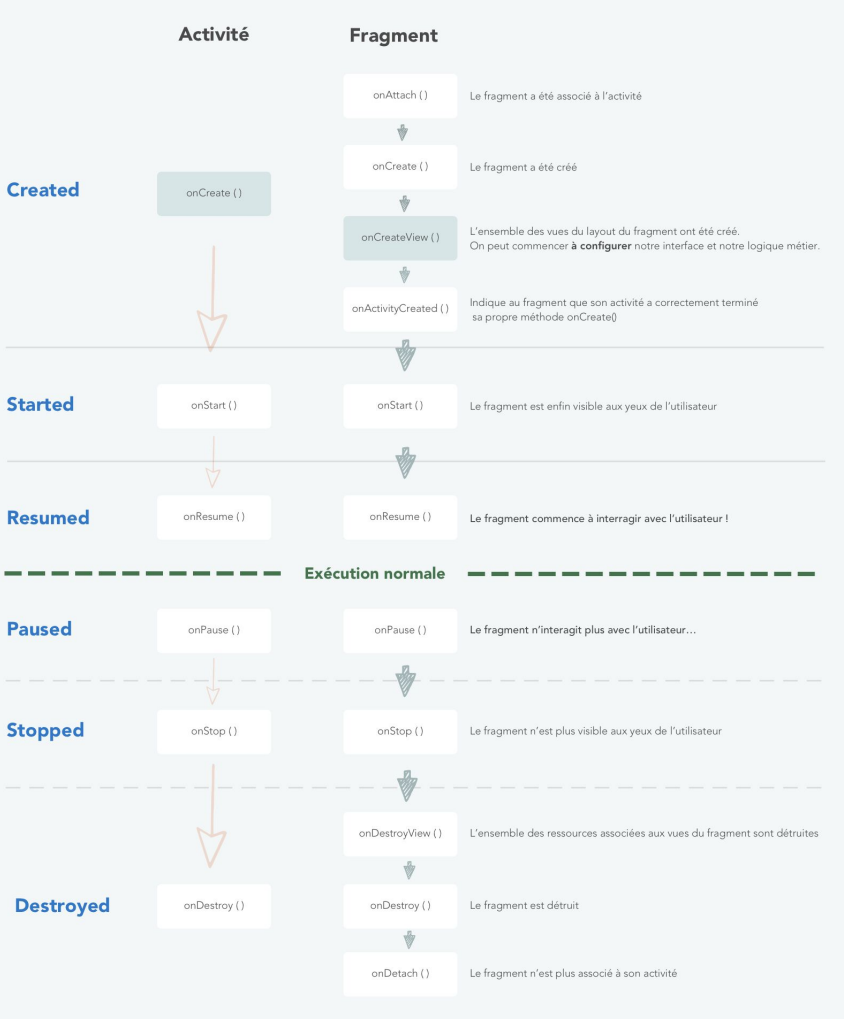
Le voici :

d’une manière générale

vous n’utiliserez que

les méthodes :

- onCreateView()
- onCreate()



# Les Fragments ( Implémentation )

- Premièrement on va créer un layout pour notre fragment :

```
<androidx.constraintlayout.widget.ConstraintLayout xmln
    xmlns:app="http://schemas.android.com/apk/res-auto":androic
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/layout_principal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

  <LinearLayout
      android:id="@+id/fragment_hello"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical" />
                                          >

</androidx.constraintlayout.widget.ConstraintLayout>
```

# Les Fragments ( Implémentation )

- Ensuite on va crée le Fragment :

```kotlin
package com.example.androidlearning

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment


class HelloFragment : Fragment() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_hello_world, container, attachToRoot: false)
    }

}
```

# Les Fragments ( Implémentation )

- Pour finir on va setup notre Fragment dans notre Activity grâce au FragmentManager

```
class MainActivity : AppCompatActivity() {


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val fragment = HelloFragment()

        val fragmentManager: FragmentManager = supportFragmentManager
        val fragmentTransaction: FragmentTransaction = fragmentManager.beginTransaction()

        fragmentTransaction.add(R.id.fragment_hello, fragment)
        fragmentTransaction.commit()

    }
```

# Les Fragments ( Implémentation )

- On aurait pu eviter d'utiliser le fragment manager en utilisant un fragment statique dans le XML

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="ht
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/layout_principal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/fragment_hello_world"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:name="com.example.androidlearning.CustomFragment"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

# Listes et adaptateurs

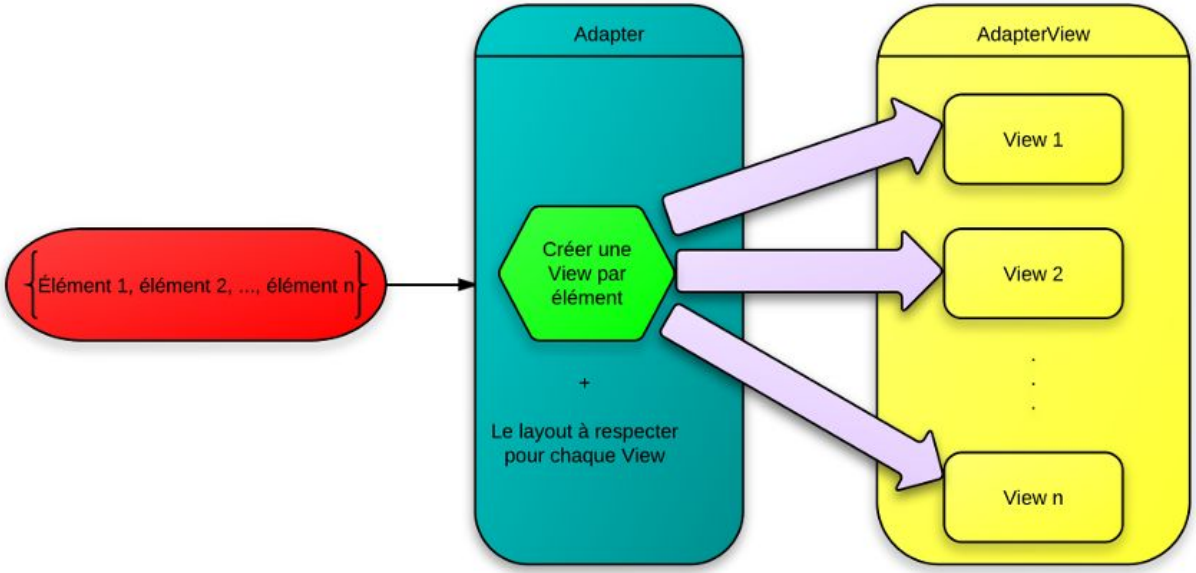
- La gestion de listes se divise en deux parties :

# Listes et adaptateurs

- La gestion de listes se divise en deux parties :
  - les Adapter ( gèrent les données mais pas leurs affichage ou leurs comportement )
  - Les AdapterView ( gèrent l'affichage et les interaction utilisateur )

# Listes et adaptateurs

- Comportement typique pour afficher une liste depuis un ensemble de données :



# Listes et adaptateurs

- Les adaptateurs fournis par Android ( pré programmé ) :
  - ArrayAdapter : permet d'afficher les informations de manières simple
  - SimpleAdapter : permet d'afficher plusieurs informations pour chaques élément ( On ne le verra pas car il est recommandé de créer son propre adaptateur pour ce genre de cas )
  - CursorsAdapter : adapte un contenu qui provient d'une base de donnée ( pareil que SimpleAdapter )

# Listes et adaptateurs

- Concernant l'ArrayAdapter il à besoin de 3 paramètre :
  - Le context d'éxécution ( Context )
  - Un layout ( qui sera utilisé pour chaque item de la liste )
  - et enfin l'ensemble de données qu'on veut afficher

```
spin_status.adapter = ArrayAdapter<STATUS>( context: this, android.R.layout.simple_list_item_1, STATUS.values())
```

# Listes et adaptateurs

- Les adaptateurs personnalisé sont donc ce qui est le plus recommandé dans le cas d’affichage d’objet complexe, ou dans tout autre cas qui ne nécessite pas un affichage simple d’une seule ligne

# Listes et adaptateurs

- Les adaptateurs personnalisé sont donc ce qui est le plus recommandé dans le cas d'affichage d'objet complexe, ou dans tout autre cas qui ne nécessite pas un affichage simple d'une seule ligne

- Nous allons donc voir ici comment créer nos propres adaptateurs !

# Listes et adaptateurs

- Pour ce faire nous avons besoin de 3 classe dont 2 qui sont obligatoire !

# Listes et adaptateurs

- Pour ce faire nous avons besoin de 3 classe dont 2 qui sont obligatoire !
  - Notre classe Adapter qui va hérité de BaseAdapter
  - Notre classe ViewHolder qui va concerner chaque item de la liste
  - Et enfin un Delegate qui servira à récupérer les actions utilisateurs sur notre liste

# Listes et adaptateurs

- Voici un exemple complet des trois classe de notre liste custom :

# Listes et adaptateurs

- Voici un exemple complet des trois classe de notre liste custom :


Ainsi qu'un layout pour notre ViewHolder

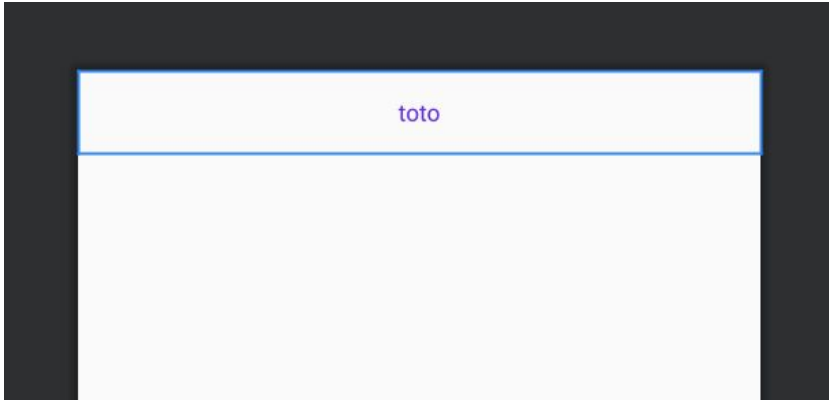
# Listes et adaptateurs

- Voici en détail le layout de notre ViewHolder :

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <TextView android:id="@+id/list_item_textView"
        android:layout_width="match_parent"
        android:layout_height="50dp"
        tools:text="toto"
        android:gravity="center"
        android:textColor="@color/colorPrimary"
        android:textAlignment="center"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        />

</androidx.constraintlayout.widget.ConstraintLayout>
```



# Listes et adaptateurs

- Voici en détail le code de notre ViewHolder :

```
class CustomViewHolder(private val context : Context, private val container: Int, private val listener: CustomViewDelegate)  {

    private val view: View =  LayoutInflater.from(context).inflate(container,  root: null)

    fun bindView(myString : String) : View {
        view.list_item_textView.text = myString
        view.tag = this
        manageOnClickItem(myString)
        return view
    }

    private fun manageOnClickItem(myString: String){
        view.setOnClickListener {  it: View!
            listener.itemSelected(myString)
        }
    }
}
```

# Listes et adaptateurs

- Voici en détail le code de notre Adaptateur Custom:

```
class CustomAdapter(private val context: Context, private val myList : ArrayList<String>, private val listener : CustomViewDelegate) : BaseAdapter() {

    override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {
        val vh = CustomViewHolder(context, R.layout.custom_list_item, listener)

        return vh.bindView(getItem(position))
    }

    override fun getItem(position: Int): String {
        return myList[position]
    }

    override fun getItemId(position: Int): Long {
        return 0
    }

    override fun getCount(): Int {
       return myList.size
    }

}
```

# Listes et adaptateurs

- Voici en détail le code de notre Delegate :

```
interface CustomViewDelegate {
    fun itemSelected(myString: String)
}
```

# Listes et adaptateurs

- Et enfin notre main :

```
class MainActivity : AppCompatActivity(), CustomViewDelegate {

    val myList : ArrayList<String> = ArrayList()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        myList.add("JEAN")
        myList.add("JACQUE")
        myList.add("MICHEL")


        val adapter = CustomAdapter( context: this, myList,  listener: this)
        list_view.adapter = adapter

    }

    override fun itemSelected(myString: String) {
        Toast.makeText( context: this,  text: "HELO FROM : $myString", Toast.LENGTH_LONG).show()
    }


}
```

# Listes et adaptateurs

- Et  Tada !!

# Les recyclerview

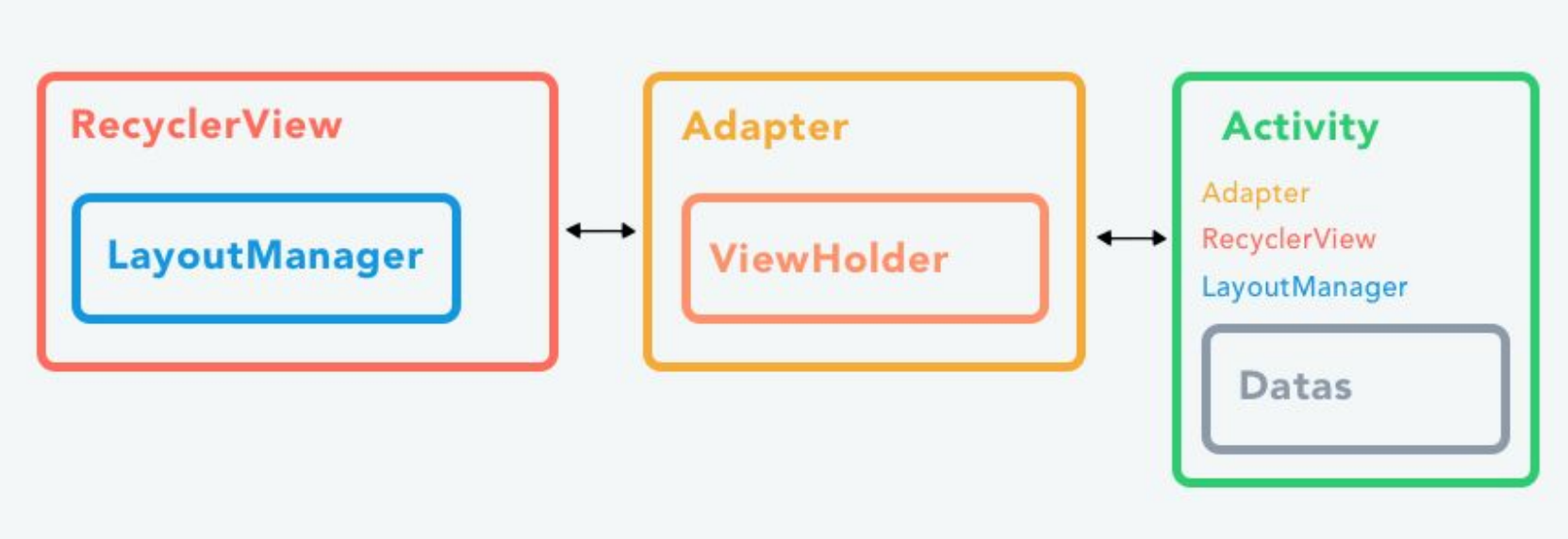
- Ce genre de liste custom est pratique lorsqu'on à pas beaucoup d'élément à afficher, cependant en cas de données qui dépasse l'écran, la liste devient assez lourde car les éléments ne sont pas recyclé

# Les recyclerview

- Ce genre de liste custom est pratique lorsqu'on à pas beaucoup d'élément à afficher, cependant en cas de données qui dépasse l'écran, la liste devient assez lourde car les éléments ne sont pas recyclé
- Android répond à ce problème grâce à ce qu'on appelle des recyclerview ( représente 80% des listes utilisables dans les apps sur le store )

# Les recyclerview

- La recyclerview utilise les mêmes classes que pour une liste normal :



# Les recyclerview

- Pour l’implémenter tout d’abord il faut ajouter cette ligne dans le build.gradle :

```
implementation 'androidx.recyclerview:recyclerview:1.1.0'
```

# Les recyclerview

- Ensuite on va changer le code de notre ViewHolder comme ceci :

```
class CustomViewHolder(private val container: View, private val listener: CustomViewDelegate) :
    RecyclerView.ViewHolder(container) {


    fun bindView(myString: String, position: Int) {
        container.list_item_textView.text = myString
        manageOnClickItem(myString)
    }


    private fun manageOnClickItem(myString: String){
        container.setOnClickListener { it: View!
            listener.itemSelected(myString)
        }
    }
}
```

# Les recyclerview

- Dans le layout de l’activity on va maintenant utiliser une RecyclerView:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_custom"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

# Les recyclerview

- l'Adaptateur va maintenant hérité de RecyclerView.Adapter<>() et va donc drastiquement changer

```
class CustomAdapter(private val context: Context, private val myList : ArrayList<String>, private val listener
: CustomViewDelegate) : RecyclerView.Adapter<CustomViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CustomViewHolder {
        val view : View! = LayoutInflater.from(context).inflate(R.layout.custom_list_item, parent, attachToRoot: false)
        return CustomViewHolder(view, listener)
    }

    override fun onBindViewHolder(holder: CustomViewHolder, position: Int) {
        holder.bindView(myList[position], position)
    }

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

}
```

# Les recyclerview

- Pour finir le main va maintenant implémenté la recyclerview avec notre adaptateurs custom

```
val adapter = CustomAdapter( context: this, myList,  listener: this)

recycler_custom.layoutManager = LinearLayoutManager( context: this, LinearLayoutManager.VERTICAL,  reverseLayout: false)

recycler_custom.adapter = adapter
```

# Les recyclerview

- Et  Tada !! ( le même résultat mais optimisé )

# Les dialogs

- Une boîte de dialogue est une petite fenêtre qui passe au premier plan pour informer, avertir, prévenir

# Les dialogs

- Une boîte de dialogue est une petite fenêtre qui passe au premier plan pour informer, avertir, prévenir
- On les utilises principalement pour afficher des informations, prévenir d'une erreur ou afficher un chargement

# Les dialogs

- Il est très simple de créer des boîtes de dialogue sur Android grâce à l'objet AlertDialog

# Les dialogs

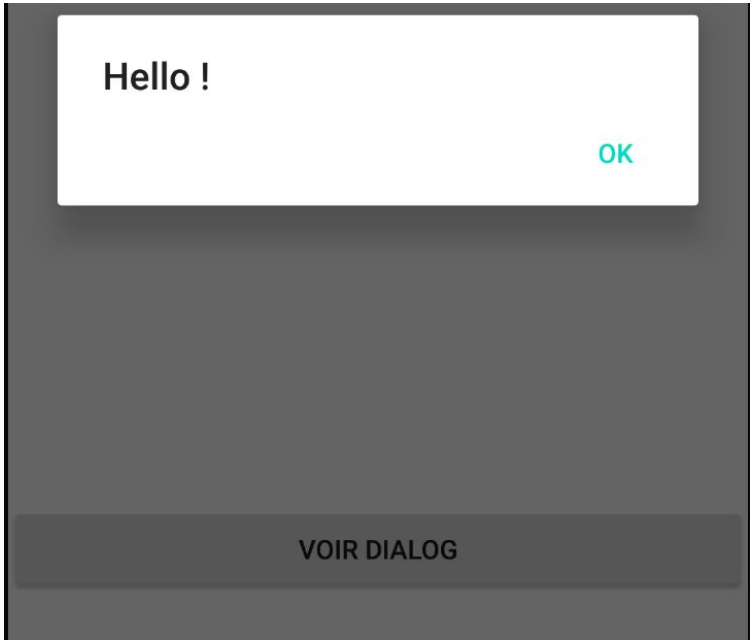
- Il est très simple de créer des boîtes de dialogue sur Android grâce à l'objet AlertDialog
- Voici un exemple :

```
btn_show_dialog.setOnClickListener { it: View!
    AlertDialog.Builder( context: this)
        .setNegativeButton( text: "Ok",  listener: null)
        .setTitle("Hello !")
        .show()

}
```

# Les dialogs

- Le résultat :



# Les dialogs

- Customisation de la dialog ( tout d'abord créer un layout pour elle ) :

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:background="@color/colorAccent">

    <ImageView
        android:id="@+id/img_dialog"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:src="@mipmap/ic_launcher"/>

    <TextView
        android:id="@+id/tv_dialog_message"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        tools:text="HELLO DE LA DIALOG"
        android:layout_marginLeft="10dp"
        app:layout_constraintLeft_toRightOf="@id/img_dialog"
        app:layout_constraintTop_toTopOf="@id/img_dialog"
        app:layout_constraintBottom_toBottomOf="@id/img_dialog"
        />


</androidx.constraintlayout.widget.ConstraintLayout>
```

# Les dialogs

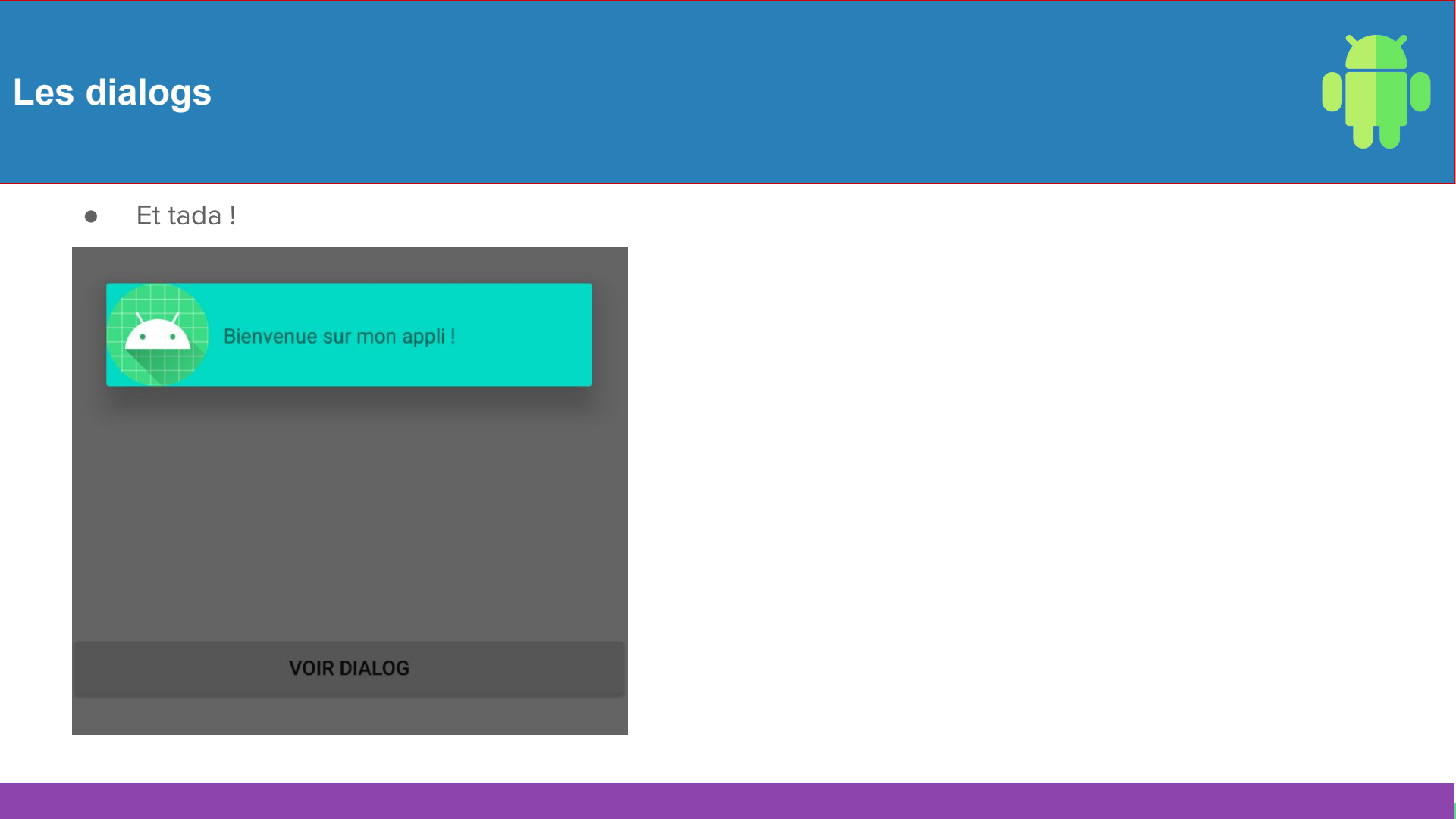
- Ensuite on va travailler avec cette vue qu'on va ensuite setup sur l'objet AlertDialog :

```
btn_show_dialog.setOnClickListener { it: View!

    val view : View! = LayoutInflater.from( context: this).inflate(R.layout.dialog_view, root: null)
    view.tv_dialog_message.text = "Bienvenue sur mon appli !"
    AlertDialog.Builder( context: this)
        .setView(view)
        .show()

}
```

# Les dialogs



- Et tada !

# Les dialogs

- Il y’a plein d’implémentation possible avec l’AlertDialog et aussi plein de méthode de modifications de celle - ci en voici quelques une en vrac :
  - setCancelable(boolean) : si la dialog est cancelable on peut la kill grâce au bouton retour du téléphone
  - setIcon(int) ou setIcon(Drawable) : permet de setup un icon
  - setMessage(String) : permet de setup un message
  - setTitle(String) : permet de setup un titre
  - setView(View) : permet de gérer toute la vue de la dialog dans un autre objet ( plus customisable )
  - setPositive/Negative/NeutralButton(String, Listener) : permet de setup les boutons d’actions de la dialog et aussi la fonction qui sera jouer pour chaque bouton

# Les imageView

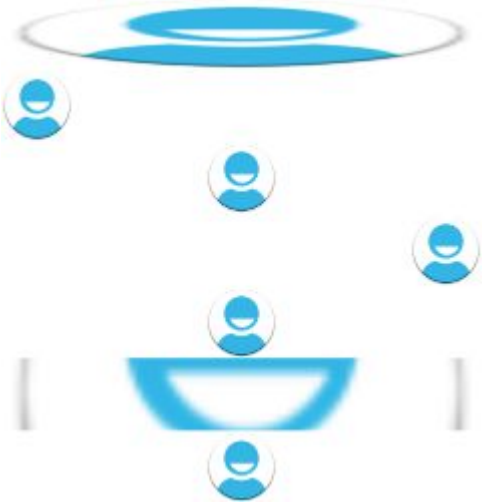
- Dans cette dialog ont à utiliser une imageView mais concrètement y’a t-il des choses à savoir dessus ?

# Les imageView

- Dans cette dialog ont à utiliser une imageView mais concrètement y’a t-il des choses à savoir dessus ? La réponse est oui !

# Les imageView

- Tout d'abord des choses simples à savoir :
  - L'attribut android:src permet de choisir l'image qui va être utilisé dans notre imageView
  - L'attribut android:scaleType permet de préciser comment notre image va être agrandie à un moment pour s'adapter à son parent ou pas, un exemple ci dessous



# Les imageView

- Voici les différentes options que peux prendre android:scaleType :
  - fitXY : l'image est redimensionnée de manière variable, elle prendra le plus de place possible
  - fitStart : l'image est redimensionnée de manière constante et ira se placer en haut à gauche de l'imageView
  - fitCenter : pareilque start mais cette fois elle ce placera au centre de l'imageView
  - fitEnd : pareil que start mais en bas à droite
  - center : l'image n'est pas redimensionnée, et ce placera au centre
  - centerCrop : l'image est redimensionnée de manière constante mais elle pourra dépasser du cadre de l'imageView
  - centerInside : l'image est redimensionnée de manière constante et prendra le plus de place possible mais elle restera dans le cadre de l'imageView

# Les imageView

- Ainsi que les deux type de src qui peuvent être utilisé :
  - Un Bitmap : une image matricielle classique
  - Un Drawable : objet qui représente tout e qui peut être dessiné. Autant une image qu'un ensemble d'images, qu'une forme, etc

# Les imageView

- Notez qu'il existe aussi un widget ImageButton, qui n'est simplement qu'un objet dérivé de imageView mais cliquable ( tout comme Button est dérivé de textView )

# Un peu d’exercice

- Pour cet exercice je vous propose de réutiliser tout ce qui à été vue dans le cours et de crée une application comme suit :
  - Une activité qui comporte un fragment
  - Dans ce fragment il y’a une liste d’objets Person ( ListView ou RecyclerView )
  - Quand je clique sur une personne dans la liste une boîte de dialogue s’ouvre me disant si la personne est majeur ou non
  - La dialog doit avoir un fond de couleur rouge si non et vert si oui, en plus de m’afficher un message
  - Aussi je veut que la boîte de dialogue m’affiche une image ( peu importe laquelle ), à côté du texte