PROGRAMMATION DE COMPOSANTS MOBILES (ANDROID)

RecyclerView

WIESLAW ZIELONKA

www.irif.fr/~zielonka

RecyclerView

`ListView` n'est pas adapté pour des longues listes surtout quand l'affichage des éléments de la liste est customisé et change d'un élément à l'autre ou les éléments de la liste changent suite aux actions de l'utilisateur.

Par exemple : chaque item de la liste est composé de

- TextView
- CheckBox

et l'utilisateur sélectionne les éléments de la liste avec un click sur les items.

Ensuite il supprime les éléments sélectionnés en appuyant sur un Button.

Une liste normale ListView ne peut pas s'acquitter correctement de cette tâche. Les Views qui affichent les items de la listes sont réutilisés quand l'utilisateur fait défiler la liste et l'état de CheckBoxes n'est pas mis à jour correctement.

`RecyclerView` permet mieux gérer l'affichage de listes et est recommandé à la place de ListView

RecyclerView

Dans le fichier build.gradle (module app)

de l'application il faut ajouter dans la section dependencies la dépendance:

```
dependencies {
implementation "androidx.recyclerview:recyclerview:1.2.1"
// For control over item selection of both touch and mouse driven selection
implementation "androidx.recyclerview:recyclerview-selection:1.2.0-alpha01"
}
```

(les versions peuvent changer, voir la documentation de RecyclerView)

définir un RecyclerView

dans le fichier layout de l'activité ou du fragment

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler"

    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:layout_marginTop="10dp"
    android:layout_marginRight="10dp"
    android:scrollbarSize="10dp"

    android:scrollbars="vertical"
    android:clickable="true"
     . . . . . .
/>
```

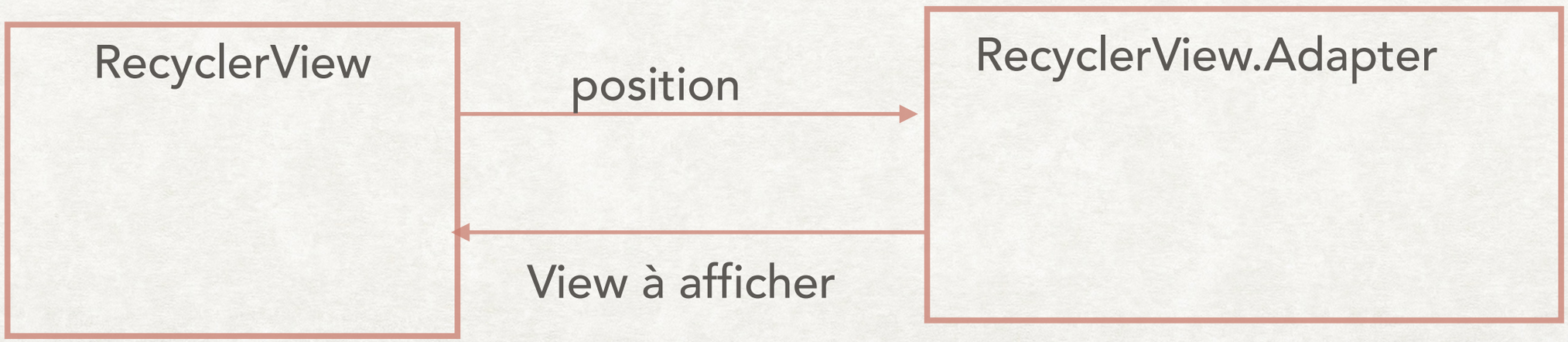
RecyclerView.Adapter

`RecyclerView` est responsable de l'affichage de données mais il ne maintient pas de données. Les données sont fournies par un `RecyclerView.Adapter.`
Chaque fois quand RecyclerView a besoin d'afficher une nouvelle donnée il s'adresse à `RecyclerView.Adapter.`

`RecyclerView.Adapter` ne fournit pas de données brutes mais il fournit au `RecyclerView` une `View` complète déjà pré-remplie avec les données à afficher. `RecyclerView` est uniquement responsable de l'affichage des `View's` fournies par `le RecyclerView.Adapter.`

Le schéma simplifié de communication entre RecyclerView et RecyclerView.Adapter:

1. RecyclerView demande une Vue à afficher à la position "position"
2. RecyclerView.Adapter envoie à RecyclerView une View à afficher, une view qui contient déjà la donnée à afficher

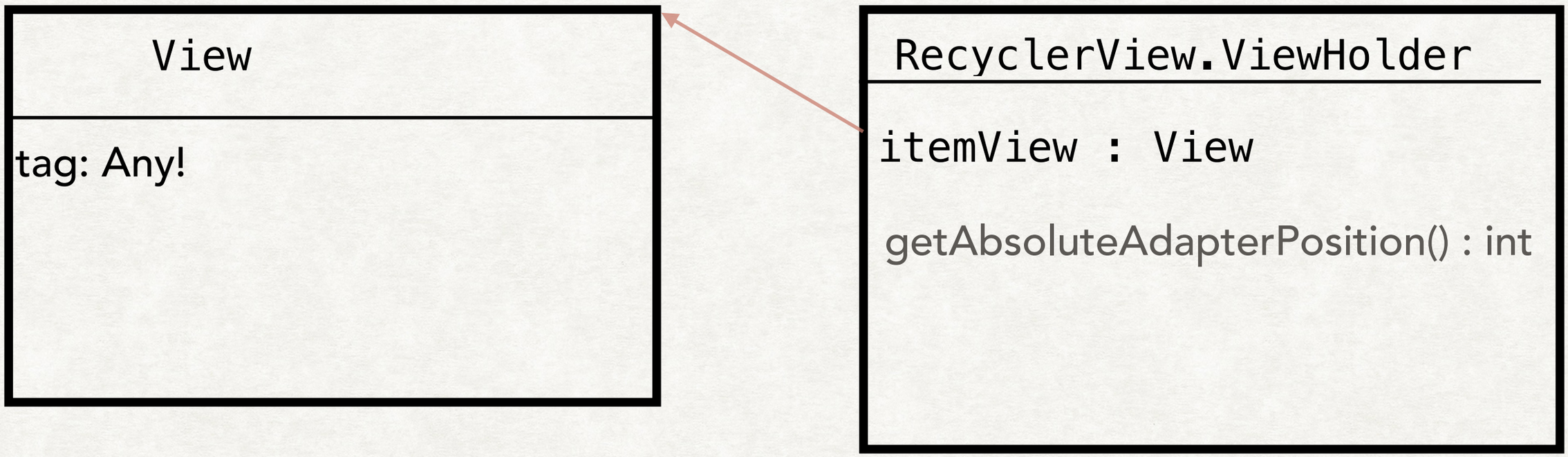


RecyclerView.Adapter

Au lieu de fournir directement une `View`, `RecyclerAdapter` envoie vers le `RecyclerView` un objet `RecyclerView.ViewHolder (classe abstraite)` qui contient une propriété

`itemView: View`

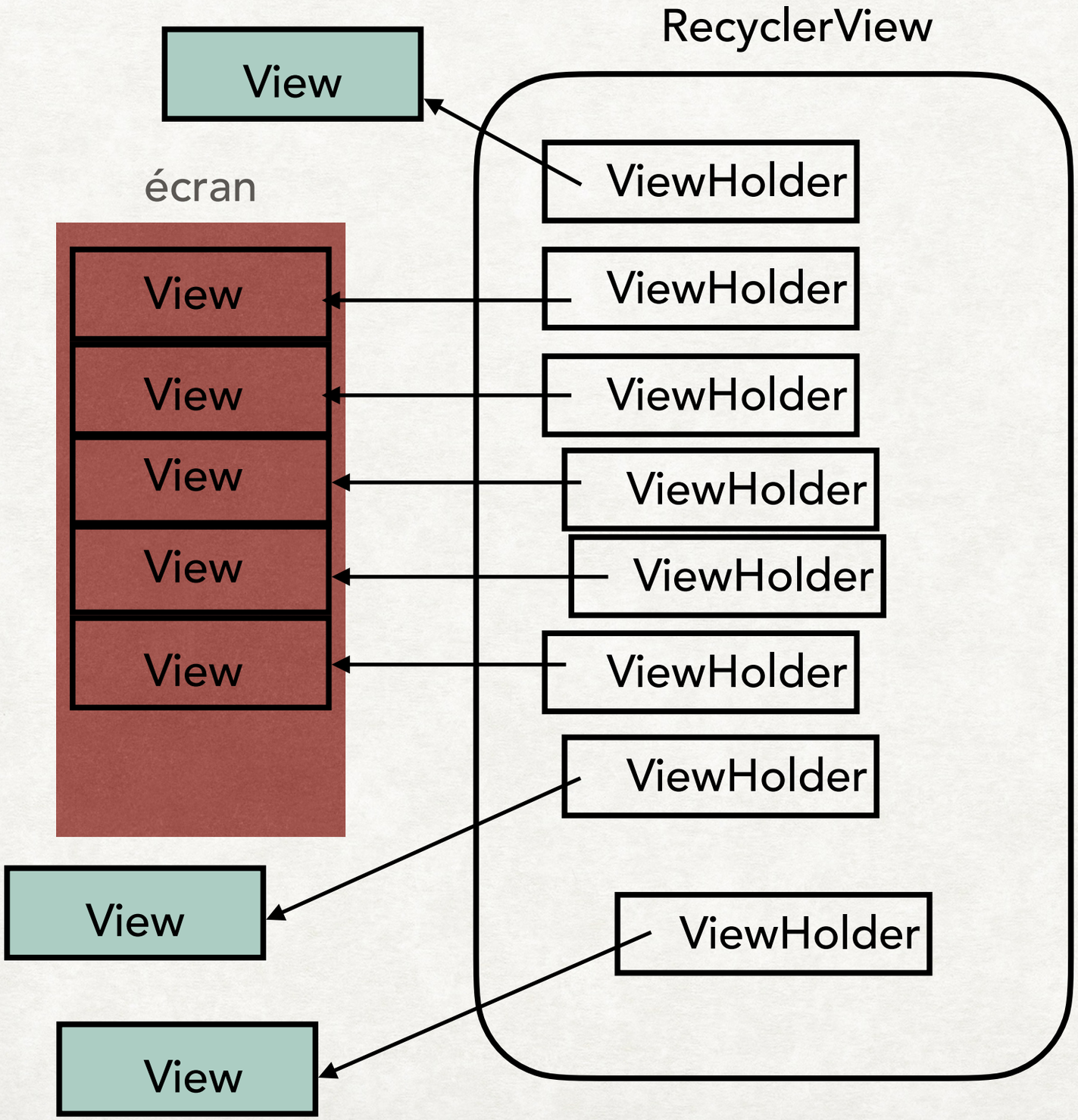
`La propriété itemView contient` une référence vers la `View.`



A chaque `View` nous pouvons associer un objet `tag (Any` est l'ancêtre de toutes les classes kotlin et remplace `Object` de java). Le tag permet de stocker dans une View des informations supplémentaires.

`absoluteAdapterPosition` donne la position de l'élément affiché, la position dans la liste de tous les objets à afficher, pas la position sur l'écran. Cette valeur a un sens seulement quand la view est effectivement affiché sur l'écran.

RecyclerView



Pour faciliter le défilement RecyclerView garde un ensemble de ViewHolders plus grand que le nombre de View's visibles sur l'écran.
Pendant le défilement RecyclerView réutilise les couples ViewHolder---View.
Si itemView de ViewHolder contient une référence vers une view qui n'est plus visible sur l'écran alors ce ViewHolder et réutilisé, la View pointée par itemView sera réutilisée et remplies de nouvelles données.

Ma création de Views a un coût élevé, la réutilisation de View existant économise la mémoire et permet un affichage fluide.

RecyclerView.Adapter

`RecyclerView`

- gère l'ensemble de couples `View <- ViewHolder`,
- `gère` le défilement de la liste sur l'écran, mais
- RecyclerView n'a pas de données à afficher et ne mets pas de données dans les Views
- RecyclerView ne construit pas les couples View <- ViewHolder.

`RecyclerView.Adapter`

- maintient des structures de données qui contiennent les données à afficher
- `RecyclerView.Adapter` construit à la demande de `RecyclerView` les couples `View <- ViewHolder`
- à la demande de `RecyclerView`, `RecyclerView.Adapter` remplit avec les données les `Views` à afficher,
- dans une classe interne RecyclerView.Adapter implémente une classe dérivée de `ViewHolder.`

RecyclerView.Adapter

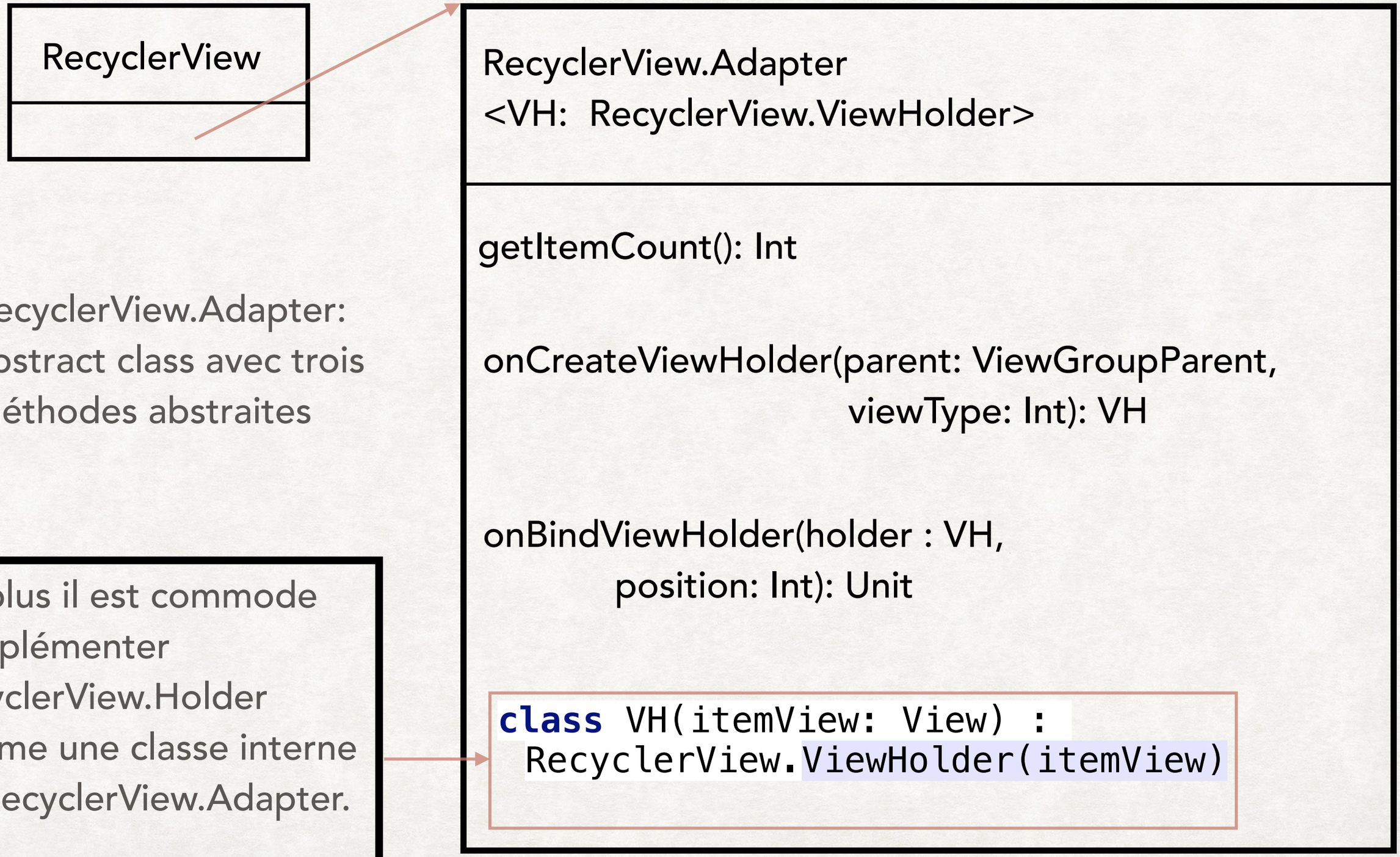
RecyclerView.Adapter est une classe abstraite paramètre par une classe qui est dérivée par de d'une autre classe abstraite RecyclerView.ViewHolder :

public abstract class RecyclerView.Adapter<VH extends **RecyclerView.ViewHolder**>

Pour implémenter RecyclerView vous devez implémenté une classe dérivée de RecyclerView.Adapter.

Mais RecyclerView.ViewHolder est aussi une classe abstraite donc vous **devez impérativement** définir une classe dérivée de RecyclerView.ViewHolder et utiliser cette classe comme paramètre de votre implémentation de RecyclerView.Adapter.

RecyclerView.Adapter



RecyclerView.Adapter: abstract class avec trois méthodes abstraites

De plus il est commode d'implémenter RecyclerView.Holder comme une classe interne de RecyclerView.Adapter.

RecyclerView.Adapter

Dans RecyclerView.Adapter vous devez implémenter les trois méthodes abstraites de cette classe et définir le holder:

```
class MyRecycleAdapter( paramètres de constructeur ) :
RecyclerView.Adapter<MyRecycleAdapter.VH>(){

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

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

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

// définir le holder

class VH(itemView: View) : RecyclerView.ViewHolder(itemView) { ... }

}//fin MyRecyclerViewAdapter
```

Bien sûr vous pouvez ajouter d'autres paramètres dans le constructeur de holder, mais `itemView` doit toujours figurer parmi les paramètres.

RecyclerView.Adapter

fonctions abstraites à implémenter

getItemCount() : Int

retourne le nombre total d'éléments à afficher dans le RecyclerView (la longueur de la liste à afficher)

---

onCreateViewHolder(parent ViewGroupParent, viewType: Int): ViewHolder

la méthode appelée par RecyclerView quand RecyclerView a besoin d'un nouveau couple View <- ViewHolder. La méthode doit construire ce couple et retourner le ViewHolder qui contient la référence vers la View

---

onBindViewHolder(holder: ViewHolder , position: Int) : Unit

quand RecylerView a besoin de remplir une View avec les données, il fait appel à cette fonction en lui passant en paramètre le holder correspondant et la position sur dans RecyclerView

---

```
class ViewHolder: RecyclerView.ViewHolder
```

Le plus simple est définir la classe ViewHolder comme une classe interne de `RecylerView.Adapter` (mais on peut aussi bien définir ViewHolder à l'extérieur de Adapter).

RecyclerView.LayoutManager pour RecyclerView

`RecyclerView.LayoutManager` positionne les items dans RecyclerView. Il y a trois implémentations :

- LinearLayoutManager
- StaggeredGridLayourManager
- GridLayoutManager

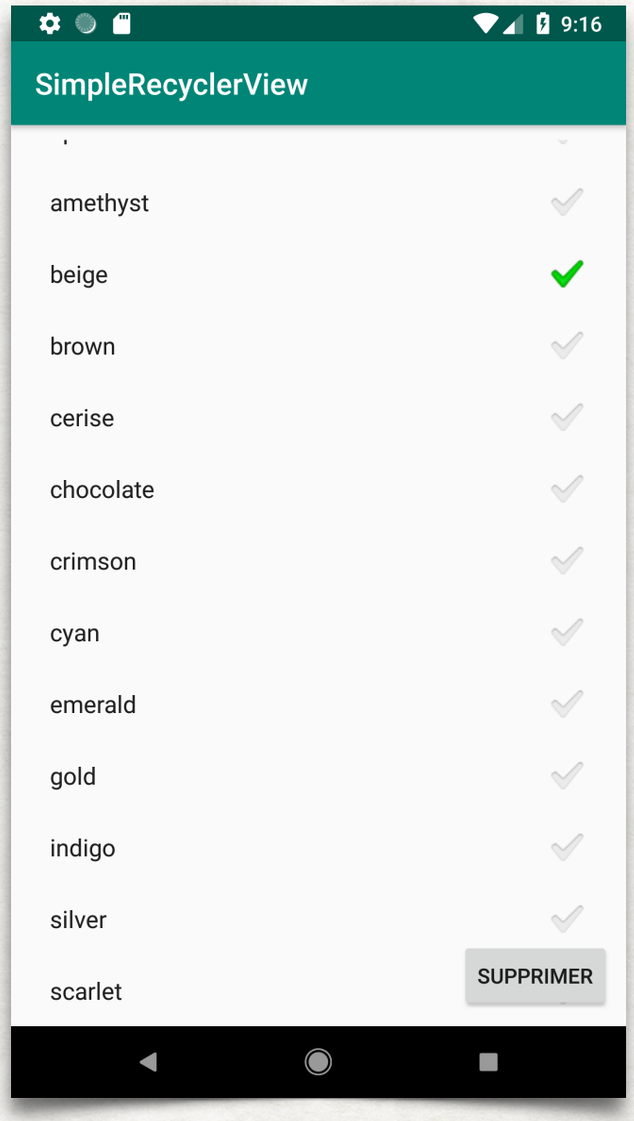
Dans Activity il faut indiquer le manager à utiliser :

```
recyclerView.layoutManager = LinearLayoutManager(this)
```

Exemple de RecyclerView

Dans l'exemple nous avons une liste d'items affichés dans RecyclerView et un bouton "supprimer".

L'utilisateur choisit des items sur la liste et quand il clique sur "supprimer" les items sélectionnés sont supprimés.



associer RecyclerView.Adapter à RecyclerView

```
class MainActivity : AppCompatActivity() {

  lateinit var adapter: MyRecycleAdapter

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

      /* récupérer la référence vers RecyclerView
      val recyclerView =  findViewById(R.id.recycler) as RecyclerView

      recyclerView.layoutManager = LinearLayoutManager(this)

      /* Une MutableList de Strings à afficher dans RecyclerView obtenue
       * depuis les ressources */
      val colors = resources.getStringArray(R.array.colors).toMutableList()

      adapter = MyRecycleAdapter(colors) /* construire adapter*/

      /* associer adapter à RecyclerView
      recyclerView.adapter = adapter
}
```

gestion de bouton SUPPRIMER dans l'activité

```
    /* au lieu de listener on implémenté directement la méthode
     * callback, le nom de la méthode est spécifier dans le layout  */

    fun supprimer(view: View?) {

        /* appeler une fonction de Adapter qui supprime
         * tous les items dans l'état "checked" */
        adapter.removeChecked()
    }
}// fin MainActivity
```

Dans le fichier layout le bouton est défini comme :

```
<Button
    android:id="@id/button"
    android:onClick="supprimer"

    android:text="supprimer"
    ....
/>
```

le nom de la méthode qui est appelée quand on clique sur le bouton (plus besoin de définir un listener)

RecyclerView.Adapter

```
class MyRecycleAdapter(val colors: MutableList<String>) :
                                RecyclerView.Adapter<RecyclerView.ViewHolder>(){
    // le paramètre colors du constructeur : une propriété de cet adapter

    /* checked : propriété qui contient la liste les Strings colors
     * sélectionnés par l'utilisateur */

    val checked = mutableListOf<String>()


    /* un lister à associer à chaque View dans RecyclerView,
     * activé quand l'utilisateur
     * sélectionne un élément dans RecyclerView */
    var listener = View.OnClickListener { view ->
        (view as CheckedTextView).toggle()
        if (view.isChecked) {
            checked.add(view.text.toString())
        } else {
            checked.remove(view.text.toString())
        }
    }

/* removeChecked() est appelé pour supprimer les items "checked" de l'adapter
  * Cette fonction n'existe pas dans la classe RecyclerView.Adapter mais
  * mais j'en ai besoin  dans mon Activity (transparent précédent)*/
fun removeChecked() {
   colors.removeAll(checked)
   checked.clear()
   notifyDataSetChanged()
}
```

Si le listener doit accéder au holder il devra être construit dans onCreateViewHolder()

onCreateViewHolder()

RecyclerView apelle onCreateViewHolder quand il a besoin d'un nouveau couple (View, Holder)

```
override fun onCreateViewHolder(parent: ViewGroup,
                                viewType: Int): RecyclerView.ViewHolder {

    //créer une View d'un élément de la liste à partir de fichier layout xml
    val v = LayoutInflater
        .from(parent.getContext())
        .inflate(android.R.layout.simple_list_item_checked,
                 parent, false)

    /*installer le même listener sur chaque View */
    v.setOnClickListener(listener)

    /*créer et retourner le ViewHolder  VH. Le paramètre du
constructeur est la View créée par inflate */
    return VH(v)
}
```

Le premier paramètre de inflate() indique le fichier xml layout qui décrit la View utilisée pour un item de la liste à afficher. Si le fichier layout est un fichier fourni par android alors il y a le préfix "android", si c'est un fichier dans le répertoire layout de votre application (un fichier que vous avez fabriqué vous-même) alors pas de préfix "android"

onCreateViewHolder() de RecyclerView.Adapter

```
override fun onCreateViewHolder(parent: ViewGroup,
                                viewType: Int): RecyclerView.ViewHolder {

    //créer View d'un élément de la liste à partir de fichier layout xml
    val v = LayoutInflater
        .from(parent.getContext())
        .inflate(android.R.layout.simple_list_item_checked, parent, false)
        as CheckedTextView
.....
}
```

android.R.layout.*simple_list_item_checked* - reference vers le fichier layout qui décrit un item de la liste. Le préfix **android.** indique que dans cet exemple il s'agit d'un fichier déjà prédéfinis dans Android.

La page web
https://android.googlesource.com/platform/frameworks/base/+/master/core/res/res/layout
donne les sources de tous les fichiers layout qui sont déjà prédéfinis dans android. En regardant le fichier **simple_list_layout_checked.xml** on trouve qu'il contient une seule View : CheckedTextView.

inflate() construit une View à partir de fichier xml. Dans notre exemple l'objet à la racine de fichier layout est un **CheckedTextView** donc dans v on récupère une référence vers un CheckedTextView (d'où le cast : as CheckedTextView)

onCreateViewHolder : avec les item views customisés

```
override fun onCreateViewHolder(parent: ViewGroup,
                    viewType: Int): RecyclerView.ViewHolder {

    //créer View d'un élément de la liste à partir de fichier layout xml
    val v : LinearLayout = LayoutInflater
        .from(parent.getContext())
        .inflate(R.layout.my_item_layout, parent, false)

    /* récupérer le CheckBox */
    val checkBox = v.findViewById(R.id.check) as CheckBox

    /* définir listener */
    var listener = View.OnClickListener { view ->   ..... }

    /* associer le listener au CheckBox */
    checkBox.setOnClickListener( listener )

    /* créer le ViewHolder etc ..*/
```

On écrit notre propre fichier layout, par exemple `my_item_layout.xml` (en le plaçant dans le répertoire layout) . La référence vers le fichier n'a pas de préfix android.
La view qu'on obtient c'est la View à la racine de layout.

// fichier my_item_layout.xml :

```
<LinearLayout ...>
   <TextView .../>
   <TextView .../>
   <Checkbox id="@+/check  .../>
</LinearLayout>
```

alors la View v est un objet de type `LinearLayout`

onBindViewHolmder() dans RecyclerView.Adapter

la méthode onBindViewHolder est appelée par RecyclerView quand l'item à la position "position"
devient visible. Le premier paramètre : le holder à remplir est fourni par le RecyclerView.

```
override fun onBindViewHolder(holder: RecyclerView.ViewHolder,
                              position: Int) {

    /* recuperer la View :
     * holder.itemView c'est la View associée à ce holder */
    val checkedTextView = holder.itemView as CheckedTextView

    /* mettre le string colors[position] dans la View */
    checkedTextView.text = colors[position]

    /* mettre à jour la propriété checked de la View */
    checkedTextView.isChecked = checked.contains(colors[position])
}
```

changer le type pour qu'il correspond à la racine de votre layout

RecyclerView.Adapter

La fonction getItemCount() doit retourner le nombre total d'éléments affichables dans RecyclerView. C'est **n'est pas** le nombre d'éléments affichés en ce moment sur l'écran mais bien le nombre total de tous les éléments à afficher.

```
override fun getItemCount(): Int = all.size
```

RecylerView.Adapter

Créer le ViewHolder comme une classe interne sde RecyclerView.Adapter :

```
class VH(itemView: View) : RecyclerView.ViewHolder(itemView)
```

Il n'est pas nécéssaire que le `ViewHolder` soit défini comme la classe interne de RecyclerView.Adapter.
On peut aussi bien le définir comme une classe à part.

Mais holder est un paramètre de `RecyclerViewAdapter` donc il est bien commode de le définir avec `RecyclerView.Adapter.` `De cette façon le holder pourra accéder aux attributs de l'adapter.`
Ceci termine la définition de l'adapteur.

RecylerView.Adaper - remarques

Souvent l'objet o à afficher dans `RecyclerView` n'est pas un `String`.

Le problème : l'utilisateur clique sur un item dans le `RecyclerView`.

Mais le contenu d'un item affiché dans le `RecyclerView` ne permet pas d'identifier l'objet sélectionné (où il est difficile de faire une correspondance entre ce qui est affiché et l'objet représenté par la View).

Le remède : associée à la `View` l'objet affiché en utilisant un `tag`.

RecylerView.Adapter avec des objets quelconques

L'utilisation de tag dans onBindViewHolder

```
onBindViewHolder( holder: RecycleAdapter.ViewHolder,
                  position : Int) {

    /* recuperer la View,
    * holder.itemView contient la référence vers la View associée au holder */
    var view = viewHolder.itemView;

    /* récupérer objet à afficher dans la View */
    val o = colors[i];

    view.tag = o; /* mémoriser la référence vers l'objet
                   * à afficher comme un tag de la View */

    //remplir la vue en utilisant l'objet o
    mettre_a_jour_la_view( view, o )
}
```

Ici on suppose que la liste "colors" est une liste d'objets quelconques dont la représentation est affichée dans RecyclerView.

RecylerView.Adapter avec des objets quelconques

l'utilisation de tags dans le listener:

```
var listener = View.OnClickListener { view ->

    /* changer l'état de l'objet quand l'utilisateur
     * sélectionne ou désélectionne la View */

    change_object_state( view.tag )
}
```

RecyclerView et changement de configuration

L'exemple précédent ne marche pas correctement quand on tourne l'appareil.
Problème : une liste de Strings ne peut pas être sauvegardée dans un Bundle.

Mais il est possible de stocker dans un Bundle une ArrayList<Int>.

RecyclerView et changement de configuration

```
private const val ALL_INDICES = "all_indices"
private const val CHECKED_INDICES = "checked_indices"

class MainActivity : AppCompatActivity() {

    val adapter: MyRecycleAdapter by lazy{ MyRecycleAdapter(colors,
                                                            colorIndices,
                                                            checkedIndices )}
    val recyclerView by lazy { findViewById(R.id.recycler) as RecyclerView }

    /* les données */
    val colors by lazy { resources.getStringArray(R.array.colors).toList() }
    val colorIndices = ArrayList<Int>()
    val checkedIndices = ArrayList<Int>()
```

/* `colors` : array de tous les Strings (noms de couleurs)
`colorIndices` : une liste d'indices de tableau colors. Contient les indices de noms que nous voulons afficher
`checkedIndices` : une liste d'indices de tableau colors, contient les indices des éléments sélectionnés par utilisateur
*/

RecyclerView et changement de configuration

suite de MainActivity :

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

    if (  savedInstanceState == null  ){
        colorIndices.addAll( colors.indices )
    }else{
        colorIndices.addAll( savedInstanceState.getIntegerArrayList( ALL_INDICES ) ?:
colors.indices )

        savedInstanceState
.getIntegerArrayList(CHECKED_INDICES)?.also{ checkedIndices.addAll( it  ) }
    }

    recyclerView.hasFixedSize() /* pour améliorer les pérformances*/
    recyclerView.layoutManager = LinearLayoutManager(this)

    Log.d("CheckBoxList", "nombre =  ${colors.size}")

    recyclerView.adapter = adapter
}
```

RecyclerView et changement de configuration

suite de MainActivity :

/* fonction callback de bouton supprimer */

```
  fun supprimer(view: View?) {
        Toast.makeText(
            this,
            "suppression de ${adapter.checkedIndices.size} items",
            Toast.LENGTH_LONG
        ).show()
        adapter.removeChecked()
    }

/* sauvegarde pour le changement de configuration */
    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        outState.putIntegerArrayList( ALL_INDICES, colorIndices)
        outState.putIntegerArrayList( CHECKED_INDICES, checkedIndices)
    }
}
```

RecyclerView et changement de configuration définition de RecylerAdapter

```
class MyRecycleAdapter(val colors: List<String>,
                       val colorIndices : ArrayList<Int>,
                       val checkedIndices : ArrayList<Int>) :
RecyclerView.Adapter<RecyclerView.ViewHolder>(){

    //checked : contient les Strings colors  à l'état "checked"
    //val checked = ArrayList<String>()

    //View.OnClickListenerr possède la méthode onClick de type View -> Unit
    val listener  = { view : View ->
         /* il faut changer la valeur de la propriété checked explicitement */
         (view as CheckedTextView).toggle()

        /* mettre à jour la liste de colors qui sont à l'état "checked" */
        if (view.isChecked) {
            checkedIndices.add(  view.tag as Int )
        } else {
            checkedIndices.remove( view.tag as Int )
        }
        Unit
    }
```

Le tag de chaque view donne indice dans le tableau colors

RecyclerView et changement de configuration définition de RecylerAdapter

```
/* fonction appelle dans MainCtivity quand on clique sur le bouton supprimer */

fun removeChecked() {
    colorIndices.removeAll( checkedIndices )
    checkedIndices.clear()
    notifyDataSetChanged()
}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RecyclerView.ViewHolder {

    //créer View d'un item de la liste à partir de fichier layout xml
    val v = LayoutInflater
        .from(parent.getContext())
        .inflate(android.R.layout.simple_list_item_checked, parent, false)

    /* installer le listener sur chaque View */
    v.setOnClickListener(listener)

    //créer et retourner le ViewHolder
    return VH(v)
}
```

RecyclerView et changement de configuration définition de RecylerAdapter

```
 override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int)
{

        /* recuperer la View :
         * holder.itemView c'est la View associée à ce holder */
        val checkedTextView = holder.itemView as CheckedTextView
        checkedTextView.tag = colorIndices[position]

        /* mettre la valeur colors[position] dans la View */
        checkedTextView.text = colors[colorIndices[position]]

        /* mettre à jour la propriété checked de la View */
        checkedTextView.isChecked =
                        checkedIndices.contains(colorIndices[position])
    }

    override fun getItemCount(): Int = colorIndices.size

    //ViewHolder : ici il ne contient rien d'utile sauf une référence vers la
View
    class VH(itemView: View) : RecyclerView.ViewHolder(itemView)
}
```

RecyclerView.ViewHolder

`ViewHolder` possède les propriétés

`View itemView` qui est la référence vers la view associée au holder.
`Int absoluteAdapterPosition` la position de l'objet affiché dans la suite de tous les objets