Chapitre 5 :
Menus et listes avec Android

Les menus

Les menus dans le contexte du développement d'applications Android sont des éléments essentiels de l'interface utilisateur permettant aux utilisateurs d'interagir de manière intuitive avec une application. Ils offrent une méthode organisée et accessible pour présenter des options et des actions, facilitant ainsi la navigation et l'utilisation de l'application.

Les menus servent à offrir des fonctionnalités supplémentaires de manière non intrusive. Ils permettent de regrouper des actions contextuelles et des options spécifiques à une vue ou à une activité, offrant ainsi une expérience utilisateur fluide et efficace. En intégrant des menus d'options, les développeurs peuvent rationaliser l'accès aux fonctionnalités clés, réduisant la complexité visuelle de l'interface utilisateur tout en fournissant un accès rapide aux actions importantes.

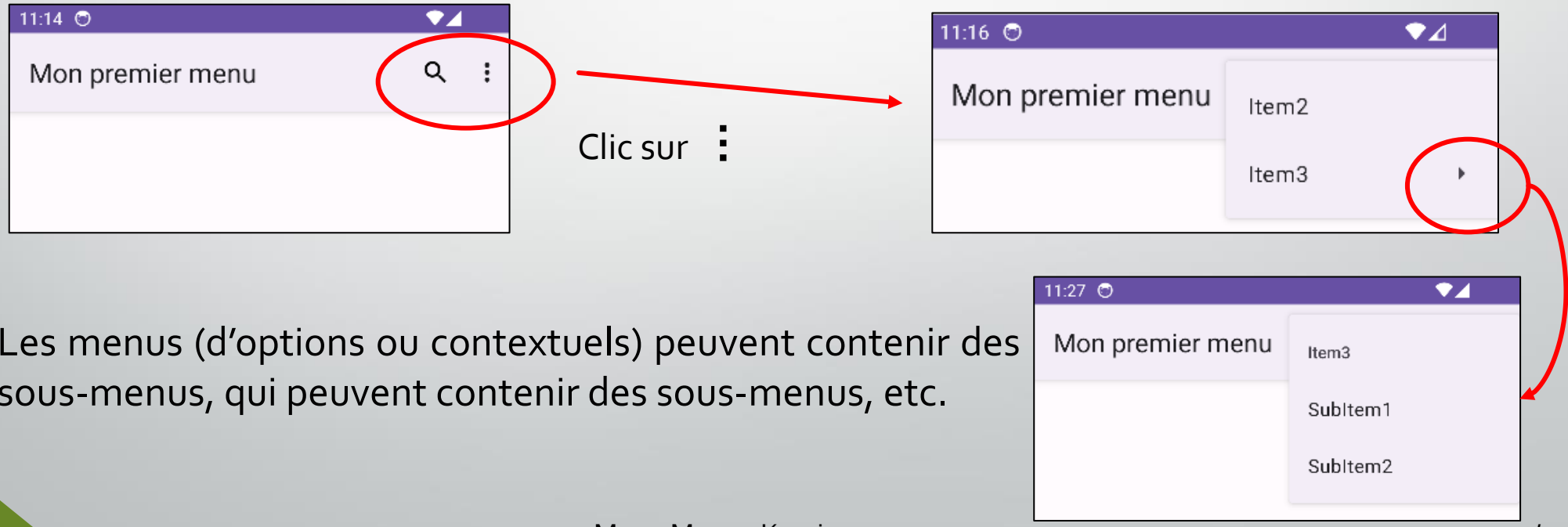
Deux principaux types de menus existent :

- ✓ Les menus d'options situés dans la barre d'action en haut de l'écran
- ✓ Les menus contextuels activés par des actions spécifiques

Les menus d'options (1/5)

Les menus d'options sont des interfaces interactives essentielles permettant aux utilisateurs d'accéder rapidement à des actions spécifiques au sein d'une application. Ils fournissant un accès rapide aux actions fréquemment utilisées, contribuant ainsi à une navigation fluide et à une interaction intuitive avec l'application.

- Le menu d'options est un type de menu qui apparaît dans la barre d'action en haut de l'interface utilisateur de l'application.



Les menus (d'options ou contextuels) peuvent contenir des sous-menus, qui peuvent contenir des sous-menus, etc.

Les menus d'options (2/5)

Pour intégrer un menu dans l'interface graphique d'une application Android, vous aurez généralement besoin de deux fichiers essentiels : le fichier xml de menu et le fichier java de l'activité. L'intégration du menu d'options passe par trois étapes essentielles :

1) Fichier de menu (res/menu/options_menu.xml) :

Ce fichier XML spécifie les détails des éléments du menu, tels que les options disponibles et leurs attributs. C'est là que vous définissez la structure et le contenu du menu.

options_menu.xml

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto" >

  <item
    android:id="@+id/menu_item1"
    android:title = "Option 1"
    app:showAsAction = "never" />

  <item
    android:id = "@+id/menu_item2"
    android:title = "Option 2"
    app:showAsAction = "never" />
</menu>
```

Les menus d'options (3/5)

Les éléments d'un menu s'appellent en XML des <item> et peuvent être personnalisés à l'aide de plusieurs attributs :

- ✓ android:id → Cet attribut spécifie l'ID de l'élément du menu pour identifier quel élément du menu a été sélectionné.
- ✓ android:title → Cet attribut définit le texte qui sera affiché pour l'élément du menu.
- ✓ android:icon → Cet attribut spécifie l'icône qui sera affichée pourl'élément du menu.
- ✓ app:showAsAction → Cet attribut contrôle comment l'élément du menu est affiché dans la barre d'action. Les valeurs possibles incluent **always**, **ifRoom**, et **never**.
- ✓ **android:enabled** → Cet attribut contrôle si l'élément du menu est activé ou désactivé. Vous pouvez le définir sur **true** ou **false**.

Les menus d'options (4/5)

2) Inflation du menu dans l'activité (au niveau du code source de l'activité) :

Dans votre activité, vous devez inflater le fichier XML de menu. Cela signifie créer les objets de menu correspondants à partir du fichier XML.
→ Ceci se fait dans la méthode **onCreateOptionsMenu**.

Cette méthode est lancée au moment de la première pression du bouton qui fait émerger le menu.

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.options_menu, menu);
    return true;
}
```

MainActivity.java

getMenuInflater() renvoie l'objet **MenuInflater**

inflate(R.menu.options_menu, menu) utilise l'objet **MenuInflater** pour intégrer les éléments du menu définis dans le fichier **options_menu.xml** dans la barre d'action.

Les menus d'options (5/5)

3) Gestion des sélections d'éléments du menu (au niveau du code source de l'activité) :

Vous pouvez définir le comportement lorsque l'utilisateur sélectionne un élément du menu.

```
@Override
public boolean onOptionsItemSelected (MenuItem item)
{
   switch (item.getItemId())
   {
      case R.id.menu_item1:
         // ...
         return true;
      case R.id.menu_item2:
         // ...
         return true;
      default:
         return super.onOptionsItemSelected(item);
   }
}
```

MainActivity.java

Utilisée pour identifier l'élément sélectionné et définir les actions correspondantes

Cette méthode est appelée lorsque l'utilisateur sélectionne un élément du menu

Appelée pour exécuter le comportement par défaut de la classe parente.

Les menus contextuels (1/4)

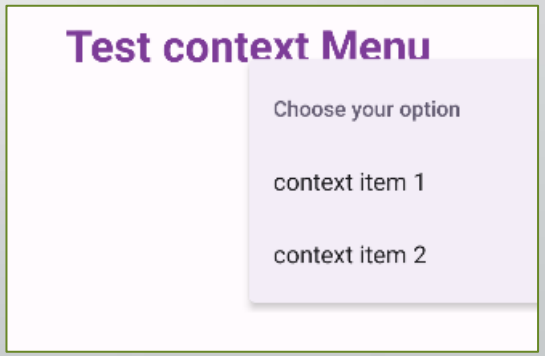
Le menu contextuel est un type de menu qui apparaît en réponse à une action de l'utilisateur, généralement une longue pression sur un élément de l'interface utilisateur.

L'intégration du menu contextuel passe différentes étapes:

1) Création du fichier de menu contextuel (res/menu/context_menu.xml) :

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
   <item
      android:id = "@+id/context_item1"
      android:title = "Context Option 1" />
   <item
      android:id = "@+id/context_item2"
      android:title = "Context Option 2" />
</menu>
```

context_menu.xml



Les menus contextuels (2/4)

2) Déclaration de la view qui possède un menu contextuel (au niveau du code source de la méthode onCreate de l'activité) :

MainActivity.java

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
        setContentView(R.layout.votre_layout);

        // Récupérer l'ID de la vue à laquelle vous souhaitez attacher le menu contextuel
        View maVue = findViewById(R.id.ma_vue);

        // Enregistrer la vue pour le menu contextuel
        registerForContextMenu(maVue);
        // ...
}
```

Exemple

```
TextView mon_text = findViewById(R.id.textid);
registerForContextMenu(mon_text);
```

Les menus contextuels (3/4)

3) Inflation du menu contextuel dans l'activité :

Dès que l'utilisateur fera un clic long sur cette vue, un menu contextuel s'ouvrira. Ce menu se définit dans la méthode suivante :

Pour garantir que toutes les fonctionnalités de base liées à la création du menu contextuel sont correctement initialisées

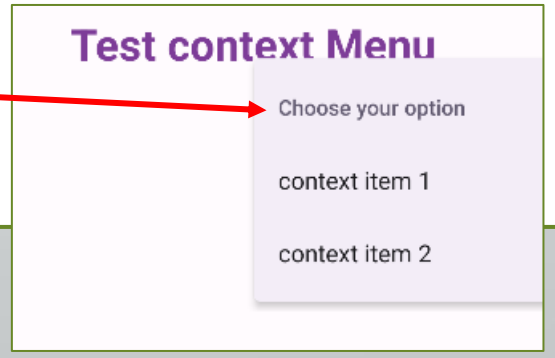
la vue sur laquelle le menu a été appelé

le menu à construire

un moyen de transmettre des informations contextuelles spécifiques à la vue qui a déclenché le menu contextuel.
Exemple : si le menu est attaché à une liste, on peut obtenir des informations sur l'élément de la liste sélectionné

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("Choose your option");
    getMenuInflater().inflate(R.menu.context_menu, menu);
}
```

MainActivity.java



Les menus contextuels (4/4)

4) **Gestion des sélections d'éléments du menu contextuel :**

MainActivity.java

```
@Override
public boolean onContextItemSelected (MenuItem item)
{
   switch (item.getItemId())
   {
      case R.id.menu_item1:
         // ...
         return true;
      case R.id.menu_item2:
         // ...
         return true;
      default:
         return super.onContextItemSelected(item);
   }
}
```

Les menus Popup (1/4)

Les menus pop-ups sont des menus contextuels qui apparaissent de manière temporaire et flottante à l'écran suite à un clic simple sur un composant, offrant une liste d'options ou d'actions à l'utilisateur. Ils sont souvent associés à des éléments particuliers de l'interface utilisateur, comme des boutons, des images ou d'autres vues interactives.

1) Création du fichier de menu popup (res/menu/popup_menu.xml) :

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
   <item
      android:id = "@+id/popup_item1"
      android:title = "popup item 1" />
   <item
      android:id = "@+id/popup_item2"
      android:title = "popup item 2" />
</menu>
```

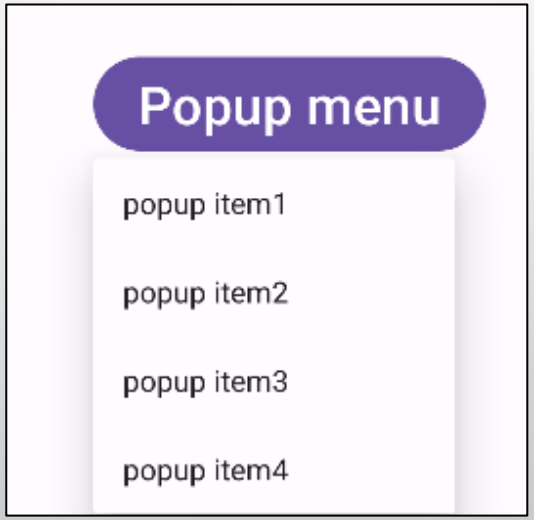
popup_menu.xml

Les menus Popup (2/4)

2) Associer un événement onClick à un bouton :

```
<Button
   android:id="@+id/button"
   android:text="Popup menu"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:onClick = "showPopup" />
```

activity_main.xml



Les menus Popup (3/4)

3) Implémenter la méthode qui se déclenche suite à un clic bouton :

Créer une instance de la classe PopupMenu

L'activité courante

La vue à partir de laquelle le menu pop-up sera affiché

```
public void showPopup (View v)
{
   PopupMenu popup = new PopupMenu(this, v);

   popup.setOnMenuItemClickListener(this);

   popup.inflate(R.menu.popup_menu);

   popup.show();
}
```

MainActivity.java

Permet d'associer un gestionnaire d'événements pour les éléments du menu pop-up.

Afficher le menu pop-up à l'emplacement de la vue spécifiée précédemment (v).

Pour associer un gestionnaire d'événements à un objet PopupMenu, la classe MainActivity doit implémenter l'interface PopupMenu.OnMenuItemClickListener.

```
public class MainActivity extends AppCompatActivity implements PopupMenu.OnMenuItemClickListener
```

Les menus Popup (4/4)

4) Gestion des sélections d'éléments du menu popup :

```
@Override
public boolean onMenuItemClick (MenuItem item)
{
   switch (item.getItemId())
   {
      case R.id.popup_item1:
         // ...
         return true;
      case R.id.popup_item2:
         // ...
         return true;
      default:
         return false;
   }
}
```

MainActivity.java

Les listes

Les listes graphiques, sont des éléments d'interface utilisateur essentiels permettant d'afficher et d'organiser des données de manière structurée. En d'autres termes, ce sont des composants visuels qui facilitent la présentation et la manipulation de listes d'éléments, tels que des images, du texte, ou même des éléments interactifs, sur les écrans des applications Android.

En utilisant des listes, les développeurs peuvent afficher efficacement des ensembles de données complexes de manière claire et concise, améliorant ainsi la navigation et l'interaction pour les utilisateurs.

Les listes graphiques permettent également une gestion efficace des ressources, car elles chargent dynamiquement les éléments visibles à l'écran plutôt que l'intégralité de la liste

- Les listes graphiques sont un outil puissant pour organiser et présenter des informations de manière conviviale, jouant un rôle clé dans la création d'applications Android performantes et agréables à utiliser.

Gestion des listes de données (1/2)

La gestion des listes se divise en deux parties distinctes:

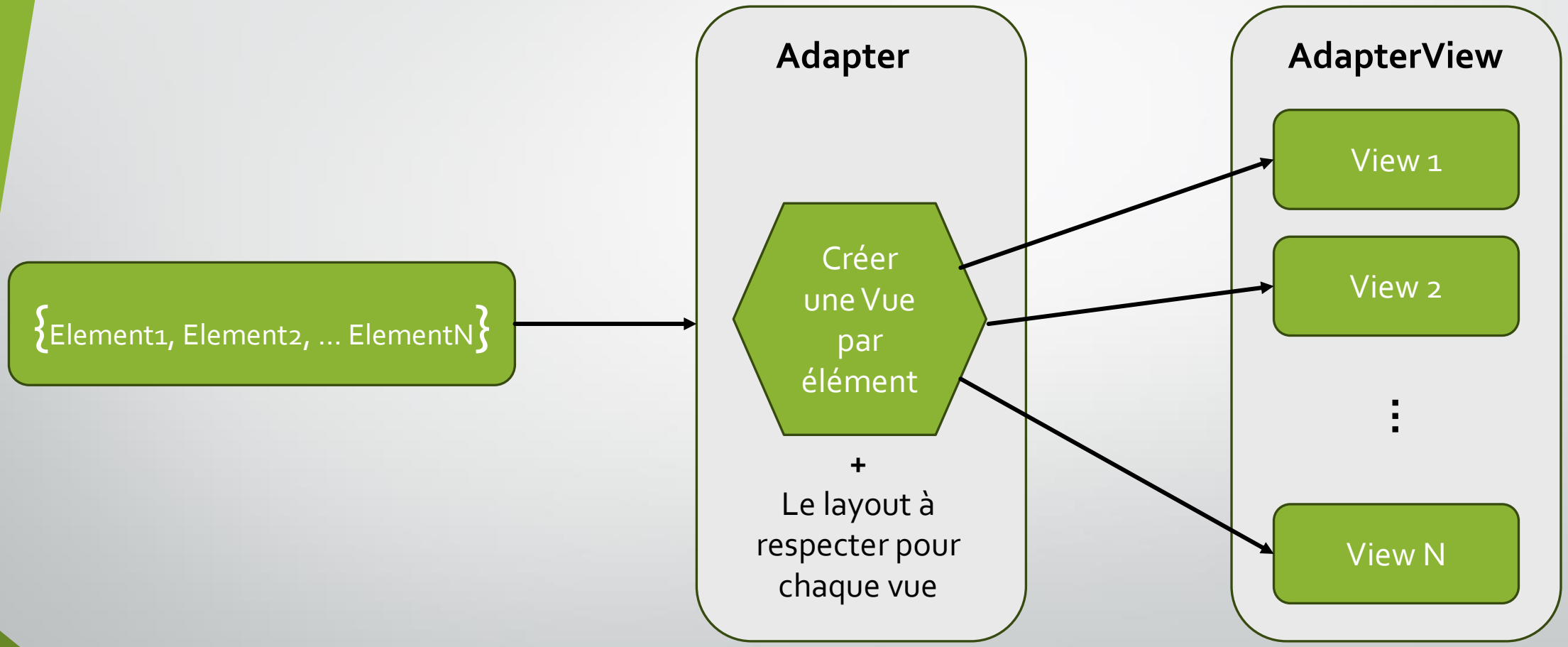
- ✓ Les **Adapter** qui sont les objets qui gèrent les données, mais pas leur affichage ou leur comportement en cas d'interaction avec l'utilisateur. On peut considérer un adaptateur comme une intermédiaire entre les données et la vue qui représente ces données.
- ✓ Les **AdapterView**, qui vont gérer l'affichage et l'interaction avec l'utilisateur, mais sur lesquels on ne peut pas effectuer d'opération de modification des données.

  → C'est un groupe de widgets qui peuvent afficher des données provenant d'un Adapter, tels que : ListView, GridView, RecycleView et Spinner.

Le processus typique pour afficher une liste depuis un ensemble de données est le suivant :

1. On donne à l'adaptateur une liste d'éléments à traiter et la manière dont ils doivent l'être.
2. On passe cet adaptateur à un AdapterView.
3. Dans l'AdapterView , l'adaptateur va créer un widget pour chaque élément en fonction des informations fournies en amont.

Gestion des listes de données (2/2)



Les Adapters (1/2)

un **Adapter** est un composant crucial dans la gestion des listes d'éléments, permettant de lier des données à des vues. Son rôle principal est de convertir un ensemble de données en une représentation visuelle cohérente à afficher dans une interface utilisateur, généralement sous forme de listes.

Exemple :

```
String[ ] restau = new String[ ]{"Pizza", "Olives", "Burger", "Cheese", "Onion"};
ArrayAdapter arrayAdapter = new ArrayAdapter(this, R.layout.single_item, restau);
```

- Ce tableau de chaines de caractères sera passé à un ArrayAdapter.
- L'ArrayAdapter prends "pizza" comme premier élément et le convertit en un view item.
- Le pizza view item est placé comme première ligne (row) dans la ListView.

Les Adapters (2/2)

Si on veut construire un widget simple, on retiendra trois principaux adaptateurs :

- ✓ ArrayAdapter → permet Principalement de lier des données simples (telles que des tableaux ou des listes) à des vues, généralement pour la création de listes simples.
- ✓ SimpleAdapter → utile dès qu'il s'agit d'écrire plusieurs informations pour chaque élément (s'il y a deux textes dans l'élément par exemple).
- ✓ CursorAdapter → permet d'adapter le contenu qui provient d'une base de données. On y reviendra dès qu'on abordera l'accès à une base de données.

ArrayAdapter (listes simples)

La classe ArrayAdapter se trouve dans le package android.widget.ArrayAdapter.

Pour instancier un objet ArrayAdapter, on va considérer les constructeurs suivants :

- **public ArrayAdapter (Context contexte, int id, T[ ] objects)**
- **public ArrayAdapter (Context contexte, int id, List<T> objects)**

✓ Contexte est l'activité dans laquelle l'adaptateur est utilisé.

✓ Id est l'identifiant de la vue de mise en page utilisé pour chaque élément de la liste. Il peut s'agir d'une vue prédéfinie du système (comme android.R.layout.simple_list_item_1) ou d'une mise en page personnalisée définie par l'utilisateur.

```
ArrayList<String> data = new ArrayList<>();
data.add("Élément 1");
data.add("Élément 2");

ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, data);

ListView listView = findViewById(R.id.listView);
listView.setAdapter(adapter);
```

SimpleAdapter (listes plus complexes)

La classe SimpleAdapter se trouve dans le package android.widget.SimpleAdapter.

Pour initialiser un objet de type SimpleAdapter, on considère la syntaxe suivante:

```
SimpleAdapter adapter = new SimpleAdapter(Context context, List<? extends Map<String, ?>> data, int resource, String[ ] from, int[ ] to);
```

- **Context context :** généralement l'activité en cours.
- **List<? extends Map<String, ?>> data :** C'est la liste des données à afficher. Chaque élément de la liste doit être une Map où les clés (de type String) sont les noms des colonnes et les valeurs peuvent être de différents types.
- **int resource :** C'est l'ID de la mise en page personnalisée pour chaque élément de la liste. Cette mise en page doit contenir les vues (par exemple, TextView, ImageView) correspondant aux clés spécifiées dans le tableau from.
- **String[ ] from :** C'est un tableau de chaînes représentant les clés des données dans la Map. Chaque élément de from correspond à une colonne de données dans la Map.
- **int[ ] to :** C'est un tableau d'entiers représentant les ID des vues dans la mise en page personnalisée (resource). Chaque élément de to correspond à une vue dans laquelle les données de la colonne spécifiée par from seront affichées.

SimpleAdapter (listes plus complexes)

Exemple :&

```
List<Map<String, String>> data = new ArrayList<>();

Map<String, String> item1 = new HashMap<>();
item1.put("Title", "Élément 1");
item1.put("Subtitle", "Sous-titre 1");
data.add(item1);

Map<String, String> item2 = new HashMap<>();
item2.put("Title", "Élément 2");
item2.put("Subtitle", "Sous-titre 2");
data.add(item2);

String[ ] from = {"Title", "Subtitle"};
int[ ] to = { R.id.titleTextView, R.id.subtitleTextView };

SimpleAdapter adapter = new SimpleAdapter(this, data, R.layout.custom_list_item, from, to);
ListView listV = findViewById(R.id.myListView);
listV.setAdapter(adapter);
```

Les AdapterView (1/3)

On trouve la classe AdapterView dans le package android.widget.AdapterView

Alors que l'adapter se chargera de construire les sous-éléments, c'est l'AdapterView qui liera ces sous-éléments et qui fera en sorte de les afficher en une liste. De plus, c'est l'AdapterView qui gérera les interactions avec les utilisateurs : l'Adapter s'occupe des éléments en tant que données, alors que l'AdapterView s'occupe de les afficher et veille aux interactions avec un utilisateur (sélection d'un élément par exemple)

Quatre principaux AdapterView sont utilisés :

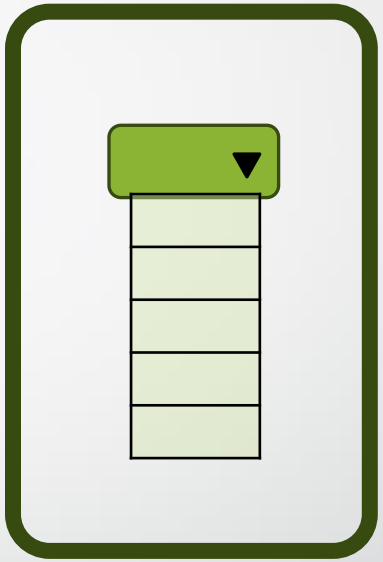
- ✓ ListView → utilisé pour afficher une liste verticale déroulante d'éléments (les uns après les autres).
- ✓ GridView → utilisé pour afficher une grille d'éléments : des images ou des données tabulaires. Il organise les éléments en colonnes et lignes.
- ✓ Spinner → utilisé pour permettre à l'utilisateur de sélectionner une valeur à partir d'une liste déroulante. Il est adapté pour des listes statiques ou des choix simples.
- ✓ RecyclerView → utilisé pour afficher des listes ou des grilles dynamiques. Il est adapté lorsque vous avez une liste longue ou dynamique d'éléments à afficher, avec des performances améliorées et une gestion flexible des mises à jour.

Les AdapterView (2/3)

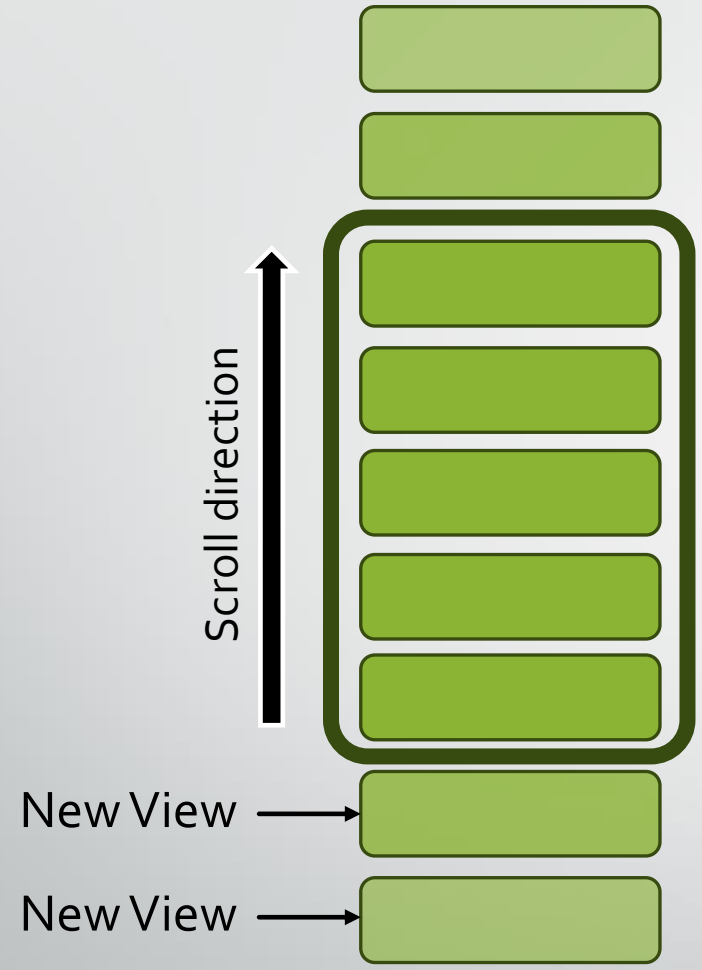
ListView

GridView

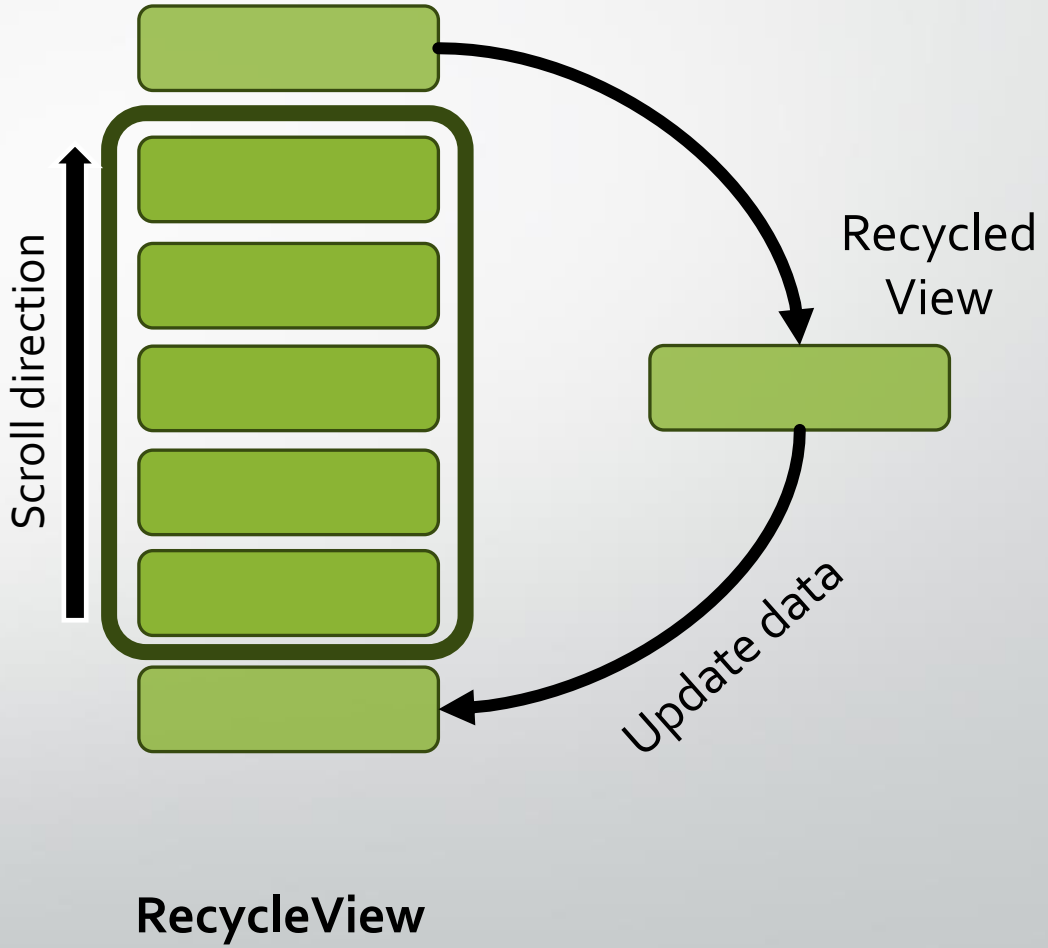
Spinner



Les AdapterView (3/3)

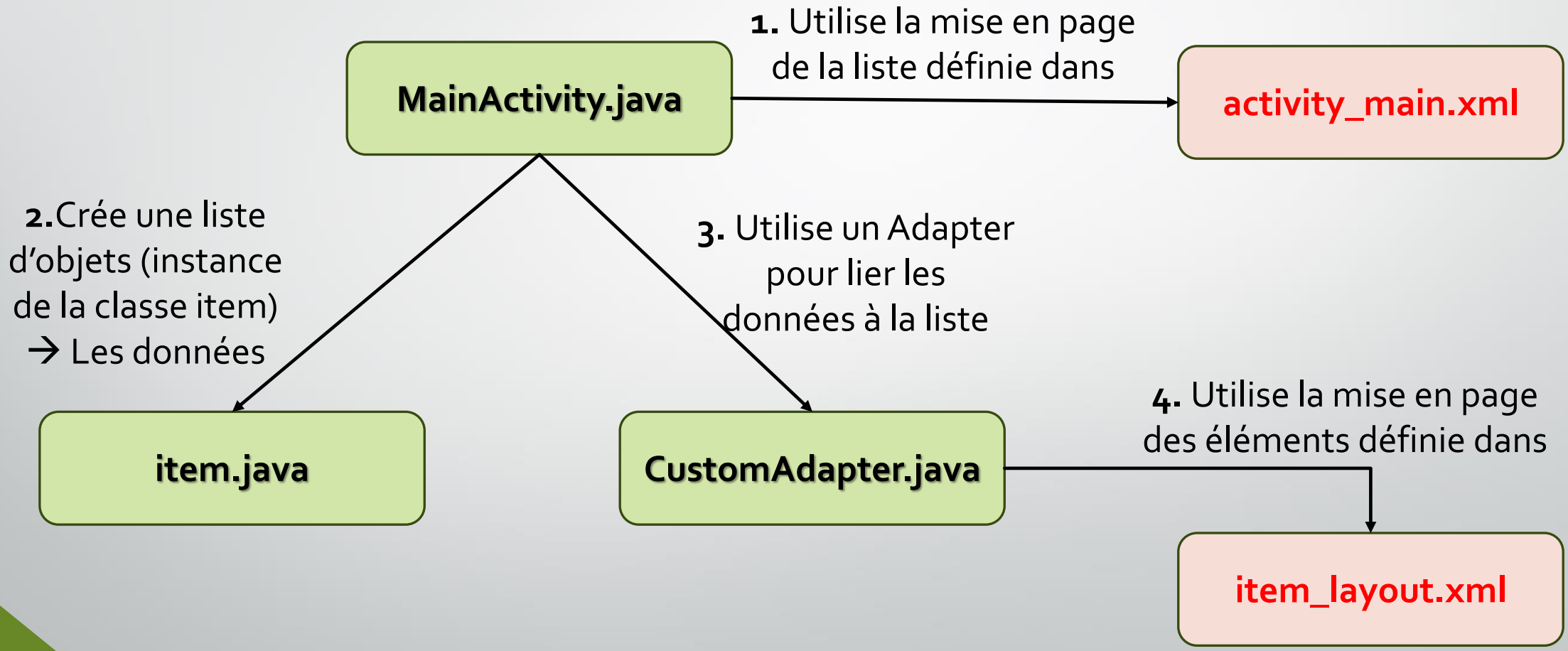


Vs.



ListView (1/2)

Dans le contexte d'une ListView dans une application Android, 5 différents fichiers interagissent:



ListView (2/2)

1.Fichier XML pour la ListView (activity_main.xml) :

Ce fichier définit la structure de l'interface utilisateur de ton activité principale. Il peut contenir la ListView où la liste d'éléments sera affichée.

2.Fichier XML pour chaque élément de la liste (item_layout.xml) :

Ce fichier définit la structure visuelle de chaque élément dans la ListView. Chaque élément de la liste sera affiché en utilisant ce modèle.

3.Classe pour l'objet représentant l'élément (Item.java) :

Cette classe définit la structure de l'objet que chaque élément de la liste représente. Elle contient les propriétés de l'objet (par exemple, le texte à afficher dans la liste).

4.Classe MainActivity (MainActivity.java) :

Cette classe gère l'activité principale de ton application Android. Elle inflates le fichier XML (activity_main.xml), initialise la ListView, crée une liste d'objets (instances de la classe Item), et utilise un adaptateur pour lier ces données à la ListView.

5.Classe de l'adaptateur (CustomAdapter.java, par exemple) :

Cette classe étend un adaptateur standard (comme ArrayAdapter). L'adaptateur est responsable de lier les données (la liste d'objets) à la ListView. Il utilise le fichier XML de l'élément de la liste (item_layout.xml) pour déterminer comment chaque élément doit être affiché.

ListView / GridView Vs. RecycleView

- **Performance :**

  RecyclerView offre généralement de meilleures performances en raison de son architecture plus moderne. Il utilise un **concept de recyclage de vues** (ViewHolder) pour réutiliser les vues hors de l'écran, améliorant ainsi l'efficacité du défilement.

- **Flexibilité :**

  RecyclerView est plus flexible et modulaire. Il permet l'utilisation de différents LayoutManager pour organiser les éléments (liste, grille, tableau).

- **Gestion des mises à jour dynamique :**

  RecyclerView est mieux adapté pour les listes avec des données dynamiques qui peuvent changer fréquemment. Il peut utiliser DiffUtil pour optimiser les mises à jour.

  ListView/GridView sont moins efficace pour les mises à jour dynamiques, nécessitant souvent un appel à **notifyDataSetChanged** qui rafraîchit toute la liste.

RecycleView

Trois éléments essentiels interagissent lors de l'élaboration d'une RecyclerView.

- ViewHolder :

  Il agit comme un conteneur léger qui maintient les références aux vues individuelles à l'intérieur d'un élément de la liste. L'utilisation de ViewHolders permet d'optimiser les performances en réutilisant les vues existantes lors du défilement d'une liste, plutôt que de créer de nouvelles vues à chaque fois.

- LayoutManager (gestionnaire de disposition) :

  Il est responsable de la disposition des éléments dans un RecyclerView. Il détermine comment les vues individuelles sont positionnées et recyclées pendant le défilement de la liste.

- DiffUtil :

  Il est utilisé pour simplifier la mise à jour des données dans un RecyclerView de manière efficace. Elle compare deux listes de données (ancienne et nouvelle) et calcule automatiquement les différences entre elles. Cela permet au RecyclerView de mettre à jour uniquement les éléments modifiés, ajoutés ou supprimés, améliorant ainsi les performances et l'efficacité lors de la gestion des mises à jour de la liste.