BURKINA FASO

***

UNITÉ-PROGRÈS-JUSTICE

MASTER2: RÉSEAUX INFORMATIQUES ET MULTIMÉDIA

PROGRAMMATION MOBILE

Vues à adaptateur et boites de dialogue

COMPAORE MOCTAR

29 September 2022

PLAN

- ✓ Vues à adaptateur (AdapterView)
- ✓ Listeners d’un AdapterView
- ✓ Vues à adaptateur personnalisé
- ✓ Boites de dialogue (Dialog)
- ✓ AlertDialog
- ✓ ProgressDialog
- ✓ Dialog personnalisés

VUES À ADAPTATEUR (ADAPTERVIEW)

Une vue à adaptateur (AdapterView) est une vue complexe (sous-classe de ViewGroup) qui contient plusieurs vues, utilisée souvent pour afficher des collections de données (List, Set, Map, …).

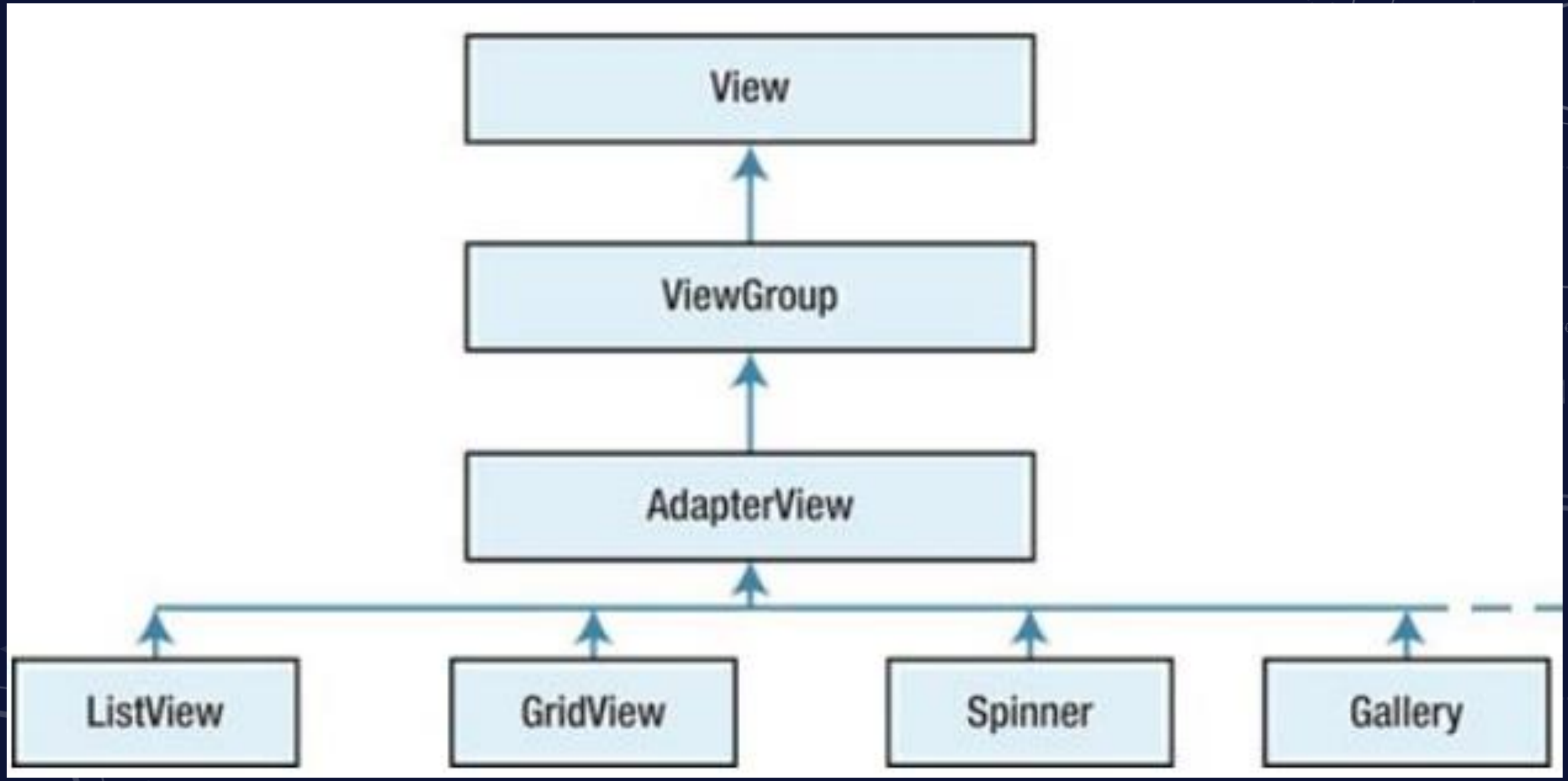
Les vues filles sont déterminées par un adaptateur (Adapter) qui relie la vue à adaptateur aux données.

VUES À ADAPTATEUR (ADAPTERVIEW)

Les vues à adaptateur les plus connues sont :

- ✓ ListView : affiche une liste d'éléments avec un défilement vertical,
- ✓ GridView : affiche des données sur une grille avec défilement vertical où les attributs columnWidth et numColumns permettent de configurer l'alignement des cellules,
- ✓ Spinner : est une liste déroulante de données à choix unique,
- ✓ Gallery : affiche une liste éléments avec un défilement horizontal qui verrouille l'élément sélectionné au centre ,
- ✓ AutoCompleteTextView : permet d'obtenir des suggestions, lorsque on écrit du texte,
- ✓ RecyclerView : permet d'afficher un grand nombre de données en améliorant les performances

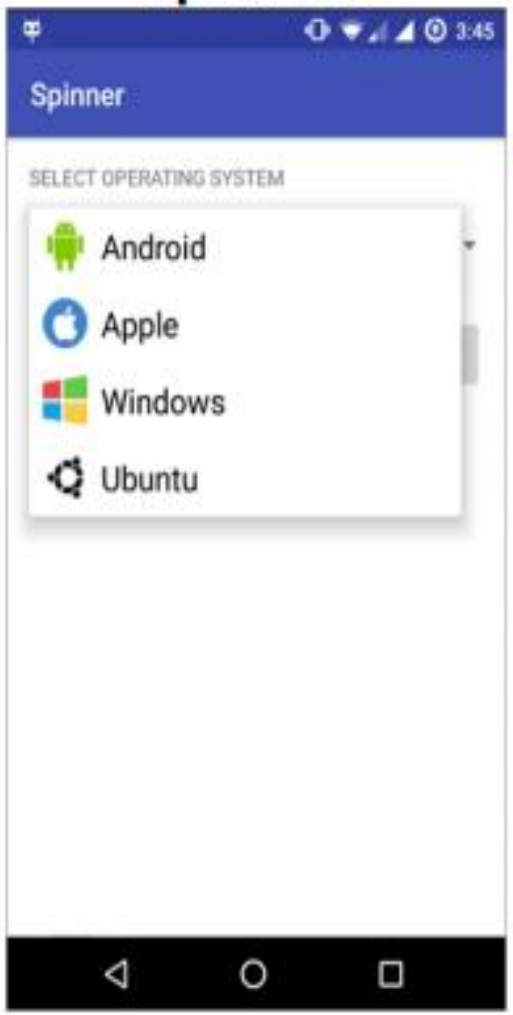
VUES À ADAPTATEUR (ADAPTERVIEW)



VUES À ADAPTATEUR (ADAPTERVIEW)

ListView

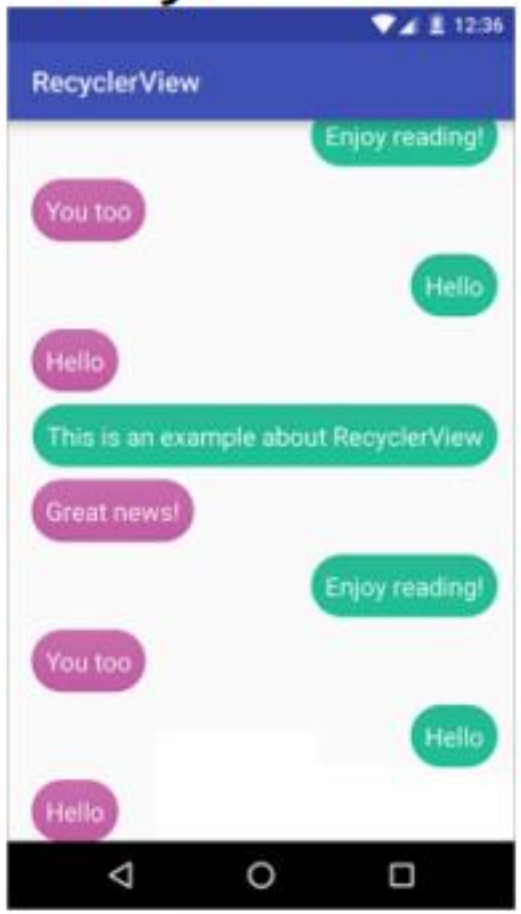
Spinner



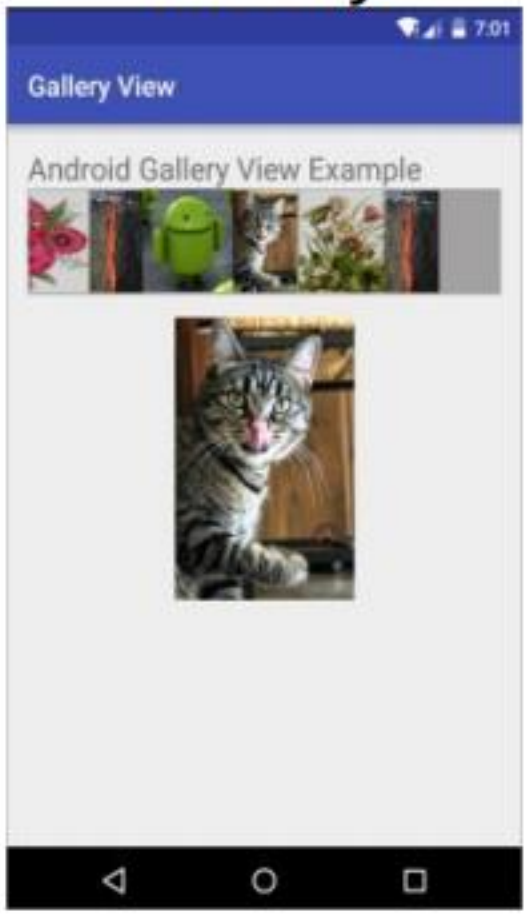
GridView

VUES À ADAPTATEUR (ADAPTERVIEW)

RecyclerView



Gallery



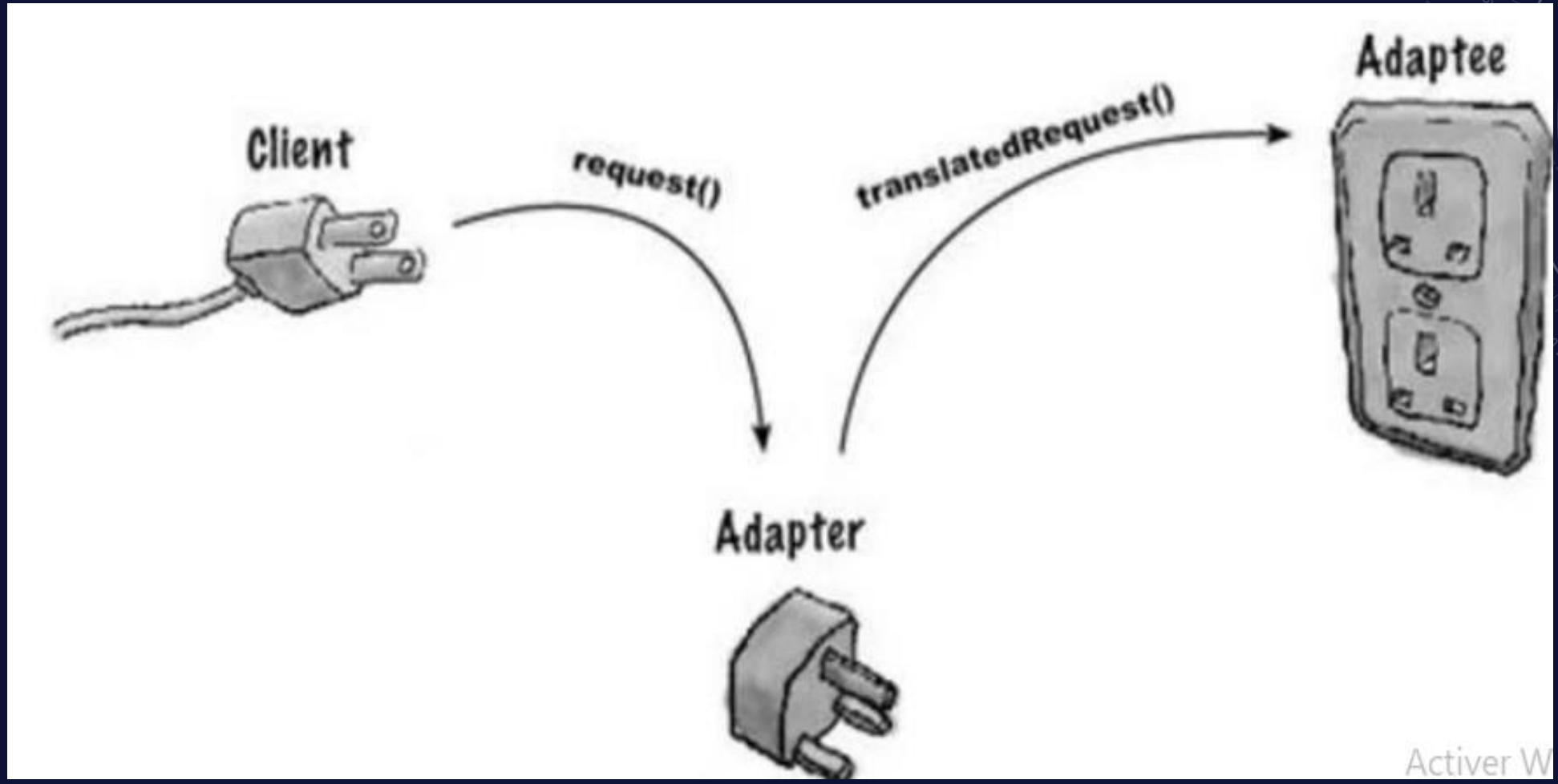
AutoCompleteTextVie

DESIGN PATTERN : ADAPTER

La vue qui permet d'afficher une liste se basent sur le design pattern Adapter pour remplir la vue. Ce pattern est utilisé dans toute l'architecture Android pour remplir des listes, de ce fait, il est indispensable d'en connaître son fonctionnement.

Le client créer une demande en appelant une méthode request() d'un adaptateur. Ce dernier traduit cette demande en un ou plusieurs appels sur l'adaptée en utilisant une méthode translateRequest(). Le client reçoit les résultats de l'appel et ne sait jamais qu'il y a un adaptateur qui effectue la traduction.

DESIGN PATTERN : ADAPTER



Donc dans notre contexte, un adaptateur permet de transformer une collection de données (un tableau, une liste ou un curseur de données) en widgets pour les insérer dans une vue à adapter (android.widget.AdapterView).

DESIGN PATTERN : ADAPTER

On propose de créer une liste déroulante semblable à la figure suivante. Pour ce faire, il faut rajouter un widget de type Spinner dans le layout. Ensuite, on remplit le spinner avec les données en utilisant un adaptateur (android.widget.Adapter)


```
<Spinner
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/spinner"  />
```

DESIGN PATTERN : ADAPTER

- L’adaptateur le plus utilisé sous Android est ArrayAdapter<T> qui est dédié aux tableaux.
- Par défaut, Android fournit un layout prédéfini qui contient un TextView (android.R.layout.simple_list_item_1), permettant à l’adaptateur de crée une vue pour chaque élément du tableau en appelant la méthode toString() de chaque élément, et en plaçant son contenu dans un objet TextView.
- Dans le code suivant, un ArrayAdapter<String> est créé à l’aide de son constructeur qui comprend 3 paramètres : le contexte de l’application (généralement l’activité dans laquelle est défini l’AdapterView, le layout de chaque élément et le tableau contenant les données.
- Enfin, l’adaptateur créé est associé au Spinner, en utilisant la méthode setAdapter(...).

DESIGN PATTERN : ADAPTER

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);

String[] items = {"-Choisir-", "Mercury", "Venus", "Earth","Mars","Jupiter" };
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
            android.R.layout.simple_list_item_1,
              items);
spinner.setAdapter(adapter);
```

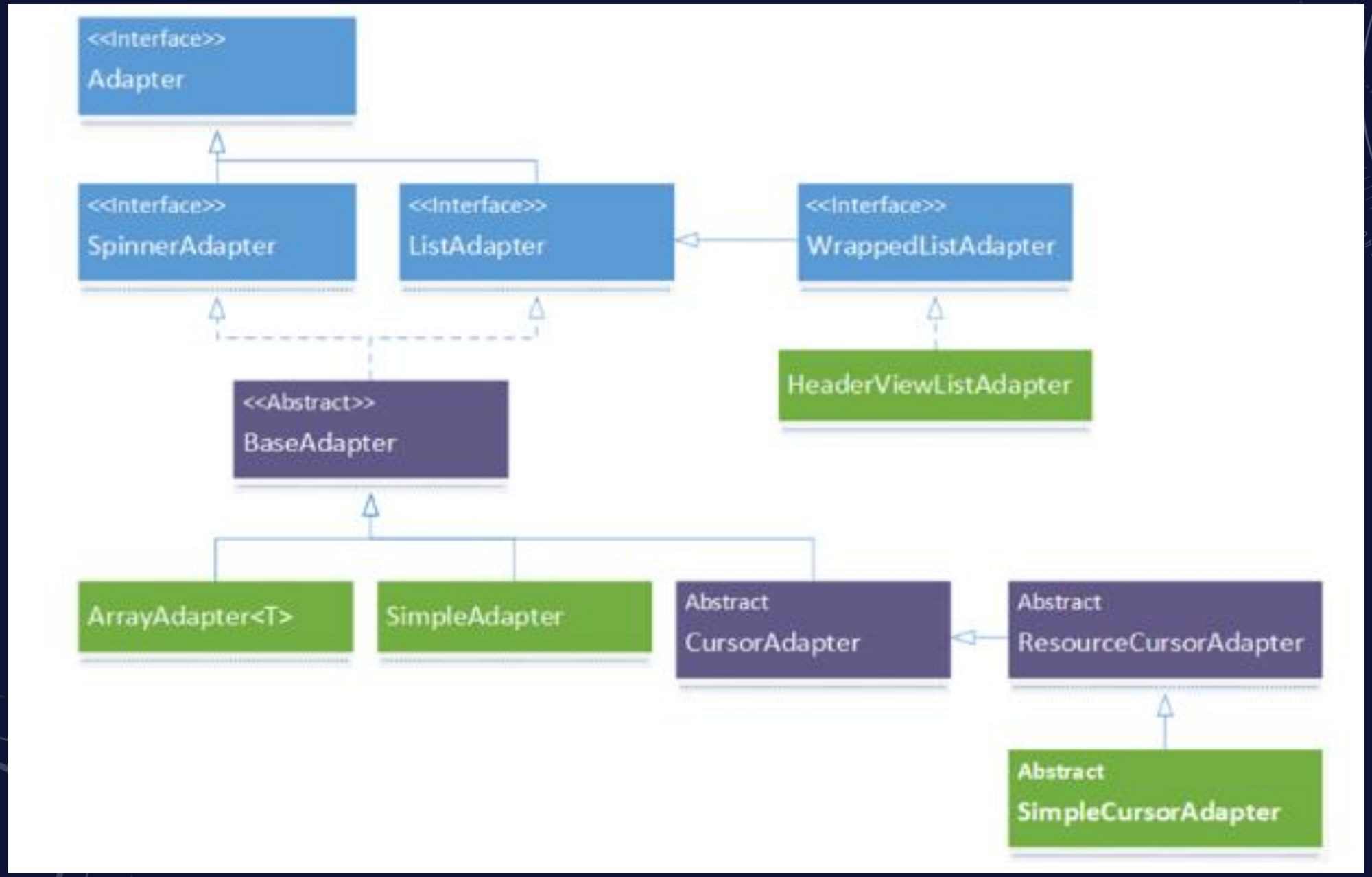
Pratiques

Hiérarchie des adaptateurs

Android fournit différents adaptateurs qui permettent de représenter les informations de façon standard dans une vue à adaptateur :

- ✓ ArrayAdapter<T> : pour tous les types de tableaux et de listes,
- ✓ BaseAdapter : sert à implémenter des adaptateurs personnalisés,
- ✓ CursorAdapter : pour traiter les données de type Cursor,
- ✓ HeaderViewListAdapter : permet d'ajouter des entêtes et pieds de page aux ListView,
- ✓ ResourceCursorAdapter : sert à créer des vues à partir d'une disposition XML,
- ✓ SimpleAdapter : pour afficher des données complexes (par exemple un tableau de tableaux),
- ✓ SimpleCursorAdapter : pour adapter les données d'un Cursor de Base de données.

Hiérarchie des adaptateurs



Hiérarchie des adaptateurs

Quelques méthodes communes à tous les adaptateurs permettant de mettre à jour les données.

```
void add(T item) // ajouter un élément en fin de l'AdapterView
void insert(T item, int index) // insérer un élément à une position donnée
void addAll(T... items) // insérer plusieurs éléments
T getItem(int index) // récupérer l'élément d'une position donnée
int getPosition(Object o) // récupérer la position d'un élément donné
void remove(T item) // supprimer un élément donnée
void clear() // supprimer tous les éléments
void notifyDataSetChanged() // notifie l'AdapterView des nouveaux
changements pour se ra
```

Listeners d’un AdapterView

Un AdapterView est conçu pour afficher une liste d'éléments à l'utilisateur et les actions que peut effectuer ce dernier est le clic ou la sélection d’un élément de l’AdapterView. Afin d’interagir avec un AdapterView, il suffit d’intercepter l’évenement déclenché (par exemple, un clic sur un élément), à l’aide de listeners (écouteurs). Voici les listeners proposés pour un AdapterView :

- ✓ OnItemClickListener : pour intercepter l’évènement du clic sur un élément. La méthode à surcharger est :
  - ✓ onItemClick(AdapterView<?> parent, View view, int position, long id)

Listeners d’un AdapterView

- ✓ OnItemClickListener : pour intercepter l’évènement du clic sur un élément. La méthode à surcharger est : onItemClick(AdapterView<?> parent, View view, int position, long id)
- ✓ OnItemLongClickListener : pour intercepter l’évènement du clic long sur un élément. La méthode à surcharger est : onLongClick(AdapterView<?> parent, View view, int position, long id)
- ✓ OnItemSelectedListener : pour intercepter l’évènement de la sélection d’éléments. Les méthodes à surcharger sont : onItemSelected(AdapterView<?> parent, View view, int position, long id) onNothingSelected(AdapterView<?> parent)

Listeners d’un AdapterView

Pour ajouter un listener XXX à un AdapterView, il suffit d’utiliser la méthode setXXX(...).
Par exemple, pour ajouter un comportement au clic sur un élément de l’AdpaterView, il faut définir un OnItemClickListener et surcharger la méthode onItemClick(...).
L’élément cliqué et sa position sont capturés par les paramètres view et position

Listeners d’un AdapterView

```
spinner.setOnItemClickListener(new AdapterView.OnItemClickListener(){

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long
id){
        String planetteClique= items[position];
        Toast t= Toast.makeText(getApplicationContext(), "La planette cliqué est
".concat(planetteClique), Toast.LENGTH_LONG);
         t.show();
    }
});
```

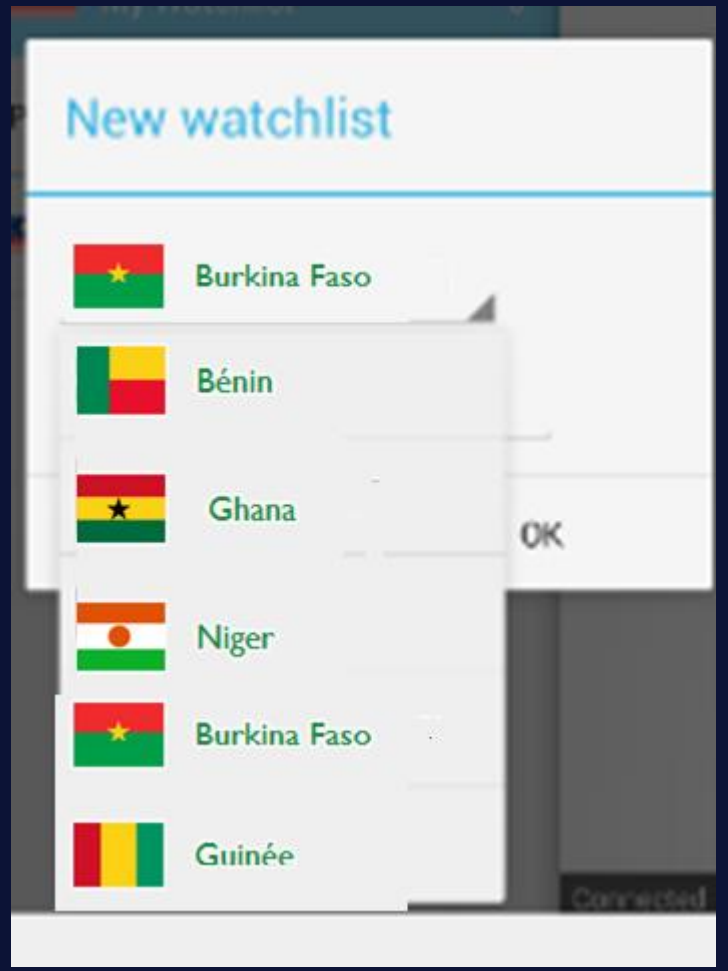
Pratiques

Vues à adaptateur personnalisé

- ✓ Dans certaines situations, les adaptateurs fournis par Android ne permettent pas de résoudre un cas particulier.
- ✓ Alors, il est possible de représenter autre chose qu'un `TextView` dans un élément d'un `AdapterView`.
- ✓ Pour ce faire, il suffit de créer un adaptateur, dit personnalisé, soit en le dérivant d'une classe concrète existante, par exemple `ArrayAdapter<T>`, ou soit en dérivant d'une classe abstraite, par exemple `BaseAdapter`, et ainsi redéfinir la méthode getView(...) qui permet de retourner à l'`AdapterView` chaque élément à afficher.

Vues à adaptateur personnalisé

- ✓ Dans l'exemple suivant, nous allons élaborer un Spinner à adaptateur personnalisé affichant une liste de pays avec leurs drapeaux. Voici les étapes à suivre :



Vues à adaptateur personnalisé

- ✓ Création de l'entité

  Afin de stocker et de manipuler les informations d'un pays, on crée la classe Country contenant le nom du pays (name), l'identifiant concernant l'image de son drapeau (flagResourceId) et un constructeur pour instancier cette classe

```
public class Country {
    private  int flagResourceId;
    private  String name;

    public Country(int flagResourceId, String name){
        this.flagResourceId = flagResourceId;
        this.name = name;
    }
}
```

Vues à adaptateur personnalisé

- ✓ Elaboration d’une vue pour chaque élément

  Il est indispensable d’élaborer la vue de chaque élément du Spinner pour organiser les informations de chaque pays à afficher. Pour ce faire, on crée un layout appelé item_country, qui comporte un ImageView et un TextView pour afficher respectivement le drapeau et le nom du pays.

Vues à adaptateur personnalisé

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ImageView
        android:id="@+id/flagIV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:id="@+id/nameTV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Vues à adaptateur personnalisé

- ✓ Création d’un adaptateur personnalisé

  Dans notre exemple, on choisit de créer un adaptateur personnalisé, dérivant de la classe ArrayAdapter<T> , ce qui impose l’implémentation d’un costructeur et la redéfinition de la méthode getView(int position, View convertView, ViewGroup parent).

Vues à adaptateur personnalisé

```
public class CountryAdapter extends ArrayAdapter<Country> {

    Activity activity;
    int itemResourceId;
    List<Country> items;

    public CountryAdapter(Activity activity, int itemResourceId, List<Country> items){
        super(activity, itemResourceId, items);
        this.activity = activity;
        this.itemResourceId = itemResourceId;
        this.items = items;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View layout = convertView;
        if(convertView == null){
            LayoutInflater inflater = activity.getLayoutInflater();
            layout = inflater.inflate(itemResourceId, parent, false);
        }
        TextView nameTV = (TextView) layout.findViewById(R.id.nameTV);
        ImageView flagIV = (ImageView) layout.findViewById(R.id.flagIV);

        nameTV.setText(items.get(position).name);
        flagIV.setImageResource(items.get(position).flagResourceId);

        return layout;
    }
}
```

LIENS UTILES

Les étudiants peuvent consulter ces références pour approfondir leurs connaissances :

- ✓ AdapterView :
  - ❖ https://developer.android.com/reference/android/widget/AdapterView.html
- ✓ ListView à adaptateurs personnalisés :
  - ❖ http://www.journaldev.com/10416/android-listview-withcustom-adapter-example-tutorial
- ✓ ProgressDialog avancé :
  - ❖ http://www.oodlestechnologies.com/blogs/Custom-Progressbar-and-ProgressDialog
- ✓ Boite de dialogue personnalisée :
  - ❖ http://www.codexpedia.com/android/android-custom-dialog-example/
- ✓ Débogage sous Android studio :
  - ❖ Https://www.learnhowtoprogram.com/android/user-interface-basics-637d41b1-35dc400a-bcc3-65794760474d/debugging-breakpoints-and-the-android-debugg