

# Android: Persistance et partage

Frédéric Lemoine

Conservatoire National des Arts et Métiers

# Introduction

Une application peut être détruite par l'utilisateur:

- Bouton Back
- Appel de la méthode `finish()`

L'état de l'activité au premier plan est perdu

Ou par le système:

- Application inactive pendant un temps trop long
- Nécessité de récupérer de la mémoire
- Changement de configuration (orientation de l'écran par exemple)

L'instance de l'activité peut être déchargée de la mémoire par le système à tout moment

L'état de l'activité est automatiquement sauvegardé dans un `Bundle`

# Persistance de l'état d'une activité

Certaines méthodes du cycle de vie sont chargées de sauvegarder/restituer l'état de l'activité:

```
protected void onCreate(Bundle savedInstanceState)
```

restitue l'état de l'activité à son retour au premier plan

Par défaut, l'état de l'écran est sauvegardé en mémoire dans un objet de type `Bundle`.

Les états de chacune des vues possédant un id sont enregistrés dans le `Bundle`.

# Persistance de l'état d'une activité

Pour sauvegarder des données complémentaires en cas de destruction système, il faut redéfinir:

```
onSaveInstanceState(Bundle bundle)  
onRestoreInstanceState(Bundle bundle)
```

`onSaveInstanceState(Bundle bundle)` n'appartient pas au cycle de vie et n'est donc pas systématiquement appelée. Elle ne sera utilisée qu'en cas de destruction système.

Après `onPause()`, l'activité est susceptible d'être détruite. Pour sauvegarder des données dans toutes les situations (destruction utilisateur ou système), il faut utiliser la méthode `onPause()` qui est appelée automatiquement avant destruction.

# Persistance de l'état d'une activité

## Mécanisme de sauvegarde



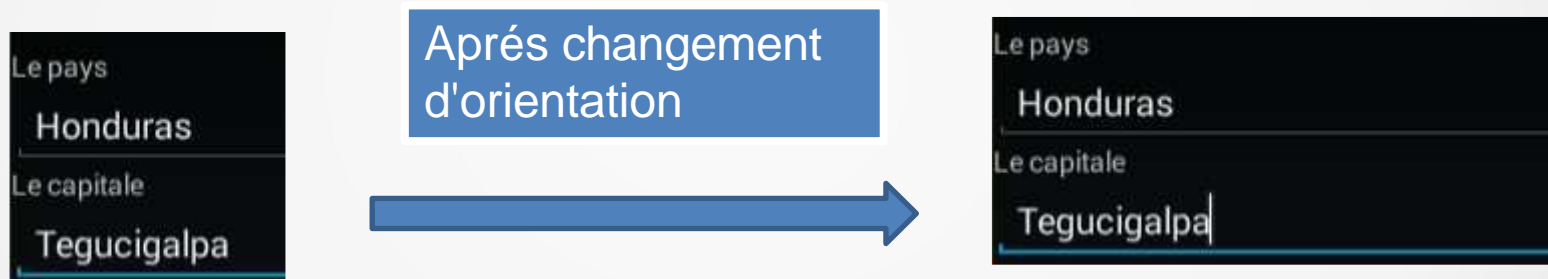
1. Avant la destruction de l'activité, le système appelle `onSaveInstanceState(Bundle)`. On peut enregistrer des données complémentaires qui seront restituées à la recréation de l'instance
2. Lorsque l'activité est recréée, le système passe l'état sauvegardé (`Bundle`) aux méthodes `onCreate(Bundle)`
3. `onRestoreInstanceState(Bundle)` appelée après `onStart()`

# Persistance de l'état d'une activité

## Mécanisme de sauvegarde

Le même mécanisme doit être utilisé pour garantir la persistance des données d'une activité en cas de :

- Changement d'orientation
- Changement de langue



Ce mécanisme ne fonctionne pas en cas de destruction de l'activité par  
C'est à dire destruction utilisateur.



# Persistance de l'état d'une activité

## Exemple: 1/6



Votre réponse

Le pays	Honduras
La capitale	<input type="text" value="Dijon"/>

Valider



L'activité est détruite par l'utilisateur, déchargée de la mémoire. L'état n'est pas sauvegardé

Votre réponse

Le pays	Honduras
La capitale	<input type="text"/>

Valider

Recréation de l'écran précédent sans restitution de l'état de chacune des vues

# Persistence de l'état d'une activité

## Exemple: 2/6

### Extrait du layout de la seconde activité

```
<TableRow android:paddingTop="100dp">
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/tvpays"
  android:paddingLeft="20dp" />
```

```
<EditText
  android:id="@+id/editPays"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/pays"
  android:textSize="14sp"
  android:inputType="text"
  android:ems="12" >
```

```
</EditText>
</TableRow>
```

```
<TableRow android:paddingTop="10dp">
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:paddingLeft="20dp"
  android:text="@string/tvcapitale"
  />
```

```
<EditText
  android:id="@+id/editCapitale"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/capitale"
  android:inputType="text"
  android:textSize="14sp"
  android:ems="12">
```

```
</EditText>
</TableRow>
```

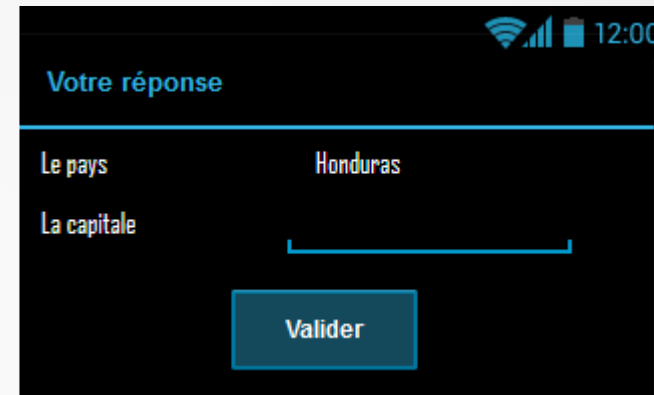


# Persistance de l'état d'une activité

## Exemple: 3/6

Extrait du code de l'activité secondaire

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
```



```
public class Second_Activity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
    }
}
```

# Persistence de l'état d'une activité

## Exemple: 4/6

Extrait du manifest de l'application

```
<application
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme">
  <activity
    android:name=".Second_Activity "
    android:label="@string/title_activity">
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <action android:name="action.CAPITALE"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </activity>
</application>
```

# Persistence de l'état d'une activité

## Exemple: 5/6

Extrait de l'activité principale de la seconde application

```
@Override
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.second);
}

public void onClick(View v){
    Intent intent = new Intent();
    intent.setAction("action.CAPITALE");
    startActivity(intent);
}
```



# Persistence de l'état d'une activité

## Exemple: 6/6

Extrait du layout: res/layout/main.xml

```
<TextView  
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:layout_centerVertical="true"  
    android:textSize="18sp"  
    android:text="@string/question"/>
```

```
<Button  
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/textView1"  
    android:layout_centerHorizontal="true"  
    android:onClick="onClick"  
    android:layout_marginTop="20dp"  
    android:text="@string/valider"/>
```

# Android: Persistance et partage

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putString("message", "Message to be reloaded");
}
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if(savedInstanceState != null )
        String mess=savedInstanceState.getString("message");
}
```

// Ou, appelée après le onStart() si redéfinie:

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    if(savedInstanceState != null )
        String mess=savedInstanceState.getString("message");
}
```

# Android: Persistance et partage

Les préférences partagées:

`SharedPreferences`

# Les préférences partagées

## Introduction

### Intérêt et mécanisme associé

#### Pourquoi

Pour enregistrer et récupérer des données propres à l'application :

- Paramètres utilisateur
- Configuration de l'application

#### Comment

- Sous forme de clé/valeur
- De type primitifs : boolean, float, int, long et de String

#### Conséquence

Les données persistent à travers les sessions utilisateur même si l'application est détruite

# Les préférences partagées

## Désignation d'un ensemble de préférences partagées

Il est possible de créer plusieurs ensembles de préférences partagées identifiés par un nom unique.

La gestion des préférences est réalisée à partir d'une instance de `SharedPreferences` obtenue par:

```
SharedPreferences getSharedPreferences(String name, int mode)
```

ou

Nom d'un ensemble  
de préférences



MODE\_PRIVATE  
MODE\_WORLD\_READABLE  
MODE\_WORLD\_WRITEABLE

```
SharedPreferences getPreferences(int mode)
```

Note: Seul le mode `MODE_PRIVATE` est autorisé aujourd'hui (API  $\geq 17$ ).



# Les préférences partagées

## Exemple (1/8)

15:28

**SENSORS**

Nom:

Cours:

Date:

SAUVE

RESTAURE

EFFACE

FERMER

15:27

**SENSORS**

Nom:  
Lemoine Frédéric

Cours:  
SMB116 : Capteurs et cartes satellite

Date:  
30 juin à 18h15

SAUVE

RESTAURE

EFFACE

FERMER

# Les préférences partagées

## Exemple (2/8)

15:28

**SENSORS**

Nom:

Cours:

Date:

SAUVE

RESTAURE

EFFACE

FERMER

15:27

**SENSORS**

Nom:  
Lemoine Frédéric

Cours:  
SMB116 : Capteurs et cartes satellite

Date:  
30 juin à 18h15

SAUVE

RESTAURE

EFFACE

FERMER

# Les préférences partagées

## Exemple (3/8)

15:28

**SENSORS**

Nom:

Cours:

Date:

SAUVE

RESTAURE

EFFACE

FERMER

15:27

**SENSORS**

Nom: Lemoine Frédéric

Cours: SMB116 : Capteurs et cartes satellite

Date: 30 juin à 18h15

SAUVE

RESTAURE

EFFACE

FERMER

finish() ou back



Persistence et partage

# Les préférences partagées

## Exemple (4/8)

Pour enregistrer une préférence partagée, il faut :

- créer un objet `Editor`
- utiliser les méthodes `putInt()`, `putString()`, ... pour passer les clé/valeur
- appeler la méthode `commit()` sur l'éditeur

Code à ajouter par exemple dans `onPause()`

Accès privé réservé aux  
appels possédant le même  
id/même signature

```
SharedPreferences pref=getPreferences(Context.MODE_PRIVATE);  
Ou  
SharedPreferences  
pref=getSharedPreferences("mes_preferences",Context.MODE_PRIVATE);  
  
Editor spEditor=pref.edit();  
EditText et1=(EditText) findViewById(R.id.ET1);  
spEditor.putString("nom", et1.getText().toString());  
spEditor.commit();
```

# Les préférences partagées

## Exemple (5/8)

MODE\_PRIVATE: accès privé réservé aux applis possédant le même id/même signature

MODE\_WORLD\_READABLE: accessible en lecture aux autres applications

MODE\_WORLD\_WRITEABLE: accessible en lecture et écriture par les autres applications

Pour récupérer les données;

```
SharedPreferences pref=getPreferences(Context.MODE_PRIVATE);  
  
ou  
  
SharedPreferences  
pref=getSharedPreferences("mes_preferences",Context.MODE_PRIVATE);  
  
EditText et1=(EditText) findViewById(R.id.ET1);  
String nom=pref.getString("nom","valeur par défaut");  
et1.setText(nom);
```

# Les préférences partagées

## Exemple (6/8)

```
public class Sensors_shared_preferences extends Activity {
    private EditText et1;
    private EditText et2;
    private EditText et3;
    SharedPreferences pref;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.sensors_shared_preferences);
        et1=(EditText) findViewById(R.id.ET1);
        et2=(EditText) findViewById(R.id.ET2);
        et3=(EditText) findViewById(R.id.ET3);
        pref=getSharedPreferences("mes_preferences",Context.MODE_PRIVATE);
        //pref=getPreferences(Context.MODE_PRIVATE);
    }
}
```

# Les préférences partagées

## Exemple (7/8)

```
public void onSave(View v) {
    Editor  spEditor=pref.edit();
    spEditor.putString("nom", et1.getText().toString());
    spEditor.putString("cours", et2.getText().toString());
    spEditor.putString("date", et3.getText().toString());
    spEditor.commit();
}
public void onSpRestore(View v) {
    et1.setText(pref.getString("nom","valeur par default"));
    et2.setText(pref.getString("cours","valeur par default"));
    et3.setText(pref.getString("date","valeur par default"));
}
public void onSpClean(View v) {
    et1.setText("");
    et2.setText("");
    et3.setText("");
}
public void onClose(View v) {
    finish();
}
}
```

# Les préférences partagées

## Exemple (8/8)

Dans /data/data/[NOM DE PACKAGE]/shared\_prefs :

The screenshot displays the Android Studio interface. On the left, the 'mes\_preferences.xml' file is open, showing the following XML code:

```
1 <?xml version='1.0' encoding='utf-8' standalone='yes' ?>
2 <map>
3   <string name="date">30 juin à 18h15</string>
4   <string name="nom">Lemoine Frédéric</string>
5   <string name="cours">SMB116 : Capteurs et cartes satellite</string>
6 </map>
7 |
```

On the right, the 'Device File Explorer' panel shows the file system of a Samsung SM-T720 Android 11, API 30. The file tree is expanded to show the 'shared\_prefs' directory, which contains several XML files, including 'mes\_preferences.xml'.



# Android: Persistance et partage

## Les menus de préférences

# Les menus de préférences

## Définition

A l'image du menu «settings» d'Android, possibilité de créer des menus de paramétrage spécifique à votre application

Une préférence est un objet qui:

- représente un simple paramètre qui apparait comme un élément dans une liste
- fournit l'UI appropriée pour modifier le paramètre

Exemples de sous-classes de `Preference`: `CheckBoxPreference`, `ListPreference`, ...

Les préférences sont enregistrées par le système en utilisant le mécanisme des `SharedPreferences`.

Les activités réalisant l'affichage des listes de préférences sont des sous-classes de `PreferenceActivity` ou `PreferenceFragment` (>API 10)

# Les menus de préférences

## Description XML

Dans `/res/xml/root_preferences.xml`

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

    <SwitchPreferenceCompat
        app:key="notifications"
        app:title="@string/enable_notifications"/>

    <CheckBoxPreference
        android:key="choix"
        android:title="@string/case_a_cocher"
        android:summary="@string/cliquer_pour_activer"
        android:summaryOn="@string/case_activee"
        android:summaryOff="@string/case_desactivee"
        android:defaultValue="true"
    />

</PreferenceScreen>
```



La clé de préférence

# Les menus de préférences

## Activité de préférence

La clé des préférences est obligatoire. Elle sert d'identifiant pour l'enregistrement en tant que `SharedPreferences`.

La valeur est présentée/modifiée à travers l'interface du widget.

La visualisation d'un menu de préférences est réalisée via une activité du type : `PreferenceFragment`.

Enregistrement automatique des paramètres de l'application.

# Les menus de préférences

## Activité de préférence

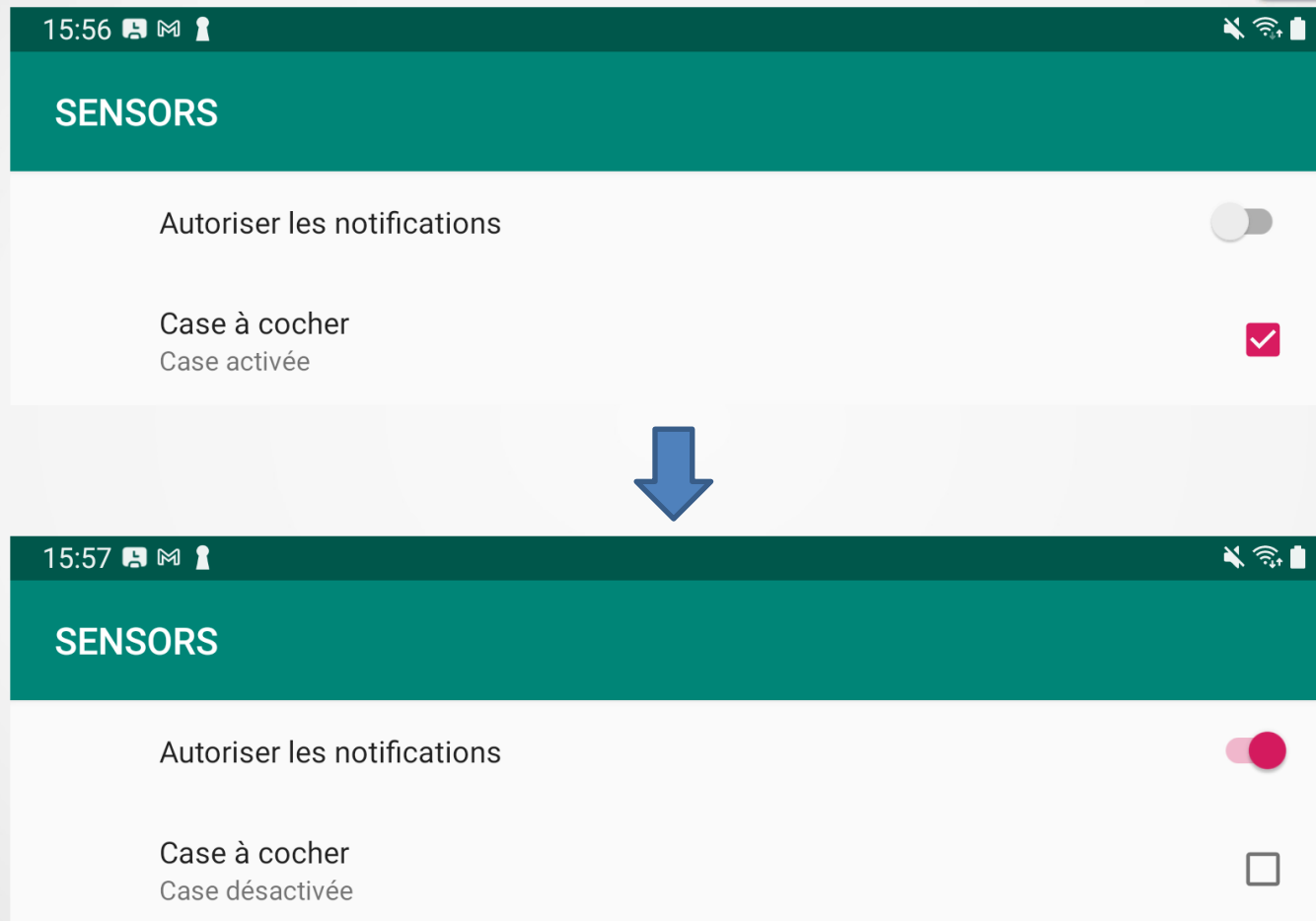
```
public class Sensors_preferences_activity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_sensors_preferences_activity);  
  
        getSupportFragmentManager()  
            .beginTransaction()  
            .setReorderingAllowed(true)  
            .add(R.id.fragment_container_view, MySettingsFragment.class, null)  
            .commit();  
    }  
}
```

```
<androidx.fragment.app.FragmentContainerView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/fragment_container_view"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
/>
```

```
public class MySettingsFragment extends PreferenceFragmentCompat {  
  
    @Override  
    public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {  
        setPreferencesFromResource(R.xml.root_preferences, rootKey);  
    }  
}
```

# Les menus de préférences

## Activité de préférence



Persistence et partage

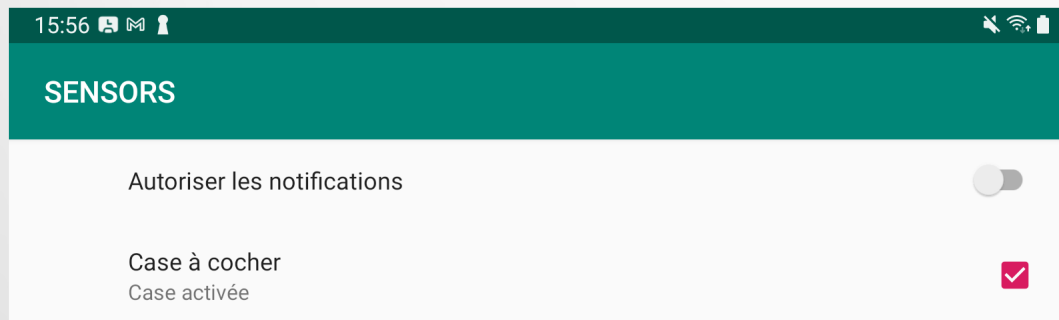
# Les menus de préférences

## Activité de préférence

Récupération de la valeur enregistrée  
Redémarrage de l'activité après destruction. On retrouve la valeur du paramètre.

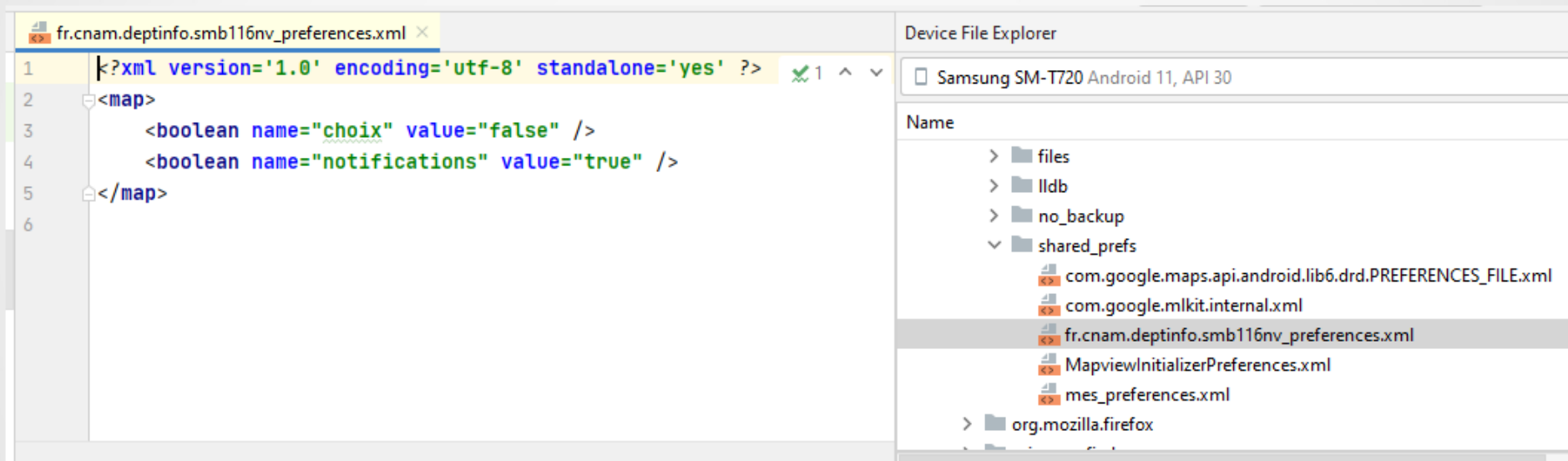
```
SharedPreferences SP =  
PreferenceManager.getDefaultSharedPreferences(this);  
boolean isChoix = SP.getBoolean("choix", false);
```

La clé de préférence



# Les menus de préférences

## Activité de préférence





# Android: Persistance et partage

## Les fichiers

# Les fichiers

## Mémoire interne et externe

### Mémoire interne

- Toujours disponible
- Les fichiers enregistrés ne sont accessibles que par l'application
- À la désinstallation, les fichiers de l'application sont supprimés

### Mémoire externe

- Pas toujours disponible
- Fichiers accessibles en lecture/écriture en dehors de notre contrôle
- Les fichiers privés d'une application ne sont supprimés à la désinstallation que s'ils sont sauvegardés dans un répertoire dédié (`getExternalFilesDir()`)

# Les fichiers

## Permissions

### Écriture sur mémoire externe

- **WRITE\_EXTERNAL\_STORAGE**

```
<uses-permission android:name =  
"android.permission.WRITE_EXTERNAL_STORAGE" />
```

- **READ\_EXTERNAL\_STORAGE**

```
<uses-permission android:name =  
"android.permission.READ_EXTERNAL_STORAGE" />
```

- **READ inclus dans WRITE**

### Ecriture sur la mémoire interne

- **Pas de restriction**

# Les fichiers

## Enregistrement en mémoire interne

- Créer un flux de sortie

```
FileOutputStream outputStream = openFileOutput(filename,  
Context.MODE_PRIVATE);
```

- Ecrire sur le flot de sortie

```
String res = "Hello";  
outputStream.write(res.getBytes());
```

- Fermeture du flot

```
outputStream.close();
```

Note: Seuls les modes `MODE_PRIVATE` et `MODE_APPEND` sont autorisés aujourd'hui (API >= 17).

Accessible  
uniquement par son  
application

`MODE_WORLD_READABLE`

En lecture seulement

`MODE_WORLD_WRITABLE`

En lecture/écriture

`MODE_APPEND`

Pour écrire en fin de fichier

# Les fichiers

## Lecture d'un fichier en mémoire interne

- Créer un flux de sortie

```
FileInputStream inputStream = openFileInput(filename);
```

- Ecrire sur le flot de sortie

```
byte[] data = new byte[512];  
inputStream.read(data);
```

- Fermeture du flot en entrée

```
inputStream.close();
```

# Les fichiers

## Autres méthodes utiles de Context

- `getFilesDir()`

Gets the absolute path to the file system directory where your internal files are saved.

- `getDir()`

Creates (or opens an existing) directory within your internal storage space.

- `deleteFile()`

Deletes a file saved on the internal storage.

- `fileList()`

Returns an array of files currently saved by your application.

# Les fichiers

## Cas des fichiers statiques

- Directement dans le .apk
- Par exemple : sons, données d'un jeu, dictionnaire, ressources alternatives...
- Les fichiers statiques sont enregistrés à la compilation dans le répertoire `/res/raw`

- Lecture des fichiers statiques

```
Resources res = getResources();  
InputStream is=res.openRawResource(R.raw.filename);
```

- Ecriture impossible puisque statique

# Les fichiers


## Enregistrement en mémoire cache interne

Enregistrer des données en mémoire cache plutôt que de manière permanente.

```
File file=  
File.createTempFile(fileName,null,context.getCacheDir());
```



prefix, suffix, directory. If suffix is null, .tmp is used



Création d'un fichier temporaire  
dans le répertoire cache interne  
de l'application

Ne pas oublier de supprimer les fichiers après usage



# Les fichiers

## Enregistrement d'un fichier en mémoire externe 1/3

Vérifier que le volume externe est monté

```
public boolean isExternalStorageWritable() {  
    String state = Environment.getExternalStorageState();  
    if (Environment.MEDIA_MOUNTED.equals(state))  
        return true;  
    return false;  
}
```

Fichier accessible en lecture/écriture

```
public boolean isExternalStorageReadable(){  
    String state = Environment.getExternalStorageState();  
    if (Environment.MEDIA_MOUNTED.equals(state) ||  
        Environment.MEDIA_MOUNTED_READ_ONLY.equals(state))  
        return true;  
    return false;  
}
```

Fichier au moins accessible en lecture

# Les fichiers

## Enregistrement d'un fichier en mémoire externe 2/3

Pour enregistrer les fichiers d'une application en mémoire externe:

- Utiliser `File.getExternalStorageDir(String type)` pour obtenir le chemin du répertoire

```
File file = new File(getExternalFilesDir(Environment.DIRECTORY_PICTURES), "DemoFile.jpg");
```

Possibilité de typer des sous répertoires pour garantir un rangement approprié des fichiers

- On spécifie des constantes: `Environment.DIRECTORY_MUSIC`, `DIRECTORY_PODCASTS`, `DIRECTORY_RINGTONES`, `DIRECTORY_ALARMS`, `DIRECTORY_NOTIFICATIONS`, `DIRECTORY_PICTURES` ou `DIRECTORY_MOVIES`.

# Les fichiers

## Enregistrement d'un fichier en mémoire externe 3/3

### Remarque :

Les fichiers sont automatiquement supprimés lors de la désinstallation de l'application

Sur les appareils avec plusieurs utilisateurs, chaque utilisateur a son propre stockage partagé isolé. Les applications n'ont accès au stockage partagé que pour l'utilisateur sous lequel elles s'exécutent.

# Les fichiers

## Fichiers publics

Pour partager des fichiers non liés à une application en mémoire externe, il faut utiliser l'un des ses sous répertoires

`getExternalStoragePublicDirectory()` pour obtenir la racine de la mémoire externe

Ces fichiers ne seront pas supprimés à la désinstallation de l'application

```
File file = new  
File(Environment.getExternalStoragePublicDirectory(  
Environment.DIRECTORY_PICTURES), albumName);
```

```
DIRECTORY_RINGTONES  
DIRECTORY_MUSIC  
DIRECTORY_PODCASTS  
...
```

Attention : Ne doit plus être utilisé en API  $\geq 29$ .  
Pour améliorer la confidentialité des utilisateurs,  
l'accès direct aux périphériques de stockage  
partagés/externes est interdit.

# Android: Persistance et partage

## Les bases de données

# Les bases de données SQLite et Android

## Base de données relationnelle

- S'exécute sans serveur
- Dans le processus de l'application
- <http://www.sqlite.org>
- API: android.database.sqlite

## Base de données dédiée à l'application

- Enregistrée en mémoire interne
- Dans un espace privé de l'application
- Inaccessible aux autres applications
- Sauf si la BD est enregistrée sur la SD card (il faut juste connaître son chemin d'accès)

## Base de données légère

- Éviter d'enregistrer des données binaires (images...)

# SQLiteOpenHelper

Un exemple de fournisseur:

Coordonnées des cinémas en latitude, longitude

Où puis je trouver un cinéma le proche de ma position ? ...

Persistance à l'aide de SQLite

```
CREATE TABLE cinemas (  
    Id integer primary key autoincrement,  
    Nom text not null,  
    Latitude text not null,  
    Longitude text not null);
```

# SQLiteOpenHelper

```
public class CinemaHelper extends SQLiteOpenHelper {  
  
    private static final String DATABASE_NAME = "cinemas.db";  
    private static final int DATABASE_VERSION = 1;  
  
    public static final String TABLE_CINEMAS = "Cinemas";  
    public static final String NOM_COLONNE_ID = "Id";  
    public static final String NOM_COLONNE_NOM = "Nom";  
    public static final String NOM_COLONNE_LATITUDE = "Latitude";  
    public static final String NOM_COLONNE_LONGITUDE = "Longitude";  
  
    public CinemaHelper(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
}
```



# SQLiteOpenHelper

```
private static final String CREATION_DB = "CREATE TABLE " + TABLE_CINEMAS + " (" +  
    NOM_COLONNE_ID + " integer primary key autoincrement, " +  
    NOM_COLONNE_NOM + " text not null, " +  
    NOM_COLONNE_LATITUDE + " text not null, " +  
    NOM_COLONNE_LONGITUDE + " text not null);";
```

@Override

```
public void onCreate(SQLiteDatabase db) {  
    db.execSQL(CREATION_DB);  
}
```

@Override

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
    if (oldVersion == newVersion)  
        return;  
    // Simplest implementation is to drop all old tables and recreate them  
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_CINEMAS + ";");  
    onCreate(db);  
}
```

Persistence et partage

# Méthodes de SQLiteDatabase

```
public Cursor query(String table,  
    String[] columns,  
    String selection,  
    String[] selectionArgs,  
    String groupBy,  
    String having,  
    String orderBy)
```

- columns : liste des colonnes à retourner (SELECT)
- selection : WHERE
- selectionArgs : arguments du WHERE
- groupBy : GROUP BY
- Having : HAVING
- orderBy : ORDER BY

```
public long insert(String table,  
    String nullColumnHack,  
    ContentValues values)
```

```
public int delete(String table,  
    String whereClause,  
    String[] whereArgs)
```

- whereClause : WHERE
- whereArgs : arguments du WHERE

```
public int update(String table,  
    ContentValues values,  
    String whereClause,  
    String[] whereArgs)
```

« SELECT \* FROM adherents WHERE nom LIKE ? AND age LIKE ? »

whereClause = "nom LIKE ? and age LIKE ?"  
String[] whereArgs = {"a%", "2%"};

Persistence et partage

# Cursor : Nombre de lignes

Méthodes	Commentaires
void close()	Closes the Cursor, releasing all of its resources and making it completely invalid.
int getCount()	Returns the numbers of rows in the cursor.

# Cursor : Gestion des colonnes

Méthodes	Commentaires
<code>int getColumnCount()</code>	Return total number of columns
<code>int getColumnIndex(String columnName)</code>	Returns the zero-based index for the given column name, or -1 if the column doesn't exist.
<code>String getColumnName(int columnIndex)</code>	Returns the column name at the given zero-based column index.
<code>String[] getColumnNames()</code>	Returns a string array holding the names of all of the columns in the result set in the order in which they were listed in the result.

# Cursor : Déplacement du curseur

Méthodes	Commentaires
boolean isAfterLast()	Returns whether the cursor is pointing to the position after the last row.
boolean isBeforeFirst()	Returns whether the cursor is pointing to the position before the first row.
boolean isFirst()	Returns whether the cursor is pointing to the first row.
boolean isLast()	Returns whether the cursor is pointing to the last row.
boolean move(int offset)	boolean move(int offset)
boolean moveToFirst()	Move the cursor to the first row.
boolean moveToLast()	Move the cursor to the last row.
boolean moveToNext()	Move the cursor to the next row.
boolean moveToPosition(int position)	Move the cursor to an absolute position.
boolean moveToPrevious()	Move the cursor to the previous row.
int getPosition()	Returns the current position of the cursor in the row set.

Persistence et partage

# Cursor : Récupération des données

Méthodes	Commentaires
<code>double getDouble(int columnIndex)</code>	Returns the value of the requested column as a double.
<code>float getFloat(int columnIndex)</code>	Returns the value of the requested column as a float.
<code>int getInt(int columnIndex)</code>	Returns the value of the requested column as an int.
<code>long getLong(int columnIndex)</code>	Returns the value of the requested column as a long.
<code>short getShort(int columnIndex)</code>	Returns the value of the requested column as a short.
<code>String getString(int columnIndex)</code>	Returns the value of the requested column as a String.
<code>int getType(int columnIndex)</code>	Returns data type of the given column's value.

# SQLiteOpenHelper

## Utilisation via un DAO par exemple

```
public final String[] ALL = new
String[]{NOM_COLONNE_ID,NOM_COLONNE_NOM,NOM_COLONNE_LATITUDE,NOM_COLONNE_LONGITUDE};

SQLiteOpenHelper dbHelper = new CinemaHelper(context);
 SQLiteDatabase db = dbHelper.getWritableDatabase();

Cursor cursor = db.query(TABLE_CINEMAS, ALL, NOM_COLONNE_ID + "=" + id, null, null, null, null);
if(cursor.moveToFirst()) {
    String nom = cursor.getString(cursor.getColumnIndex(NOM_COLONNE_NOM));
    String latitude = cursor.getString(cursor.getColumnIndex(NOM_COLONNE_LATITUDE));
    String longitude = cursor.getString(cursor.getColumnIndex(NOM_COLONNE_LONGITUDE));
    ...
    cursor.close();
}

dbHelper.close();
```

# SQLiteOpenHelper

## Utilisation via un DAO par exemple

Pour maintenir la consistance des données  
et prévenir la perte de données due à une terminaison anormale de l'exécution.  
Une transaction permet le traitement atomique d'une suite de requêtes.

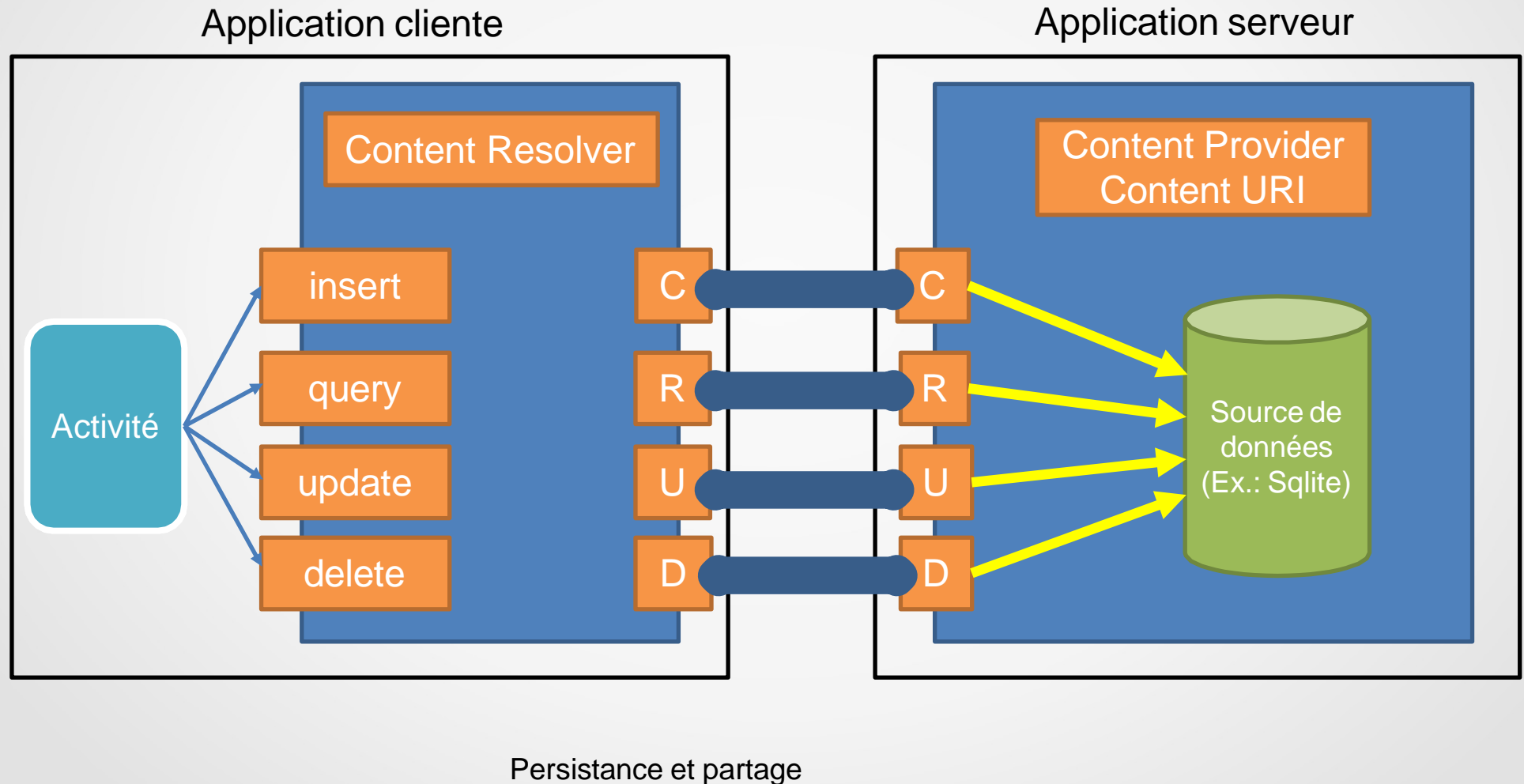
```
db.beginTransaction();

try {
    //traitement des requêtes
    ...
    // validation des modifications
    db.setTransactionSuccessful();
}
catch (SQLException e) {
    //signalement d'erreur
}
finally {
    // fin de la transaction
    db.endTransaction();
}
```

Persistence et partage



# ContentProvider et ContentResolver



# Exemple de ContentProvider

ContentProvider	Commentaires
Browser	Historique de navigation + marques-pages
CalendarContract	Gestion de l'agenda
CallLog	Historique des appels
ContactsContract	Gestion des contact
MediaStore	Musique, vidéo, images
Settings	Configuration de l'appareil
UserDictionary	Les mots ajoutés au dictionnaire

Persistence et partage

# ContentProvider et URI

**ACTION\_VIEW**: *content://contacts/people/1* -- Display information about the person whose identifier is "1".

**ACTION\_DIAL**: *content://contacts/people/1* -- Display the phone dialer with the person filled in.

**ACTION\_EDIT**: *content://contacts/people/1* -- Edit information about the person whose identifier is "1".

**ACTION\_VIEW**: *content://contacts/people/* -- Display a list of people, which the user can browse through.

# Son propre ContentProvider

```
public class CinemaProvider extends ContentProvider {
```

ContentProvider est une classe abstraite

Il faut:

- Persistance: quelconque, SQLite, fichier XML ...
- Une Uri, l'accès à mon propre « *content provider* »  
content://fr.cnam.deptinfo.smb116nv.provider/cinemas  
content://fr.cnam.deptinfo.smb116nv.provider/cinemas/3
- Réalisation avec implémentation des méthodes insert, update, delete et select
- Et dans le fichier de configuration

```
<provider
    android:name=".contentprovider.CinemaProvider"
    android:authorities="fr.cnam.deptinfo.smb116nv.provider"
    android:enabled="true"
    android:exported="true" />
```

# ContentProvider

```
public class CinemaProvider extends ContentProvider {  
  
    private CinemaHelper dbHelper = null;  
  
    @Override  
    public boolean onCreate() {  
        dbHelper = new CinemaHelper(getContext());  
        return true;  
    }  
}
```

# ContentProvider

```
public static final String CONTENT_TYPE = "text/plain";
public static final String CONTENT_ITEM_TYPE = "text/plain";

public static final String AUTHORITY = "fr.cnam.deptinfo.smb116nv.provider";
public static final String TABLE = "cinemas";
public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" + TABLE);

private static final String CONTENT_PATH = "cinemas";

private static final int URI_DIR = 2;
private static final int URI_ITEM = 1;
private static final UriMatcher URI_MATCHER;

URI_MATCHER = new UriMatcher(UriMatcher.NO_MATCH);
URI_MATCHER.addURI(CinemaContract.AUTHORITY, CONTENT_PATH, URI_DIR);
URI_MATCHER.addURI(CinemaContract.AUTHORITY, CONTENT_PATH + "/" + URI_ITEM);

@Override
public String getType(@NonNull Uri uri) {
    switch (URI_MATCHER.match(uri)) {
        case URI_DIR:
            return CinemaContract.CONTENT_TYPE;
        case URI_ITEM:
            return CinemaContract.CONTENT_ITEM_TYPE;
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}
```

Persistence et partage

# ContentProvider - CRUD

@Override

```
public Cursor query(@NonNull Uri uri, @Nullable String[] projection, @Nullable String selection, @Nullable String[] selectionArgs, @Nullable String sortOrder) {
```

```
    SQLiteDatabase db = dbHelper.getWritableDatabase();
```

```
    final SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
```

```
    switch (URI_MATCHER.match(uri)) {
```

```
        case URI_ITEM:
```

```
            queryBuilder.appendWhere(CinemaContract.NOM_COLONNE_ID + "=" + uri.getLastPathSegment());
```

```
        case URI_DIR:
```

```
            queryBuilder.setTables(CinemaContract.TABLE);
```

```
            break;
```

```
        default:
```

```
            throw new IllegalArgumentException("Unsupported URI:" + uri);
```

```
    }
```

```
    Cursor cursor = queryBuilder.query(db, projection, selection, selectionArgs, null, null, sortOrder);
```

```
    cursor.setNotificationUri(getContext().getContentResolver(), uri);
```

```
    return cursor;
```

```
}
```

Persistence et partage

# ContentProvider - CRUD

```
public Uri insert(@NonNull Uri uri, @Nullable ContentValues values)
```

```
    long nbLignes = db.insert(CinemaContract.TABLE, null, values);
```

```
public int delete(@NonNull Uri uri, @Nullable String selection, @Nullable String[]  
selectionArgs)
```

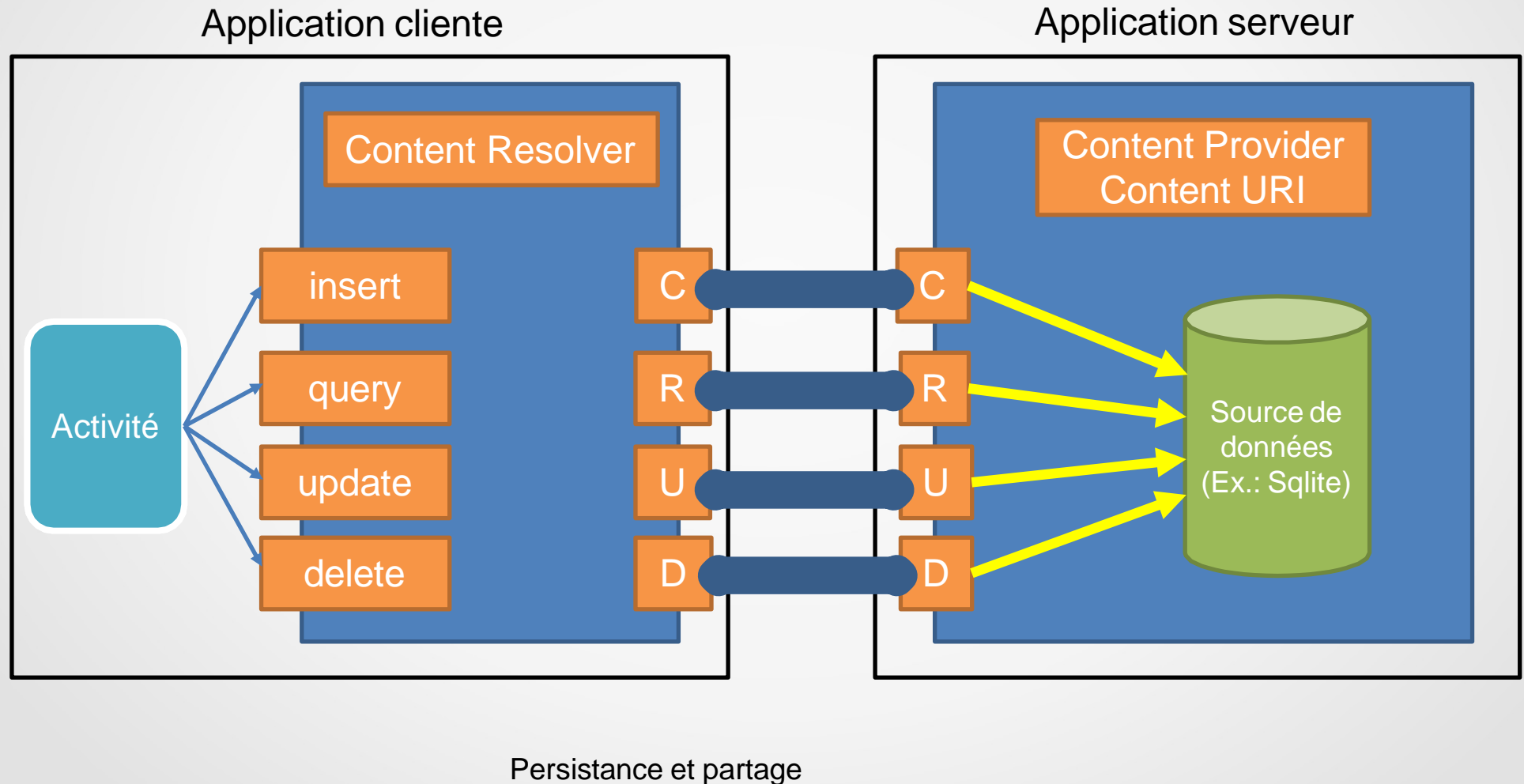
```
    int delCount = db.delete(CinemaContract.TABLE, where, selectionArgs);
```

```
public int update(@NonNull Uri uri, @Nullable ContentValues values, @Nullable String  
selection, @Nullable String[] selectionArgs)
```

```
    int updateCount = db.update(CinemaContract.TABLE, values, where, selectionArgs);
```



# ContentProvider et ContentResolver



# Comme les utiliser ?

Le ContentResolver :

```
ContentResolver cr = getContentResolver();
```

```
ContentValues values=new ContentValues();
```

```
values.put(NOM_COLONNE_NOM,« Cinéma Frères Lumières");
```

```
values.put(NOM_COLONNE_LATITUDE, 48.923775f);
```

```
values.put(NOM_COLONNE_LONGITUDE, 2.25632f);
```

```
Uri uri = cr.insert(CONTENT_URI, values);
```

Persistence et partage

# Méthodes du ContentResolver

```
public final int delete (Uri url,  
                        String where,  
                        String[] selectionArgs)
```

```
public final Uri insert (Uri url,  
                        ContentValues values)
```

```
public final Cursor query (Uri uri,  
                          String[] projection,  
                          String selection,  
                          String[] selectionArgs,  
                          String sortOrder)
```

```
public final int update(Uri uri,  
                      ContentValues values,  
                      String where,  
                      String[] selectionArgs)
```

- projection : liste des colonnes à retourner (SELECT)
- selection : WHERE
- selectionArgs : arguments du WHERE
- sortOrder : Ordre des colonne (ORDER BY)

# Sqlite3

```
F:\sdk\platform-tools>adb shell
root@android:/ # cd /data/data/fr.cnam.myfirstapp/databases
cd /data/data/fr.cnam.myfirstapp/databases
root@android:/data/data/fr.cnam.myfirstapp/databases # sqlite3 cinemas.db
sqlite3 cinemas.db
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> SELECT * FROM cinemas;
SELECT * FROM cinemas;
sqlite> SELECT * FROM cinemas;
SELECT * FROM cinemas;
1|Cinema Freres Lumiere|48.923775|2.25632
```

# Sqlite3

The screenshot displays the Android Studio IDE. At the top, a Java file named `fr.cnam.deptinfo.smb116n` contains the following code:

```
public List<Cinema> findAll() throws Exception {  
    List<Cinema> cinemas = new ArrayList<>();  
    Cursor cursor = db.query(TABLE_CINEMAS, ALL, selection: null, selectionArgs: null);  
    cursor.moveToFirst();  
    while (cursor.moveToNext()) {
```

Below the code editor is the **Database Inspector** window. It shows the database `cinemas.db` on a Samsung SM-T720 device. The `Cinemas` table is selected, and its schema is shown on the left:

- `Id` : INTEGER
- `Nom` : TEXT, NOT NULL
- `Latitude` : TEXT, NOT NULL
- `Longitude` : TEXT, NOT NULL

The table data is displayed in a grid with the following columns: `Id`, `Nom`, `Latitude`, and `Longitude`. The data is as follows:

Id	Nom	Latitude	Longitude
1	UGC Ciné Cité Les Halles	1.23000001907349	1.45000004768372
2	Majestic Bastille	2.23000001907349	2.45000004768372
3	Gaumont Champs-Élysées	3.23000001907349	3.45000004768372
4	Frères Lumières	0.0	0.0
5	UGC Ciné Cité Les Halles	1.23000001907349	1.45000004768372
6	Gaumont Champs-Élysées	3.23000001907349	3.45000004768372

The bottom of the screen shows the **Logcat** and **Database Inspector** tabs, along with other standard Android Studio tools like **Profiler**, **TODO**, **Problems**, **Debug**, **Terminal**, and **Build**.

Persistence et partage



# AdapterView et Adapter

Persistence et partage

# Listview



Persistence et partage

# Listview

Les composants graphiques qui affichent un ensemble d'items sont des **AdapterViews**

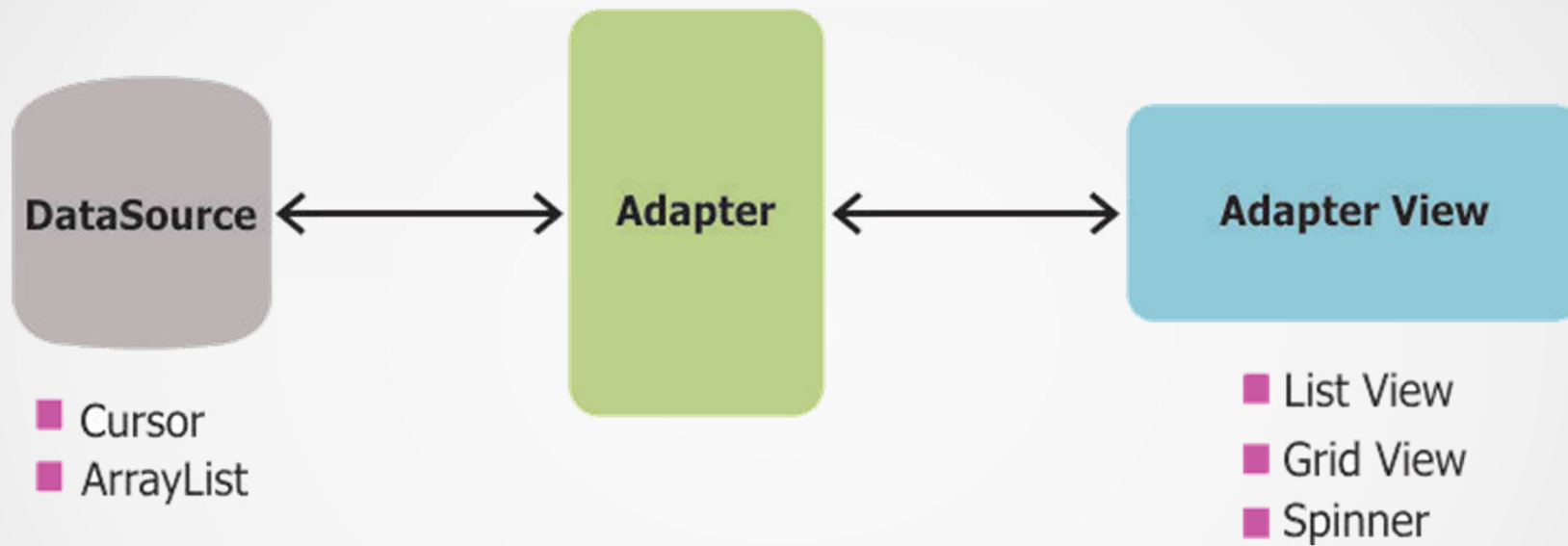
Pour fournir les items à ces composants graphiques, on utilise des **Adapter**

L'**Adapter** sert de lien avec les données à afficher et l'**AdapterView**. Bref c'est le design pattern adapter

"An Adapter object acts as a bridge between an AdapterView and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a View for each item in the data set."



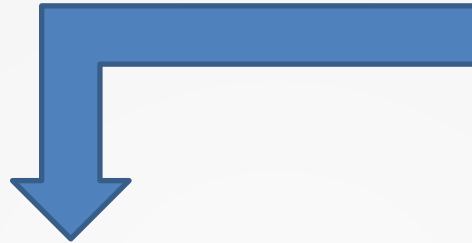
# Listview



Persistence et partage

# Listview (rowvoiture.xml)

Représentation d'une ligne, doit contenir un TextView



```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical|center_horizontal"
    android:minHeight="?android:attr/listPreferredItemHeight"
    android:textAppearance="?android:attr/textAppearanceLarge" />
```

Persistence et partage

# Listview (mylistview.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/listView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>

</LinearLayout>
```



# Listview - Objet métier: Voiture

```
public class Voiture {  
    private String id;  
    private String marque;  
    private String modele;  
    private String couleur;  
    private String immatriculation;
```

```
    public Voiture(String id, String marque, String modele, String couleur, String  
immat) {  
        this.id=id;  
        this.marque = marque;  
        this.modele = modele;  
        this.couleur = couleur;  
        this.immatriculation = immat;  
    }
```

```
@Override  
public String toString() {  
    return marque+" "+modele;  
}
```

```
public String getId() {  
    return id;  
}
```

```
public void setId(String id) {  
    this.id = id;    Persistence et partage  
}
```

...

Voiture
+id: String +marque: String +modele: String +couleur: String +immatriculation: String
+Voiture(id: String, marque: String, modele: String, couleur: String, immatriculation: String) +toString(): String +getId(): String +setId(id: String): void +getMarque(): String +setMarque(marque: String): void +getModele(): String +setModele(modele: String): void +getCouleur(): String +setCouleur(couleur: String): void +getImmatriculation(): String +setImmatriculation(immatriculation: String): void

# Listview (Activity)

```
public class MyListview extends Activity {

private ListView listview;
private ArrayList<Voiture> arrList;

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

    listview=(ListView) findViewById(R.id.listView1);

    arrList=new ArrayList<Voiture>();
    arrList.add(new Voiture("1","Peugeot","206","bleu","2563HX75"));
    arrList.add(new Voiture("2","Peugeot","508","rouge","2541VB62"));
    arrList.add(new Voiture("3","Ford","Fiesta","jaune","1254VC77"));
    arrList.add(new Voiture("4","Citroen","C4","blanc","2145DF55"));

    ArrayAdapter<Voiture> arrAdapt=
    new ArrayAdapter<Voiture>(this, R.layout.rowvoiture, arrList);
    listview.setAdapter(arrAdapt);
}
```

Persistence et partage

# Listview (Activity)

```
listview.setOnItemClickListener(new OnItemClickListener() {  
    public void onItemClick(AdapterView<?> parent, View view,int position, long id) {  
        Voiture v = (Voiture) parent.getItemAtPosition(position);  
        Log.i("##### LISTVIEW #####", v.getMarque()+" "+v.getModele());  
    }  
});
```

# Loader et CursorLoader

- Il n'est plus nécessaire de relancer une requête après un changement de configuration. Le curseur est automatiquement reconnecté.
- Loader : Pour charger des données depuis un ContentProvider de manière asynchrone.
- LoaderManager pour gérer plusieurs instances de Loader.
- CursorLoader, sous-classe d'AsyncTaskLoader pour effectuer des requêtes sur un ContentProvider et retourner un Cursor.
- Les requêtes ne sont plus exécutées dans l'UI thread.
- Le LoaderManager maintient les données en cas de changement de configuration (chgt d'orientation par exemple) en appelant les callbacks.

# Loader et CursorLoader

// Prépare le loader. Soit on reconnecte un existant, soit on en démarre un nouveau.  
`getLoaderManager().initLoader(0, null, this);`

// Il s'agit de l'adaptateur utilisé pour afficher les données de la liste.  
`SimpleCursorAdapter mAdapter;`

// Créez un adaptateur vide que nous utiliserons pour afficher les données chargées.  
`mAdapter = new SimpleCursorAdapter(...);`

`setListAdapter(mAdapter);`

`setListShown(false);`



# Loader et CursorLoader

Crée et renvoie un CursorLoader qui se chargera de créer un curseur pour les données affichées.

```
public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
    ...  
    return new CursorLoader(context, uri, projection,  
selection, selectionArgs, sortOrder);  
}
```

# Loader et CursorLoader

Échange l'ancien et le nouveau curseur. (Le framework se chargera de fermer l'ancien curseur).

```
public void onLoadFinished(Loader<Cursor> loader, Cursor data)
{
    mAdapter.swapCursor(data);

    // The list should now be shown.
    if (isResumed()) {
        setListShown(true);
    } else {
        setListShownNoAnimation(true);
    }
}
```

# Loader et CursorLoader

Ceci est appelé lorsque le dernier curseur fourni à `onLoadFinished()` ci-dessus est sur le point d'être fermé. Nous devons nous assurer que nous ne l'utilisons plus.

```
public void onLoaderReset(Loader<Cursor> loader) {  
    mAdapter.swapCursor(null);  
}
```