

Développement d'Applications Mobiles sous Android



Abdelhak-Djamel Seriai

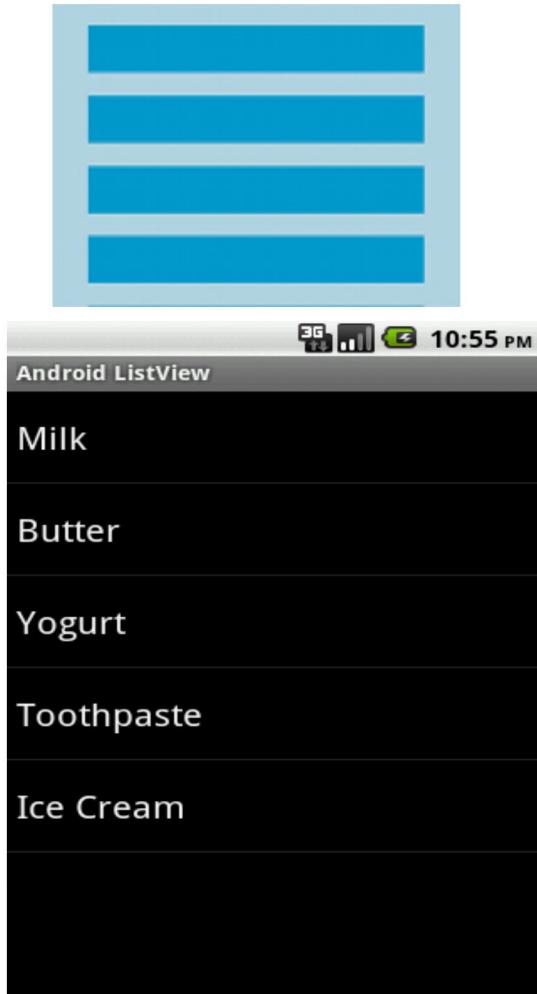
Gabarits avec un adaptateur

Gabarits avec un adaptateur

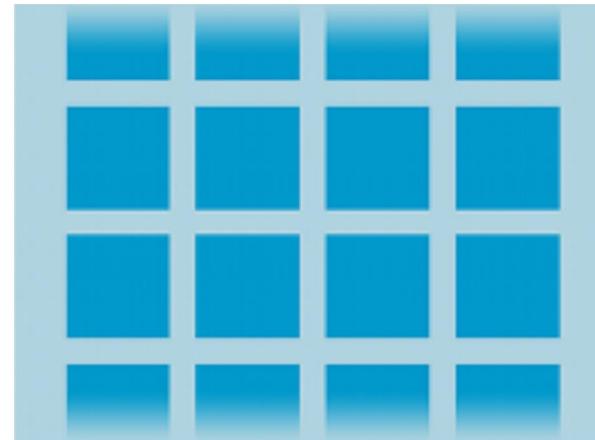
- ◆ Les éléments (vues) d'un gabarit (layout) peuvent être
 - statiques : Les éléments (les vues) qui composent le gabarit n'évoluent pas ni par rapport à leur identité ni par rapport à leur nombre
 - Exemple du **LinearLayout**, **RelativeLayout**
 - Dynamiques : Les éléments (les vues) qui composent le gabarit peuvent évoluer (changent) par rapport à leur identité comme par rapport à leur nombre
 - Exemple du **ListView** et du **GridView**

Layouts Dynamiques

List View



Grid View



Layouts Dynamiques

- ◆ Les identités de leurs éléments ainsi que leur nombre peuvent changer dynamiquement
- ◆ L'adaptation dynamique de leur contenu (quels éléments à afficher) est difficile à gérer par le programmeur
 - Gestion (Suppression/remplacement) des éléments (vues) en cas de défilement (scrolling)
 - Adaptation du nombre d'éléments par rapport à la taille de l'écran

Objet/class Adapter

- ◆ Les gabarits dynamiques dont les élément sont récupérés dynamiquement sont créés comme une sous classe de AdapterView
- ◆ Une sous classe AdapterView utilise un objet de la classe Adapter pour lier le gabarit correspondant à ses données (ses éléments/ses vues).
- ◆ Un objet Adapter agit comme un intermédiaire entre une source de données et un gabarit AdapterView
- ◆ L'objet Adapter récupère les données d'une source comme un tableau ou une requête d'une base de données et convertit chaque entrée en une vue qui peut être ajoutée dans un gabarit AdapterView

Remplissage d'un Adapter View

- ◆ Les deux Adapter les plus courants sont :
 - ArrayAdapter : utilisé quand la source de données est un tableau
 - SimpleCursorAdapter : Utilisé quand la source de donnée est un curseur (Cursor)

Exemple ArrayAdapter

- ◆ Affichage des éléments d'un tableau comme une ListView

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

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

Personnaliser un Adapter

- ◆ Par défaut, **ArrayAdapter** crée une vue pour chaque élément(item)
- ◆ La méthode **toString()** est appelée sur chaque élément du tableau pour retourner le contenu d'une vue **TextView**
- ◆ Pour personnaliser l'apparence de chaque items (éléments) de la liste il faut surcharger la méthode **toString()** pour les objets du tableau
- ◆ Pour personnaliser les vues à intégrer au gabarit, par exemple afficher une image dans un **ImageView** au lieu de **TextView** : Spécialiser **ArrayAdapter** et surcharger la méthode **getView()** pour retourner le type de vue désirée.

Persistante des données

Persistante de l'état des applications/activités

Persistante de l'état des applications/activités

- Besoin de conserver l'état de l'interface utilisateur
 - Lorsqu'un utilisateur navigue dans une application
 - Pour préparer le retour sur les écrans précédents
- Android gère le cycle de vie d'une application
 - Une activité d'arrière plan peut être déchargée de la mémoire en fonction de la politique de gestion des ressources du système
 - Besoin de restaurer l'état d'une activité entre deux sessions → sauvegarde de l'état d'une activité
 - Utilisation des méthodes de cycle de vie de l'activité

Gestion de la persistance des activités

- La méthode **onSaveInstanceState()** d'une activité est appelée lorsque le système a besoin de libérer des ressources et de détruire l'activité
 - Utilisation d'un objet de **Bundle** pour stocker les données
 - Passé en paramètre aux méthodes
 - **OnCreate()** : rétablir l'interface lors de la création
 - **OnRestoreInstanceState()** : rétablir l'interface lors de la restauration
 - Par défaut :
 - Les valeurs de toutes les vues possédant un attribut **id** renseigné sont enregistrées, puis restaurées
 - **OnSaveInstanceState()** enregistre l'état des vues identifiées dans un objet **Bundle**.
 - L'objet **Bundle** est ensuite passé aux **onCreate()** et **onRestoreInstanceState()** pour restaurer l'état de l'activité

Gestion de la persistance des activités

```
import android.R;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class SauvegardeEtatActivite extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        ...
    }

    @Override
    protected void onSaveInstanceState(Bundle savedInstanceState) {
        ...
    }

    @Override
    protected void onDestroy() {
        ...
    }
}
```

Gestion de la persistance des activités

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

```
@Override  
protected void onSaveInstanceState(Bundle savedInstanceState) {  
    super.onSaveInstanceState(savedInstanceState);  
    Toast.makeText(this, "Etat de l'activité sauvegardé",  
        Toast.LENGTH_SHORT).show();    }
```

```
@Override  
protected void onRestoreInstanceState(Bundle savedInstanceState) {  
    super.onRestoreInstanceState(savedInstanceState);  
    Toast.makeText(this, "Etat de l'activité restauré",  
        Toast.LENGTH_SHORT).show();    }
```

```
@Override  
protected void onDestroy() {  
    super.onDestroy();  
    Toast.makeText(this, "L'activité est détruite", Toast.LENGTH_SHORT)  
        .show();    }
```

Gestion de la persistance des activités

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

    <TextView android:id="@+id/nomDescription" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Saisissez votre nom :>
    </TextView>

    <EditText android:id="@+id/nom" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:textSize="18sp">
    </EditText>

    <TextView android:id="@+id/messageDescription"
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:text="Saisissez un message (qui ne sera pas enregistré) :>
    </TextView>

    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:textSize="18sp">
    </EditText>
</LinearLayout>
```

Configurer le mode de conservation des activités

- Mode de sauvegarde par défaut
 - Avantage : pas de code spécifique
 - Inconvénient : pas adapté à certaines situations
 - Ne pas sauvegarder les valeurs de certains champs qui possèdent un identifiant
- Personnalisation de l'enregistrement de l'état d'une activité
 - Redéfinitions des méthodes **onSaveInstanceState()**, **onCreate()** et **onRestoreInstanceState()**
 - Utilisation de l'objet **Bundle** pour lire et écrire des valeurs

Configurer le mode de conservation des activités

```
import projet.seraii.android.R;
import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class SauvegardeEtatActivite extends Activity {

    private final static String MA_CLE = "MaCle";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
    }
    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        ...
    }
    @Override
    protected void onSaveInstanceState(Bundle savedInstanceState) {
        ...
    }
    @Override
    protected void onDestroy() {
        ...
    }
}
```

Configurer le mode de conservation des activités

```
@Override  
protected void onSaveInstanceState(Bundle savedInstanceState) {  
    super.onSaveInstanceState(savedInstanceState);  
    savedInstanceState.putString(MA_CLE, "Ma valeur !");  
    Toast.makeText(this, "onSaveInstanceState",  
        Toast.LENGTH_SHORT).show();  
}
```

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    if ((savedInstanceState != null) &&(savedInstanceState.containsKey(MA_CLE))) {  
        String val = savedInstanceState.getString(MA_CLE);  
        Toast.makeText(this, "onCreate() : " + val,  
            Toast.LENGTH_SHORT).show();  
    }  
    setContentView(R.layout.persistence_etat_activite1);  
}
```

Stockage dans des fichiers

Stockage dans des fichiers

- Lire, écrire dans le système de fichiers
 - **OpenFileOutput()** : ouvrir un fichier en écriture ou de le créer s'il n'existe pas
 - Retourne un **FileOutputStream**
 - Par défaut : le fichier est écrasé s'il existe
 - Ecrire dans le fichier : **write()**
 - Fermer le fichier : **close()**
 - **DeleteFile()** : supprimer un fichier à partir de son nom
- Gestion de fichiers
 - **FileList()** : retourne tous les fichiers locaux de l'application
 - **GetFileDir()** : retourne le chemin absolu du répertoire où tous les fichiers créés par openFileOutput
 - **GetFileStreamStore** : retourne le chemin absolu du répertoire du fichier passé en paramètre

Stockage dans des fichiers

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Partager un fichier avec d'autres applications

- Par défaut :
 - Les fichiers créés par la méthode **openFileOutput** sont restreints à l'application
- Spécification d'un mode d'ouverture
 - **MODE PRIVATE** : mode par défaut, fichier accessible uniquement par l'application
 - **MODE WORLD READABLE** : lecture pour les autres application mais pas la modification
 - **MODE WORLD WRITABLE** : lecture/écritures aux autres applications
 - **MODE APPEND** : ajouter des données en fin de fichier. Peut être combiné avec un autre mode

Base de données SQLite

SQLite

- SQLite : base de données relationnelle
 - Légère, gratuite et open Source
 - [Www.sqlite.org](http://www.sqlite.org)
 - S'exécute sans nécessiter de serveur → exécution des requêtes dans le même processus de l'application
 - Chaque BD SQLite est réservée à son application créatrice
 - Utilisation d'un fournisseur de contenu pour partager une BD
 - Possibilité de créer plusieurs BDs par application

Création et mise à jour de BD SQLite

- Utilisation de la classe **SQLiteOpenHelper**

```
private class MaBaseOpenHelper extends SQLiteOpenHelper {  
  
    public MaBaseOpenHelper(Context context, String nom, CursorFactory  
                           cursorfactory, int version) {  
        super(context, nom, cursorfactory, version);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        //code de création}  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        // code de mise à jour  
    }  
}
```

Création et mise à jour de BD SQLite

```
class MaBaseOpenHelper extends SQLiteOpenHelper {  
  
    public MaBaseOpenHelper(Context context, String nom, CursorFactory  
                           cursorfactory, int version) {  
        super(context, nom, cursorfactory, version);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL(REQUETE_CREATION_BD);  
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        //Dans notre cas, nous supprimons la base et les données pour en  
        // créer une nouvelle ensuite.  
  
        db.execSQL("drop table " + TABLE_PLANETES + ";");  
        // Création de la nouvelle structure.  
  
        onCreate(db);  
    }  
}
```

Création et mise à jour de BD SQLite

```
private static final int BASE_VERSION = 1;
private static final String BASE_NOM = "planetes.db";

private static final String TABLE_PLANETES = "table_planetes";

public static final String COLONNE_ID = "id";
public static final int COLONNE_ID_ID = 0;
public static final String COLONNE_NOM = "nom";
public static final int COLONNE_NOM_ID = 1;
public static final String COLONNE_RAYON = "rayon";
public static final int COLONNE_RAYON_ID = 2;

/**
 * La requête de création de la structure de la base de données.
 */
private static final String REQUETE_CREATION_BD = "create table "
    + TABLE_PLANETES + " (" + COLONNE_ID
    + " integer primary key autoincrement, " + COLONNE_NOM
    + " text not null, " + COLONNE_RAYON + " text not null);";

/**
 * L'instance de la base qui sera manipulée au travers de cette classe
 */
private SQLiteDatabase maBaseDonnees;
```

Création et mise à jour de BD SQLite

- Accéder à une base de données
 - Utilisation des méthodes **getReadableDatabase()** et **getWritableDatabase**

```
private MaBaseOpenHelper baseHelper;  
  
public PlanetesDBAdaptateur(Context ctx) {  
    baseHelper = new MaBaseOpenHelper(ctx, BASE_NOM, null, BASE_VERSION);  
}
```

```
public SQLiteDatabase open() {  
    maBaseDonnees = baseHelper.getWritableDatabase();  
    return maBaseDonnees;  
}
```

Accéder à une base de données

- Besoin de rendre les traitements indépendants par rapport au type de la source des données
 - Utilisation des adaptateurs : encapsuler toutes les actions sur la BD dans une classe dédiée
 - Voir code associé

Effectuer une requête

- Utilisation de la méthode query()
 - Retourne un curseur permettant de naviguer dans les résultats
 - Objet Cursor

```
public Planete getPlanete(String nom) {  
    Cursor c = maBaseDonnees.query(TABLE_PLANETES,  
        new String[] {  
            COLONNE_ID, COLONNE_NOM, COLONNE_RAYON },  
        null, null, null,  
        COLONNE_NOM + " LIKE " + nom, null);  
  
    return cursorToPlanete(c);  
}
```

```
public Cursor getAllPlanetesCurseur() {  
    return maBaseDonnees.query(TABLE_PLANETES, new String[] {  
        COLONNE_ID,  
        COLONNE_NOM, COLONNE_RAYON },  
        null, null, null, null);  
}
```

Effectuer une requête

- **Opération sur objet Cursor**

abstract boolean moveToFirst()

Move the cursor to the first row.

abstract boolean moveToLast()

Move the cursor to the last row.

abstract boolean moveToNext()

Move the cursor to the next row.

abstract boolean moveToPosition(int position)

Move the cursor to an absolute position.

abstract boolean moveToPrevious()

Move the cursor to the previous row

Utilisation d'un CursorAdapter

- ◆ Exemple : création une liste de personnes avec leurs noms et leurs numéros de téléphones
 - Exécuter une requête qui retourne un objet **Cursor** qui contient une ligne pour chaque personne et deux colonnes pour le nom et le numéro de téléphone respectivement.
 - Créer un « **string array** » qui spécifie quelle colonne dans les lignes du Curseur à insérer dans le gabarit et un « **integer array** » qui spécifie la vue qui correspond à chaque colonne.

Utilisation d'un CursorAdapter

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,  
                      ContactsContract.CommonDataKinds.Phone.NUMBER};  
  
int[] toViews = {R.id.display_name, R.id.phone_number};
```

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,  
          R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);  
ListView listView = getListView();  
listView.setAdapter(adapter);
```