

Développement Mobile sur Android

Master CCI

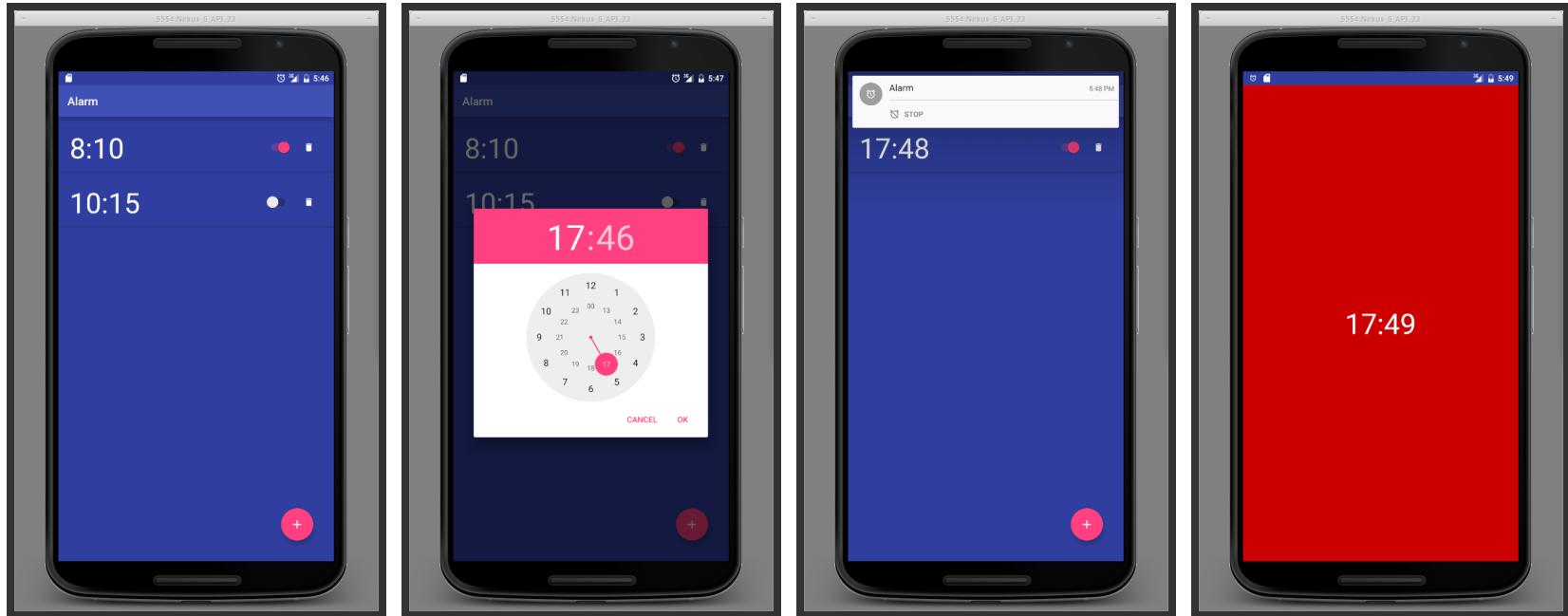
Bertrand Estellon - DII - Aix-Marseille Université

Organisation de l'enseignement

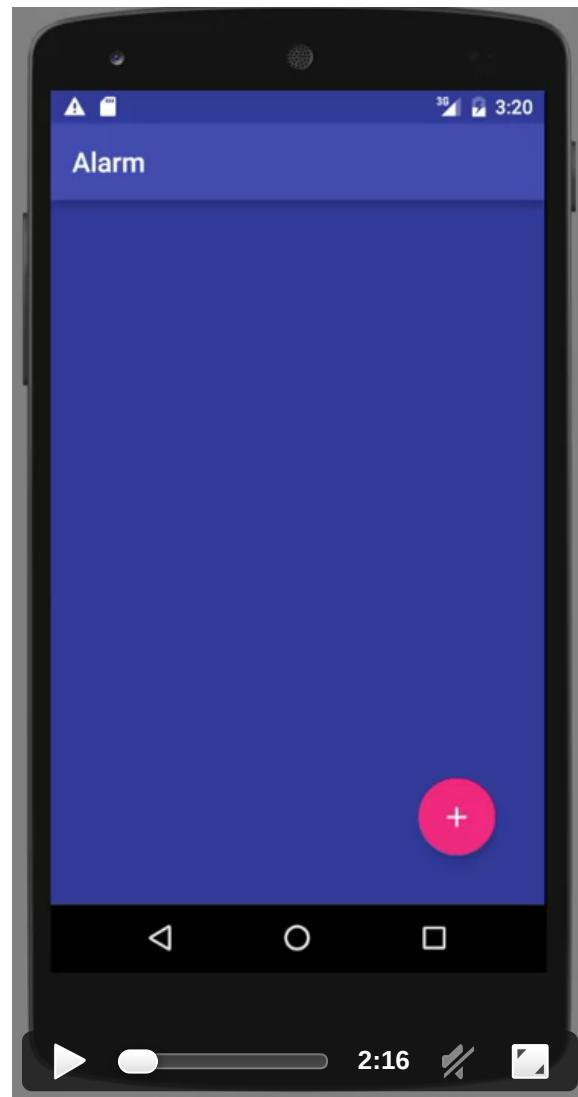
- Objectifs :
 - Connaître les différents composants d'une application Android
 - Savoir organiser une application
 - Savoir utiliser une documentation
 - Progresser en Java
- Évaluation :
 - Seuls les TP seront évalués
 - Il est important de venir à tous les TP
 - Les notes sont individuelles
 - Vous devez donc travailler seul

Le projet

Un réveil :



Démo du projet



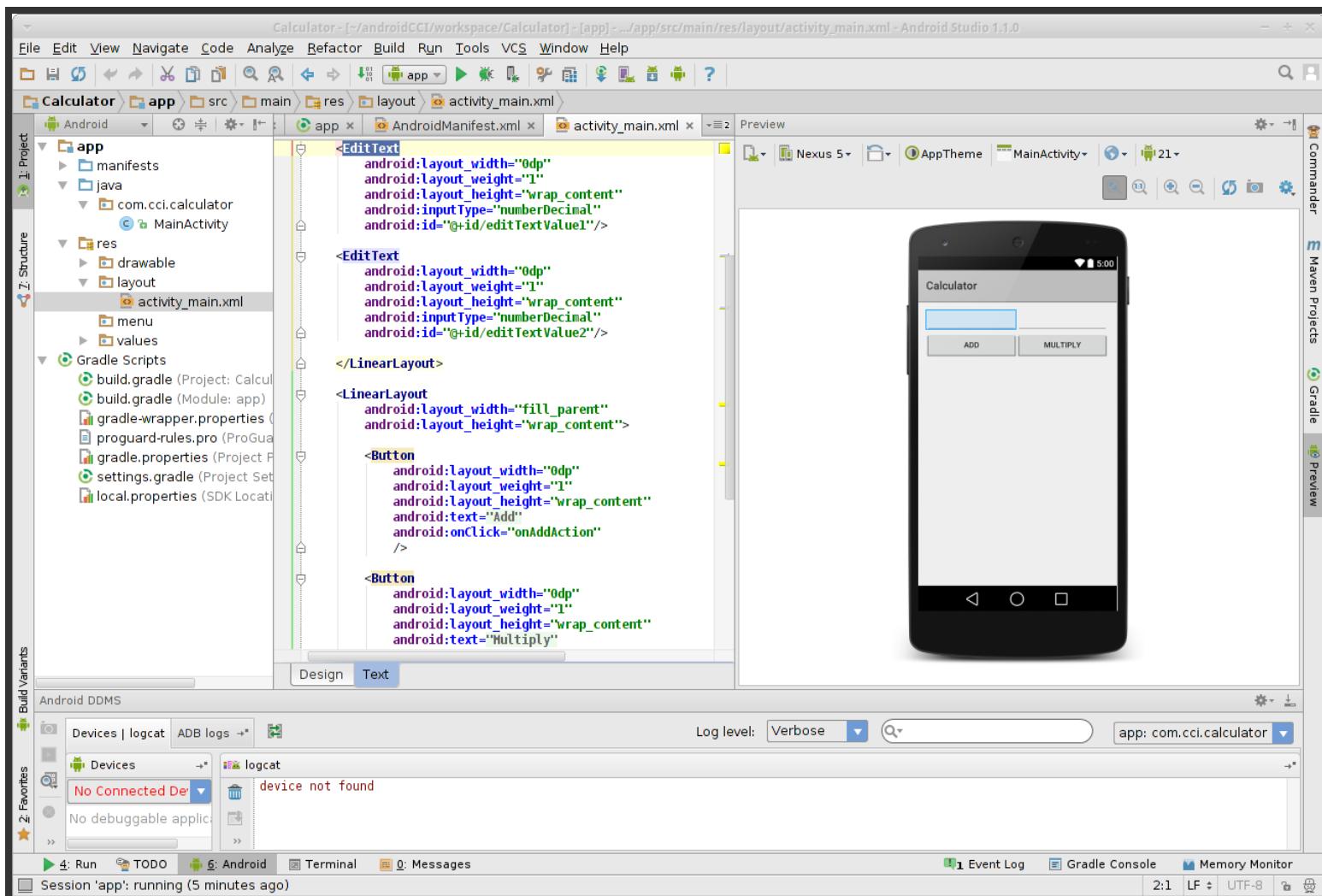
Les fonctionnalités du projet

- Ajout d'une alarme
- Modification de l'heure des alarmes
- Activation/Désactivation d'une alarme
- Suppression d'une alarme
- Notification de l'utilisation
- Sonnerie de l'alarme
- Réveil de l'écran en cas de sonnerie
- Possibilité d'arrêter l'alarme

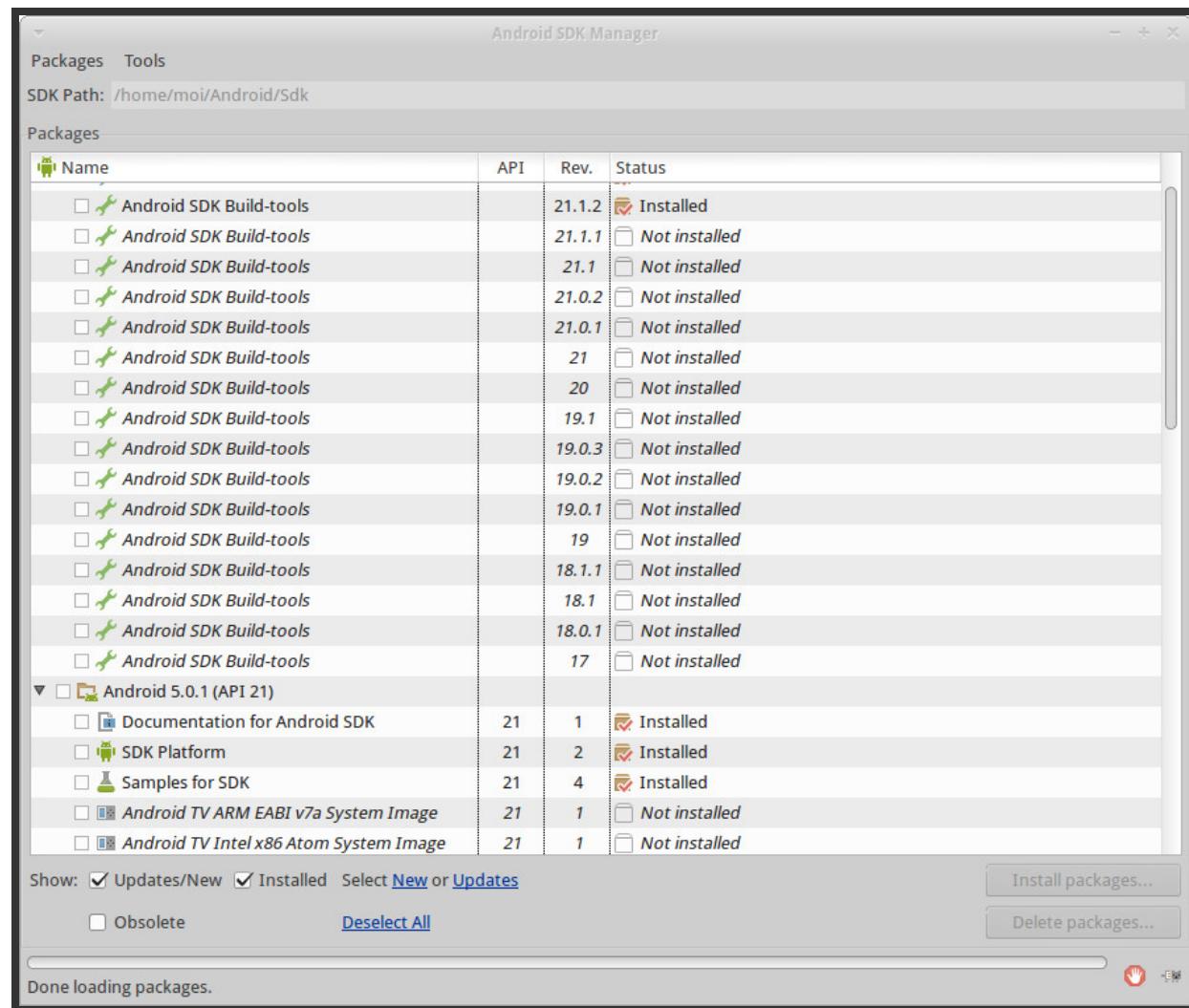
Les outils pour développer sur Android

- Nous devons utiliser le SDK d'Android qui fournit :
 - les librairies Java d'Android
 - un émulateur pour tester vos applications
 - des images du système Android
- Nous devons utiliser Java donc il faut :
 - Maîtriser les concepts de la programmation objet
 - Savoir lire une documentation Java
- Nous devons utiliser un IDE adapté, c'est-à-dire, Android Studio :
 - Première étape du TP, la configuration de l'IDE
 - Il est basé sur IntelliJ IDEA de Jet Brains

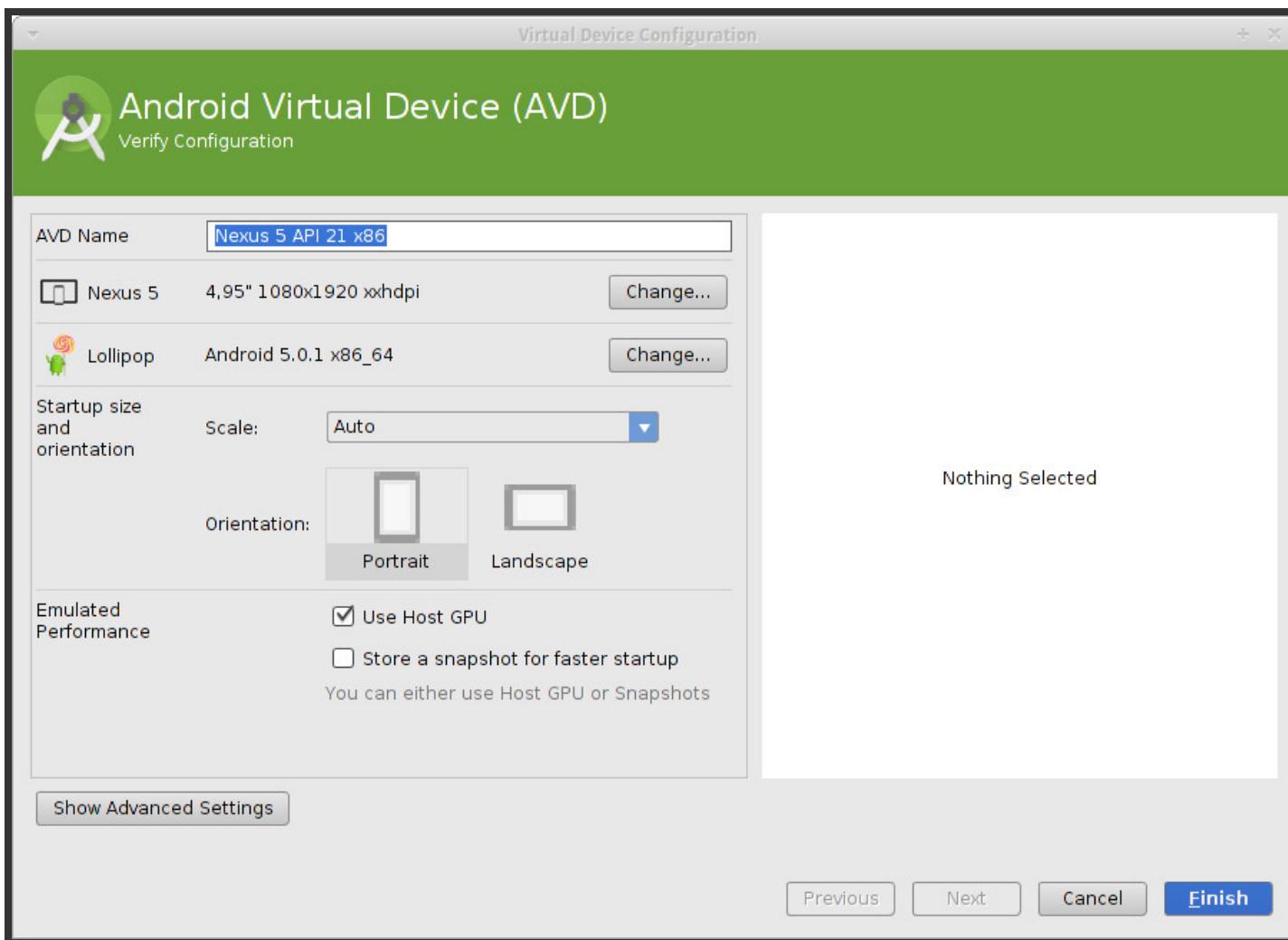
Android Studio



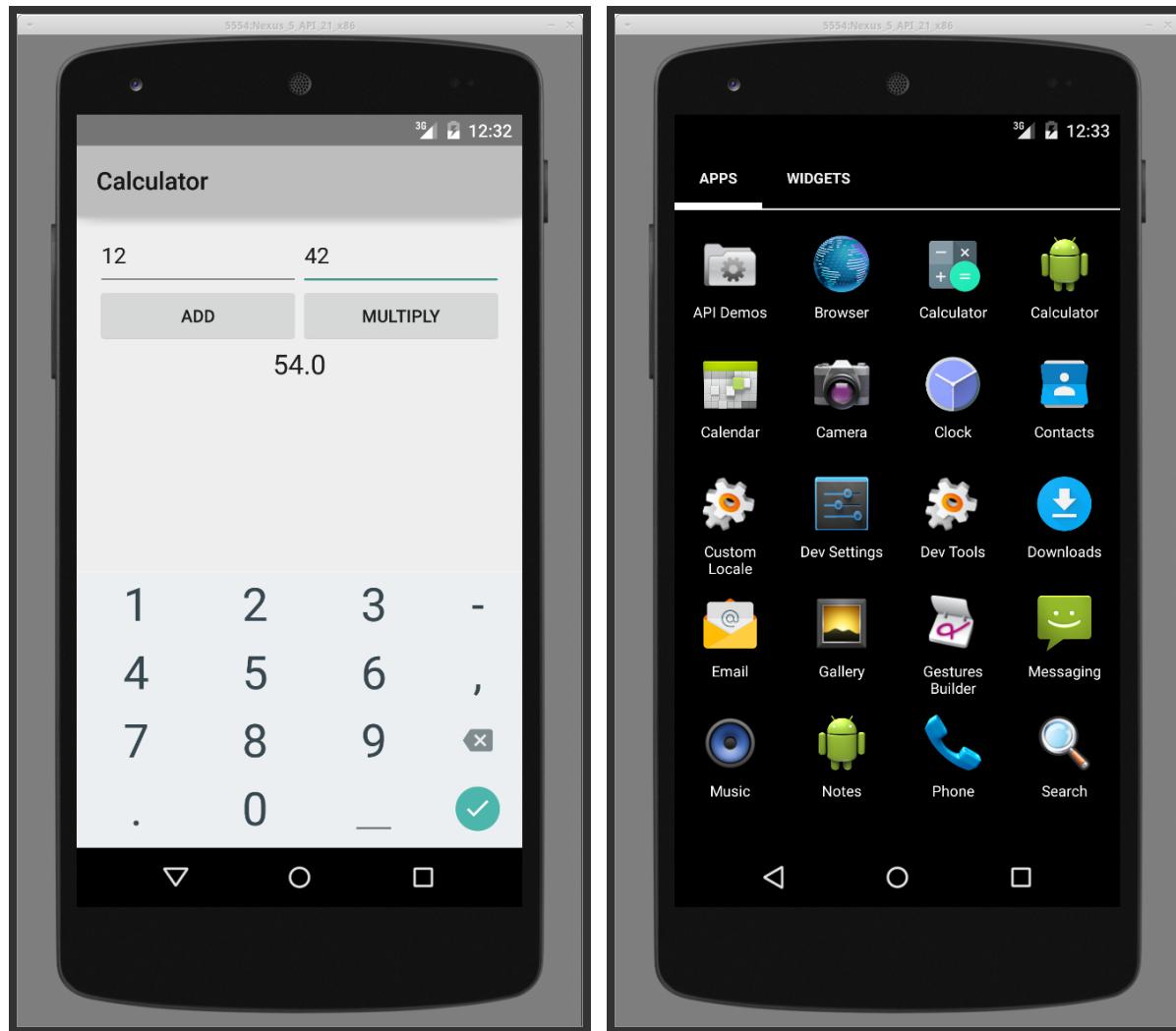
Android SDK



AVD Manager



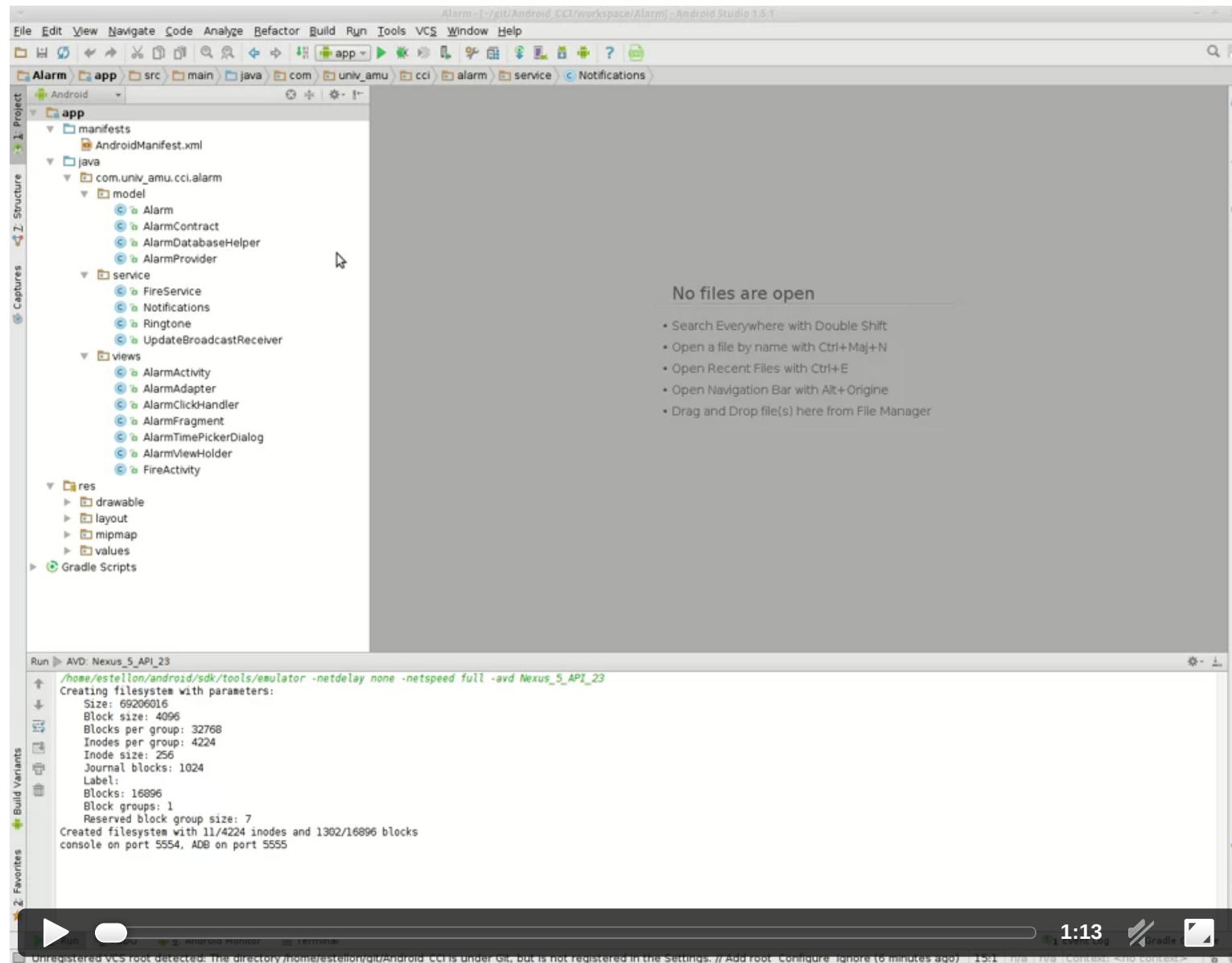
Émulateur



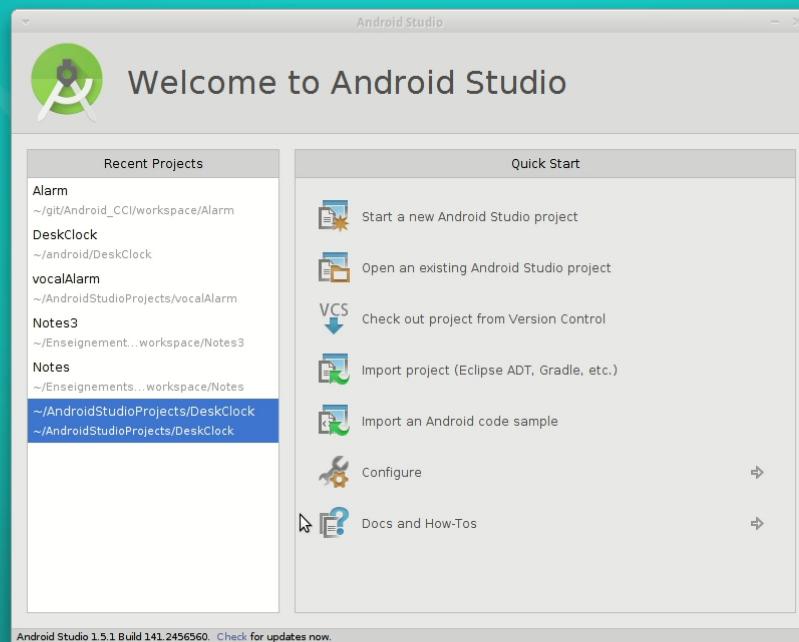
Installation



Création d'une machine virtuelle



Création d'un projet

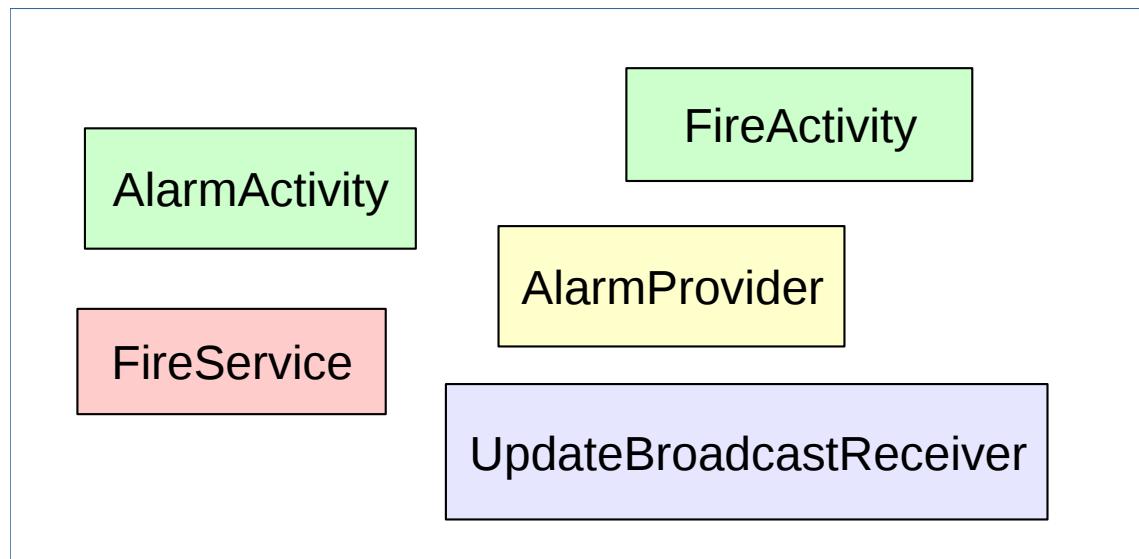


1:51



Les composants d'une application

- **Activités** : les différents écrans de l'application
- **Services** : un composant qui s'exécute en tâche de fond
- **Fournisseur de contenu** : gestion et partage des données
- **Receveur de *broadcasts*** : reception de notifications



Sécurité dans le système Android

- Chaque application est exécutée dans une sandbox (bac à sable) :
 - Android est système Linux multi-utilisateur
 - Chaque application est un utilisateur différent
 - Chaque fichier de l'application n'est accessible que par elle
 - Par défaut, chaque processus possède sa machine virtuelle
 - Chaque application a son propre processus Linux

Communication entre composants

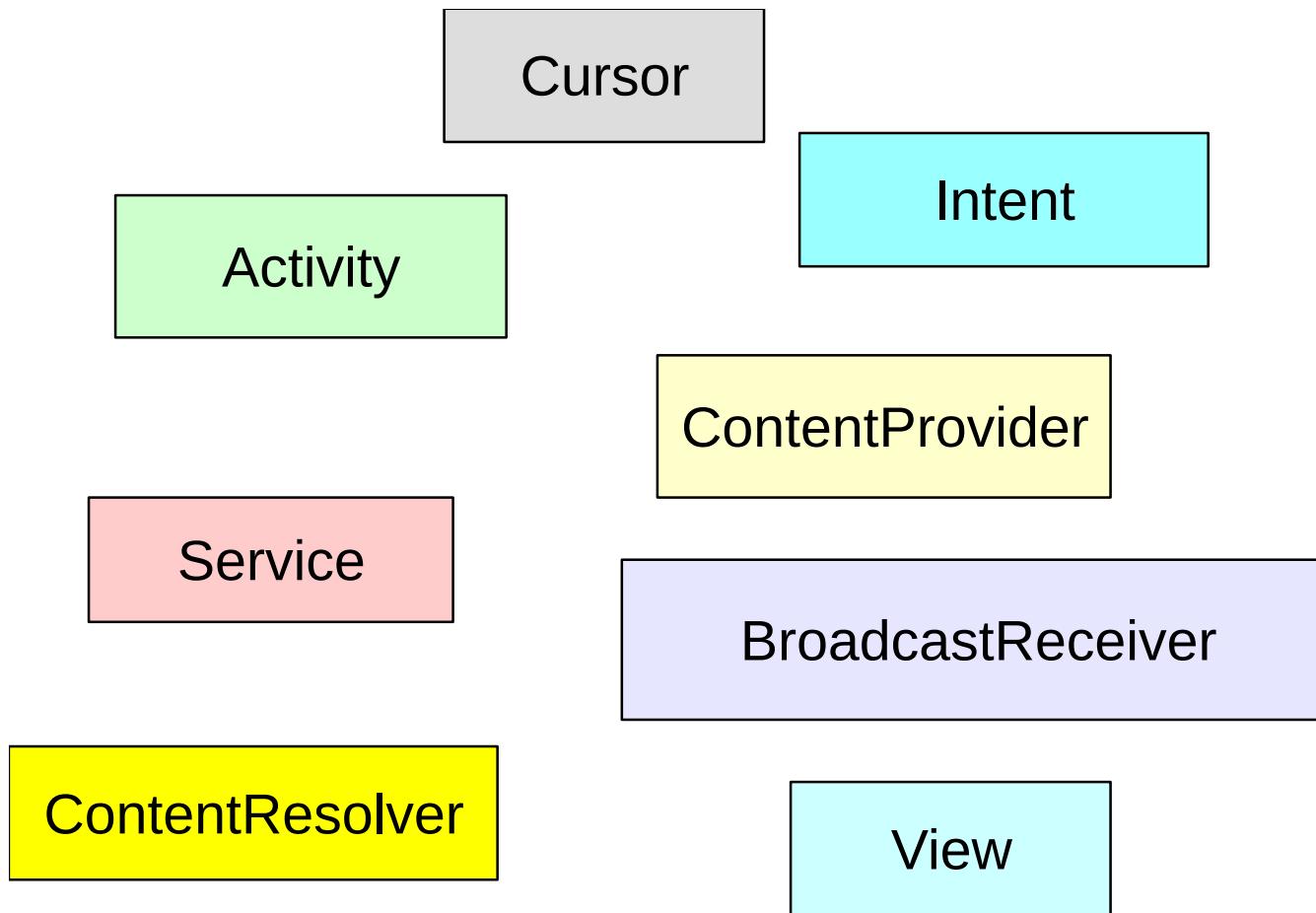
- Les **activités**, les **services** et les **recepteurs de broadcasts** sont activés par des messages *asynchrones* : les **intents**

```
Intent intent = new Intent(context, FireService.class)
intent.setAction(FireService.ACTION_FIRE);
```

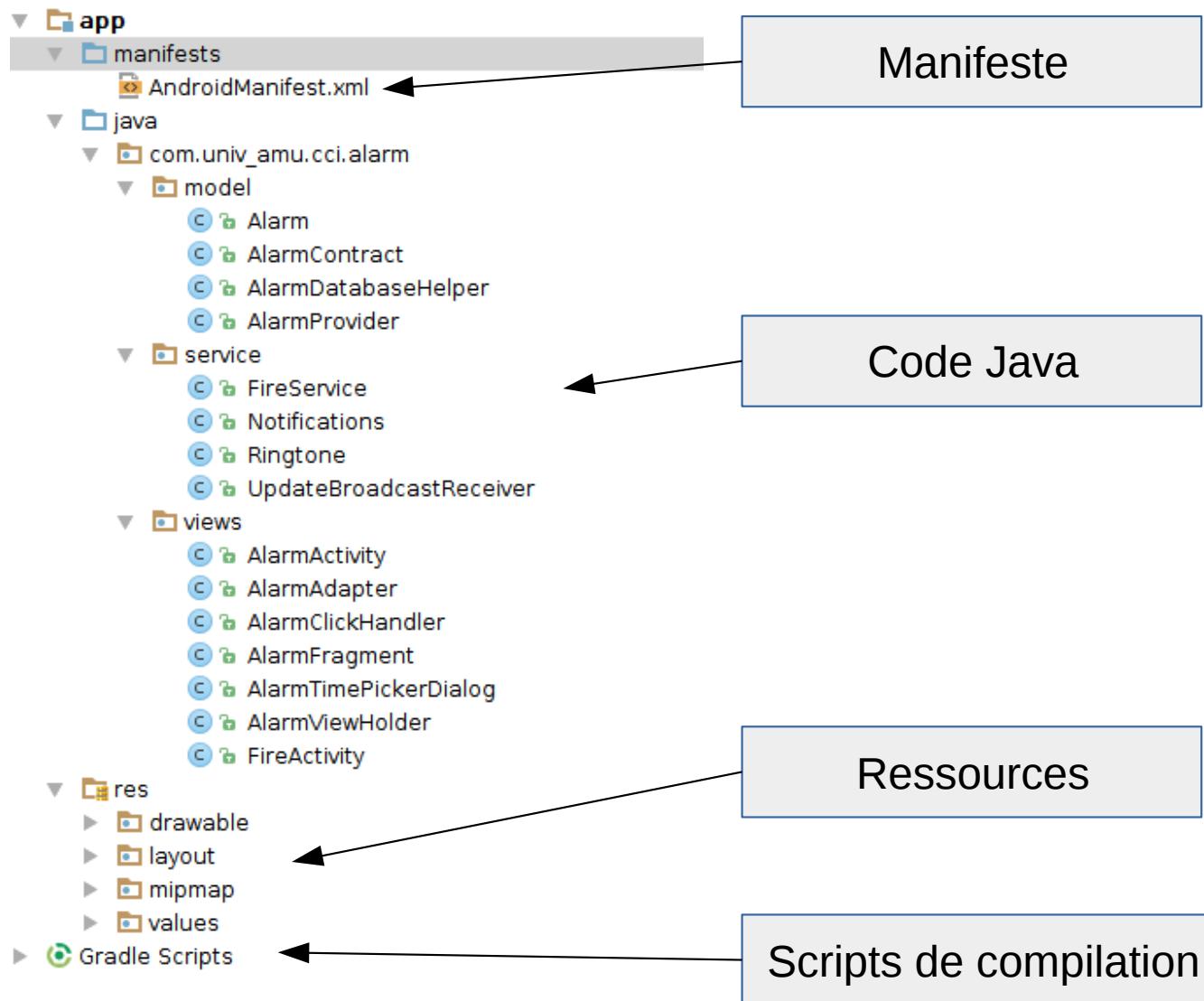
- Les **fournisseurs de contenu** sont activés par des requêtes effectuées à l'aide d'un **résolveur de contenu**.
- Une **requête** est composée :
 - d'une URI de la forme content://authority/path
 - d'une action query, delete, update ou insert

```
Uri uri = Uri.parse("content://fr.univamu.cci.alarm/alarms");
resolver.insert(uri, values);
cursor = resolver.query(uri, null, null, null, null);
```

Quelques classes d'Android



Structure d'un projet

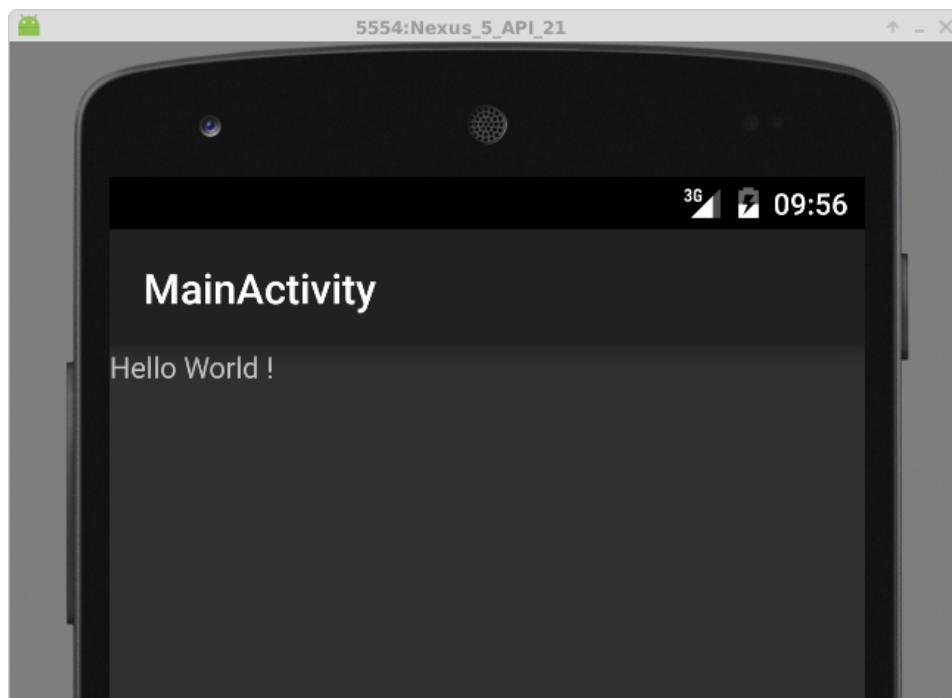


Manifeste d'une application

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.univ_amu.cci.com.alarm">
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <application android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name">
        <activity android:name="com.univ_amu.cci.alarm.views.AlarmActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:authorities="com.univ_amu.cci.alarm"
            android:name="com.univ_amu.cci.alarm.model.AlarmProvider" />
        <service android:name="com.univ_amu.cci.alarm.service.FireService"/>
        <receiver
            android:name="com.univ_amu.cci.alarm.service.UpdateBroadcastReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

Une activité :

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        RelativeLayout layout = new RelativeLayout(this);  
        TextView text = new TextView(this);  
        text.setText("Hello World !");  
        layout.addView(text);  
        setContentView(layout);  
    }  
}
```

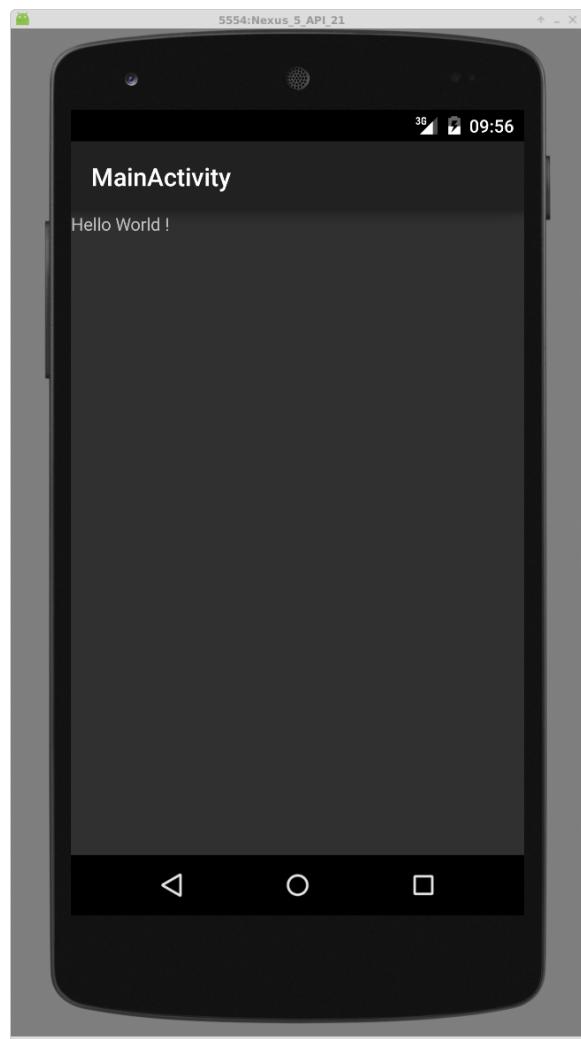




Déclaration de l'activité dans le manifeste

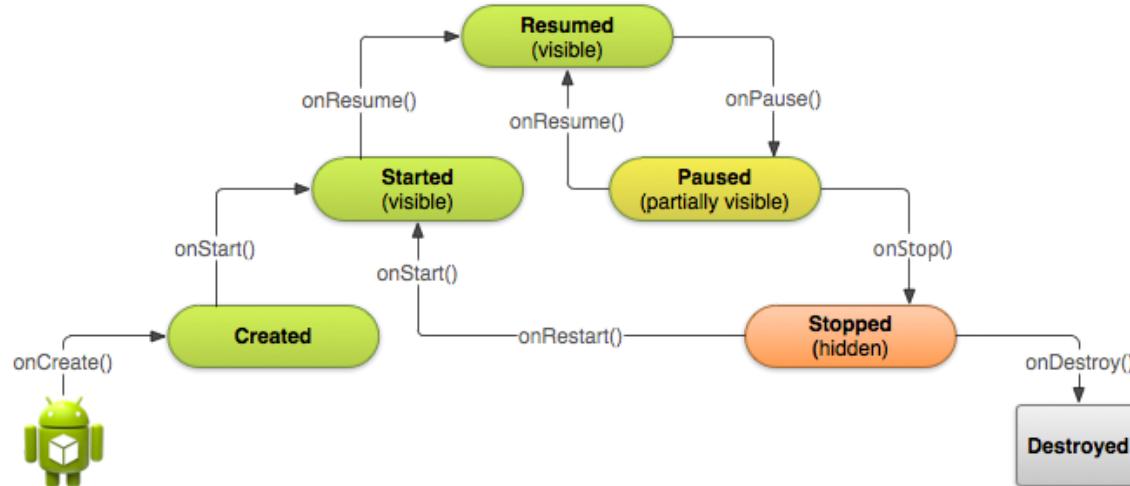
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cci.calculator" >
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Le résultat :



Cycle de vie d'une activité :

- Aucune méthode main dans un programme Android
- Android exécute le code d'une activité en appelant des *callbacks*
- Ces callbacks correspondent aux phrases de la vie d'une activité
- Il n'est pas nécessaire d'implémenter toutes les callbacks



États durables d'une activité

Resumed

L'activité est au premier plan et l'utilisateur peut interagir avec elle
On dit aussi qu'elle est en train d'être exécutée

Paused

L'activité est partiellement recouverte par une autre activité qui se trouve au premier plan

L'activité en pause ne peut pas recevoir d'action de l'utilisateur

Stopped

L'activité est totalement cachée et ne peut plus être exécutée de code
Toutes ses informations sont conservées

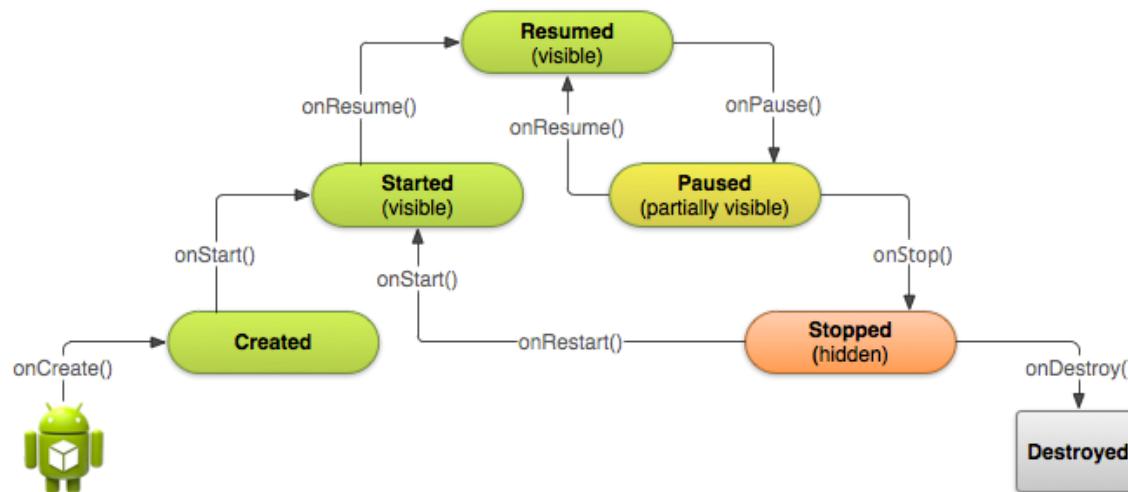
États transitoires d'une activité

Created

L'activité vient d'être créée.

Started

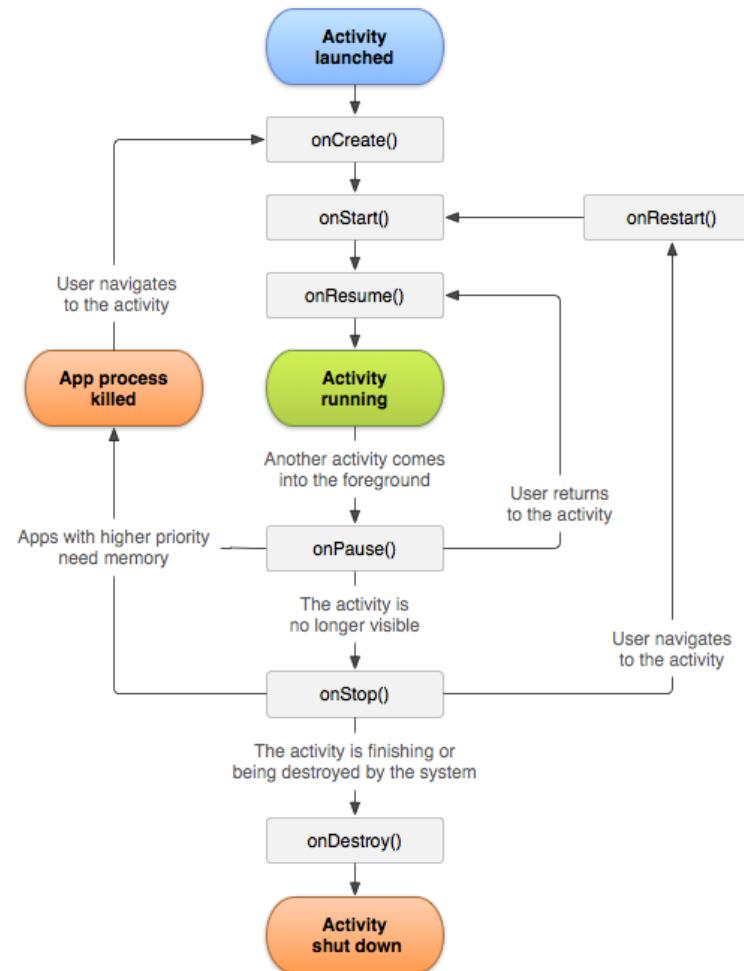
L'activité vient de devenir visible.



Les callbacks

```
public class ExampleActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) { /* ... */ }  
  
    @Override  
    protected void onStart() { /* ... */ }  
  
    @Override  
    protected void onResume() { /* ... */ }  
  
    @Override  
    protected void onPause() { /* .... */ }  
  
    @Override  
    protected void onStop() { /* ... */ }  
  
    @Override  
    protected void onDestroy() { /* ... */ }  
}
```

Cycle de vie des activités



Work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License

Mise en pause et reprise de l'activité

```
public class MainActivity extends Activity {

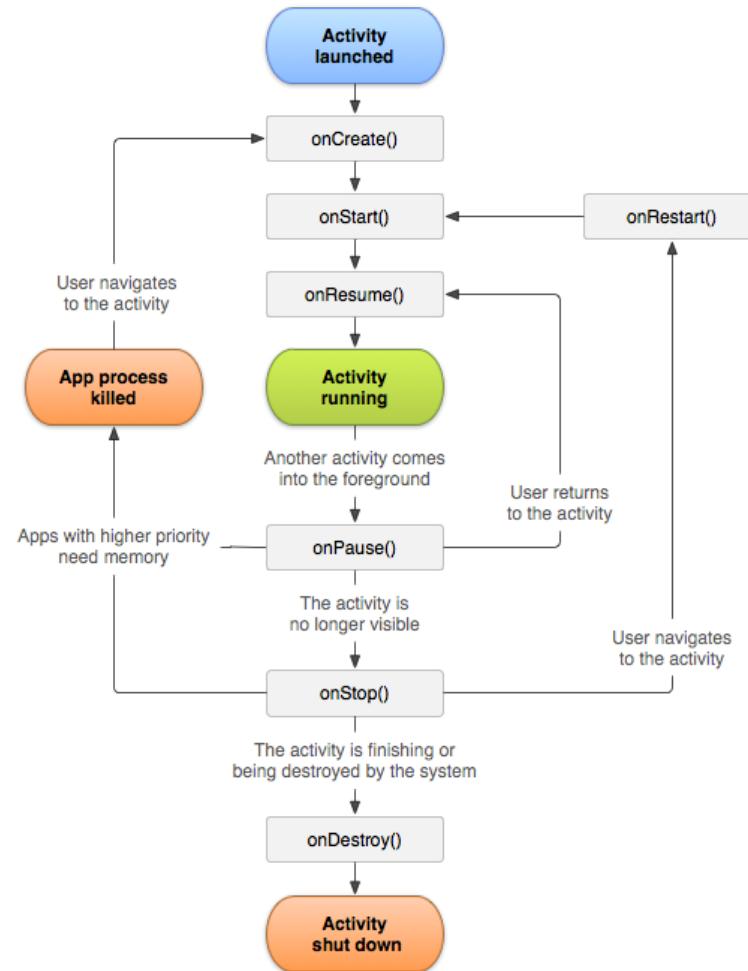
    @Override
    protected void onResume() {
        super.onResume();
        openCamera();
    }

    @Override
    protected void onPause() {
        super.onPause();
        releaseCamera();
    }

}
```

Sauvegarde de l'état de l'application

L'application est détruite dans les états **rouges** :



Sauvegarde de l'état de l'application

```
public class MainActivity extends Activity {
    private int value = 0;
    private Button button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        button = new Button(this);
        button.setOnClickListener(new OnClickListener());
        updateButtonLabel();
        layout.addView(button);
        setContentView(layout);
    }

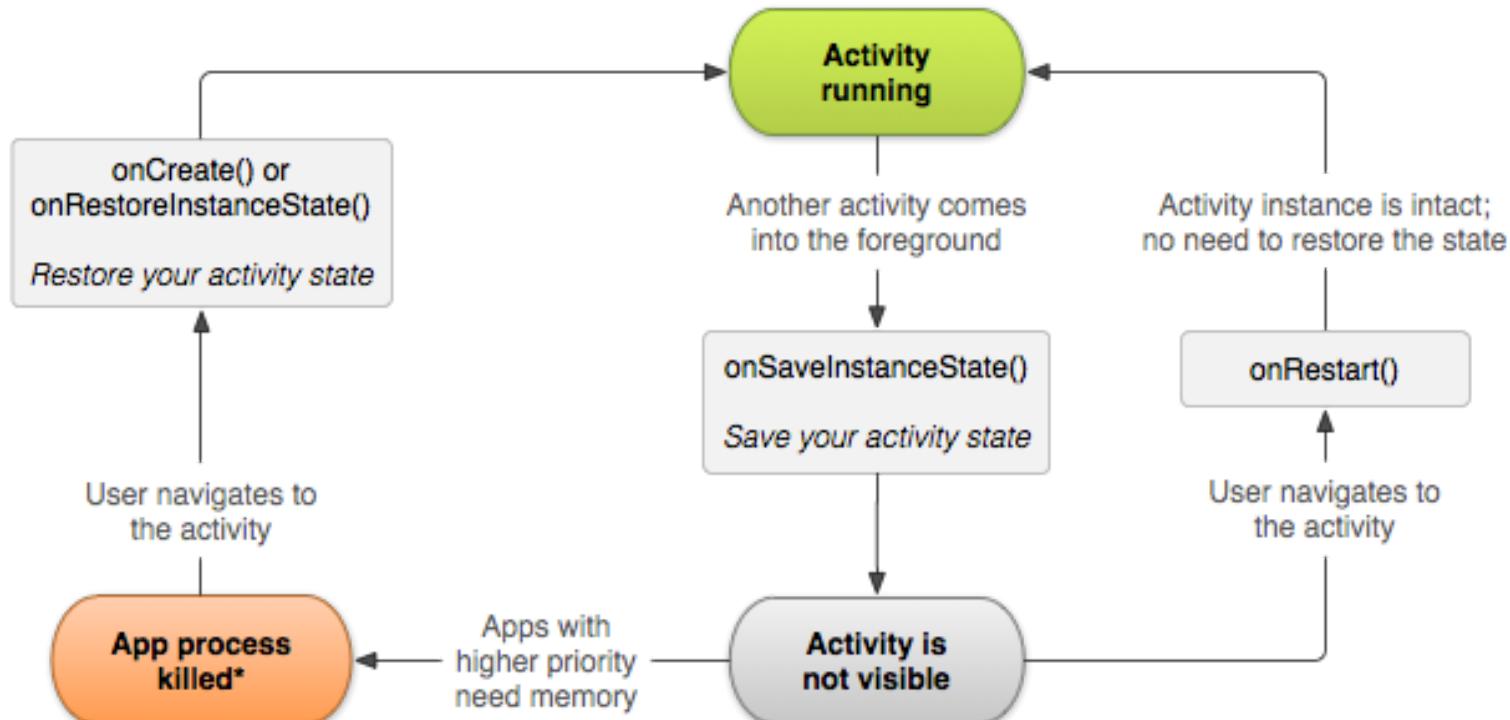
    public void updateButtonLabel() { button.setText(""+value); }

    private class OnClickListener implements View.OnClickListener {
        @Override
        public void onClick(View v) { value++; updateButtonLabel(); }
    }
}
```

Sauvegarde de l'état de l'application



Sauvegarde de l'état de l'application



Work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License

Sauvegarde de l'état de l'application

```
public class MainActivity extends Activity {  
    /* ... */  
    @Override  
    protected  
    void onRestoreInstanceState(Bundle savedInstanceState) {  
        super.onRestoreInstanceState(savedInstanceState);  
        value = savedInstanceState.getInt("value");  
        updateButtonLabel();  
    }  
  
    @Override  
    protected void onSaveInstanceState(Bundle outState) {  
        super.onSaveInstanceState(outState);  
        outState.putInt("value", value);  
    }  
}
```

Les Bundles

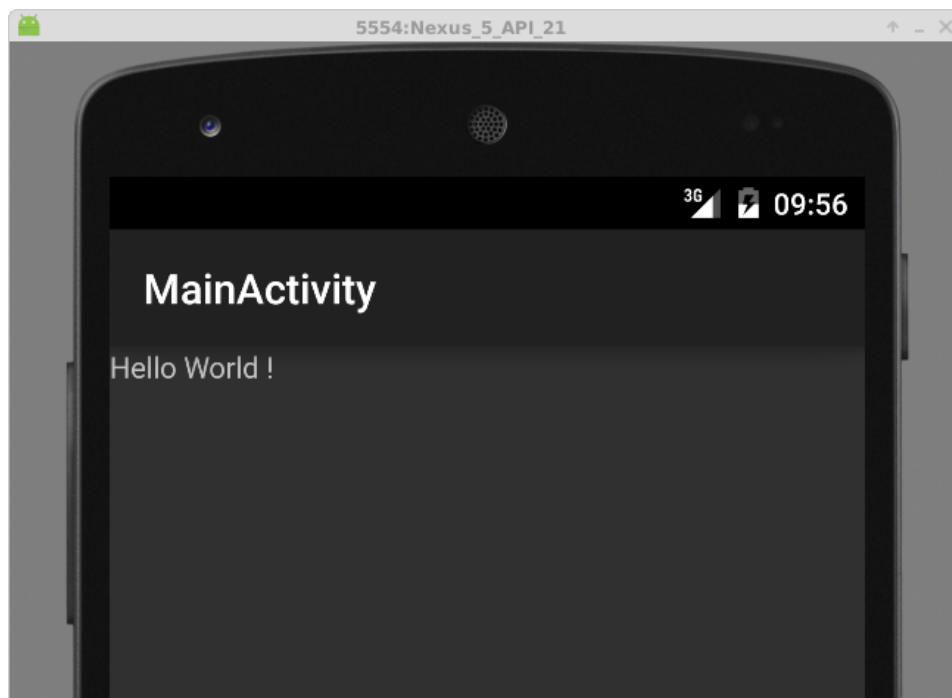
- Les Bundles peuvent associer des chaînes de caractères à des :
 - des types simples : entiers, flottants, caractères...
 - des chaînes de caractères
 - des tableaux et des listes
 - des objets Parcelable
- Quelques remarques :
 - Un Bundle est Parcelable
 - Un objet Parcelable peut être écrit dans un objet Parcel
 - Un Parcel est un conteneur pour un message

Quelques méthodes de la classe Bundle

- `.putInt(String key, int value)`
- `int getInt(String key)`
- `putDouble(String key, double value)`
- `double getDouble(String key)`
- `putString(String key, String value)`
- `String getString(String key)`
- `putStringArrayList(String key, ArrayList<String> list)`
- `ArrayList<String> getStringArrayList(String key)`
- `putCharArray(String key, char[] array)`
- `char[] getCharArray(String key)`

Les vues

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        RelativeLayout layout = new RelativeLayout(this);  
        TextView text = new TextView(this);  
        text.setText("Hello World !");  
        layout.addView(text);  
        setContentView(layout);  
    }  
}
```





Les classes View et ViewGroup

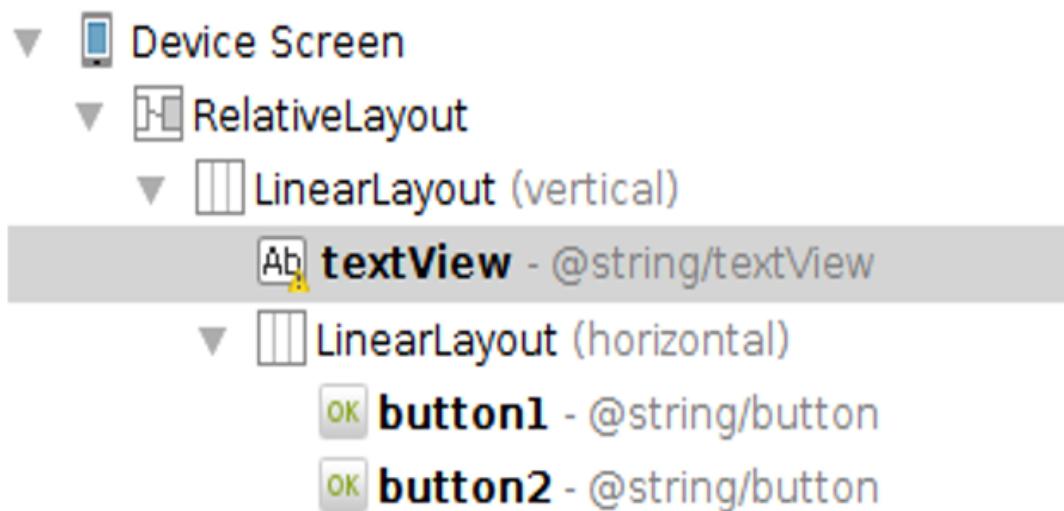
```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        RelativeLayout layout = new RelativeLayout(this);  
        TextView text = new TextView(this);  
        text.setText("Hello World !");  
        layout.addView(text);  
        setContentView(layout);  
    }  
}
```

- Dans cette exemple :
 - `RelativeLayout` étend `ViewGroup`
 - `ViewGroup` étend `View`
 - `TextView` étend `View`

Les classes View et ViewGroup

- Une vue est une classe qui étend View
- Une vue peut être dessinée à l'écran
- Un groupe de vue étend ViewGroup et donc View
- Un groupe de vues peut contenir d'autres vues
- Un groupe de vues organise l'affichage des vues qu'il contient
- setContentView de Activity pour préciser la vue à afficher

Hiérarchie des vues



Construction d'une vue en Java

```
public class MainActivity extends Activity {  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        RelativeLayout relativeLayout = createRelativeLayout();  
        setContentView(relativeLayout);  
    }  
  
    private RelativeLayout createRelativeLayout() {  
        RelativeLayout relativeLayout = new RelativeLayout(this);  
        relativeLayout.setPadding(16, 16, 16, 16);  
        relativeLayout.addView(createVerticalLayout());  
        return relativeLayout;  
    }  
  
    private LinearLayout createVerticalLayout() {  
        LinearLayout verticalLayout = new LinearLayout(this);  
        verticalLayout.setOrientation(LinearLayout.VERTICAL);  
        verticalLayout.setLayoutParams(  
            new RelativeLayout.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,  
                ViewGroup.LayoutParams.MATCH_PARENT));  
        verticalLayout.addView(createTextView());  
        verticalLayout.addView(createHorizontalLayout());  
        return verticalLayout;  
    }  
}
```

Construction d'une vue en Java

```
public class MainActivity extends Activity {  
    /* ... */  
    private TextView createTextView() {  
        TextView textView = new TextView(this);  
        textView.setLayoutParams(new LinearLayout.LayoutParams(  
            ViewGroup.LayoutParams.MATCH_PARENT, 0, 1.0f));  
        textView.setGravity(Gravity.CENTER);  
        textView.setTextSize(50);  
        textView.setText("TextView");  
        return textView;  
    }  
  
    private LinearLayout createHorizontalLayout() {  
        LinearLayout horizontalLayout = new LinearLayout(this);  
        horizontalLayout.setOrientation(LinearLayout.HORIZONTAL);  
        horizontalLayout.setLayoutParams(  
            new LinearLayout.LayoutParams(  
                ViewGroup.LayoutParams.MATCH_PARENT, 0, 1.0f));  
        horizontalLayout.addView(createButton());  
        horizontalLayout.addView(createButton());  
        return horizontalLayout;  
    }  
}
```

Construction d'une vue en Java

```
public class MainActivity extends Activity {  
    /* ... */  
    private Button createButton() {  
        Button button = new Button(this);  
        button.setLayoutParams(new LinearLayout.LayoutParams(  
            0, ViewGroup.LayoutParams.MATCH_PARENT, 1.0f));  
        button.setText("Button");  
        return button;  
    }  
}
```

Les évènements

On ajoute des écouteurs aux vues pour écouter les évènements :

```
public class MainActivity extends Activity {
    public int count = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Button button =new Button(this);
        button.setOnClickListener(new MyOnClickListener(this));
        setContentView(button);
    }
}
```

Les écouteurs

La méthode `setOnClickListener` prend en paramètre un objet qui implémente l'interface `View.OnClickListener` :

```
public interface OnClickListener {  
    void onClick(View view);  
}
```

- Une interface est associée à chaque type d'évènements
- On observe les évènements en connectant des listeners
- Pour cela, on utilise les méthodes `setOn*Listener`
- Ensuite, un évènement entraîne l'appel d'une méthode de l'interface sur l'objet que vous avez connecté à la vue

Implémentation avec une classe classique

```
class MyOnClickListener implements View.OnClickListener {
    private MainActivity mainActivity;

    public MyOnClickListener(MainActivity mainActivity) {
        this.mainActivity = mainActivity;
    }

    @Override
    public void onClick(View v) {
        mainActivity.count++;
    }
}
```

Implémentation avec une classe interne

```
public class MainActivity extends Activity {
    public int count = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* ... */
        button.setOnClickListener(new MyOnClickListener());
        /* ... */
    }

    class MyOnClickListener implements View.OnClickListener {
        @Override
        public void onClick(View v) { count++; }
    }
}
```

Implémentation avec une classe anonyme

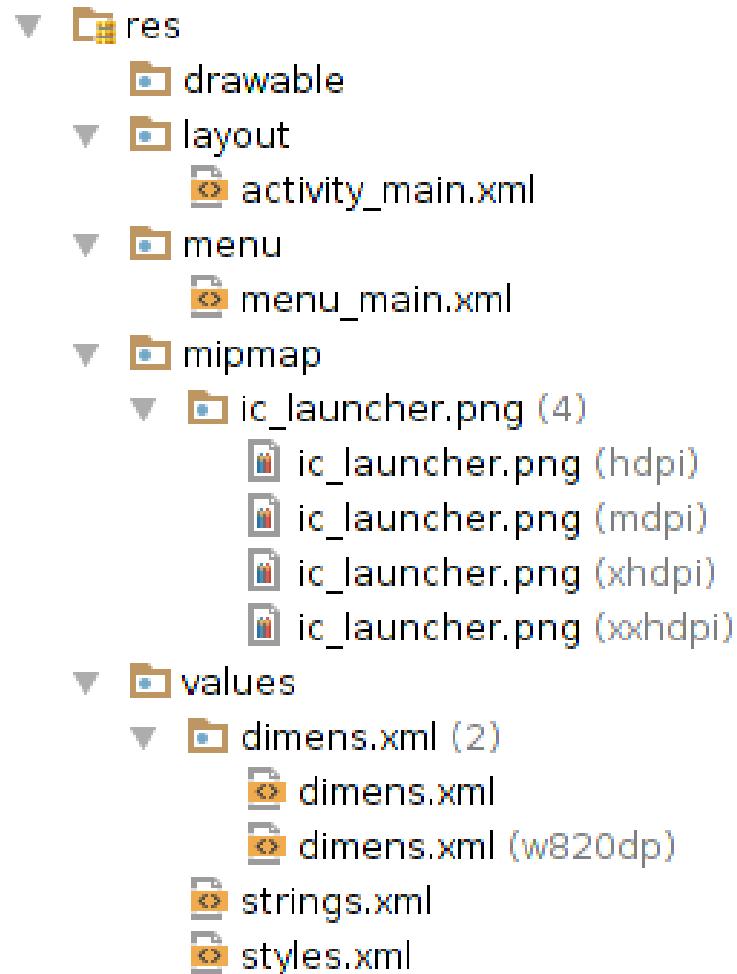
```
public class MainActivity extends Activity {
    public int count = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Button button =new Button(this);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { count++; }
        });
        setContentView(button);
    }
}
```

Les ressources

- elles sont en dehors du code et contiennent
 - les images
 - les chaînes de caractères
 - les modèles d'interfaces utilisateur
 - les animations...
- elles sont choisies en fonction de la configuration :
 - taille de l'écran
 - orientation de l'écran
 - langue...
- de maintenir ces éléments sans connaître Java.

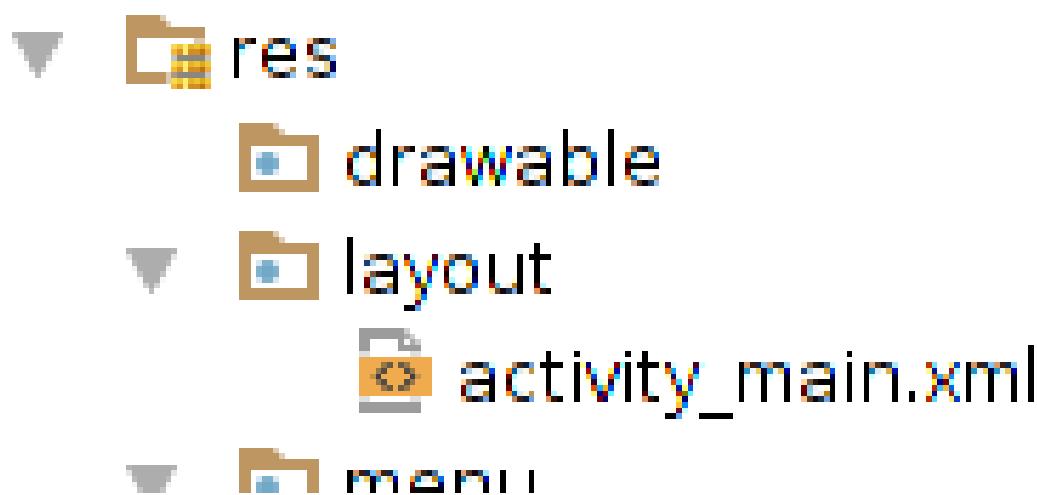
Les ressources



Elles sont dans les sous-répertoires du répertoire res

La classe R

- Elle est générée automatiquement par Android
- Elle permet de désigner les ressources dans le code Java
- Par exemple, `activity_main.xml = R.layout.activity_main`



Les layouts

La méthode `setContentView` peut prendre l'identifiant d'un layout :

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

Le contenu du fichier `activity_main.xml` :

```
<RelativeLayout  
    xmlns:android="..."  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" >  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:textSize="100dp"  
        android:text="Hello World !" />  
</RelativeLayout>
```

Les layouts

Le contenu du fichier `activity_main.xml` :

```
<RelativeLayout
    xmlns:android="..."
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="100dp"
        android:text="Hello World !" />
</RelativeLayout>
```



Les chaînes de caractères

Pour faciliter la traduction des applications, il est préférable d'utiliser des références à des chaînes de caractères dans les programmes ou les layouts :

```
<RelativeLayout
    xmlns:android="..."
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

Les chaînes de caractères

Elles se trouvent dans le fichier `strings.xml` :

```
<resources>
    <string name="hello_world">Hello world!</string>
</resources>
```

Pour accéder aux chaînes de caractères en XML :

```
<RelativeLayout ...>
    <TextView ...
        android:text="@string/hello_world" />
</RelativeLayout>
```

Les chaînes de caractères

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        RelativeLayout layout = new RelativeLayout(this);  
        TextView text = new TextView(this);  
        Context context = this;  
        text.setText(R.string.hello_world);  
        /* String helloWorld =  
           context.getResources().getText(R.string.hello_world);  
        text.setText(helloWorld); */  
        layout.addView(text);  
        setContentView(layout);  
    }  
}
```

Les références

Syntaxe pour désigner une ressource dans un document XML

```
@[<package_name>:]<resource_type>/<resource_name>
```

où

- `<package_name>` est le nom du paquet
- `<resource_type>` est le type de la ressource
- `<resource_name>` est
 - le nom du fichier qui contient la ressource
 - ou la valeur de l'attribut `android:name` d'un élément XML

```
<ImageView  
    android:contentDescription="@android:string/ok"  
    android:src="@drawable/ic_launcher" />
```

Les dimensions

```
<RelativeLayout xmlns:android="..."  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin"  
    android:paddingBottom="@dimen/activity_vertical_margin">  
</RelativeLayout>
```

Les dimensions sont définies dans le fichier `dimens.xml` :

```
<resources>  
    <dimen name="activity_horizontal_margin">16dp</dimen>  
    <dimen name="activity_vertical_margin">16dp</dimen>  
</resources>
```

Il est également possible d'accéder aux dimensions en Java :

```
float dimension = getResources().getDimension(  
        R.dimen.activity_horizontal_margin);
```

Prise en charge des différentes configurations

Il est possible de définir des ressources en fonction de :

- l'orientation du terminal
- la résolution
- la taille de l'écran
- la version de l'API
- de la langue, etc.

Par exemple :

- `values/dimens.xml` : dimensions par défaut
- `values-w820dp/dimens.xml` : dim. pour des écrans de 820p
- `layout` : les layouts par défaut
- `layout-land` : les versions *paysage* des layouts
- `values-fr/strings.xml` : chaînes en français

Les images

- Il est possible de stocker des images dans les ressources
- Les images peuvent être stockées sous différentes résolutions
- La résolution est choisie en fonction du terminal de l'utilisateur



Les images

Avec un `ic_launcher.png` dans les sous-répertoires `drawable-*` :

```
<ImageView  
    android:contentDescription="@string/hello_world"  
    android:src="@drawable/ic_launcher"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

```
<ImageButton  
    android:id="@+id/button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:contentDescription="@string/count"  
    android:src="@drawable/ic_launcher"  
    android:onClick="count" />
```

Les images

On désigne une image par son identifiant :

```
public class MainActivity extends Activity {  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        getWindow().setBackgroundDrawableResource(  
            R.drawable.ic_launcher);  
        ImageView image = (ImageView) findViewById(R.id.image);  
        image.setImageResource(R.drawable.ic_launcher);  
    }  
  
}
```

Les images

un Drawable décrit par un fichier XML :

```
<ripple xmlns:android="..."  
        android:color="?android:colorControlHighlight">  
    <item>  
        <shape android:shape="oval">  
            <solid android:color="?android:colorAccent"/>  
        </shape>  
    </item>  
</ripple>
```

qui change avec l'état de l'élément sur lequel il est utilisé :

```
<selector xmlns:android="...">  
    <item  
        android:state_selected="true"  
        android:drawable="@drawable/selected_note_background"/>  
    <item  
        android:drawable="@drawable/normal_note_background"/>  
</selector>
```

Les couleurs

Contenu du fichier `res/color/button_text.xml` :

```
<selector
    xmlns:android="...">
    <item android:state_pressed="true"
        android:color="#ffff0000"/> <!-- pressed -->
    <item android:state_focused="true"
        android:color="#ff0000ff"/> <!-- focused -->
    <item android:color="#ff000000"/> <!-- default -->
</selector>
```

Exemple d'utilisation :

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:textColor="@color/button_text" />
```

Les animations

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true">
        <objectAnimator
            android:propertyName="translationZ"
            android:duration="@android:integer/config_shortAnimTime"
            android:valueFrom="@dimen/elevation_low"
            android:valueTo="@dimen/elevation_high"
            android:valueType="floatType"/>
    </item>
    <item>
        <objectAnimator
            android:propertyName="translationZ"
            android:duration="@android:integer/config_shortAnimTime"
            android:valueFrom="@dimen/elevation_high"
            android:valueTo="@dimen/elevation_low"
            android:valueType="floatType"/>
    </item>
</selector>
```

Les styles

Définition d'un style à partir d'un autre style :

```
<resources>
    <style name="AppTheme"
        parent="android:Theme.Material.Light.DarkActionBar">
        <item name="android:colorPrimary">
            @android:color/holo_blue_light
        </item>
        <item name="android:colorPrimaryDark">
            @android:color/holo_blue_dark
        </item>
        <item name="android:textColor">
            @android:color/black
        </item>
        <!-- ... -->
    </style>
</resources>
```

Les styles

Utilisation du style dans le manifeste de l'application :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.univ_amu.cci.notes" >
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Les identifiants

Association des identifiants aux membres d'un layout :

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</RelativeLayout>
```

Accès aux éléments d'un layout avec son identifiant :

```
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView text = (TextView) findViewById(R.id.text);
        text.setText(R.string.hello_world);
    }
}
```

Les fragments

Un fragment :

- représente une portion d'interface utilisateur
- peut être inséré dans une activité
- peut être utilisé dans plusieurs activités
- possède son propre cycle de vie, layout, etc.

Une activité :

- peut afficher plusieurs fragments
- peut utiliser une pile pour gérer la navigation entre fragments

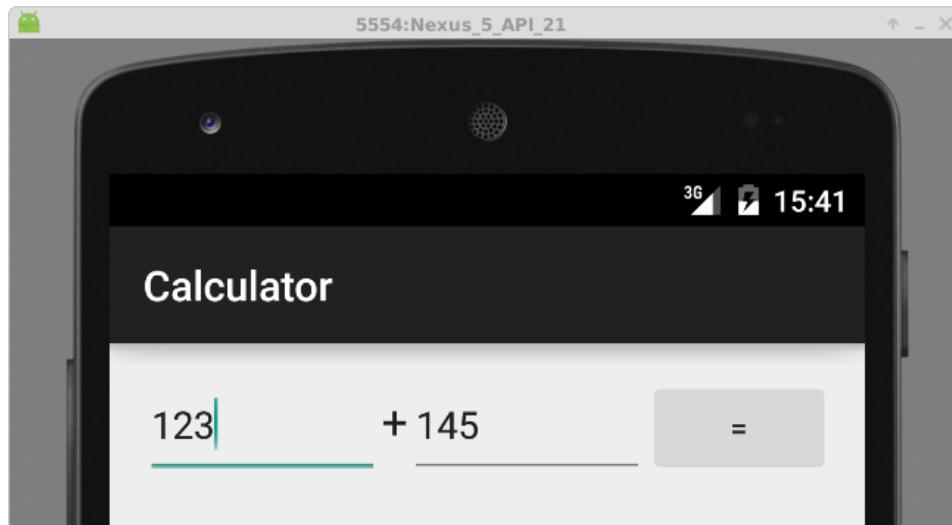
Une application avec deux fragments



Premier layout : le formulaire

Le layout correspondant au formulaire du premier écran :

```
<LinearLayout xmlns:android="...">
    <EditText
        android:id="@+id/editText1"
        android:inputType="number" />
    <TextView android:text="@string/plus" />
    <EditText
        android:id="@+id/editText2"
        android:inputType="number" />
    <Button
        android:id="@+id/button"
        android:text="@string>equals" />
</LinearLayout>
```





Deuxième layout : le résultat

```
<TextView xmlns:android="..."  
    xmlns:tools="..."  
    android:id="@+id/textView"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:gravity="center"  
    android:textSize="100dp"  
    android:text="+" />
```



Le layout de l'activité principale

Le layout de l'activité principale :

```
<FrameLayout xmlns:android="..."  
    xmlns:tools="..."  
    android:id="@+id/container"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```

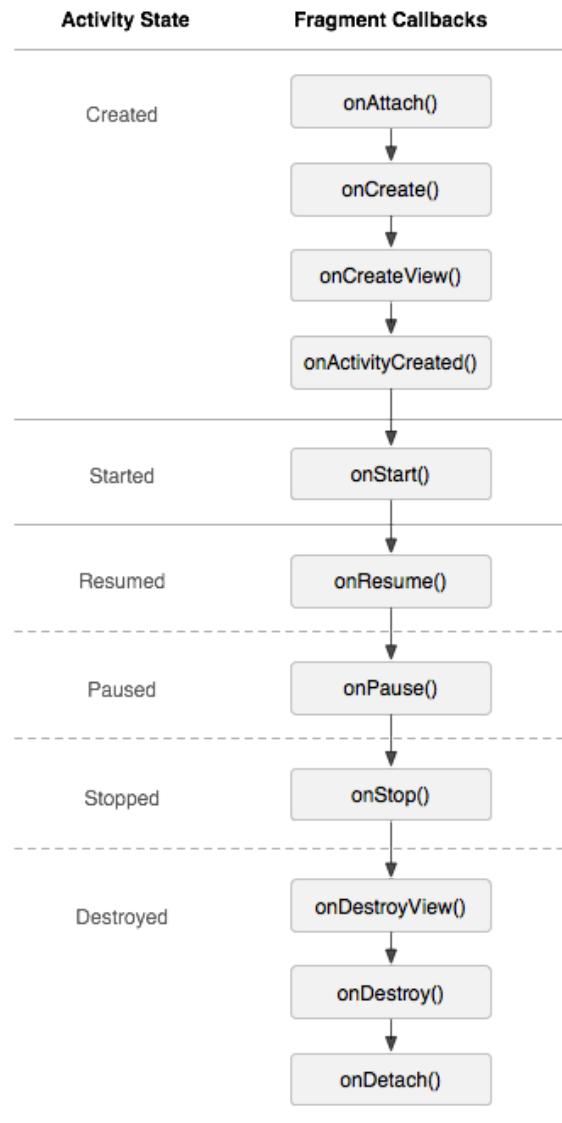
Le code de l'activité principale :

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        if (savedInstanceState == null) {  
            getFragmentManager().beginTransaction()  
                .add(R.id.container, new FormFragment())  
                .commit();  
        }  
    }  
}
```

Création de la vue dans le premier fragment

```
public class FormFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
                             ViewGroup container,  
                             Bundle savedInstanceState) {  
        View view = inflater.inflate(R.layout.fragment_form,  
                                    container,  
                                    false);  
        return view;  
    }  
}
```

Cycle de vie d'un fragment



Notification entre le fragment et l'activité

- FormFragment doit notifier l'activité de la validation du formulaire
- Interface pour rendre le fragment réutilisable :

```
public interface FormFragmentListener {  
    void onEquals(double value1, double value2);  
}
```

- L'activité va implémenter cette interface
- Lors de l'exécution de la méthode onAttach du fragment, nous allons conserver une référence afin de pouvoir notifier l'activité.

Notification entre le fragment et l'activité

Nous conservons la référence de l'activité :

```
public class FormFragment extends Fragment {
    private FormFragmentListener listener;

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            listener = (FormFragmentListener)activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                + " must implement FormFragmentListener");
        }
    }
}
```

Notification entre le fragment et l'activité

Nous faisons en sorte d'écouter les clics sur le bouton :

```
public class FormFragment extends Fragment {
    private EditText editText1, editText2;
    private FormFragmentListener listener;

    @Override
    public View onCreateView(LayoutInflater inflater,
                            ViewGroup container,
                            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_form,
                                    container, false);
        editText1 = (EditText)view.findViewById(R.id.editText1);
        editText2 = (EditText)view.findViewById(R.id.editText2);
        Button button = (Button)view.findViewById(R.id.button);
        button.setOnClickListener(new OnClickListener());
        return view;
    }
}
```

Notification entre le fragment et l'activité

Nous notifions l'activité si un clic se produit :

```
public class FormFragment extends Fragment {
    private EditText editText1, editText2;
    private FormFragmentListener listener;

    private class OnClickListener implements View.OnClickListener {
        @Override
        public void onClick(View v) {
            double value1 = Double.parseDouble(editText1.getText().toString());
            double value2 = Double.parseDouble(editText2.getText().toString());
            listener.onEquals(value1, value2);
        }
    }
}
```

Notification entre le fragment et l'activité

Réception de la notification par l'activité :

```
public class MainActivity extends Activity
    implements FormFragmentListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (savedInstanceState == null) {
            getFragmentManager().beginTransaction()
                .add(R.id.container, new FormFragment())
                .commit();
        }
    }

    @Override
    public void onEquals(double value1, double value2) {
        /* TODO : afficher le résultat */
    }
}
```

Le deuxième fragment

```
public class ResultFragment extends Fragment {  
  
    public static ResultFragment getInstance(double value) {  
        ResultFragment fragment = new ResultFragment();  
        Bundle bundle = new Bundle();  
        bundle.putDouble("value", value);  
        fragment.setArguments(bundle);  
        return fragment;  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        View view = inflater.inflate(R.layout.fragment_result,  
                                    container, false);  
        TextView textView = (TextView) view.findViewById(R.id.textView);  
        textView.setText("" + value());  
        return view;  
    }  
  
    public double value() {  
        return getArguments().getDouble("value");  
    }  
}
```

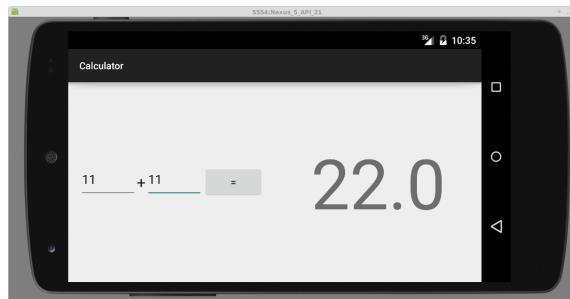
Affichage du deuxième fragment par l'activité

```
public class ResultFragment extends Fragment
    implements FormFragmentListener {

    @Override
    public void onEquals(double value1, double value2) {
        double value = value1+value2;
        getFragmentManager().beginTransaction()
            .replace(R.id.container,
                ResultFragment.getInstance(value))
            .addToBackStack("result")
            .commit();
    }

}
```

Affichage de deux fragments par l'activité



Affichage de deux fragments par l'activité

Le layout de l'activité par défaut :

```
<LinearLayout xmlns:android="..."  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <fragment class="com.univ_amu.cci.myapplication.FormFragment"  
        android:id="@+id/form"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"/>  
    <FrameLayout android:id="@+id/container"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:background="?android:attr/detailsElementBackground" />  
</LinearLayout>
```

Affichage de deux fragments par l'activité

Le layout de l'activité en mode paysage (répertoire layout-land) :

```
<LinearLayout xmlns:android="..."  
    android:orientation="horizontal"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <fragment class="com.univ_amu.cci.myapplication.FormFragment"  
        android:id="@+id/form"  
        android:layout_weight="1"  
        android:layout_width="0px"  
        android:layout_height="match_parent" />  
    <FrameLayout android:id="@+id/container" android:layout_weight="1"  
        android:layout_width="0px"  
        android:layout_height="match_parent"  
        android:background="?android:attr/detailsElementBackground" />  
</LinearLayout>
```

Affichage de deux fragments par l'activité

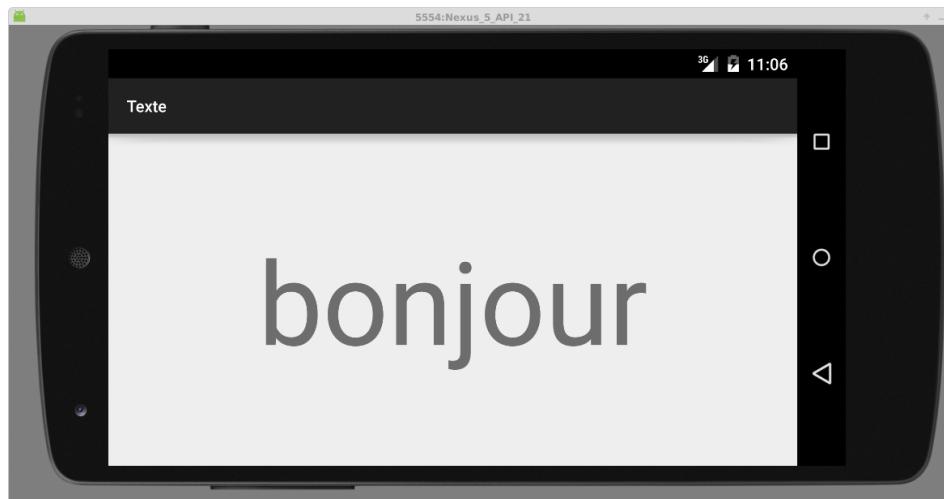
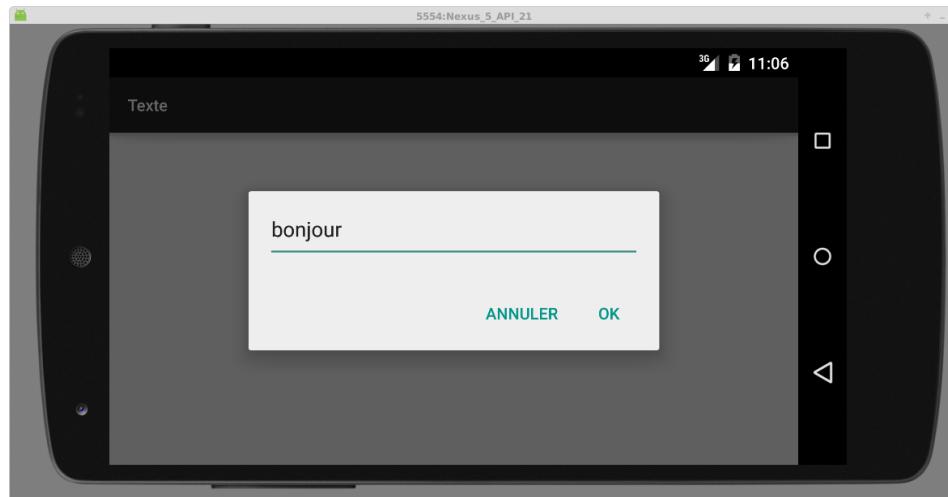
Le code de l'activité :

```
public class MainActivity extends Activity
    implements FormFragmentListener {

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

    @Override
    public void onEquals(double value1, double value2) {
        double value = value1+value2;
        getFragmentManager().beginTransaction()
            .replace(R.id.container, ResultFragment.getInstance(value))
            .addToBackStack("result")
            .commit();
    }
}
```

Les boîtes de dialogue



Les boites de dialogue

Les boites de dialogue sont des fragments :

```
public class DialogFragment extends android.app.DialogFragment {
    private EditText editText;

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        LayoutInflater inflater = getActivity().getLayoutInflater();
        View view = inflater.inflate(R.layout.dialog, null);
        editText = (EditText)view.findViewById(R.id.editText);
        builder.setView(view)
            .setPositiveButton(android.R.string.ok,
                new OnPositiveButtonClickListener())
            .setNegativeButton(android.R.string.cancel,
                new OnNegativeButtonClickListener());
        return builder.create();
    }
}
```

Les boites de dialogue

L'interface de communication :

```
public interface DialogFragmentListener {  
    void onChangeText(String text);  
}
```

On redéfinit ensuite la méthode onAttach :

```
public class DialogFragment extends android.app.DialogFragment {  
    private DialogFragmentListener listener;  
  
    @Override  
    public void onAttach(Activity activity) {  
        super.onAttach(activity);  
        try { listener = (DialogFragmentListener)activity; }  
        catch (ClassCastException e) {  
            throw new ClassCastException(activity.toString()  
                + " must implement DialogFragmentListener");  
        }  
    }  
}
```

Les boîtes de dialogue

Les écouteurs des événements :

```
public class DialogFragment extends android.app.DialogFragment {
    private class OnPositiveButtonClickListener
        implements DialogInterface.OnClickListener {
        @Override
        public void onClick(DialogInterface dialog, int id) {
            listener.onChangeText(editText.getText().toString());
            DialogFragment.this.getDialog().dismiss();
        }
    }

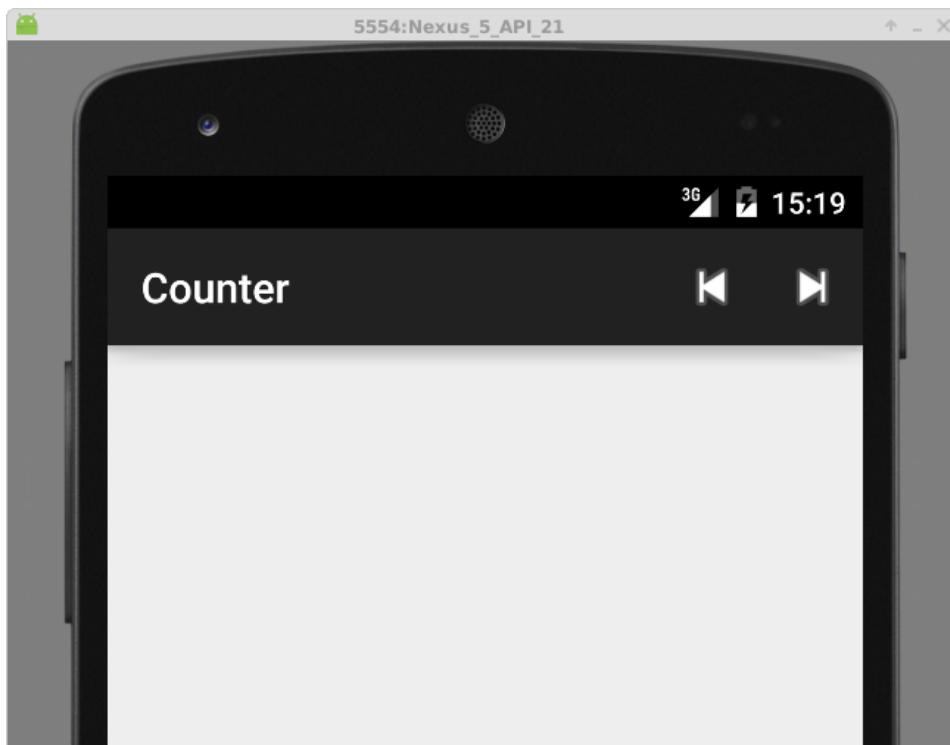
    private class OnNegativeButtonClickListener
        implements DialogInterface.OnClickListener {
        public void onClick(DialogInterface dialog, int id) {
            DialogFragment.this.getDialog().cancel();
        }
    }
}
```

Les boites de dialogue

```
public class MainActivity extends Activity implements DialogFragmentListener {  
    private TextView textView;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        textView = (TextView) findViewById(R.id.textView);  
        textView.setOnClickListener(new OnClickListener());  
    }  
  
    @Override  
    public void onChangeText(String text) { textView.setText(text); }  
  
    private class OnClickListener implements View.OnClickListener {  
        @Override  
        public void onClick(View v) {  
            DialogFragment newFragment = new DialogFragment();  
            newFragment.show(getFragmentManager(), "dialog");  
        }  
    }  
}
```

Les menus

```
<menu xmlns:android="....">
    <item android:id="@+id/action_decrement"
        android:title="@string/increment"
        android:icon="@drawable/previous"
        android:showAsAction="ifRoom" />
    <item android:id="@+id/action_increment"
        android:title="@string/decrement"
        android:icon="@drawable/next"
        android:showAsAction="ifRoom" />
</menu>
```



6



Chargement et traitement des actions du menu

```
public class MainActivity extends Activity {  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        getMenuInflater().inflate(R.menu.main, menu);  
        return true;  
    }  
  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item) {  
        switch (item.getItemId()) {  
            case R.id.action_decrement:  
                decrement();  
                return true;  
            case R.id.action_increment:  
                increment();  
                return true;  
            default:  
                return super.onOptionsItemSelected(item);  
        }  
    }  
    /* les méthodes decrement, increment et onCreate */  
}
```

Chargement et traitement des actions du menu

```
public class MainActivity extends Activity {
    /* ... */

    private TextView counterTextView;
    private int counter = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        counterTextView = (TextView) findViewById(R.id.counter);
        updateTextView();
    }

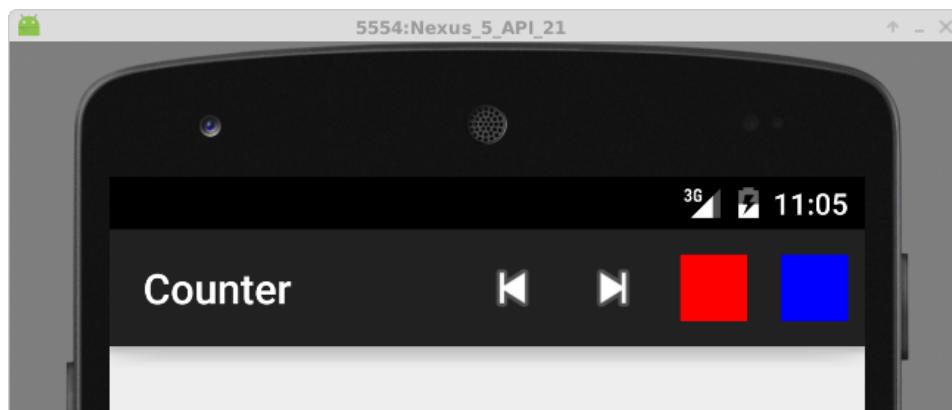
    private void updateTextView() { counterTextView.setText("" + counter); }
    private void increment() { counter++; updateTextView(); }
    private void decrement() { counter--; updateTextView(); }
}
```

Sauvegarde de l'état de l'activité

```
public class MainActivity extends Activity {  
    /* ... */  
  
    @Override  
    protected void onSaveInstanceState(Bundle outState) {  
        super.onSaveInstanceState(outState);  
        outState.putInt("counter", counter);  
    }  
  
    @Override  
    protected void onRestoreInstanceState(Bundle savedInstanceState) {  
        super.onRestoreInstanceState(savedInstanceState);  
        counter = savedInstanceState.getInt("counter");  
        updateTextView();  
    }  
}
```

Menus et fragments

```
<LinearLayout xmlns:android="..."  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical">  
  
<fragment  
    android:id="@+id/counterFragment"  
    android:name=".CounterFragment"  
    android:layout_weight="1"  
    android:layout_width="match_parent"  
    android:layout_height="0dp" />  
  
<fragment  
    android:id="@+id/colorFragment"  
    android:name=".ColorFragment"  
    android:layout_weight="1"  
    android:layout_width="match_parent"  
    android:layout_height="0dp" />  
  
</LinearLayout>
```



4



Menus et fragments



Menus et fragments

```
public class CounterFragment extends Fragment {
    private TextView counterTextView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_counter, container, false);
        counterTextView = (TextView) view.findViewById(R.id.counter);
        updateTextView();
        return view;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.menu_counter, menu);
    }
    /* ... */
}
```

Menus et fragments

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/action_decrement"
        android:title="@string/increment"
        android:icon="@android:drawable/ic_media_previous"
        android:showAsAction="always" />
    <item android:id="@+id/action_increment"
        android:title="@string/decrement"
        android:icon="@android:drawable/ic_media_next"
        android:showAsAction="always" />
</menu>
```

Menus et fragments

```
public class CounterFragment extends Fragment {
    private int counter = 0;

    /* ... */

    private void updateTextView() { counterTextView.setText("" + counter); }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_decrement: decrement(); return true;
            case R.id.action_increment: increment(); return true;
            default: return super.onOptionsItemSelected(item);
        }
    }

    private void increment() { counter++; updateTextView(); }
    private void decrement() { counter--; updateTextView(); }
}
```

- Tous les fragments peuvent recevoir les notifications
- Le premier qui retourne true arrête la propagation

Menus et fragments

```
public class ColorFragment extends Fragment {
    private ImageView imageView;
    private int color = Color.RED;

    @Override
    public View onCreateView(LayoutInflater inflater,
                           ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_color,
                                     container, false);
        imageView = (ImageView) view.findViewById(R.id.color);
        updateImageView();
        return view;
    }

    private void updateImageView() { imageView.setBackgroundColor(color); }
}
```

Menus et fragments

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/action_red"
        android:title="@string/red"
        android:icon="@drawable/ic_red"
        android:showAsAction="always" />
    <item android:id="@+id/action_blue"
        android:title="@string/blue"
        android:icon="@drawable/ic_blue"
        android:showAsAction="always" />
</menu>
```

Menus et fragments

```
public class ColorFragment extends Fragment {
    private ImageView imageView;
    private int color = Color.RED;

    /* ... */

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu,
                                MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.menu_color, menu);
    }
}
```

Menus et fragments

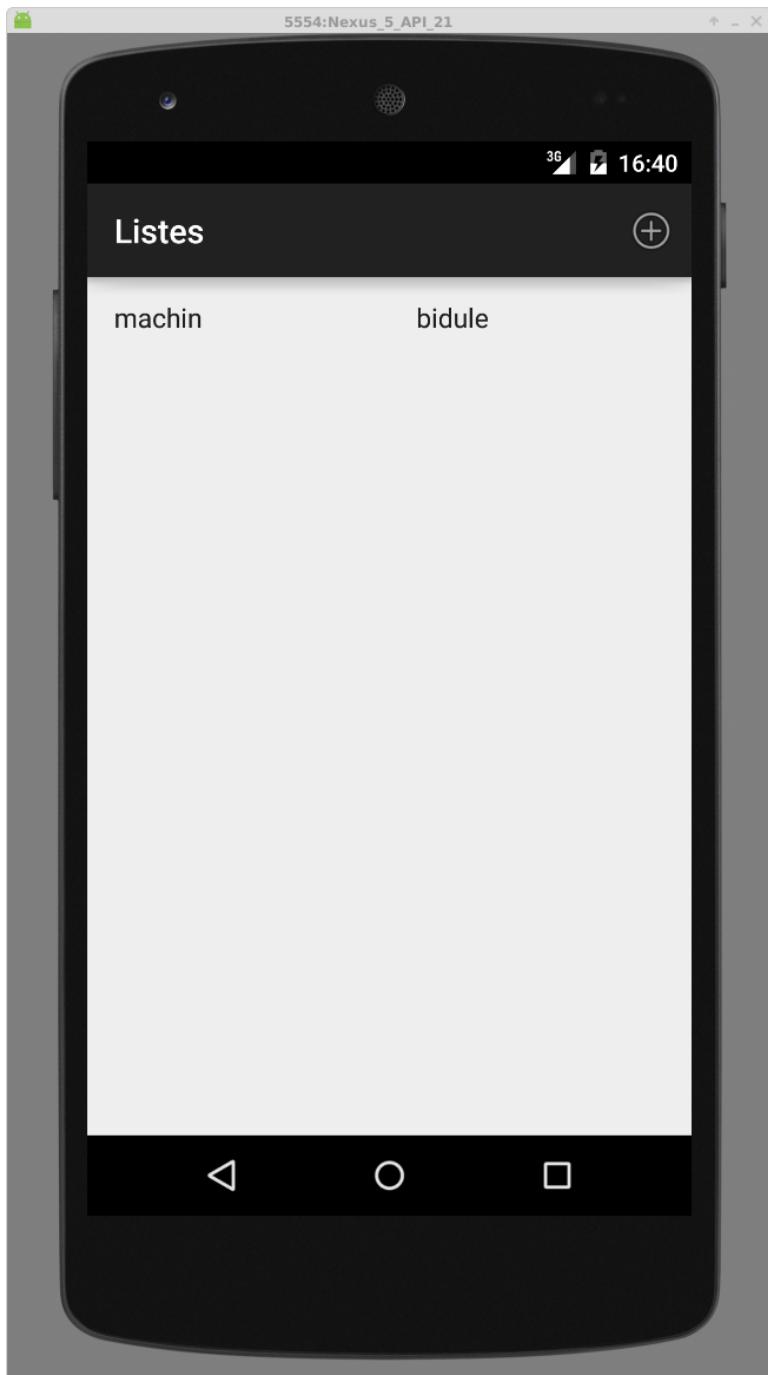
```
public class ColorFragment extends Fragment {

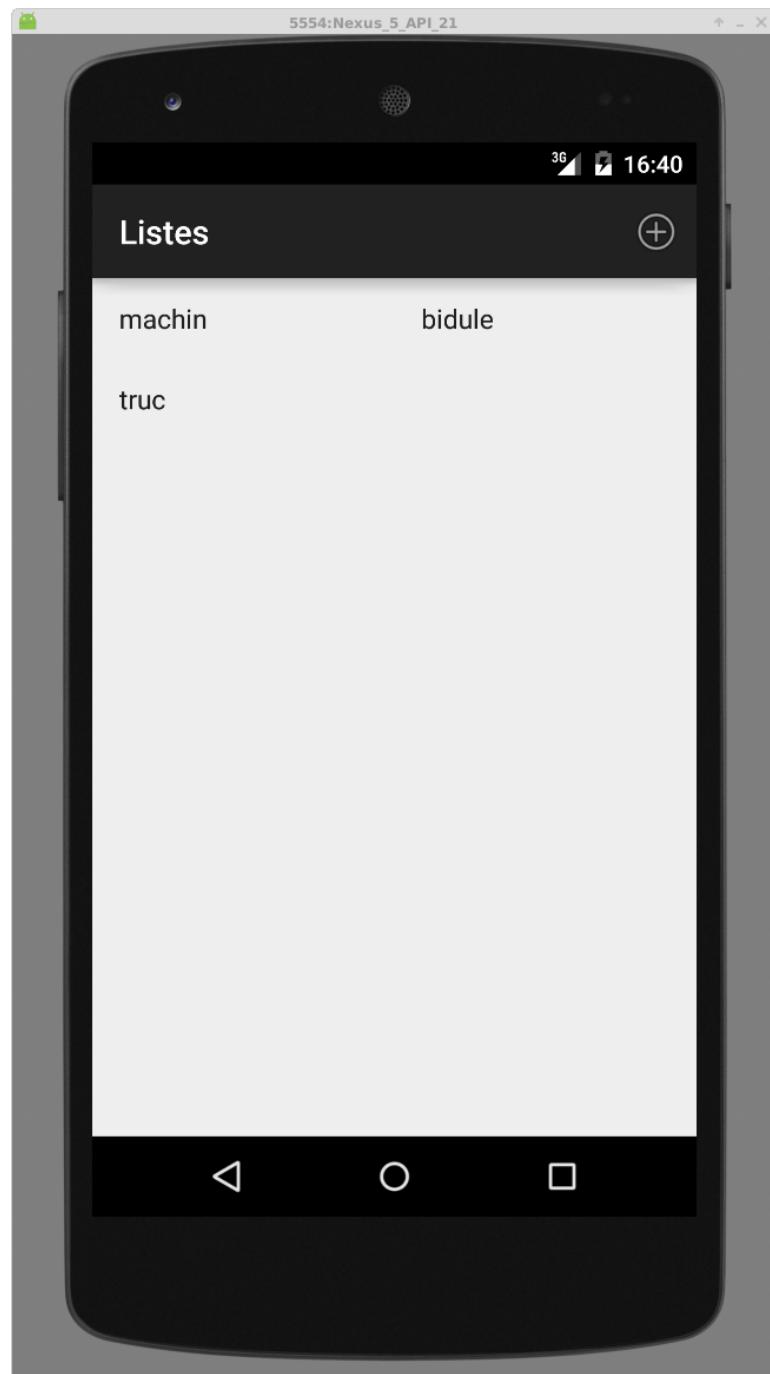
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_red:
                changeColor(Color.RED); return true;
            case R.id.action_blue:
                changeColor(Color.BLUE); return true;
            default: return super.onOptionsItemSelected(item);
        }
    }

    private void changeColor(int color) {
        this.color = color; updateImageView();
    }
}
```

Les listes

Dans le cours et les TP, nous allons utiliser des RecyclerView :

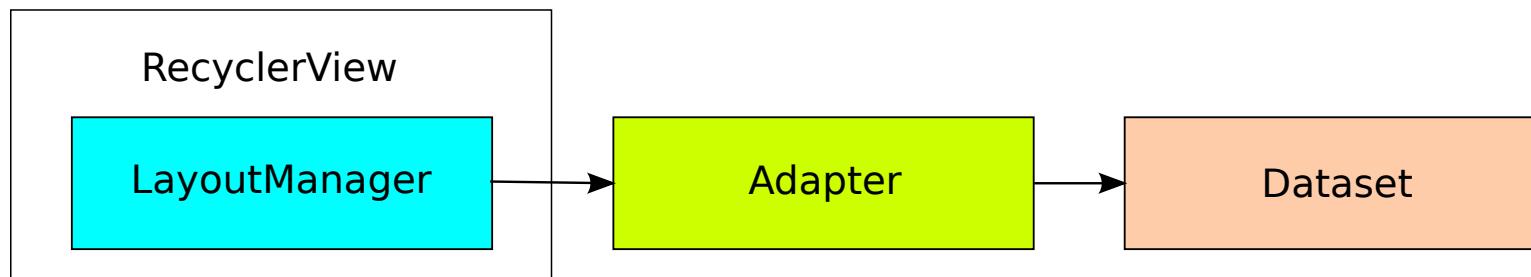




Il est également possible d'utiliser l'ancienne classe `ListView`.

Les listes

- Un RecyclerView permet l'affichage d'un grand nombre de vues
- Un LayoutManager permet de gérer la position des éléments
- Un Adapter va adapter les données aux vues



Le layout principal de l'application

```
<android.support.v7.widget.RecyclerView  
    xmlns:android="..."  
    android:id="@+id/recyclerView"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
/>
```

```
public class MainActivity extends Activity {  
    private ItemAdapter adapter;  
    private ArrayList<String> dataset;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recyclerView);  
        RecyclerView.LayoutManager layoutManager = new GridLayoutManager(this, 2);  
        recyclerView.setLayoutManager(layoutManager);  
        dataset = new ArrayList<>();  
        adapter = new ItemAdapter(dataset);  
        recyclerView.setAdapter(adapter);  
    }  
}
```

Code de l'adaptateur

```
public class ItemAdapter
    extends RecyclerView.Adapter<ItemAdapter.ViewHolder> {
    private final List<String> dataset;

    public ItemAdapter(List<String> dataset) { this.dataset = dataset; }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup group, int typeView) {
        View view = LayoutInflater.from(group.getContext())
            .inflate(android.R.layout.simple_list_item_1, group, false);
        ViewHolder viewHolder = new ViewHolder(view);
        return viewHolder;
    }

    public class ViewHolder extends RecyclerView.ViewHolder {
        final TextView textView;

        public ViewHolder(View itemView) {
            super(itemView);
            textView = (TextView) itemView.findViewById(android.R.id.text1);
        }

        public void bind(int index) { textView.setText(dataset.get(index)); }
    }
}
```

Code de l'adaptateur

```
public class ItemAdapter
    extends RecyclerView.Adapter<ItemAdapter.ViewHolder> {
    /* ... */

    @Override
    public void onBindViewHolder(ViewHolder viewHolder, int index) {
        viewHolder.bind(index);
    }

    @Override
    public int getItemCount() { return dataset.size(); }

}
```

Mise en place du dialogue d'ajout d'un item

Interface de communication :

```
public interface DialogFragmentListener {  
    void onAddItem(String text);  
}
```

L'activité implémente cette interface :

```
public class MainActivity extends Activity  
    implements DialogFragmentListener {  
  
    @Override  
    public void onAddItem(String text) {  
        /* TODO */  
    }  
}
```

Notification de l'adaptateur de la modification du modèle

Modification des données et notification de l'adaptateur :

```
public class MainActivity extends Activity
    implements DialogFragmentListener {

    private ItemAdapter adapter;
    private ArrayList<String> dataset;

    @Override
    public void onAddItem(String text) {
        dataset.add(text);
        adapter.notifyDataSetChanged();
    }
}
```

Il est possible de notifier l'adaptateur plus finement avec les méthodes `notifyItemChanged`, `notifyItemInserted`, `notifyItemMoved`, `notifyItemRangeChanged`, etc.

Ajout d'un menu à l'activité

```
public class MainActivity extends Activity
    implements DialogFragmentListener {

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if (item.getItemId() == R.id.action_add) {
            DialogFragment newFragment = new DialogFragment();
            newFragment.show(getFragmentManager(), "dialog");
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Code de la boîte de dialogue

```
public class DialogFragment extends android.app.DialogFragment {
    private EditText editText;

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder =
            new AlertDialog.Builder(getActivity());
        LayoutInflater inflater = getActivity().getLayoutInflater();
        View view = inflater.inflate(R.layout.dialog, null);
        editText = (EditText)view.findViewById(R.id.editText);
        builder.setView(view)
            .setPositiveButton(android.R.string.ok,
                new OnPositiveButtonClickListener())
            .setNegativeButton(android.R.string.cancel,
                new OnNegativeButtonClickListener());
        return builder.create();
    }
}
```

Code de la boîte de dialogue

```
<RelativeLayout xmlns:android="..."  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin">  
    <EditText  
        android:id="@+id/editText"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:ems="10" />  
</RelativeLayout>
```

Code de la boîte de dialogue

```
public class DialogFragment extends android.app.DialogFragment {
    private EditText editText;
    private DialogFragmentListener listener;

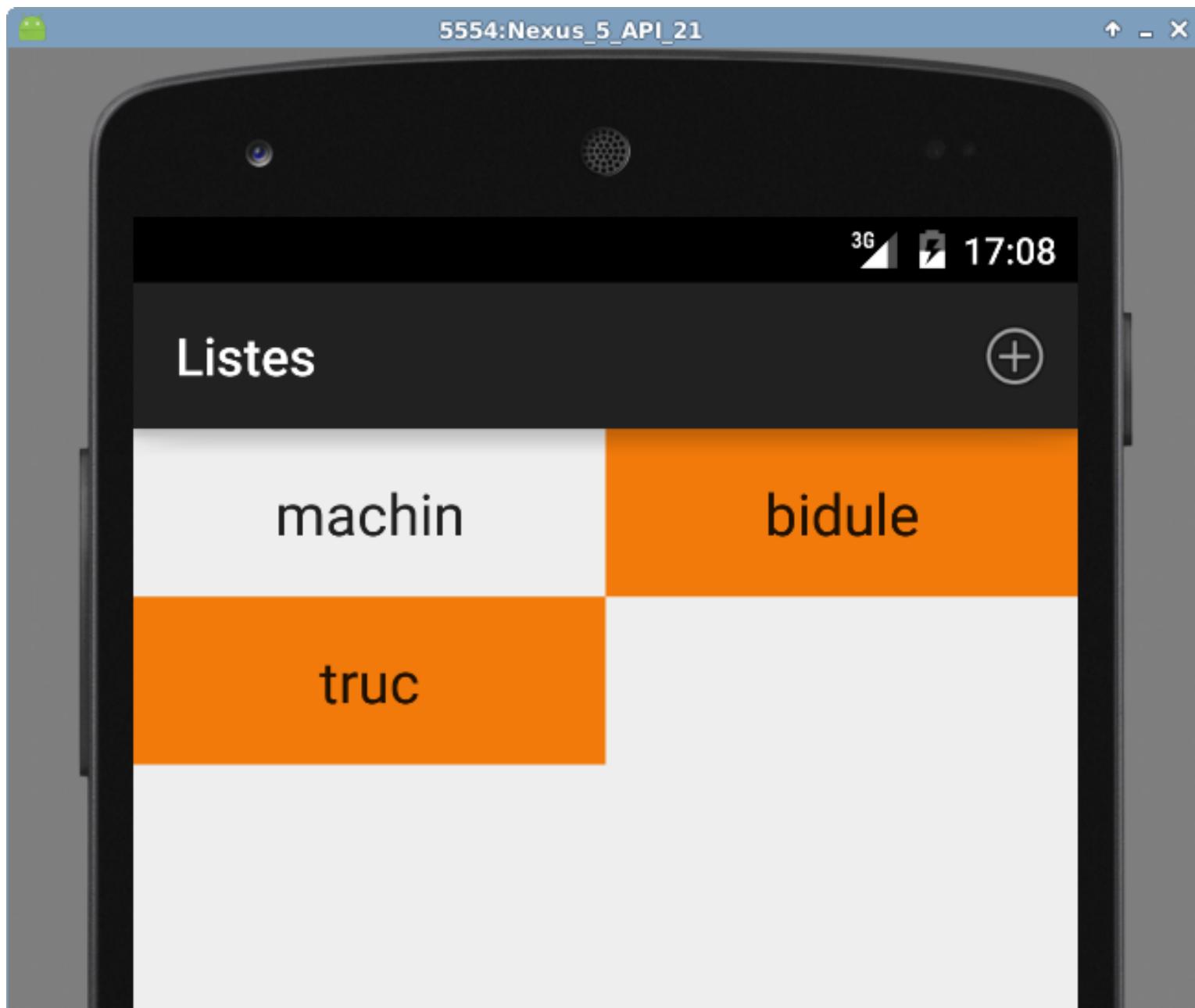
    /* ... */

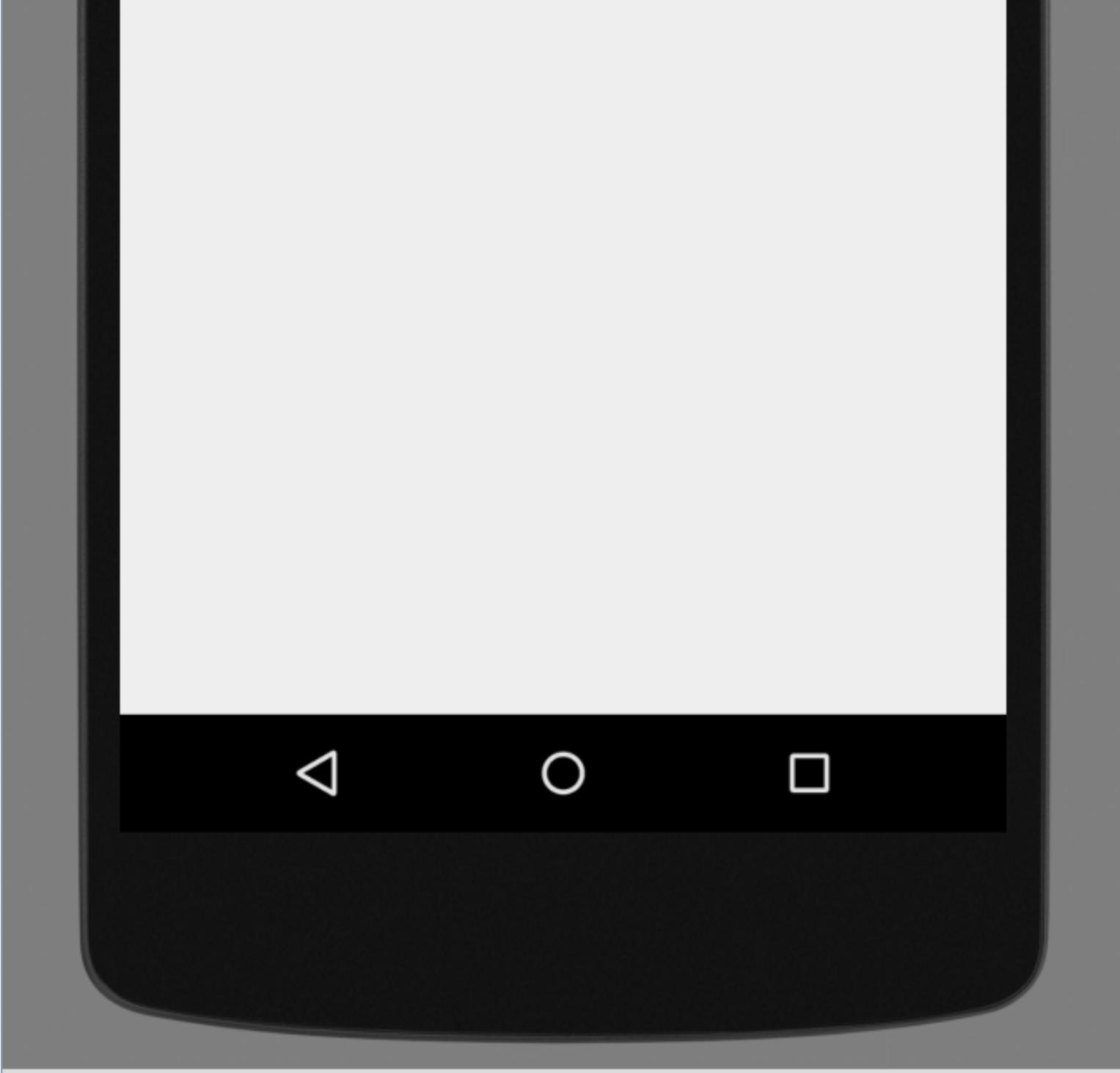
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            listener = (DialogFragmentListener)activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                + " must implement DialogFragmentListener");
        }
    }
}
```

Code de la boîte de dialogue

```
public class DialogFragment extends android.app.DialogFragment {  
    /* ... */  
    private class OnPositiveButtonClickListener  
        implements DialogInterface.OnClickListener {  
        @Override  
        public void onClick(DialogInterface dialog, int id) {  
            listener.onAddItem(editText.getText().toString());  
            DialogFragment.this.getDialog().dismiss();  
        }  
    }  
  
    private class OnNegativeButtonClickListener  
        implements DialogInterface.OnClickListener {  
        public void onClick(DialogInterface dialog, int id) {  
            DialogFragment.this.getDialog().cancel();  
        }  
    }  
}
```

Sélection des éléments





Sélection des éléments

Notification de l'activité lorsque l'utilisateur clique sur un élément :

```
public class ItemAdapter
    extends RecyclerView.Adapter<ItemAdapter.ViewHolder> {
    private List<OnItemClick> listeners = new ArrayList<>();

    public interface OnItemClick { void onClick(int index); }

    public void setOnItemClick(OnItemClick listener) {
        listeners.add(listener);
    }

    public void notifyOnItemClick(int index) {
        for (OnItemClick listener : listeners)
            listener.onClick(index);
    }
}
```

Sélection des éléments

Notification de l'activité lorsque l'utilisateur clique sur un élément :

```
public class ItemAdapter
    extends RecyclerView.Adapter<ItemAdapter.ViewHolder> {
    public class ViewHolder extends RecyclerView.ViewHolder {
        int index;

        public ViewHolder(View itemView) {
            /* ... */ itemView.setOnClickListener(new OnClickListener());
        }

        public void bind(int index) { this.index = index; /* ... */ }

        private class OnClickListener implements View.OnClickListener {
            @Override
            public void onClick(View view) { notifyOnItemClick(index); }
        }
    }
}
```

Sélection des éléments

Réception des notifications par l'activité :

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    private ItemAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* ... */
        adapter = new ItemAdapter(dataset, selectedItems);
        adapter.setOnItemClick(new OnItemClick());
    }

    private class OnItemClick implements ItemAdapter.OnItemClick {
        @Override
        public void onClick(int index) { /* TODO */ }
    }
}
```

Sélection des éléments

Méthodes pour gérer les sélections :

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    private SparseBooleanArray selectedItems = new SparseBooleanArray();
    private int numberOfSelectedItems;

    private void swapItemSelection(int index) {
        selectedItems.put(index, !selectedItems.get(index));
        adapter.notifyDataSetChanged();
        numberOfSelectedItems += selectedItems.get(index) ? 1 : -1;
    }

    private void clearSelection() {
        selectedItems.clear();
        numberOfSelectedItems = 0;
        adapter.notifyDataSetChanged();
    }
}
```

Sélection des éléments

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    private ItemAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* ... */
        adapter = new ItemAdapter(dataset, selectedItems);
        adapter.setOnItemClick(new OnItemClick());
    }

    private class OnItemClick implements ItemAdapter.OnItemClick {
        @Override
        public void onClick(int index) { swapItemSelection(index); }
    }
}
```

Sélection des éléments

item_background.xml :

```
<selector xmlns:android="...">
    <item android:state_selected="true"
          android:drawable="@drawable/selected_item_background" />
    <item android:drawable="@drawable/normal_item_background"/>
</selector>
```

selected_item_background.xml :

```
<ripple xmlns:android="..."
        android:color="?android:colorMultiSelectHighlight">
    <item android:id="@+id/mask"
          android:drawable="@color/white" />
</ripple>
```

Sélection des éléments

normal_item_background.xml :

```
<ripple xmlns:android="..."  
        android:color="@android:color/white">  
    <item android:drawable="?android:colorMultiSelectHighlight"/>  
</ripple>
```

layout_item.xml :

```
<RelativeLayout xmlns:android="..."  
        android:layout_width="fill_parent"  
        android:layout_height="fill_parent"  
        android:background="@drawable/item_background">  
    <TextView  
        android:id="@+id/text"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent" />  
</RelativeLayout>
```

Sélection des éléments

Communication des éléments sélectionnés à l'adaptateur :

```
public class ItemAdapter
    extends RecyclerView.Adapter<ItemAdapter.ViewHolder> {
    private final List<String> dataset;
    private final SparseBooleanArray selectedItems;

    public ItemAdapter(List<String> dataset,
                      SparseBooleanArray selectedItems) {
        this.dataset = dataset; this.selectedItems = selectedItems;
    }
}
```

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /*...*/ adapter = new ItemAdapter(dataset, selectedItems); /*...*/
    }
}
```

Sélection des éléments

Mise en place du nouveau layout :

```
public class ItemAdapter
    extends RecyclerView.Adapter<ItemAdapter.ViewHolder> {

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup viewGroup,
                                         int index) {
        View view = LayoutInflater.from(viewGroup.getContext())
            .inflate(R.layout.layout_item, viewGroup, false);
        ViewHolder viewHolder = new ViewHolder(view);
        viewHolder.bind(index);
        return viewHolder;
    }

}
```

Sélection des éléments

Modification de la vue en fonction de la sélection de l'élément :

```
public class ItemAdapter
    extends RecyclerView.Adapter<ItemAdapter.ViewHolder> {
    private final SparseBooleanArray selectedItems;

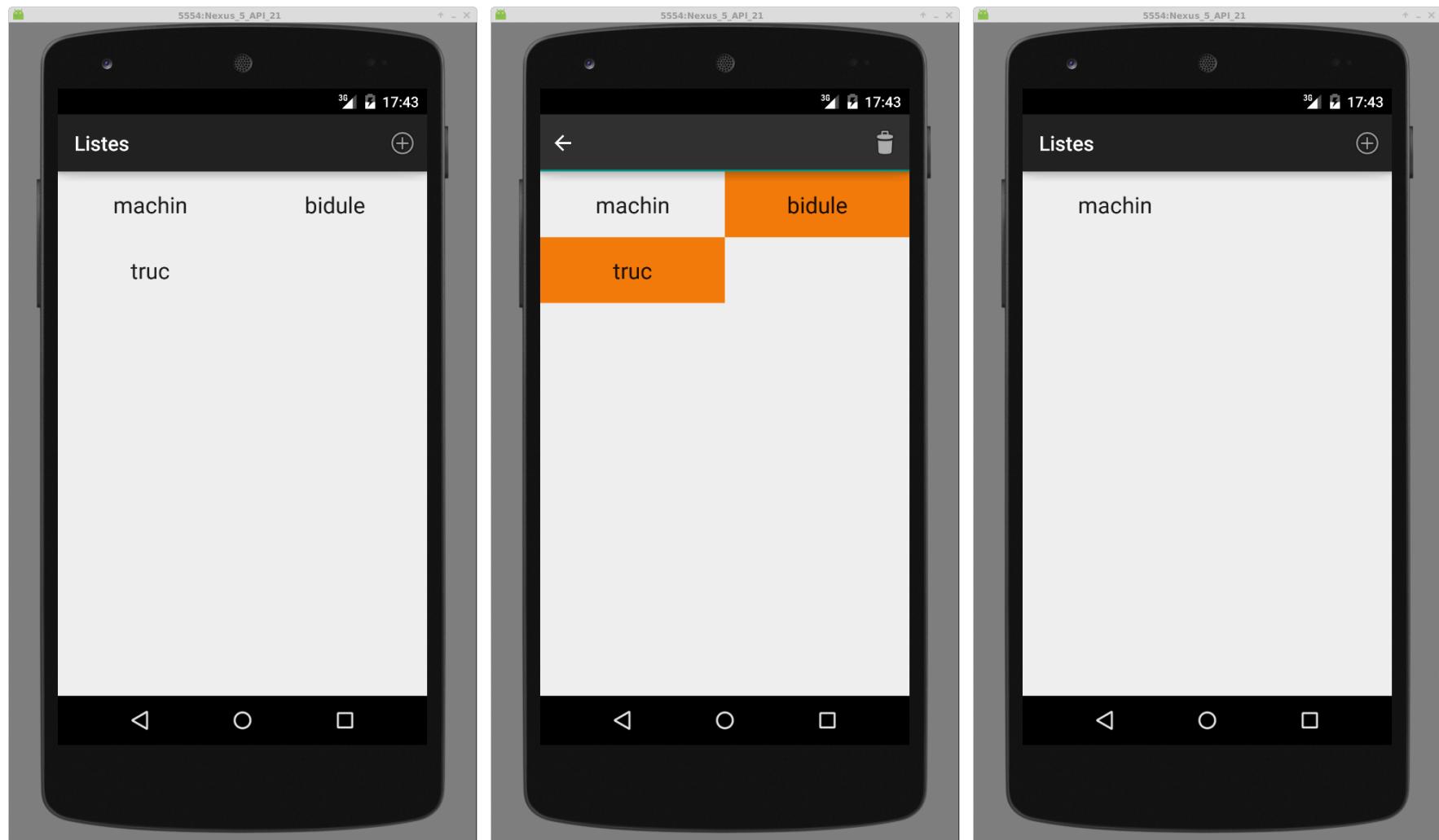
    public class ViewHolder extends RecyclerView.ViewHolder {
        int index;

        public void bind(int index) {
            /* ... */
            itemView.setSelected(selectedItems.get(index));
        }

        /* ... */
    }
}
```

Le mode "action"

Nous souhaitons pouvoir supprimer les éléments sélectionnés :



Le mode "action"

```
<menu xmlns:android="...">
    <item android:id="@+id/action_delete"
        android:title="@string/delete"
        android:icon="@android:drawable/ic_menu_delete"
        android:showAsAction="ifRoom" />
</menu>
```

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    /* ... */
    private void swapItemSelection(int index) {
        /* ... */ updateActionMode();
    }

    private void updateActionMode() {
        boolean oldSelectionModeActivated = actionMode!=null;
        boolean selectionModeActivated = numberOfSelectedItems > 0;
        if (selectionModeActivated==oldSelectionModeActivated) return;
        if (selectionModeActivated) {
            actionMode = startActionMode(actionModeCallback);
            return;
        } else actionMode.finish();
    }
}
```

Le mode "action"

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    private ActionMode.Callback actionModeCallback
        = new ActionModeCallback();

    private class ActionModeCallback implements ActionMode.Callback {
        @Override
        public boolean onCreateActionMode(ActionMode mode, Menu menu) {
            MenuInflater inflater = mode.getMenuInflater();
            inflater.inflate(R.menu.menu_actionmode, menu);
            return true;
        }

        /* ... */
    }
}
```

Le mode "action"

Gestion des événements du menu du mode ``action'' :

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    private class ActionModeCallback implements ActionMode.Callback {
        /* ... */
        @Override
        public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
            switch (item.getItemId()) {
                case R.id.action_delete: deleteSelectedItems(); return true;
                default: return false;
            }
        }
        /* ... */
    }
}
```

Le mode "action"

Gestion de la destruction du mode ``action'' :

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    private class ActionModeCallback implements ActionMode.Callback {
        /* ... */

        @Override
        public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
            return false;
        }

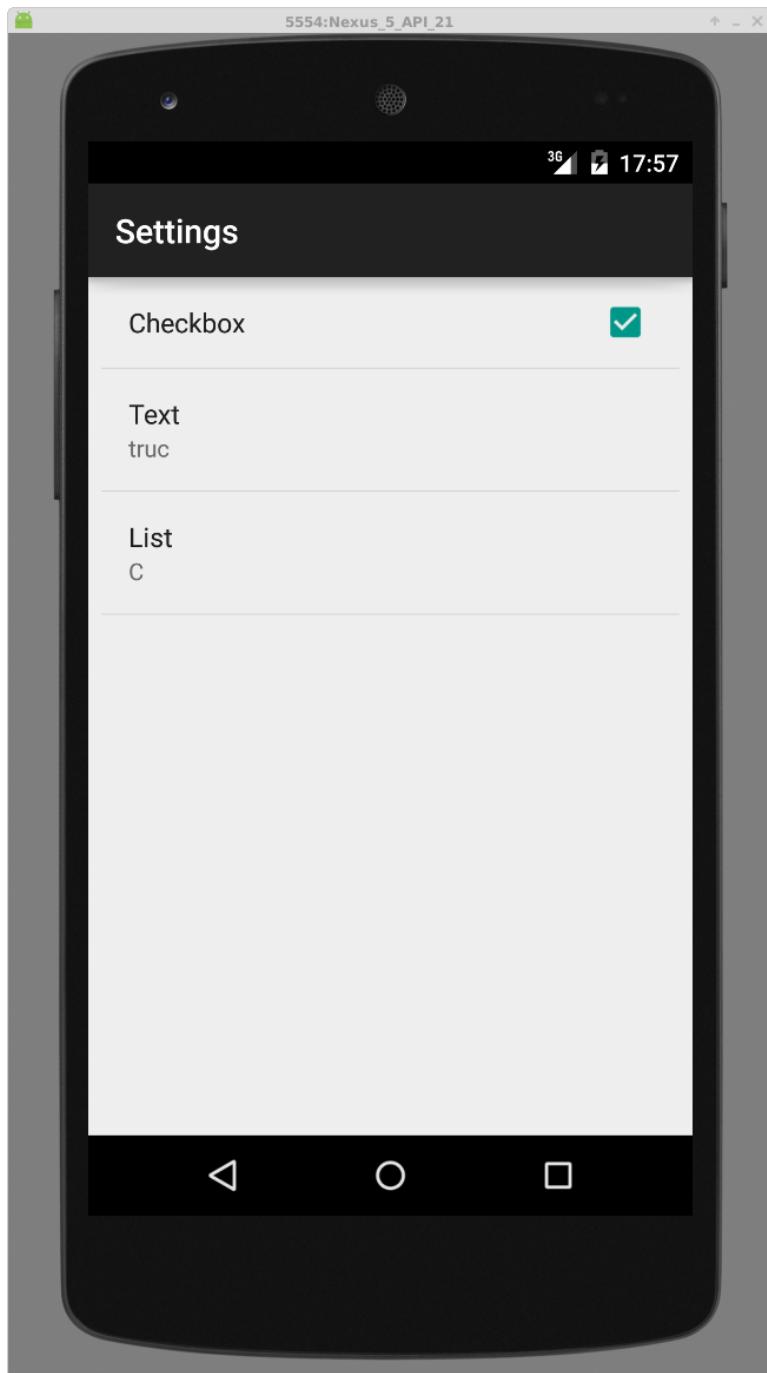
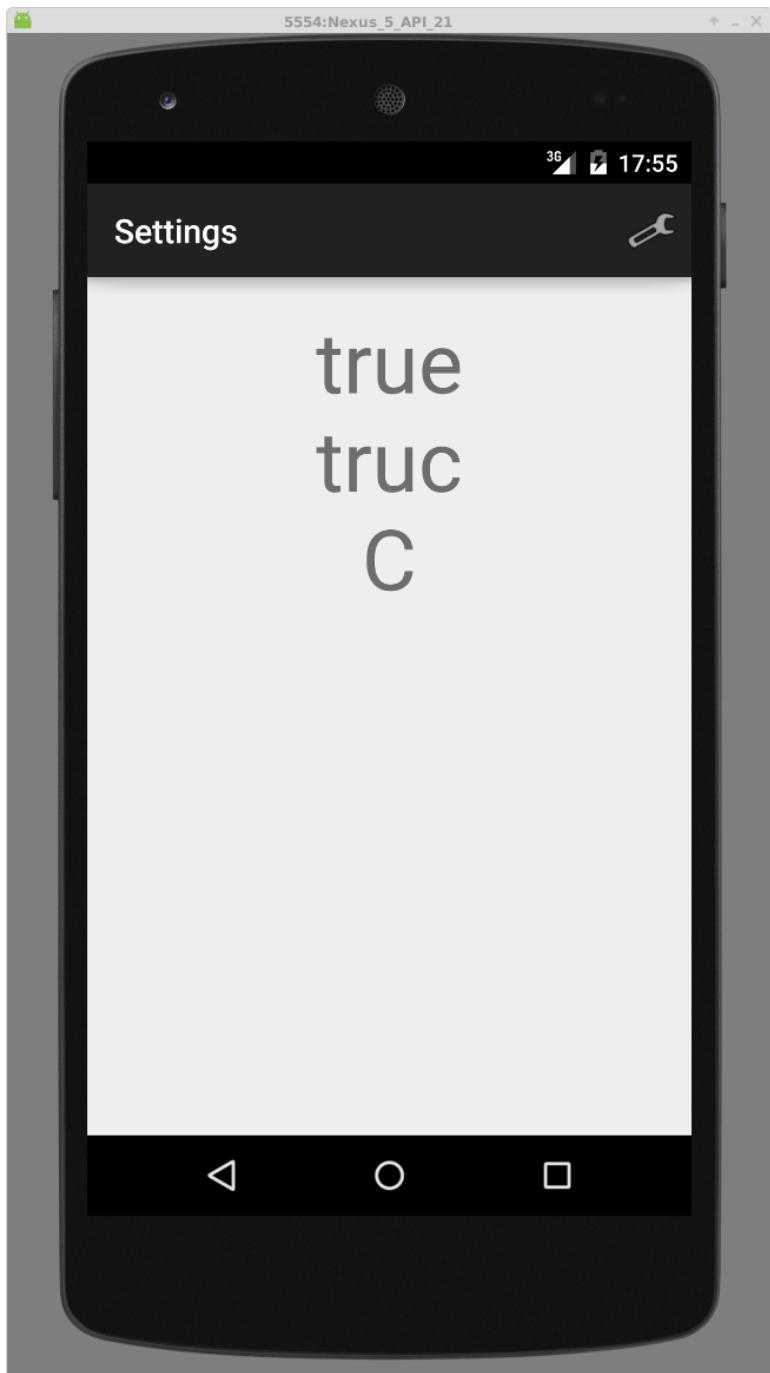
        @Override
        public void onDestroyActionMode(ActionMode mode) {
            clearSelection();
            actionMode = null;
        }
    }
}
```

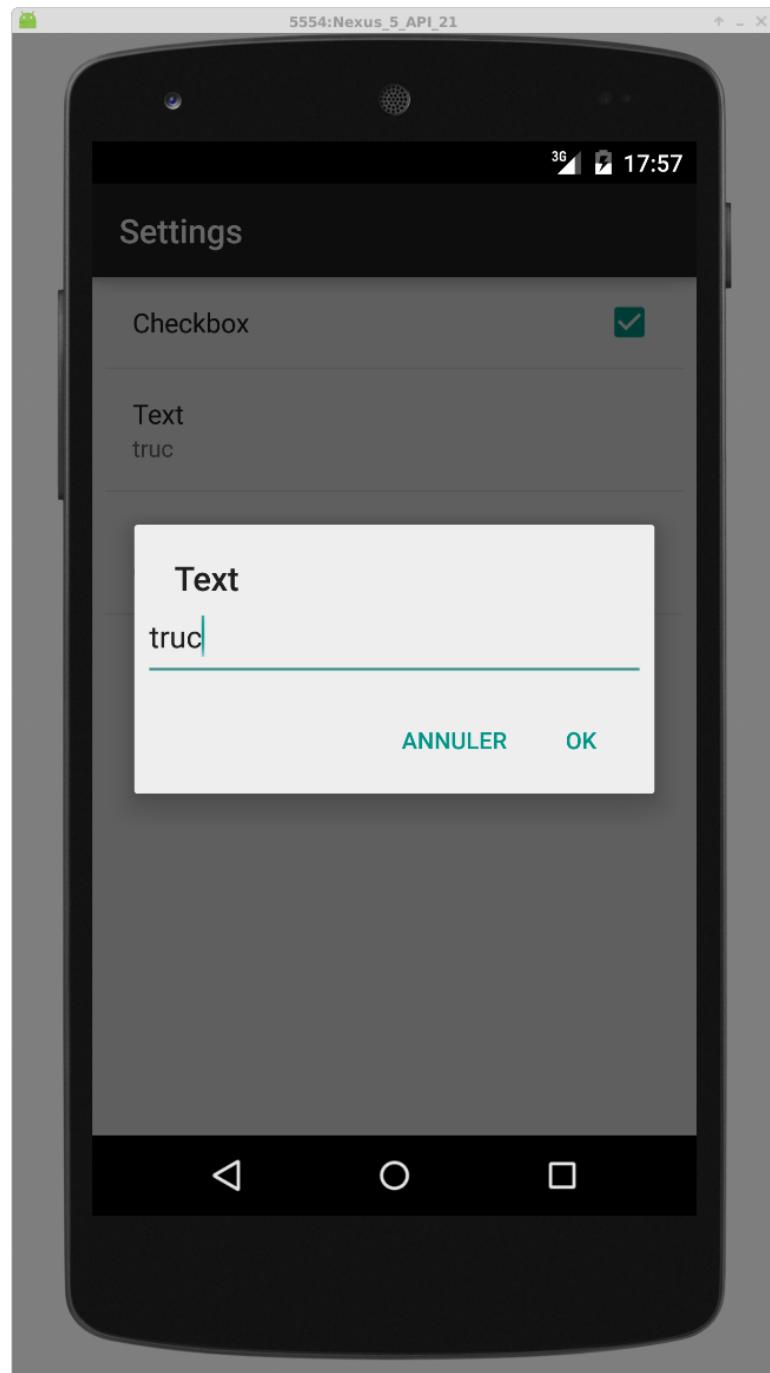
Le mode "action"

Suppression des éléments du modèle :

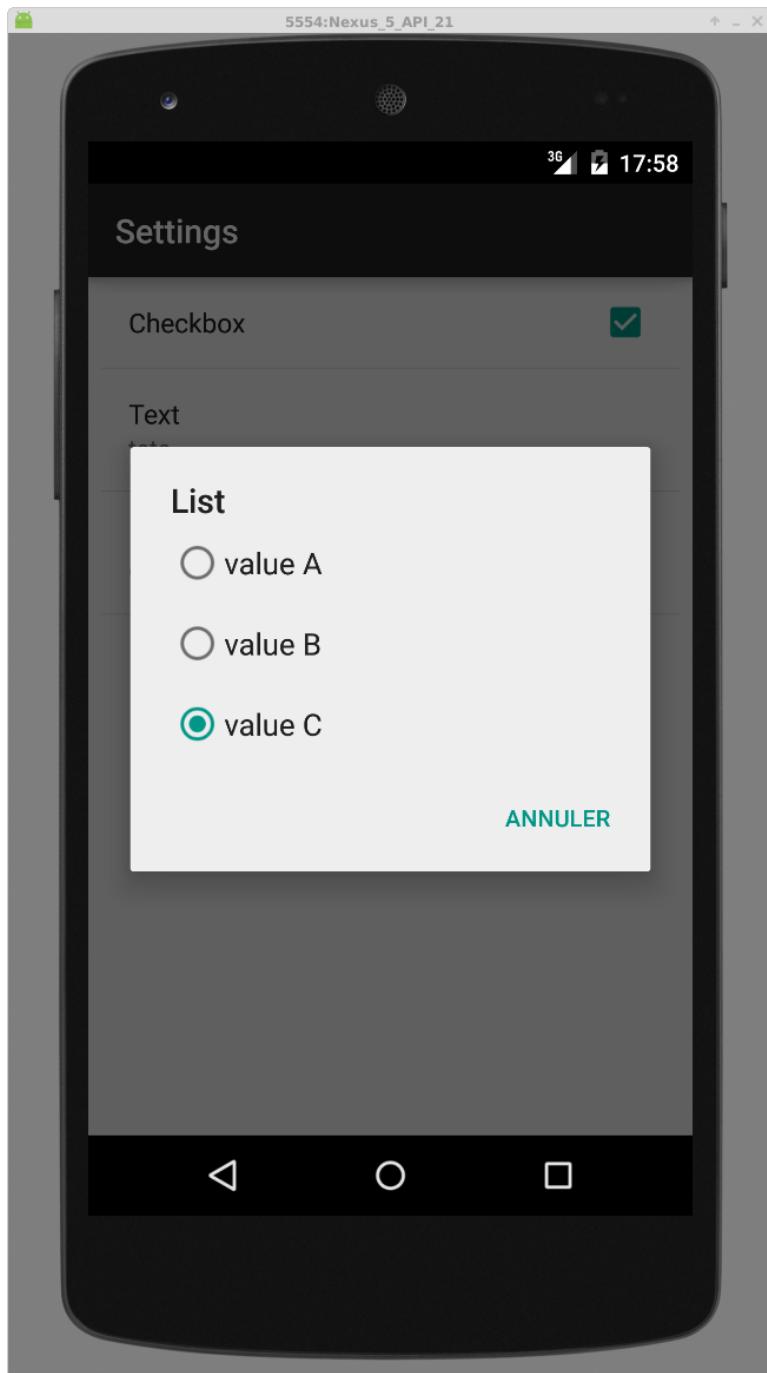
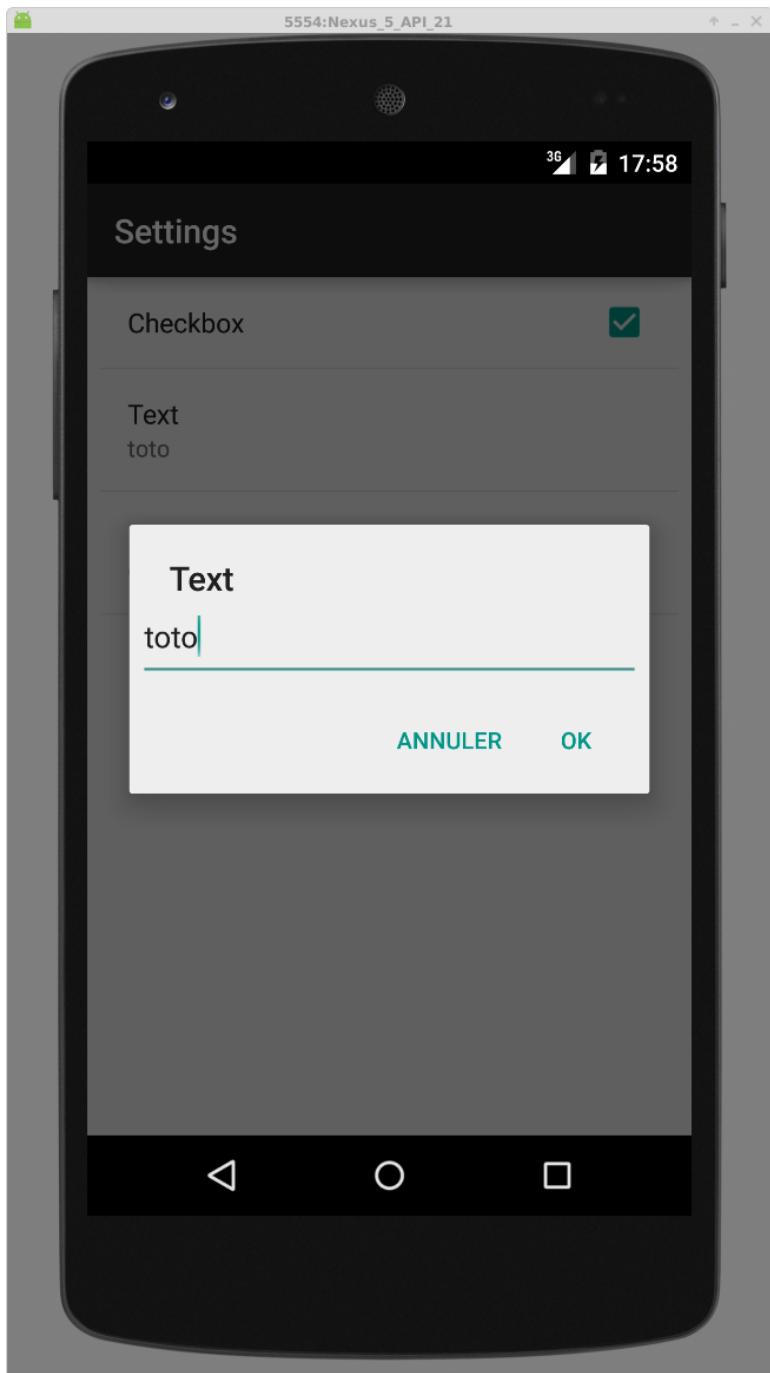
```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    private void deleteSelectedItems() {
        for (int index = selectedItems.size() - 1; index >= 0; index--) {
            if (selectedItems.valueAt(index))
                dataset.remove(selectedItems.keyAt(index));
            clearSelection();
            updateActionMode();
            adapter.notifyDataSetChanged();
        }
    }
}
```

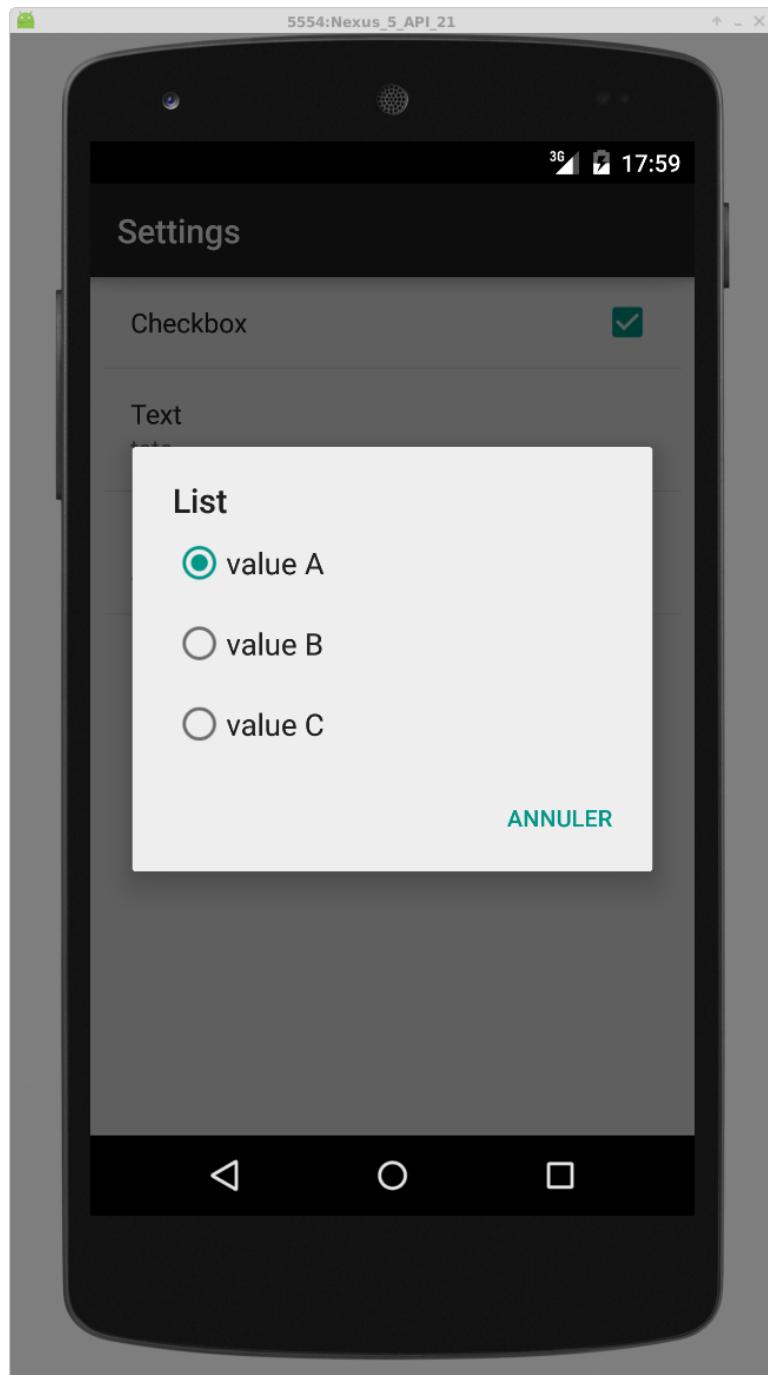
Exemple d'utilisation des préférences



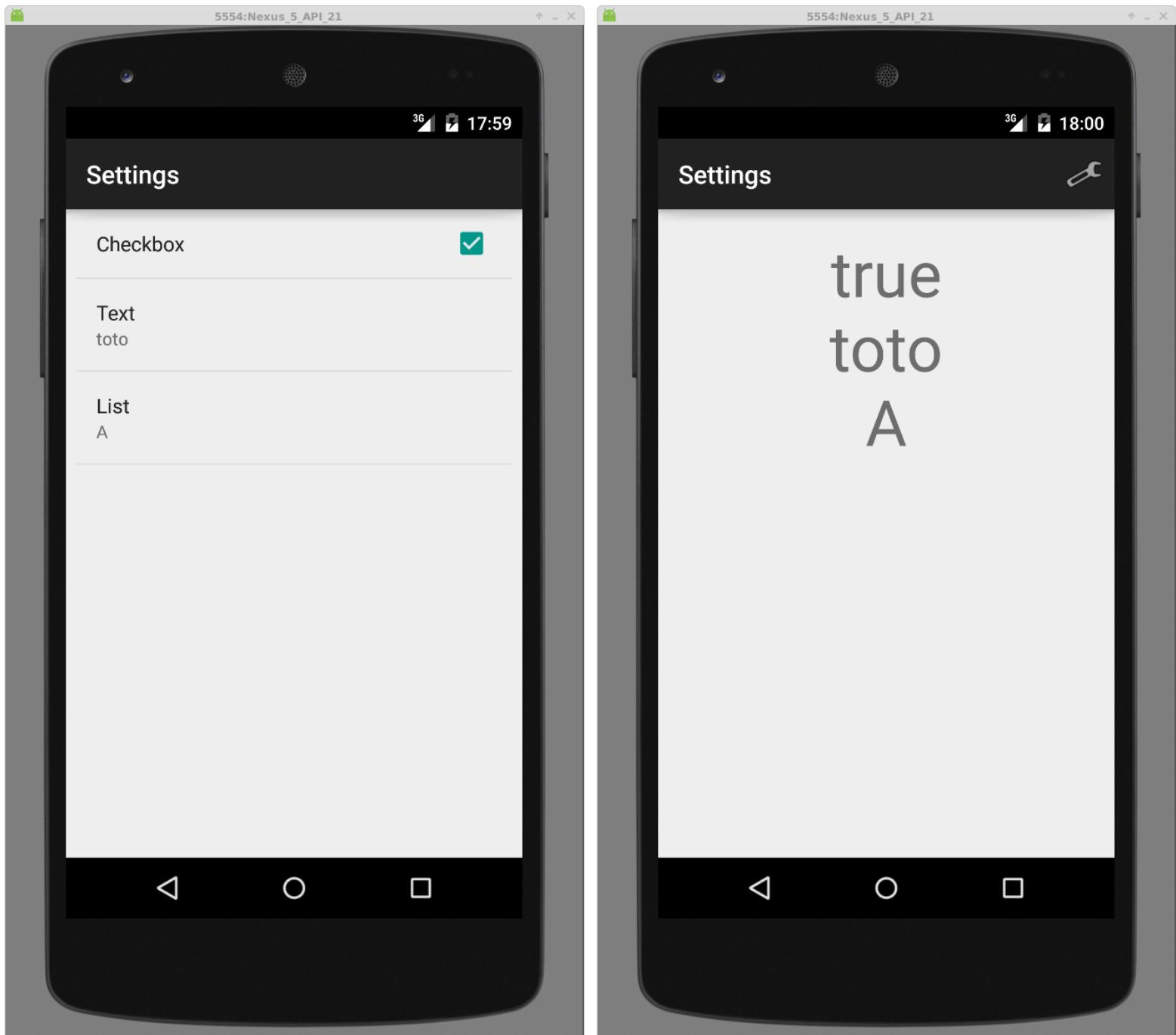


Exemple d'utilisation des préférences





Exemple d'utilisation des préférences



Structure des préférences

Un fichier XML décrit la structure des préférences :

```
<PreferenceScreen xmlns:android="...">
    <CheckBoxPreference
        android:key="pref_checkbox"
        android:title="@string/checkbox_title"
        android:defaultValue="true" />
    <EditTextPreference
        android:key="pref_text"
        android:dependency="pref_checkbox"
        android:title="@string/text_title" />
    <ListPreference
        android:key="pref_list"
        android:title="@string/list_title"
        android:entries="@array/list_entries"
        android:entryValues="@array/list_values"
        android:defaultValue="@string/list_default" />
</PreferenceScreen>
```

Les tableaux de chaînes de caractères

Les tableaux de valeurs pour la liste (fichier `values/arrays.xml`) :

```
<sources>
    <string-array name="list_entries">
        <item>value A</item>
        <item>value B</item>
        <item>value C</item>
    </string-array>
    <string-array name="list_values">
        <item>A</item>
        <item>B</item>
        <item>C</item>
    </string-array>
</resources>
```

Création du Fragment Préférences

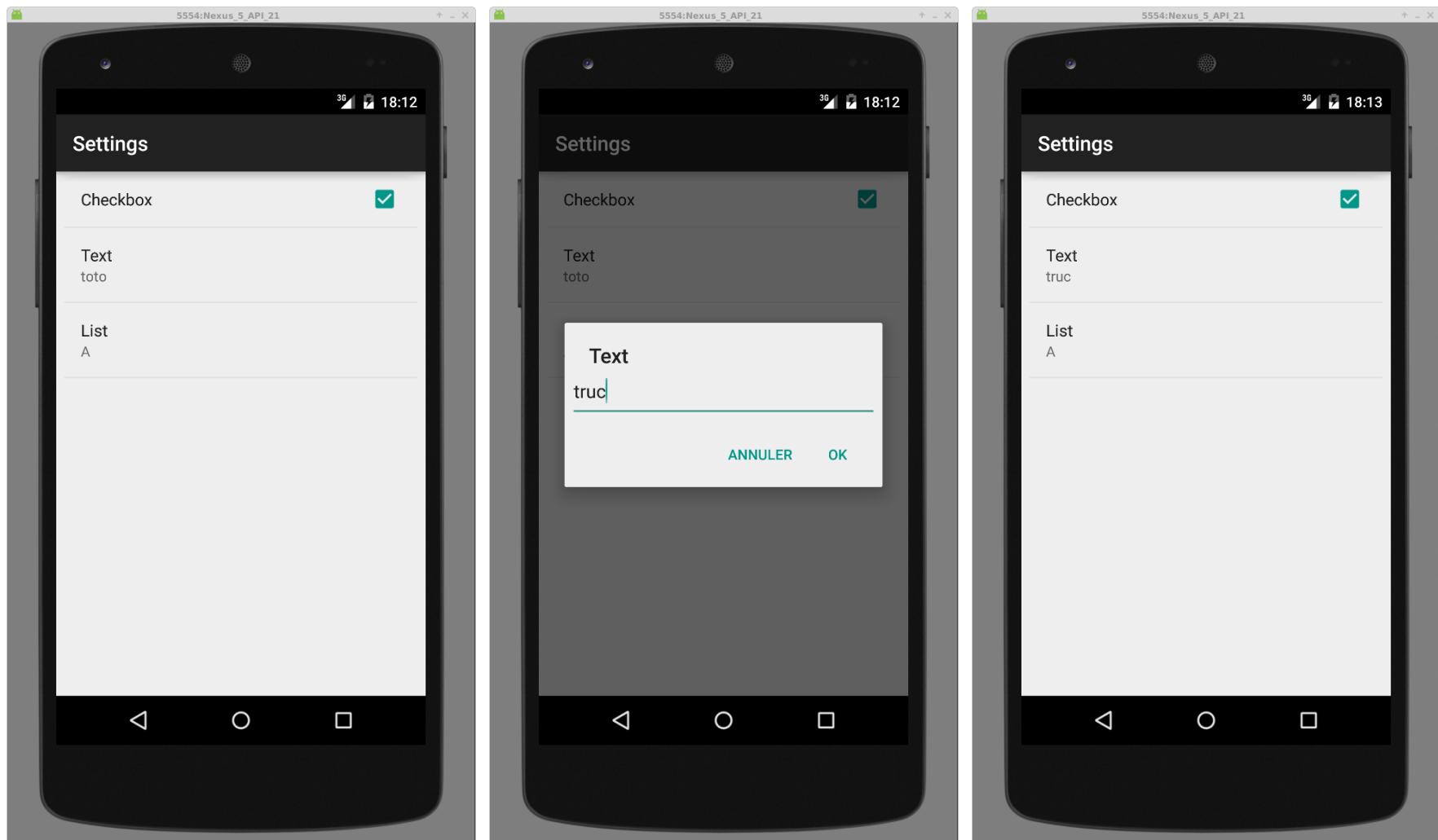
Le code d'un Fragment affichant les préférences :

```
public class SettingsFragment extends PreferenceFragment {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        addPreferencesFromResource(R.xml.settings);  
    }  
  
}
```

Les préférences sont automatiquement sauvegardées

Mise à jour du Fragment

Mise en jour du Fragment en fonction des changements :



Écouter les changements

Mise en jour du Fragment en fonction des changements :

```
public class SettingsFragment extends PreferenceFragment
    implements SharedPreferences.OnSharedPreferenceChangeListener {
    @Override
    public void onResume() {
        super.onResume();
        getPreferenceScreen().getSharedPreferences()
            .registerOnSharedPreferenceChangeListener(this);
    }

    @Override
    public void onPause() {
        super.onPause();
        getPreferenceScreen().getSharedPreferences()
            .unregisterOnSharedPreferenceChangeListener(this);
    }
}
```

Écouter les changements

Implémentation de la méthode du Listener et mise à jour de la vue :

```
public class SettingsFragment extends PreferenceFragment
    implements SharedPreferences.OnSharedPreferenceChangeListener {

    @Override
    public void onSharedPreferenceChanged(SharedPreferences preferences,
                                         String key) {
        updateSummary(preferences, key);
    }

    private void updateSummary(SharedPreferences preferences,
                               String key) {
        if (key.equals("pref_text") || key.equals("pref_list")) {
            Preference preference = findPreference(key);
            preference.setSummary(preferences.getString(key, ""));
        }
    }
}
```

Écouter les changements

Mise à jour de la vue à la création :

```
public class SettingsFragment extends PreferenceFragment
    implements SharedPreferences.OnSharedPreferenceChangeListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.settings);
        SharedPreferences sharedpreferences =
            getPreferenceScreen().getSharedPreferences();
        updateSummary(sharedpreferences, "pref_text");
        updateSummary(sharedpreferences, "pref_list");
    }
}
```

Écouter les changements

Mise à jour de la vue à la création :

```
public class MainFragment extends Fragment {
    private TextView textView;

    @Override
    public View onCreateView(LayoutInflater inflater,
                            ViewGroup container,
                            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_main, container, false);
        textView = (TextView) view.findViewById(R.id.textView);
        SharedPreferences sharedpreferences =
            PreferenceManager.getDefaultSharedPreferences(getActivity());
        updateViewWithSettings(sharedpreferences);
        return view;
    }

    private void updateViewWithSettings(SharedPreferences sharedpreferences) {
        boolean checkboxValue = sharedpreferences.getBoolean("pref_checkbox", true);
        String textViewValue = sharedpreferences.getString("pref_text", "");
        String listValue = sharedpreferences.getString("pref_list", "");
        textView.setText(checkboxValue + "\n" + textViewValue + "\n" + listValue);
    }
}
```

Vue du fragment principal

Création de la vue :

```
public class MainFragment extends Fragment {
    private TextView textView;

    @Override
    public View onCreateView(LayoutInflater inflater,
                            ViewGroup container,
                            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_main,
                                    container, false);
        textView = (TextView) view.findViewById(R.id.textView);
        SharedPreferences sharedpreferences =
            PreferenceManager.getDefaultSharedPreferences(getActivity());
        updateView(sharedpreferences);
        return view;
    }

    private void updateView(SharedPreferences preferences) { /* ... */ }
}
```

Récupération des préférences

Récupération des valeurs des préférences :

```
public class MainFragment extends Fragment {
    private TextView textView;

    private void updateView(SharedPreferences preferences) {
        boolean checkboxValue =
            preferences.getBoolean("pref_checkbox", true);
        String textValue =
            preferences.getString("pref_text", "");
        String listValue =
            preferences.getString("pref_list", "");
        textView.setText(checkboxValue +
            "\n" + textValue +
            "\n" + listValue);
    }
}
```

Écoute des changements de préférences

Écoute du changement de préférences :

```
public class MainFragment extends Fragment
    implements SharedPreferences.OnSharedPreferenceChangeListener {
    @Override
    public void onResume() {
        super.onResume();
        PreferenceManager.getDefaultSharedPreferences(getActivity()) .
            registerOnSharedPreferenceChangeListener(this);
    }

    @Override
    public void onPause() {
        PreferenceManager.getDefaultSharedPreferences(getActivity()) .
            registerOnSharedPreferenceChangeListener(this);
        super.onPause();
    }
}
```

Écoute des changements de préférences

Modification de la vue lorsque les préférences changent :

```
public class MainFragment extends Fragment
    implements SharedPreferences.OnSharedPreferenceChangeListener {
@Override
public void onSharedPreferenceChanged(SharedPreferences preferences,
                                         String key) {
    updateView(preferences);
}
}
```

Mise en place du menu

Insertion du menu dans le Fragment principal :

```
public class MainFragment extends Fragment
    implements SharedPreferences.OnSharedPreferenceChangeListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        inflater.inflate(R.menu.menu_main, menu);
    }
}
```

Mise en place du menu

```
<menu xmlns:android="...">
    <item
        android:id="@+id/action_settings"
        android:title="@string/action_settings"
        android:orderInCategory="100"
        android:icon="@android:drawable/ic_menu_preferences"
        android:showAsAction="always" />
</menu>
```



{}

L'activité principale

Création de la vue :

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (savedInstanceState == null) {
            getFragmentManager().beginTransaction()
                .add(R.id.container, new MainFragment())
                .commit();
        }
    }
}
```

L'activité principale

Traitement des événements du menu :

```
public class MainActivity extends Activity {  
    @Override  
    public boolean onOptionsItemSelected(MenuItem item) {  
        if (item.getItemId() == R.id.action_settings) {  
            openSettings();  
            return true;  
        }  
        return super.onOptionsItemSelected(item);  
    }  
}
```

L'activité principale

Ouverture du Fragment Préférences :

```
public class MainActivity extends Activity {  
    private void openSettings() {  
        getFragmentManager().beginTransaction()  
            .replace(R.id.container, new SettingsFragment())  
            .addToBackStack("result")  
            .commit();  
    }  
}
```

Les fichiers

- Stockage interne :
 - Toujours disponible
 - Par défaut, accessible uniquement par votre application
 - Fichiers Supprimés à la désinstallation de l'application
- Stockage externe :
 - Pas toujours disponible (USB, etc.)
 - Fichiers Accessibles et modifiables par tout le monde
 - Seuls les fichiers du répertoire `getExternalFilesDir()` sont supprimés à la désinstallation de l'application
 - Nécessite des permissions
 - Utile pour partager des données entre applications

Créer un fichier dans le stockage interne

Deux répertoires :

- `getFilesDir()` : Le répertoire interne de votre application
- `getCacheDir()` : le cache interne de votre application

Pour créer un fichier dans l'un de ces répertoires :

```
File file = new File(context.getFilesDir(), "fichier.txt");
```

ou

```
FileOutputStream outputStream  
= openFileOutput("fichier.txt", Context.MODE_PRIVATE);
```

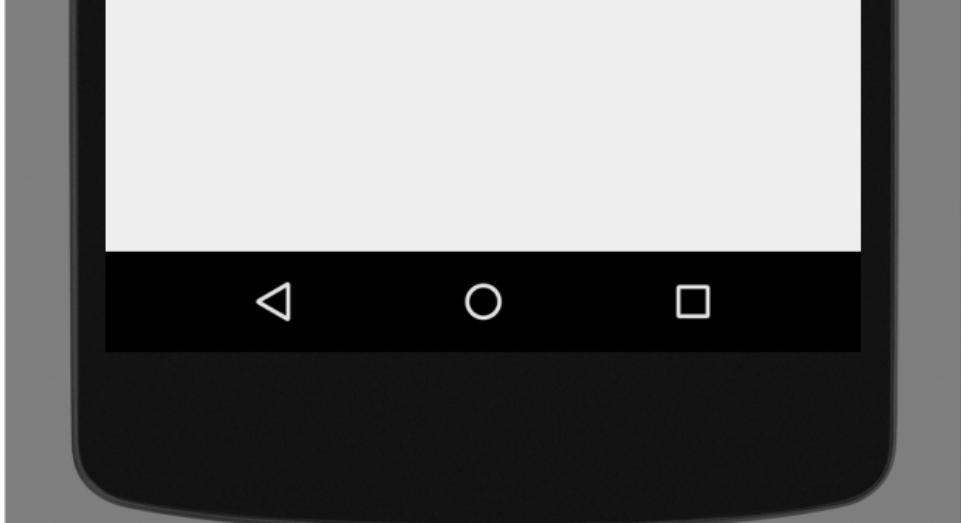
ou

```
File file  
= File.createTempFile("fichier", "tmp", context.getCacheDir());
```

Écrire et lire des données dans un fichier

Nous souhaitons écrire un éditeur de fichier :





Écrire et lire des données dans un fichier

Supposons que nous ayons l'activité suivante :

```
public class MainActivity extends Activity {  
    private EditText editText;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        editText = (EditText) findViewById(R.id.editText);  
    }  
}
```

Écrire et lire des données dans un fichier

Déclenchement de l'écriture et de la lecture du fichier :

```
public class MainActivity extends Activity {  
    private EditText editText;  
  
    @Override  
    protected void onStart() {  
        super.onResume();  
        editText.setText(load());  
    }  
  
    @Override  
    protected void onStop() {  
        super.onPause();  
        save(editText.getText().toString());  
    }  
}
```

Écrire et lire des données dans un fichier

La sauvegarde :

```
private void save(String string) {  
    File file = new File(getFilesDir(), filename);  
    try (BufferedWriter writer =  
        new BufferedWriter(new FileWriter(file))) {  
        writer.write(string);  
    } catch (IOException e) {  
        Toast.makeText(this,  
            getString(R.string.read_error),  
            Toast.LENGTH_LONG);  
    }  
}
```

Écrire et lire des données dans un fichier

La lecture :

```
private String load() {  
    File file = new File(getFilesDir(), filename);  
    if (!file.exists()) return "";  
    try (BufferedReader reader =  
            new BufferedReader(new FileReader(file))) {  
        return load(reader);  
    } catch (IOException e) {  
        Toast.makeText(this,  
                        getString(R.string.read_error),  
                        Toast.LENGTH_LONG);  
        return "";  
    }  
}
```

Écrire et lire des données dans un fichier

La lecture :

```
private String load(BufferedReader reader) throws IOException {  
    StringBuilder builder = new StringBuilder();  
    for(;;) {  
        String line = reader.readLine();  
        if (line==null) break;  
        builder.append(line).append("\n");  
    }  
    return builder.toString();  
}
```

Permissions pour le stockage externe

Il suffit d'ajouter les permissions dans le manifeste de l'application :

- Pour l'écriture :

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

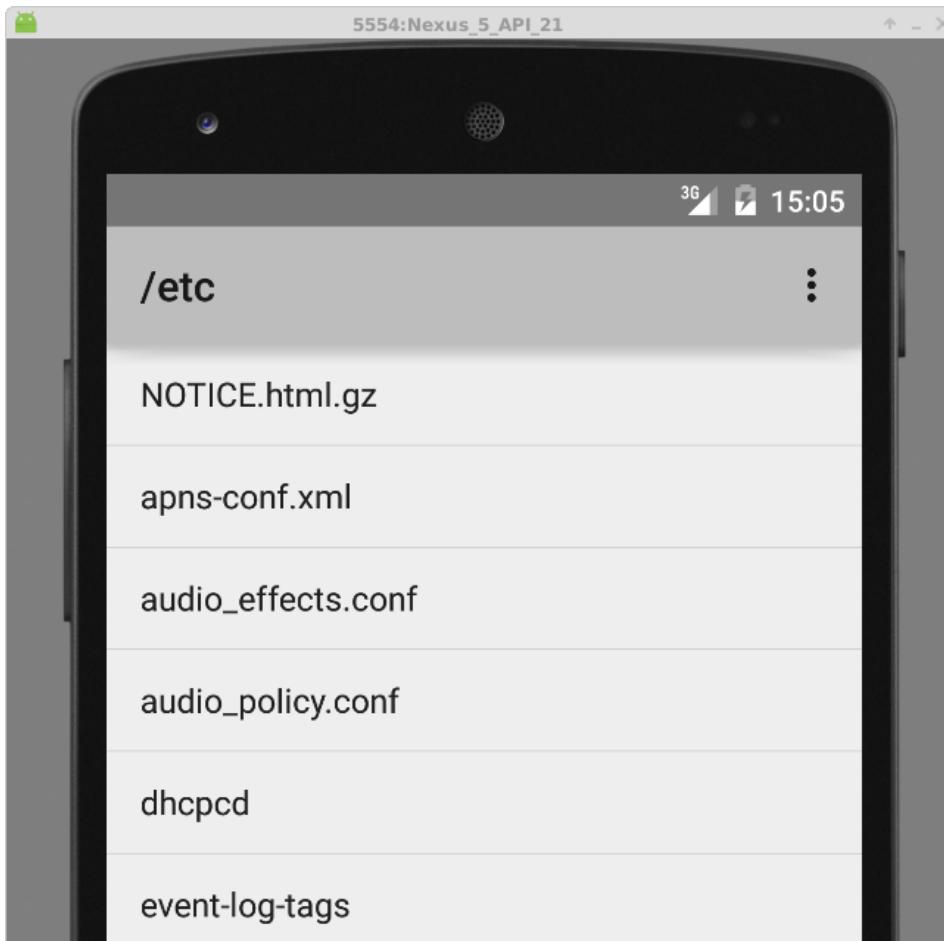
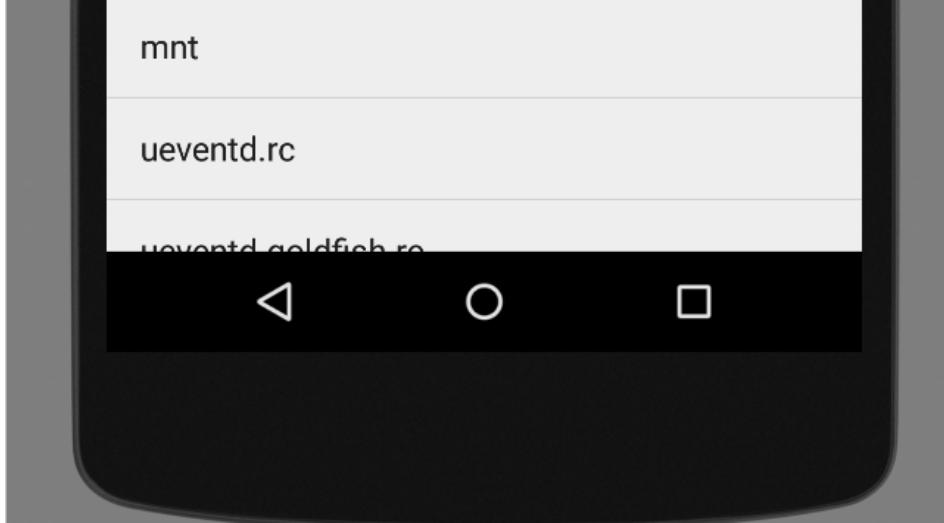
- Pour la lecture :

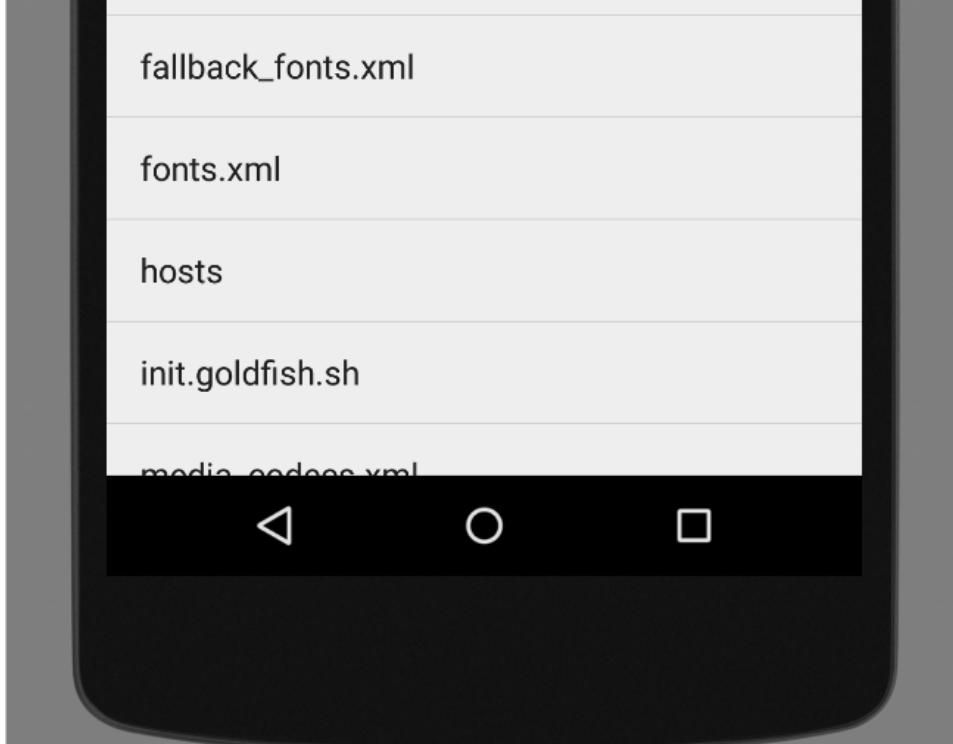
```
<manifest>
    <uses-permission
        android:name="android.permission.READ_EXTERNAL_STORAGE" />
</manifest>
```

Un exemple d'application

Nous souhaitons parcourir et lire les fichiers du stockage externe :

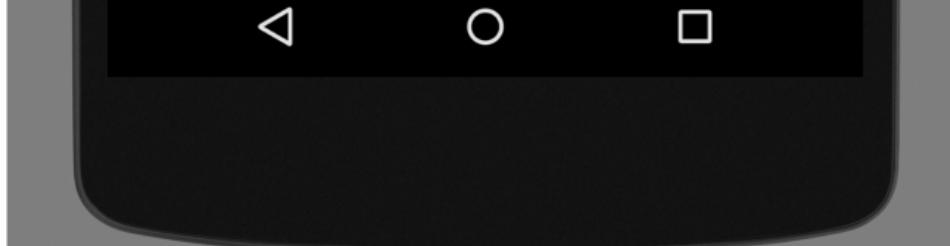






All fonts without names are added to the default list.
Fonts are chosen
based on a match: full BCP-47 language tag including
script, then just
language, and finally order (the first font containing the
glyph).

Order of appearance is also the tiebreaker for weight
matching. This is
the reason why the 900 weights of Roboto precede the
700 weights - we
prefer the former when an 800 weight is requested.
Since bold spans
effectively add 300 to the weight, this ensures that 900
is the bold
paired with the 500 weight, ensuring adequate contrast.
-->
<familyset version="22">
 <!-- first font is default -->
 <family name="sans-serif">



Deux types de fragments :

- Affichage d'un répertoire : `DirectoryFragment`
- Affichage du contenu d'un fichier : `FileFragment`

Affichage du contenu d'un fichier

```
<FrameLayout xmlns:android="..."  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <ScrollView  
        android:layout_width="match_parent"  
        android:layout_height="match_parent">  
        <TextView android:layout_width="match_parent"  
            android:layout_height="wrap_content"  
            android:id="@+id/textView"/>  
    </ScrollView>  
</FrameLayout>
```

Affichage du contenu d'un fichier

Création de l'instance et conservation du chemin :

```
public class FileFragment extends Fragment {

    public static FileFragment newInstance(String filename) {
        FileFragment fragment = new FileFragment();
        Bundle args = new Bundle();
        args.putString("path", filename);
        fragment.setArguments(args);
        return fragment;
    }

    private String path() { return getArguments().getString("path"); }

}
```

Affichage du contenu d'un fichier

Création de la vue et gestion du cycle de vie :

```
public class FileFragment extends Fragment {
    private TextView textView;

    public View onCreateView(LayoutInflater inflater,
                           ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_file,
                                     container, false);
        textView = (TextView)view.findViewById(R.id.textView);
        return view;
    }

    public void onResume() {
        super.onResume(); textView.setText(load());
        getActivity().setTitle(path());
    }
}
```

Affichage du contenu d'un fichier

Chargement du contenu du fichier :

```
public class FileFragment extends Fragment {
    private String load() {
        File file = new File(path());
        if (!file.exists()) return "";
        try (BufferedReader reader =
                new BufferedReader(new FileReader(file))) {
            return load(reader);
        } catch (IOException e) {
            Toast.makeText(this.getActivity(),
                           getString(R.string.read_error),
                           Toast.LENGTH_LONG);
            return "";
        }
    }
}
```

Affichage du contenu d'un fichier

Chargement du contenu du fichier :

```
public class FileFragment extends Fragment {  
    private String load(BufferedReader reader) throws IOException {  
        StringBuilder builder = new StringBuilder();  
        for(;;) {  
            String line = reader.readLine();  
            if (line==null) break;  
            builder.append(line).append("\n");  
        }  
        return builder.toString();  
    }  
}
```

Affichage du contenu d'un répertoire

Création de l'instance :

```
public class DirectoryFragment extends ListFragment {  
    public static DirectoryFragment newInstance(String path) {  
        DirectoryFragment fragment = new DirectoryFragment();  
        Bundle args = new Bundle();  
        args.putString("path", path);  
        fragment.setArguments(args);  
        return fragment;  
    }  
  
    private String path() {return getArguments().getString("path");}  
}
```

Affichage du contenu d'un répertoire

Création de la vue et gestion du cycle de vie :

```
public class DirectoryFragment extends ListFragment {
    private String[] files;

    void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        File directory = new File(path());
        files = directory.list();
        setListAdapter(new ArrayAdapter<>(getActivity(),
            android.R.layout.simple_list_item_1,
            android.R.id.text1, files));
    }

    public void onResume() {
        super.onResume();
        getActivity().setTitle(path());
    }
}
```

Affichage du contenu d'un répertoire

Mise en place des notifications de l'activité :

```
public class DirectoryFragment extends ListFragment {  
    private OnDirectoryFragmentListener listener;  
  
    public interface OnDirectoryFragmentListener {  
        public void onChangePath(String path);  
    }  
}
```

Affichage du contenu d'un répertoire

Mise en place des notifications de l'activité :

```
public class DirectoryFragment extends ListFragment {
    private OnDirectoryFragmentListener listener;

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            listener = (OnDirectoryFragmentListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString()
                + " must implement OnDirectoryFragmentListener");
        }
    }

    @Override
    public void onDetach() { super.onDetach(); listener = null; }
}
```

Affichage du contenu d'un répertoire

Mise en place des notifications de l'activité :

```
public class DirectoryFragment extends ListFragment {
    private OnDirectoryFragmentListener listener;

    @Override
    public void onListItemClick(ListView list,
                               View view,
                               int position, long id) {
        super.onListItemClick(list, view, position, id);
        if (null != listener) {
            String path = path().equals("//") ? "" : path();
            listener.onChangePath(path + "/" + files[position]);
        }
    }
}
```

L'activité principale

Création de la vue :

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        if (savedInstanceState == null) {  
            getFragmentManager().beginTransaction()  
                .add(R.id.container, DirectoryFragment.newInstance("//"))  
                .commit();  
        }  
    }  
}
```

L'activité principale

Réception des notifications des fragments :

```
public class MainActivity extends Activity
    implements DirectoryFragment.OnDirectoryFragmentListener {

    @Override
    public void onChangePath(String path) {
        File file = new File(path);
        Fragment fragment =
            (file.isDirectory())
                ?DirectoryFragment.newInstance(path)
                :FileFragment.newInstance(path);
        getFragmentManager().beginTransaction()
            .replace(R.id.container, fragment)
            .addToBackStack(path)
            .commit();
    }
}
```

Bases de données SQLite

Comment stocker des infos structurées dans base de données ?

Pour cela, nous allons :

- définir un modèle de données (les tables) ;
- représenter ce modèle sous la forme de classes ;
- définir un SQL Helper pour faciliter la création des tables ;
- voir comment modifier, consulter les données.

Définition du modèle

Définition du modèle de données, c'est-à-dire, les tables :

```
public class DatabaseContract {  
    public interface Articles extends BaseColumns {  
        String tableName = "articles";  
        String columnGuid = "guid";  
        String columnGuidType = "TEXT";  
        String columnTitle = "title";  
        String columnTitleType = "TEXT";  
        String columnContent = "content";  
        String columnContentType = "TEXT";  
    }  
}
```

- Les autres types possibles : NULL, INTEGER, REAL et BLOB ;
- Il peut y avoir plusieurs tables.

Définition du SQLiteOpenHelper

Instructions SQL à partir de la définition du modèle :

```
public class DatabaseOpenHelper extends SQLiteOpenHelper {  
  
    private static final String SQLCreateTableArticles =  
        "CREATE TABLE " + DatabaseContract.Articles.tableName + " (" +  
        DatabaseContract.Articles._ID + " INTEGER PRIMARY KEY, " +  
        DatabaseContract.Articles.columnGuid + " " +  
        DatabaseContract.Articles.columnGuidType + ", " +  
        DatabaseContract.Articles.columnTitle + " " +  
        DatabaseContract.Articles.columnTitleType + ", " +  
        DatabaseContract.Articles.columnContent + " " +  
        DatabaseContract.Articles.columnContentType +  
        " )";  
  
    private static final String SQLDeleteTableArticles =  
        "DROP TABLE IF EXISTS " + DatabaseContract.Articles.tableName;  
}
```

Définition du SQLiteOpenHelper

Création de la base de données :

```
public class DatabaseOpenHelper extends SQLiteOpenHelper {
    public static final int databaseVersion = 1;
    public static final String databaseName = "articles.db";

    public DatabaseOpenHelper(Context context) {
        super(context, databaseName, null, databaseVersion);
    }

    @Override
    public void onCreate(SQLiteDatabase database) {
        database.execSQL(SQLCreateTableArticles);
    }
}
```

Définition du SQLiteOpenHelper

Mise à jour de la base de données lors d'un changement de version :

```
public class DatabaseOpenHelper extends SQLiteOpenHelper {
    public static final int databaseVersion = 1;

    @Override
    public void onUpgrade(SQLiteDatabase database, int oldVersion,
                          int newVersion) {
        database.execSQL(SQLDeleteTableArticles);
        onCreate(database);
    }

    @Override
    public void onDowngrade(SQLiteDatabase database, int oldVersion,
                           int newVersion) {
        onUpgrade(database, oldVersion, newVersion);
    }
}
```

Utilisation du SQLiteOpenHelper

Avant d'utiliser le SQLiteOpenHelper, il faut l'instancier :

```
databaseOpenHelper = new DatabaseOpenHelper(context);
```

Insertion (INSERT INTO) :

```
public void insert(Article article) {  
    SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();  
    ContentValues values = new ContentValues();  
    values.put(DatabaseContract.Articles.columnGuid, article.guid);  
    values.put(DatabaseContract.Articles.columnContent, article.content);  
    values.put(DatabaseContract.Articles.columnTitle, article.title);  
    database.insert(DatabaseContract.Articles.tableName, null, values);  
}
```

Utilisation du SQLiteOpenHelper

Mise en jour (UPDATE) :

```
public void update(Article article) {  
    SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();  
    ContentValues values = new ContentValues();  
    values.put(DatabaseContract.Articles.columnContent, article.content);  
    values.put(DatabaseContract.Articles.columnTitle, article.title);  
    String selection = DatabaseContract.Articles.columnGuid + " = ?";  
    String[] selectionArgs = {article.guid};  
    database.update(DatabaseContract.Articles.tableName, values,  
                    selection, selectionArgs);  
}
```

Utilisation du SQLiteOpenHelper

Suppression (DELETE) :

```
public void delete(String guid) {  
    SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();  
    String selection = DatabaseContract.Articles.columnGuid + " = ?";  
    String[] selectionArgs = {guid};  
    database.delete(DatabaseContract.Articles.tableName,  
                    selection,  
                    selectionArgs);  
}
```

Utilisation du SQLiteOpenHelper

SELECT * FROM articles :

```
public Cursor getArticles() {  
    SQLiteDatabase database = databaseOpenHelper.getReadableDatabase();  
    return database.query(DatabaseContract.Articles.tableName,  
        null, null, new String[] {}, null, null, null);  
}
```

SELECT guid, title FROM articles WHERE title LIKE "%s%" :

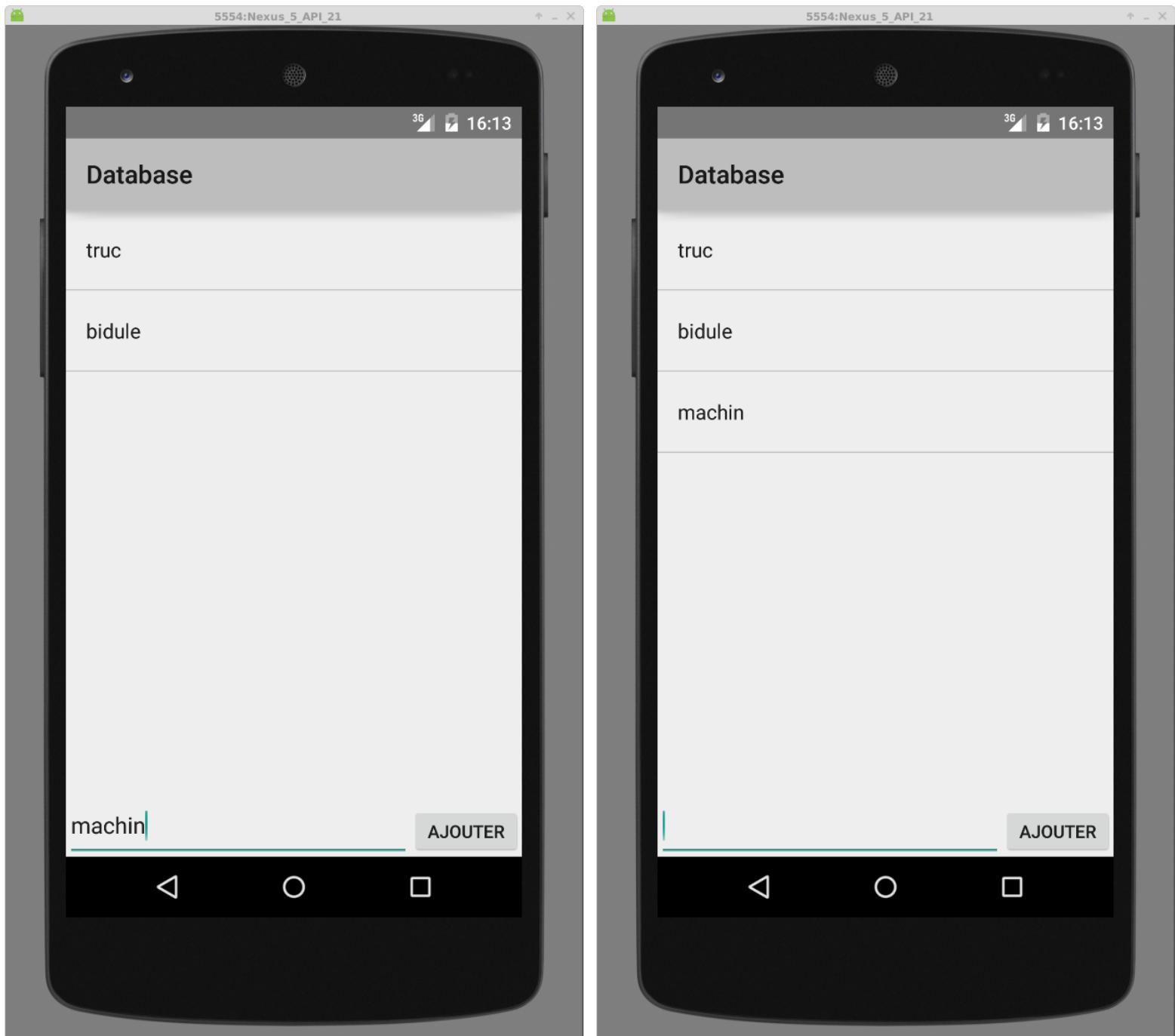
```
public Cursor getArticles(String s) {  
    SQLiteDatabase database = databaseOpenHelper.getReadableDatabase();  
    String selection = DatabaseContract.Articles.columnTitle + " LIKE ?";  
    String[] selectionArgs = {"%" + s + "%"};  
    String[] columns = { DatabaseContract.Articles.columnGuid,  
                        DatabaseContract.Articles.columnTitle};  
    return database.query(DatabaseContract.Articles.tableName,  
        columns, selection, selectionArgs, null, null, null);  
}
```

Les curseurs

Les curseurs permettent de parcourir le résultat d'une requête :

- `int getCount();`
- `boolean moveToFirst();`
- `boolean moveToNext();`
- `boolean moveToPosition(int position);`
- `int getColumnIndex(String columnName);`
- `String getString(int cId), double getDouble(int cId) ...`
- `void close();`
- `...`

Exemple d'application



Les tables

La structure de la base de données :

```
public class DatabaseContract {  
  
    public interface Items extends BaseColumns {  
        String tableName = "items";  
        String columnText = "text";  
        String columnTextType = "TEXT";  
    }  
  
}
```

Le code de l'activité principale

En supposant que nous avons écrit le SQLiteOpenHelper :

```
public class MainActivity extends Activity {
    private DatabaseOpenHelper databaseOpenHelper;
    private ItemAdapter adapter;
    private EditText editText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        databaseOpenHelper = new DatabaseOpenHelper(this);
        setContentView(R.layout.activity_main);
        recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        adapter = new ItemAdapter(getItems());
        recyclerView.setAdapter(adapter);
        editText = (EditText) findViewById(R.id.editText);
    }
}
```

Le code de l'activité principale

Les méthodes qui interagissent avec la base de données :

```
public class MainActivity extends Activity {
    private DatabaseOpenHelper databaseOpenHelper;

    private Cursor getItems() {
        SQLiteDatabase database = databaseOpenHelper.getReadableDatabase();
        return database.query(DatabaseContract.Items.tableName,
            null, null, new String[] {}, null, null, null);
    }

    private void insertItem(String text) {
        SQLiteDatabase database = databaseOpenHelper.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(DatabaseContract.Items.columnText, text);
        database.insert(DatabaseContract.Items.tableName, null, values);
    }
}
```

Le code de l'activité principale

Traitement des clics sur le bouton "Ajouter" :

```
public class MainActivity extends Activity {  
    private ItemAdapter adapter;  
    private EditText editText;  
  
    public void onAddItem(View view) {  
        insertItem(editText.getText().toString());  
        adapter.changeCursor(getItems());  
        editText.setText("");  
    }  
}
```

Le code de l'adaptateur

Création des vues :

```
public class ItemAdapter extends RecyclerView.Adapter<ViewHolder> {

    private Cursor cursor;

    public ItemAdapter(Cursor cursor) {
        this.cursor = cursor;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.item, parent, false);
        ViewHolder viewHolder = new ViewHolder(view);
        return viewHolder;
    }

}
```

Le code de l'adaptateur

Mise à jour des vues et nombre d'éléments dans le curseur :

```
public class ItemAdapter extends RecyclerView.Adapter<ViewHolder> {
    private Cursor cursor;

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        cursor.moveToPosition(position);
        int index =
            cursor.getColumnIndex(DatabaseContract.Items.columnText);
        String text = cursor.getString(index);
        holder.bind(text);
    }

    @Override
    public int getItemCount() { return cursor.getCount(); }

}
```

Le code de l'adaptateur

Changement de curseur :

```
public class ItemAdapter extends RecyclerView.Adapter<ViewHolder> {

    private Cursor cursor;

    public void changeCursor(Cursor cursor) {
        this.cursor.close();
        this.cursor = cursor;
        notifyDataSetChanged();
    }

}
```

Le code de ViewHolder

```
public class ViewHolder extends RecyclerView.ViewHolder {  
    private final TextView textView;  
  
    public ViewHolder(View itemView) {  
        super(itemView);  
        textView = (TextView)itemView.findViewById(R.id.text);  
    }  
  
    public void bind(String text) {  
        textView.setText(text);  
    }  
}
```

Fournisseur de contenu

- Un fournisseur de contenu est un composant d'une application
- Il permet l'accès à des données
- Il présente les données sous la forme d'une ou plusieurs tables
- Chaque ligne possède un identifiant

<u>ID</u>	firstname	lastname
1	Bob	Machin
2	Alice	Truc
3	Oscar	Chose
4	Trudy	Bidule

Fournisseur de contenu

- On accède à un fournisseur de contenu via un ContentResolver

```
ContentResolver contentResolver = context.getContentResolver();
```

- Il permet les quatres opérations de base (CRUD) :
 - Create : contentResolver.insert
 - Read : contentResolver.query
 - Update : contentResolver.update
 - Delete : contentResolver.delete

Content URIs

- Un URI (Uniform Resource Identifier) identifie une donnée
- La classe Uri va nous permettre de manipuler des URI

```
String authority = "com.univ_amu.cci";
String users    = "users";
Uri contentUri = Uri.parse("content://" + authority + "/" + users);
// -> content://com.univ_amu.cci/users
Uri user2Uri = ContentUris.withAppendedId(contentUri, 2);
// -> content://com.univ_amu.cci/users/2
```

- Pour faire une requête, il faut un Uri :

```
cursor = contentResolver.query(contentUri, /*...*/);
contentResolver.insert(contentUri, /*...*/);
contentResolver.delete(user2Uri, /*...*/);
```

Schéma de la base de données

```
public class Contract {  
    public static final String AUTHORITY = "com.univ_amu.cci";  
    public static final String USERS = "users";  
    public static final Uri CONTENT_URI =  
        Uri.parse("content://" + AUTHORITY + "/" + USERS);  
  
    public interface Users extends BaseColumns {  
        String FIRSTNAME = "FIRSTNAME";  
        String LASTNAME = "LASTNAME";  
    }  
}
```

L'interface BaseColumns contient `_ID`

Effectuer d'une requête

```
cursor = contentResolver.query(Contract.CONTENT_URI,  
                             projection, selection, selectionArgs, sortOrder);
```

Selection des colonnes

```
String[] projection = { Contract.Users._ID,  
                        Contract.Users.FIRSTNAME,  
                        Contract.Users.LASTNAME };
```

Ajout d'une clause "where"

```
String selection = Contract.Users.FIRSTNAME + " = ?";  
String[] selectionArgs = { "Bob" };
```

Ajout d'un "order by"

```
String sortOrder = Contract.Users.LASTNAME+" ASC";
```

```
Select _ID, FIRSTNAME, LASTNAME  
FROM USERS WHERE FIRSTNAME = 'bob' ORDER BY LASTNAME ASC;
```

Insérer des données

```
newUri = contentResolver.insert(Contract.CONTENT_URI, values);
```

Les valeurs à ajouter

```
ContentValues values = new ContentValues();
values.put(Contract.Users.FIRSTNAME, "Bob");
values.put(Contract.Users.LASTNAME, "Machin");
```

Uri retourné

```
content://com.univamu.cci/users/[identifiant de la nouvelle ligne]
```

```
INSERT INTO USERS(FIRSTNAME, LASTNAME) VALUES ("Bob", "Machin");
```

Modifier des données

```
rowsUpdatedCount = contentResolver.update(Contract.CONTENT_URI,  
                                         values, selection, selectionArgs);
```

Données à mettre à jour

```
ContentValues values = new ContentValues();  
values.put(Contract.Users.LASTNAME, "Truc");
```

Ajout d'une clause "where"

```
String selection = Contract.Users.FIRSTNAME + " = ?";  
String[] selectionArgs = { "Bob" };
```

```
UPDATE USERS SET LASTNAME = "Truc" WHERE FIRSTNAME = "Bob";
```

Supprimer des données

```
rowsDeletedCount = contentResolver.delete(Contract.CONTENT_URI,  
                                         selection, selectionArgs);
```

Ajout d'une clause "where"

```
String selection = Contract.Users.FIRSTNAME + " = ?";  
String[] selectionArgs = { "Bob" };
```

```
DELETE FROM USERS WHERE FIRSTNAME = "Bob";
```

Implémentation d'un fournisseur de contenu

```
public class UsersProvider extends ContentProvider {  
    @Override  
    public boolean onCreate() { /* ... */ }  
  
    @Override  
    public Cursor query(Uri uri,  
                        String[] projection,  
                        String selection, String[] selectionArgs,  
                        String sortOrder) { /* ... */ }  
  
    @Override  
    public Cursor insert(Uri uri, ContentValues values) { /* ... */ }  
  
    @Override  
    public Cursor update(Uri uri, ContentValues values,  
                        String selection, String[] selectionArgs) { /* ... */ }  
  
    @Override  
    public Cursor delete(Uri uri,  
                        String selection, String[] selectionArgs) { /* ... */ }  
  
    @Override  
    public String getType(Uri uri) { /* ... */ }  
}
```

Déclaration du fournisseur de contenu

Dans le manifeste de l'application :

```
<manifest ...>
    <application ...>
        ...
        <provider
            android:authorities="com.univ_amu.cci"
            android:name=".UsersProvider"
            android:exported="true"/>
        <application>
    </manifest>
```

DatabaseHelper pour SQLite

```
public class UserDatabaseHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "users.db";

    public UserDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) { createTable(db); }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        dropTable(db); onCreate(db);
    }

    @Override
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }

    private static void createTable(SQLiteDatabase db) { /* .. */ }
    private static void dropTable(SQLiteDatabase db) { /* ... */ }
}
```

Utilisation du DatabaseHelper

```
public class UsersProvider extends ContentProvider {
    private UserDatabaseHelper openHelper;

    @Override
    public boolean onCreate() {
        openHelper = new UserDatabaseHelper(getContext());
    }

    @Override
    public Cursor query(Uri uri,
                        String[] projection,
                        String selection, String[] selectionArgs,
                        String sortOrder) {
        SQLiteDatabase database = openHelper.getReadableDatabase(); /* ... */
    }

    @Override
    public Cursor update(Uri uri, ContentValues values,
                        String selection, String[] selectionArgs) {
        SQLiteDatabase database = openHelper.getWritableDatabase(); /* ... */
    }
    /* ... */
}
```

Détection du type d'URI

```
public class UsersProvider extends ContentProvider {
    private static final int USERS = 1;
    private static final int USERS_ID = 2;

    private static final UriMatcher uriMatcher =
            new UriMatcher(UriMatcher.NO_MATCH);

    static {
        uriMatcher.addURI(Contract.AUTHORITY, Contract.USERS, USERS);
        uriMatcher.addURI(Contract.AUTHORITY, Contract.USERS+"/#", USERS_ID);
    }

    @Override
    public Cursor query(Uri uri, String[] projection,
                        String selection, String[] selectionArgs,
                        String sortOrder) {
        switch (uriMatcher.match(uri)) {
            case ALARMS: /* ... */ break;
            case ALARMS_ID:
                String id = uri.getLastPathSegment();
                /* ... */ break;
            default: throw new IllegalArgumentException("Unknown URI " + uri);
        } /* ... */
    }
}
```

Type MIME (Multipurpose Internet Mail Extensions)

```
public class UsersProvider extends ContentProvider {
    /* ... */

    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)) {
            case USERS:
                return "vnd.android.cursor.dir/vnd.com.univamu.cci.users";
            case USERS_ID:
                return "vnd.android.cursor.item/vnd.com.univamu.cci.users";
            default:
                throw new IllegalArgumentException("Unknown URI " + uri);
        }
    }
}
```

- Description du type des données retournées par un URI
- Format : vnd.android.cursor.[dir|item]/vnd.<name>.<type>

Adaptateur utilisant un fournisseur de contenu

```
public class UsersAdapter extends RecyclerView.Adapter<UserViewHolder> {
    private Cursor cursor;

    @Override
    public void onBindViewHolder(UserViewHolder viewHolder, int position) {
        if (!cursor.moveToPosition(position)) { return; }
        viewHolder.bind(cursor);
    }

    @Override
    public int getItemCount() {
        return cursor == null ? 0 : cursor.getCount();
    }

    public void swapCursor(Cursor cursor) {
        if (this.cursor == cursor) { return; }
        if (this.cursor != null) { this.cursor.close(); }
        this.cursor = cursor;
        notifyDataSetChanged();
    }

    /* ... */
}
```

Chargement des curseurs

```
public class UserFragment extends Fragment {
    private UsersAdapter adapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getLoaderManager().initLoader(0, null, new LoaderCallbacks());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        adapter = new UsersAdapter();
        /* ... */
    }

    private class LoaderCallbacks
        implements LoaderManager.LoaderCallbacks<Cursor> {
        /* ... */
    }
}
```

Chargement des curseurs

```
public class UserFragment extends Fragment {
    private UsersAdapter adapter;

    /* ... */

    private class LoaderCallbacks
        implements LoaderManager.LoaderCallbacks<Cursor> {
        @Override
        public Loader onCreateLoader(int id, Bundle args) {
            return new CursorLoader(getActivity(), Contract.CONTENT_URI,
                null, null, null, null);
        }

        @Override
        public void onLoadFinished(Loader loader, Cursor data) {
            adapter.swapCursor(data);
        }

        @Override
        public void onLoaderReset(Loader loader) { adapter.swapCursor(null); }
    }
}
```

Notification des modifications

```
public class UsersProvider extends ContentProvider {

    @Override
    public Cursor query(Uri uri, String[] projection,
                        String selection, String[] selectionArgs,
                        String sortOrder) {
        Cursor cursor = /* .... */
        cursor.setNotificationUri(getContext().getContentResolver(), uri);
        return cursor;
    }

    @Override
    public Cursor update(Uri uri, ContentValues values,
                        String selection, String[] selectionArgs) {
        getContext().getContentResolver().notifyChange(uri, null);
    }

    /* ... */
}
```

Le curseur sera redemandé à chaque modification

Animation dans les RecyclerView

```
public class UsersAdapter extends RecyclerView.Adapter<UserViewHolder> {  
    private Cursor cursor;  
  
    public UsersAdapter() { setHasStableIds(true); }  
  
    @Override  
    public long getItemId(int position) {  
        if (cursor == null || !cursor.moveToPosition(position)) {  
            return RecyclerView.NO_ID;  
        }  
        return cursor.getLong(cursor.getColumnIndex(Contract.Users._ID));  
    }  
  
    /* ... */  
}
```

Les RecyclerView peuvent animer la modification de la liste :

- Les identifiants doivent être stables
- Il faut implémenter la méthode `getItemId`

Services

- Un service est un composant
- Il ne fournit pas d'interface utilisateur
- Il peut exécuter des tâches de fond
- Un service peut être démarré par une autre application
- Un composant peut s'attacher (bind) à un service
- Cela lui permet de communiquer avec le service
- Entre applications, communications inter-processus (IPC)

Les deux types de service

"Started"

Le service est démarré en utilisant `startService` :

```
Intent intent = new Intent(context, DownloadService.class);
intent.setData(Uri.parse(url));
context.startService(downloadIntent);
```

"Bound"

Le service est attaché au composant en utilisant `bindService` :

```
Intent intent = new Intent(this, LocalService.class);

connection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) { ... }

    @Override
    public void onServiceDisconnected(ComponentName name) { ... }
};

context.bindService(intent, connection, Context.BIND_AUTO_CREATE);
```

Création d'un service

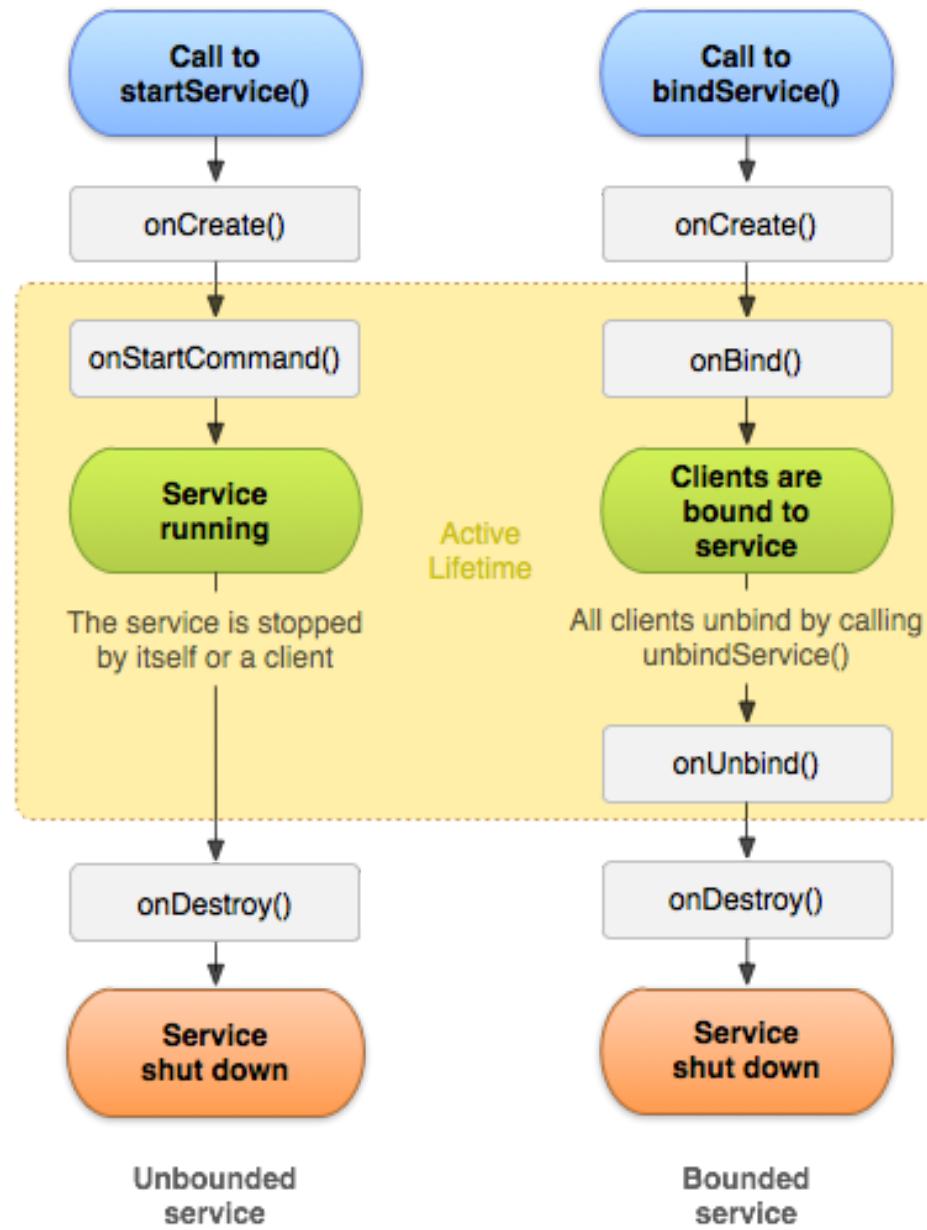
```
public class MyService extends Service {  
    @Override  
    public void onCreate() { ... }  
  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        // Pour redémarrer automatiquement si le service est tué :  
        return START_STICKY;  
    }  
  
    @Override  
    public IBinder onBind(Intent intent) { ... }  
  
    @Override  
    public boolean onUnbind(Intent intent) { ... }  
  
    @Override  
    public void onRebind(Intent intent) { ... }  
  
    @Override  
    public boolean stopService(Intent intent) {  
        ...  
        return super.stopService(intent);  
    }  
  
    @Override  
    public void onDestroy() { ... }
```

```
}
```

Déclaration dans le manifeste

```
<manifest ...>
    <application ...>
        ...
        <service android:name=".MyService" android:exported="false"/>
    </application>
</manifest>
```

Cycle de vie d'un service



Arrêt d'un service "started"

- Le système peut arrêter le service s'il a besoin de mémoire
- Le système peut s'arrêter lui-même en utilisant `stopSelf()`
- Un composant peut demander l'arrêt du service :

```
Intent intent = new Intent(context, MyService.class);
context.stopService(intent);
```

`stopService(Intent intent)` du service est alors appelée

- Il est important de ne pas arrêter un service qui travaille

Les `startId` de `onStartCommand` peuvent être conservés :

```
public int onStartCommand(Intent intent, int flags, int startId)
```

Ensuite, on peut utiliser `stopSelf(startId)` pour préciser à quelle requête correspond la demande d'arrêt.

Création d'un service "bound" local

```
public class LocalService extends Service {
    private final IBinder binder = new LocalBinder();

    private final Random generator = new Random();

    public class LocalBinder extends Binder {
        LocalService getService() {
            return LocalService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    public int getRandomNumber() {
        return generator.nextInt(100);
    }
}
```

Connexion à un service de la même application

```
public class MyActivity extends Activity {
    LocalService service;
    boolean bound = false;

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, LocalService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (bound) { unbindService(connection); bound = false; }
    }

    /* ... */
}
```

Connexion à un service de la même application

```
public class MyActivity extends Activity {
    LocalService service;
    boolean bound = false;

    /* ... */

    private ServiceConnection connection = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName className,
                                       IBinder iBinder) {
            LocalService.LocalBinder binder =
                (LocalService.LocalBinder) iBinder;
            MyActivity.this.service = binder.getService();
            bound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName className) {
            bound = false;
        }
    };
}
```

Utilisation du service

```
public class MyActivity extends Activity {
    LocalService service;
    boolean bound = false;

    /* ... */

    public int getNumberFromService() {
        return (bound) ? service.getRandomNumber() : 0;
    }

    /* ... */
}
```

Création d'un service "bound" accessible d'une autre application

```
public class MessengerService extends Service {
    private final Random generator = new Random();

    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            int number = generator.nextInt(msg.arg2 - msg.arg1 + 1) + msg.arg1;
            /* TODO : envoyer la réponse */
        }
    }

    final Messenger messenger = new Messenger(new IncomingHandler());

    @Override
    public IBinder onBind(Intent intent) {
        return messenger.getBinder();
    }
}
```

Connexion au service

```
public class MyActivity extends Activity {
    Messenger service;
    boolean bound = false;

    @Override
    protected void onStart() {
        super.onStart();
        Intent intent = new Intent(this, MessengerService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (bound) { unbindService(connection); bound = false; }
    }

    /* ... */
}
```

Connexion au service

```
public class MyActivity extends Activity {
    Messenger service;
    boolean bound = false;

    /* ... */

    private ServiceConnection connection = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName className,
                                       IBinder iBinder) {
            MyActivity.this.service = new Messenger(iBinder);
            bound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName className) {
            bound = false;
        }
    };
}
```

Communication avec le service

```
public class MyActivity extends Activity {
    Messenger service;
    boolean bound = false;
    /* ... */
    public void getNumberFromService() {
        if (bound) {
            Message msg = Message.obtain(null, 0, 20, 100);
            try { service.send(msg); }
            catch (RemoteException e) { e.printStackTrace(); }
        }
    }
    /* ... */
}
```

```
public class MessengerService extends Service {
    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            int number = generator.nextInt(msg.arg2 - msg.arg1 + 1) + msg.arg1;
            /* TODO : envoyer la réponse */
        }
    }
}
```

Ajout d'un replyTo pour recevoir la réponse

```
public class MyActivity extends Activity {
    Messenger service;
    boolean bound = false;

    /* ... */

    public void getNumberFromService() {
        if (bound) {
            Message msg = Message.obtain(null, 0, 20, 100);
            try { msg.replyTo = messenger; service.send(msg); }
            catch (RemoteException e) { e.printStackTrace(); }
        }
    }

    class ResultHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            Log.d("MyActivity", "Result : "+msg.arg1);
        }
    }

    final Messenger messenger = new Messenger(new ResultHandler());
}
```

Communication de la réponse par le service

```
public class MessengerService extends Service {
    private final Random generator = new Random();

    class IncomingHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            int number = generator.nextInt(msg.arg2 - msg.arg1 + 1) + msg.arg1;
            Message resultMessage = Message.obtain(null, 0, number, 0);
            try {
                msg.replyTo.send(resultMessage);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }

    /* ... */
}
```

Service en premier plan

- Un service est **foreground** si l'utilisateur est conscient de son existence (lecture d'une musique, téléchargement en cours, etc.)
- Une notification est affiché à l'utilisateur pour lui permettre d'interagir avec le service (passer à la musique suivante, etc.)
- Un service **foreground** n'est pas un candidat pour libérer de la mémoire (de la même façon qu'une activité en premier plan)

Service en premier plan

Passage en premier plan :

```
Notification notification = new Notification.Builder(this)
    .setContentTitle("My notification")
    .setContentText("Content text")
    .setSmallIcon(R.drawable.icon)
    .build();

Intent notificationIntent = new Intent(this, MyActivity.class);
PendingIntent pendingIntent =
    PendingIntent.getActivity(this, 0, notificationIntent, 0);
startForeground(NOTIFICATION_ID, notification);
```

Sortie du premier plan :

```
stopForeground(true /* pour supprimer la notification */);
```

Cycle de vie des processus

Android classe les processus par ordre d'importance :

1. Foreground process

Activité en cours d'utilisation par l'utilisateur

Service connecté à une activité en cours d'utilisation

Service qui a appelé `startForeground()`

Service qui exécute une de ses callbacks

BroadcastReceiver qui exécute `onReceive`

2. Visible process

Activité visible

Service connecté à une activité visible

Cycle de vie des processus

3. Service process

Service qui a été démarré avec startService

4. Background process

Contient des composants

5. Empty process

Ne contient rien

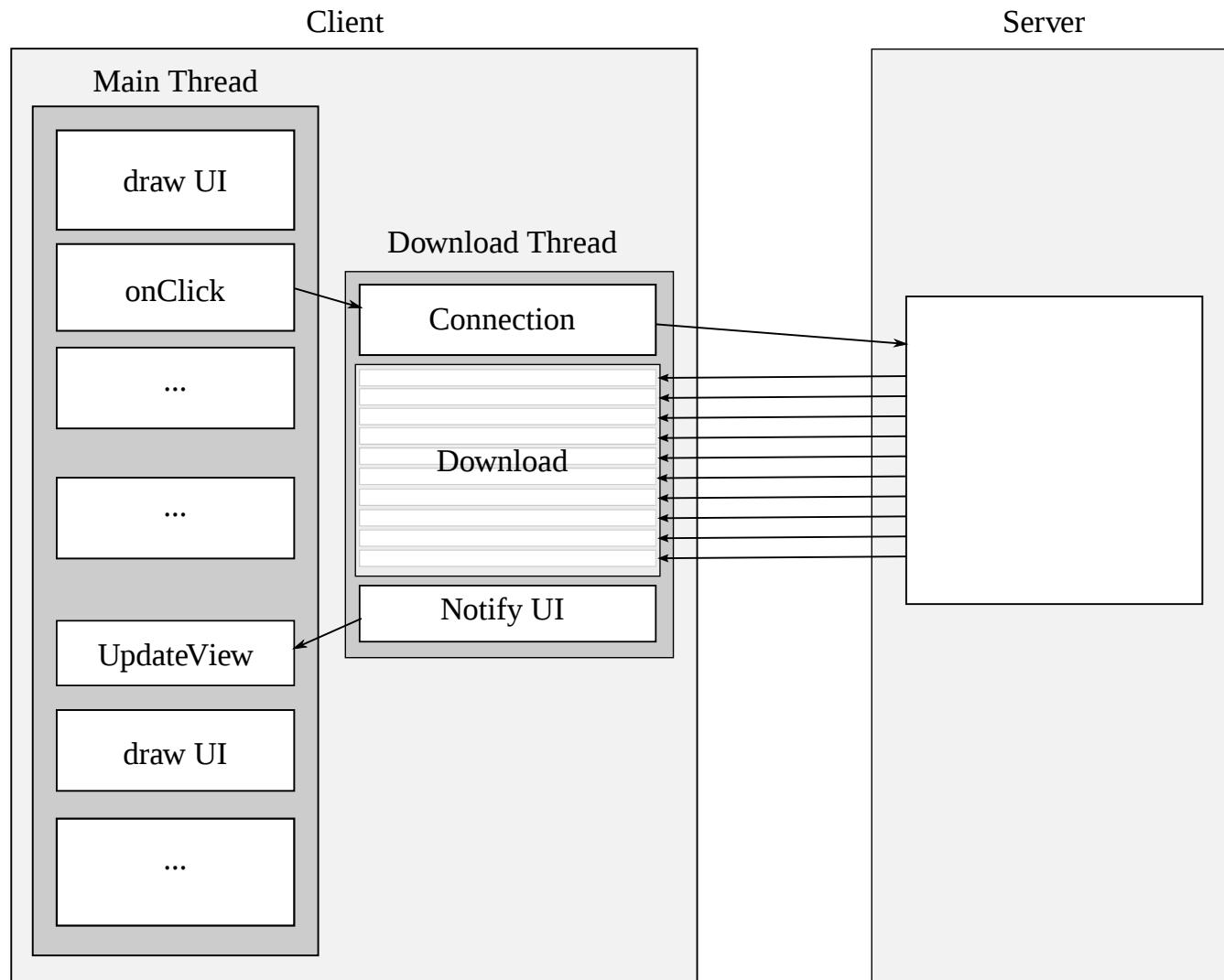
Les threads

- Les threads sont des processus léger ou fils d'exécution
- Les threads s'exécutent (ou semblent s'exécuter) en parallèle
- Les threads d'un même processus partagent une zone mémoire
- Sous Android, une application n'utilise qu'un thread principal
- Seul le thread principal peut modifier les vues de l'interface
- Le thread principal gère la queue d'événements

Pourquoi créer des threads ?

Pour ne jamais bloquer le thread principal

Un exemple



L'interface Runnable et la classe Thread

L'interface Runnable :

```
public interface Runnable { void run(); }
```

Définition d'une tâche à exécuter :

```
public class DownloadTask implements Runnable {
    public void run() { /* TODO Download */ }
}
```

Création d'un nouveau thread :

```
Thread thread = new Thread(new DownloadTask());
```

Exécution du thread :

```
thread.start();
```

Synchronisation avec le thread principal

Seul le thread principal peut modifier les vues de l'interface :

```
public class DownloadTask implements Runnable {
    private final ImageView imageView;

    public DownloadTask(ImageView imageView) {
        this.imageView = imageView;
    }

    public void run() {
        Bitmap bitmap = loadImage("http://example.com/a.jpg");
        /* INTERDIT : imageView.setImageBitmap(bitmap); */
    }
}
```

Synchronisation avec le thread principal

Réintégration dans le thread principal :

```
public void run() {  
    Bitmap bitmap = loadImage("http://example.com/a.jpg");  
    imageView.post(new Runnable() {  
        public void run() { imageView.setImageBitmap(bitmap); }  
    });  
}
```

Pour exécuter des tâches dans le thread principal :

- `Activity.runOnUiThread(Runnable)`
- `View.post(Runnable)`
- `View.postDelayed(Runnable, long)`

La classe AsyncTask

Simplification de l'écriture d'un thread avec AsyncTask :

```
public class ImageDownloader extends AsyncTask<URL, Integer, Bitmap> {
    private ImageView view;

    public ImageDownloader(ImageView imageView) { this.view = imageView; }

    @Override
    protected Bitmap doInBackground(URL... urls) {
        return downloadImage(urls[0]);
    }

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        view.setImageBitmap(bitmap);
    }
}
```

La classe AsyncTask

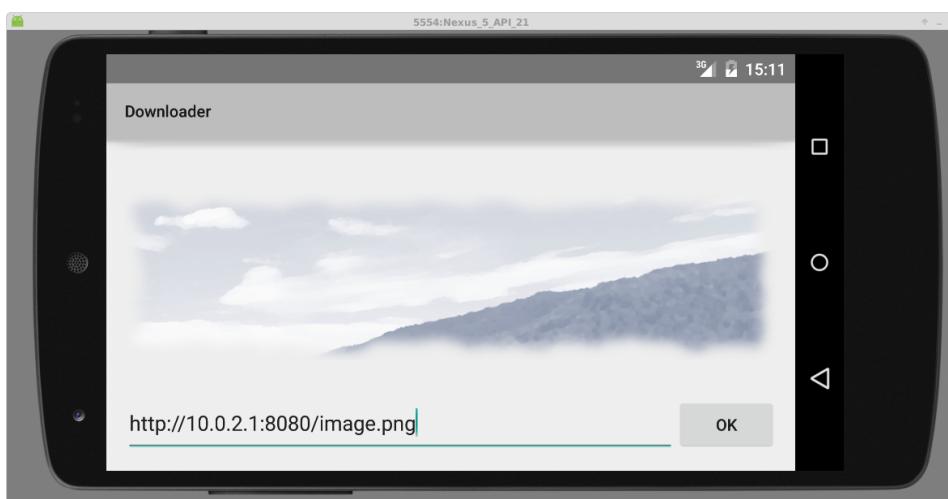
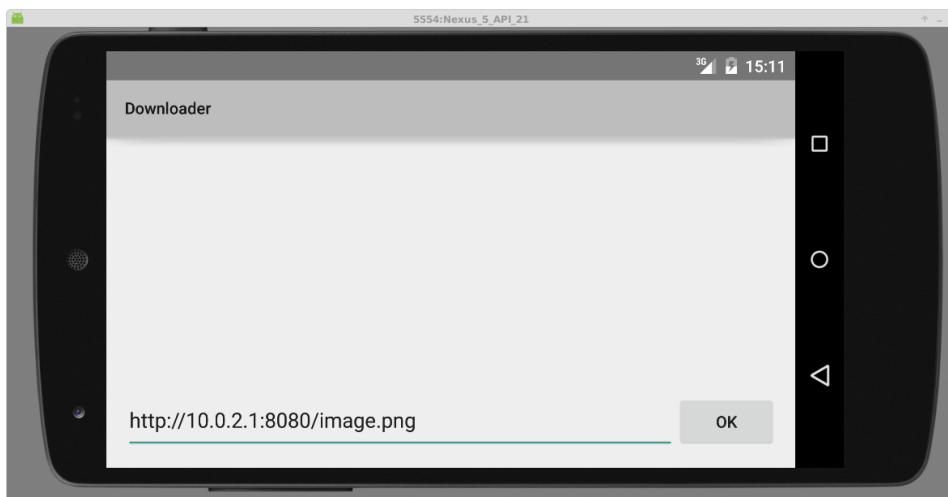
Utilisation de la classe ImageDownloader :

```
ImageDownloader imageDownloader = new ImageDownloader(imageView);  
imageDownloader.execute(url);
```

Les autres méthodes de AsyncTask :

- void onPreExecute()
- void onProgressUpdate(Progress... values)
- void publishProgress(Progress... values)
- void onCancelled(Result result)
- boolean cancel(boolean mayInterruptIfRunning)
- boolean isCancelled()

Exemple - Connexion réseau



Exemple - Connexion réseau

Nous devons :

- Ajouter une permission à l'application
- Créer un thread
- Ouvrir une connexion
- Lire les données provenant d'un flux
- Injecter les données téléchargées dans l'application

Exemple - Connexion réseau

Ajout des droits dans le manifeste de l'application :

```
<manifest
    xmlns:android="...."
    package="com.univ_amu.cci.imagedownloader" >

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

    <!-- ... -->
</manifest>
```

Exemple - Connexion réseau

Création de la tâche permettant le téléchargement de l'image :

```
public class ImageDownloader extends AsyncTask<URL, Integer, Bitmap> {
    private ImageView imageView;

    public ImageDownloader(ImageView imageView) {
        this.imageView = imageView;
    }

    @Override
    protected Bitmap doInBackground(URL... urls) {
        URL url = urls[0];
        try (InputStream stream = url.openConnection().getInputStream()) {
            Bitmap bitmap = BitmapFactory.decodeStream(stream);
            return bitmap;
        } catch (IOException e) { return null; }
    }

    /* ... */
}
```

Exemple - Connexion réseau

Création de la tâche permettant le téléchargement de l'image :

```
public class ImageDownloader extends AsyncTask<URL, Integer, Bitmap> {
    private ImageView imageView;

    public ImageDownloader(ImageView imageView) {
        this.imageView = imageView;
    }

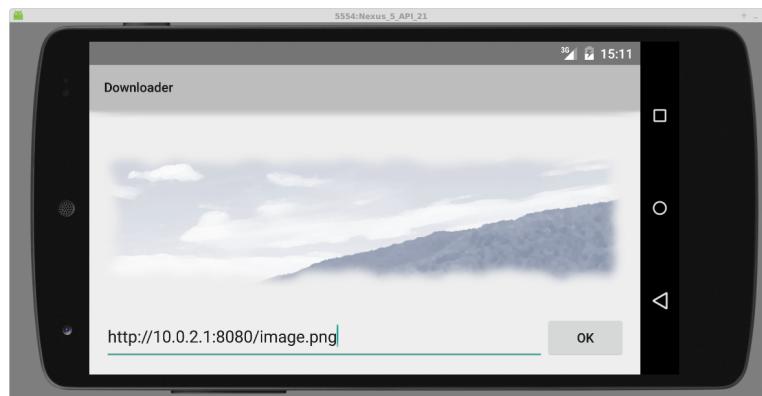
    @Override
    protected Bitmap doInBackground(URL... urls) {
        /* ... */
    }

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (bitmap!=null)
            imageView.setImageBitmap(bitmap);
    }
}
```

Exemple - Connexion réseau

L'activité principale :

```
public class MainActivity extends Activity {  
    private ImageView imageView;  
    private EditText urlEditText;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        imageView = (ImageView) findViewById(R.id.imageView);  
        urlEditText = (EditText) findViewById(R.id.urlEditText);  
    }  
}
```



Exemple - Connexion réseau

L'activité principale :

```
public class MainActivity extends Activity {
    private ImageView imageView;
    private EditText urlEditText;

    /* ... */

    public void onClick(View view) {
        ImageDownloader imageDownloader = new ImageDownloader(imageView);
        try {
            URL url = new URL(urlEditText.getText().toString());
            imageDownloader.execute(url);
        } catch (MalformedURLException e) {
            Toast.makeText(this,
                getString(R.string.malformed_url),
                Toast.LENGTH_LONG).show();
        }
    }
}
```

Services et threads

Nous voulons créer un service pour télécharger des images

Ce service doit :

- pouvoir être utilisé à partir d'une autre application
- "broadcaster" le fait qu'il a terminé un téléchargement
- Ne pas effectuer les téléchargements sur le thread principal
- Traiter de façon séquentielle les téléchargements

IntentService

La classe IntentService étend Service et permet de réaliser cela :

```
public class MyIntentService extends IntentService {  
  
    public MyIntentService() {  
        super("MyIntentService");  
    }  
  
    @Override  
    protected void onHandleIntent(Intent intent) {  
        try {  
            Thread.sleep(5000);  
        } catch (InterruptedException e) {  
            Thread.currentThread().interrupt();  
        }  
    }  
}
```

Le service est arrêté quand tous les traitements ont été effectués

Communication de l'URL à télécharger

```
public class DownloaderService extends IntentService {
    public static final String URL = "url";
    public static final String FILENAME = "filename";

    public DownloaderService() {
        super("DownloaderService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        String intentUrl = intent.getStringExtra(URL);
        String intentFilename = intent.getStringExtra(FILENAME);
        download(intentUrl, intentFilename);
    }
}
```

```
Intent intent = new Intent(this, DownloaderService.class);
intent.putExtra(DownloaderService.URL, "http://....");
intent.putExtra(DownloaderService.FILENAME, "image.jpg");
startService(intent);
```

Téléchargement

```
private void download(String intentUrl, String intentFilename) {  
    try {  
        URL inputUrl = new URL(intentUrl);  
        File outputFile = new File(  
            Environment.getExternalStoragePublicDirectory(  
                Environment.DIRECTORY_DOWNLOADS  
)  
,  
            intentFilename);  
        try (InputStream inputStream =  
                inputUrl.openConnection().getInputStream();  
                OutputStream outputStream = new FileOutputStream(outputFile))  
        {  
            copy(inputStream, outputStream);  
            broadcastSuccess(intentUrl, outputFile.getAbsolutePath());  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
        broadcastError();  
    }  
}
```

Téléchargement

```
private void copy(InputStream inputStream,
                  OutputStream outputStream) throws IOException {
    byte[] buffer = new byte[1024];
    for (;;) {
        int bytesCount = inputStream.read(buffer);
        if (bytesCount == -1) break;
        outputStream.write(buffer, 0, bytesCount);
    }
}
```

Téléchargement

```
public static final int DOWNLOAD_SUCCESS = 1;
public static final int DOWNLOAD_ERROR = 2;
public static final String URL = "url";
public static final String FILENAME = "filename";
public static final String RESULT_CODE = "result_code";
public static final String DONE_ACTION =
    "com.univ_amu.cci.downloader.DONE_ACTION";
```

```
private void broadcastSuccess(String url, String filename) {
    Intent intent = new Intent(DONE_ACTION);
    intent.putExtra(URL, url);
    intent.putExtra(FILENAME, filename);
    intent.putExtra(RESULT_CODE, DOWNLOAD_SUCCESS);
    sendBroadcast(intent);
}
```

```
private void broadcastError() {
    Intent intent = new Intent(DONE_ACTION);
    intent.putExtra(RESULT_CODE, DOWNLOAD_ERROR);
    sendBroadcast(intent);
}
```

Le manifeste de l'application

```
<manifest ...>

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

    <application ...>
        <service android:name="fr.univamu.cci.downloader.DownloaderService"
            android:exported="true"/>
    </application>

</manifest>
```

Appel au service dans la même application

```
public class MainActivity extends Activity {
    private ImageView imageView;
    private EditText editText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
        imageView = (ImageView) findViewById(R.id.image_view);
        editText = (EditText) findViewById(R.id.edit_text);
    }

    public void onClick(View view) {
        Uri uri = Uri.parse(editText.getText().toString());
        Intent intent = new Intent(this, DownloaderService.class);
        intent.putExtra(DownloaderService.URL, uri.toString());
        intent.putExtra(DownloaderService.FILENAME,
                        uri.getLastPathSegment());
        startService(intent);
    }
}
```

Appel au service dans la même application

```
public class MainActivity extends Activity {
    private ImageView imageView;
    private EditText editText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
        imageView = (ImageView) findViewById(R.id.image_view);
        editText = (EditText) findViewById(R.id.edit_text);
    }

    public void onClick(View view) {
        Uri uri = Uri.parse(editText.getText().toString());
        Intent intent = new Intent(this, DownloaderService.class);
        intent.putExtra(DownloaderService.URL, uri.toString());
        intent.putExtra(DownloaderService.FILENAME,
                        uri.getLastPathSegment());
        startService(intent);
    }
}
```

Écoute des "broadcasts" du service

```
public class MainActivity extends Activity {  
    @Override  
    protected void onResume() {  
        super.onResume();  
        IntentFilter filter = new IntentFilter();  
        filter.addAction(DownloaderService.DONE_ACTION);  
        registerReceiver(receiver, filter);  
    }  
  
    @Override  
    protected void onPause() {  
        super.onPause();  
        unregisterReceiver(receiver);  
    }  
  
    final private BroadcastReceiver receiver = new BroadcastReceiver() {  
        @Override  
        public void onReceive(Context context, Intent intent) {  
            /* TODO */  
        }  
    };  
}
```

Implémentation de la méthode onReceive

```
public class MainActivity extends Activity {

    final private BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            switch (intent.getIntExtra(DownloaderService.RESULT_CODE, 0)) {
                case DownloaderService.DOWNLOAD_SUCCESS:
                    String url =
                        intent.getStringExtra(DownloaderService.URL);
                    String filename =
                        intent.getStringExtra(DownloaderService.FILENAME);
                    setBackgroundImage(url, filename);
                    break;
                case DownloaderService.DOWNLOAD_ERROR:
                    Toast.makeText(getApplicationContext(),
                        R.string.download_error, Toast.LENGTH_LONG).show();
                    break;
            }
        }
    };
}
```

Changement de l'image de fond de la vue

```
public class MainActivity extends Activity {  
  
    private void setBackgroundImage(String url, String filename) {  
        if (!editText.getText().toString().equals(url)) return;  
        Bitmap bitmap = BitmapFactory.decodeFile(filename);  
        imageView.setImageBitmap(bitmap);  
    }  
}
```

Utilisation du service de l'extérieur de l'application

```
public class MainActivity extends Activity {

    public void onClick(View view) {
        Uri uri = Uri.parse(editText.getText().toString());
        Intent intent = new Intent();
        intent.setComponent(new ComponentName("fr.univamu.cci.downloader",
                "fr.univamu.cci.downloader.DownloaderService"));
        intent.putExtra(DownloaderService.URL, uri.toString());
        intent.putExtra(DownloaderService.FILENAME, uri.getLastPathSegment());
        startService(intent);
    }

}
```

```
<manifest ...>

    <application ...>
        <service android:name="fr.univamu.cci.downloader.DownloaderService"
            android:exported="true"/>
    </application>

</manifest>
```