

Programmation Mobile – Android – Master CCI

Bertrand Estellon

Aix-Marseille Université

March 23, 2015

Organisation de l'UE

► Objectifs du cours :

- Faire un petit tour du framework Android ;
- Apprendre à organiser une application ;
- Apprendre à utiliser une documentation ;
- Progresser en Java.

► Limites du cours :

- Le cours est essentiellement basé sur des exemples pratiques et ne présente pas toutes les fonctionnalités d'Android ;
- Vous devrez donc chercher dans la documentation afin de trouver les informations nécessaires à la réalisation de certaines parties des TP.

► Evaluation :

- Seuls les TP seront évalués aux cours des séances ;
- Il est important de venir à tous les TP ;
- Les notes sont individuelles ;
- Vous devez donc travailler seul.

Android SDK

On peut développer sur Android sous Linux, Windows et MacOS avec :

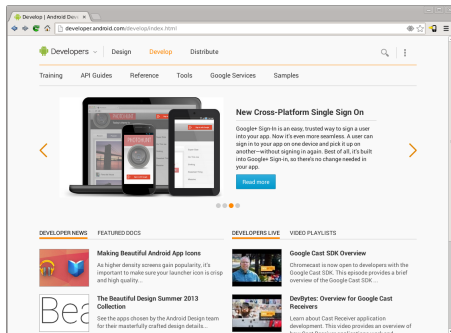
- ▶ un plugin pour Eclipse ;
- ▶ Android-Studio.

Ces deux solutions utilisent le SDK d'Android qui contient :

- ▶ Les librairies Java d'Android ;
- ▶ Des outils de développement ;
- ▶ un émulateur pour tester vos applications ;
- ▶ des images du système Android...

Android SDK

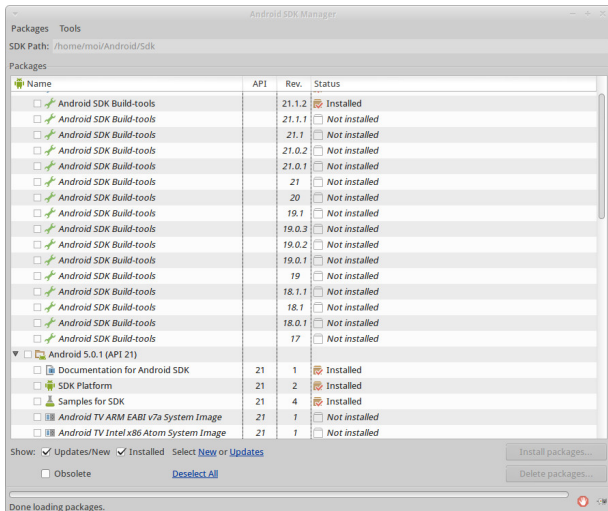
L'Android SDK peut être téléchargé sur le site dédié aux développeurs.



Il n'est pas nécessaire de télécharger le SDK séparément si vous vous utilisez Android Studio.

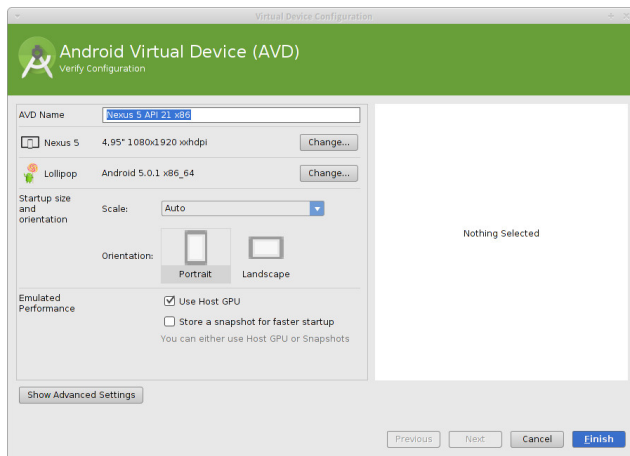
Android SDK

Vous pouvez exécuter le *Android SDK Manager*, en lançant l'exécutable *android* qui se trouve dans le répertoire *tools* du SDK :



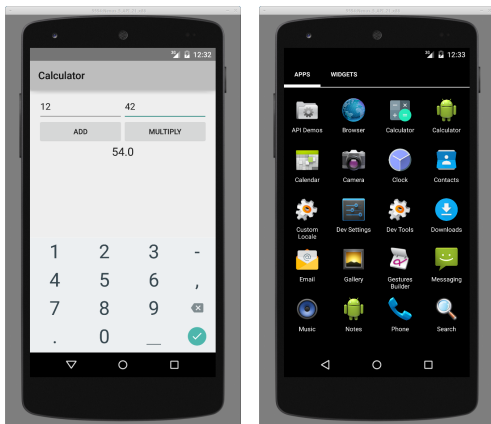
AVD Manager

Pour tester un programme sur votre ordinateur, il sera également nécessaire de créer une machine virtuelle à l'aide de l'*AVD Manager* :



Émulateur

L'émulateur vous permet de tester votre application :



Les ROM Intel permettent de bénéficier de l'accélération matérielle.

Android Studio

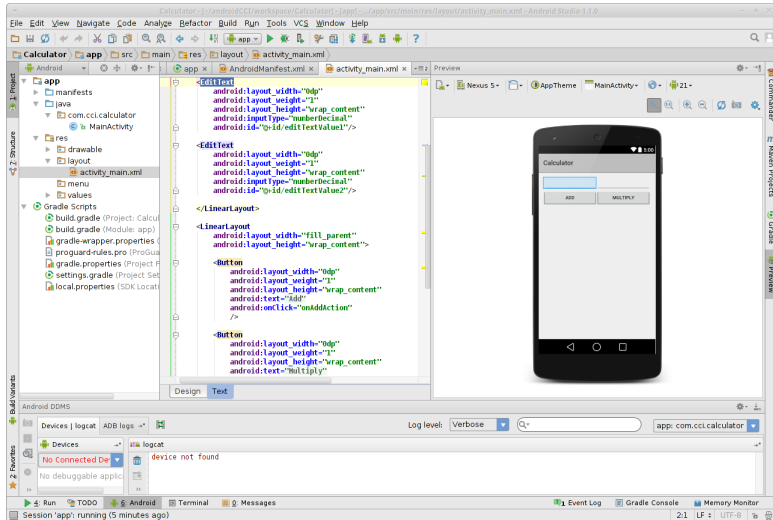
Pour installer Android Studio :

- ▶ Télécharger le logiciel via le site d'Android ;
- ▶ Extraire l'archive dans un répertoire ;
- ▶ Exécuter le fichier `studio.sh` du répertoire `bin` ;

À son premier démarrage, Android Studio :

- ▶ télécharge le `sdk` ;
- ▶ va vous proposer de configurer une machine virtuelle.

Android Studio




Android SDK et Eclipse

En TP, nous allons utiliser Android Studio :

- ▶ Il est installé dans mon répertoire personnel ;
- ▶ Il reste à configurer le lien vers le SDK d'Android ;
- ▶ Il n'est pas possible d'installer de nouveaux packages.

Création d'une application

Create New Project

 **New Project**
Android Studio

Configure your new project

Application name:

Company Domain:

Package name: [Edit](#)

Project location: ...

Previous Next Cancel Finish

Création d'une application

Create New Project

Target Android Devices

Select the form factors your app will run on

Different platforms require separate SDKs

☒ Phone and Tablet

Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately 90,4% of the devices that are active on the Google Play Store. [Help me choose.](#)

☐ TV

Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ Wear

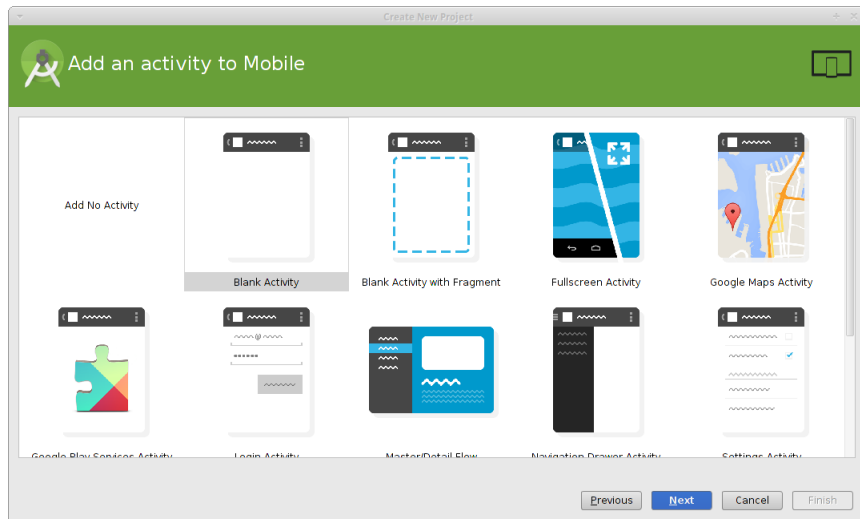
Minimum SDK: API 21: Android 5.0 (Lollipop)

☐ Glass (Not Installed)

Minimum SDK:

Previous Next Cancel Finish

Création d'une application




Création d'une application

Create New Project

Customize the Activity

Creates a new blank activity with an action bar.

 Blank Activity

Activity Name:

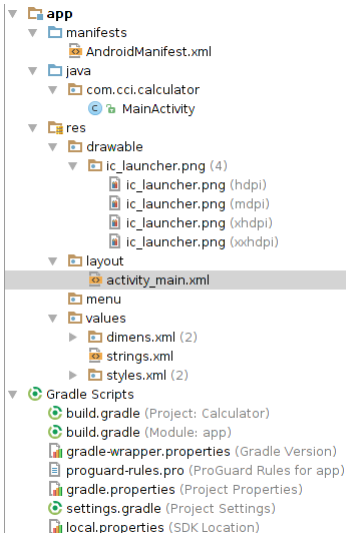
Layout Name:

Title:

Menu Resource Name:

The name of the activity class to create

Structure d'une application



App manifest

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

Les composants d'une application

Une application peut être composée :

- ▶ d'*activités* : interface de l'application ;
- ▶ de *services* : “tâches” en arrière-plan de l'application ;
- ▶ de *fournisseurs de contenu* : accès à/partage de données structurées ;
- ▶ de *widgets* : intégrable dans d'autres applications (écran d'accueil...).

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 manifest

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

Activité

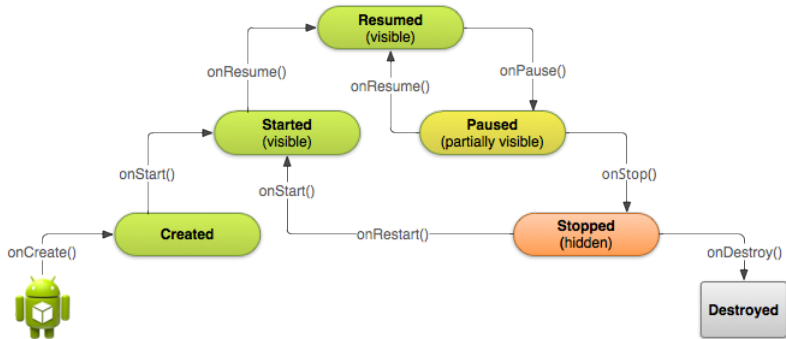


Cycle de vie d'une activité

Généralités sur le 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` ;

Cycle de vie “pyramidal”



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

Les états d'une activité

Les trois 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 exécutée de code. En revanche, toutes ses informations sont conservées.

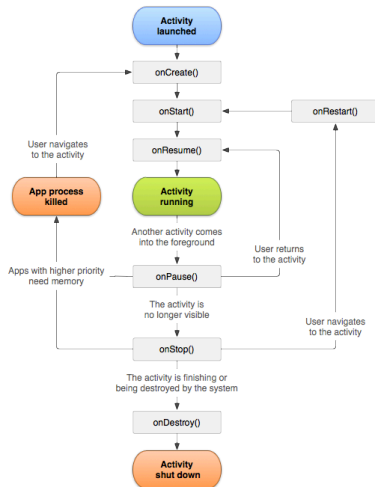
Les deux états transitoires d'une activité :

- ▶ Created : L'activité vient d'être créée.
- ▶ Started : L'activité vient de devenir visible.

Cycle de vie des activités

```
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)

Pourquoi implémenter ces méthodes ?

Cela est important afin que votre application fonctionne correctement :

- ▶ Réception d'une appel et basculement sur une autre application ;
- ▶ Ne pas consommer trop de ressources système ;
- ▶ Ne pas avoir de problème lors de la création/restauration de l'application par le système (lors d'une rotation de l'écran par exemple).

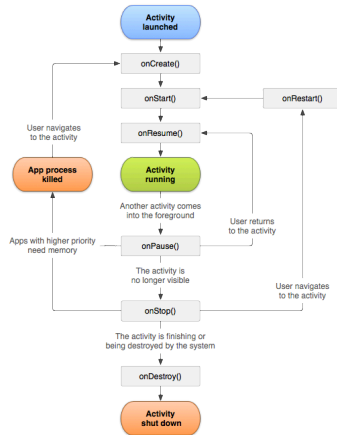
Remarque : avant de distribuer une application, Il est donc important d'avoir fait des tests dans les situations évoquées ci-dessus.

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 :



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

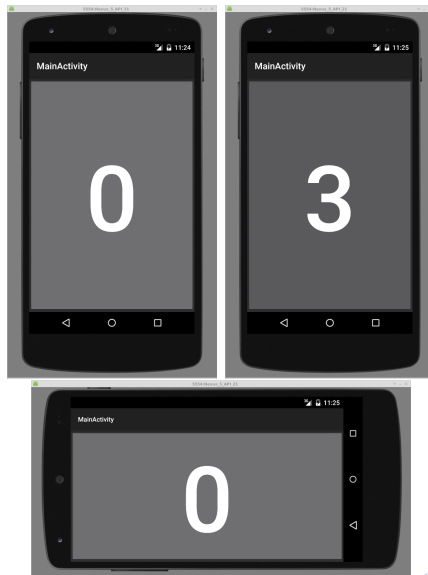
Sauvegarde de l'état de l'application

```
public class MainActivity extends Activity {  
    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);  
    }  
    /* ... */  
}
```

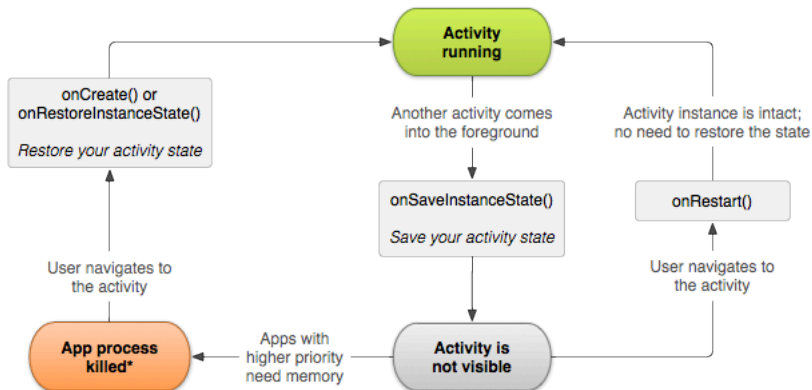
Sauvegarde de l'état de l'application

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



*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved

(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, etc.) ;
- ▶ des chaînes de caractères ;
- ▶ des tableaux et des listes ;
- ▶ des objets Parcelable.

Quelques remarques :

- ▶ Un objet Parcelable peuvent être écrit dans un objet Parcel ;
- ▶ Une instance de Parcel est un conteneur pour un message ;
- ▶ Un Bundle est Parcelable.

Les Bundles

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

La structure d'une interface avec l'utilisateur :

- ▶ Les interfaces sont composées d'objets étendant View et ViewGroup ;
- ▶ Une vue se dessine et permet les interactions avec l'utilisateur ;
- ▶ Un groupe de vues contient d'autres vues ;
- ▶ Un groupe de vues organise l'affichage des vues qu'il contient ;
- ▶ Un groupe de vues est une vue.

Android fournit une collection de vues et de groupes de vues qui offrent :

- ▶ les principales entrées (textes, champs de texte, listes, etc.) ;
- ▶ différents modèles d'organisation (linéaire, etc.).

Une classe est associée à chaque type de vue ou groupe de vue.

Les vues

La structure d'une interface avec l'utilisateur :

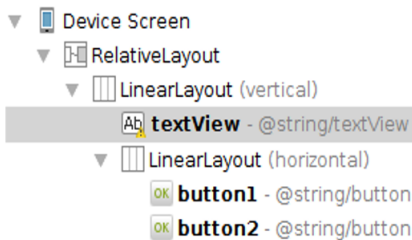
- ▶ Les interfaces sont composées d'objets étendant View et ViewGroup ;
- ▶ Une vue se dessine et permet les interactions avec l'utilisateur ;
- ▶ Un groupe de vues contient d'autres vues ;
- ▶ Un groupe de vues organise l'affichage des vues qu'il contient ;
- ▶ Un groupe de vues est une vue.

Android fournit une collection de vues et de groupes de vues qui offrent :

- ▶ les principales entrées (textes, champs de texte, listes, etc.) ;
- ▶ différents modèles d'organisation (linéaire, etc.).

Une classe est associée à chaque type de vue ou groupe de vue.

Les vues



Construction d'une vue en Java

```
public class MainActivity extends Activity {  
    @Override  
    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;  
    }  
}
```

Construction d'une vue en Java

```
public class MainActivity extends Activity {  
  
    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;  
    }  
  
}
```

Construction d'une vue en Java

```
public class MainActivity extends Activity {  
  
    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 Listeners 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 évènements

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

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

De façon générale :

- ▶ 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.

Les évènements

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++;  
    }  
}
```

Les évènements

Utilisation d'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++; }  
    }  
}
```

Les évènements

Utilisation d'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 évènements

En Java 8 (pas encore supporté par le framework Android) :

```
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(v->count++);  
        setContentView(button);  
    }  
}
```

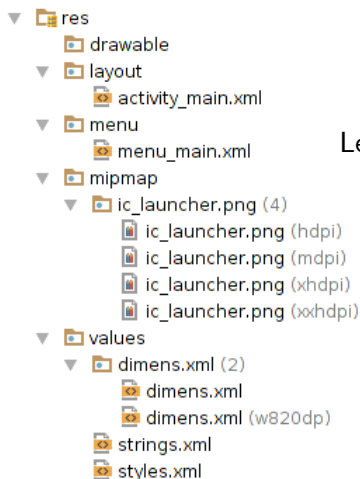
L'utilisation des lambda-expressions simplifiera la gestion des évènements.

Les ressources

Les ressources permettent :

- ▶ d'externaliser :
 - ▶ les images ;
 - ▶ les chaînes de caractères ;
 - ▶ les modèles d'interfaces utilisateur ;
 - ▶ les animations...
- ▶ de fournir des ressources différentes en fonction de la configuration :
 - ▶ taille de l'écran ;
 - ▶ orientation de l'écran ;
 - ▶ langue...
- ▶ de maintenir ces éléments sans connaître Java.

Les ressources

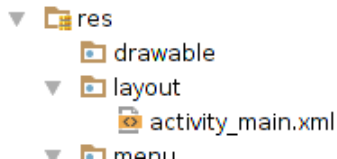


Les ressources :

- ▶ sont dans le répertoire res ;
- ▶ sont organisées dans des répertoires ;
- ▶ contiennent des images, fichiers XML... ;

Les ressources

- ▶ Le framework Android génère et maintient à jour automatiquement une classe R contenant les identifiants (valeur entière) des ressources ;
- ▶ Ils permettent de désigner les ressources dans le code Java ;
- ▶ Par exemple, le fichier `activity_main.xml` a comme identifiant `R.layout.activity_main` :



Les layouts

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

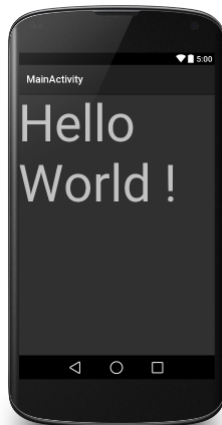
```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
}
```

- ▶ Un layout est une ressource au format XML.

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

Les chaînes de caractères se trouvent dans le fichier `strings.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello_world">Hello world!</string>
</resources>
```

Pour accéder à la chaîne de caractères :

- ▶ en XML : `@string/hello_world` ;
- ▶ en Java : `getResources().getText(R.string.hello_world)`.

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 (sans l'extension) ou la valeur de l'attribut `android:name` d'un élément XML (pour les valeurs simples).

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

Internationalisation

Il est possible de définir des valeurs pour plusieurs langues :

- ▶ `values/strings.xml` contient le texte en anglais des chaînes ;
- ▶ `values-fr/strings.xml` contient le texte en français des chaînes ;
- ▶ `values-ja/strings.xml` contient le texte en japonais des chaînes ;

Si `title` n'est défini qu'en français et en anglais et que l'on fait référence à `R.string.title`, alors :

- ▶ Si le terminal n'est pas configuré en français, Android charge le texte dans le fichier `values/strings.xml` ;
- ▶ Si le terminal est configuré en français, Android charge le texte dans le fichier `values-fr/strings.xml` .

Les dimensions

Pour faciliter la prise en charge des différentes tailles d'écran, il est préférable d'utiliser des références vers 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

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, de la résolution, de la taille de l'écran, de la version de l'API, etc :

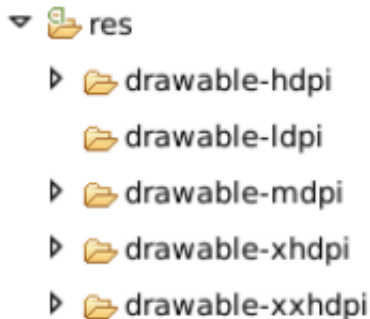
- ▶ `values/dimens.xml` contient les dimensions par défaut ;
- ▶ `values-w820dp/dimens.xml` contient les dimensions pour les écrans dont la largeur est supérieure à 820dp ;
- ▶ `layout/activity.xml` contient la version par défaut d'un layout ;
- ▶ `layout-land/activity.xml` contient la version paysage de ce même layout.

Quelques remarques :

- ▶ Le système va aller chercher la bonne ressource automatiquement ;
- ▶ Cela est valable pour tous les types de ressources.

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

Une collection d'icônes est proposée au téléchargement sur le site d'Android (<https://developer.android.com/design/downloads>)

Action Bar Icon Pack

Action bar icons are graphic buttons that represent the most important actions people can take within your app. [More on Action Bar Iconography](#)

The download package includes icons that are scaled for various screen densities and suitable for use with the Holo Light and Holo Dark themes. The package also includes unstyled icons that you can modify to match your theme, plus source files.



Elle fournit les icônes standards à utiliser dans les barres d'action

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

Il est également possible de définir un Drawable via un fichier XML :

```
<?xml version="1.0" encoding="utf-8"?>
<ripple xmlns:android="..."
    android:color="?android:colorControlHighlight">
    <item>
        <shape android:shape="oval">
            <solid android:color="?android:colorAccent"/>
        </shape>
    </item>
</ripple>
```

Les images

Il est également possible de définir un Drawable qui change en fonction de l'état de l'élément sur lequel il est utilisé :

```
<?xml version="1.0" encoding="utf-8" ?>
<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

```
<?xml version="1.0" encoding="utf-8"?>
<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 :

```
<?xml version="1.0" encoding="utf-8"?>
<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 manifest de l'application :

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

Types de ressources

Les différents types de ressources :

- ▶ Animations ;
- ▶ Couleurs ;
- ▶ Images (Drawable Resources) ;
- ▶ Layouts ;
- ▶ Menus ;
- ▶ Chaînes de caractères ;
- ▶ Styles ;
- ▶ Préférences ;
- ▶ etc...

Les identifiants

Il est possible d'associer 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>
```

Ensuite, l'identifiant est accessible par `R.id.text` dans le code Java.

Les identifiants

Accès à un élément d'un layout dans une activité par 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 boutons

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/count"
    />

</LinearLayout>
```


Les boutons

```
public class MainActivity extends Activity {  
    private TextView text;  
    private int count;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState); count = 0;  
        setContentView(R.layout.activity_main);  
        text = (TextView)findViewById(R.id.text);  
        Button button = (Button)findViewById(R.id.button);  
        text.setText(""+count);  
        button.setOnClickListener(new MyOnClickListener());  
    }  
}
```

Les boutons

Utilisation de l'attribut onClick :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/count"
        android:onClick="count"
    />

</LinearLayout>
```

Les boutons

Définition de la méthode count dans la classe de l'activité :

```
public class MainActivity extends Activity {  
    private TextView text;  
    private int count;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState); count = 0;  
        setContentView(R.layout.activity_main);  
        text = (TextView)findViewById(R.id.text);  
        text.setText(""+count);  
    }  
  
    public void count(View view) {  
        count++; text.setText(""+count);  
    }  
}
```

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 ;

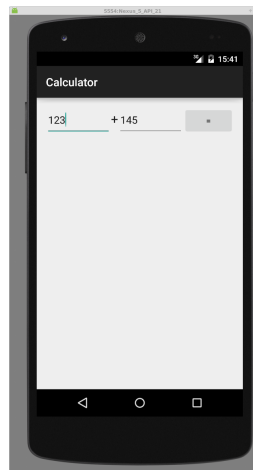
Une application avec deux fragments



Premier layout : le formulaire

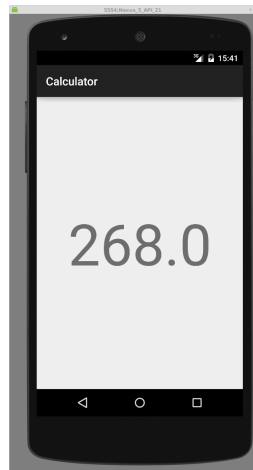
Le layout correspondant au formulaire du premier écran :

```
<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 resultat

```
<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" />
```


L'activité principale

Le code de l'activité principale :

```
public class MainActivity extends Activity {

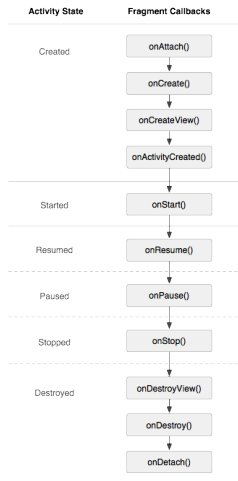
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                .add(R.id.container, new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

La création de la vue dans le premier fragment

Le code du 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



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

Système de notification entre le fragment et l'activité

- ▶ Le fragment `FormFragment` doit notifier l'activité de la validation du formulaire lorsque l'utilisateur clique sur le bouton ;
- ▶ Nous allons définir une 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 d'être capable de notifier l'activité.

Système de 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 OnClickListener");  
        }  
    }  
}
```

Système de 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;  
    }  
}
```

Système de 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);  
        }  
    }  
}
```

Système de 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

Mise en place du layout du deuxième fragment :

```
public class ResultFragment extends 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() { /* TODO */ }
}
```

Les paramètres d'un fragment

Les paramètres sont conservés même si le fragment est détruit :

```
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;  
    }  
  
    public double value() {  
        return getArguments().getDouble("value");  
    }  
}
```

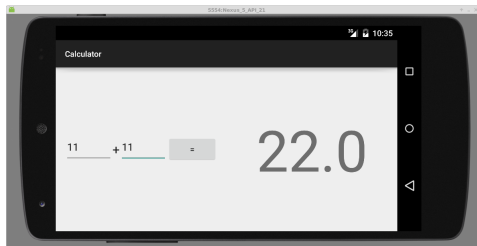
Affichage de deux fragments par l'activité

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 :

```
<?xml version="1.0" encoding="utf-8"?>
<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 (dans le répertoire layout-land) :

```
<?xml version="1.0" encoding="utf-8"?>
<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 boîtes de dialogue

Les boîtes de dialogue sont des fragments particuliers :

```
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 MyOnNegativeButtonClickListener());
        return builder.create();
    }
}
```

Les boîtes de dialogue

Nous allons faire communiquer le fragment et l'activité via l'interface :

```
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 OnClickListener");  
        }  
    }  
}
```

Les boîtes de dialogue

On implémente ensuite les deux classes internes de façon à traiter correctement la validation et l'annulation du formulaire :

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

Les boîtes de dialogue

Code de l'activité principale :

```
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());
    }

    /* ... */
}
```

Les boîtes de dialogue

Code de l'activité principale :

```
public class MainActivity extends Activity
    implements DialogFragmentListener {
    private TextView textView;

    /* ... */

    @Override
    public void onChangeText(String text) { textView.setText(text); }

    private class MyOnClickListener implements View.OnClickListener {
        @Override
        public void onClick(View v) {
            DialogFragment newFragment = new DialogFragment();
            newFragment.show(getFragmentManager(), "dialog");
        }
    }
}
```

Les menus

Code XML d'un menu :

```
<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>
```



Chargement du menu dans l'activité

Si le fichier contenant la définition du menu est menu/menu_main.xml alors :

```
public class MainActivity extends Activity {  
  
    @Override  
    public boolean onCreateOptionsMenu(Menu menu) {  
        getMenuInflater().inflate(R.menu.main, menu);  
        return true;  
    }  
  
}
```

Traitement des actions du menu

La méthode suivante permet de traiter les actions des menus :

```
public class MainActivity extends Activity {  
  
    @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);  
        }  
    }  
}
```


Traitement des actions du menu

Les autres parties du code de l'activité :

```
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

Les autres parties du code 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

Les fragments peuvent ajouter des éléments dans les menus :

```
<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>
```



Menus et fragments

Création de la vue pour la classe CounterFragment :

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

    @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;
    }

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

Menus et fragments

Le code XML du menu :

```
<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

Mise en place du menu :

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

    @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

Traitement des événements du menu :

```
public class CounterFragment extends Fragment {  
    @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

Création de la vue pour la classe ColorFragment :

```
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

Le code XML du menu :

```
<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

Mise en place du menu :

```
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

Traitement des événements du menu :

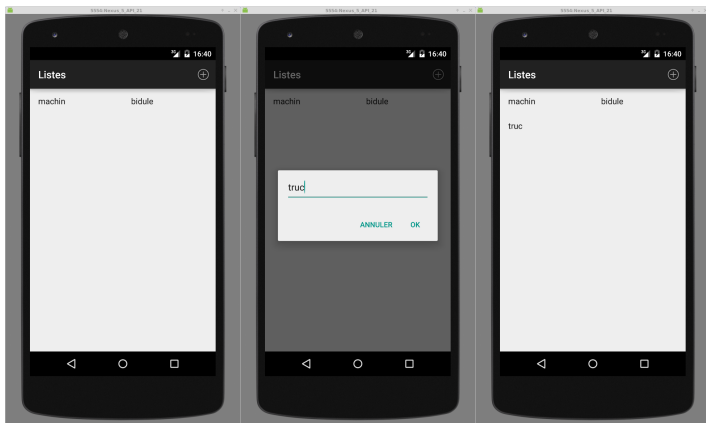
```
public class ColorFragment extends Fragment {

    @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

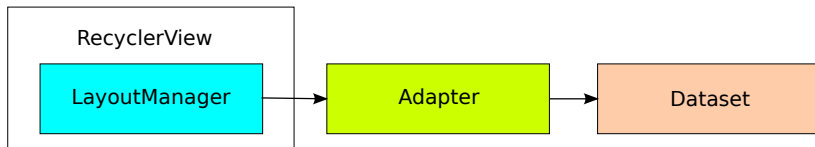
- Dans le cours et les TP, nous allons utiliser des RecyclerView :



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

Les listes

- ▶ Les RecyclerViews permettent l'affichage de grande quantité de données et sont efficaces (en construisant un petit nombre de vues) ;
- ▶ Les LayoutManagers permettent de gérer la position des éléments ;
- ▶ Les Adapters vont adapter les données aux vues.



Le layout principal de l'application

Le layout de l'activité principale :

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

Mise en place de l'activité principale

```
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;
    }
}
```


Code de l'adaptateur

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

    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

Le dialogue va communiquer avec l'activité via l'interface suivante :

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

L'activité va implémenter cette interface :

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

Notification de l'adaptateur de la modification du modèle

Il faut notifier l'adaptateur de tout changement des données du modèle :

```
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();
    }
}
```

Remarque : 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

Création de la vue :

```
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 MyOnNegativeButtonClickListener());
        return builder.create();
    }
}
```

Code de la boîte de dialogue

Code XML du layout 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 boite de dialogue

Connexion de la boite de dialogue et de l'activité :

```
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 OnClickListener");
        }
    }
}
```


Code de la boîte de dialogue

Traitement des actions 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 MyOnNegativeButtonClickListener
        implements DialogInterface.OnClickListener {
        public void onClick(DialogInterface dialog, int id) {
            DialogFragment.this.getDialog().cancel();
        }
    }
}
```

Sélection des éléments

Nous souhaitons pouvoir sélectionner les éléments de la liste :



Sélection des éléments

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

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

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

    public void setOnItemClickListener(OnItemClickListener listener) {
        listeners.add(listener);
    }

    public void notifyOnItemClickListener(int index) {
        for (OnItemClickListener 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.setOnItemClickListener(new OnItemClickListener());
    }

    private class OnItemClickListener implements ItemAdapter.OnItemClickListener {
        @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

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.setOnItemClickListener(new OnItemClickListener());
    }

    private class OnItemClickListener implements ItemAdapter.OnItemClickListener {
        @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="@android:id/mask"
        android:drawable="@android: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”

Le menu du 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>
```

Le mode “action”

Changement de mode à la suite d'une sélection :

```
public class MainActivity extends Activity
    implements DialogFragmentListener {

    private void swapItemSelection(int index) {
        /* ... */ updateActionMode();
    }

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

Le mode “action”

Création du menu du 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;
        }
    }
}
```

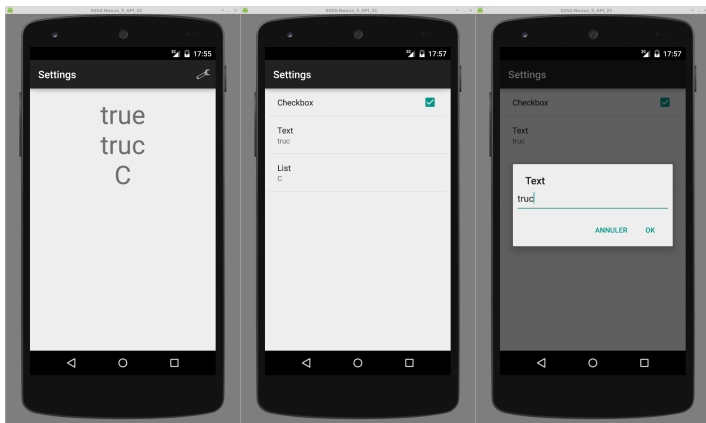
Le mode “action”

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();
    }
}
```

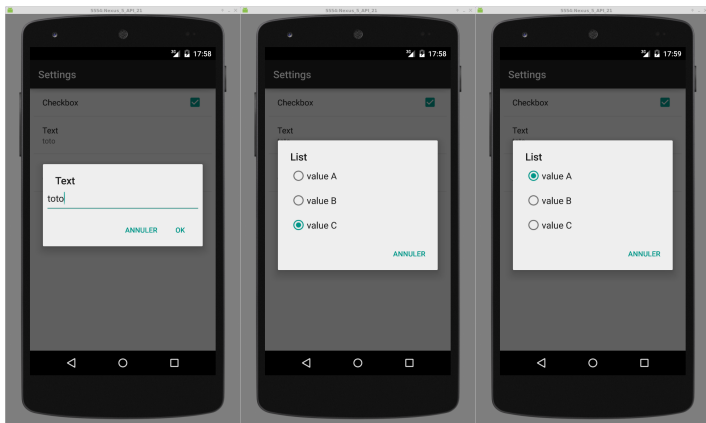
Exemple d'utilisation des préférences

Exemple d'utilisation des préférences:



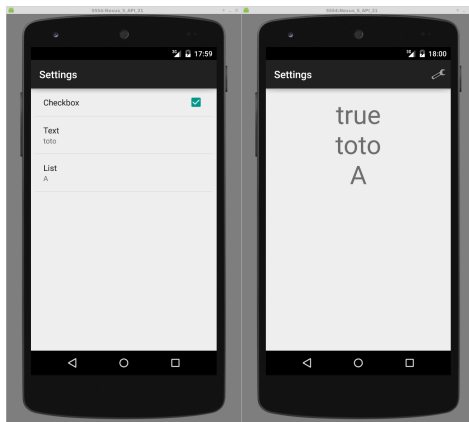
Exemple d'utilisation des préférences

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 :

```
<?xml version="1.0" encoding="utf-8"?>
<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) :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <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”

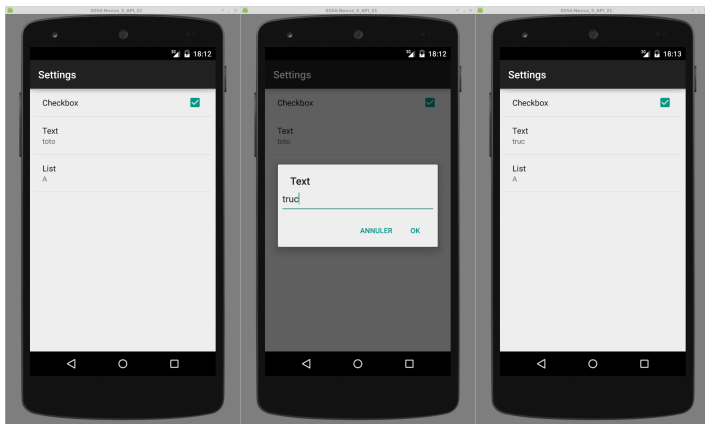
Le code d'un Fragment affichant les préférences :

```
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", false);
        String textValue = sharedPreferences.getString("pref_text", "");
        String listValue = sharedPreferences.getString("pref_list", "");
    }
}
```

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 :

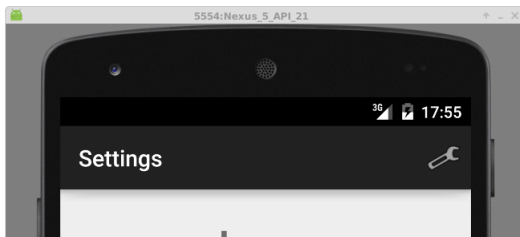
```
public class MainFragment extends Fragment
    implements SharedPreferences.OnSharedPreferenceChangeListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

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

Mise en place du menu

Code XML 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) {  
            getSupportFragmentManager().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

Deux espaces de stockage :

- ▶ 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

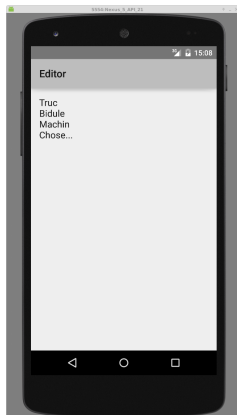
```
FileOutputStream outputStream  
    = openFileOutput("fichier.txt", Context.MODE_PRIVATE);
```

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 :

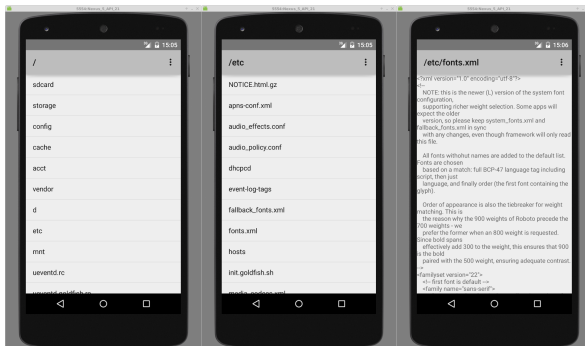
```
<manifest>
  <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

- 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 :

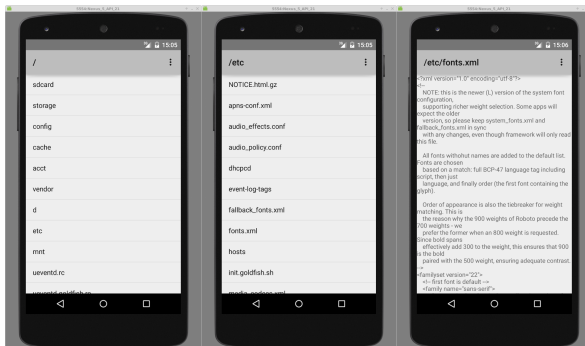


Deux types de fragments :

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

Un exemple d'application

Nous souhaitons parcourir et lire les fichiers du stockage externe :



Deux types de fragments :

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

Affichage du contenu d'un fichier

Le layout :

```
<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 OnDirectoryChangeListener listener;  
  
    public interface OnDirectoryChangeListener {  
        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 OnFragmentInteractionListener");
        }
    }

    @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 onItemClick(ListView list,  
                            View view,  
                            int position, long id) {  
        super.onItemClick(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) {  
            getSupportFragmentManager().beginTransaction()  
                .add(R.id.container, DirectoryFragment.newInstance("/"))  
                .commit();  
        }  
    }  
}
```

L'activité principale

Réception des notifications des fragments :

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

    @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();
    }
}
```

Le format JSON

JSON (JavaScript Object Notation) est un format de données dérivé de la notation des objets et tableaux de ECMAScript (donc de JavaScript) :

```
{ "machin": {  
  "taille": 12,  
  "style": "gras",  
  "bidule": {  
    "machin": [  
      { "style" : "italique" },  
      { "style" : "gras" }  
    ]  
  }  
}}
```

L'avantage de JSON est qu'il est reconnu nativement par JavaScript.

Le format JSON

Les éléments en JSON :

- ▶ Les objets : {chaîne : valeur, chaîne : valeur...}
- ▶ Les tableaux : [valeur, valeur, ...]
- ▶ Les valeurs : chaîne, nombre, objet, tableau, true, false, null
- ▶ Les chaînes : "abcdef" ou "abcd\n\t"
- ▶ Les nombres : -1234.12

```
{ "unObjet": {  
    "unTableau": [12, 13, 53],  
    "unNombre" : 53,  
    "unChaîne" : "truc\n"  
    "unObjet" : { "style" : "gras" }  
}}
```


Le format JSON

Les éléments en JSON :

- ▶ Les objets : {chaîne : valeur, chaîne : valeur...}
- ▶ Les tableaux : [valeur, valeur, ...]
- ▶ Les valeurs : chaîne, nombre, objet, tableau, true, false, null
- ▶ Les chaînes : "abcdef" ou "abcd\n\t"
- ▶ Les nombres : -1234.12

```
{ "unObjet": {  
  "unTableau": [12, 13, 53],  
  "unNombre" : 53,  
  "unChaîne" : "truc\n"  
  "unObjet" : { "style" : "gras" }  
}}
```

Le format JSON sous Android

La gestion des objets est réalisée par la classe `JSONObject` :

- ▶ `JSONObject put(String key, JSONObject object) ;`
- ▶ `JSONObject put(String key, JSONArray array) ;`
- ▶ `JSONObject put(String key, boolean|int|long|double b) ;`
- ▶ `JSONObject getJSONObject(String key) ;`
- ▶ `JSONArray getJSONArray(String key) ;`
- ▶ `boolean getBoolean(String key), ... ;`

Le format JSON sous Android

La gestion des tableaux est réalisée par la classe `JSONArray` :

- ▶ `JSONArray put([int index,] JSONObject object) ;`
- ▶ `JSONArray put([int index,] JSONArray array) ;`
- ▶ `JSONArray put([int index,] boolean|int|long|double b) ;`
- ▶ `JSONObject getJSONObject(int index) ;`
- ▶ `JSONArray getJSONArray(int index) ;`
- ▶ `boolean getBoolean(int index), ... ;`
- ▶ `int length() ;`

Le format JSON sous Android

Un objet JSON :

```
{"array": [1, 2, 3], "boolean" : false }
```

Construction d'une chaîne de caractères au format JSON en Java :

```
String toJSON() throws JSONException {  
    JSONObject object = new JSONObject();  
    JSONArray array = new JSONArray();  
    array.put(1).put(2).put(3);  
    object.put("array", array);  
    object.put("boolean", false);  
    return object.toString();  
}
```

Le format JSON sous Android

Un objet JSON :

```
{"array": [1, 2, 3], "boolean" : false }
```

Construction d'une chaîne de caractères au format JSON en Java :

```
String toJSON() throws JSONException {  
    JSONObject object = new JSONObject();  
    JSONArray array = new JSONArray();  
    array.put(1).put(2).put(3);  
    object.put("array", array);  
    object.put("boolean", false);  
    return object.toString();  
}
```

Le format JSON sous Android

Un objet JSON :

```
{"array": [1, 2, 3], "boolean" : false }
```

Lecture d'une chaîne de caractères au format JSON en Java :

```
void readJSON(String json) throws JSONException {  
    JSONObject object = new JSONObject(json);  
    JSONArray array = object.getJSONArray("array");  
    for (int i = 0; i < array.length(); i++)  
        Log.d("json", ""+array.get(i));  
    Log.d("json", ""+object.getBoolean("boolean"));  
}
```

Bases de données SQLite

Nous allons voir comment stocker de l'information structurée sous la forme d'une base de données SQLite. 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

Construction des 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 guild, title FROM articles WHERE title LIKE "%s%" :

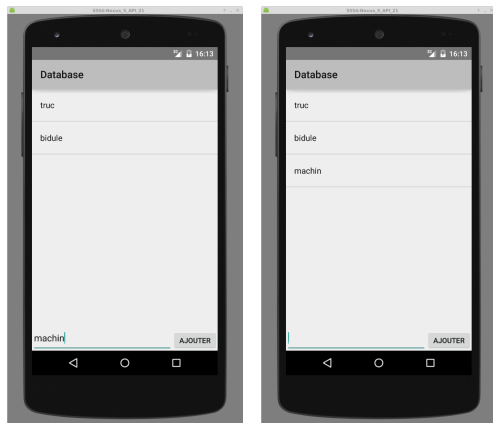
```
public Cursor getArticles(String s) {  
    SQLiteDatabase database = databaseOpenHelper.getReadableDatabase();  
    String selection = DatabaseContract.Articles.columnTitle + " LIKE ?";  
    String[] selectionArgs = {"%" + s + "%"};  
    String[] columns = { DatabaseContract.Articles.columnGuid,  
        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 columnIndex)`
 `double getDouble(int columnIndex)`
 ...
- ▶ `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);  
    }  
}
```

Les threads

Quelques remarques générales :

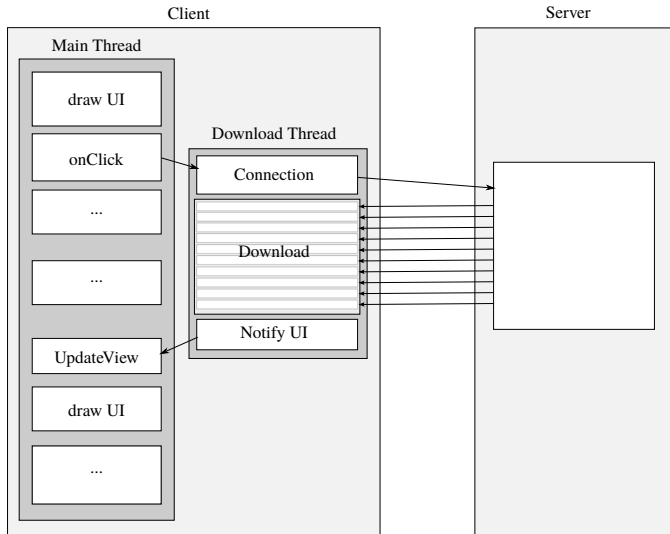
- ▶ Les threads sont des fils d'exécution ou tâches ;
- ▶ Les threads s'exécutent (ou semblent s'exécuter) en parallèle ;
- ▶ Les threads d'un même processus partagent le même espace mémoire ;
- ▶ Par défaut, une application n'utilise qu'un thread principal ;
- ▶ Seul le thread principal peut modifier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

Pourquoi créer des threads ?

- ▶ Pour paralléliser des tâches (des calculs par exemple) ;
- ▶ Pour attendre la réponse d'un serveur sans bloquer l'application...

⇒ **Pour ne jamais bloquer le thread principal.**

Un exemple



L'interface Runnable et la classe Thread

L'interface Runnable :

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

Implémentation de l'interface pour définir la tâche à exécuter :

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

Création et exécution d'un nouveau thread :

```
public void onClick(View v) {
    Thread thread = new Thread(new DownloadTask());
    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); }  
    });  
}
```

Méthodes permettant d'exécuter des tâches dans le thread principal :

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

La classe AsyncTask

La classe AsyncTask permet de simplifier l'écriture d'un thread et sa synchronisation avec le thread principal :

```
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);
```

La classe AsyncTask contient d'autres méthodes permettant de gérer la progression de l'exécution de la tâche, son annulation, etc. :

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

Exemple – Connexion réseau

Un exemple d'application :



Exemple – Connexion réseau

Pour cela, 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();  
        }  
    }  
}
```

Navigation entre applications

Nous allons voir comment envoyer un utilisateur :

- ▶ vers une autre application :
 - ▶ Carte,
 - ▶ Contact,
 - ▶ Appel...
- ▶ vers une autre activité de votre application.

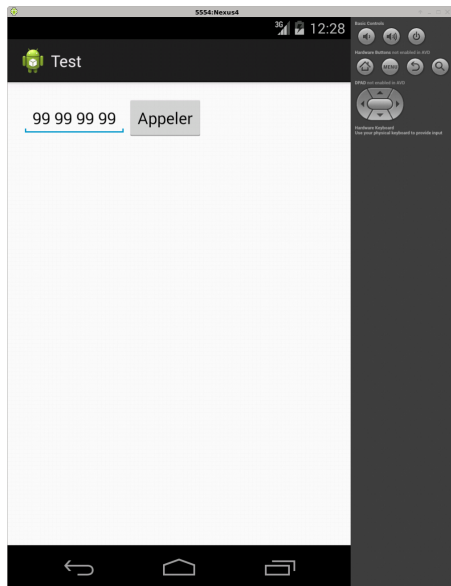
Pour cela, nous allons utiliser les intentions ou “intents”.

Dans un Intent, on précise :

- ▶ une action (voir, éditer, envoyer, etc.)
- ▶ une donnée (souvent une Uri désignant une ressource)

Nous verrons également comment répondre à certains intents.

Construire un Intent



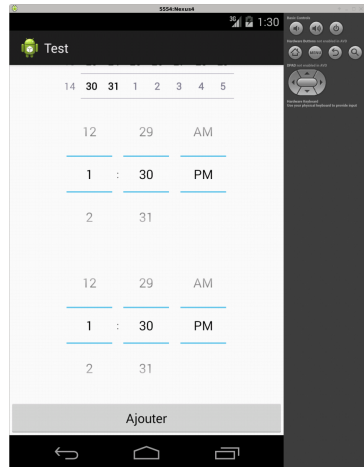
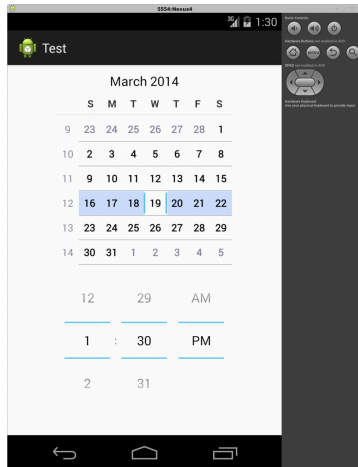
Construire un Intent

```
<LinearLayout xmlns:android="..."
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <EditText
        android:id="@+id/number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="phone" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="call"
        android:text="@string/call" />
</LinearLayout>
```

Construire un Intent

```
public class MainActivity extends Activity {  
    private EditText number;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        number = (EditText)findViewById(R.id.number);  
    }  
  
    public void call(View v) {  
        Uri uri = Uri.parse("tel:"+number.getText());  
        Intent callIntent = new Intent(Intent.ACTION_DIAL, uri);  
        startActivity(callIntent);  
    }  
}
```

Construire un Intent



Construire un Intent

```
public class MainActivity extends Activity {  
  
    private DatePicker datePicker;  
    private TimePicker startTimePicker, endTimePicker;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        startTimePicker = (TimePicker)findViewById(R.id.startTimePicker);  
        endTimePicker = (TimePicker)findViewById(R.id.endTimePicker);  
        datePicker = (DatePicker)findViewById(R.id.datePicker);  
    }  
  
    /* ... */  
}
```

Construire un Intent

```
public class MainActivity extends Activity {  
  
    private DatePicker datePicker;  
    private TimePicker startTimePicker, endTimePicker;  
  
    /* ... */  
  
    private long getTimeInMillis(TimePicker timePicker) {  
        int year = datePicker.getYear();  
        int month = datePicker.getMonth();  
        int day = datePicker.getDayOfMonth();  
        int hour = timePicker.getCurrentHour();  
        int minute = timePicker.getCurrentMinute();  
        Calendar calendar = Calendar.getInstance();  
        calendar.set(year, month, day, hour, minute);  
        return calendar.getTimeInMillis();  
    }  
    /* ... */  
}
```

Construire un Intent

```
public class MainActivity extends Activity {  
  
    private DatePicker datePicker;  
    private TimePicker startTimePicker, endTimePicker;  
  
    /* ... */  
  
    public void addEvent(View view) {  
        Intent calendarIntent = new Intent(Intent.ACTION_INSERT,  
                                           Events.CONTENT_URI);  
        calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_BEGIN_TIME,  
                                getTimeInMillis(startTimePicker));  
        calendarIntent.putExtra(CalendarContract.EXTRA_EVENT_END_TIME,  
                                getTimeInMillis(endTimePicker));  
        startActivity(calendarIntent);  
    }  
}
```

Construire un Intent

Dans la classe `Intent`, des actions sont définies :

- ▶ `ACTION_CALL` → Initier un appel
- ▶ `ACTION_EDIT` → Afficher et éditer des données
- ▶ `ACTION_MAIN` → Démarrer une activité
- ▶ `ACTION_SYNC` → Synchroniser des données avec un serveur
- ▶ `ACTION_BATTERY_LOW` → Batterie faible
- ▶ `ACTION_HEADSET_PLUG` → Écouteur branché
- ▶ `ACTION_SCREEN_ON` → L'écran s'allume
- ▶ `ACTION_TIMEZONE_CHANGED` → Changement de fuseau horaire

Navigation dans l'application

Une application peut contenir plusieurs activités :

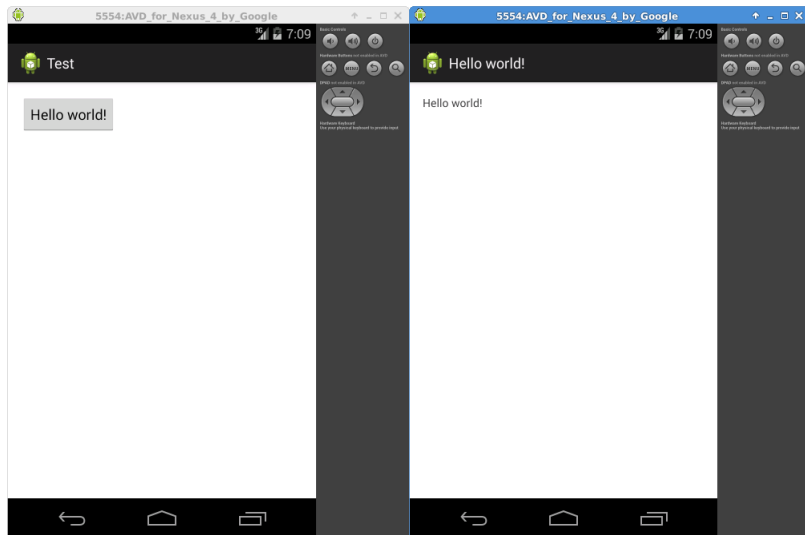
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="...">
  <application>
    <activity
      android:name="com.example.test.MainActivity"
      android:label="@string/app_name" >
      <!-- --->
    </activity>
    <activity
      android:name="com.example.test.OtherActivity"
      android:label="@string/title_other_activity" >
    </activity>
  </application>
</manifest>
```

Navigation dans l'application

Une application peut contenir plusieurs activités :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="...">
    <application>
        <activity
            android:name="com.example.test.MainActivity"
            android:label="@string/app_name" >
            <!-- --->
        </activity>
        <activity
            android:name="com.example.test.HelloWorldActivity"
            android:label="@string/hello_world" >
        </activity>
    </application>
</manifest>
```

Navigation dans l'application



Navigation dans l'application

L'activité principale :

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
    public void sayHelloWorld(View v) {  
        Intent intent = new Intent(this, HelloWorldActivity.class);  
        startActivity(intent);  
    }  
}
```

Avec un bouton dans le layout avec l'attribut suivant :

```
<Button onClick="sayHelloWorld" />
```

Navigation dans l'application

La seconde activité :

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

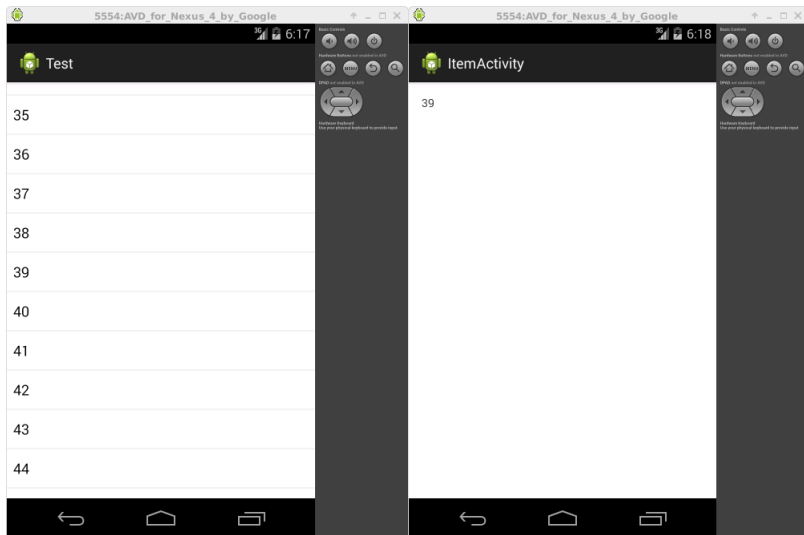
Avec une zone de texte avec l'attribut suivant :

```
<TextView text="@string/hello_world" />
```

Remarque :

- ▶ Ne pas oublier de déclarer l'activité dans le manifeste de l'application.

Navigation dans l'application



Navigation dans l'application

L'activité principale :

```
public class MainActivity extends ListActivity {  
  
    private ArrayAdapter<String> adapter;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        List<String> list = new ArrayList<String>();  
        for (int i = 0; i < 100; i++) list.add(""+i);  
        adapter = new ArrayAdapter<String>(this,  
                                           android.R.layout.simple_list_item_1, list);  
        setListAdapter(adapter);  
    }  
  
    /* ... */  
}
```

Navigation dans l'application

L'activité principale :

```
public class MainActivity extends ListActivity {

    private ArrayAdapter<String> adapter;

    /* ... */

    @Override
    protected void onListItemClick(ListView l, View v, int pos, long id) {
        String element = adapter.getItem(pos);
        Intent intent = new Intent(this, ItemActivity.class);
        intent.putExtra("item", element);
        startActivity(intent);
    }
}
```


Navigation dans l'application

Le layout de la deuxième activité :

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

    <TextView
        android:id="@+id/item"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Navigation dans l'application

Le code de la deuxième activité :

```
public class ItemActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_item);  
        TextView textView = (TextView)findViewById(R.id.item);  
        String item = getIntent().getStringExtra("item");  
        textView.setText(item);  
    }  
}
```

Remarques :

- ▶ Les informations qui proviennent de l'Intent sont conservées lorsque le système met en “hibernation” l'activité ;
- ▶ Ces informations peuvent donc être utilisées à chaque exécution de la méthode onCreate.

Navigation dans l'application



Navigation dans l'application

```
public class MainActivity extends ListActivity {  
    private ArrayAdapter<String> adapter;  
    private ArrayList<String> list;  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        list = new ArrayList<String>();  
        adapter = new ArrayAdapter<String>(this,  
            android.R.layout.simple_list_item_1, list);  
        setListAdapter(adapter);  
    }  
  
    public boolean onCreateOptionsMenu(Menu menu) {  
        MenuInflater inflater = new MenuInflater(this);  
        inflater.inflate(R.menu.main, menu);  
        return super.onCreateOptionsMenu(menu);  
    }  
  
    public void onAddItemAction(MenuItem menuItem) { /* ... */ }  
}
```

Navigation dans l'application

La description du menu :

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:orderInCategory="100"
        android:showAsAction="always"
        android:onClick="onAddItemAction"
        android:title="@string/add"/>
</menu>
```

La callback associée à l'item du menu :

```
public class MainActivity extends ListActivity {

    /* ... */

    public void onAddItemAction(MenuItem menuItem) { /* ... */ }
}
```

Navigation dans l'application

Exécution d'une activité avec attente de réponse :

```
public class MainActivity extends ListActivity {
    final int ADD_ITEM_ACTIVITY = 0;

    public void onAddItemAction(MenuItem menuItem) {
        Intent intent = new Intent(this, AddItemActivity.class);
        startActivityForResult(intent, ADD_ITEM_ACTIVITY);
    }

    @Override
    protected void onActivityResult(int requestCode,
                                     int resultCode, Intent data) {
        if (requestCode != ADD_ITEM_ACTIVITY)
            return;
        /* TODO : récupérer l'élément dans la variable item */
        adapter.add(item);
    }
}
```

Navigation dans l'application

Définition du layout de la seconde activité :

```
<LinearLayout xmlns:android="..."
    android:orientation="horizontal" >

    <EditText
        android:id="@+id/item"
        android:inputType="text" />

    <Button
        android:onClick="addItem" />

    <Button
        android:onClick="cancel" />

</LinearLayout>
```



Navigation dans l'application

Définition de la seconde activité :

```
public class AddItemActivity extends Activity {  
    private EditText editText;  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_add_item);  
        editText = (EditText)findViewById(R.id.item);  
    }  
  
    public void addItem(View v) {  
        Intent intent = new Intent();  
        intent.putExtra("item", editText.getText().toString());  
        setResult(RESULT_OK, intent);  
        finish();  
    }  
  
    public void cancel(View v) { setResult(RESULT_CANCELED); finish(); }  
}
```


Navigation dans l'application

Récupération du résultat dans l'activité principale :

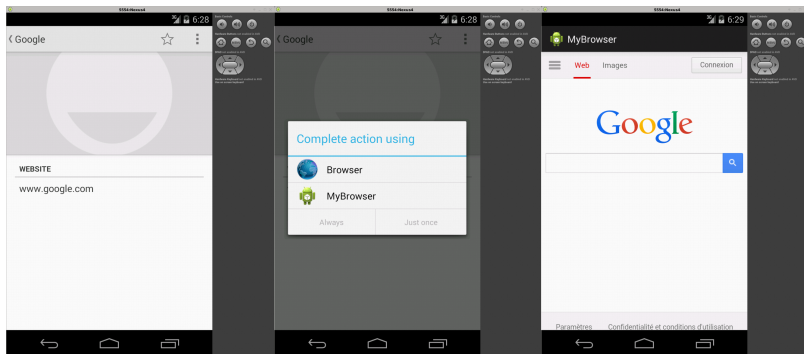
```
public class MainActivity extends ListActivity {  
    final int ADD_ITEM_ACTIVITY = 0;  
  
    /* ... */  
  
    @Override  
    protected void onActivityResult(int requestCode,  
                                    int resultCode,  
                                    Intent data) {  
        if (requestCode != ADD_ITEM_ACTIVITY) return;  
        if (resultCode != RESULT_OK) return;  
        String item = data.getStringExtra("item");  
        adapter.add(item);  
    }  
}
```

Navigation dans l'application

Gestion du cycle de vie de l'activité principale :

```
public class MainActivity extends ListActivity {  
    private ArrayAdapter<String> adapter;  
    private ArrayList<String> list;  
  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        if (savedInstanceState != null)  
            list = savedInstanceState.getStringArrayList("list");  
        else list = new ArrayList<String>();  
        /* ... */  
    }  
  
    protected void onSaveInstanceState(Bundle outState) {  
        super.onSaveInstanceState(outState);  
        outState.putStringArrayList("list", list);  
    }  
  
    /* ... */  
}
```

Répondre à des intentions



Répondre à des intentions

Ajout d'un intent-filter dans le manifeste de l'application :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="..." >
  <application>
    <activity
      android:name="com.example.test.BrowserActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
      </intent-filter>

    </activity>
  </application>
</manifest>
```

Répondre à des intentions

Le layout de l'activité :

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

    <WebView
        android:id="@+id/webView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

Répondre à des intentions

Le code de l'activité :

```
public class BrowserActivity extends Activity {
    private WebView webView;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        webView = (WebView) findViewById(R.id.webView);
        webView.loadUrl(getIntent().getData().toString());
        webView.setWebViewClient(new WebViewClient());
    }

    private class WebViewClient extends android.webkit.WebViewClient {
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            view.loadUrl(url);
            return true;
        }
    }
}
```