Projet Android

À la création d'un *projet Android* (IDE netbeans) tout un ensemble de répertoires et de fichiers sont engendrés.

- `Source Packages` : là où seront les sources de votre application.
- `Generated Source Packages` : là où se trouve la classe `R` qui fournit les *identificateurs de ressources* de l'application.
- `Resources` : là où se trouvent les ressources de l'application ; images, *widgets*, messages (chaînes de caractères) et fichiers de mise en page des interfaces graphiques.
- `Libraries` : la bibliothèque Android
- `Importants files` : surtout, `AndroidManifest.xml` qui donne au système Android les caractéristiques de votre application ; en particulier, la liste de ses composants, dont, en particulier, ses *activités*.

Application Android par défaut

Classe JAVA engendrée par défaut à la création d'un projet (Source Package)

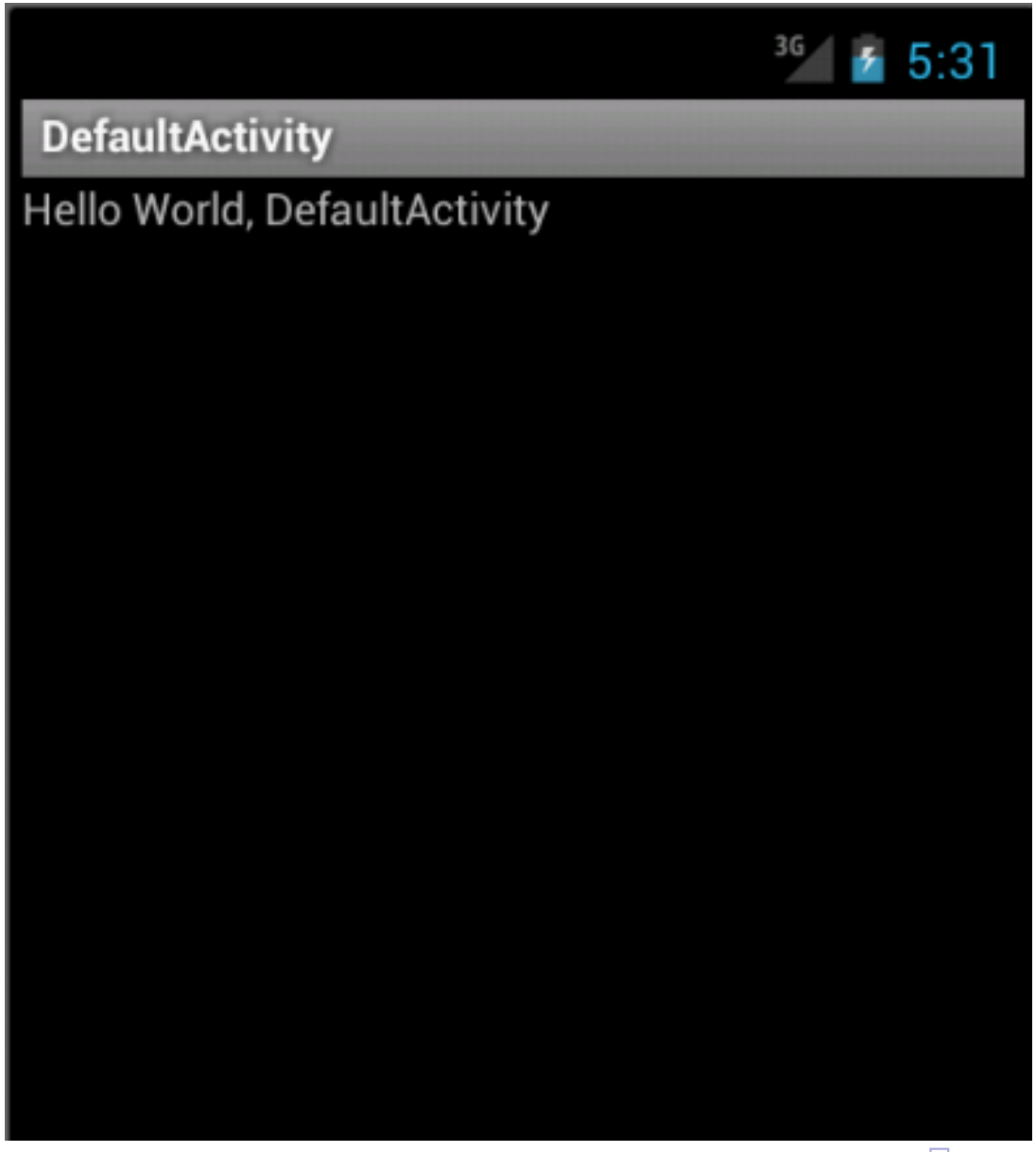
```
package eleph.android;

import android.app.Activity;
import android.os.Bundle;

public class DefaultActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

L'application `DefaultActivity`

On peut lancer l'application et on obtient :



Hello World, par défaut

Que s'est-il passé ? Android a lu le fichier `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
   xmlns:android="http://schemas.android.com/apk/res/android"
   package="eleph.android"
   android:versionCode="1" android:versionName="1.0">
 <application android:label="@string/app_name"
              android:icon="@drawable/ic_launcher">
   <activity android:name="DefaultActivity"
             android:label="@string/app_name">
     <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER
     </intent-filter>
   </activity>
 </application>
</manifest>
```

Le manifeste de l'application

Le manifeste déclare une `application`
qui contient une `activity`
nommée `DefaultActivity` déclarée comme
l'activité principale de l'application
(`android.intent.action.MAIN`)

Une instance de `DefaultActivity` est *créée par Android*

La méthode `OnCreate` est invoquée
qui invoque la méthode `setContentView`
avec l'argument $R.layout.main$

La *vue* (`View`) de l'application est créée avec les données fournies
par la valeur de l'argument $R.layout.main$.

Ici : un entier, identifiant une *ressource*.

La ressource R

Engendrée automatiquement par l'environnement Android de NetBeans : `Generated Source Packages/ ... /R.java`
Elle définit un identifiant pour chaque fichier contenu dans les sous-répertoires du répertoire `Resources`.
Ici, le fichier `layout/main.xml` [1]

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

1. engendré à la création du projet

Composer la vue de l'activité

Un fichier de ressources `xml`

- spécifie les éléments affichés et leur mise en page ;
- est analysé pour créer l'instance décrite d'une sous-classe de `View`.

La classe `View` est la mère des interfaces utilisateur d'Android.
Elle gère les affichages et les interactions.

`LinearLayout` est petite-fille de `View`, fille de `ViewGroup`.

- `ViewGroup` est le réceptacle (*container*) d'autres vues ;
- `LinearLayout` organise les vues qu'elle contient selon une direction : verticale ou horizontale.

Ici, un seul composant :
`TextView`, fille de `View` permet l'affichage d'un texte ;
éventuellement, son édition.

XML pour JAVA

XML	JAVA
Balise	Classe
Attribut(s) android :*	Méthode

Pour LinearLayout :
android :orientation="vertical"

```
⇝ setOrientation(LinearLayout.VERTICAL)
```

Pour TextView :
android :text="Hello World, DefaultActivity"

```
⇝ setText("Hello World, DefaultActivity")
```

Pour tous :
android :layout_width="fill_parent" et
android :layout_height="fill_parent"

```
⇝ generateLayoutParams(...)
```

XML et JAVA

Identifier un composant dans un fichier XML :

```
<LinearLayout
    ... >
  <TextView
      android:id="@+id/thisText"
      ... />
</LinearLayout>
```

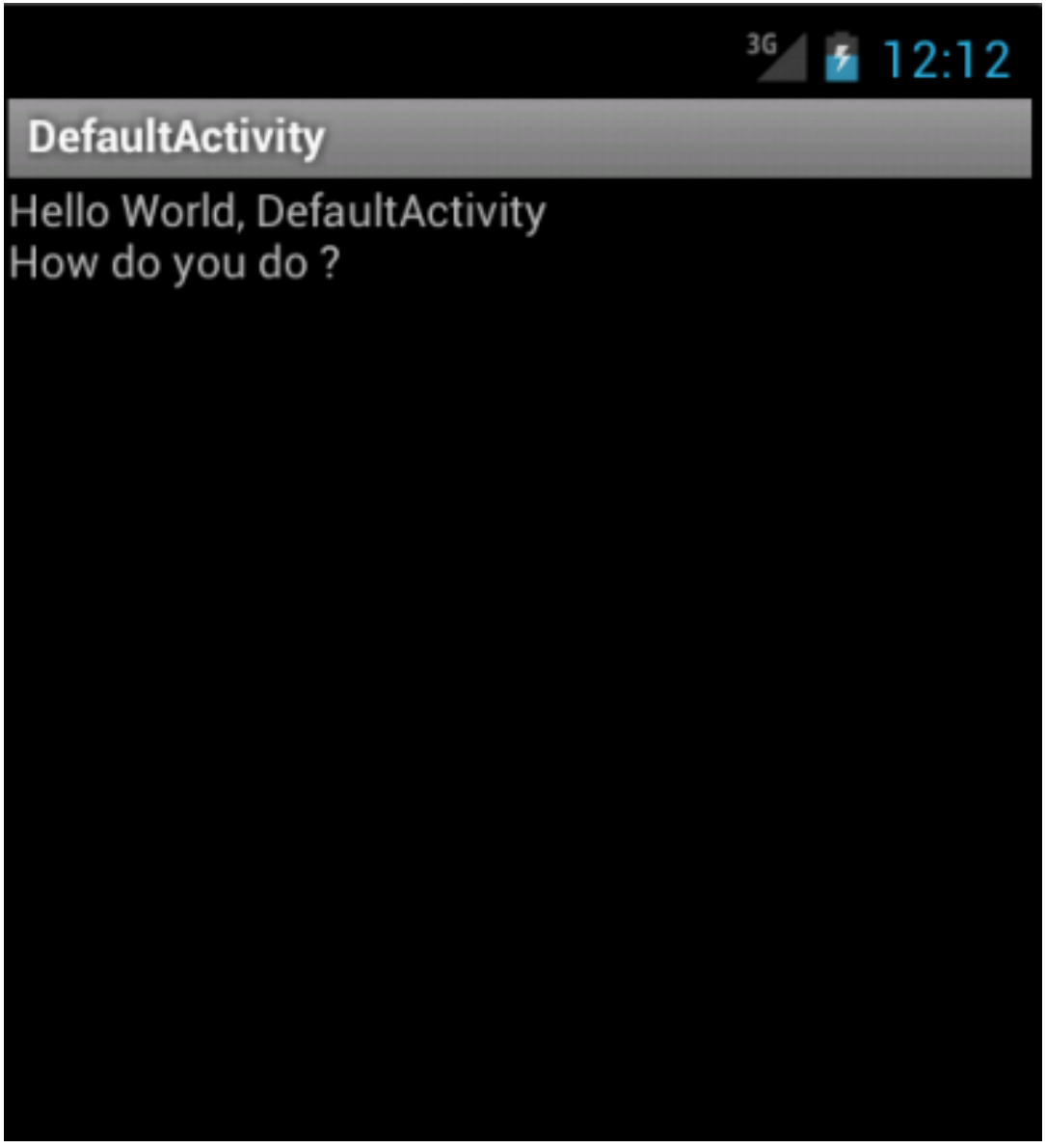
Accéder au composant et le modifier par programme

```
public void onCreate(Bundle savedInstanceState)
{
  ...
  TextView theText = (TextView)(findViewById(R.id.thisText));
  String message = (String)theText.getText();

  theText.setText(message+"\nHow do you do ?");
}
```

Ressource modifiée

On obtient :



Interaction : un bouton (1)

Classe `Button` (hérite de `TextView`)
XML : (ici, `main.xml`)

```
<Button
    android:id="@+id/exit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="EXIT"
    android:onClick="onClickExit"
/>
```

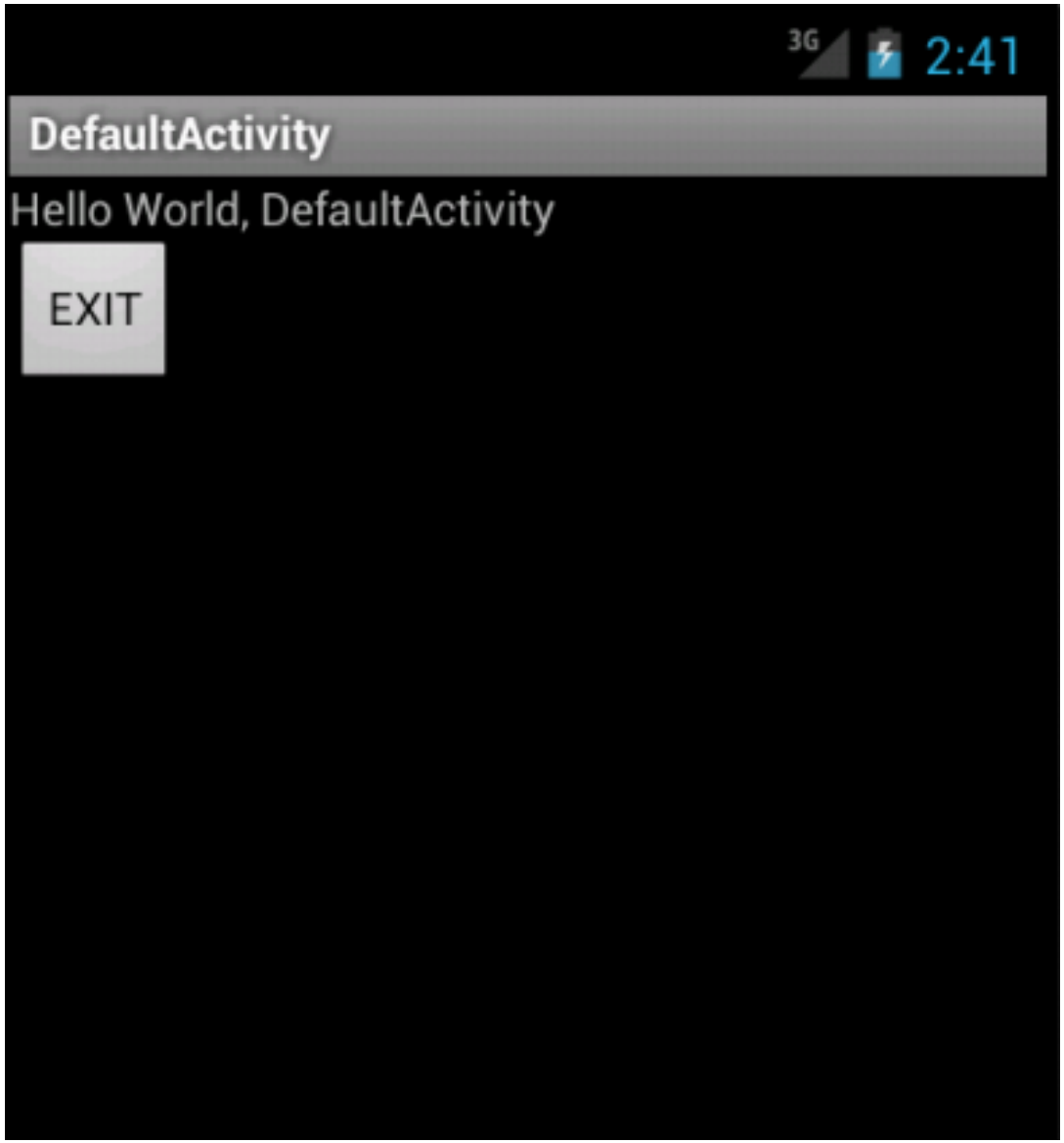
Associer une action à l'activation du bouton

JAVA : dans l'activité (ici, classe `DefaultActivity`)

```
public void onClickExit(View view) {
  finish();
}
```

Un bouton

On obtient :



(Démo)

Afficher des images

Pour afficher une image : `ImageView`

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/double6"
/>
```

L'image source (`src`) est dans le sous-répertoire `drawable` de `Resources`.
Ajuster les attributs de taille

```
android:maxWidth="60dip"
android:maxHeight="30dip"
android:scaleType="centerInside"
android:adjustViewBounds="true"
```

Une image

On obtient

Grille de jeu

Zone d'affichage interactive et dynamique

- **zone d'affichage :** `SurfaceView`
- **interactive :** `View.OnTouchListener`
- **dynamique :** un *thread* d'affichage dédié

Contrôler la surface et le *thread* associé (interfaces)

- `SurfaceHolder` : accéder aux ressources de la portion d'écran de la surface ;
- `SurfaceHolder.Callback` : synchroniser le *thread* avec la création et la destruction de la surface.

Une SurfaceView

```
public class FlowGridView extends SurfaceView
  implements SurfaceHolder.Callback, View.OnTouchListener {
  // Thread d'affichage
  private FlowGridThread gridThread;
  // Constructeur
  public FlowGridView(Context context) { ... }
  // Méthodes de SurfaceHolder.Callback
  public void surfaceCreated(SurfaceHolder sh) { ... }
  public void surfaceDestroyed(SurfaceHolder sh) { ... }
  // Méthode de View.OnTouchListener
  public boolean onTouch(View view, MotionEvent event) { ... }
  // Méthode de dessin de SurfaceView
  @Override
  public void onDraw(Canvas canvas) { ... }
}
```

SurfaceView et XML

Ajouter la vue de la grille à l'interface

≡

Ajouter sa decription dans `layout/main.xml`

```
<LinearLayout  [...]
    <TextView [...] />
    <eleph.android.games.flowfree.FlowGridView
        android:id="@+id/flowGridView"
        android:layout_width="150dp"
        android:layout_height="150dp" />
    <Button [...] />
</LinearLayout>
```

Une nouvelle balise XML :
`eleph.android.games.flowfree.FlowGridView`

Une SurfaceView : constructeur

Attacher les méthodes de contrôles de SurfaceHolder.Callback

```
public FlowGridView(Context context, AttributeSet attrs) {

    super(context, attrs);

    getHolder().addCallback(this);
    setFocusable(true);

    /* il y aura d'autres choses ici */
}
```

Une SurfaceView : *thread* et *callback* à la création

```
public void surfaceCreated(SurfaceHolder holder) {
  // création et démarage du thread d'affichage
  gridThread = new FlowGridThread(getHolder(), this);
  gridThread.setRunning(true);
  gridThread.start();
  // «écouter» les événements tactiles
  setOnTouchListener(this);
}
```

1. création et lancement du *thread*.
2. attachement de l'instance à l'écoute des événements (*touch*)

Une SurfaceView : *callback* à la disparition

```
public void surfaceDestroyed(SurfaceHolder holder) {
  boolean retry = true;
  gridThread.setRunning(false);
  while (retry) {
    try {
      gridThread.join();
      retry = false;

    } catch (InterruptedException e) {
    }
  }
}
```

1. signaler au *thread* sa fin
2. et l'attendre

Dessiner avec un `SurfaceView`

Surcharger `void OnDraw(Canvas canvas)`.
Utiliser :

- des méthodes de dessin de la classe `Canvas`

`http://developer.android.com/reference/android/graphics/Canvas.html`

- des méthodes d'attribut de dessin de la classe `Paint`

`http://developer.android.com/reference/android/graphics/Paint.html`

Réagir avec un `SurfaceView`

Implémenter `boolean onTouch(View view, MotionEvent event)`
La classe `MotionEvent` fournit :

- `float getX()` et `float getY()` coordonnées de l'événement ;
- `int getAction()` type d'action de l'événement.
  Valeurs : `MotionEvent.ACTION_DOWN` ;
  `MotionEvent.ACTION_MOVE` ; `MotionEvent.ACTION_UP`.

`http://developer.android.com/reference/android/view/MotionEvent.html`

Le *thread* (copié/collé)

```
public class FlowGridThread extends Thread {
    private final SurfaceHolder _surfaceHolder;
    private FlowGridView _panel;
    private boolean _run = false;
    public FlowGridThread(SurfaceHolder surfaceHolder, FlowGridView panel)
    { _surfaceHolder = surfaceHolder; _panel = panel; }
    public void setRunning(boolean run)
    { _run = run; }
    @Override
    public void run() {
        long previousTime, currentTime, refresh_rate;
        refresh_rate = 100 ;
        previousTime = System.currentTimeMillis();
        Canvas c;
        while (_run) {
            currentTime=System.currentTimeMillis();
            while ((currentTime-previousTime)<refresh_rate) {
                 currentTime=System.currentTimeMillis(); }
            previousTime=currentTime;
            try {
                c = _surfaceHolder.lockCanvas(null);
                try {
                    synchronized (_surfaceHolder)
                    { if (_run) { _panel.onDraw(c); } } }
                finally { // do this in a finally so that if an exception is thrown
                    if (c != null)
                    { _surfaceHolder.unlockCanvasAndPost(c); } }
            } catch(Exception e) { }
            try {  // Wait some time till I need to display again
                Thread.sleep(refresh_rate-5);
           } catch (InterruptedException e) {
                e.printStackTrace();
           } } } }
```

La grille du jeu, où est-elle ?

On a

- modèle de la grille : instance de `FlowModel`
- visualisation et interaction : instance de `FlowGridView`

Le modèle de la grille doit être *persistant*.

- résister aux changements d'orientation
- survivre à un appel téléphonique
- etc.

$\Rightarrow$ créer la grille au niveau de *l'application*

La classe Application

L'ajouter à AndroidManifest.xml

```
<manifest [...]
    <application
        android:name="FlowFreeApplication"
        [...] >
        <activity [...] </activity>
    </application>
</manifest>
```

Créer la classe

```
public class FlowFreeApplication extends Application {
  FlowModel theGrid;
  @Override
  public void onCreate() {
    super.onCreate();
    theGrid = new FlowModel(); // on fera mieux après
  }
  FlowModel getTheGrid() { return theGrid; }
}
```

Application et activité

Connaître l'application depuis l'activité

```
public class FlowFreeActivity extends Activity
{

  FlowFreeApplication app;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        [...]
        app = (FlowFreeApplication)this.getApplication();
    }
}
```

Application et vues

Connaître l'application depuis la surface de jeu

```
public class FlowGridView extends SurfaceView
  implements SurfaceHolder.Callback, View.OnTouchListener {
  FlowFreeApplication app;
  FlowModel theGrid;
  [...]
  public FlowGridView(Context context, AttributeSet attrs) {
    [...]
    app = (FlowFreeApplication)(context.getApplicationContext())
    theGrid = app.getTheGrid();
  }
  [...]
  @Override
  public void onDraw(Canvas canvas) {
    theGrid = app.getTheGrid();
    [...]
  }
}
```

Boîte de dialogue : ALERT

Exemple : signaler qui est le premier joueur

```
AlertDialog alert = new AlertDialog.Builder(this).create();
alert.setTitle("Beginner");
if (beginnerId == machineId)
  alert.setMessage("Machine with "+beginnerTile.toString());
else // beginnerId == humanId
  alert.setMessage("Human with "+beginnerTile.toString());

alert.setButton("OK", new DialogInterface.OnClickListener() {

  public void onClick(DialogInterface dialog, int which) {
// ici, on ne fait rien, mais ça n'est pas le ca en général
  }
});

alert.show();
```