Programmation Android
L3 informatique

Étienne Payet

Département de mathématiques et d'informatique

Plan

Ressources

- http://developer.android.com
- Livres : il y en a beaucoup...
  - *Android Programming : The Big Nerd Ranch Guide* (Phillips & Hardy)
  - voir à la BU
  - privilégiez ceux traitant des dernières versions d'Android

Pour pouvoir travailler

- Développement :
  - Windows $\geq$ XP, Mac OS X $\geq$ 10.5.8 (x86), Linux
  - JDK $\geq$ 5,
  - Android SDK
  - au moins une plateforme Android

  Recommandé :
  - l'IDE Android Studio (attention : Eclipse+ADT *deprecated*)

- Déploiement :
  - un appareil Android

Workflow Android

1. Développement	2. Test	3. Publication
Android Studio	Émulateur ou Appareil mobile	Google Play

Architecture d'Android

Voir ici pour une description détaillée

Quelques éléments sont présentés ci-après

Architecture d'Android

Linux Kernel

noyau Linux 2.6, sécurité, gestion mémoire, gestion processus, ...

Architecture d'Android

Android Runtime

fonctionnalités de la bibliothèque Java standard, Dalvik VM

Linux Kernel

noyau Linux 2.6, sécurité, gestion mémoire, gestion processus, ...

Architecture d'Android

Libraries	bibliothèques C/C++
Android Runtime	fonctionnalités de la bibliothèque Java standard, Dalvik VM
Linux Kernel	noyau Linux 2.6, sécurité, gestion mémoire, gestion processus, ...

Architecture d'Android

Application framework	API Java (gestion de fenêtres, de téléphonie, . . .)
Libraries	bibliothèques C/C++
Android Runtime	fonctionnalités de la bibliothèque Java standard, Dalvik VM
Linux Kernel	noyau Linux 2.6, sécurité, gestion mémoire, gestion processus, . . .

Architecture d'Android

Applications	des applications écrites en Java (navigateur Web, contacts, . . .)
Application framework	API Java (gestion de fenêtres, de téléphonie, . . .)
Libraries	bibliothèques C/C++
Android Runtime	fonctionnalités de la bibliothèque Java standard, Dalvik VM
Linux Kernel	noyau Linux 2.6, sécurité, gestion mémoire, gestion processus, . . .

Exercice

Installez sur votre machine un environnement de développement Android fonctionnel :

- JDK
- Android Studio (voir http://developer.android.com)

Plan

Développement d'une application Android

Code source :

- écrit en Java
- compilé en Dalvik bytecode (`.dex`)
- Dalvik bytecode $\neq$ Java bytecode

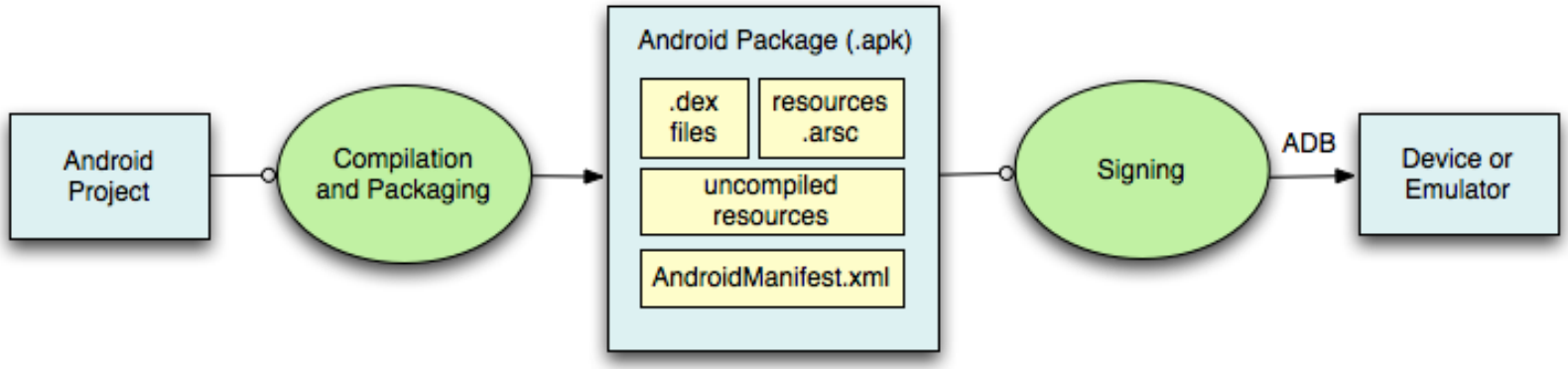
Configuration :

- fichier `AndroidManifest.xml`

Ressources :

- chaînes de caractères, images, audio, interfaces graphiques, ...
- certaines peuvent être décrites en XML

Compilation et exécution



Exécution

Chaque application tourne :

- dans son propre processus
- sur sa propre machine virtuelle
- isolément des autres applications (bac-à-sable de sécurité)

AndroidManifest.xml

Décrit :

- les composants de l'application
- les permissions requises par l'application (accès à Internet, lecture-écriture dans la liste de contacts, géo-localisation, ...)
- les besoins matériels et logiciels de l'application (appareil photo, Bluetooth, ...)
- ...

Composants d'une application

Un composant = un point d'entrée potentiel dans l'application

Une application peut lancer un composant d'une autre application

$\Rightarrow$ Plusieurs points d'entrée potentiels
(pas de méthode `main`)

$\neq$ application Java classique

Composants d'une application

Quatre types de composants :

- activités (*activities*)
- services (*services*)
- fournisseurs de contenu (*content providers*)
- récepteurs de diffusion (*broadcast receivers*)

Composants d'une application : AndroidManifest.xml

```
<activity android:name=".HelloWorldActivity"
          android:label="@string/app_name">
   <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
   </intent-filter>
</activity>
```

- Action `MAIN` = un point d'entrée principal (pas d'information particulière à fournir)
- Catégorie `LAUNCHER` = point d'entrée apparaissant dans le lanceur

Activité

- Un écran avec une interface utilisateur graphique
- Structurée par une hiérarchie de vues et de contrôles
  - des éléments définis par du code Java dans l'activité
  - des éléments définis dans des fichiers XML (ressources)
- Sous-classe de `android.app.Activity`

Activité

Exemple : application *Email*

- Une activité *afficherEmails*,
- une activité *composerEmail*,
- une activité *lireEmail*

Une application *Photo* pourra lancer l'activité *composerEmail* pour envoyer une photo.

Activité : squelette minimal

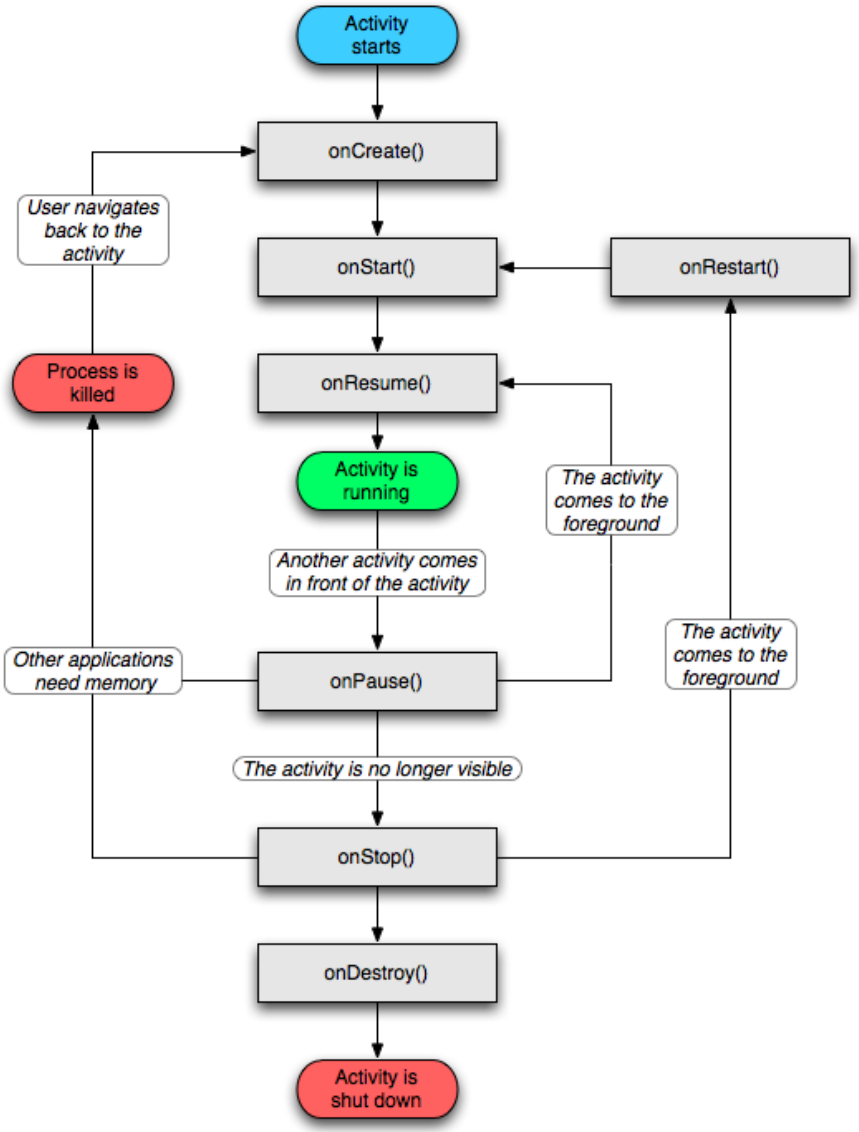
```
import android.app.Activity;
import android.os.Bundle;

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

Activité : cycle de vie

États principaux :

- active (*active*) : l'activité est visible et détient le focus (méthode `onResume`)
- suspendue (*paused*) : l'activité est en partie visible et ne détient pas le focus (méthode `onPause`)
- arrêtée (*stopped*) : l'activité n'est pas visible (méthode `onStop`)

Activité : cycle de vie



Service

- Composant qui fonctionne en tâche de fond
- Ne fournit pas d'interface utilisateur (invisible)
- Sous-classe de `android.app.Service`

Exemple : musique en tâche de fond

Un service pourrait jouer de la musique en tâche de fond pendant que l'utilisateur est dans une application différente.

Fournisseur de contenu

- Permet de gérer et de partager des données
- Sous-classe de `android.content.ContentProvider`

Exemple : liste de contacts

La liste de contacts de l'utilisateur est gérée par un fournisseur de contenu auquel toute application ayant les droits appropriés peut envoyer des requêtes de lecture/écriture.

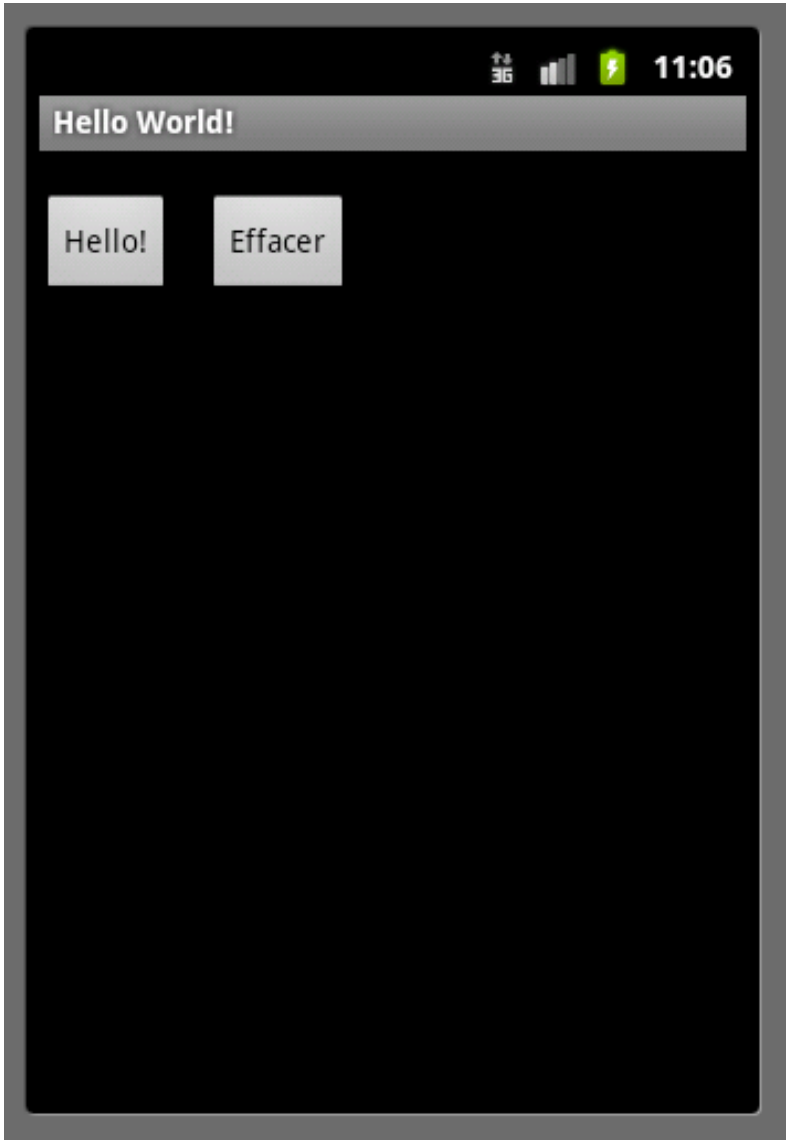
Récepteur

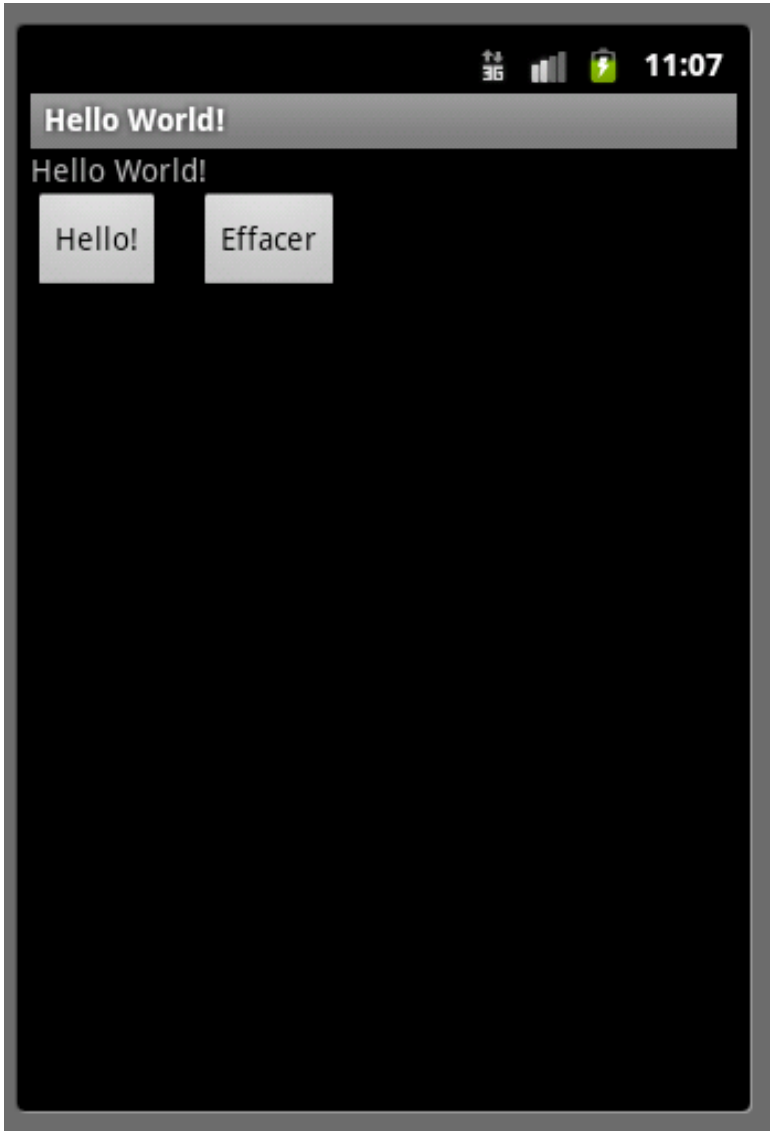
- Permet de réagir à la diffusion de messages
- Sous-classe de `android.content.BroadcastReceiver`

Exemples d'annonces système

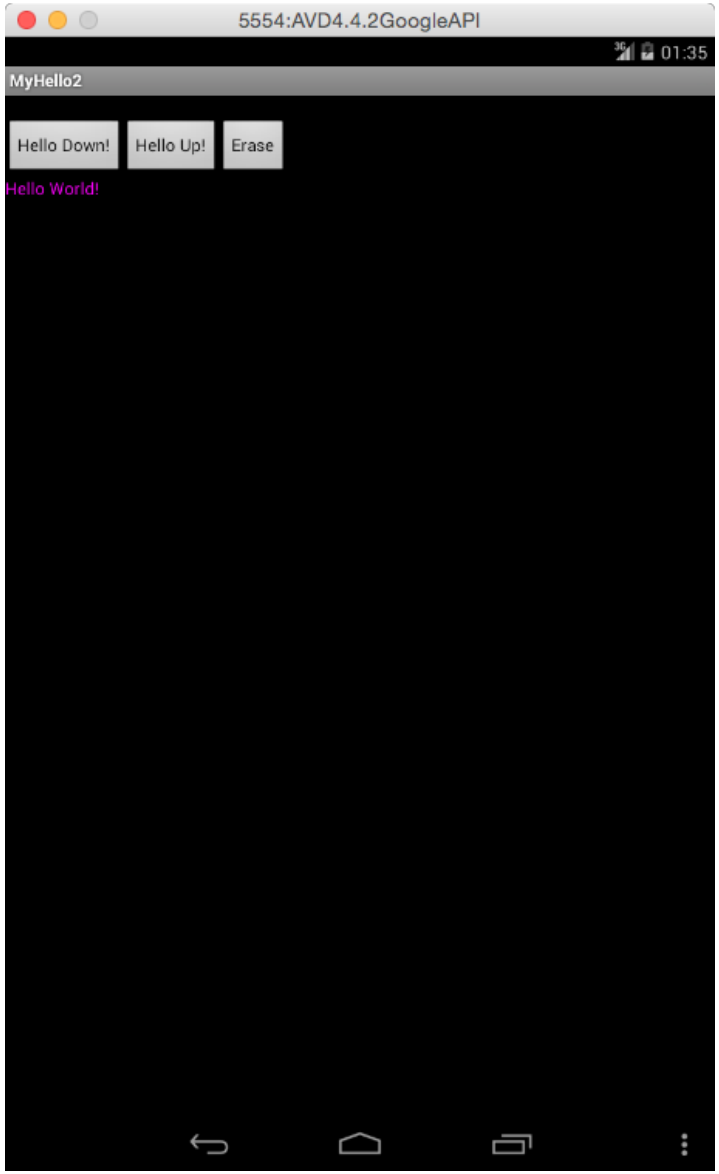
- L'écran s'est éteint
- La batterie est faible

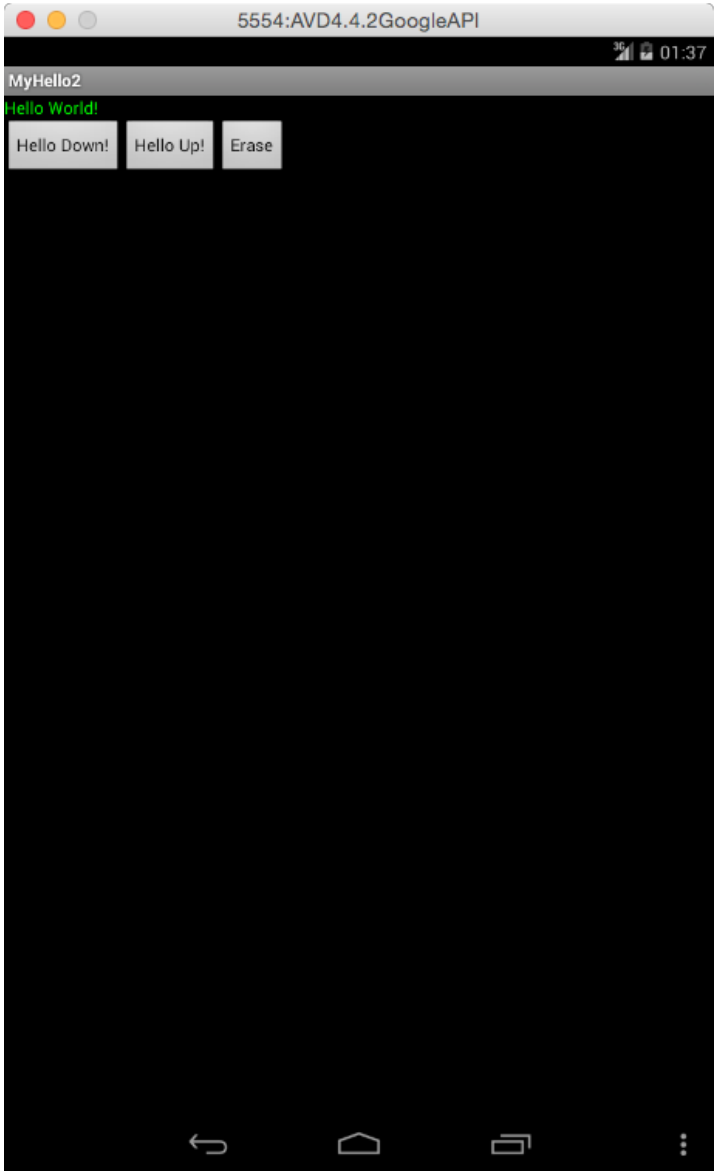
Exercice : *Hello World !*





Exercice : *Hello World !* (variante 1)





Exercice : *Hello World !* (variante 2)

La couleur du texte change à chaque affichage

Plan

Au préalable

- `AndroidManifest.xml` : application *debuggable*
- Sur l'appareil Android :
  - autoriser *Unknown sources*
    (Settings → Applications)
  - autoriser *USB debugging*
    (Settings → Applications → Development)
- L'ordinateur doit détecter l'appareil Android
  (`adb devices` dans le terminal pour vérifier si connecté)

Depuis Android Studio

Comme avec l'émulateur :

- bouton Run
- choisir l'appareil dans le *Device Chooser* qui apparaît

Exercice

Déployez vos applications *Hello World !* sur un appareil Android.

Plan

Ressources

- Chaînes de caractères, images, audio, interfaces graphiques, ...
- Dossier `res/` du projet

Structure habituelle du dossier res

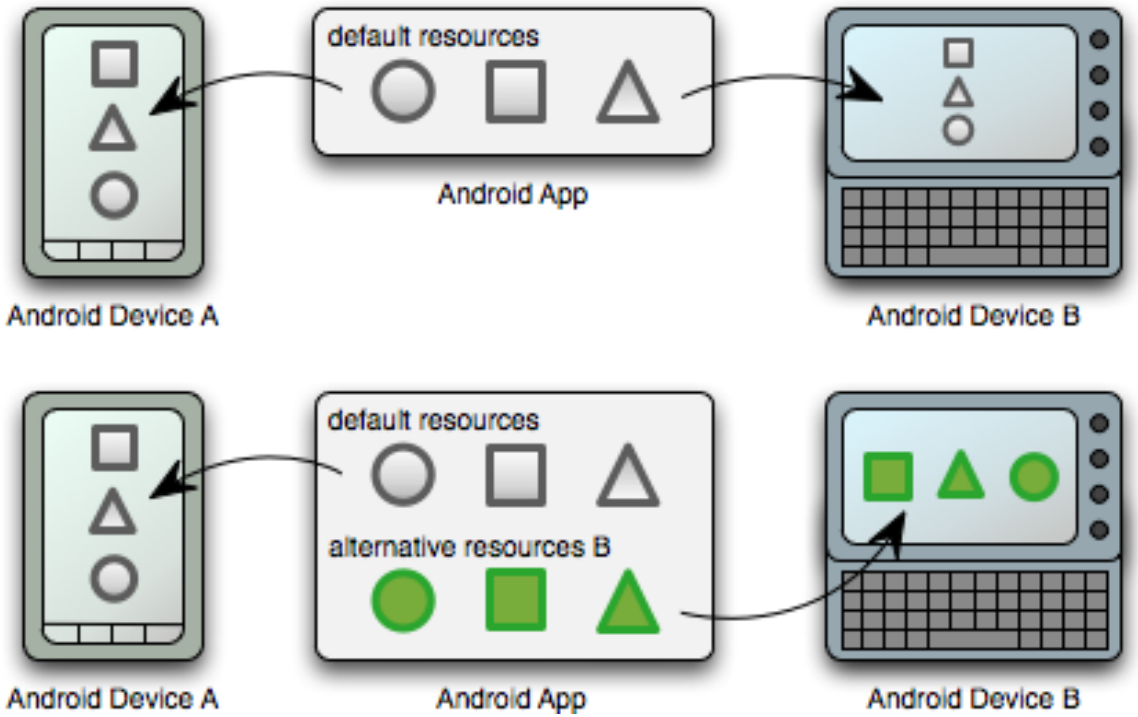
```
MonProjet/

    src/
        MonActivite.java

    res/
        drawable/
           icon.png
        layout/
           main.xml
        values/
           strings.xml
```

Ressources : séparées du code source

Externalisation :

- maintenance indépendante du code source
- ressources alternatives pour des configurations spécifiques (langues, tailles d'écran, . . .)

Ressources alternatives



Ressources alternatives

```
MonProjet/
    src/
        MonActivite.java
    res/
        drawable/
            icon.png
        drawable-hdpi/
            icon.png
        layout/
            main.xml
        layout-land/
            main.xml
        values/
            strings.xml
        values-fr/
            strings.xml
```

Ressources : chaînes de caractères

Fichier `strings.xml` dans le dossier `res/values` (et ses variantes)

```
<resources>
    <string name="app_name">Exercice HelloWorld</string>
    <string name="hello">Hello World!</string>
</resources>
```

Ressources : interfaces graphiques

Dans le dossier `res/layout` (et ses variantes)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/msgText"/>
</LinearLayout>
```

Ressources : utilisation dans le code

Classe R :

- générée automatiquement à la compilation
- contient l'ID de chaque ressource

```
public final class R {
    public static final class drawable {
        public static final int icon=0x7f020000; }
    public static final class id {
        public static final int msgText=0x7f050000; }
    public static final class layout {
        public static final int main=0x7f030000; }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000; }
}
```

Ressources : utilisation dans le code

Plusieurs méthodes acceptent un ID de ressource en paramètre

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        String maChaine = getString(R.string.hello);

        TextView message = (TextView) findViewById(R.id.msgText);
        message.setText(R.string.hello);
     }
}
```

Ressources : utilisation dans d'autres ressources

res/layout/main.xml :

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

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

</LinearLayout>
```

Ressources systèmes

Dans le code :

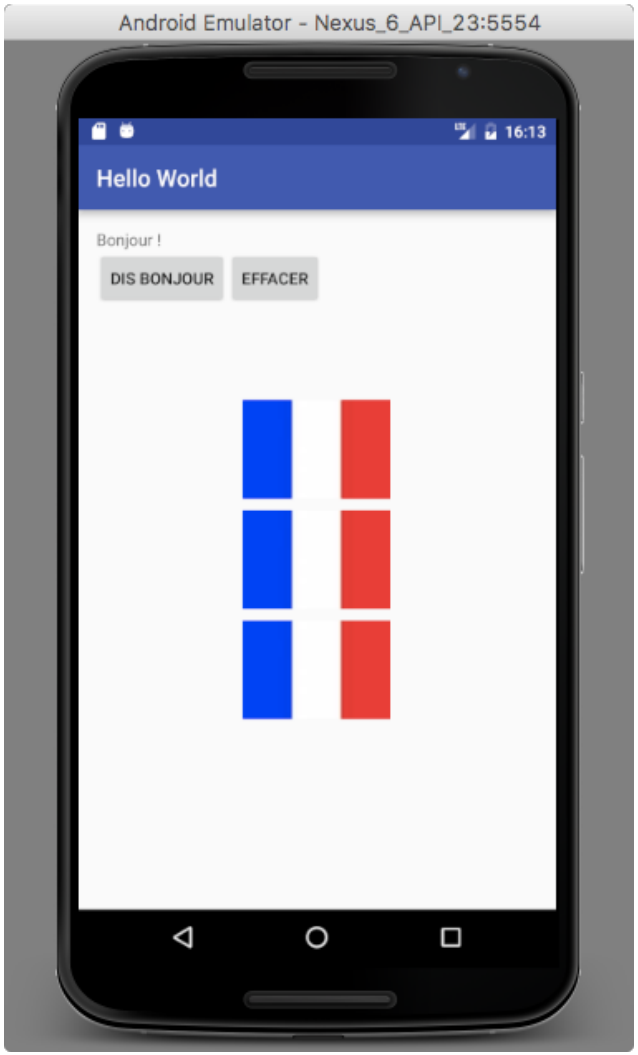
```
android.R.string.unknownName
```

Dans une autre ressource :

```
<?xml version="1.0" encoding="utf-8"?>
<EditText id="text"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text="@android:string/unknownName"/>
```

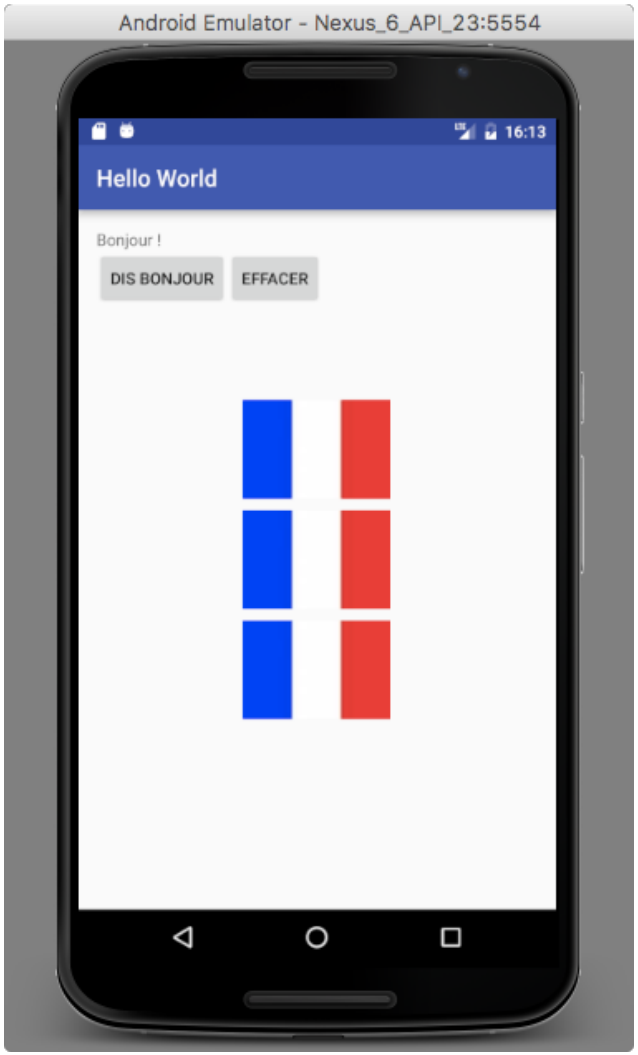
Exercice : *Hello World !*

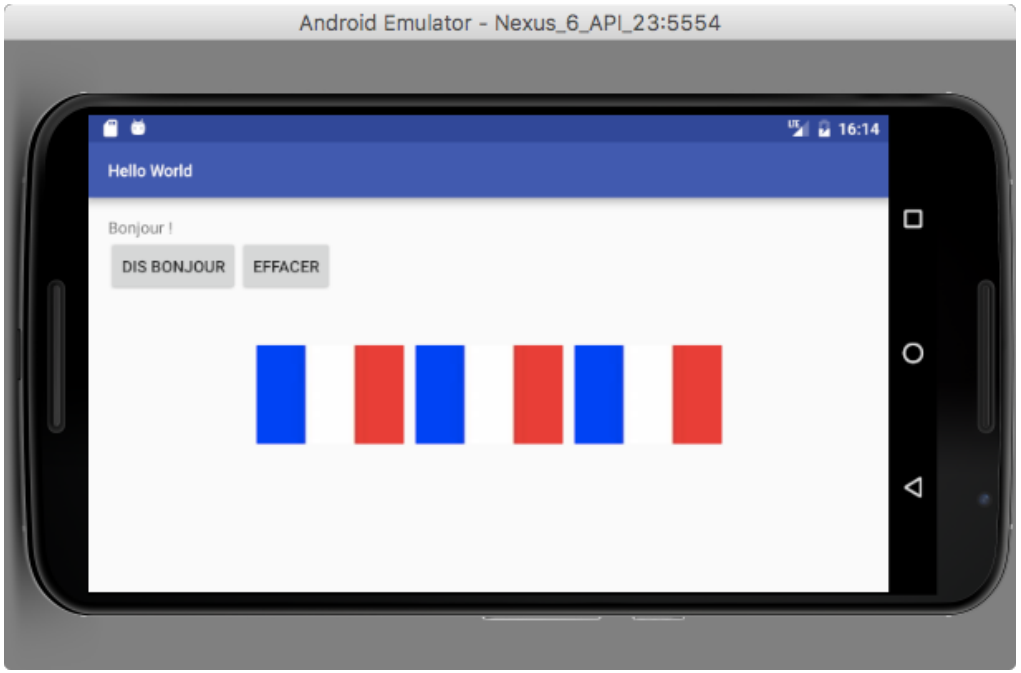
L'affichage change en fonction de la langue du terminal :



Exercice : *Hello World !*

La disposition des drapeaux change en fonction de l'orientation de l'écran :





Plan

Description des interfaces graphiques

Deux possibilités :

- description déclarative en XML (dossier `res/layout/`)
- en Java

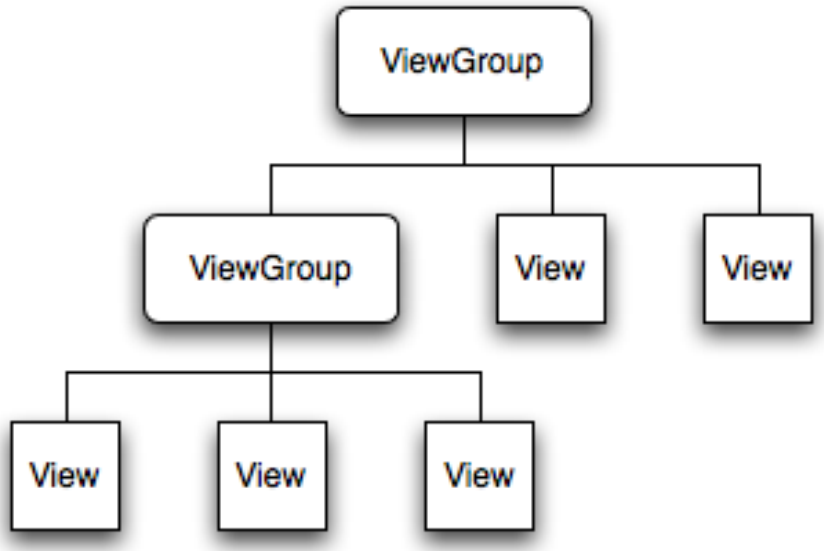
Vue

Composant élémentaire des interfaces graphiques sous Android :

- instance de la classe `View`
- rectangle où on peut dessiner, sensible aux événements
- tous les composants graphiques (*widgets*) d'Android sont des vues : boutons, étiquettes, zones de texte...

Organisation hiérarchique des vues

La classe `ViewGroup` (hérite de `View`) permet de regrouper des vues



Associer une hiérarchie de vues à une activité

Appeler la méthode `setContentView` :

- dans `onCreate`
- paramètre = une référence à la racine de la hiérarchie

Layout

- C'est une organisation particulière d'un groupe de vues
- Sous-classe de `ViewGroup`
- Android fournit plusieurs *layouts*

Quelques *layouts* fournis par Android

- `FrameLayout` : le plus basique, chaque élément est positionné dans le coin en haut à gauche, par-dessus les éléments ajoutés avant

  usage : affichage d'un seul élément

- `LinearLayout` : les éléments sont positionnés les uns à la suite des autres, de gauche à droite et de haut en bas

- `TableLayout` : les éléments sont positionnés dans un tableau

- `RelativeLayout` : les éléments sont positionnés les uns par rapport aux autres

Layout : exemple

res/layout/main.xml

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

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

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

</LinearLayout>
```

Layout : taille

Les *attributs* `layout_width` et `layout_height` peuvent contenir :

- une taille précise (unités : dp, sp, pt, px, mm, in)
- l'une des constantes :
  - `match_parent` : prendre toute la place disponible
    (`fill_parent` dans les versions d'Android < 2.2)
  - `wrap_content` : ne prendre que la place nécessaire

Définition d'une interface en XML

Chaque fichier XML décrivant l'interface :

- doit être placé dans le dossier `res/layout`
- a un identifiant unique :
  - généré automatiquement
    (ex : `monLayout.xml` → `R.layout.monLayout`)
  - permet de faire référence au fichier dans le code
    (ex : `setContentView(R.layout.monLayout)`)

Définition d'une interface en XML

res/layout/main.xml

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

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

</LinearLayout>
```

Définition d'une interface en XML

Attribut `id` :

- associe un *identifiant* unique à une vue
- le signe `+` dans `@+id` signifie qu'il s'agit d'un nouveau nom de ressource, à créer et à ajouter à la classe `R`
- identifiants fournis par le système :

```
android:id="@android:id/empty"
```

Définition d'une interface en XML

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        TextView message = (TextView) findViewById(R.id.msgText);
        message.setText("Hello, World!");
     }
}
```

Définition d'une interface en Java

Pour obtenir le même résultat que précédemment, en Java uniquement :

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView message = new TextView(this);
        message.setText("Hello, World!");
        setContentView(message);
     }
}
```

Définition d'une interface en Java : *layouts*

```
import android.app.Activity;  import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;

public class HelloWorldActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        LinearLayout mLayout = new LinearLayout(this);
        mLayout.setOrientation(LinearLayout.VERTICAL);

        TextView message1 = new TextView(this);
        TextView message2 = new TextView(this);
        message1.setText("Hello, World!");
        message2.setText("Bonjour tout le monde !");

        mLayout.addView(message1);  mLayout.addView(message2);

        setContentView(mLayout);
     }
}
```

Définition d'une interface : XML ou Java ?

- Interface en XML : séparation stricte de la présentation et de la logique fonctionnelle de l'application

  $\Rightarrow$ c'est la meilleure pratique en général

- Ce qui est faisable en XML est faisable en Java

- À chaque vue, attribut, constante... utilisé en XML correspond une vue, un attribut, une constante... en Java

Gestion des événements : attribut XML

Attribut android:onClick de la classe View

res/layout/main.xml

```
<Button ... android:onClick="sayHello" .../>
```

HelloWorldActivity.java

```
public class HelloWorldActivity extends Activity {
  private TextView helloLabel;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    helloLabel = (TextView) findViewById(R.id.hello_label);
  }
  public void sayHello(View v) {
    helloLabel.setText(R.string.hello);
  }
}
```

Gestion des événements : écouteurs

Écouteur (*listener*) :

- au cœur du mécanisme d'interception des événements
- associe un événement à une méthode à appeler lorsque l'événement survient
- les événements interceptés et les méthodes associées sont prédéfinis

  ex : événement `onClick` → méthode `onClick()`

Gestion des événements : écouteurs

res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_vertical|center_horizontal">

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

</LinearLayout>
```

Gestion des événements : écouteurs, solution 1

```
public class MyListener implements OnClickListener {
    @Override
    public void onClick(View v) {
        Toast.makeText(this, "click", Toast.LENGTH_LONG).show();
    }
}
```

```
public class MyActivity extends Activity {

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

        setContentView(R.layout.main);
        Button b = (Button) findViewById(R.id.mButton);
        b.setOnClickListener(new MyListener());
    }
}
```

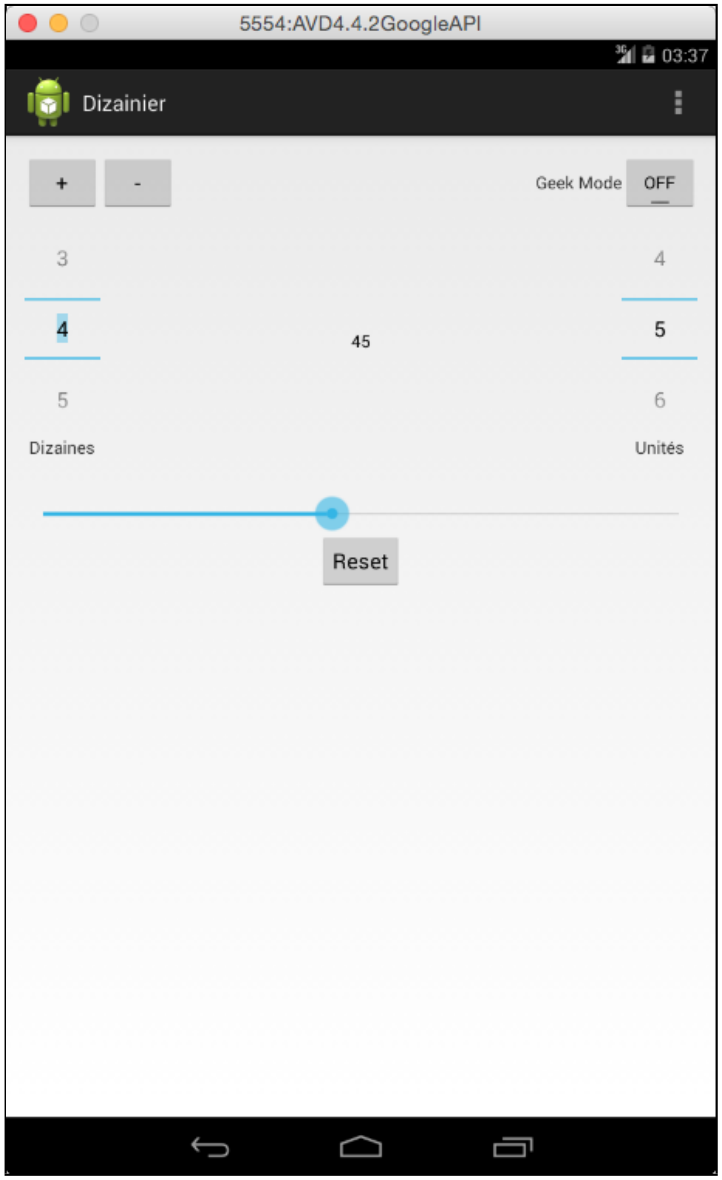
Gestion des événements : écouteurs, solution 2

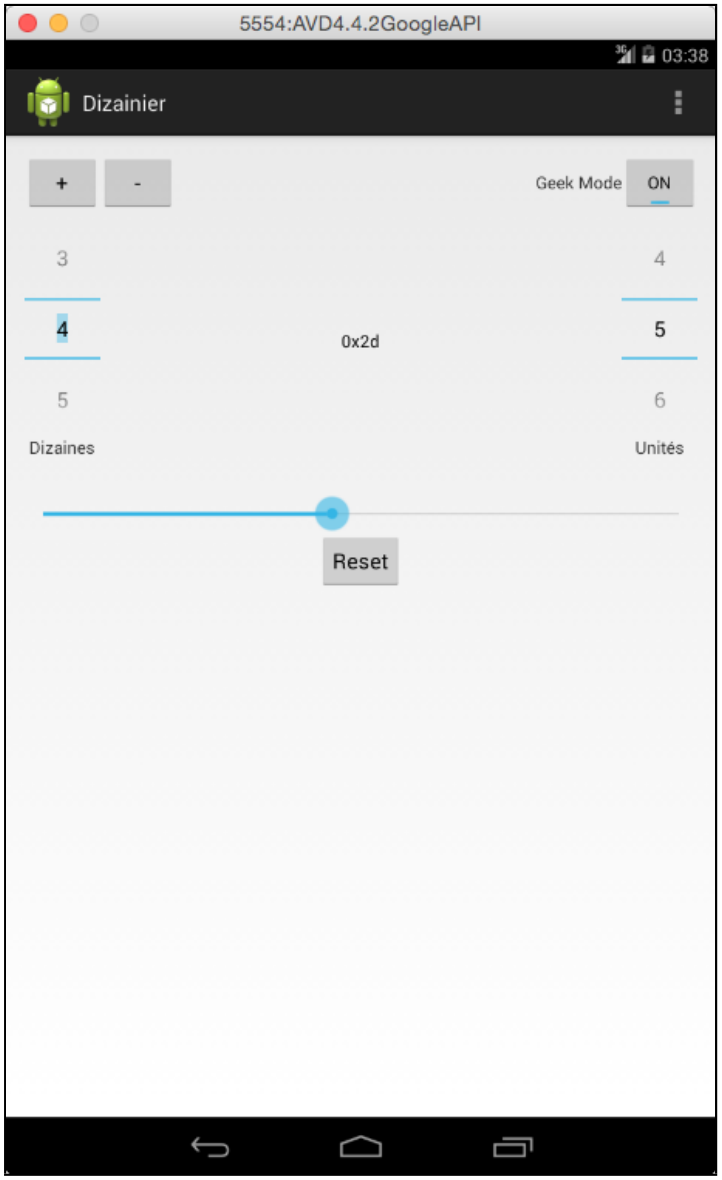
```
public class MyActivity extends Activity {

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

        setContentView(R.layout.main);
        Button b = (Button) findViewById(R.id.mButton);
        b.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(MyActivity.this, "click",
                                            Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Gestion des événements : écouteurs, solution 3

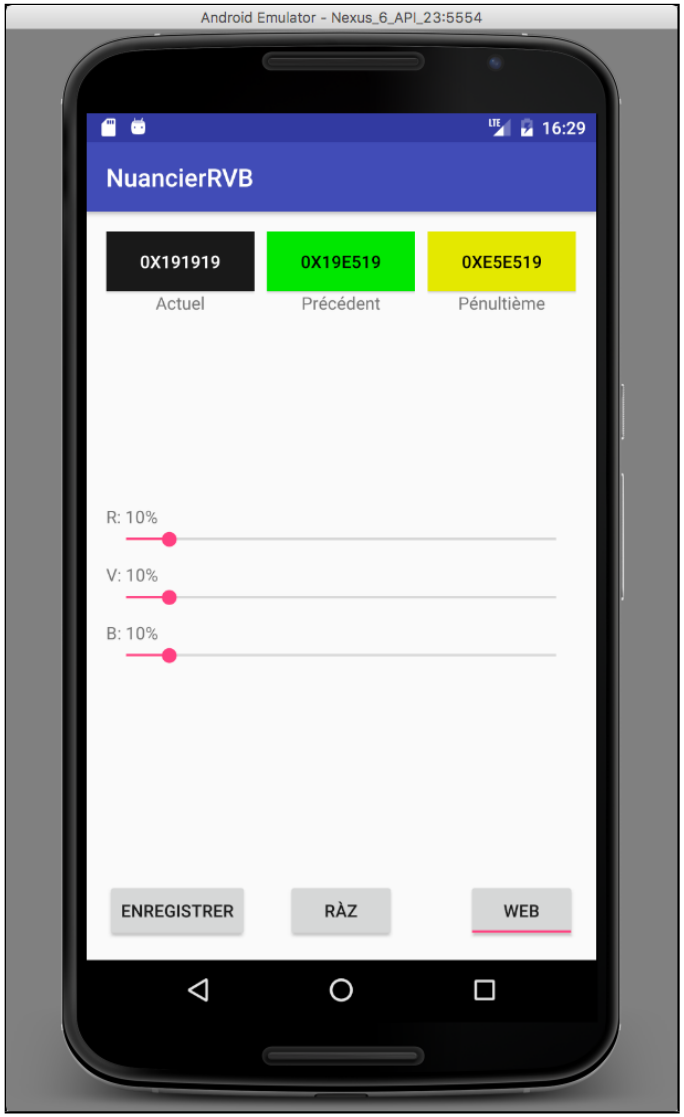
```
public class myActivity extends Activity implements OnClickListener {
    @Override
    public void onCreate(Bundle savedValues) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button b = (Button) findViewById(R.id.mButton);
        b.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        Toast.makeText(this, "click", Toast.LENGTH_LONG).show();
    }
}
```

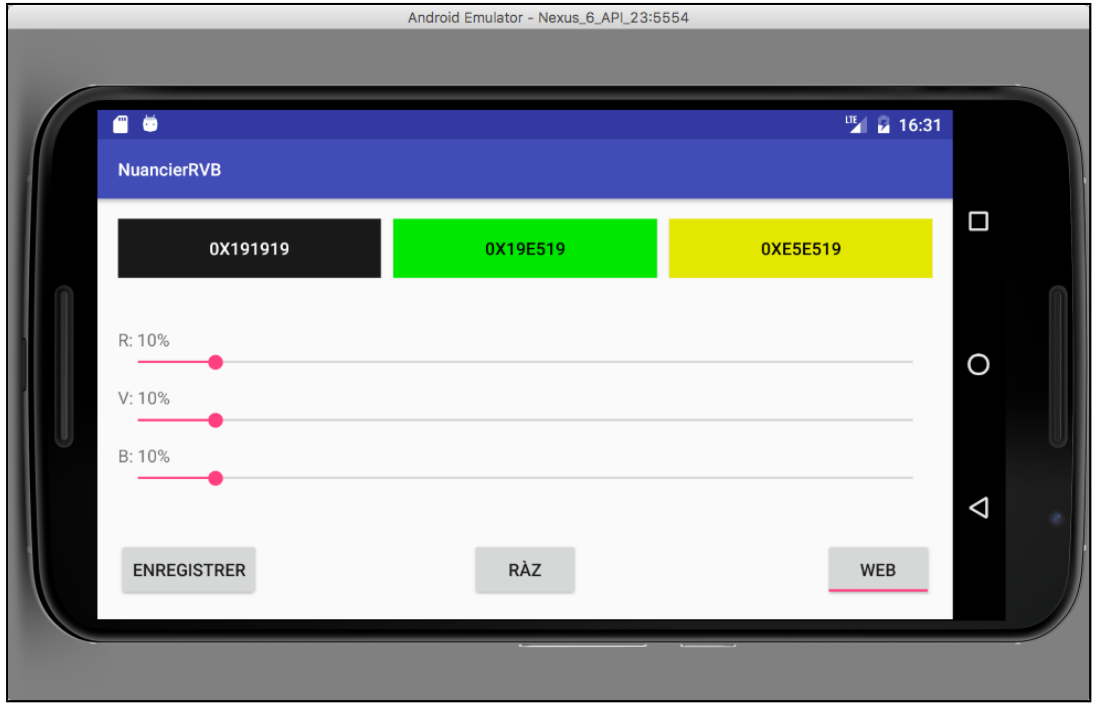
Exercice : *Dizainier*





Exercice : *Nuancier RVB*





Exercice : *Nuancier RVB*

- Les boutons colorés en haut de l'écran occupent toujours toute la largeur de l'écran
- Le titre des boutons colorés doit toujours être lisible (noir si fond clair, blanc si fond sombre, voir ici)
- Lorsque l'appareil est en mode portrait, les labels Actuel, Précédent et Pénultième sont affichés pour indiquer la signification de chaque bouton coloré
- Bien gérer l'orientation de l'écran : la couleur des boutons, les pourcentages affichés, ... doivent être identiques avant et après un changement d'orientation

Construire ses propres vues

Créer une classe héritant de `View` (ou d'une de ses sous-classes)

- 3 constructeurs :

```
public MyView(Context context, AttributeSet attrs, int defStyle)
public MyView(Context context, AttributeSet attrs)
public MyView(Context context)
```

  Le paramètre `attrs` sert lors de définitions XML d'interfaces

- Affichage :

```
protected void onDraw(Canvas canvas)
```

- Forcer le rafraîchissement :

```
void invalidate()
```

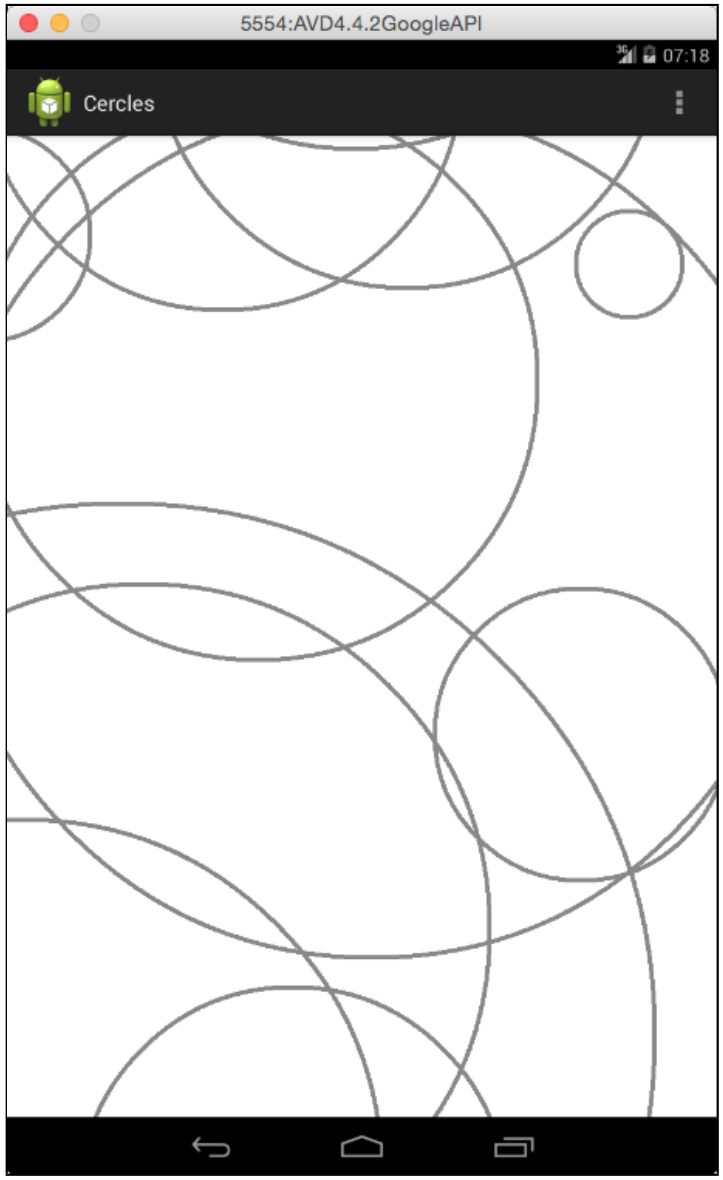
Construire ses propres vues : exemple

```
package com.example;

public class MyView extends View {

  public MyView(Context context, AttributeSet attrs) {
    super(context, attrs);
    ...
  }

  protected void onDraw(Canvas canvas) {
    int x = getWidth();
    int y = getHeight();

    Paint paint = new Paint();
    paint.setColor(android.graphics.Color.WHITE);

    canvas.drawLine(0, 0, x, y, paint);

    super.onDraw(canvas);
  }
}
```

Scroll : classes `ScrollView` et `HorizontalScrollView`

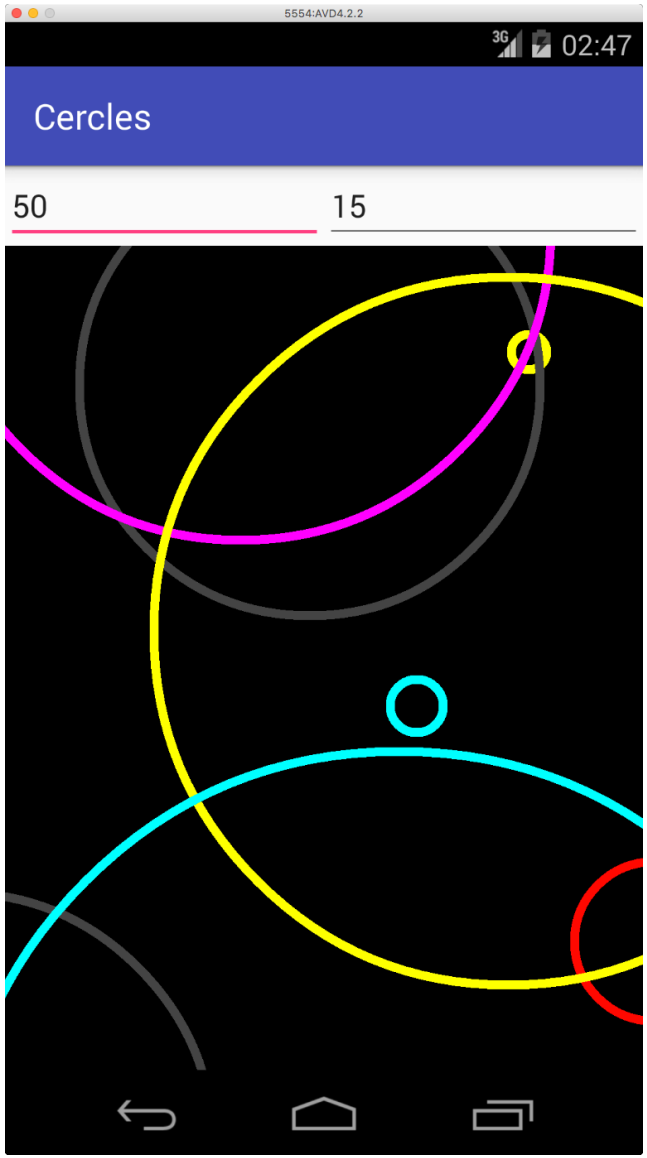
```
<ScrollView
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <HorizontalScrollView
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <LinearLayout
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:orientation="horizontal" >

      <com.example.MyView
        android:layout_width="2000dp"
        android:layout_height="3000dp" />
    </LinearLayout>

  </HorizontalScrollView>

</ScrollView>
```

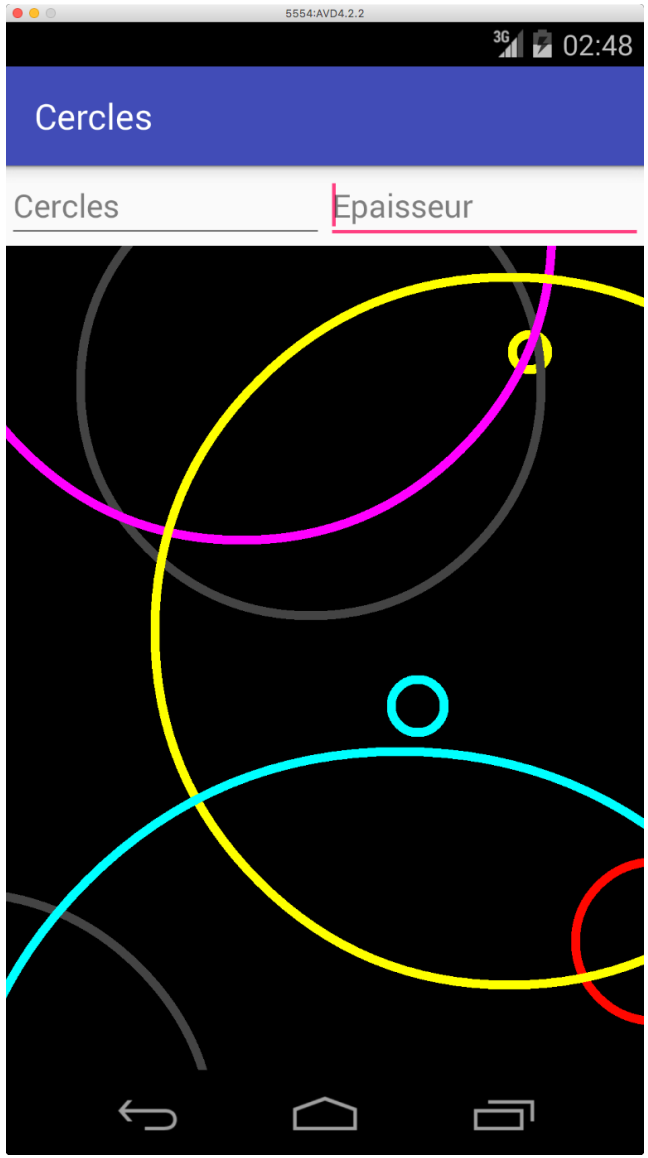
Exercice : *Cercles*



- créer une classe `CerclesView`
- compléter `onDraw` : dessiner 20 cercles gris, position au hasard, rayon croissant, épaisseur du trait : 5 pixels
- un *tap* sur l'écran efface les cercles et en dessine 20 nouveaux
- scroll vertical et horizontal

Exercice : *Cercles* (variante)





Exercice : *Cercles* (variante)

- l'utilisateur ne doit pouvoir entrer que des nombres entiers d'au plus 3 chiffres (voir attributs XML `android:inputType` et `android:maxLength`)
- gérer le cas où l'utilisateur a laissé une zone de texte vide
- anglais par défaut + proposer une version française
- tenir compte des changements d'orientation de l'écran :
  - le nombre et l'épaisseur des cercles ne doivent pas changer après un changement d'orientation
  - voir le premier chapitre du cours de M1
- zoom ?

Plan

Listes

créer une instance de ListActivity, qui intègre automatiquement :

- une vue, instance de la classe ListView
- un adaptateur, instance d'une implémentation de ListAdapter

Adaptateurs

- réalisent l'interfaçage entre l'affichage (widgets) et les données
- pour l'interfaçage avec une `ListView` : *interface* `ListAdapter`
- implémentations de `ListAdapter`, selon la source de données :
  - `ArrayAdapter` : données stockées dans un tableau
  - `CursorAdapter` : données de type `Cursor`
  - `SimpleCursorAdapter` : sur-couche de `CursorAdapter`
  - ...

Cellules à 1 ligne

les données à afficher (tableau de chaînes de caractères) :

res/values/strings.xml

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

    ...

    <string-array name="systems">
      <item>Android</item>
      <item>iOS</item>
      <item>BlackBerry</item>
      <item>Bada</item>
    </string-array>
</resources>
```

Cellules à 1 ligne

```
public class MyListActivity extends ListActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    String[] systems = getResources().getStringArray(R.array.systems);
    setListAdapter(new ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1, systems));
  }

  @Override
  protected void onListItemClick(ListView l, View v, int position, long id) {
     super.onListItemClick(l, v, position, id);
     Toast.makeText(this, "item " + id + " clicked", Toast.LENGTH_SHORT).show();
  }
}
```

Cellules à 2 lignes

```
public class MyListActivity extends ListActivity  {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ArrayList<Map<String,String>> list = new ArrayList<Map<String,String>>();

    Map<String, String> maplist;
    maplist = new HashMap<String, String>();
    maplist.put("ligne1", "Prendre une douche");
    maplist.put("ligne2", "priorité: 0");
    list.add(maplist);
    maplist = new HashMap<String, String>();
    maplist.put("ligne1", "Faire une sieste");
    maplist.put("ligne2", "priorité: 2");
    list.add(maplist);

    String[] from = { "ligne1", "ligne2" };
    int[] to = { android.R.id.text1, android.R.id.text2 };
    SimpleAdapter adapter = new SimpleAdapter(this, list,
                            android.R.layout.simple_list_item_2, from, to);
    setListAdapter(adapter);
  }
```

Cellules personnalisées

res/layout/cell_layout.xml :

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

  <ImageView android:id="@+id/image"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:src="@drawable/icon" />

  <TextView android:id="@+id/label"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20dp" />
</LinearLayout>
```

Cellules personnalisées

```
public class MyListActivity extends ListActivity {

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    String[] values = new String[] { "Android", "iOS",
                                     "Blackberry", "Bada" };
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                               R.layout.cell_layout, R.id.label, values);
    setListAdapter(adapter);
  }

}
```

Adaptateurs personnalisés

```
public class MyArrayAdapter extends ArrayAdapter<String> {

  private final Context context;

  public MyArrayAdapter(Context context, String[] values) {
    super(context, R.layout.cell_layout, values);
    this.context = context;
  }

  public View getView(int position, View convertView, ViewGroup parent)
  {
    View cellView = convertView;
    if (cellView == null) {
      LayoutInflater inflater = (LayoutInflater) context
                  .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
      cellView = inflater.inflate(R.layout.cell_layout, parent, false);
    }
    ...
```

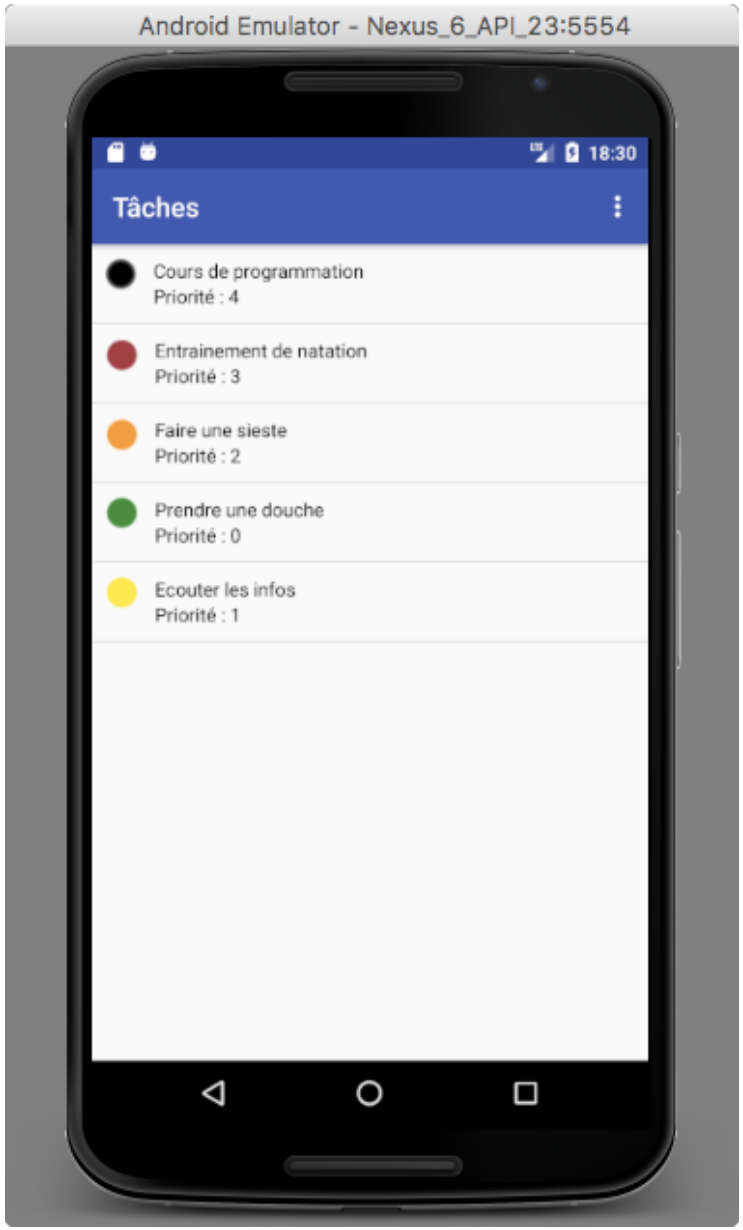
Adaptateurs personnalisés

```
    ...
    TextView textView = (TextView)cellView.findViewById(R.id.label);
    ImageView imageView = (ImageView)cellView.findViewById(R.id.image);

    String s = getItem(position);
    textView.setText(s);
    if (s.startsWith("B"))
      imageView.setImageResource(R.drawable.icon0);
    else imageView.setImageResource(R.drawable.icon1);

    return cellView;

  } // fin de la méthode getView

} // fin de la classe MyArrayAdapter
```

Adaptateurs personnalisés

```
public class MyListActivity extends ListActivity {

  private String tasks[] = { "Prendre une douche", "Faire une sieste" };

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setListAdapter(new MyArrayAdapter(this, tasks));
  }

  ...

}
```

Exercice : *Tâches*



- créer une classe `Tache` :
  - `String titre`
  - `int priorite` ($\in [0, 4]$)
- créer un type de cellule personnalisé
- créer un adaptateur personnalisé

Créer un menu pour une activité

- dans l'activité, implémenter les méthodes suivantes :
  - pour créer les éléments du menu :

    ```
    public boolean onCreateOptionsMenu(Menu menu)
    ```

  - pour réagir aux clics sur les éléments du menu :

    ```
    public boolean onOptionsItemSelected(MenuItem item)
    ```

- dans `res/menu` créer un fichier xml décrivant les items du menu

Créer un menu pour une activité : exemple

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

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
    case R.id.action_settings:
      ...
      return true;
    }
    return super.onOptionsItemSelected(item);
  }
}
```

Créer un menu pour une activité : exemple

fichier `optionsmenu.xml` :

```
<menu ...>

   <item
     android:id="@+id/action_settings"
     android:orderInCategory="100"
     android:title="@string/action_settings"/>
</menu>
```

Menu contextuel pour une activité

dans l'activité, implémenter les méthodes suivantes :

- pour créer les éléments du menu :

```
public boolean onCreateContextMenu(ContextMenu menu,
                                   View v,
                                   ContextMenuInfo menuInfo)
```

- pour réagir aux clics sur les éléments du menu :

```
public boolean onContextItemSelected(MenuItem item)
```

et dans `onCreate` enregistrer les vues réagissant au menu :

- `registerForContextMenu(une vue)`

Menu contextuel pour une activité : exemple

```
public class MyListActivity extends ListActivity {

  private ArrayAdapter<String> adapter;

  @Override
  public void onCreate(Bundle savedInstanceState) {
     ...
     adapter = ...;
     setListAdapter(adapter);
     registerForContextMenu(getListView());
  }

  ...
```

Menu contextuel pour une activité : exemple

```
...
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   getMenuInflater().inflate(R.menu.contextmenu, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
   AdapterContextMenuInfo menuInfo =
                  (AdapterContextMenuInfo) item.getMenuInfo();
   switch(item.getItemId()) {
   case R.id.action_delete:
        adapter.remove(adapter.getItem(menuInfo.position));
        return true;
   }
   return super.onContextItemSelected(item);
}
...
```

Menu contextuel pour une activité : exemple

fichier `contextmenu.xml` :

```
<menu ...>
    <item
        android:id="@+id/action_delete"
        android:orderInCategory="100"
        android:title="@string/action_delete"/>
</menu>
```

Exercice : *Tâches*

menus permettant :

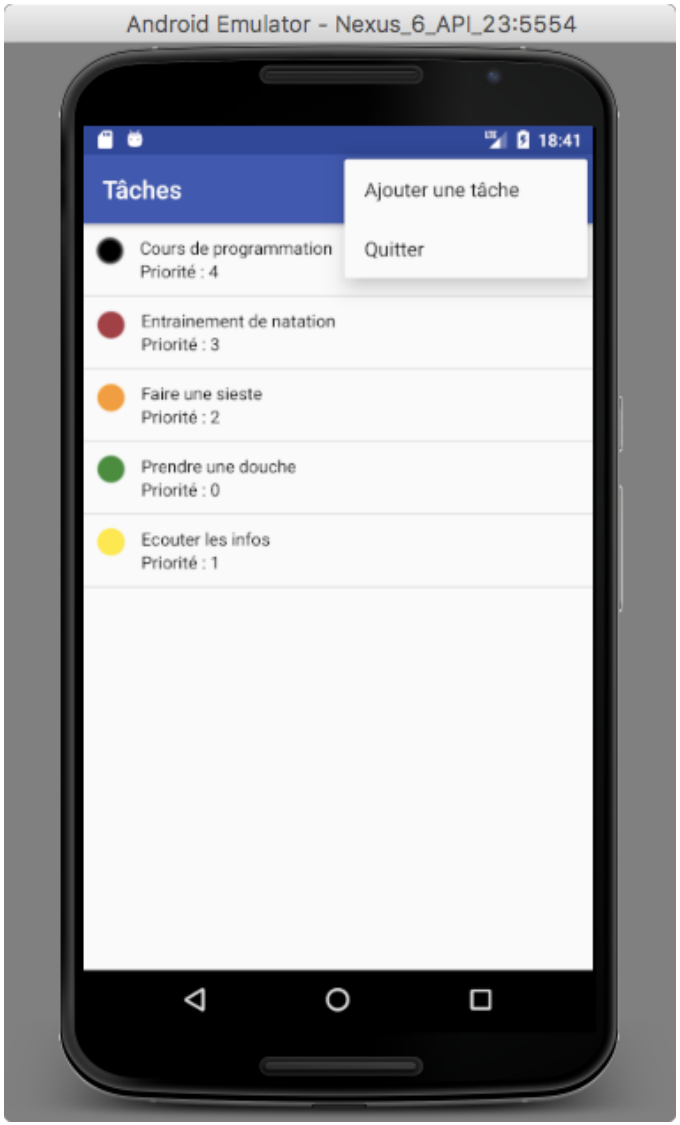
- d'ajouter une tâche (titre et priorité fixés)
- de supprimer une tâche sélectionnée
- de quitter l'application

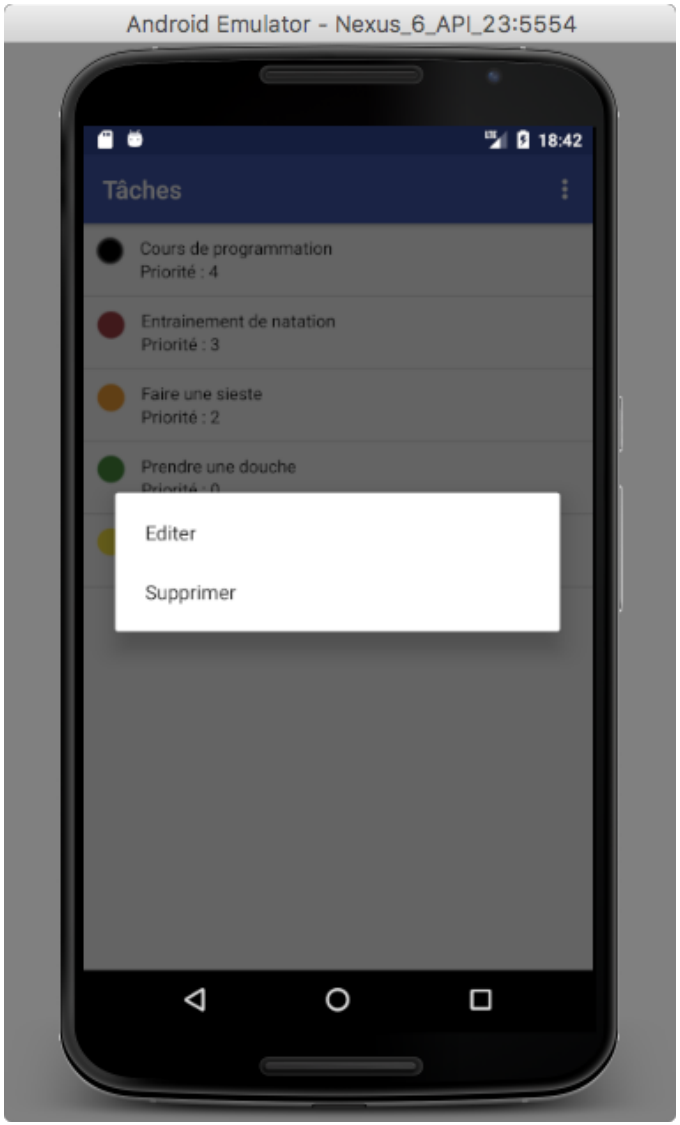
utiliser les méthodes de la classe `ArrayAdapter` :

- `add`
- `remove`

pour ajouter et supprimer un élément de la source de données de l'adaptateur (l'affichage est rafraîchi automatiquement)

Exercice : *Tâches*





Plan

Navigation entre écrans

- un écran = une activité
- passage d'un écran à un autre = une activité en démarre une autre
- objet de la classe `Intent` pour le démarrage d'une activité

Démarrer une activité de façon explicite

- déclarer l'activité à démarrer dans `AndroidManifest.xml` :

```
<application ...>
    <activity android:name=".ActiviteADemarrer" ...>
```

- démarrage de l'activité appelée dans l'activité appelante :

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

Démarrer une activité et obtenir un retour

- identification de l'activité appelée dans l'activité appelante :

```
private static final int ACTIVITE_A_DEMARRER = 1;
```

- démarrage de l'activité appelée dans l'activité appelante :

```
Intent intent = new Intent(this, ActiviteADemarrer.class);
startActivityForResult (intent, ACTIVITE_A_DEMARRER);
```

- traitement du retour dans l'activité appelante :

```
@Override
protected void onActivityResult (int requestCode,
                                 int resultCode, Intent data) {
  switch(requestCode) {
  case ACTIVITE_A_DEMARRER:
    switch(resultCode) {
    case RESULT_OK: ...
    case RESULT_CANCELED: ...
```

Démarrer une activité et obtenir un retour

dans l'activité appelée :

```
@Override
public void onClick(View v) {
  switch (v.getId()) {
  case R.id.buttonOK:
    setResult(RESULT_OK);
    finish();
    break;
  case R.id.buttonCancel:
    setResult(RESULT_CANCELED);
    finish();
    break;
  }
}
```

Échange de données entre activités

- association d'une clé (chaîne de caractères) à la donnée à échanger
- ajouter une donnée à une instance de `Intent` :

  méthode `putExtra(clé, donnée)`
- récupérer une donnée intégrée à une instance de `Intent` :

  `intent.getExtras().getXXX(clé)` où `XXX` est le type de la donnée

Échange de données entre activités

envoyer des données à l'activité appelée :

- `startActivity(Intent intent)`
- `startActivityForResult(Intent intent, int requestCode)`
- méthode `getIntent()` de la classe `Activity`

envoyer des données à l'activité appelante :

- `setResult(int resultCode, Intent data)`
- `onActivityResult(int requestCode, int resultCode, Intent data)`

Exercice : *Tâches*

- item ajouter (menu *option*) : lance une activité permettant à l'utilisateur de saisir une nouvelle tâche (titre et priorité) et intégrant un bouton OK et un bouton Annuler
- item éditer (menu *contextuel*) : lance une activité permettant à l'utilisateur de modifier la tâche sur laquelle il a appuyé et intégrant un bouton OK et un bouton Annuler

Exercice

Plan

Sauvegarde de données

trois possibilités :

- classe `SharedPreferences` : sauvegarde d'une petite quantité de paires (clé,valeur) dans un fichier spécial (par ex. sauvegarde du meilleur score d'un jeu)
- sauvegarde dans un fichier
- sauvegarde dans une base de données relationnelle (méthode décrite dans ce chapitre)

voir `http://developer.android.com/training/basics/data-storage/index.html`

SQLite

- système de gestion de bases de données relationnelles
- http://www.sqlite.org

Gestion d'une base de données

créer une classe adaptateur qui :

- encapsule la base de données (instance de `SQLiteDatabase`)
- encapsule une instance de `SQLiteOpenHelper`
- propose des méthodes pour l'ouverture et la fermeture de la base
- propose des méthodes pour obtenir, ajouter, supprimer, ... des données dans la base

classe SQLiteOpenHelper

aide à créer, ouvrir et mettre à jour une base de données SQLite :

- `public void onCreate(SQLiteDatabase db)`
- `public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`
- `public SQLiteDatabase getReadableDatabase()`
- `public SQLiteDatabase getWritableDatabase()`

Classe adaptateur : exemple

```
public class MyDBAdapter {

  public static final int DB_VERSION = 1;
  public static final String DB_NAME = "clients_database.db";

  private static final String TABLE_CLIENTS = "table_clients";
  public static final String COL_ID = "_id";
  public static final String COL_NAME = "name";
  public static final String COL_ADDR = "address";

  private static final String CREATE_DB =
      "create table " + TABLE_CLIENTS + " ("
      + COL_ID + " integer primary key autoincrement, "
      + COL_NAME + " text not null, "
      + COL_ADDR + " text not null);";

  private SQLiteDatabase mDB;
  private MyOpenHelper mOpenHelper; // MyOpenHelper : voir plus bas
```

Classe adaptateur : exemple

```
...

public MyDBAdapter(Context context) {
    mOpenHelper = new MyOpenHelper(context, DB_NAME,
                                                null, DB_VERSION);
}

public void open() {
    mDB = mOpenHelper.getWritableDatabase();
}

public void close() {
    mDB.close();
}
```

Requête de sélection sur une table

méthode `query` de la classe `SQLiteDatabase` :

```
public Cursor query ( String table,
                      String[] columns,
                      String where,
                      String[] whereArgs,
                      String groupBy,
                      String having,
                      String orderBy )
```

Requête de sélection sur une table

la méthode `query` renvoie une instance de la classe `Cursor` qui

- permet de gérer un ensemble de données fournies par une requête
- intègre une gestion optimisée des ressources

Gestion d'un curseur

méthodes de la classe `Cursor` :

- navigation dans un curseur : `moveToFirst()`, `moveToLast()`, `moveToNext()`, `moveToPrevious()`, ...
- informations sur un curseur : `getCount()`, ...
- extraction de valeurs d'un curseur : `getXXX(i)` où `i` est l'indice de la colonne et `XXX` est le type de la valeur
- fermeture d'un curseur : `close()`

Classe adaptateur : exemple

on suppose écrite une classe `Client` de la forme :

```
public class Client {
    private long id;
    private String name;
    private String address;

    public Client(long id, String name, String address) { ... }

    ...
}
```

Classe adaptateur : exemple

suite de la classe `MyDBAdapter` :

```
...
public Client getClient(long id) throws SQLException {
    Client cl = null;

    Cursor c = mDB.query(TABLE_CLIENTS,
          new String [] {COL_ID, COL_NAME, COL_ADDR},
          COL_ID + " = " + id, null, null, null, null);

    if (c.getCount() > 0) {
        c.moveToFirst();
        cl = new Client(c.getLong(0), c.getString(1), c.getString(2));
    }
    c.close();

    return cl;
}
```

Classe adaptateur : exemple

```
...

public List<Client> getAllClients() {
    List<Client> clients = new ArrayList<Client>();

    Cursor c =  mDB.query(TABLE_CLIENTS,
          new String[] {COL_ID, COL_NAME, COL_ADDR},
          null, null, null, null, null);

    c.moveToFirst();
    while (!c.isAfterLast()) {
        clients.add(new Client(
              c.getLong(0), c.getString(1), c.getString(2)));
        c.moveToNext();
    }
    c.close();

    return clients;
}
```

Insertion et modification d'une ligne

on utilise une instance de la classe `ContentValues`

- permet de spécifier les valeurs de la ligne
- encapsule une collection de couples (`nom de colonne`, `valeur`)
- méthode `put` pour indiquer la valeur d'une colonne

Classe adaptateur : exemple

```
...
public long insertClient(String name, String address) {
    ContentValues values = new ContentValues();
    values.put(COL_NAME, name);
    values.put(COL_ADDR, address);
    return mDB.insert(TABLE_CLIENTS, null, values);
}

public int updateClient(long id, String name, String address) {
    ContentValues values = new ContentValues();
    values.put(COL_NAME, name);
    values.put(COL_ADDR, address);
    return mDB.update(TABLE_CLIENTS, values, COL_ID + "=" + id, null);
}

public int removeClient(long id) {
    return mDB.delete(TABLE_CLIENTS, COL_ID + " = " + id, null);
}
```

Classe adaptateur : exemple

```
  ...
  private class MyOpenHelper extends SQLiteOpenHelper {
     public MyOpenHelper(Context context, String name,
                         CursorFactory cursorFactory, int version) {
        super(context, name, cursorFactory, version);
     }
     @Override
     public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_DB);
     }
     @Override
     public void onUpgrade(SQLiteDatabase db,
                         int oldVersion, int newVersion) {
        db.execSQL("drop table " + TABLE_CLIENTS + ";");
        onCreate(db);
     }
  }
} // fin de la classe MyDBAdapter
```

Afficher les données d'une table sous forme de liste

on procède comme précédemment (voir chapitre 6)

```
public class MyListActivity extends ListActivity {

  private MyDBAdapter dbAdapter;

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

     dbAdapter = new MyDBAdapter(this);
     dbAdapter.open();

     setListAdapter(new MyArrayAdapter(
             this, dbAdapter.getAllClients()));
  }
```

Afficher les données d'une table sous forme de liste

```
...

  @Override
  public void onDestroy() {
      dbAdapter.close();
      super.onDestroy();
  }

} // fin de la classe MyListActivity
```

Exercices

- tutoriel *Notepad Tutorial* sur `http://developer.android.com`
- application *Tâches* : rendez les tâches persistantes