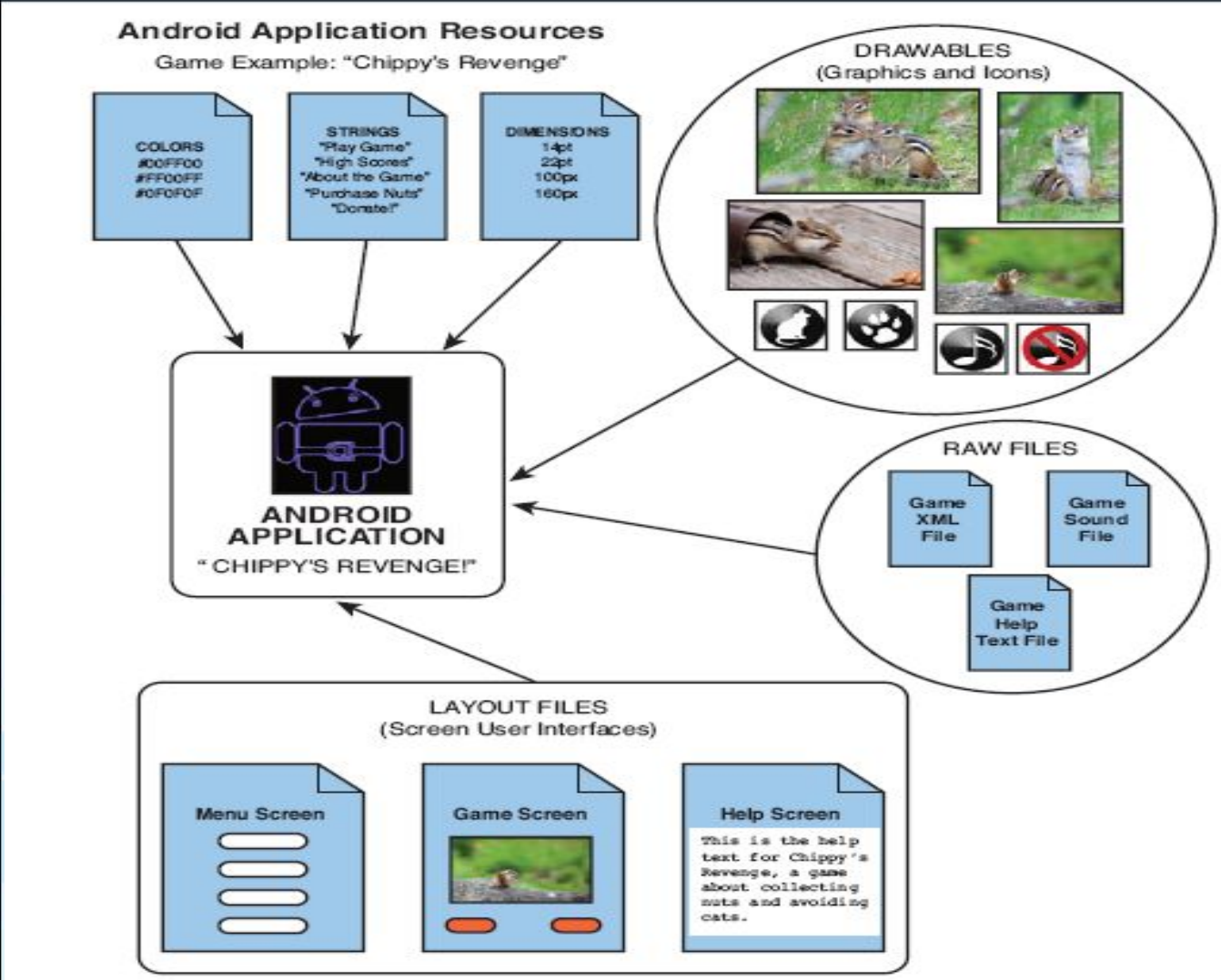
Partie III : Développement (suite)

- Les IHM
  - Principe
  - Layout
  - Taille d'écran
  - Fragments
- Les services
- Le stockage des données
  - XML
  - SQLite
  - ContentProvider


IHM - Principes

IHM : Principes de base

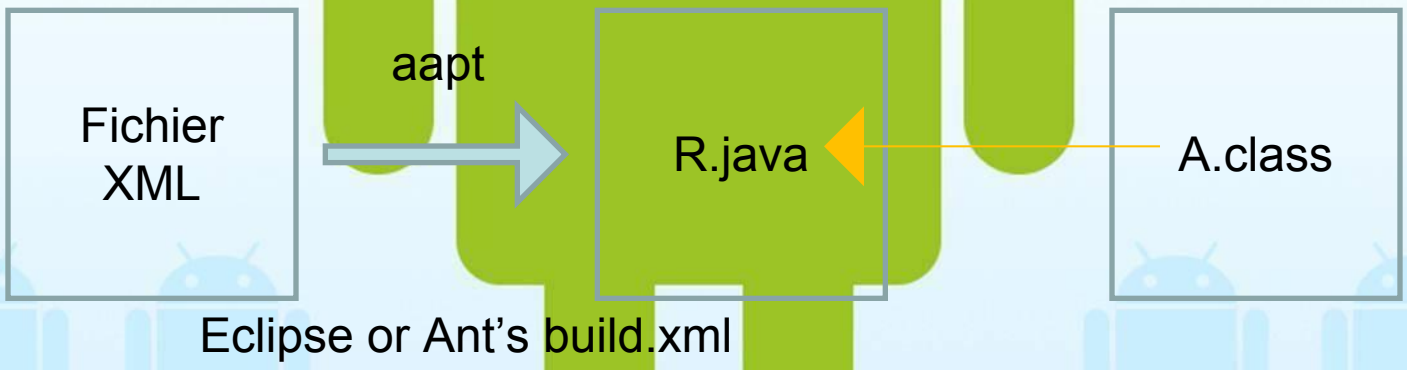


Une IHM graphique est composée d'objets graphiques, de textes …nous allons voir dans ce chapitre la construction des IHM

Using XML-Based Layouts

- Les IHM sont liées aux activités
- Il est possible de coder toutes les IHM en Java avec du code intégré dans les activités : si affichage dynamique

Il est préférable d'utiliser une description XML hiérarchique pour décrire les composants graphiques



Au moment de la compilation les fichiers XML sont parsés et compilés en un fichier binaire : il n'est donc pas possible de modifier le contenu.

Using XML-Based Layouts

Structure MV ou le code est dissocié du graphique.

Le format XML – GUI est devenu un standard : Microsoft's Extensible Application Markup Language (XAML), Adobe's Flex, and Mozilla's XML User.Interface Language (XUL)

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

A quoi ressemble cette IHM?

Using XML-Based Layouts

- Il s'agit d'un objet « button »
- L'élément racine déclare le namespace Android XML :

xmlns:android=http://schemas.android.com/apk/res/android

- Tous les autres éléments de la racine héritent de ce namespace

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

Using XML-Based Layouts

- Depuis le code Java nous avons besoin d'un identifiant avec android:id attribute. Tous les widget sont identifiés de cette façon.

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

Using XML-Based Layouts

- android:text: indique le texte à afficher dans le bouton (ici le texte est vide)

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

Using XML-Based Layouts

- android:layout_width et android:layout_height: Indique que la largeur et hauteur sont identiques à l'objet parent c'est à dire tout l'écran.

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

Using XML-Based Layouts

- **Comment attacher le layout au code Java ?**

```
Chargement du layout --> setContentView(R.layout.main);
Utiliser --> findViewById();
Utilisation du widget --> Button myButton = (Button) findViewById(R.id.button);
```

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

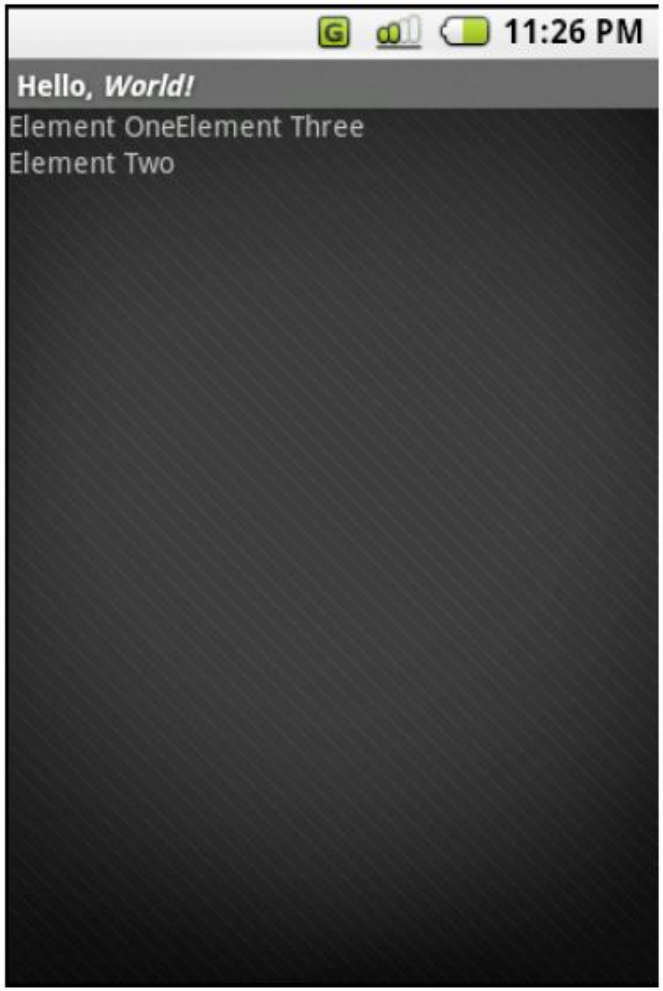
Using XML-Based Layouts

- Dans une activité, charger les ressources dans la méthode onCreate()

```
package com.commonsware.android.layouts;
import android.app.Activity;
import java.util.Date;
…
public class NowRedux extends Activity
implements View.OnClickListener {
Button btn;

@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);
btn=(Button)findViewById(R.id.button);
btn.setOnClickListener(this);
updateTime();
}
```

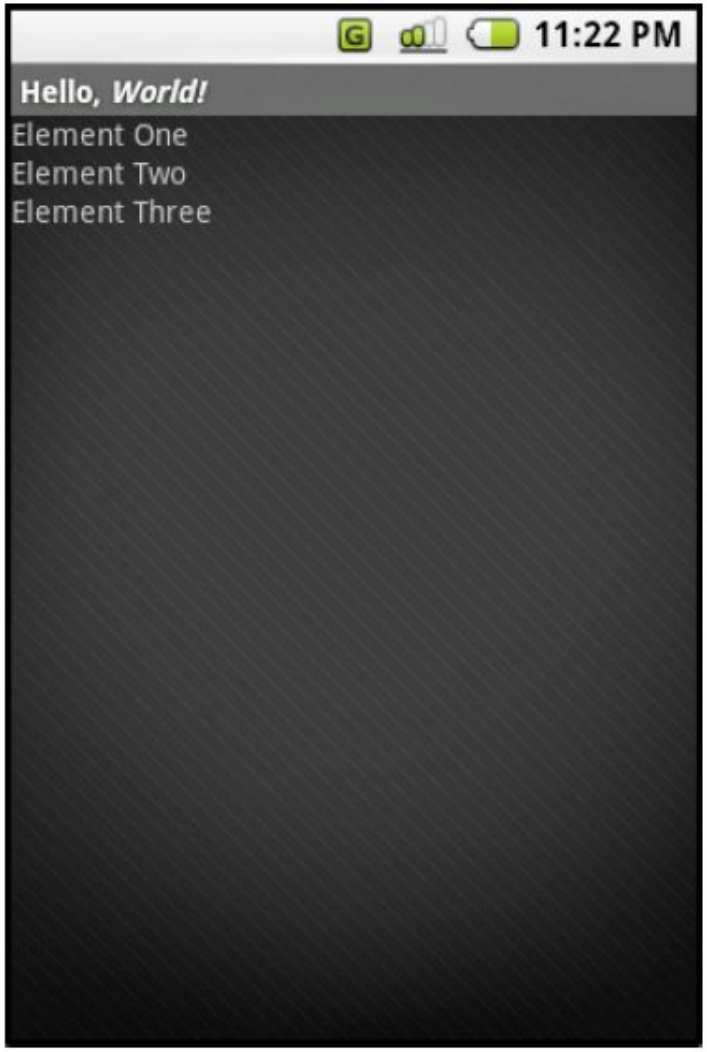
IHM : Principes de base

RelativeLayout




```
<RelativeLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:id="@+id/EL01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element One"
/>
<TextView
android:id="@+id/EL02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element Two"
android:layout_below="@id/EL01"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element Three"
android:layout_toRight="@id/EL02"
/>
</RelativeLayout>
```

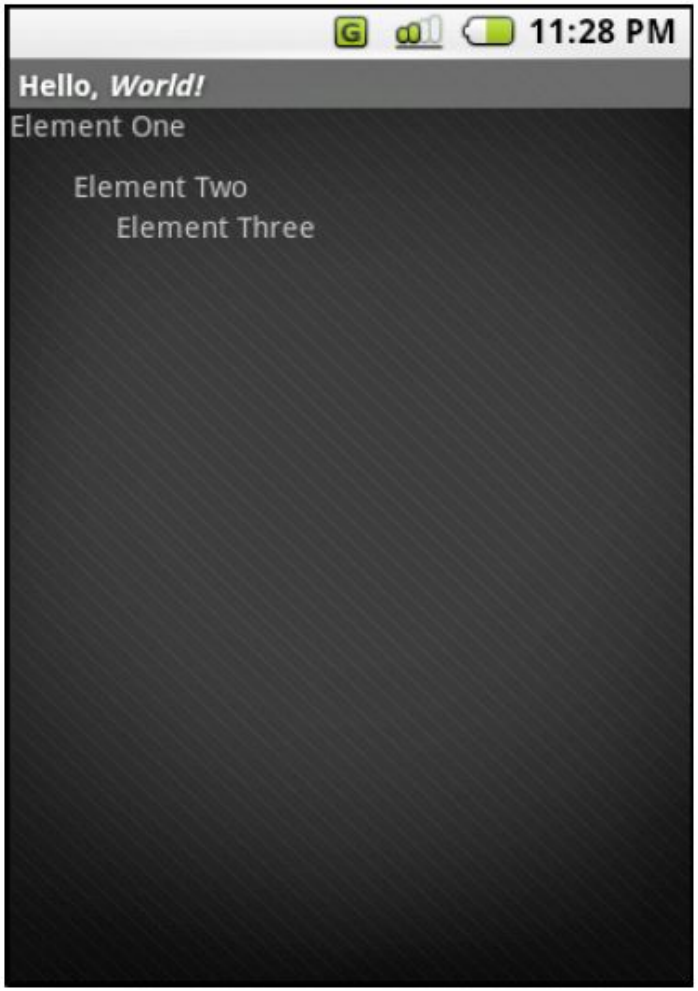
IHM : Principes de base



LinearLayout

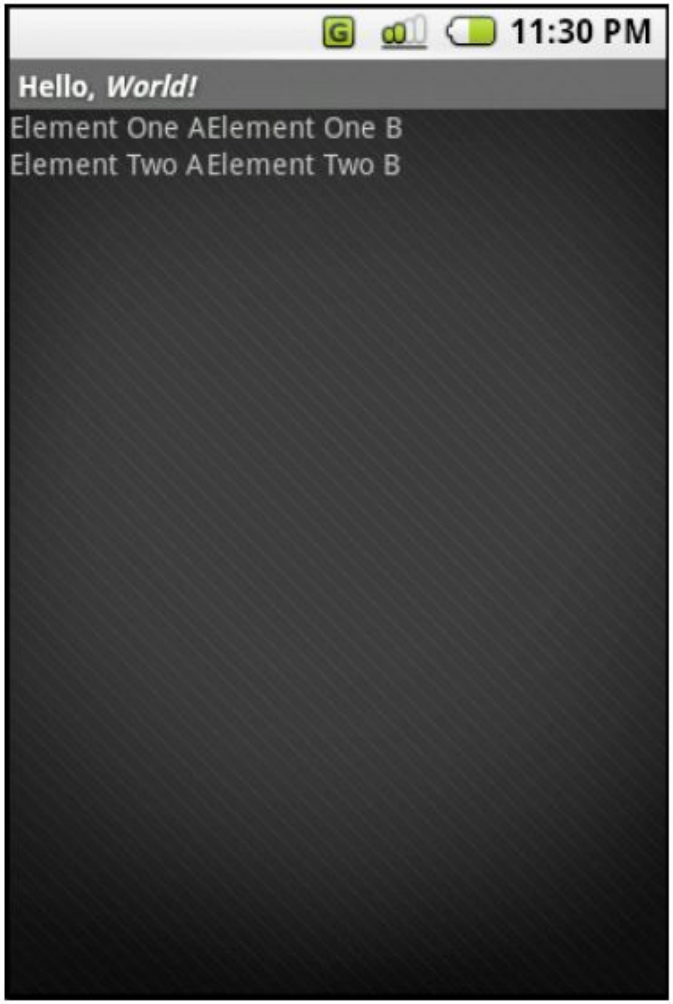
```
<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element One"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element Two"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element Three"
/>
</LinearLayout>
```

AbsoluteLayout Do not use `AbsoluteLayout` (it's deprecated)




```
<AbsoluteLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element One"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element Two"
android:layout_x="30px"
android:layout_y="30px"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element Three"
android:layout_x="50px"
android:layout_y="50px"
/>
</AbsoluteLayout>
```

IHM : Principes de base



TableLayout

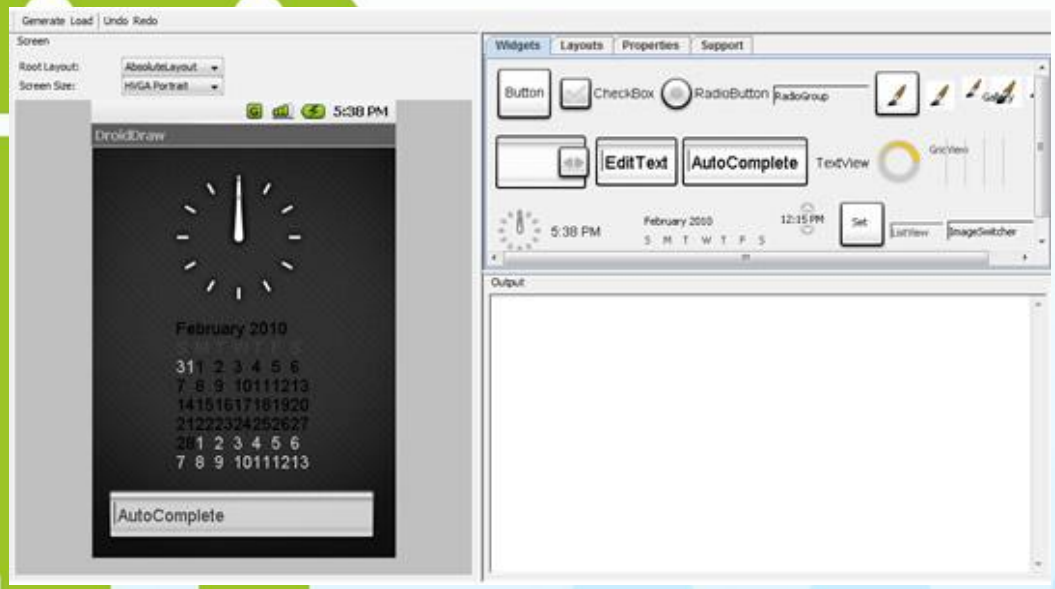
```
<TableLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TableRow>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element One A"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element One B"
/>
</TableRow>
<TableRow>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element Two A"
/>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Element Two B"
/>
</TableRow>
</TableLayout>
```

IHM : Principes de base

Générateur IHM

http://www.droiddraw.org/

http://android-ui-utils.googlecode.com/



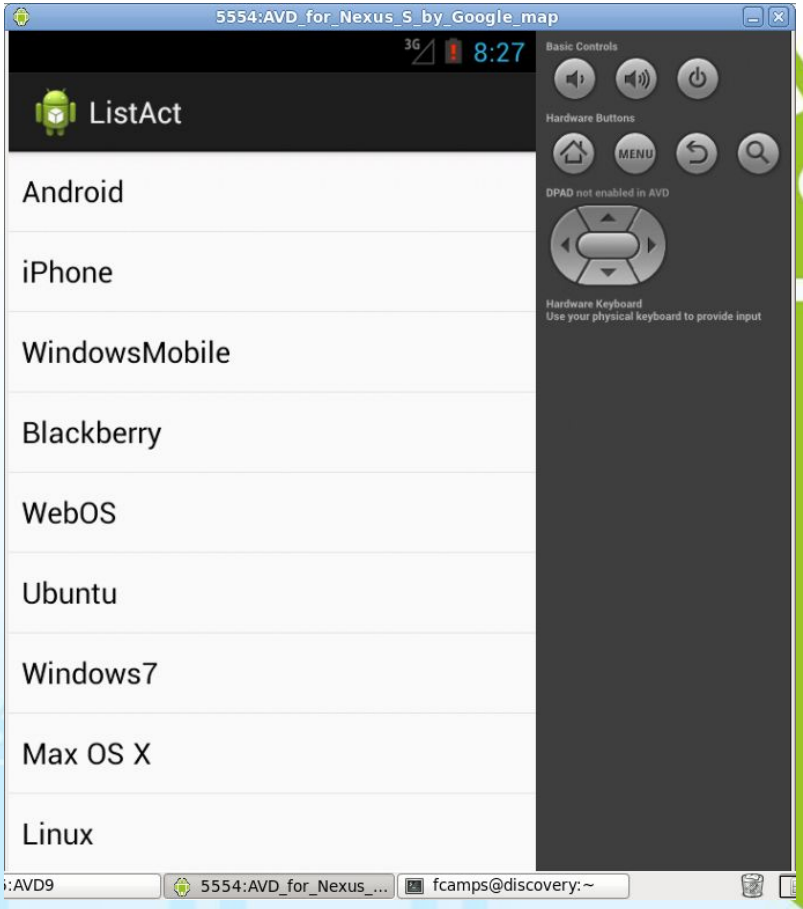
IHM : Principes de base

Générateur IHM

http://pencil.evolus.vn/Features.html

IHM : Principes de base

ListActivity




```
public class MainActivity extends ListActivity {
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    String[] values = new String[] { "Android",
"iPhone", "WindowsMobile",
        "Blackberry", "WebOS", "Ubuntu", "Windows7",
"Max OS X",
        "Linux", "OS/2" };
    ArrayAdapter<String> adapter = new
ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, values);
    setListAdapter(adapter);
  }
}
```

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity" >
<ListView
 android:id="@android:id/list"
 android:layout_width="match_parent"
 android:layout_height="wrap_content" >
</ListView>
</RelativeLayout>
```

Event Listeners

Exemple : listener / Button

Les listener sont créés dans le code de façon dynamique

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
}
```

Event Listeners

Exemple : listener / Button

Il est préférable d'implémenter des interfaces pour éviter des allocations de code dans des classes externes.

```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

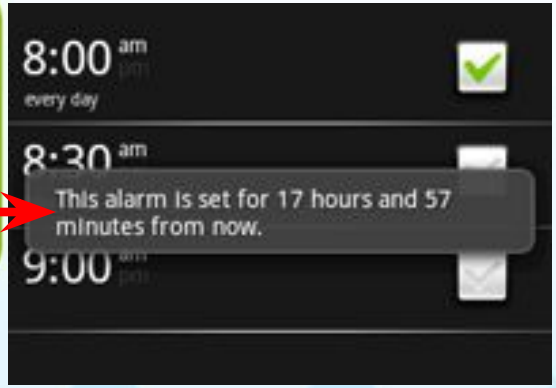
IHM : Principes de base

Récupérer les données utilisateur


```
//Grab handles to both text-entry fields
EditText username =
(EditText) findViewById(R.id.username);
EditText pwd =
(EditText) findViewById(R.id.password);
//Extract Strings from the EditText objects
// and format them in strings
String usrTxt = username.getText().toString();
String pwdTxt = pwd.getText().toString();
```

Creating Toast Notifications

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```



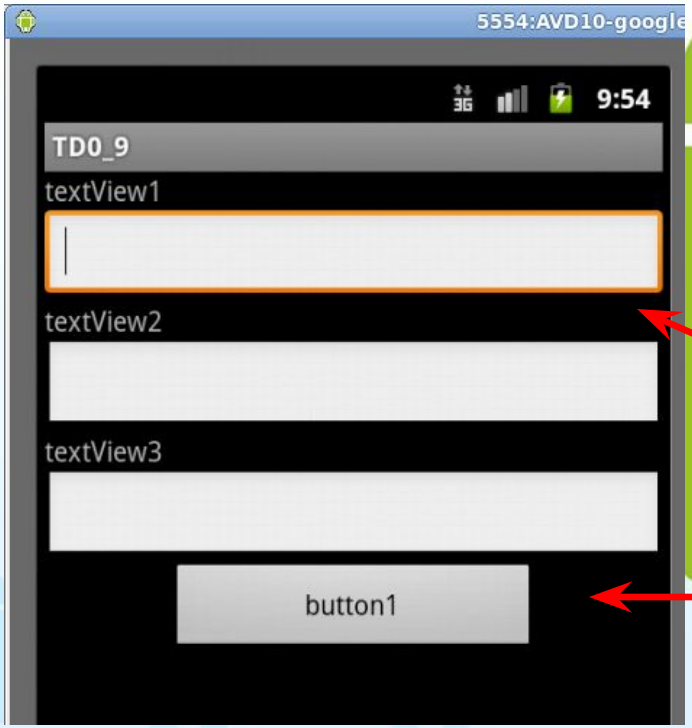
IHM : Principes de base

Découper ses interfaces avec XML/include (création de gabarit) :

- **Pour les applications qui comportent des portions d'interface commune :**
  - **Formulaire**
  - **Entête de page**
  - **Pied de page**

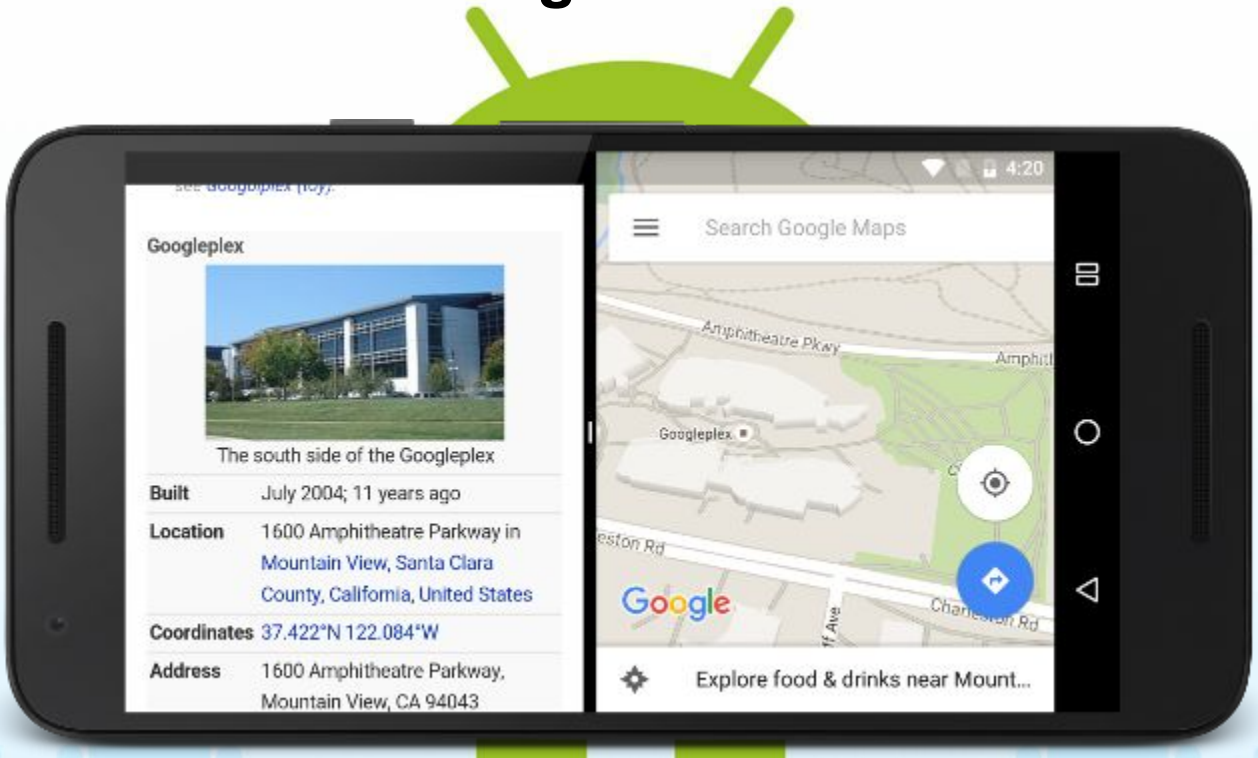
- **Evite un travail fastidieux pour la maintenance lors de la modification des IHM**
- **Demande un découpage parfois difficile entre les layouts (linearLayout, relativeLayout …)**

Découper ses interfaces avec XML/include :




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/Layout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <include
        android:id="@+id/header"
        layout="@layout/header">
    </include>

    <include
        android:id="@+id/footer"
        layout="@layout/footer">
    </include>

</LinearLayout>
```

Multi fenêtrage avec Android N



http://developer.android.com/preview/features/multi-window.html

Multi fenêtrage avec Android N

Le cycle de vie de l'activité ne change pas !

In multi-window mode, only the activity the user has most recently interacted with is active at a given time. This activity is considered *topmost*. All other activities are in the paused state, even if they are visible. However, the system gives these paused-but-visible activities higher priority than activities that are not visible. If the user interacts with one of the paused activities, that activity is resumed, and the previously topmost activity is paused

http://developer.android.com/preview/features/multi-window.html

IHM : Principes de base

TD

IHM : ANDROID_TD1 → TD02

IHM sans chargement de layout : ANDROID_TD1 → TD04

Chargement dynamique : ANDROID_TD1 → TD05

DEBUG : ANDROID_TD1 → TD06

Listener + widget : ANDROID_TD1 → TD07

Gabarit+ include : ANDROID_TD1 → TD09

Revenir au TD intent + SMS et créer l'IHM

Adapter l'affichage au type d'équipement

Par défaut, Android redimensionne votre mise en page (layout) pour s'adapter à l'écran de l'appareil. Dans la plupart des cas, cela fonctionne très bien. Dans d'autres cas, votre interface utilisateur peut ne pas être adaptée et nécessite des ajustements pour différentes tailles d'écran. Par exemple, sur un grand écran, vous pouvez ajuster la position et la taille de certains éléments afin de profiter de tout l'espace de l'écran, ou sur un écran plus petit, vous devrez peut-être ajuster la taille afin que tout puisse tenir sur l'écran.

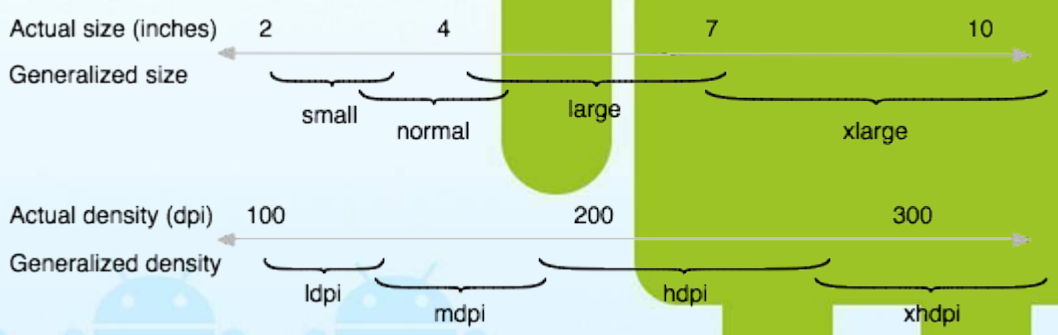

```
public class MyActivity extends Activity {
    @Override protected void onCreate(Bundle savedInstanceState)
{
        super.onCreate();

        Configuration config =
getResources().getConfiguration();
        if (config.smallestScreenWidthDp >= 600) {
            setContentView(R.layout.main_activity_tablet);
        } else {
            setContentView(R.layout.main_activity);
        }
    }
}
```

http://developer.android.com/guide/topics/resources/providing-resources.html

Adapter l'affichage au type d'équipement

Android peut organiser dynamiquement l'affichage en fonction du type d'équipement. L'affichage peut être défini en fonction des critères suivants :

- Screen size : diagionale de l'écran (normalisé en : small, normal, large, extra large).
- Screen density : Nombre de pixel par pouce .
- Resolution : Le nombre total de pixels physiques sur un écran.
- Oriantation : paysage ou portrait.



Normalisation

- xlarge screens are at least 960dp x 720dp
- large screens are at least 640dp x 480dp
- normal screens are at least 470dp x 320dp
- small screens are at least 426dp x 320dp

http://developer.android.com/guide/practices/screens_support.html
http://developer.android.com/guide/topics/resources/providing-resources.html

IHM : Principes de base

Adapter l'affichage au type d'équipement : écrans multiples && qualifier

Pour spécifier spécifiques une configuration alternative il faut utiliser un qualifier ou qualificatif :

Qualificatif (type size group) : <resources_name>-<qualifier>

avec :

<resources_name> : drawable | layout

<qualifier> : Size (small, normal, large, xlarge) | Density (ldpi, mdpi, hdpi, xhdpi, nodpi, tvdpi) | Orientation (land, port)

ATTENTION -> small, normal, large and xlarge : jusqu'à 3.2

```
res/layout/my_layout.xml            // layout for normal screen size ("default")
res/layout-small/my_layout.xml      // layout for small screen size
res/layout-large/my_layout.xml      // layout for large screen size
res/layout-xlarge/my_layout.xml     // layout for extra large screen size
res/layout-xlarge-land/my_layout.xml // layout for extra large in landscape
orientation

res/drawable-mdpi/my_icon.png       // bitmap for medium density
res/drawable-hdpi/my_icon.png       // bitmap for high density
res/drawable-xhdpi/my_icon.png      // bitmap for extra high density
```

http://developer.android.com/guide/practices/screens_support.html

IHM : Principes de base

Adapter l'affichage au type d'équipement : écrans multiples && qualifier

Qualificatif de largeur minimal (Android 3.2 et sup.), largeur / longueur indépendant de l'orientation :

```
smallestWidth sw<N>dp (taille min en largeur / longueur de l'écran)
```

Exemple pour une tablette / smartphone :

```
res/layout/main_activity.xml            # For handsets
res/layout-sw600dp/main_activity.xml    # For tablets
```

Exemple pour une configuration pour 7" et 10"

```
res/layout/main_activity.xml            # For handsets (smaller than 600dp available width)
res/layout-sw600dp/main_activity.xml    # For 7” tablets (600dp wide and bigger)
res/layout-sw720dp/main_activity.xml    # For 10” tablets (720dp wide and bigger)
```

Code pays voir : ISO_639-1 - http://fr.wikipedia.org/wiki/Liste_des_codes_ISO_639-1

http://developer.android.com/guide/practices/screens_support.html#qualifiers

IHM : Principes de base

Adapter l'affichage au type d'équipement : Manifest

- Déclarer explicitement dans le fichier manifest les tailles d'écran de votre application prend en charge au moins 600 dpi en largeur minimale, ceci permet de définir au moment du lancement de l'application si le terminal est compatible:

```
<supports-screens android:resizeable=["true"| "false"] -----> déprécié
                  android:smallScreens=["true" | "false"]
                  android:normalScreens=["true" | "false"]
                  android:largeScreens=["true" | "false"]
                  android:xlargeScreens=["true" | "false"]
                  android:anyDensity=["true" | "false"]
                  android:requiresSmallestWidthDp="integer"
                  android:compatibleWidthLimitDp="integer"
                  android:largestWidthLimitDp="integer"/>
```

```
<manifest ... >
    <supports-screens android:requiresSmallestWidthDp="600" />
    ...
</manifest>
```

ou encore : <supports-screens android:xlargeScreens="true" />

http://developer.android.com/guide/practices/screen-compat-mode.html
http://developer.android.com/guide/practices/screens_support.html#qualifiers

IHM : Principes de base

Adapter l'affichage au type d'équipement : Best-matching Resource

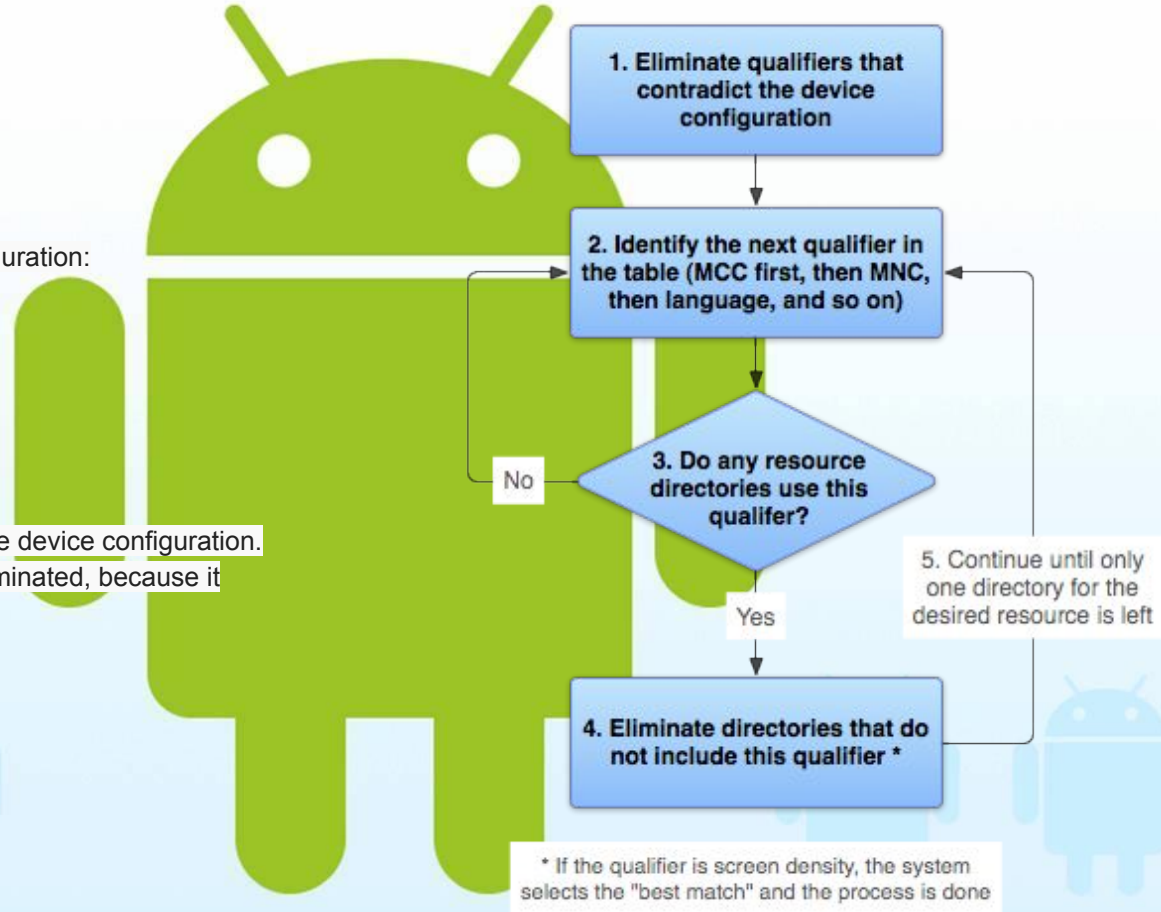
```
drawable/
drawable-en/
drawable-fr-rCA/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```

Assume the following is the device configuration:

Locale = `en-GB`
Screen orientation = `port`
Screen pixel density = `hdpi`
Touchscreen type = `notouch`
Primary text input method = `12key`

Eliminate resource files that contradict the device configuration.
The `drawable-fr-rCA/` directory is eliminated, because it contradicts the `en-GB`locale.

```
drawable/
drawable-en/
drawable-fr-rCA/
drawable-en-port/
drawable-en-notouch-12key/
drawable-port-ldpi/
drawable-port-notouch-12key/
```



http://developer.android.com/guide/topics/resources/providing-resources.html

IHM : Principes de base

TD

Projet Android SCREEN :

- réaliser les écrans pour tablettes et smartphones en anglais + allemand
- en mode portrait + paysage

IHM Java 2D- Principes

IHM : Principes de base

Mode graphique Java :

Possibilité de dessiner dans un canvas avec Java 2D. Performance relativement bonne, dépend de la plateforme (plusieurs cœurs, de la mémoire, de la carte graphique, de la qualité du programme …)

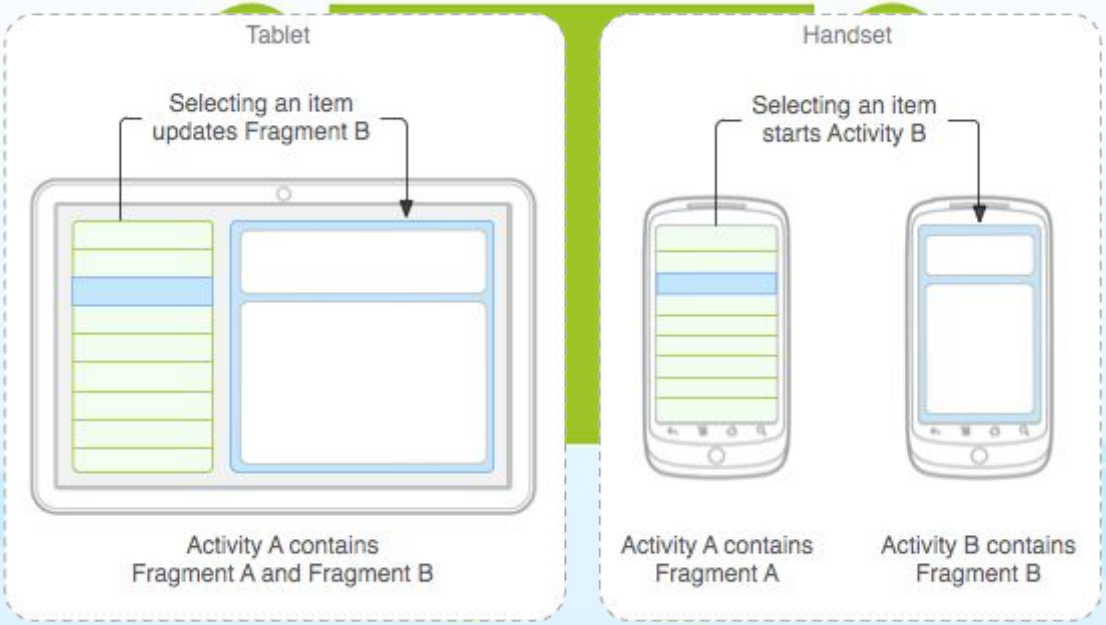
Voir TP ANDROID_2D (dessiner sur le canvas)

IHM Fragments - Principes

Interface flexible avec les fragments

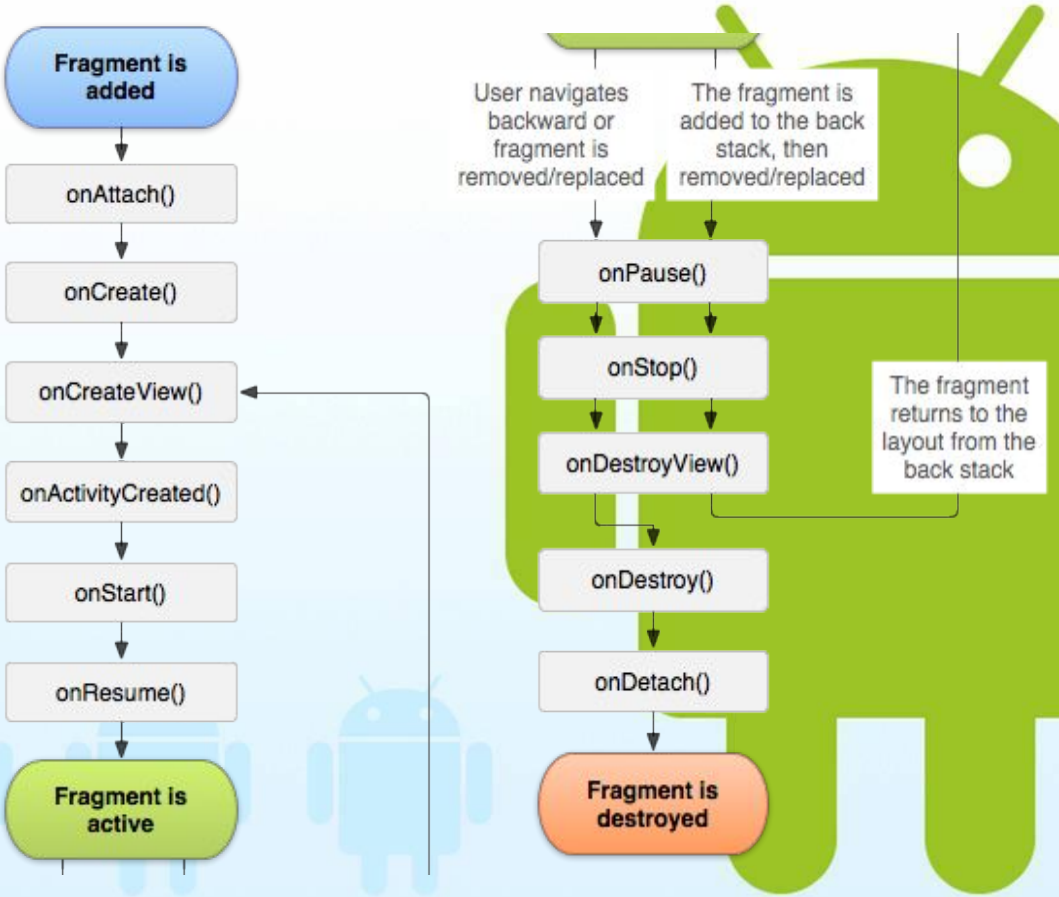
- Permet de fragmenter graphiquement une application
- Permet de changer les blocs d'affichage, plus de dynamique
- Réutilisation des codes liés au IHM

→ Demande une analyse pour le découpage



http://developer.android.com/guide/components/fragments.html

Fragment - cycle de vie



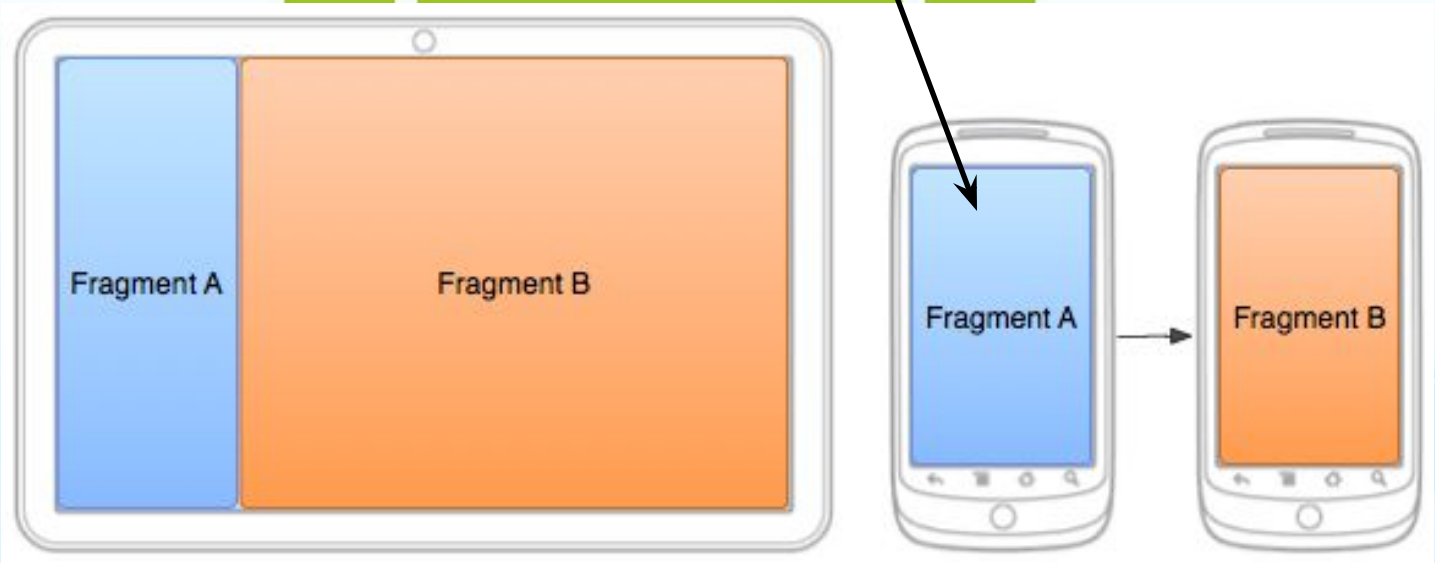
onCreate()
The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

onCreateView()
The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.

onPause()
The system calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

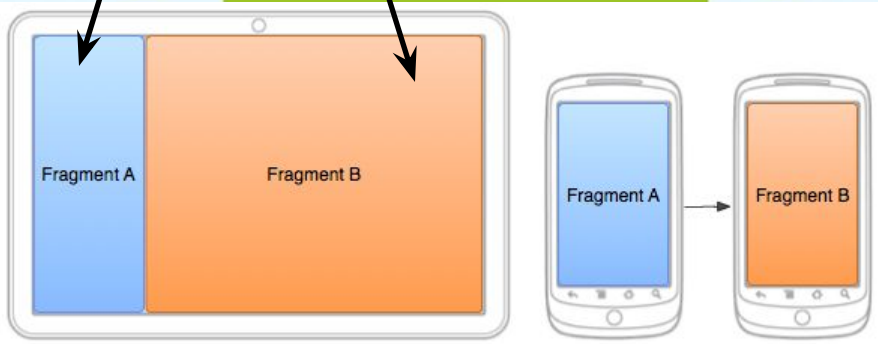
Interface flexible / fragment / mode portrait

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <fragment class="com.example.testfrag.FragmentLayout$TitlesFragment"
            android:id="@+id/titles"
            android:layout_width="match_parent" android:layout_height="match_parent" />
</FrameLayout>
```



Interface flexible / fragment / mode paysage

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

    <fragment class="com.example.testfrag.FragmentLayout$TitlesFragment"
            android:id="@+id/titles" android:layout_weight="1"
            android:layout_width="0px" android:layout_height="match_parent" />

    <FrameLayout android:id="@+id/details" android:layout_weight="1"
            android:layout_width="0px" android:layout_height="match_parent"
            android:background="?android:attr/detailsElementBackground" />

</LinearLayout>
```



Fragment - liens avec les classes / mode paysage

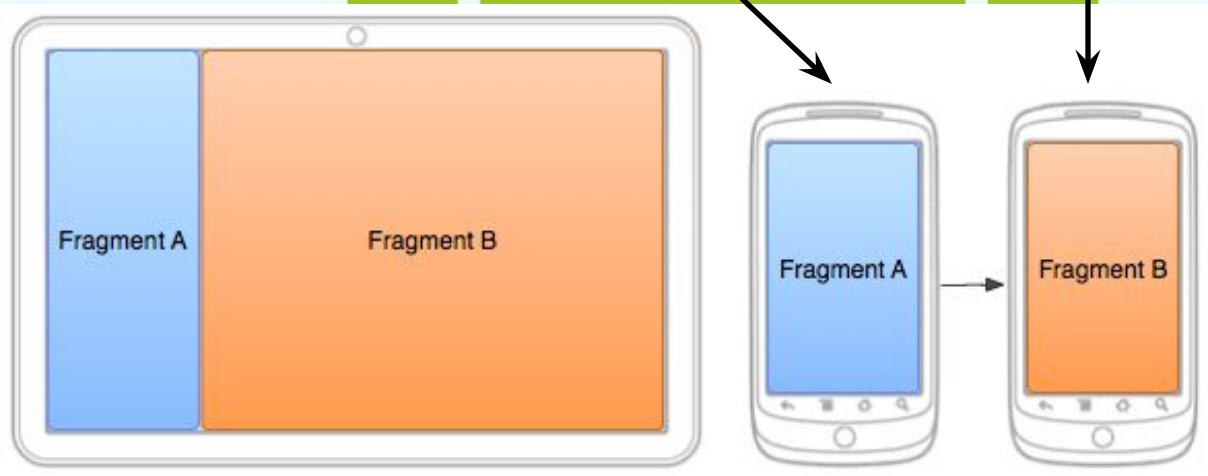
```
public static class TitlesFragment
extends ListFragment
```

```
public static class DetailsFragment
extends Fragment
```

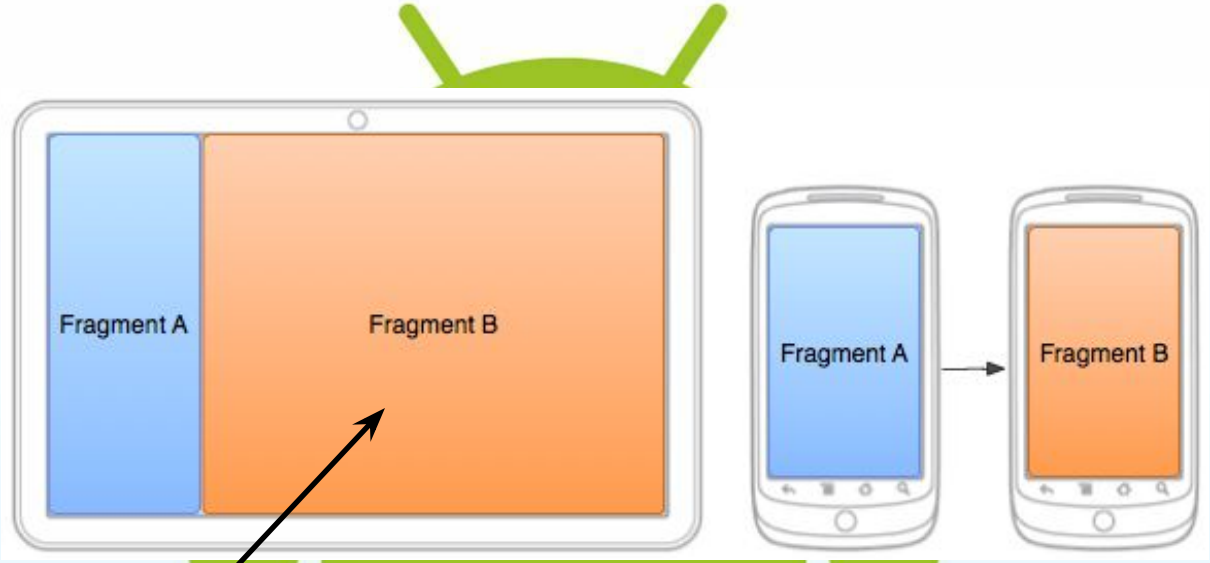
Interface fexible / fragment / mode portrait

```
public static class DetailsActivity
extends Activity
```

```
public static class TitlesFragment
extends ListFragment
```



Fragment - générique




```
<FrameLayout android:id="@+id/details" android:layout_weight="1"
    android:layout_width="0px" android:layout_height="match_parent"
    android:background="?android:attr/detailsElementBackground" />
```

TD

- Réaliser une application avec le wizard pour obtenir une application avec fragment
- Android Fragment: TD1_1 : suivre les instructions dans le code

Services - Principes

Qu’est ce qu’un Service ?

Un service est un processus qui fourni des informations, qui peut aussi réagir en fonction d’un contexte, exemple : le téléphone sonne :

- Le service téléphone réagi et peut faire vibrer le téléphone
- Un SMS arrive, le téléphone le notifie avec un « bip »
- On recherche une localisation, le service map présente une carte …

Les services sont des processus qui sont exécutés en tâche de fond et sont la plupart du temps découplés des activités (pas d’IHM).

http://developer.android.com/reference/android/app/Service.html

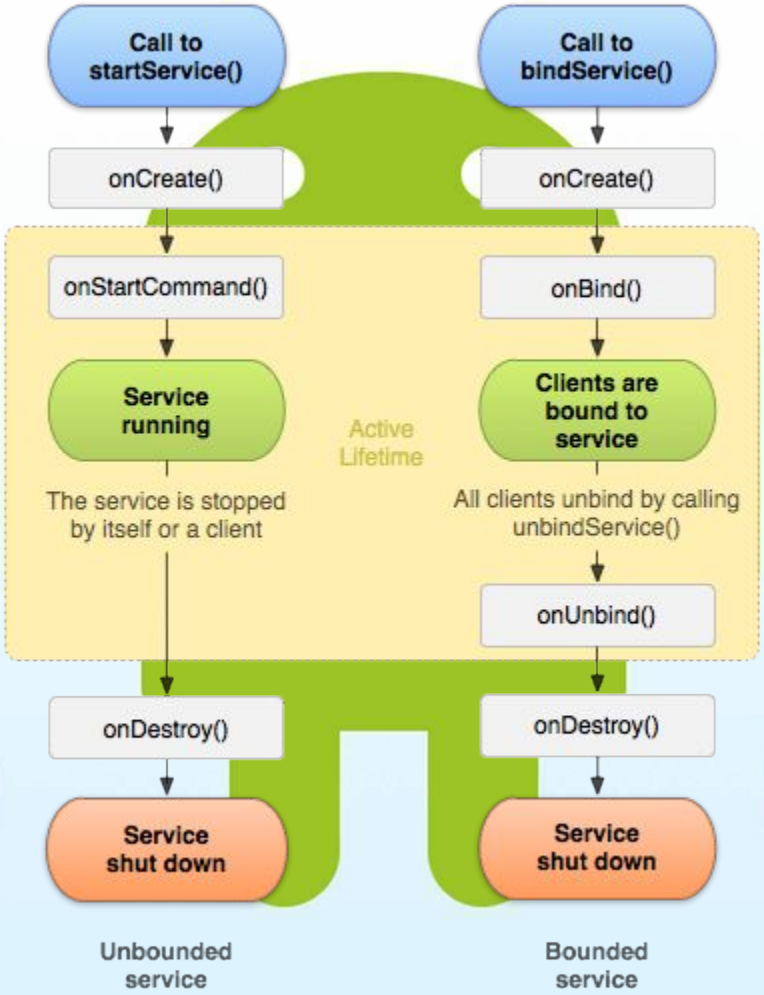
Principales caractéristiques

- Un moyen pour une application d'exécuter une opération en tâche de fond même si l'utilisateur n'interagit pas directement avec l'application. Cela correspond à un appel de Context.startService().

- Un moyen pour une application d'exposer ses fonctionnalités à d'autre application, cela correspond à l'appel de Context.bindService ()

- Un service n'est pas un processus séparé, il n'est pas exécuté dans son propre processus, il se trouve dans le même processus que l'application

Creating a local Service : lifecycle



Creating a local Service : Framework

```
public class ExampleService extends Service {
    int mStartMode;       // indicates how to behave if the service is
killed
    IBinder mBinder;      // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```

http://developer.android.com/guide/components/services.html

Creating a local Service : Framework

Service et IPC :

Les services peuvent redéfinir trois méthodes : OnCreate, OnStart, OnDestroy.

La méthode onBind est la seule méthode que vous devez obligatoirement implémenter dans un service. Cette méthode retourn un objet IBinder qui permet au mécanisme IPC de fonctionner.

Arrêt :

Si un composant démarre le service en appelant **startService** () (qui se traduit par un appel à **onStartCommand** ()), le service reste actif jusqu'à ce qu'il s'arrête lui-même avec **stopSelf** () ou qu'un autre composant l'arrête en appelant **StopService** ().

Si un composant appelle **bindService** () pour créer le service (**onStartCommand** () n'est pas appelée), le service fonctionne tant que le composant est attaché. Une fois que le service est détaché de tous les clients, le système le détruit.

Service : Foreground / Background

Foreground :

Un service de premier plan (foreground) est un service qui est considéré comme quelque chose que l'utilisateur utilise activement et n'est donc pas un candidat pour la suppression lorsque le système à peu de mémoire. Un service de premier plan doit fournir une notification à la barre d'état comme application en cours, ce qui signifie que la notification ne peut être supprimée que si le service est arrêté ou enlevé du premier plan.

Background :

Par défaut un service est lancé en background et peu passer en foreground qu'avec l'appel à startForeground().

http://developer.android.com/guide/components/services.html

Service : AndroidManifest.xml

- The service entry has to be done in the AndroidManifest.xml file along with the service intent as shown below:

Déclaration du service

Le service

```
…
<service class=".service.MyService">
<intent-filter>
        <action android:value="com.testApp.service.MY_SERVICE" />
</intent-filter>
</service>
…
```

Démarrer et arrêter un service : startService

startService :

```
Intent serviceIntent = new Intent();
serviceIntent.setAction("com.testApp.service.MY_SERVICE");
startService(serviceIntent);
```

ou

```
startService(new Intent(this. monService.class));
```

stopService :

```
stopService(new Intent(this, monService.getClass());
```

Bind sur un service local

```
public class BindingActivity extends Activity {
    LocalService mService;
    boolean mBound = false;

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

    @Override
    protected void onStop() {
        super.onStop();
        // Unbind from the service
        if (mBound) {
            unbindService(mConnection);
            mBound = false;
        }
    }

    /** Called when a button is clicked (the button in the layout file attaches to
     * this method with the android:onClick attribute) */
    public void onButtonClick(View v) {
        if (mBound) {
            // Call a method from the LocalService.
            // However, if this call were something that might hang, then this request should
            // occur in a separate thread to avoid slowing down the activity performance.
            int num = mService.getRandomNumber();
            Toast.makeText(this, "number: " + num, Toast.LENGTH_SHORT).show();
        }
    }
```

voir page suivante

```
    /** Defines callbacks for service binding, passed to bindService() */
    private ServiceConnection mConnection = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName className,
                IBinder service) {
            // We've bound to LocalService, cast the IBinder and get LocalService instance
            LocalBinder binder = (LocalBinder) service;
            mService = binder.getService();
            mBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            mBound = false;
        }
    };
}
```

- Pour pouvoir cummuniquer avec un service local, il est également possible d'utiliser bind(). Le bind permet d'accéder aux primitives du service local
- Dans le cas d'un service local, c'est à dire exécuté dans le même procesus que l'activité, il n'est pas nécessaire d'utiliser l'AIDL.

Bind sur un service local - ServiceConnection

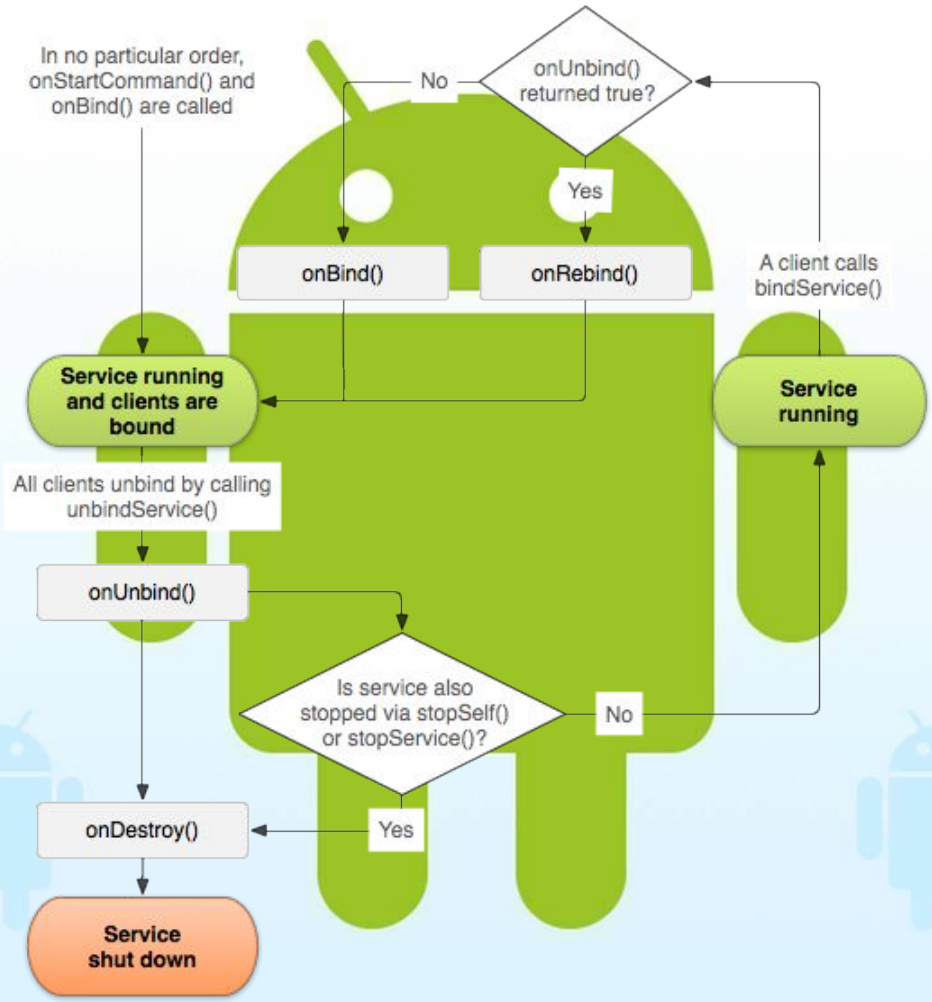
```
private ServiceConnection mConnection = new ServiceConnection() {
    // Called when the connection with the service is established
    public void onServiceConnected(ComponentName className, IBinder service) {
        // Because we have bound to an explicit
        // service that is running in our own process, we can
        // cast its IBinder to a concrete class and directly access it.
        LocalBinder binder = (LocalBinder) service;
        mService = binder.getService();
        mBound = true;
    }

    // Called when the connection with the service disconnects unexpectedly
    public void onServiceDisconnected(ComponentName className) {
        Log.e(TAG, "onServiceDisconnected");
        mBound = false;
    }
};
```

http://developer.android.com/guide/components/bound-services.html

Bind sur un service local (bound service) - Service

```
public class LocalService extends Service {
    // Binder given to clients
    private final IBinder mBinder = new LocalBinder();
    // Random number generator
    private final Random mGenerator = new Random();

    /**
     * Class used for the client Binder.  Because we know this service always
     * runs in the same process as its clients, we don't need to deal with IPC.
     */
    public class LocalBinder extends Binder {
        LocalService getService() {
            // Return this instance of LocalService so clients can call public methods
            return LocalService.this;
        }
    }

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

    /** method for clients */
    public int getRandomNumber() {
      return mGenerator.nextInt(100);
    }
}
```

Binding un service local : cycle de vie



Exemple avec un service téléphonique local

```
package exemple.android.phoneService;

import …

public class ServiceCall extends Service implements ServiceConnection
{
	private static final String TAG = "AdditionService";
	private ServiceCall serviceBinder;
	private final IBinder binder = new MyBinder();

	@Override
	public void onServiceConnected(ComponentName name, IBinder service)
	{
		serviceBinder = ((ServiceCall.MyBinder)service).getService();
		Log.i("ServiceCall", "Connexion du service");
	}

	@Override
	public void onServiceDisconnected(ComponentName name)
	{
		serviceBinder = null;
		Log.i("ServiceCall", "Deconnexion du service");
	}
}
```

On veut créer un service qui donne des informations à chaque appel téléphonique entrant

Creating a local Service : le service

```
@Override
public IBinder onBind(Intent intent)
{
        Log.d(TAG, "##########>>>>  Checked for permission: \nresult: " +
        checkCallingPermission("exemple.android.phoneService.permission.MY_FIRST_SERVICE"));
        return binder;
}

@Override
public void onStart(Intent intent, int startId)
{
        String txtToast = "** Service d'appels démarré **: "+startId;
        Toast toast = Toast.makeText(getApplicationContext(), txtToast, Toast.LENGTH_SHORT);
        toast.show();

        // Mise en place du listener pour les appels
        TelephonyManager tManager = (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
        ListenerPhoneState listenerAppels = new ListenerPhoneState();
        tManager.listen(listenerAppels, PhoneStateListener.LISTEN_CALL_STATE);
}

        public class MyBinder extends Binder {
        ServiceCall getService()
        {
                return ServiceCall.this;
        }
}
}
```

Création du Service local : le listener

```
class ListenerPhoneState extends PhoneStateListener
{
     private static final String TAG = "PhoneService";

     public void onCallStateChanged(int state, String incomingNumber)
     {
          switch (state)
          {
               case TelephonyManager.CALL_STATE_RINGING:
               {
                // Actions à effectuer à la reception d'un appel
                Log.d(TAG, "RINGING\n");
                    break;
               }
          }
     }
}
```

Création de l'activity

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

         setContentView(R.layout.main);

         ServiceCall mConnection = new ServiceCall();

         Intent bindIntent = new Intent(CallNotify.this, ServiceCall.class);

         bindService(bindIntent, mConnection, Context.BIND_AUTO_CREATE);

         startService(bindIntent);

         unbindService(mConnection);
     }
}
```

Ici on se connecte au service

Le fichier manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="example.android"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
          <activity android:label="@string/app_name"
      android:name="exemple.android.phoneService.CallNotify">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"></action>
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
      <service android:name="exemple.android.phoneService.ServiceCall"
android:permission="android.permission.READ_PHONE_STATE"></service>
</application>
<uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
   <uses-sdk android:minSdkVersion="9" />
</manifest>
```

Service distant (remote service)

Pour se connecter à un service distant, c'est à dire un service qui n'est pas dans le même processus, il existe deux méthodes :

1) Utilisation de l'AIDL qui permet de définir des interfaces que l'on publie dans le service. Toutes les autres applications peuvent alors se connecter et accéder aux méthodes distantes.

http://developer.android.com/guide/components/aidl.html

2) Utiliser l'API Messenger qui permet de transmettre des informations entre les applications et le service.

http://developer.android.com/guide/components/bound-services.html#Messenger

TD

Projet Android Service2
- BindLocal
- Service

Android Telephony Service

Services / boot - Principes

Start at boot time

Now we are going to learn how to start a service at boot time, i.e. to start our service when device starts up.

First we have to create a BroadcastReceiver which will be started when boot completes as

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;


  public class MyStartupIntentReceiver extends BroadcastReceiver
  {
    @Override
    public void onReceive(Context context, Intent intent)
   {
   }
}
```

First we have to create a BroadcastReceiver which will be started when boot completes as

Start at boot time

onRecieve() will be first called when the BroadcastReceiver MyStartupIntentReceiver starts, Next make an entry of this receiver in AndroidManifest.xml as :

```
<receiver android:name="MyStartupIntentReceiver">
<intent-filter>
<action
android:name="android.intent.action.BOOT_COMPLETED" />
<category android:name="android.intent.category.HOME" />
</intent-filter>
</receiver>
```

Here in intent filter we have declared the action as android.intent.action.BOOT_COMPLETED , so that this receiver and intent with action android.intent.action.BOOT_COMPLETED

Now this receiver will be intimated when boot completes

Start at boot time : Create a service as

```
import …;

public class MyService extends Service {

	@Override
	public IBinder onBind(Intent intent) {
		return null;
	}
	@Override
	public void onCreate() {
		super.onCreate();
		Toast.makeText(this, "Service Created", Toast.LENGTH_LONG).show();
	}

	@Override
	public void onDestroy() {
		super.onDestroy();
		Toast.makeText(this, "Service Destroyed", Toast.LENGTH_LONG).show();
	}
	@Override
	public void onStart(Intent intent, int startId) {
		super.onStart(intent, startId);
		Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();

	}
}
```

Start at boot time :

```
<service android:name="MyService">
<intent-filter>
<action
android:name="com.wissen.startatboot.MyService" />
</intent-filter>
</service>
```

Make an entry of this service in AndroidManifest.xml as

```
public void onReceive(Context context, Intent intent)
{
Intent serviceIntent = new Intent();
serviceIntent.setAction("com.wissen.startatboot.MyService");
context.startService(serviceIntent);

}
```

Now start this service in the BroadcastReceiver MyStartupIntentReceiver's onReceive method as

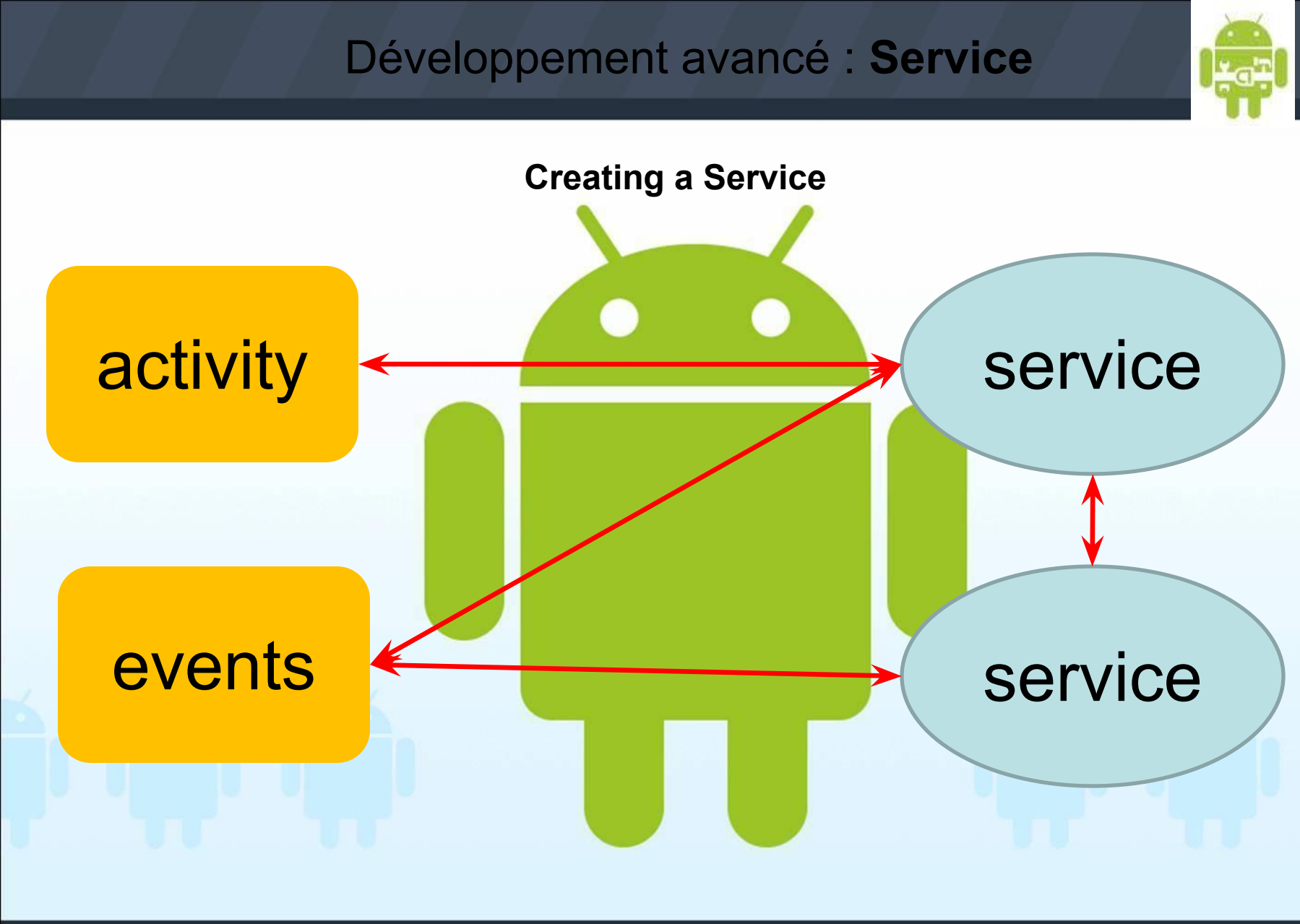
TD

Démarrage d'un service au boot: Projet Android Service2 → boot avec BroadcastReceiver

A warning about long-running services

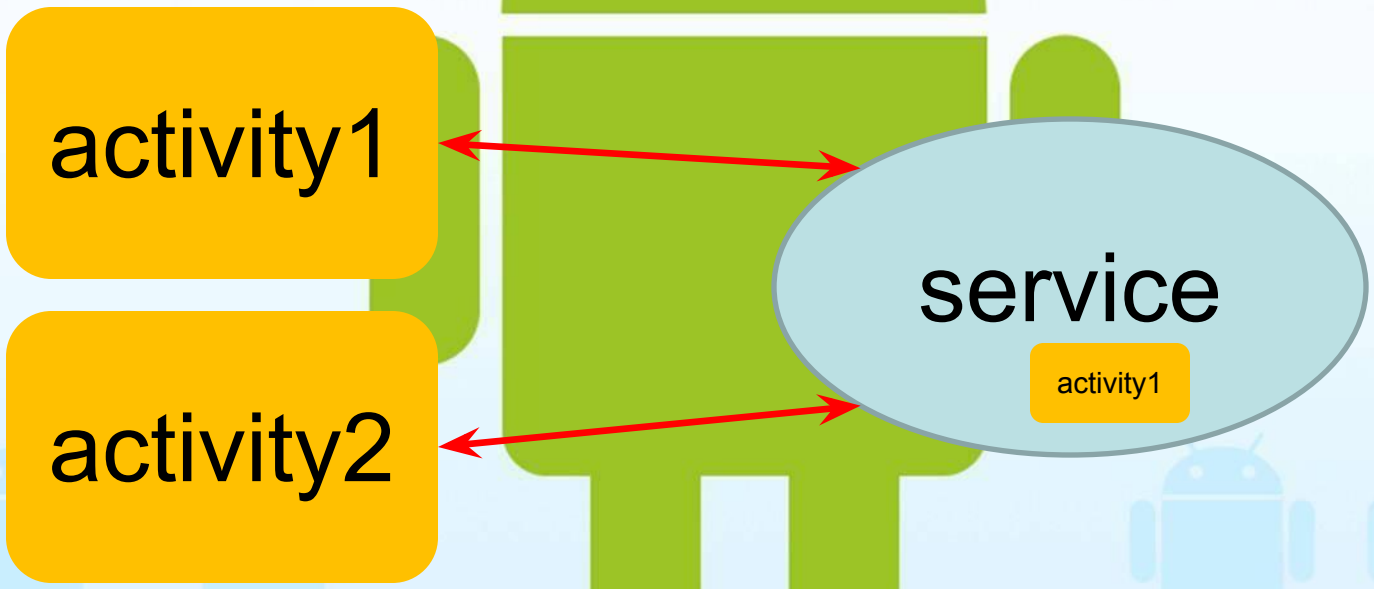
- Our service is designed to have a minimal footprint (when the polling is tuned)
- In general long-running services are strongly discouraged
- If your use case doesn't require it, you should make sure to stop any services you have started when your application exits
- Services are a bit of a paradox in this sense; they are for background tasks, but background is not intended to mean forever

http://groups.google.com/group/android-developers/browse_thread/thread/fa2848e31636af70


Creating a Service
activity
events
service
service

Creating a Service

Si plusieurs activités accède au même service, une seule à la fois sera exécuté (FIFO)



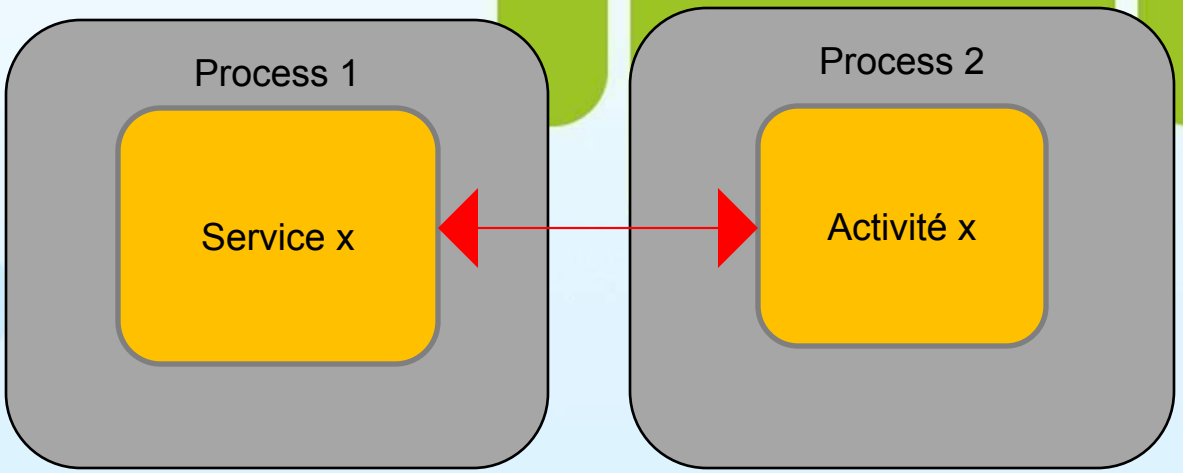
Processus et intercommunications

Communication with Remote Service

- The services that we defined until now run in the application processes, you can define service that can run in their own process.

- For two processes to communicate with each other they need to marshal the object to sent to other process.

- Android provide an AIDL tool (Android Interface definition Language) to handle all marshalling and communication part.



http://developer.android.com/guide/developing/tools/aidl.html

Communication with Remote Service with AIDL (IPC)

- To talk, they need to decompose their objects into primitives that the operating system can understand, and "marshall" the object across that boundary for you.
- If you have code in one process (for example, in an Activity) that needs to call methods on an object in another process (for example, a Service), you would use AIDL to generate code to marshall the parameters.

Implementing IPC Using AIDL

Create your .aidl file. This file defines an interface (YourInterface.aidl) that defines the methods and fields available to a client.

→

Implement your interface methods The AIDL compiler creates an interface in the Java programming language from your AIDL interface

→

Expose your interface to clients - If you're writing a service, you should extend Service and override Service.onBind(Intent)

Communication with Remote Service with AIDL (IPC)

An example .aidl remote interface definition language file

```
package com.msi.manning.binder;

interface ISimpleMathService {
    int add(int a, int b);
    int subtract(int a, int b);
    String echo(in String input);
}
```

1. Define the package
2. Declare the interface name
3. Describe a method

1.The package B, import statements (of which we have none here)

2.Interface C constructs in AIDL are straightforward—they are analogous to regular Java

3) When you define methods, you must specify a directional tag for all nonprimitive types with each parameter (in, out, or inout). Primitives are allowed only as in and are therefore treated as in by default (and thus don't need the tag). It's better to go in only one direction where you can, for performance reasons.

Communication with Remote Service with AIDL (IPC)

When using AIDL you also have to be aware that only certain types are allowed; these types are shown in table

Type	Description	Import Required
Java primitives	`boolean`, `byte`, `short`, `int`, `float`, `double`, `long`, `char`.	No
`String`	`java.lang.String`.	No
`CharSequence`	`java.lang.CharSequence`.	No
`List`	Can be generic; all types used in collection must be one of IDL allowed. Ultimately implemented as an `ArrayList`.	No
`Map`	Can be generic, all types used in collection must be one of IDL allowed. Ultimately implemented as a `HashMap`.	No
Other AIDL interfaces	Any other AIDL-generated interface type.	Yes
Parcelable objects	Objects that implement the Android Parcelable interface (more about this in section 4.4.3).	Yes

Exemple objet Parcelable:

```
import android.os.Parcel;
import android.os.Parcelable;

public final class Rect implements Parcelable {
    public int left;
    public int top;
    public int right;
    public int bottom;

    public static final Parcelable.Creator<Rect>
CREATOR = new
Parcelable.Creator<Rect>() {
        public Rect createFromParcel(Parcel in) {
            return new Rect(in);
        }

        public Rect[] newArray(int size) {
            return new Rect[size];
        }
    };

    public Rect() {
    }

    private Rect(Parcel in) {
        readFromParcel(in);
    }

    public void writeToParcel(Parcel out) {
        out.writeInt(left);
        out.writeInt(top);
        out.writeInt(right);
        out.writeInt(bottom);
    }

    public void readFromParcel(Parcel in) {
        left = in.readInt();
        top = in.readInt();
        right = in.readInt();
        bottom = in.readInt();
    }
}
```

http://developer.android.com/guide/components/aidl.html

Communication with Remote Service with AIDL (IPC)

Exposing a remote interface

The glue in all of the moving parts of AIDL that we have discussed up to now is the point where a remote interface is exposed—via a Service

```
public class SimpleMathService extends Service {

   private final ISimpleMathService.Stub binder =
    new ISimpleMathService.Stub() {
      public int add(int a, int b) {
         return a + b;
      }
      public int subtract(int a, int b) {
         return a - b;
      }
      public String echo (String input) {
         return "echo " + input;
      }
   };

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

1. Implement the remote interface
2. Return an IBinder representing the remotable object

Communication with Remote Service with AIDL (IPC)

Exposing a remote interface

The glue in all of the moving parts of AIDL that we have discussed up to now is the point where a remote interface is exposed—via a Service

```
public class SimpleMathService extends Service {

    private final ISimpleMathService.Stub binder =
      new ISimpleMathService.Stub() {
        public int add(int a, int b) {
            return a + b;
        }
        public int subtract(int a, int b) {
            return a - b;
        }
        public String echo (String input) {
            return "echo " + input;
        }
    };

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

1. Implement the remote interface
2. Return an IBinder representing the remotable object

A concrete instance of the generated AIDL Java interface is required to return an IBinder to any caller than binds to a Service

Communication with Remote Service with AIDL (IPC)

Binding to a Service

- Now that we have seen where a caller can hook into a Service and get a reference to a remotable object, we need to walk through finishing that connection by binding to a Service from an Activity

- When an Activity class binds to a Service, which is done using the Context. bindService(Intent i, ServiceConnection connection, int flags) method, the ServiceConnection object that is passed in is used to send several callbacks, from the Service back to the Activity.

Communication with Remote Service with AIDL (IPC)

Binding to a Service

```
public class ActivityExample extends Activity {

   private ISimpleMathService service;
   private boolean bound;

   . . . View element declarations omitted for brevity

   private ServiceConnection connection = new ServiceConnection() {
      public void onServiceConnected(ComponentName className,
        IBinder iservice) {
          service = ISimpleMathService.Stub.asInterface(iservice);
          Toast.makeText(ActivityExample.this,
             "connected to Service", Toast.LENGTH_SHORT).show();
          bound = true;
        }
      public void onServiceDisconnected(ComponentName className) {
         service = null;
         Toast.makeText(ActivityExample.this,
            "disconnected from Service", Toast.LENGTH_SHORT).show();
         bound = false;
      }
   };
```

1. Define remote interface type variable
2. Define bound state boolean
3. Include ServiceConnection implementation
4. React to onServiceConnected callback
5. Establish remote interface type
6. React to onServiceDisconnected callback

Communication with Remote Service with AIDL (IPC)

Binding to a Service

```
@Override
public void onCreate(Bundle icicle) {
   . . . View element inflation omitted for brevity

    this.addButton.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
         try {
            int result = service.add(
              Integer.parseInt(inputa.getText().toString()),
              Integer.parseInt(inputb.getText().toString()));
            output.setText(String.valueOf(result));
         } catch (DeadObjectException e) {
            Log.e("ActivityExample", "error", e);
         } catch (RemoteException e) {
            Log.e("ActivityExample", "error", e);
         }
      }
   });

   . . . subtractButton, similar to addButton, omitted for brevity
}

@Override
public void onStart() {
   super.onStart();
```

Use remote object for operations ❼

Communication with Remote Service with AIDL (IPC)

Binding to a Service

```
        if (!bound) {
            this.bindService(
                    new Intent(ActivityExample.this,
                        SimpleMathService.class),
                    connection,
                    Context.BIND_AUTO_CREATE);      ⟵ ❽ Perform binding
        }
    }

    @Override
    public void onPause() {
        super.onPause();
        if (bound) {
            bound = false;
            this.unbindService(connection);         ⟵ ❾ Perform unbinding
        }
    }
}
```

Communication with Remote Service with AIDL (IPC)

Binding to a Service : manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.msi.manning.binder">
   <application android:icon="@drawable/icon">
      <activity android:name=".ActivityExample" android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <service android:name=".SimpleMathService" />
   </application>
</manifest>
```

Communication with Remote Service with AIDL (IPC)

Binding to a Service : Starting versus binding

```
Starting—Context.startService(Intent service, Bundle b)
Binding—Context.bindService(Intent service, ServiceConnection c, int flag)
```

- Starting a Service tells the platform to launch it in the background and keep it running, without any particular connection to any other Activity or application. We used the service in this manner to run in the background.

- Binding to a Service, as we did with our sample SimpleMathService, is how you get a handle to a remote object and call methods defined there from an Activity. As we have discussed, because every Android application is running in its own process, using a bound Service (which returns an IBinder through ServiceConnection) is how you pass data between processes

TD

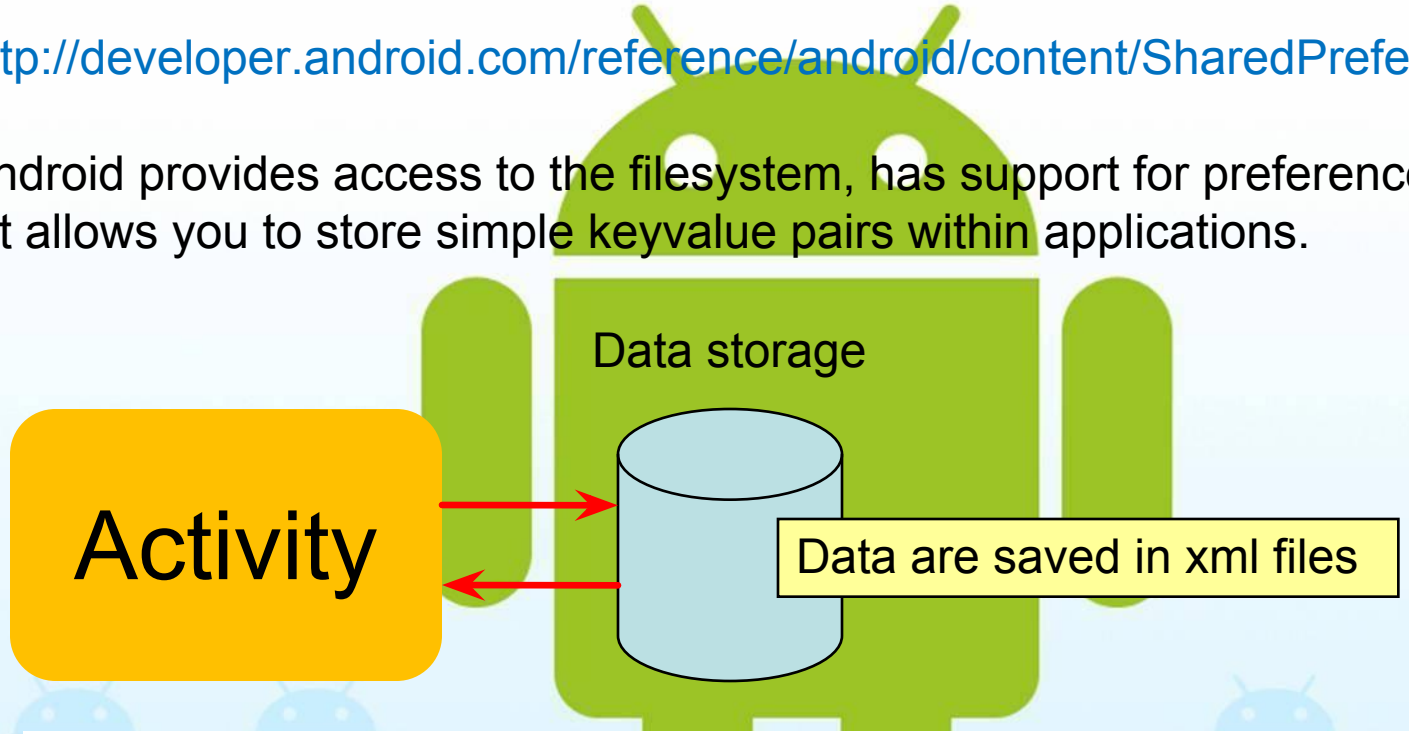
ANDROID_TD2_AIDL : suivre les instructions

Les données persistantes

Storing and retrieving data with SharedPreferences

- http://developer.android.com/reference/android/content/SharedPreferences.html

- Android provides access to the filesystem, has support for preferences system that allows you to store simple keyvalue pairs within applications.




```
private SharedPreferences prefsPrivate;
private SharedPreferences prefsWorldRead;
private SharedPreferences prefsWorldWrite;
private SharedPreferences prefsWorldReadWrite;
```

1 **Declare SharedPreferences variables**

Storing data with SharedPreferences

```
...
prefsPrivate = getSharedPreferences("mon fichier", Context.MODE_PRIVATE);

Editor prefsPrivateEditor = prefsPrivate.edit();

prefsPrivateEditor.putString("ma clé", "mon texte");

prefsPrivateEditor.commit();
...
```

Retrieving data with SharedPreferences

```
private SharedPreferences prefsPrivate;

this.prefsPrivate = getSharedPreferences("ma clé",
Context.MODE_PRIVATE);
```

Directories with the other x permission

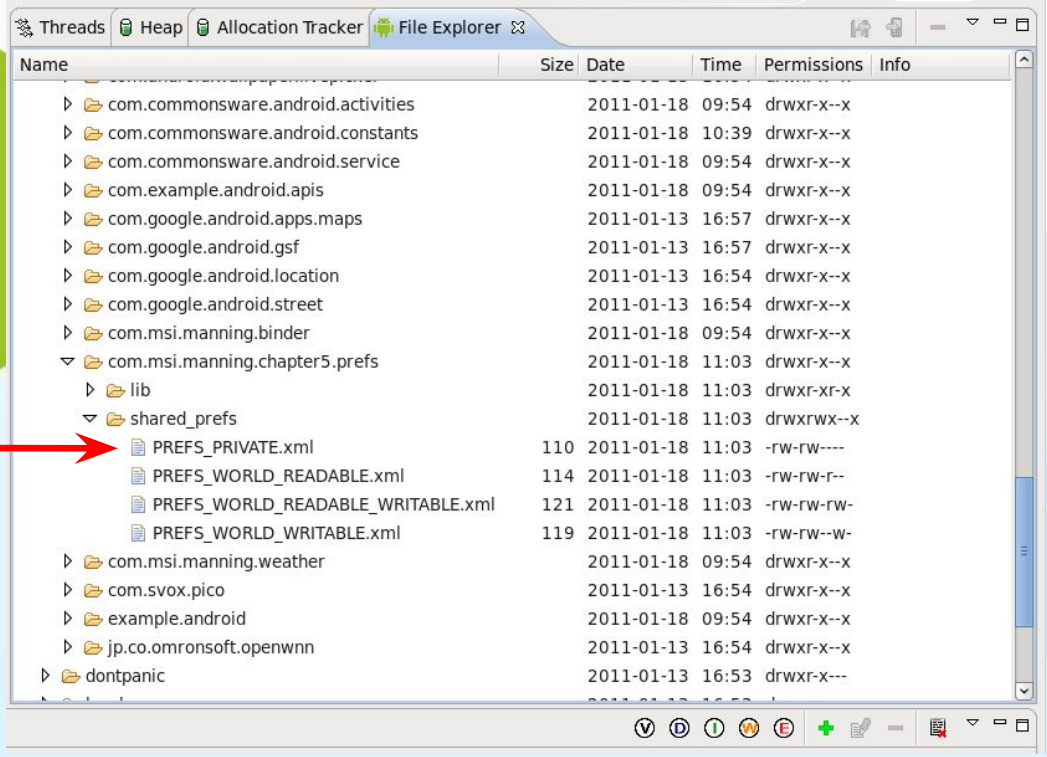
Directory permissions can be confusing. The important thing to remember with regard to Android, though, is that each package directory is created with the other x permission. This means anyone can search and list the files in the directory. This, in turn, means that Android packages have directory-level access to one another's files—from there the file-level access determines file permissions.

Retrieving data with SharedPreferences : ***Preference access permissions***

The supported mode constants are as follows:

- Context.MODE_PRIVATE *(value 0)*
- Context.MODE_WORLD_READABLE *(value 1)*
- Context.MODE_WORLD_WRITEABLE *(value 2)*

SharedPreferences XML files are placed in the /data/data/PACKAGE_NAME/shared_prefs path on the filesystem.



Développement avancé : Les données

TD

TD2 → SAVE DATA: suivre les instructions dans le code et visualiser les fichiers

Using the filesystem from an **Activity : Using the Internal Storage**

- As you have seen, Android has a filesystem that is based on Linux and supports mode based permissions

```
this.createButton.setOnClickListener(new OnClickListener() {
   public void onClick(final View v) {
      FileOutputStream fos = null;
      try {
         fos = openFileOutput("filename.txt",
          Context.MODE_PRIVATE);
         fos.write(createInput.getText().toString().getBytes());
      } catch (FileNotFoundException e) {
         Log.e("CreateFile", e.getLocalizedMessage());
      } catch (IOException e)
{
         Log.e("CreateFile", e.getLocalizedMessage());
      } finally {
         if (fos != null) {
            try {
               fos.flush();
               fos.close();
            } catch (IOException e) {
               // swallow
```

1 Use openFileOutput

2 Write data to stream

3 Flush and close stream

Using the filesystem from an **Activity : : Using the Internal Storage**

Similarly to openFileOutput, the Context also has a convenience openFileInput method. This method can be used to access a file on the filesystem and read it in, as shown in listing :

```
public class ReadFile extends Activity {

   private TextView readOutput;
   private Button gotoReadResource;

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      this.setContentView(R.layout.read_file);

      this.readOutput =
        (TextView) this.findViewById(R.id.read_output);

      FileInputStream fis = null;
      try {
         fis = this.openFileInput("filename.txt");
         byte[] reader = new byte[fis.available()];
         while (fis.read(reader) != -1) {}
         this.readOutput.setText(new String(reader));
      } catch (IOException e) {
         Log.e("ReadFile", e.getMessage(), e);
      } finally {
         if (fis != null) {
            try {
               fis.close();
            } catch (IOException e) {
```

1. Use openFileInput for stream
2. Read data from stream
3. Clean up when finished

Using the filesystem

Similarly to openFileOutput, the Context also has a convenience openFileInput method. This method can be used to access a file on the filesystem and read it in, as shown in listing :

```
public class ReadFile extends Activity {

   private TextView readOutput;
   private Button gotoReadResource;

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      this.setContentView(R.layout.read_file);

      this.readOutput =
        (TextView) this.findViewById(R.id.read_output);

      FileInputStream fis = null;
      try {
         fis = this.openFileInput("filename.txt");
         byte[] reader = new byte[fis.available()];
         while (fis.read(reader) != -1) {}
         this.readOutput.setText(new String(reader));
      } catch (IOException e) {
         Log.e("ReadFile", e.getMessage(), e);
      } finally {
         if (fis != null) {
            try {
               fis.close();
            } catch (IOException e) {
```

1. Use openFileInput for stream
2. Read data from stream
3. Clean up when finished

Using the filesystem

Running a bundle of apps with the same user ID

Though it is the exception rather than rule, there are times when setting the user ID your application runs as can be extremely useful (most of the time it's fine to allow the platform to select a unique ID for you). For instance, if you have multiple applications that need to store data among one another, but you also want that data to not be accessible outside that group of applications, you may want to set the permissions to private and share the UID to allow access. You can allow a shared UID by using the sharedUserId attribute in your manifest: `android:sharedUserId="YourFancyID"`.

Using the filesystem : raw file

If you want to include raw files with your application of any form, you can do so using the res/raw resources location

```
public class ReadRawResourceFile extends Activity {

   private TextView readOutput;
   private Button gotoReadXMLResource;

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      this.setContentView(R.layout.read_rawresource_file);

      this.readOutput =
        (TextView) this.findViewById(R.id.readrawres_output);

      Resources resources = this.getResources();
      InputStream is = null;
      try {
         is = resources.openRawResource(R.raw.people);
         byte[] reader = new byte[is.available()];
         while (is.read(reader) != -1) {}
         this.readOutput.setText(new String(reader));
      } catch (IOException e) {
         Log.e("ReadRawResourceFile", e.getMessage(), e);
      } finally {
         if (is != null) {
            try {
               is.close();
            } catch (IOException e) {
               // swallow
            }
         }
      }

      . . . goto next Activity via startActivity omitted for brevity

   }
}
```

❶ Hold raw resource with InputStream

❷ Use getResources().openRawResource()

Using the filesystem : XML file

- XML data are stored in res/xml

- For example : an XML file that defines multiple <person> elements and uses attributes for firstname and lastname— people.xml.

```
<people>
<person firstname="John" lastname="Ford" />
<person firstname="Alfred" lastname="Hitchcock" />
<person firstname="Stanley" lastname="Kubrick" />
<person firstname="Wes" lastname="Anderson" />
</people>
```

Once a file is in the res/xml path, it will be automatically picked up by the platform (if you are using Eclipse) and compiled into a resource asset. This asset can then be accessed in code by parsing the binary XML format Android supports

Using the filesystem : XML file

```
public class ReadXMLResourceFile extends Activity {

   private TextView readOutput;

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      this.setContentView(R.layout.read_xmlresource_file);

      this.readOutput = (TextView)
       this.findViewById(R.id.readxmlres_output);

      XmlPullParser parser = this.getResources().getXml(R.xml.people);
      StringBuffer sb = new StringBuffer();

      try {
         while (parser.next() != XmlPullParser.END_DOCUMENT) {
```

Parse XML with XMLPullParser ❶

❷ Walking the XML tree

To process a binary XML resource you use an XmlPullParser (1). This class can walk though the XML tree SAX style. The parser provides an event type represented by an int for each element it encounters, such as DOCDECL, COMMENT, START_DOCUMENT, START_TAG, END_TAG, END_DOCUMENT, and so on. Using the next() method you can retrieve the current event type value and compare it to event constants in the class (2)

Using the filesystem : XML file

```
String name = parser.getName();
String first = null;
String last = null;
if ((name != null) && name.equals("person")) {
   int size = parser.getAttributeCount();
   for (int i = 0; i < size; i++) {
      String attrName =
        parser.getAttributeName(i);
      String attrValue =
        parser.getAttributeValue(i);
      if ((attrName != null)
        && attrName.equals("firstname")) {
          first = attrValue;
      } else if ((attrName != null)
        && attrName.equals("lastname")) {
          last = attrValue;
      }
   }
   if ((first != null) && (last != null)) {
      sb.append(last + ", " + first + "\n");
   }
}
}
this.readOutput.setText(sb.toString());
} catch (Exception e) {
  Log.e("ReadXMLResourceFile", e.getMessage(), e);
}
```

3 Get attributeCount for element

4 Get attribute name and value

Each element encountered has a name, a text value, and an optional set of attributes. You can walk through the document as we are here by getting the attributeCount (3) for each item and grabbing the name and value (4).

External storage via an SD card

One of the advantages the Android platform provides over some other similar device competitors is that it offers access to an available Secure Digital (SD) flash memory card.

Flash memory is a non-volatile computer storage chip that can be electrically erased and reprogrammed.
Flash memory is non-volatile, meaning no power is needed to maintain the information stored in the chip

External storage via an SD card

SD cards and the emulator

In order to work with an SD card image in the Android Emulator, you will first need to use the mksdcard tool provided to set up your SD image file (you will find this executable in the tools directory of the SDK). After you have created the file, you will need to start the emulator with the `-sdcard <path_to_file>` option in order to have the SD image mounted.

Using the SD card makes a lot of sense if you are dealing with large files or when you don't necessarily need to have permanent secure access to certain files. Obviously, if you are working with image data, audio files, or the like, you will want to store these on the SD card

On the other hand, for application-specialized data that you do need to be permanent and for which you are concerned about secure access, you should use the internal filesystem (or an internal database).

External storage via an SD card

The SD card is impermanent (the user can remove it), and SD card support on most devices, including Android-powered devices, supports the FAT (File Allocation Table) filesystem.

That's important to remember because it will help you keep in mind that the SD card doesn't have the access modes and permissions that come from the Linux filesystem.

External storage via an SD card

```
public class ReadWriteSDCardFile extends Activity {

   private TextView readOutput;

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
      this.setContentView(R.layout.read_write_sdcard_file);

      this.readOutput = (TextView)
        this.findViewById(R.id.readwritesd_output);

      String fileName = "testfile-"
       + System.currentTimeMillis() + ".txt";
```

❶ Establish filename

```
File sdDir = new File("/sdcard/");
if (sdDir.exists() && sdDir.canWrite()) {
   File uadDir = new File(sdDir.getAbsolutePath()
    + "/unlocking_android");
   uadDir.mkdir();
   if (uadDir.exists() && uadDir.canWrite()) {
      File file = new File(uadDir.getAbsolutePath()
       + "/" + fileName);
      try {
         file.createNewFile();
      } catch (IOException e) {
         // log and or handle
      }

      if (file.exists() && file.canWrite()) {
         FileOutputStream fos = null;
         try {
            fos = new FileOutputStream(file);
            fos.write("I fear you speak upon the rack,"
              + "where men enforced do speak "
              + "anything.".getBytes());
```

❷ Get /sdcard directory reference

❸ Instantiate File for path

❹ Use mkdir() to create directory

❺ Get reference to File

❻ Create file

❼ Write with FileInputStream

The first thing we need to do in the ReadWriteSDCardFile class is to establish a filename for the file we want to create (1).

we create a File object reference to the /sdcard directory (2)

We create a File reference to a new subdirectory, /sdcard/unlocking_android (3)

We instantiate a reference File object (5), and we call createFile() to create a file on the filesystem (6)

we then use a FileInputStream to write some data into the file

External storage via an SD card

```
}
File rFile =
 new File("/sdcard/unlocking_android/" + fileName);
if (rFile.exists() && rFile.canRead()) {
   FileInputStream fis = null;
   try {
      fis = new FileInputStream(rFile);
```

8 Use new File object for reading

After we create the file and have data in it, we create another File object with the full path to read the data back

```
   byte[] reader = new byte[fis.available()];
   while (fis.read(reader) != -1) {
   }
   this.readOutput.setText(new String(reader));
} catch (IOException e) {
```

9 Read with FileOutputStream

With the File reference we then create a FileOutputStream and read back the data that was earlier stored in the file

Support BDD SQLite

- One nice convenience that the Android platform provides is the fact that a relational database is built in.

- Android uses SQLite (open-source, stand-alone SQL database)

- SQLite doesn't have all of the features of larger client/server database products, but it does cover just about anything you might need for local data storage. SQL usage in general : CREATE, INSERT, UPDATE, DELETE, and SELECT

- Any databases you create will be accessible by name to any class in the application, but not outside the application.

http://www.sqlite.org/

Persisting data to a database

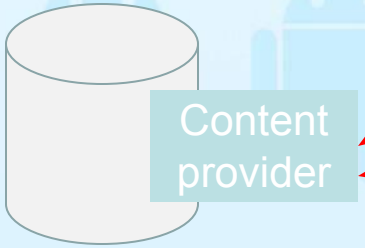
Databases are package private

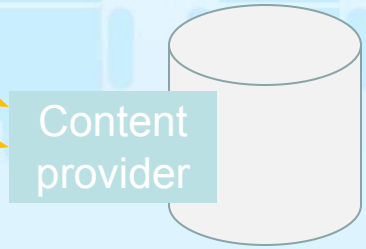
Unlike the SharedPreferences we saw earlier, you can't make a database WORLD_READABLE. Each database is accessible only by the package in which it was created—this means accessible only to the process that created it. If you need to pass data across processes, you can use AIDL/Binder (as spoke about) or create a ContentProvider (as we will discuss next), but you can't use a database directly across the process/package boundary.


ContentProvider

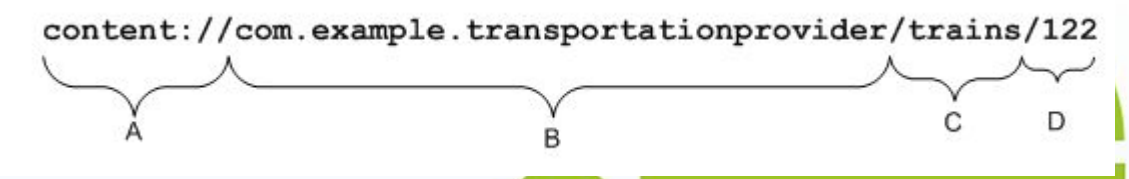
ContentProvider

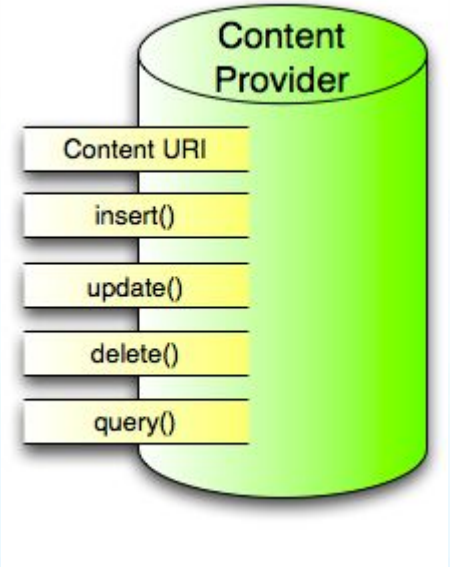
Un contentProvider sert à stocker et récupérer des données et ainsi les rendre accessibles à toutes les applications. C'est le moyen le plus efficace pour partager des données entres différentes applications. Par exemple, il existe unContent Provider gérant les Contacts du téléphone.





Content Provider : URI



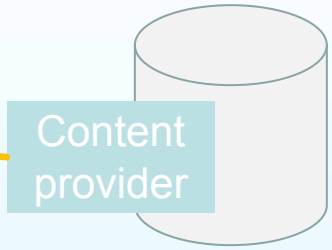


- A : Un préfixe standard, il sert à indiquer que les données sont contrôlées par un ContentProvider.
- B : L'autorité qui contrôle cette URI. Elle identifie le ContentProvider responsable de cette URI.
- C : Permet au ContentProvider de savoir quelle donnée est requêtée par l'url. Ce segment est optionnel. Un content provider peut exposer plusieurs données (ici les trains) mais on pourrait avoir par exemple les voitures, donc ce ContentProvider pourra gérer deux types de données.
- D : L'id de la donnée qu'on souhaite récupérer. (optionnel)

Content Provider : URI existantes

http://developer.android.com/reference/android/provider/package-summary.html

URI	Purpose
content://contacts/people/	Return List of all people from provider registered to handle content://contacts
content://contacts/people/1	Return or manipulate single person with ID 1 from provider registered to handle content://contacts



Content Provider : resolver

Resolver(URI)

android.provider.Contacts.Phones.CONTENT_URI
android.provider.Contacts.Photos.CONTENT_URI

Every ContentResolver method takes the URI as its first argument

Content provider 1

Content provider 2

Content provider 3

Querying a Content Provider :

You need three pieces of information to query a content provider:

The URI that identifies the provider
The names of the data fields you want to receive
The data types for those fields

*

Content Provider : modèle de données

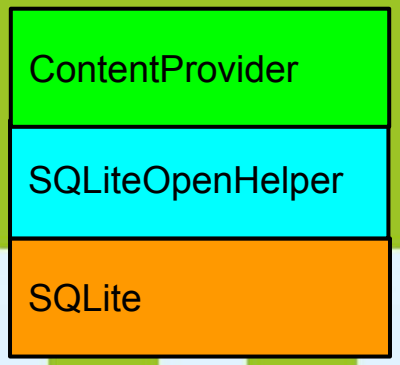
Content providers expose their data as a simple table on a database model, where each row is a record and each column is data of a particular type and meaning. For example :

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

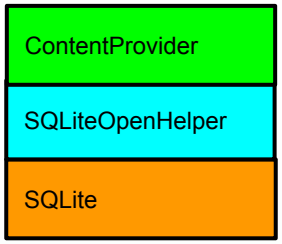
Content Provider : Structure logiciel

Pour créer un ContentProvider :

- La classe **ContentProvider**
- Les contents providers utilisent généralement **SQLite**, la classe **SQLiteOpenHelper**
- Déclarer votre Content Provider dans le manifest (AndroidManifest.xml)



Content Provider : la classe ContentProvider

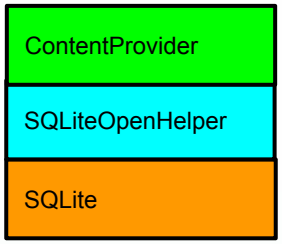


Un Content Provider doit surcharger les 6 méthodes suivantes :

- query() : Cette méthode retourne un objet Cursor sur lequel vous pouvez itérer pour récupérer les données.
- insert() : Cette méthode est utilisé pour rajouter des données à notre ContentProvider.
- update() : Cette méthode est utilisé pour mettre à jour une données déjà existante dans notre Content Provider.
- delete() : Cette méthode permet de supprimer une données du Content Provider.
- getType() : Retourne le type MIME des données contenues dans le Content Provider.
- onCreate() : Appeler afin d'initialiser le Content Provider

```
public class WidgetProvider extends ContentProvider
{
.....
}
```

Content Provider : la classe SQLiteOpenHelper



SQLiteOpenHelper est une classe qui permet de gérer principalement la création de la base de données, il est nécessaire de surcharger les méthodes suivantes:

```
synchronized void    close() : Close any open database object.
String  getDatabaseName() : Return the name of the SQLite database being opened, as given to the constructor.
SQLiteDatabase       getReadableDatabase() : Create and/or open a database.
SQLiteDatabase       getWritableDatabase() : Create and/or open a database that will be used for reading and writing.
void    onConfigure(SQLiteDatabase db) : Called when the database connection is being configured, to enable features such
as write-ahead logging or foreign key support.
abstract void onCreate(SQLiteDatabase db) : Called when the database is created for the first time.
void    onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) : Called when the database needs to be downgraded.
void    onOpen(SQLiteDatabase db) : Called when the database has been opened.
abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) : Called when the database needs to be
upgraded.
void    setWriteAheadLoggingEnabled(boolean enabled) : Enables or disables the use of write-ahead logging for the database.
```

SQLiteOpenHelper fait ensuite appel explicitement à un objet SQLite pour gérer les requêtes.

http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

Content Provider : la classe SQLiteOpenHelper

The recommended method to create a new SQLite database is to create a subclass of SQLiteOpenHelper and override the onCreate() method, in which you can execute a SQLite command to create tables in the database

```
private static class DatabaseHelper extends SQLiteOpenHelper
 {
 DatabaseHelper(Context context)
 {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}


 @Override public void onCreate(SQLiteDatabase db)
{
db.execSQL(DATABASE_CREATE);
 }


@Override public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
 {
 Log.w(TAG, "Upgrading database from version " + oldVersion + " to " + newVersion + ", which will destroy all old data");
db.execSQL("DROP TABLE IF EXISTS titles");
onCreate(db);
 }
 }
```

Content Provider : préparer la requête avec cursor

```
import android.provider.Contacts.People;
import android.database.Cursor;

// Form an array specifying which columns to return.
String[] projection = new String[] {
                        People._ID,
                        People._COUNT,
                        People.NAME,
                        People.NUMBER
                      };

// Get the base URI for the People table in the Contacts content provider.
Uri contacts =  People.CONTENT_URI;

// Make the query.
Cursor managedCursor = managedQuery(contacts,
                    projection, // Which columns to return
                    null,       // Which rows to return (all rows)
                    null,       // Selection arguments (none)
                    // Put the results in ascending order by name
                    People.NAME + " ASC");
```

An example query to retrieve a list of contact names and their primary phone numbers:

Content Provider : résultat du cursor

The Cursor object returned by a query provides access to a recordset of results.

```
import android.provider.Contacts.People;

private void getColumnData(Cursor cur){
    if (cur.moveToFirst()) {

        String name;
        String phoneNumber;
        int nameColumn = cur.getColumnIndex(People.NAME);
        int phoneColumn = cur.getColumnIndex(People.NUMBER);
        String imagePath;

        do {
            // Get the field values
            name = cur.getString(nameColumn);
            phoneNumber = cur.getString(phoneColumn);

            // Do something with the values.
            ...

        } while (cur.moveToNext());

    }
}
```

If a query can return binary data, such as an image or sound, the data may be directly entered in the table or the table entry for that data may be a string specifying a content: URI that you can use to get the data. In general, smaller amounts of data (say, from 20 to 50K or less) are most often directly entered in the table and can be read by calling Cursor.getBlob(). It returns a byte array.

Content Provider : Modifying Data

Adding records

To add a new record to a content provider, first set up a map of key-value pairs in a ContentValues object, where each key matches the name of a column in the content provider and the value is the desired value for the new record in that column.

```
import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;

ContentValues values = new ContentValues();

// Add Abraham Lincoln to contacts and make him a favorite.
values.put(People.NAME, "Abraham Lincoln");
// 1 = the new contact is added to favorites
// 0 = the new contact is not added to favorites
values.put(People.STARRED, 1);

Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

Content Provider : Data changed

```
package com.test.contentobserver;

import android.app.Activity;
import android.database.ContentObserver;
import android.os.Bundle;
import android.provider.Contacts.People;

public class TestContentObserver extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        this.getApplicationContext().getContentResolver().registerContentObserver (People.CONTENT_URI, true, contentObserver);
    }

    private class MyContentObserver extends ContentObserver {

        public MyContentObserver() {
            super(null);
        }
        @Override
        public void onChange(boolean selfChange) {
            super.onChange(selfChange);
        }
    }
    MyContentObserver contentObserver = new MyContentObserver();
}
```

What if the content changes after the fact?

When you use a `ContentProvider`, which by definition is accessible by any application on the system, and you make a query, you are getting only the current state of the data back. The data could change after your call, so how do you stay up to date? To be notified when a `Cursor` changes, you can use the `ContentObserver` API. `ContentObserver` supports a set of callbacks that are invoked when data changes. `Cursor` has `register` and `unregister` methods for `ContentObserver` objects.

Content Provider : manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.msi.manning.chapter5.widget">
   <application android:icon="@drawable/icon"
      android:label="@string/app_short_name">
      <activity android:name=".WidgetExplorer"
         android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <provider android:name="WidgetProvider"
            android:authorities=
             "com.msi.manning.chapter5.Widget" />
   </application>
</manifest>
```

1 Use provider element to define class and authority

Content Provider : sqlite3 Databases from a Remote Shell

```
adb [-d|-e|-s {<serialNumber>}] shell
```

```
$ adb -s emulator-5554 shell
# sqlite3 /data/data/com.google.rss.rssexample/databases/rssitems.db
SQLite version 3.3.12
Enter ".help" for instructions
.... enter commands, then quit...
sqlite> .exit
```

Content Provider : sécurité

Le contentProvider peut utiliser un base dans son propre package, ce qui rend l'accès impossible si l'on ne passe pas vers une URI.

Pour les applications externe au package il es possible de demander des droits en lecture/écriture et ainsi limité les accès. Dans ce cas le contentProvider doit déclarer les droits nécessaire pour les requêtes par exemple en lecture /écriture :

```
<permission android:name="android.provider.diary.READ"/>

<provider android:authorities="android.provider.diary"
     android:name="android.provider.diary.DiaryContentProvider
"
     android:readPermission="android.provider.diary.READ" />
```

Pour les applications appelantes :

```
<permission android:name="android.provider.diary.READ"/>
<uses-permission android:name="android.provider.diary.READ"
/>
```

TD

Analyser le code :

- ANDROID_TD4_0 : Analyser le code d'accès aux contacts
- ANDROID_TD3 : content provider
- ANDROID_TD3_0 (testeur)