Android

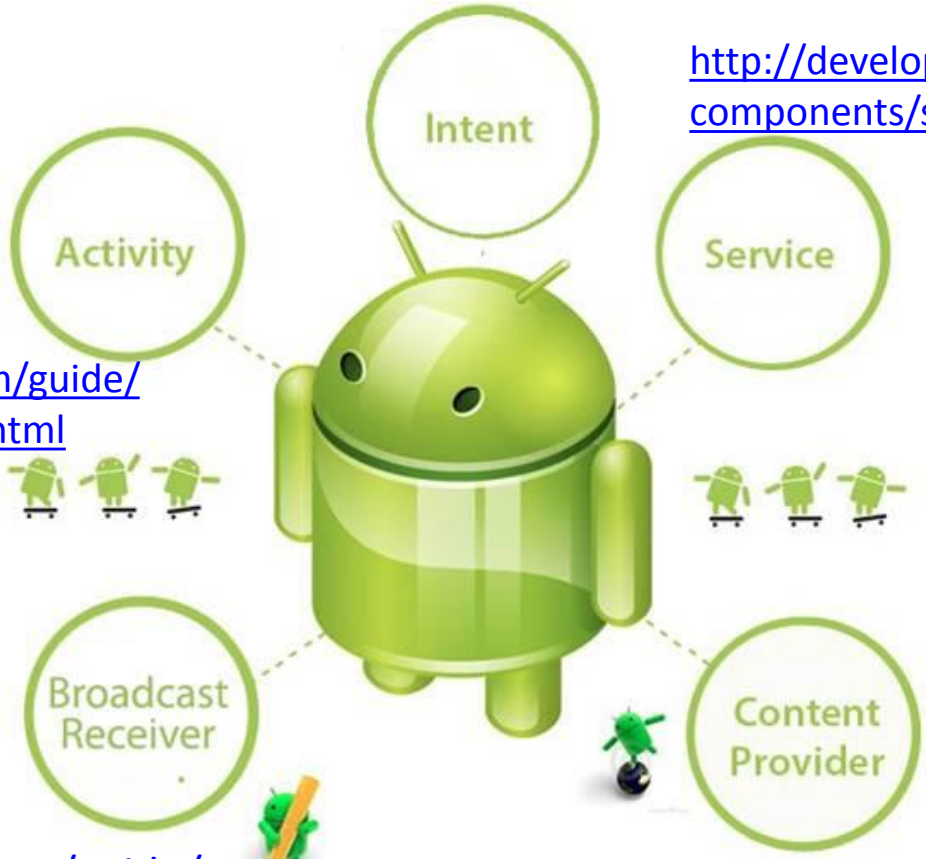

Cupcake
Donut
Eclair
Froyo
Gingerbread
Honeycomb
ICE Cream-Sandwitch
Jelly Bean
Kitkat
lollipop
Lollipop
Marshmallow
Nougat

/

Principes Android

http://developer.android.com/guide/components/intents-filters.html

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



https://developer.android.com/guide/components/activities/index.html

https://developer.android.com/guide/components/broadcasts.html

http://developer.android.com/guide/topics/providers/content-providers.html

On a vu !

Avec JavaFX

- Le partage Vues en XML et code en Java
- Des layouts
- Des vues spécifiques

Les spécificités Android : dispositifs et types d'applications visées

- Les activités : une vue = une activité
- L'accès aux capteurs du dispositif
- Le principe des Intents : communications entre « composants »
- L'importance des tâches de fond : AsyncTask
- Les ressources pour l'adaptation aux dispositifs, aux utilisateurs
- Le découpage de l'IHM en fragments pour la réutilisation et l'adaptation aux dispositifs

Il reste à voir

Pour le background et son impact sur la fluidité ou sur les interactions

- En savoir plus sur les fragments
- Le passage de données entre composants avec les intents
- L'accès aux Bases de données

Il reste à améliorer vos connaissances pour l'IHM

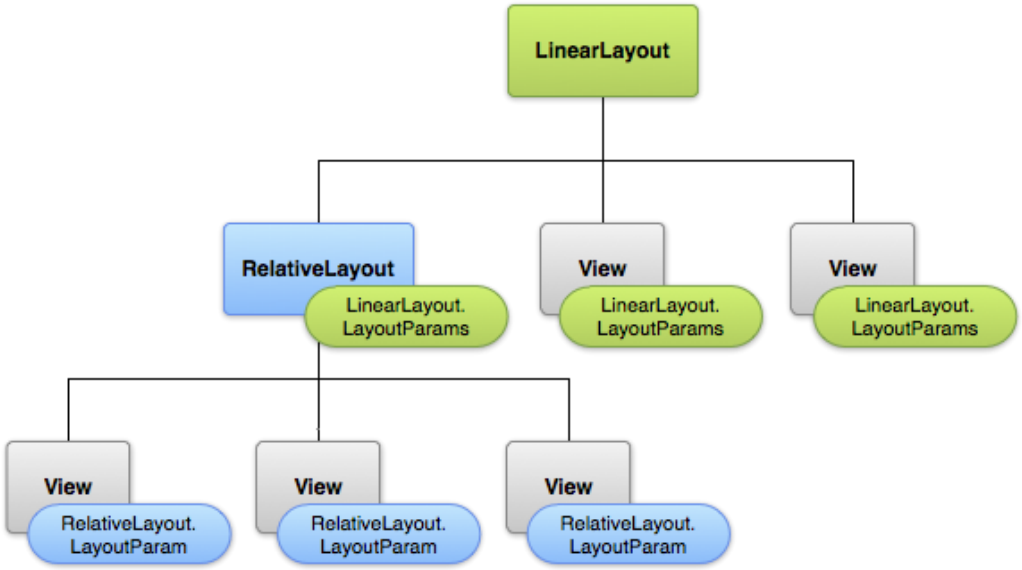
- Des layouts et des adapters
- Des vues spécifiques
- L'introduction de « l'élévation » dans le Material Design

LAYOUT

LAYOUT : Organisation générale d'une vue

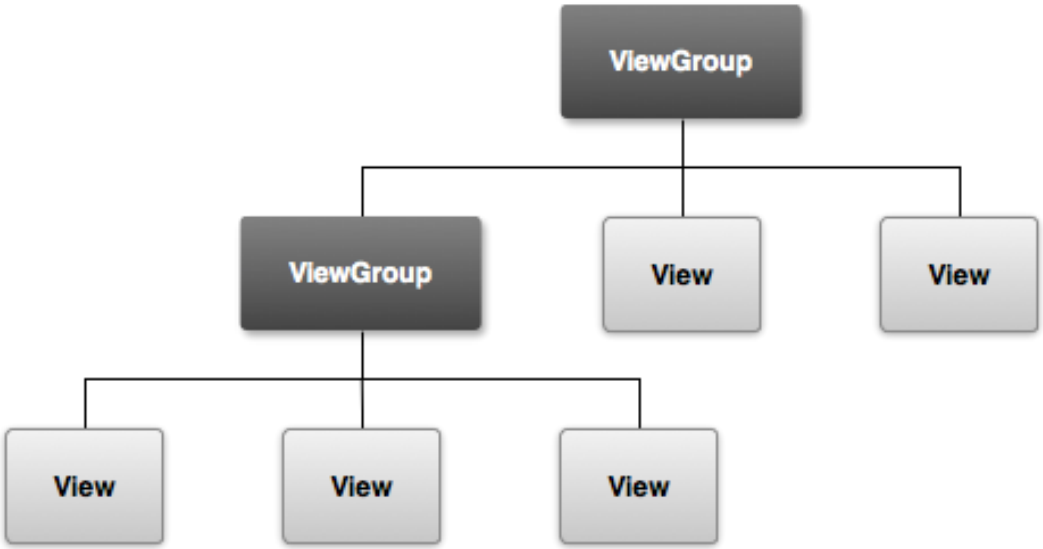
LAYOUT : Groupe de vues dérivé de *ViewGroup,* modèle de présentation des vues "filles".

.



Création de layout par héritage de classes existantes

Arborescences de vues



NE PAS CONFONDRE
HIERARCHIE DE COMPOSANTS
(structuration)

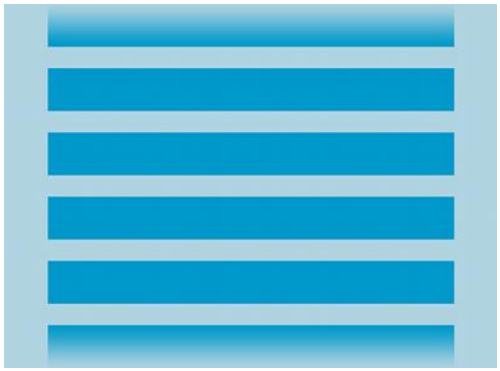
ET HERITAGE DE CLASSES

Exemples de layout

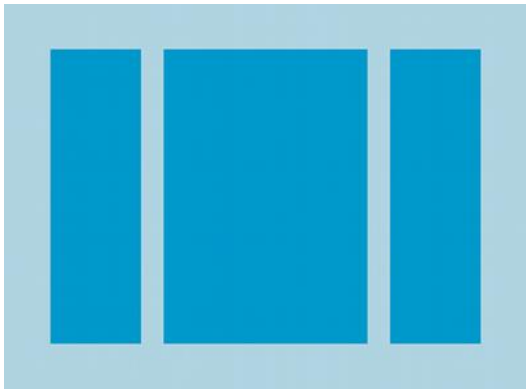
Conçus pour le Responsive design

- En lignes
- En colonnes
- En grille
- Relatif

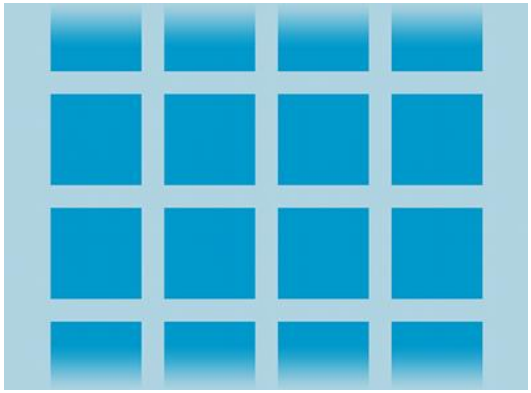
ListView



LinearLayout



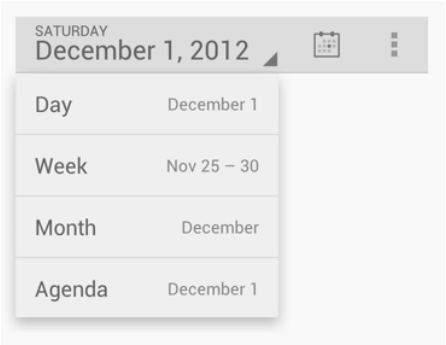
GridView

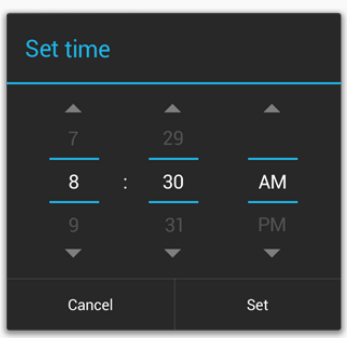


RelativeLayout

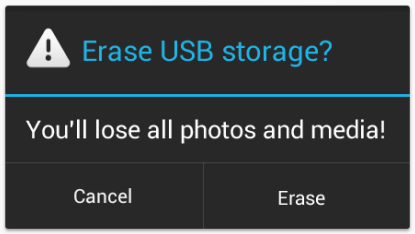
https://developer.android.com/guide/topics/ui/declaring-layout.html

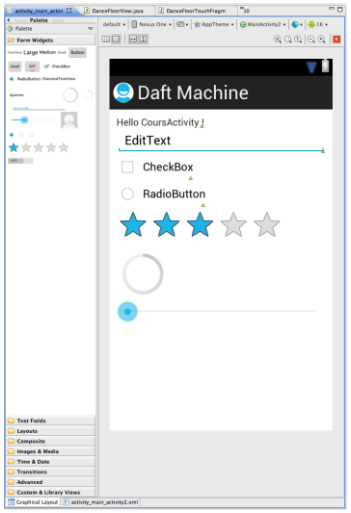
WIDGETS

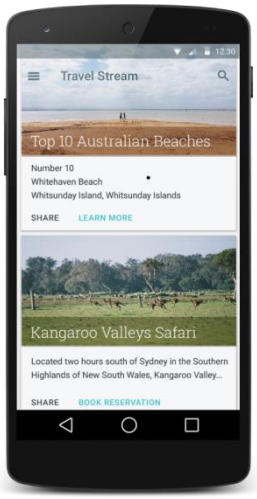

SATURDAY
December 1, 2012
Day December 1
Week Nov 25 – 30
Month December
Agenda December 1

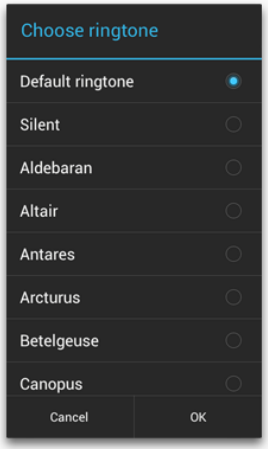

Set time
7 29
8 : 30 AM
9 31 PM
Cancel
Set

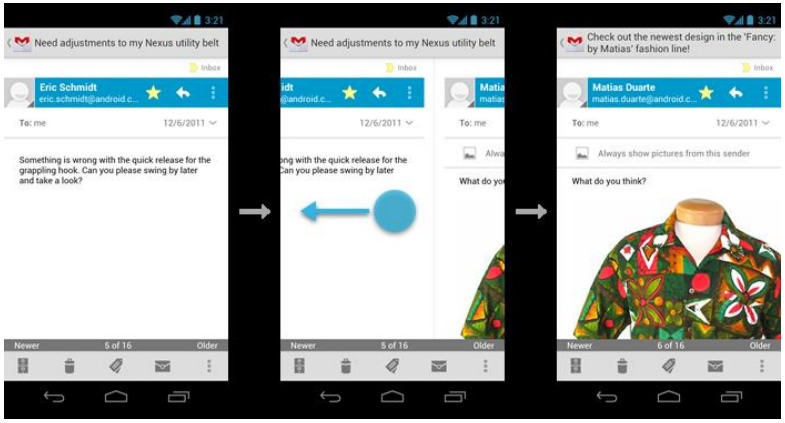

Alarm
Alarm

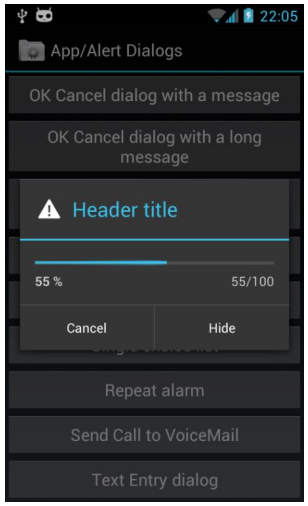

Erase USB storage?
You'll lose all photos and media!
Cancel
Erase

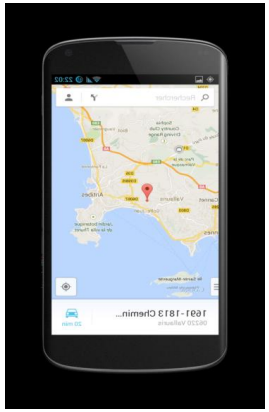

Daft Machine
Hello CoursActivity !
EditText
CheckBox
RadioButton

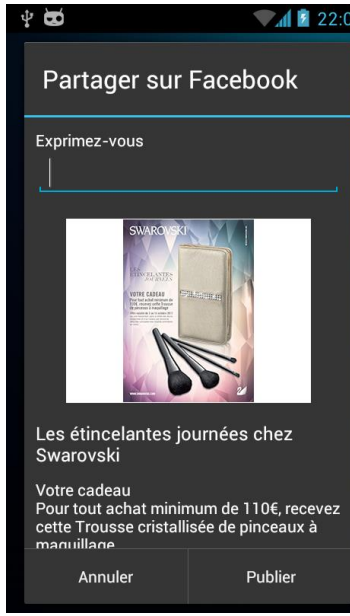

Travel Stream
Top 10 Australian Beaches
Number 10
Whitehaven Beach
Whitsunday Island, Whitsunday Islands
SHARE LEARN MORE
Kangaroo Valleys Safari
Located two hours south of Sydney in the Southern Highlands of New South Wales, Kangaroo Valley...
SHARE BOOK RESERVATION


Choose ringtone
Default ringtone
Silent
Aldebaran
Altair
Antares
Arcturus
Betelgeuse
Canopus
Cancel
OK


Need adjustments to my Nexus utility belt
Eric Schmidt
Something is wrong with the quick release for the grappling hook. Can you please swing by later and take a look?
Need adjustments to my Nexus utility belt
Check out the newest design in the 'Fancy: by Matias' fashion line!
Matias Duarte
What do you think?
Newer 5 of 16 Older


22:05
App/Alert Dialogs
OK Cancel dialog with a message
OK Cancel dialog with a long message
Header title
55 %
55/100
Cancel
Hide
Repeat alarm
Send Call to VoiceMail
Text Entry dialog

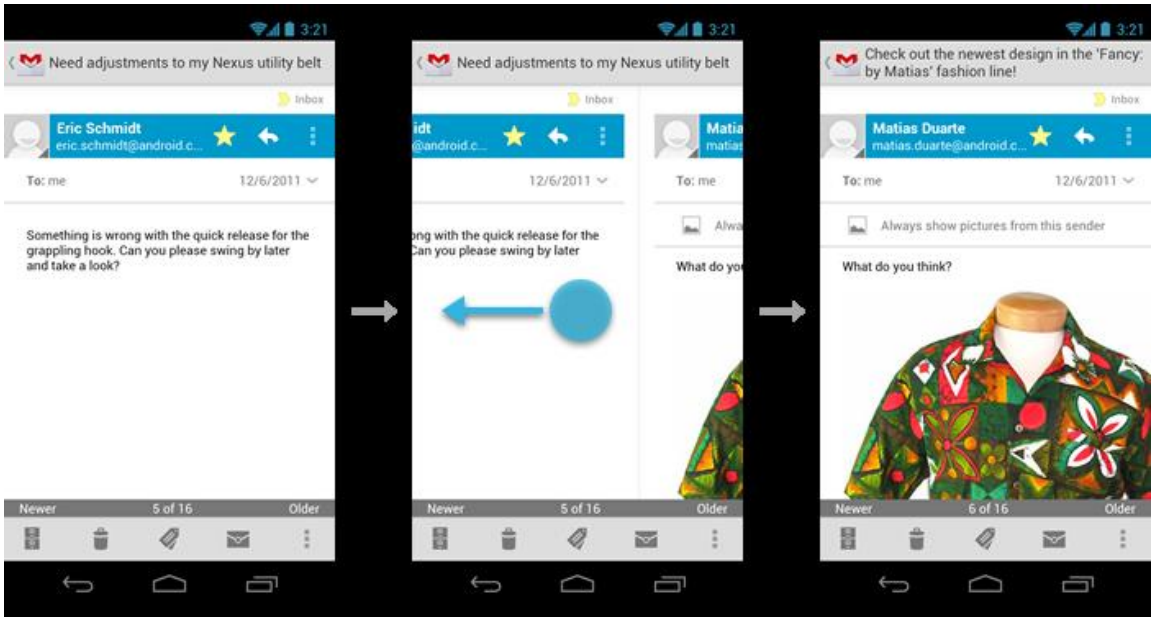


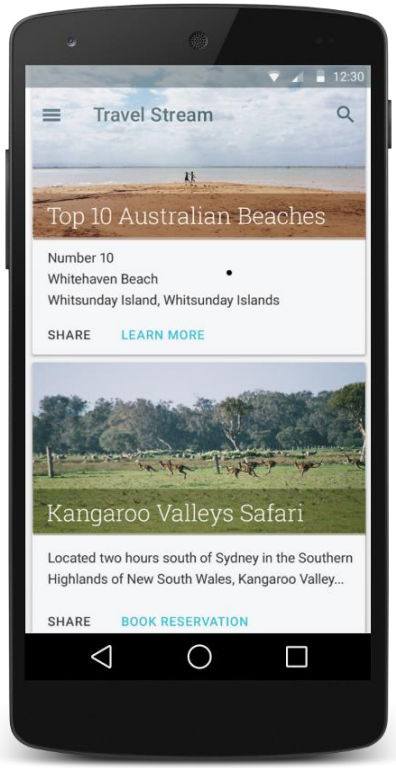

Partager sur Facebook
Exprimez-vous
SWAROVSKI
Les étincelantes journées chez Swarovski
Votre cadeau
Pour tout achat minimum de 110€, recevez cette Trousse cristallisée de pinceaux à maquillage
Annuler
Publier

Concrètement en Android

Vous pouvez utiliser des "Widgets" Android pour concevoir votre IHM

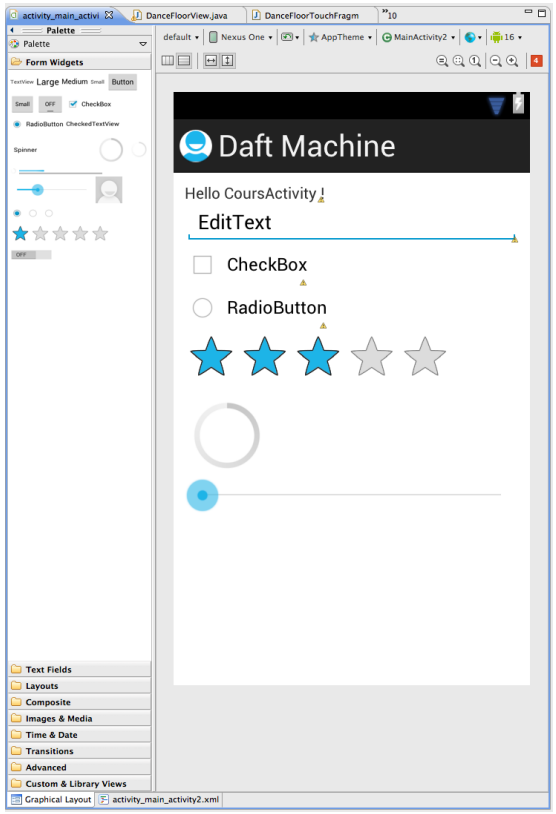
Vous pouvez aussi créer en sous classant des classes existantes vos propres vues.



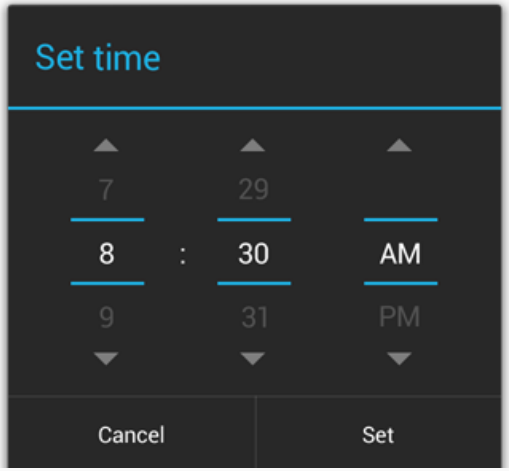


« WIDGETS classiques »

Formulaire

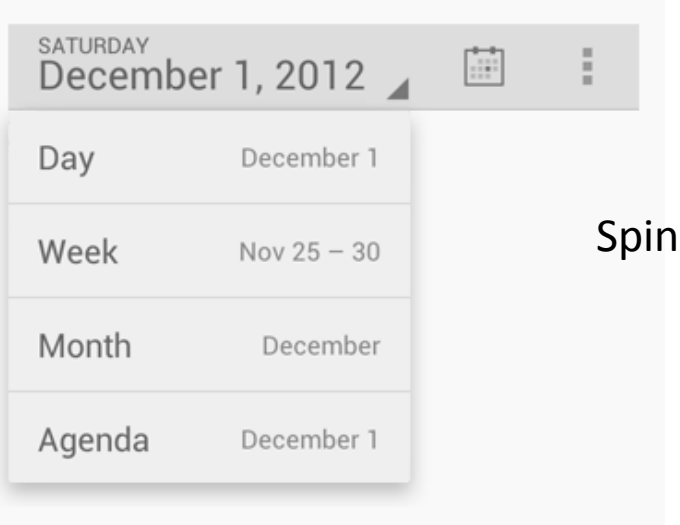


Gestion du temps



Boutons

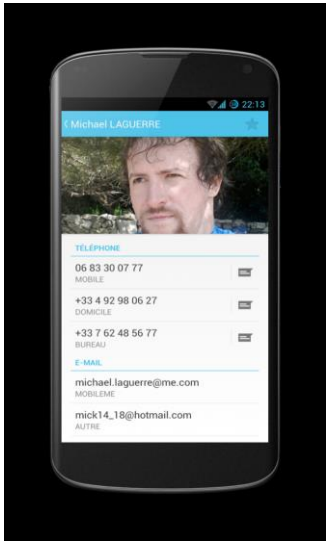
Data Time Picker

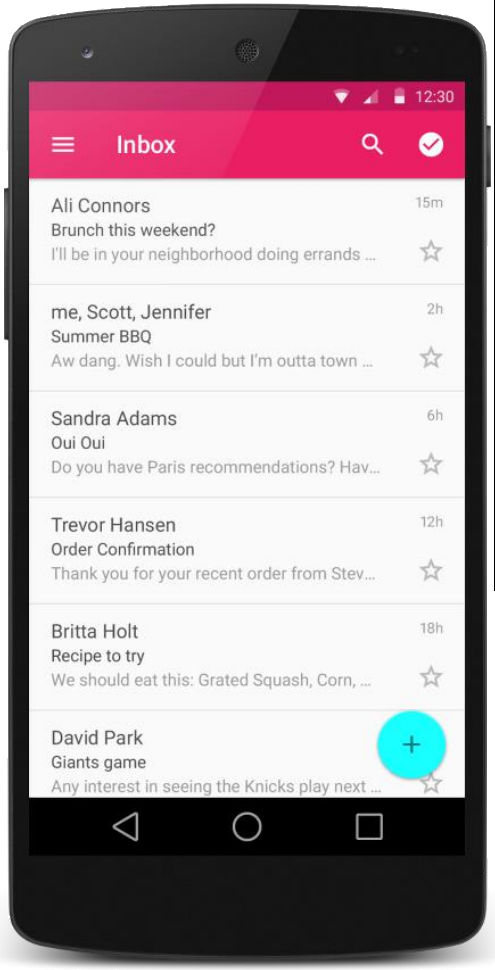


Spinner

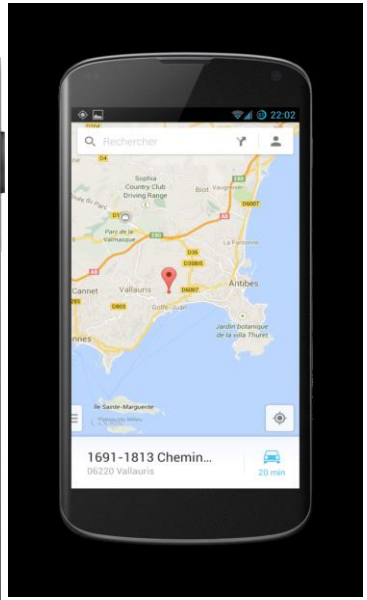
Vues spécialisées

WebView





ScrollView



MapView

ViewPager : galerie plein écran

1 Déclarer le Viewpager dans un layout.

2 Utiliser un adapter pour remplir le ViewPager
Le PageAdapter est le plus courant.

getCount(): retourne le nombres de pages du ViewPager
instantiateItem(View collection, int position): Crée une view à une position donnée.
destroyItem(View collection, int position, Object view): Supprime une vue d'une position donnée.

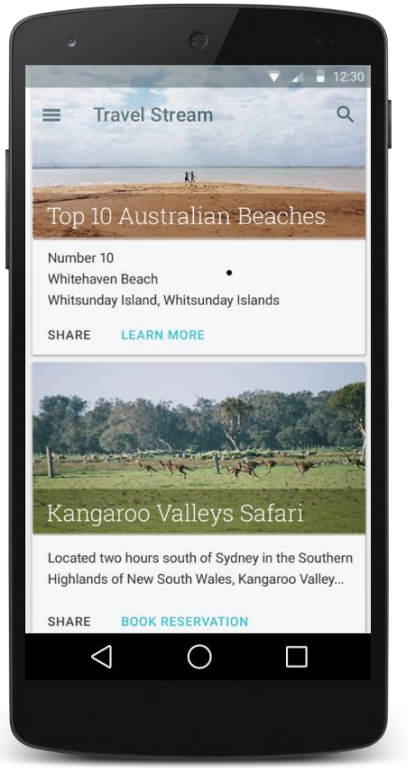
3 **Agir sur le Viewer depuis le code.**
Listener : setOnPageChangeListener(myPageChangeListener).
Cela permet par exemple d'indiquer le numéro de page sur une TextView.

Souvent lié aux Fragments pour réutiliser des parties d'activité dans d'autres activités (cf. **FragmentPagerAdapter**).

https://developer.android.com/training/animation/screen-slide.html

CardView

CardView est un élément respectant le style Material Design.
Ajout d'une profondeur dans l'application : 3ème index (z-index en css), l'élévation.
Une image et un texte associé



CardView hérite de FrameLayout
Peuvent être personalisées avec des ombres et des coins arrondis

Ombres : via l'attribut card_view:cardElevation.

Autres attributs utiles :
card_view:cardCornerRadius ouCardView.setRadius.
card_view:cardBackgroundColor.

https://developer.android.com/training/material/lists-cards.html

>

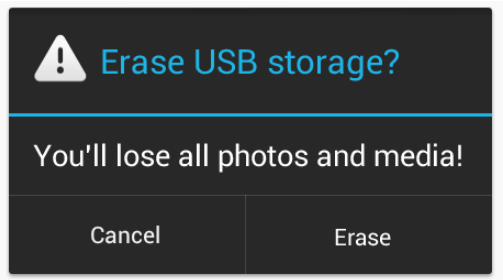
Exemple CardView

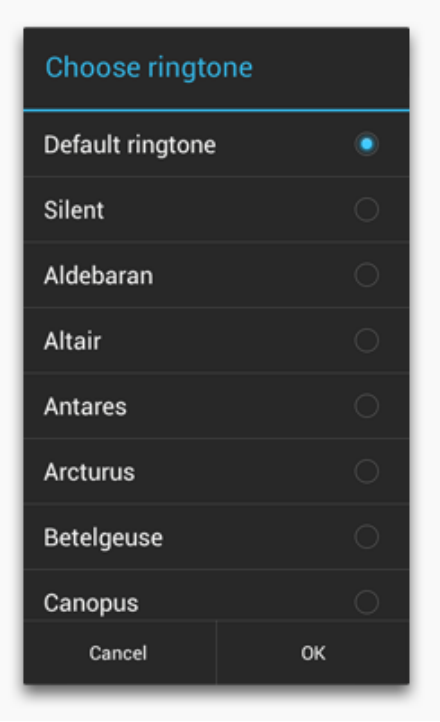
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    ... >
    <!-- A CardView that contains a TextView -->
    <android.support.v7.widget.CardView
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:id="@+id/card_view"
        android:layout_gravity="center"
        android:layout_width="200dp"
        android:layout_height="200dp"
        card_view:cardCornerRadius="4dp">

        <TextView
            android:id="@+id/info_text"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </android.support.v7.widget.CardView>
</LinearLayout>
```

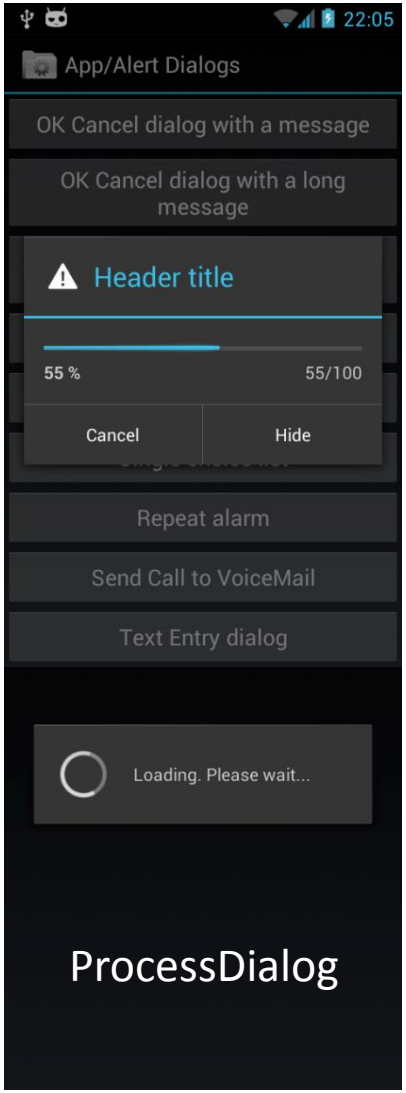
http://developer.android.com/design/building-blocks/dialogs.html

Dialogues

AlertDialog

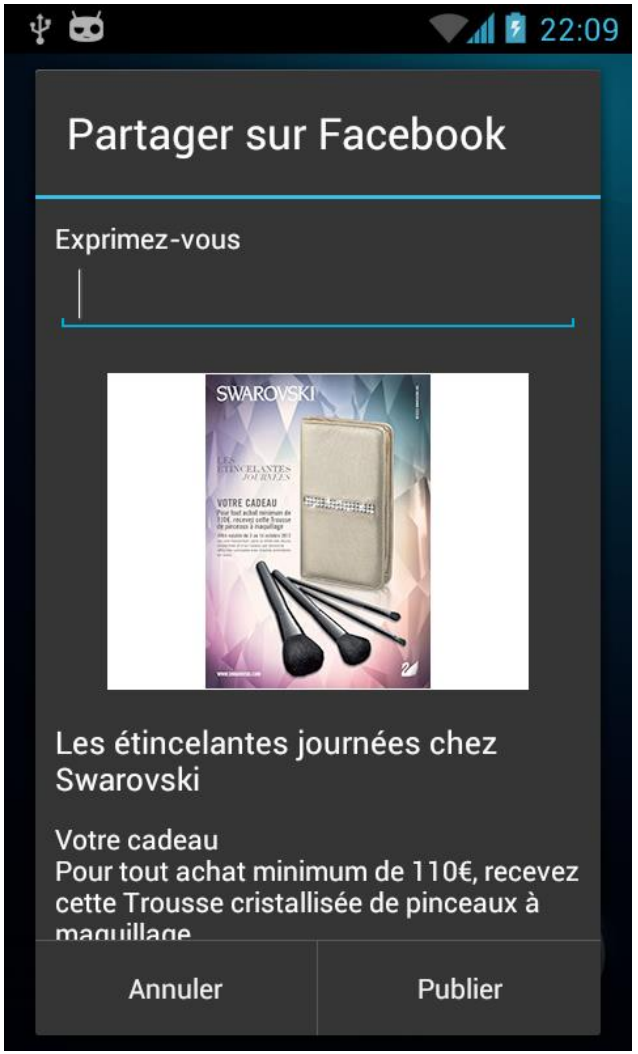






ProcessDialog

CustomDialog



COHERENCE ENTRE ACTIVITES

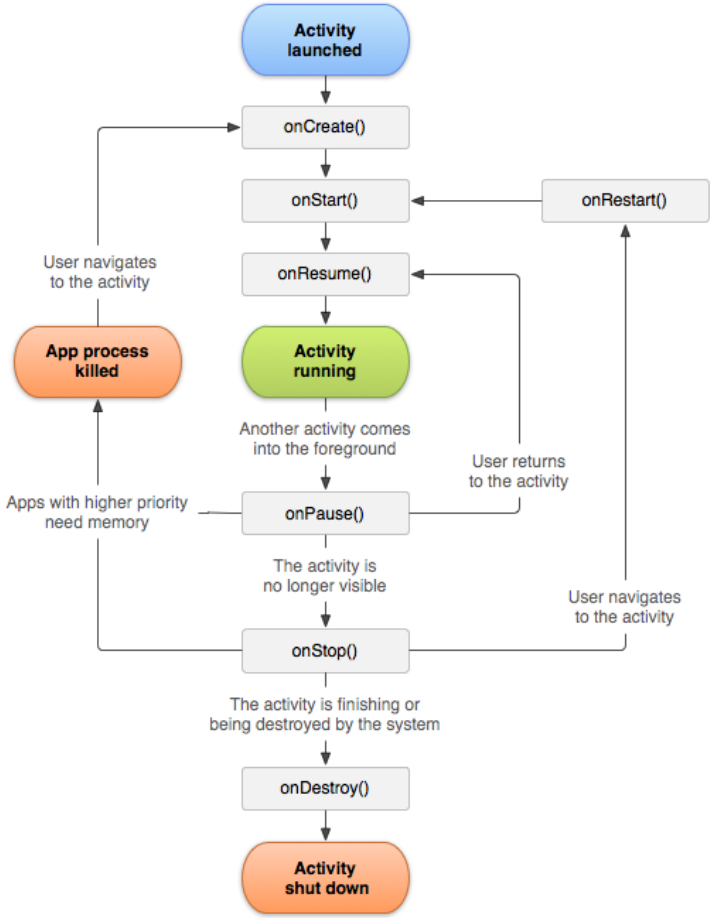
Cycle de vie d'une activité

Cycle de vie ***global***
onCreate() -> onDestroy()

Cycle de vie ***visible***
onStart() -> onStop()

Affichée à l'écran mais peut ne pas être utilisable
(en second plan)

- Cycle de vie ***en premier plan***
- onResume() -> onPause()



http://supertos.free.fr/supertos.php?page=1076

Gestion de la cohérence entre activités

Cycles de transitions de 2 Activités dépendantes liés

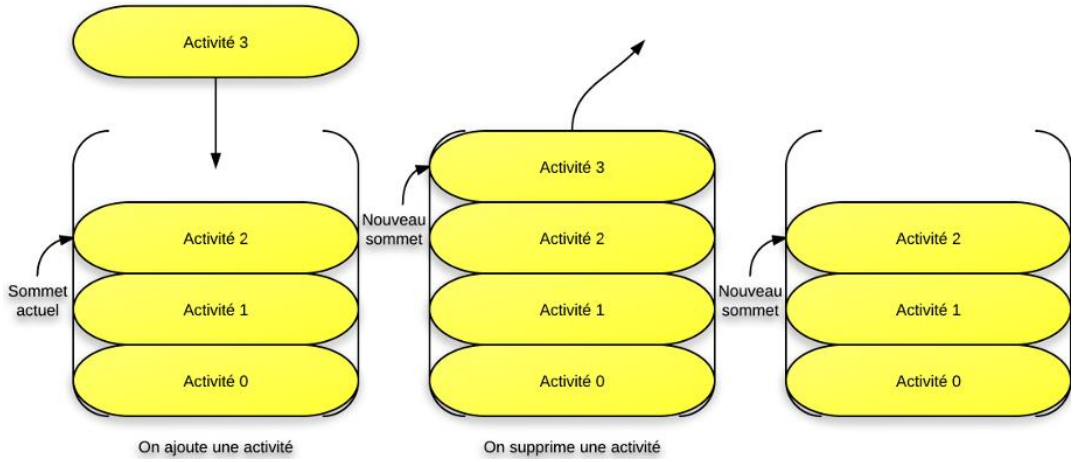
Si A démarre B alors

La méthode onPause() de A est exécutée

Les méthodes onCreate(), onStart(), et onResume() de B sont ensuite exécutées en séquence.

B a le focus.

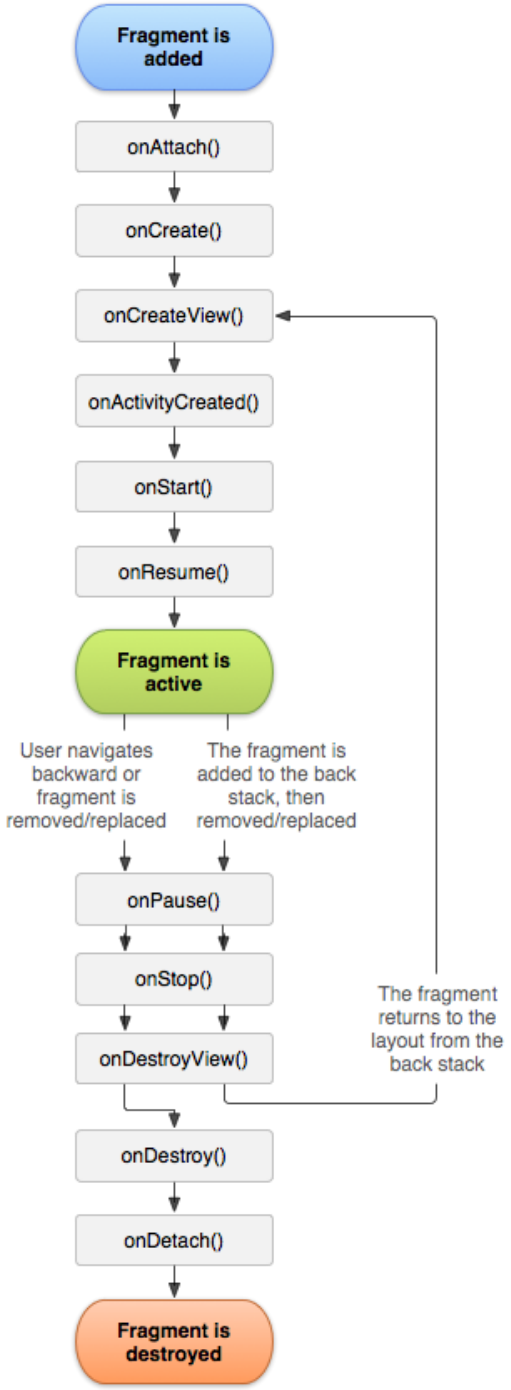
Si A reste longtemps non visible à l'écran alors sa méthode onStop() est appelée



ATTENTION aux données persistantes partagées. I

Ecrire dans la base de données pendant onPause au lieu de onStop

FRAGMENTS : COHERENCE



Coordination avec le cycle de vie d'une activité

onAttach() : appelée quand le fragment est associé à une Activité

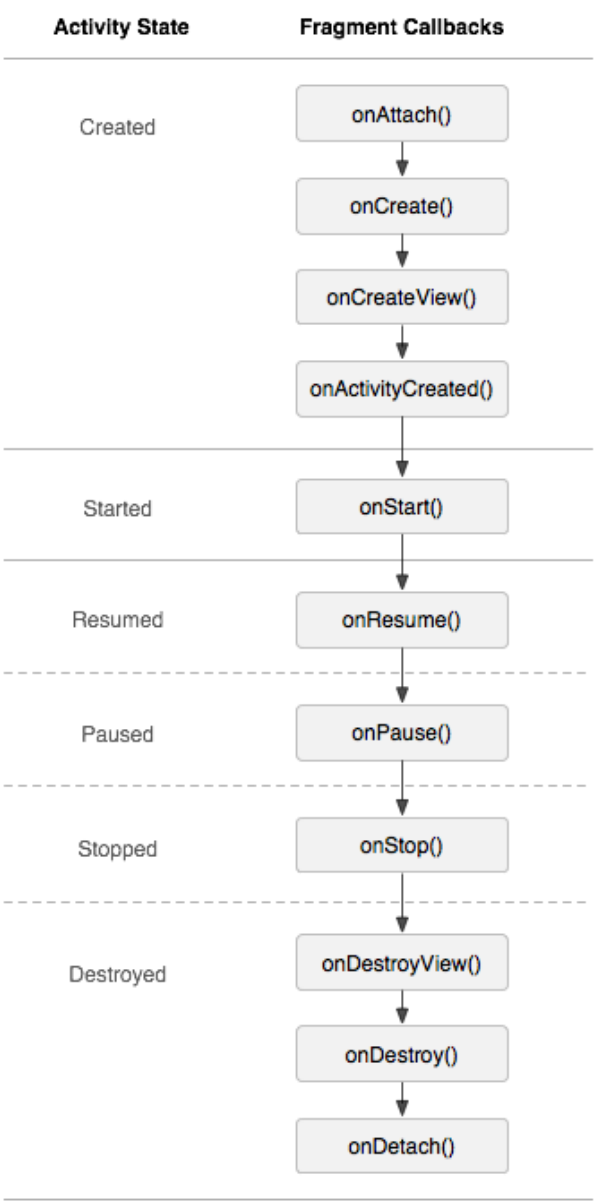
onCreateView() : appelée pour créer la vue associée au fragment

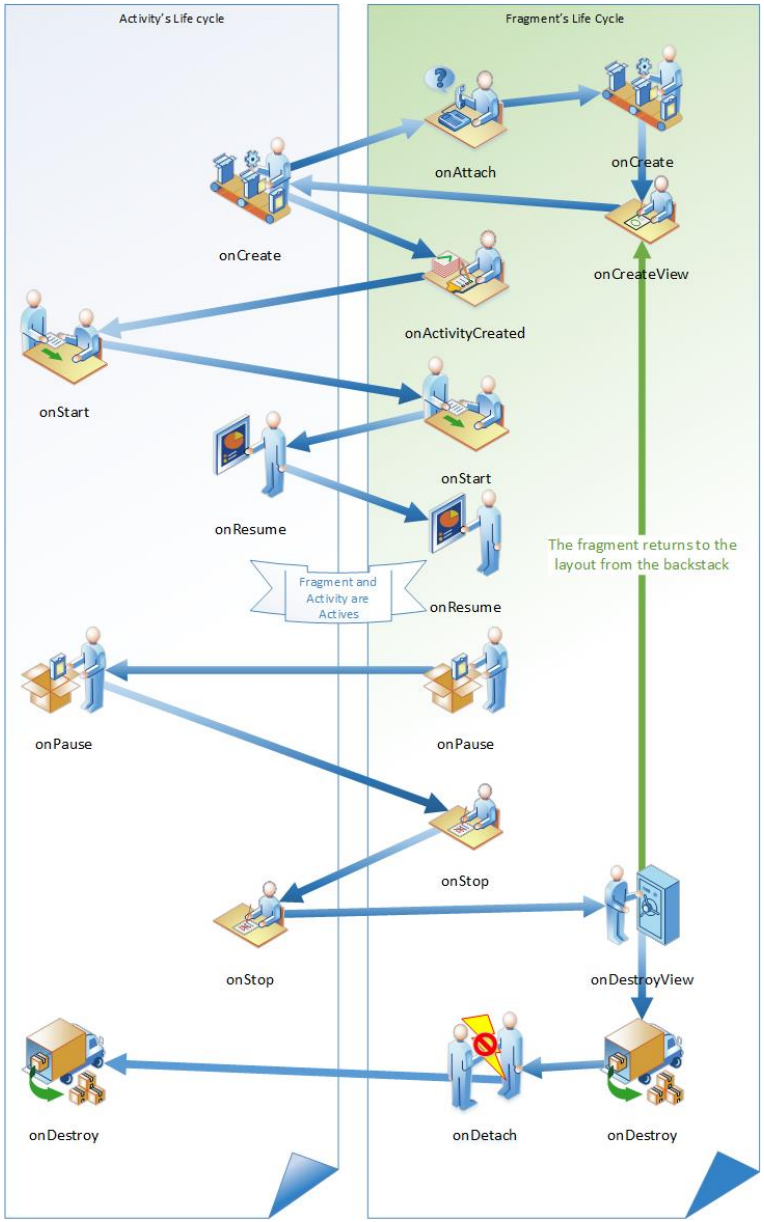
onActivityCreated() : lorsque l'activité est créée

onDestroyView() : lorsque la vue associée au fragment est tuée

onDetach() : lorsque le fragment est dissocié de l'Activité

Ce n'est que lorsqu'une activité est dans l'état *resumed*, qu'il est possible d'ajouter et effacer des fragments à l'activité …




Activity's Life cycle
Fragment's Life Cycle
onAttach
onCreate
onCreate
onCreateView
onActivityCreated
onStart
onStart
onResume
Fragment and Activity are Actives
onResume
The fragment returns to the layout from the backstack
onPause
onPause
onStop
onStop
onDestroyView
onDestroy
onDetach
onDestroy

Création de fragments

- Dynamique ou statique ?
  - A la déclaration d'un Layout
  - A une activité
  - Par programmation

Création à la déclaration du layout de l'activité

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Ajout d'un fragment d'UI à une activité

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

R.layout.example_fragment référence un layout

Container : endroit où placer le fragment dans l'activité qui va l'accueillir,

Bundle : données stockées si c'est le cas.

La méthode **inflate** prend l'ID du layout à étendre, le ViewGroup où intégrer, un booleen à true ou false si l'intégration doit être faite ou pas.

Evolution d'UI via les fragments à l'exécution

Ajouter un Fragment, en remplacer un ou en supprimer un

Pré-conditions :
Ajouter le fragment initial à la création : onCreate() method.

Le layout de l'activité doit inclure une View qui permet d'insérer le fragment

Pour ajouter ou enlever un fragment : utiliser une transaction

Pourquoi une transaction ?
Comment ça marche ?
Combien de transactions ?

Evolution d'UI via les fragments à l'exécution

Utiliser un **FragmentManager** pour créer une **FragmentTransaction**

Les méthodes à utiliser :
getSupportFragmentManager() pour récupérer un FragmentManager à partir d'une activité.
beginTransaction() pour créer une FragmentTransaction
add() pour ajouter un fragment.
replace() pour remplacer un fragment.
remove() pour supprimer un fragment.

commit() pour effectuer les changements

Transactions et retour arrière

Conserver le "undo" en cas de retour arrière.

=> Nécessité de gérer la pile d'exécution de l'activité

En cas de remove ou replace, appeler addToBackStack() pour stocker la transaction FragmentTransaction, avant le commit.

=> Le fragment qui est détruit est stoppé en cas de retour arrière avec restauration du fragment il est redémarré (restart).

Dans le cas contraire il est détruit

addToBackStack() prend en argument le nom unique de la transaction. Utile seulement pour des opérations avancées sur le fragment via FragmentManager.BackStackEntry APIs.

Ajout

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.news_articles);

      if (findViewById(R.id.fragment_container) != null) {
         if (savedInstanceState != null) {
            return;
         }

         HeadlinesFragment firstFragment = new HeadlinesFragment();
         firstFragment.setArguments(getIntent().getExtras());
         getSupportFragmentManager().beginTransaction()
               .add(R.id.fragment_container, firstFragment).commit();
      }
   }
}
```

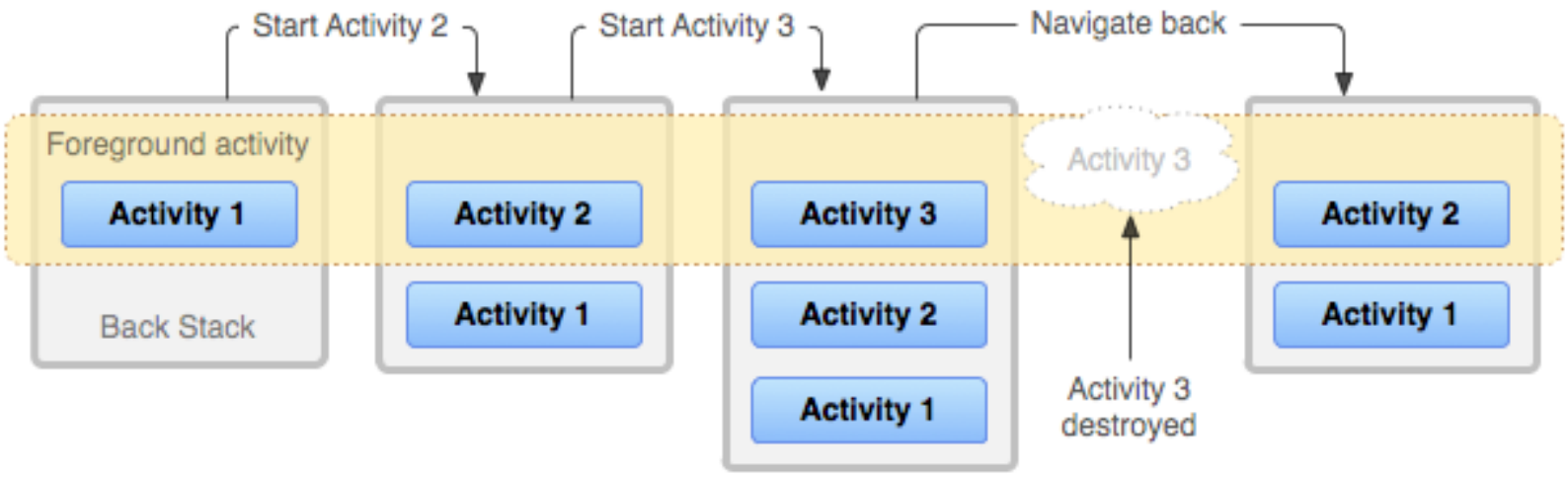
Remplacement

```
ArticleFragment newFragment = new ArticleFragment();
Bundle args = new Bundle();
args.putInt(ArticleFragment.ARG_POSITION, position);
newFragment.setArguments(args);

FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();


transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

transaction.commit();
```

https://developer.android.com/training/basics/fragments/fragment-ui.html

Gérer le retour arrière

Retrouver l'état du fragment par retour arrière



Gestion de pile implicite vs gestion de pile explicite

Gestion explicite de la pile

Pile d'activités gérée par le système ou par le bouton back
Fragment lié à une pile associée à l'activité Hote seulement avec **addToBackstack**

Conserver l'état :

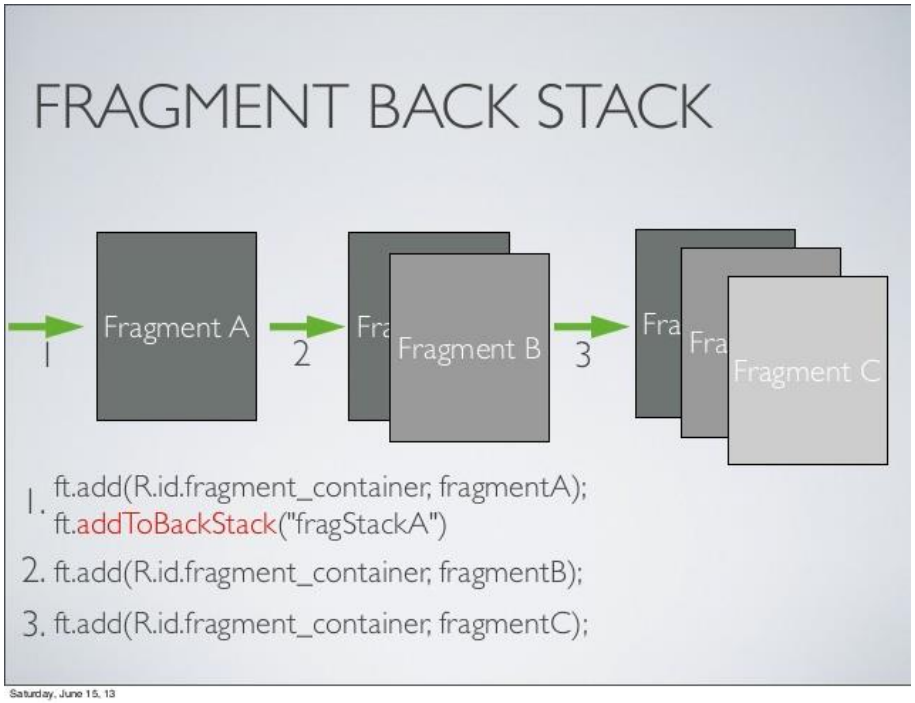
Stocker l'état d'un fragment : utiliser un Bundle avec la méthode *onSaveInstanceState* pour le récupérer dans *onCreate /onCreateView* ou *onActivityCreated*

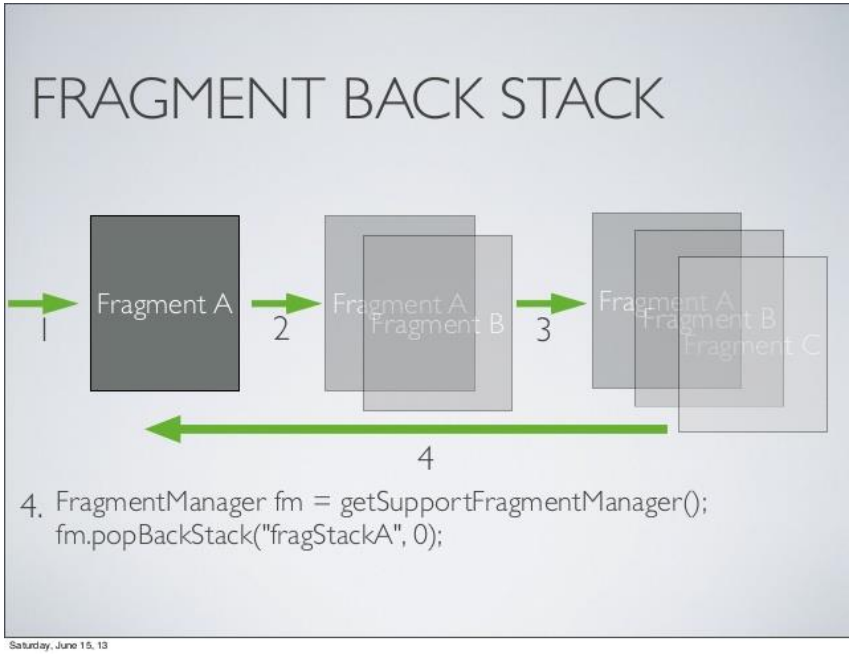
Les transactions dans lesquelles les fragments sont modifiés peuvent être placées dans la pile interne de l'activité et sont donc dépilées avant la fin de l'activité

Gestion de la mémoire

Quand on manipule les fragments dynamiquement, il est toujours nécessaire de savoir où on en est ; quel est l'état de la backstack, quels sont les fragments instanciés, détruits. Le choix de la méthode show, add ou replace est primordial, de même que l'appel aux méthodes *addToBackStack* et *popBackStack.*

A Faire avant le *commit*





Communication Fragments/Activités

Un fragment **DOIT** être implémenté indépendamment de l'activité.

Accès à l'activité à partir du fragment

Pour accéder à l'instance de l'activité : *getActivity*()

Par exemple pour récupérer un élément.

```
View listView = getActivity().findViewById(R.id.list);
```

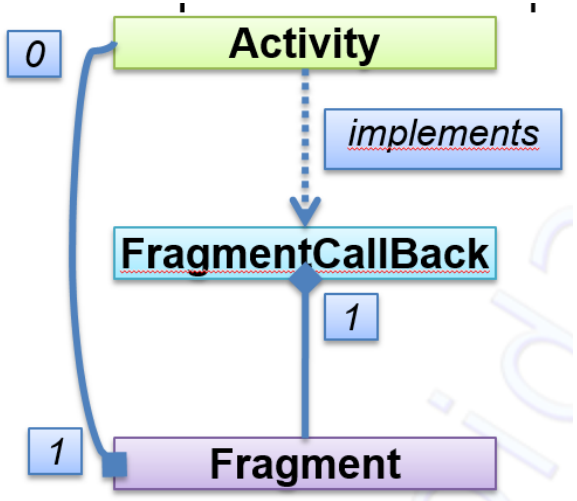
Accès au fragment à partir de l'activité

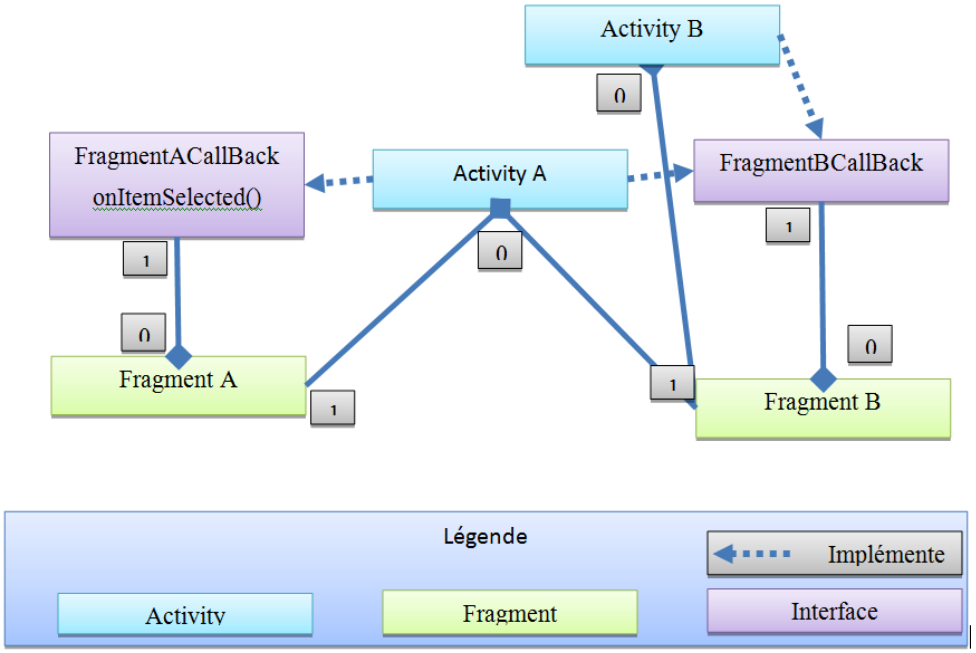
En utilisant le *FragmentManager*, et les méthodes *findFragmentById*() ou *findFragmentByTag*().

Par exemple

```
ExampleFragment fragment = (ExampleFragment) getFragmentManager().findFragmentById(R.id.example_fr)
```

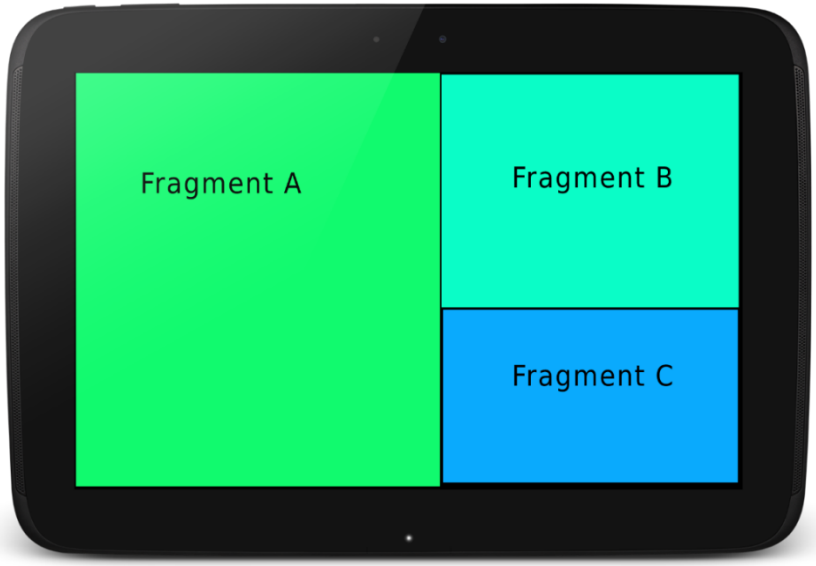
Conseils de communication

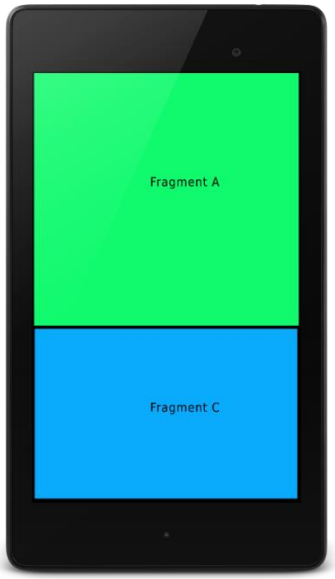




ORIENTATION DES DISPOSITIFS

Fragments et Layout : réutilisation





Activités et Orientation

Impact sur le cycle de vie des changements d'orientation

Appel des fonctions onResume() et onPause()

On peut fixer une orientation et interdire les autres :

< **activity**

android :name= ".MainActivity"

...

android :screenOrientation= "portrait"

android :configChanges= "orientation">

android:screenOrientation indique que l'activité doit rester bloquée en mode portrait.

android:configChanges empêche les fonctions *onResume()* et *onPause()* de l'activité d'être appelées lorsque l'orientation de l'appareil change. Sans cette ligne, la vue ne changera pas si l'utilsateur tourne son téléphone, mais ces méthodes seraient malgré tout appelées .

INTENTS : PLUS DE DETAILS

Comment ?

Pour démarrer une **activité** :

en paramètre de *startActivity* ou

de startActivityForResult() pour recevoir un résultat sous forme d'objet message à la fin de l'activité.

Et aussi…

Pour démarrer un **service** : un service est un composant qui exécute des opérations en background (sans UI), par exemple pour charger un fichier.

En passant un Intent à *startService()* ou *bindService* (pour les services Client/serveur).

Intent explicite: appel d'une activité

Création d'un Intent explicite : Intent(Context, Class<?>)

Passer des valeurs

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

un Intent qui décrit l'activité à démarrer : l'activité exacte (son nom) ou sa description.
Le plus souvent l'activité est connue : il suffit alors de passer son nom.

Récupérer un résultat

Appel à *startActivityForResult()*

*I*mplémenter la méthode *onActivityResult()* appelée lorsque l'activité est terminée.
Le résultat est dans l'intent passé en paramètre de onActivityResult().

Passage de paramètres intra app

Utiliser EXTRA équivalent à une MAP.
putExtra("CLE",VALEUR) : Accepte les types primitifs : Boolean, Integer, String, Float, Double, Long. Les Objets si Serializable.

```
Intent intent = new Intent(this,DetailActivity.class);
intent.putExtra("name","Florent");
intent.putExtra("age",24);
startActivity(intent);
```

Pour récupérer L'EXTRA à partir de l'activité : *getIntent().getTYPEExtras()*,
par exemple :

getIntent().getIntExtra("age",0) 0 est ici la valeur par défaut à retourner dans le cas où « age » n'ait pas été transmis lors de la création de l'activité.

```
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);

    Intent intent = getIntent();
    String name = intent.getStringExtra("name",null); //"florent
    int age = intent.getIntExtra("age",0); //24
}
```

Intent implicite: appel d'une « action »

Création d'un Intent implicite : Intent(String action, URI)
Rechercher une activité à partir de l'action

Exemple : un envoi de mail.

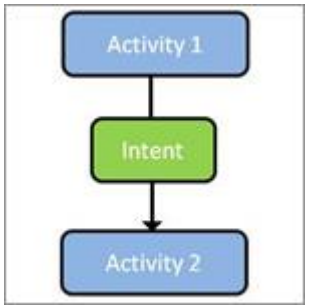
```
        Intent intent = new Intent(Intent.ACTION_SEND);
I       ntent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
        startActivity(intent);
```

recipientArray contient la liste d'adresses mails auxquelles envoyer le mail.

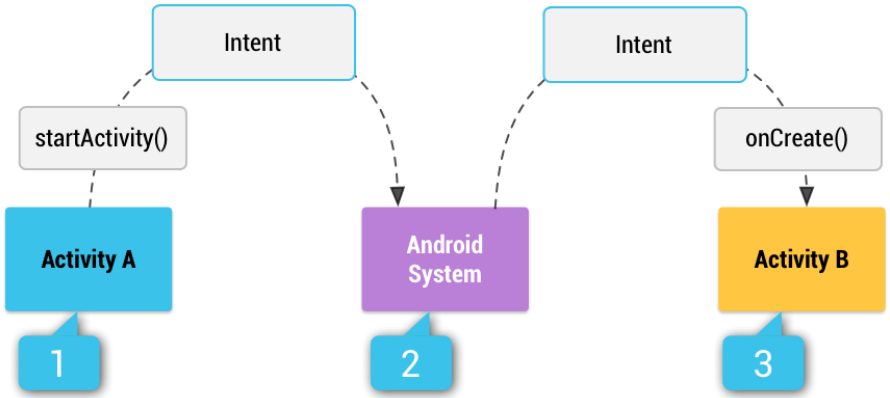
Lorqu'une application de gestion de mails reçoit cet Intent, elle remplit le champ *To* avec la liste dans le formulaire d'envoi de mails.

L'activité *email* démarre, l'utilisateur effectue l'envoi de mail et l'activité appelante est *resumed*

Propagation d'Intent



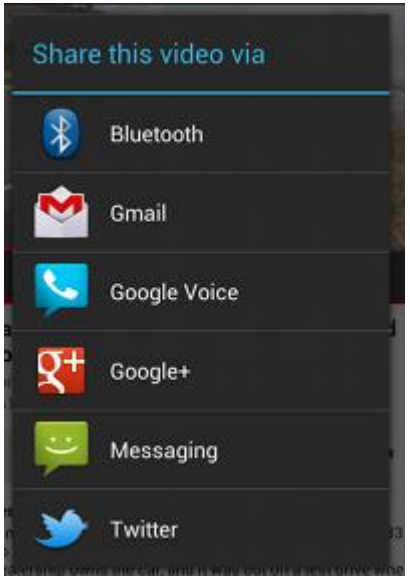
Comment se propage un intent implicite ?



Le système compare le contenu de l'Intent avec les intent-filters déclarés dans le fichier *Manifest* des autres app du device.
Si ok le système démarre l'app
Si il y en a plusieurs qui correspondent le choix est laissé à l'utilisateur.

Exemple d’appels

```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
...

// Always use string resources for UI text.
// This says something like "Share this photo with«

String title = getResources().getString(R.string.chooser_title);

// Create intent to show the chooser dialog
Intent chooser = Intent.createChooser(sendIntent, title);

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```



Récupérer un résultat

Exemple : Récupérer un contact sélectionné dans la liste des contacts par l'utilisateur

.

```
private void pickContact() {
    // Create an intent to "pick" a contact, as defined by the content provider URI
    Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
    startActivityForResult(intent, PICK_CONTACT_REQUEST);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // If the request went well (OK) and the request was PICK_CONTACT_REQUEST
    if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST) {
        // Perform a query to the contact's content provider for the contact's name
        Cursor cursor = getContentResolver().query(data.getData(),
        new String[] {Contacts.DISPLAY_NAME}, null, null, null);
        if (cursor.moveToFirst()) { // True if the cursor is not empty
            int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);
            String name = cursor.getString(columnIndex);
            // Do something with the selected contact's name...
        }
    }
```

Anatomie d’un Intent

Un Intent est constitué de:

1. D’un nom (optionnel)
2. D’une action à réaliser
3. De données sous forme d'URI (setData()) et/ou d'un type MIME (setType())
4. De paramètres optionnels (EXTRA)…

addCategory(String category) ajout de catégories
putExtra(String key,value)
setFlags(flags) permission sur les données, relation activité/BackStack

Intent en détail

Le nom du composant à démarrer (optionnel)
indispensable pour les intents explicites (les services entre autres)
Un objet *ComponentName* qui doit avoir le nom complet du composant.

Action une chaîne qui spécifie l'action générique effectuée par le composant (Activité / Service)

Ensemble d'actions définies

ACTION_VIEW : pour montrer des informations à un utilisateur comme une photo dans une galerie ou un adresse sur une carte.
ACTION_SEND : pour partager des données par exemple avec une application d'email ou sociale

Ou définir sa propre action attention à bien la nommer

```
static final String ACTION_TIMETRAVEL = "com.example.action.TIMETRAVEL";
```

ACTION_MAIN : action principale
ACTION_VIEW : visualiser une donnée
ACTION_ATTACH_DATAT: attachement de donnée
ACTION_EDIT : Edition de donnée
ACTION_PICK : Choisir un répertoire de donnée
ACTION_CHOOSER: menu de choix pour l'utilisateur

EXTRA_INTENT contient l'Intent original, EXTRA_TITLE le titre du menu

ACTION_GET_CONTENT: obtenir un contenu suivant un type MIME
ACTION_SEND: envoyé un message (EXTRA_TEXT|EXTRA_STREAM) à un destinataire non spécifié
ACTION_SEND_TO: on spécifie le destinataire dans l'URI
ACTION_INSERT: on ajoute un élement vierge dans le répertoire spécifié par l'URI
ACTION_DELETE: on supprime l'élement désigné par l'URI
ACTION_PICK_ACTIVITY: menu de sélection selon l'EXTRA_INTENT mais ne lance pas l'activité
ACTION_SEARCH: effectue une recherche
etc...

Données sous forme d'URI

Data

Un objet Uniform Resource Identifier qui référence la donnée et/ou le type MIME de la donnée. Le type est souvent déductible de l'action.
Par exemple, pour ACTION_EDIT, la donnée pourrait contenir l'URI du document à éditer.
Spécifier le type MIME aide le système à choisir le composant le plus adapté à la requête surtout si plusieurs types peuvent être déduits de l'URI.

setData() : pour affecter l'URI de la donnée.
setType() : pour affecter le type MIME

setDataAndType() : pour affecter les 2.

Category

Une chaine contenant une information sur le type de composant qui pourrait savoir répondre à cet Intent.

CATEGORY_BROWSABLE
Peut être démarrée dans un Browser Web

CATEGORY_LAUNCHER
Est le point de départ d'une app.

….
Cf API Intent

addCategory() permet d'ajouter une catégoie

Le nom, l'action, la donnée et la catégorie permettent au système Android de sélectionner le composant qu'il doit démarrer.

Autres informations

Extras

Paires Clé-Valeur pour passer des paramêtres qui ne sont pas des URI

putExtra() avec la clé et la valeur comme parametres
putExtras() qui prend un Bundle avec l'ensemble des paires

Les classes d'Intent spécifient plusieurs constantes EXTRA_*.

Par exemple, pour envoyer un email via ACTION_SEND, pour spécifier le "to" il y a EXTRA_EMAIL et pour le sujet EXTRA_SUBJECT.

Pour définir ses propres constantes
static final String EXTRA_GIGAWATTS = "com.example.EXTRA_GIGAWATTS";

Flags

Pour donner des informations sur le lancement d'une activité par rapport à la gestion de la pile

Zoom sur Intent et Activités

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
```

Dans le Manifest

Declarer comment les autres composants peuvent activer l'activité.

L'élément action donne le point d'entrée ici *main*, la catégorie spécifie que l'activité doit être listée dans le *launcher*.

Pour qu'une activité réponde à des intents implicites délivrés par d'autres applications, il faut ajouter des intents-filters dans l'activité contenant <action> et en option une <category> et/ou une <data>.

Une activité qui sait lire et éditer les images JPEG

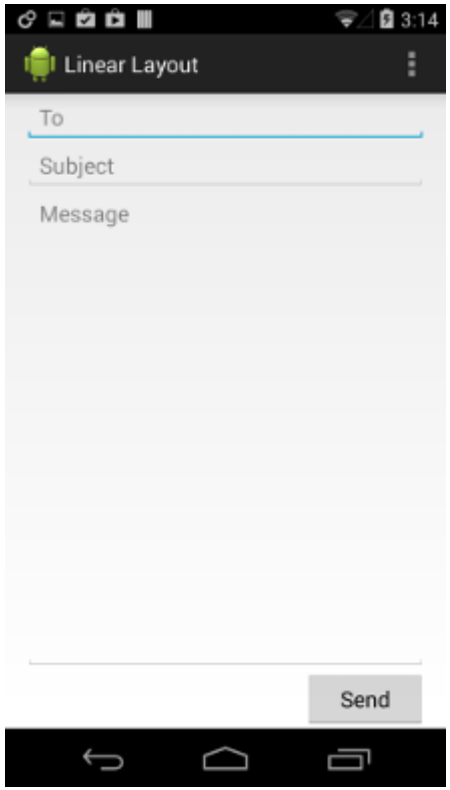
```
<intent-filter android:label="@string/jpeg_editor">
<action android:name="android.intent.action.VIEW" />
<action android:name="android.intent.action.EDIT" />
<data android:mimeType="image/jpeg" />
</intent-filter>
```

Partage de textes et de médias

```
<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name="android.intent.action.SEND_MULTIPLE"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

LAYOUT & ADAPTATEURS & VIEWHOLDER

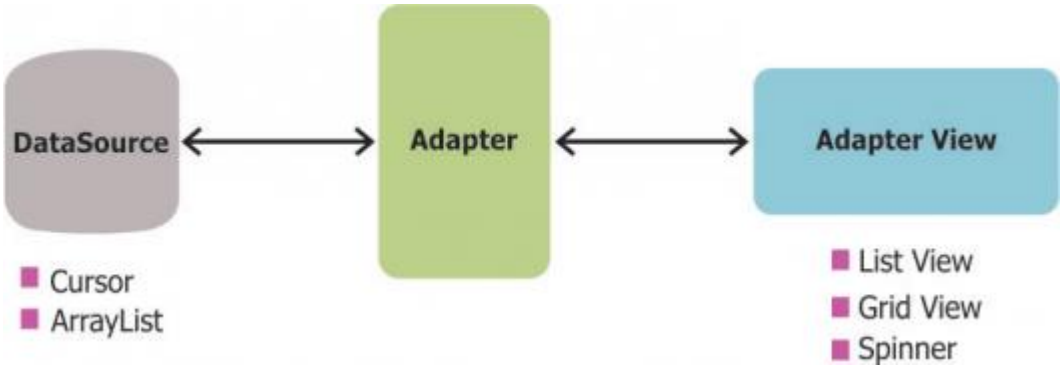
Linear Layout

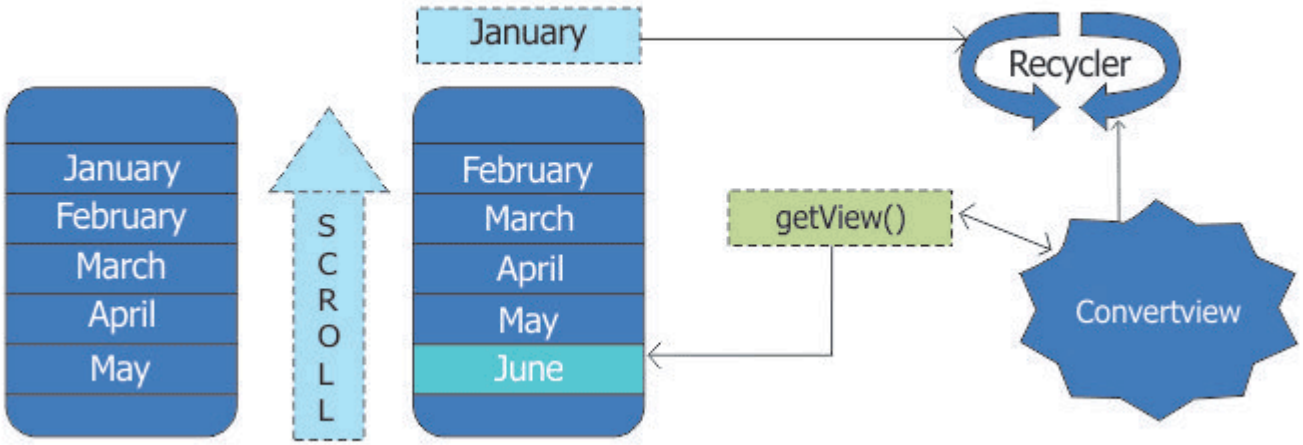


match_parent/-1 taille du parent - padding
wrap_content/-2 Taille suffisante pour le contenu + padding

```
<?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:paddingLeft="16dp"
   android:paddingRight="16dp"
   android:orientation="vertical" >
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/to" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:hint="@string/subject" />
   <EditText
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:gravity="top"
      android:hint="@string/message" />
   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:layout_gravity="right"
      android:text="@string/send" />
</LinearLayout>
```

Pourquoi des Adapters ?





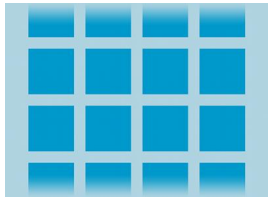
https://www.edureka.co/blog/what-are-adapters-in-android/

Adaptateurs et Layout

Pour un contenu dynamique non prédéfini,
AdapterView (par spécialisation…) permet de placer les vues dans le layout à l'éxécution.
Il faut un Adaptateur pour relier les données au layout. .

Par exemple **ListView** permet d'avoir des items scrollables. Les items sont automatiquement insérés dans la liste via un **Adaptateur** qui met les items d'un tableau ou d'une base de données.

Ce n'est pas le seul Layout qui se construit avec des Adaptateurs regardez aussi GridLayout



Un peu plus sur les adaptateurs

ArrayAdapter si source de données tableau : utilise toString() pour chaque item et met le résultat dans un TextView.

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, myStringArray)
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);

parma 1 : contexte de l'appli
param 2 : layout pour 1 item
param 3 : le tableau de chaîne

Pour faire votre propre visualisation de chaque item

- surcharger le toString
- ou spécialiser ArrayAdapter et surcharge de *getView()* pour remplacer le TextView (par exemple, par un ImageView)

Adaptateurs : suite

SimpleCursorAdapter données issues de Cursor (read-write access to the result set returned by a database query).
Implique de : spécifier un layout pour chaque ligne
et quelles colonnes insérer.
Dans notre cas, le résultat de la requête peut retourner une ligne pour chaque personne
Et 2 colonnes : une pour le nom et l'autre pour les numéros.

Créer un tableau de chaînes pour identifier les colonnes et un tableau d'entiers spécifiant chaque vue correspondant

```
String[] fromColumns =    {ContactsContract.Data.DISPLAY_NAME,
                           ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};

SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
                            R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);
ListView listView = getListView(); listView.setAdapter(adapter);
```

Implique la création d'une vue pour chaque rangée en utilisant le layout fourni et les deux tableaux.

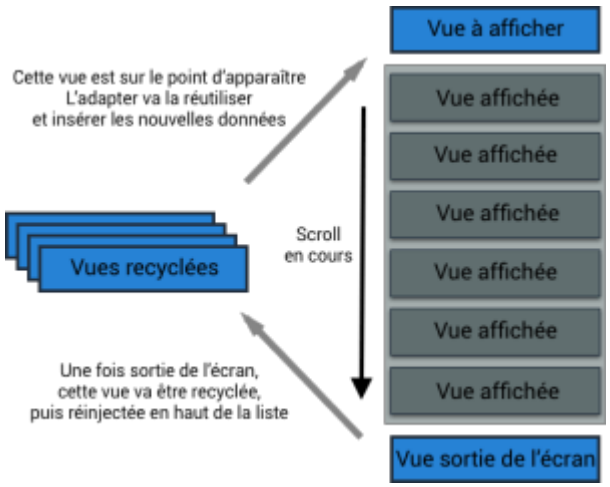
ListView layout

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

    // Create a progress bar to display while the list loads
    ProgressBar progressBar = new ProgressBar(this);
    progressBar.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT, Gravity.CENTER));
    progressBar.setIndeterminate(true);
    getListView().setEmptyView(progressBar);

    // Must add the progress bar to the root of the layout
    ViewGroup root = (ViewGroup) findViewById(android.R.id.content);
    root.addView(progressBar);

    // For the cursor adapter, specify which columns go into which views
    String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME};
    int[] toViews = {android.R.id.text1}; // The TextView in simple_list_item_1

    // Create an empty adapter we will use to display the loaded data.
    // We pass null for the cursor, then update it in onLoadFinished()
    mAdapter = new SimpleCursorAdapter(this,
            android.R.layout.simple_list_item_1, null,
            fromColumns, toViews, 0);
    setListAdapter(mAdapter);

    // Prepare the loader.  Either re-connect with an existing one,
    // or start a new one.
    getLoaderManager().initLoader(0, null, this);
}
```

ListView Layout : accès aux contacts

```
    // Called when a new Loader needs to be created
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        // Now create and return a CursorLoader that will take care of
        // creating a Cursor for the data being displayed.
        return new CursorLoader(this, ContactsContract.Data.CONTENT_URI,
                PROJECTION, SELECTION, null, null);
    }

    // Called when a previously created loader has finished loading                    " />
    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        // Swap the new cursor in.  (The framework will take care of closing the
        // old cursor once we return.)
        mAdapter.swapCursor(data);
    }

    // Called when a previously created loader is reset, making the data unavailable
    public void onLoaderReset(Loader<Cursor> loader) {
        // This is called when the last Cursor provided to onLoadFinished()
        // above is about to be closed.  We need to make sure we are no
        // longer using it.
        mAdapter.swapCursor(null);
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Do something when a list item is clicked
    }
}
```

Fonctionnement de ListView : recyclage et fluidité

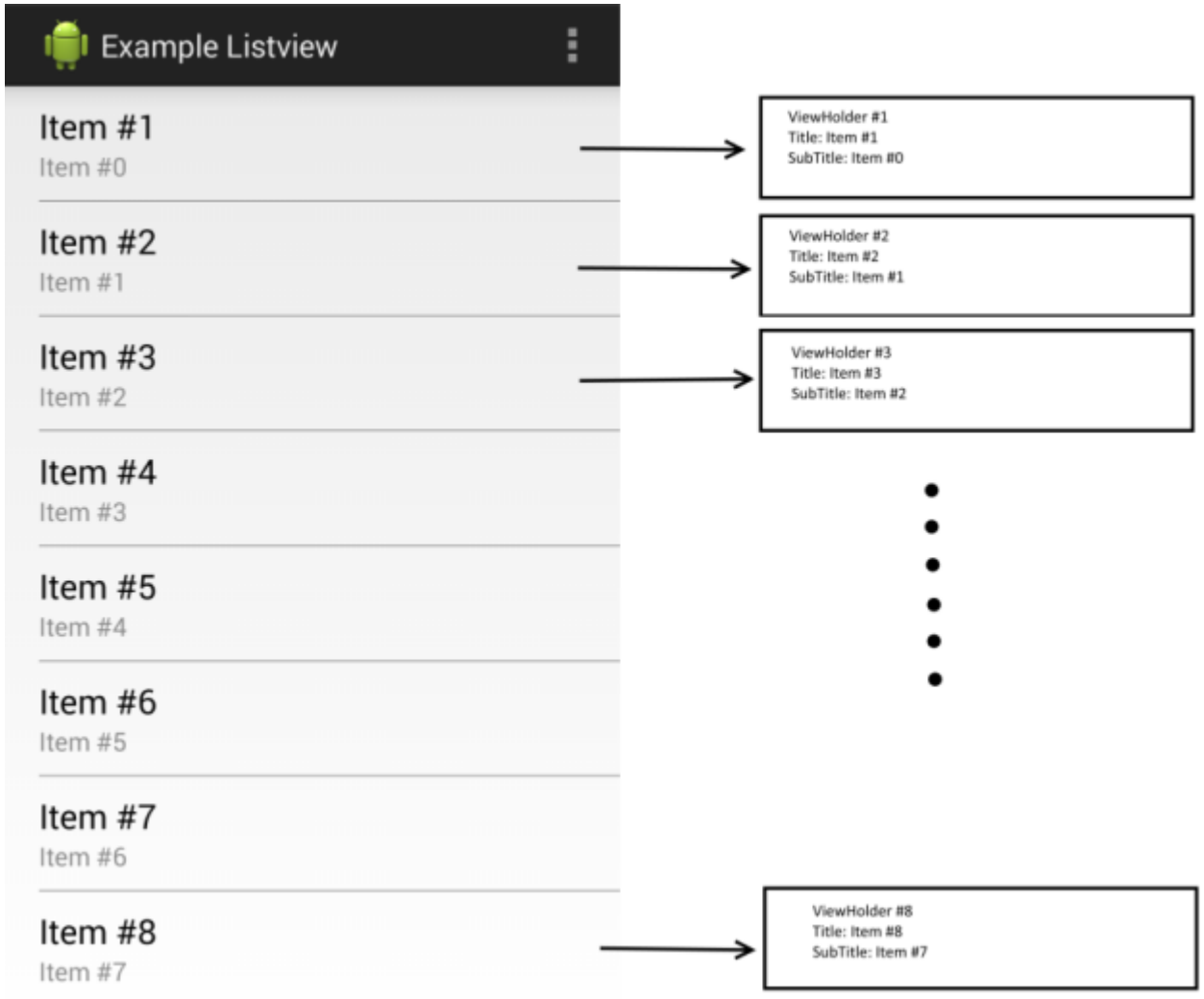


Pour réduire la consommation en mémoire la ListView stocke seulement les vues qu'elle a la capacité d'afficher,
Lorsqu'une vue sort de l'écran (scroll) elle est réutilisée pour la nouvelle vue à apparaître.

Afin d'éviter d'appeler les méthodes findViewById à chaque réutilisation des vues, Android a rajouté un concept, le ViewHolder (gardien/protecteur de vue) : mini controleur, associé a chaque cellule, et qui va stocker les références vers les sous vues.
C'est une propriété de la vue (dans l'attribut tag) : une vue n'a qu'un seul ViewHolder, et inversement.

http://tutos-android-france.com/listview-afficher-une-liste-delements/

Pourquoi des ViewHolder



Exemple de ListView

```
class TweetViewHolder{
    public TextView pseudo;
    public TextView text;
    public ImageView avatar;
}
View cellule = ...;
TweetViewHolder viewHolder = (TweetViewHolder) cellule.getTag();
if(viewHolder == null){
viewHolder = new TweetViewHolder();

    //récupérer nos sous vues
    viewHolder.pseudo = (TextView)  cellule.findViewById(R.id.pseudo);
    viewHolder.text   = (TextView)  cellule.findViewById(R.id.text);
    viewHolder.avatar = (ImageView) cellule.findViewById(R.id.avatar);

    //puis on sauvegarde le mini-controlleur dans la vue
    cellule.setTag(viewHolder);
}
```

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
 if(convertView == null){
       //Nous récupérons notre row_tweet via un LayoutInflater,
       //qui va charger un layout xml dans un objet View
       convertView = LayoutInflater.from(getContext()).inflate(R.layout.row_tweet,parent, false);
   }

   TweetViewHolder viewHolder = (TweetViewHolder) convertView.getTag();
   if(viewHolder == null){
       viewHolder = new TweetViewHolder();
       viewHolder.pseudo = (TextView) convertView.findViewById(R.id.pseudo);
       viewHolder.text = (TextView) convertView.findViewById(R.id.text);
       viewHolder.avatar = (ImageView) convertView.findViewById(R.id.avatar);
       convertView.setTag(viewHolder);
   }

   //nous renvoyons notre vue à l'adapter, afin qu'il l'affiche
   //et qu'il puisse la mettre à recycler lorsqu'elle sera sortie de l'écran
   return convertView;
}....
```

Remplir les cellules avec les données d’un Tweet

Nouveau : RecyclerView

Besoin : visualiser la collection de musiques d'un utilisateur
Chaque View holder pourrait représenter un album et pourrait contenir :
le titre, le nom de l'artiste et pourrait permettre en cliquant de faire jouer la musique 'demarrage, stop)

Le RecyclerView crée autant de view holders que de positions visibles sur l'écran + 1

Au scroll le RecyclerView reaffecte correctement les vues.

Par exemple si on peut afficher 10 albums le RecyclerView crée et lie les view holders de laposition 0 à 9 et aussi celle en position 10

http://feanorin.developpez.com/tutoriels/android/composant-graphique-recyclerview/

Recycler View

RecyclerView fonctionne en étendant *la classe abstraite* RecyclerView.ViewHolder.

The RecyclerView utilise le *layout manager* fourni pour organiser les items gérés par des instances de sous classes de RecyclerViewHolder
Chaque view holder se charge d'afficher un item dans sa propre vue.

Le conteneur de l'interface dynamique est un objet RecyclerView à ajouter au layout de l'activité ou du fragment hôte.

Les view holders sont gérés par un adapter qui doit étendre la classe abstraite RecyclerView.Adapter .
L'adaptateur crée les view holders au besoin : les relie à leurs données respectives et les affecte à leur position en appelant la méthode onBindViewHolder()..

https://developer.android.com/guide/topics/ui/layout/recyclerview.html

Recycler View : Optimisations

Au scroll, le RecyclerView crée autant de nouveaux view holders que nécessaire.

Il conserve ceux qui ont été scrollés afin de les réutiliser en cas de scroll arrière

Sion poursuit le scroll descendant les plus anciens View Holders peuvent être réaffectés à dde nouvelles données

Il n'y a ni recréation ni expansion (inflated) juste une mise à jour des liaisons avec les items via la méthode RecyclerView.Adapter.notify…() appelée lorsqu'il y a changement d'items affichés.

https://developer.android.com/guide/topics/ui/layout/recyclerview.html