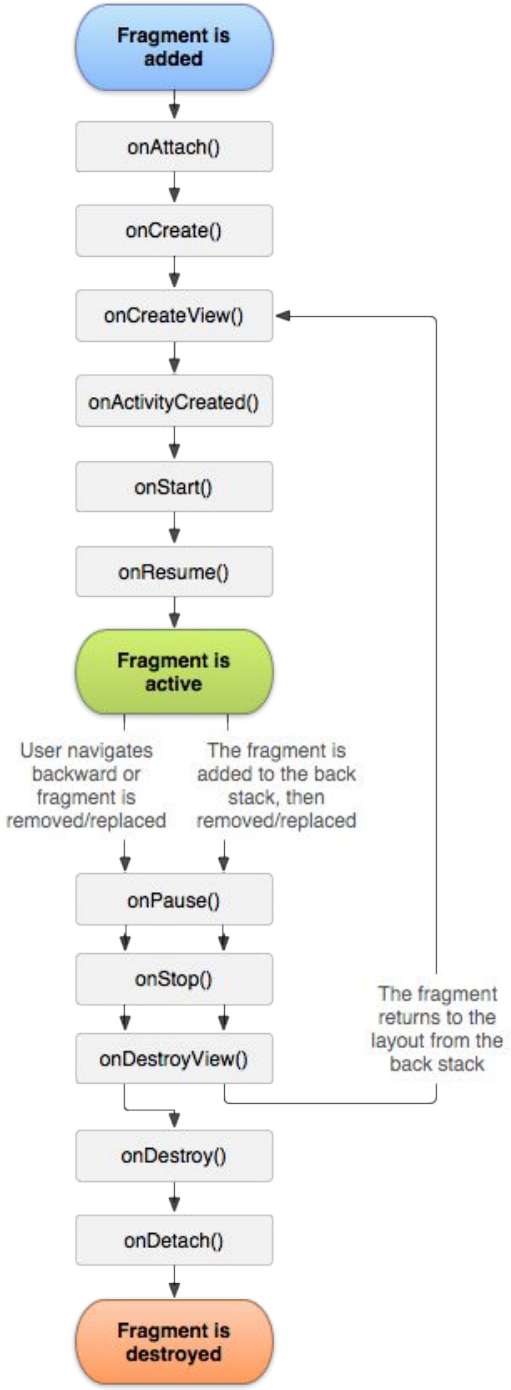
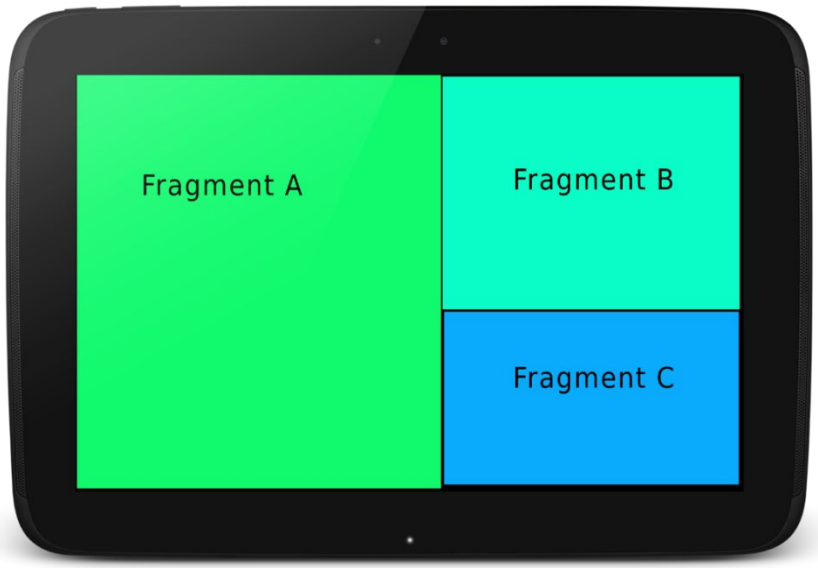
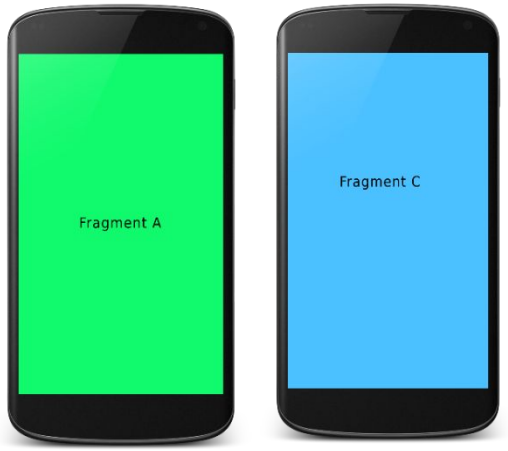
Android


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

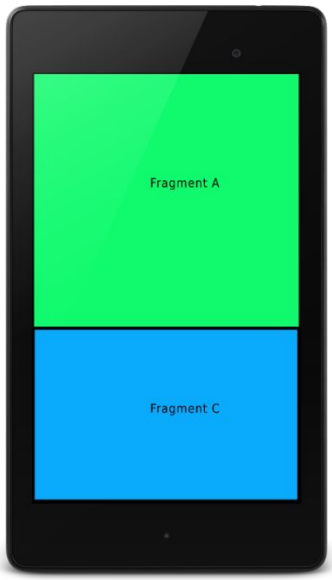
PLUS SUR LES FRAGMENTS




Fragment A
Fragment B
Fragment C


Fragment A
Fragment C


Fragment B


Fragment A
Fragment C


Fragment A
Fragment B

Fragment et cycle de vie ?

Sous classe de Fragment (ou d'une de ses sous classes).
Comme pour les activités, il est possible d'implémenter : *onCreate(), onStart(), onPause()* et *onStop()*.

Si une activité devient un fragment, copier directement le code de ces méthodes.

Il faut au minimum implémenter les méthodes :
onCreate() : appelée à la création du fragment : initialiser les principaux composants.
onCreateView() : appelée la première fois que l'interface du fragment est dessinée : renvoyer la vue racine du fragment (null si ce n'est pas un fragment d'IHM)
onPause() : appelée lorsque l'utilisateur quitte le fragment : gérer la persistance si nécessaire

onResume, *onPause* et *onStop* : même principe que pour les activités

Coordination avec le cycle de vie d'une activité

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

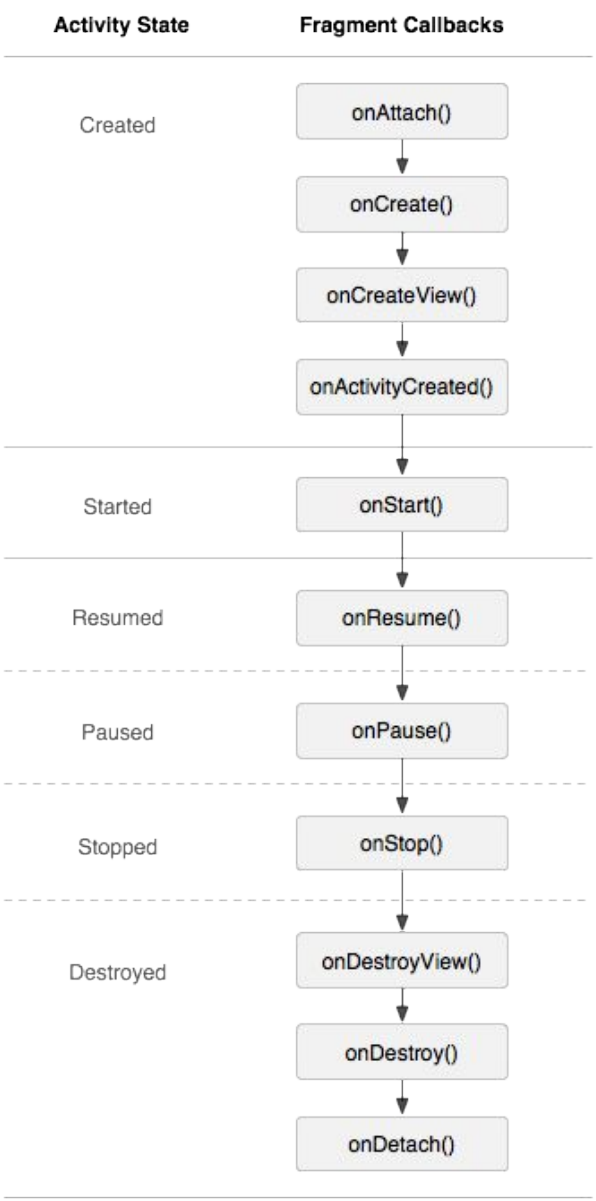
onCreateView() : appelée pour créer la vue associée au fragment

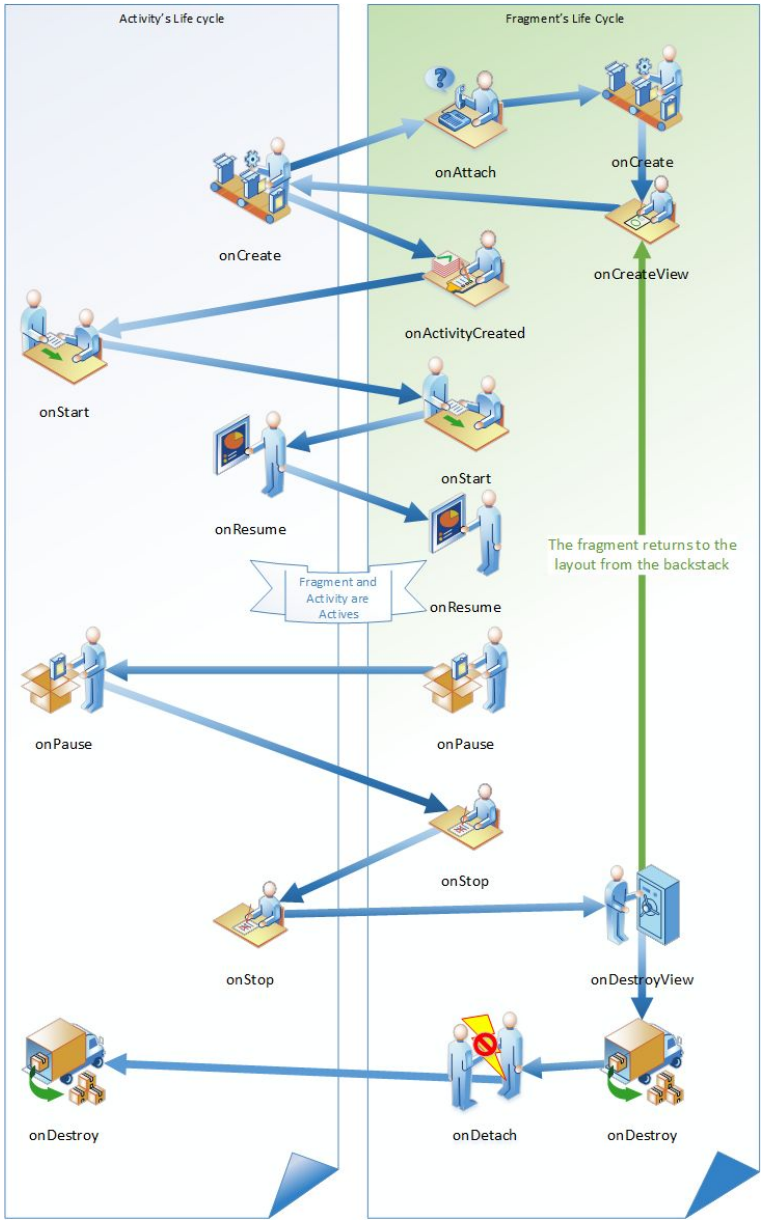
onActivityCreated() : lorsque l'activité est créée

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

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

Le container passé en paramètre correspond à l'endroit où placer le fragment dans l'activité qui va l'accueillir,
le Bundle fournit les données stockées si c'est le cas.
La méthode inflate prend l'ID du layout à étendre, le ViewGroup où intégrer, un booléen à true ou false si l'intégration doit être faite ou pas.

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>
```

?

Création par programmation

Dynamique vs statique

?

Les interaction avec les fragments se font via FragmentManager

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction =
fragmentManager.beginTransaction();
```

Une transaction correspond à un ensemble de modifications à faire au même moment.

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container,
fragment);
```

Les méthodes à utiliser peuvent être : *add(), remove()* et *replace().*

```
fragmentTransaction.commit();
```

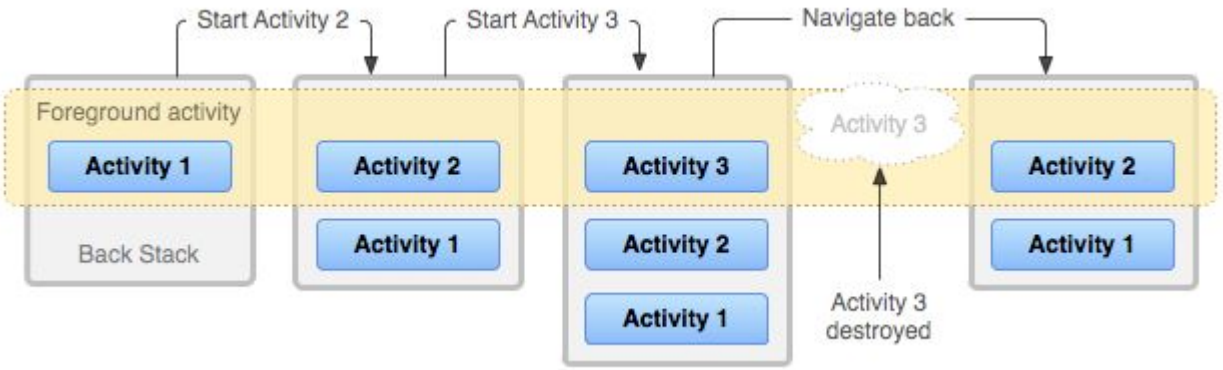
Le commit() rend la transaction effective

.

Conserver l'état : gérer le retour arrière

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

Gestion de pile différente des activités : Activité liée à une pile gérée par le système ou par le bouton back

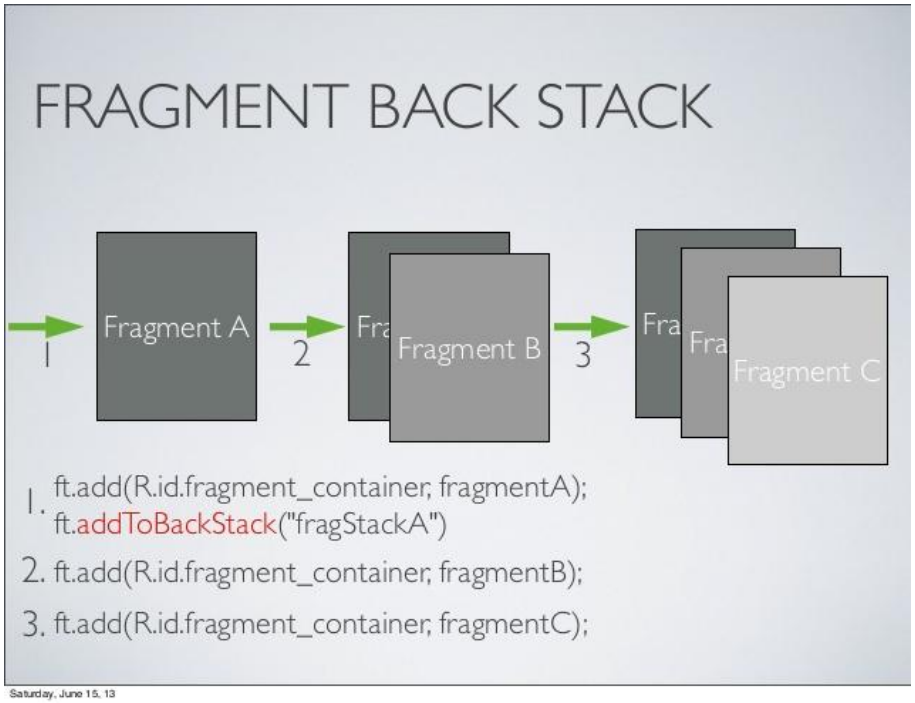


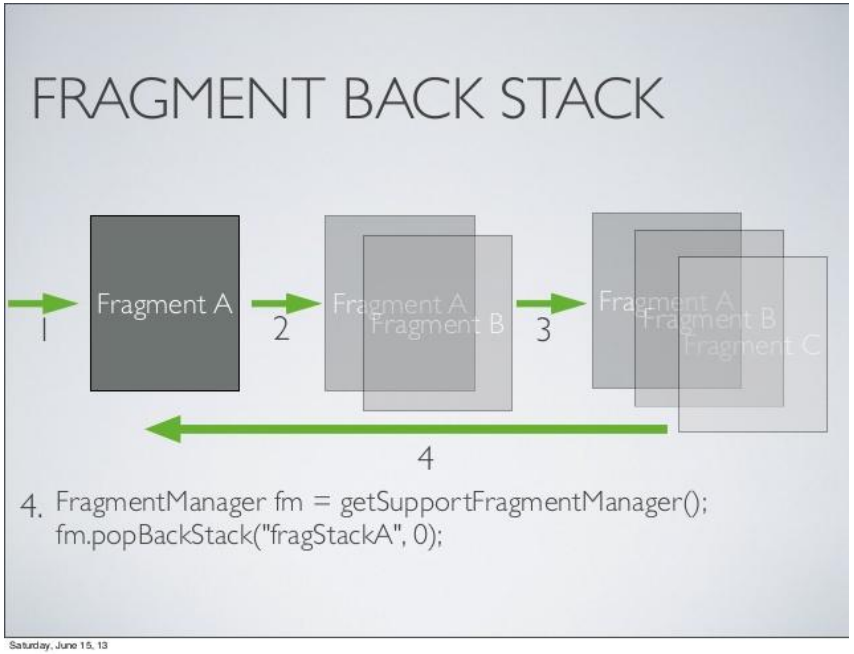
Fragment lié à une pile associée à l'activité Hote seulement avec addToBackstack
La transaction dans laquelle les fragments sont modifiés peuvent être placées dans la pile interne de l'activité et sont donc dépilées avant la fin de l'activité

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.

Un fragment peut accéder à l'instance de l'activité avec *getActivity*() pour récupérer un élément.

Par exemple

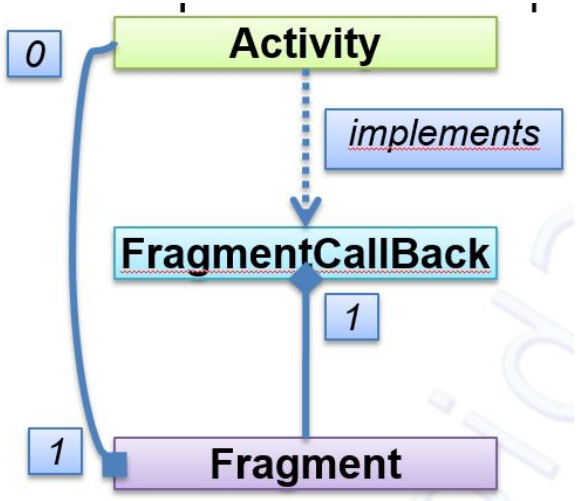
```
View listView = getActivity().findViewById(R.id.list);
```

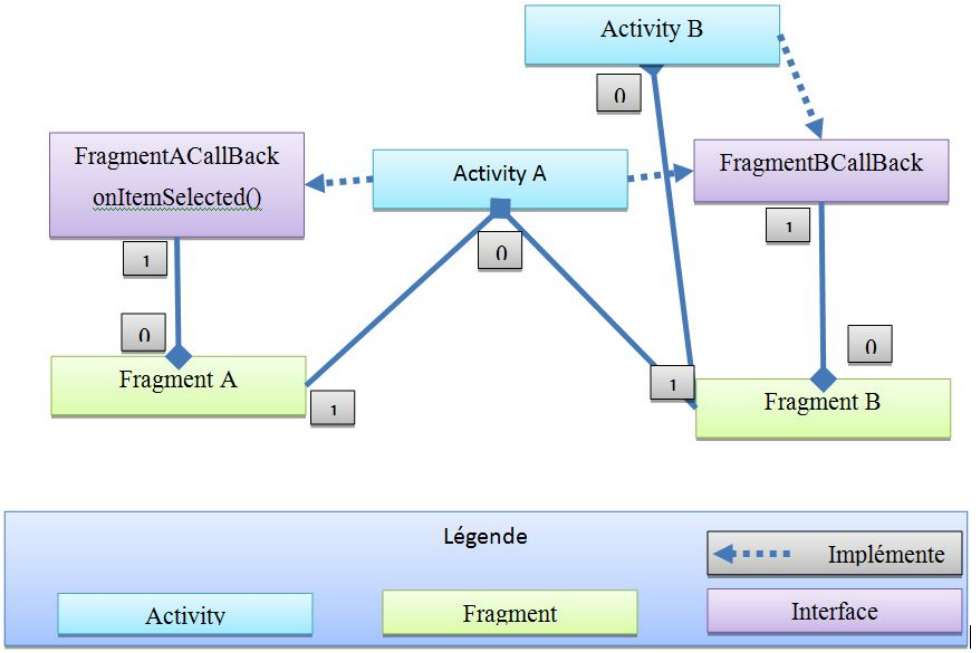
L'activité peut appeler des méthodes de ses fragments via FragmentManager, en utilisant *findFragmentById*() or *findFragmentByTag*().

Par exemple

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

Conseils de communication





INTENTS

Pourquoi des Intents ?

Séparation forte entre chaque application : 1 application / 1 processus minimum
mais nécessité de communiquer entre applications et activités

Solution : envoi de messages synchrones
Intra application : facile, même espace mémoire
Inter-applications : IPC spécifique Android (AIDL)

Bienvenue dans le monde du réseau !

Objets messages pour faciliter la communication entre composants.

Pour démarrer une activité
Pour démarrer un service
Pour délivrer un broadcast

Comment ?

Pour démarrer une **activité** :

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

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).

Pour délivrer un **broadcast** : un broadcast est un message que n'importe quelle application peut recevoir. Par exemple les messages systèmes sont envoyés lorsque le systeme reboote.

En passant un Intent à *sendBroadcast()*, *sendOrderedBroadcast(),* or *sendStickyBroadcast*()

C’est quoi au juste ?

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)…

Création d'un Intent :
Intent(Context, Class<?>) appel explicite ou Intent(String action, URI) appel implicite

?

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)
L'action détermine les autres informations.
Il est possible de spécifier ses propres actions en interne à une app mais il est préférable d'utiliser les actions définies pour être utilisé par d'autres.
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
Pour 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 élément vierge dans le répertoire spécifié par l'URI
ACTION_DELETE: on supprime l'élément 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 chaîne 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 paramètres qui ne sont pas des URI

putExtra() avec la clé et la valeur comme paramètres
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

Types d'Intent

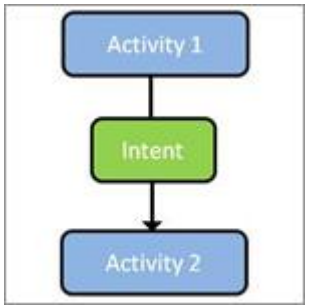
Intents explicites : interne à une app avec le nom complet de l'activité ou du service à appeler.
Par exemple pour démarrer une activité en fonction d'une action de l'utilisateur.

Intents implicites en déclarant une action générale qui est supportée par une autre app.
Par exemple pour montrer un emplacement sur une carte

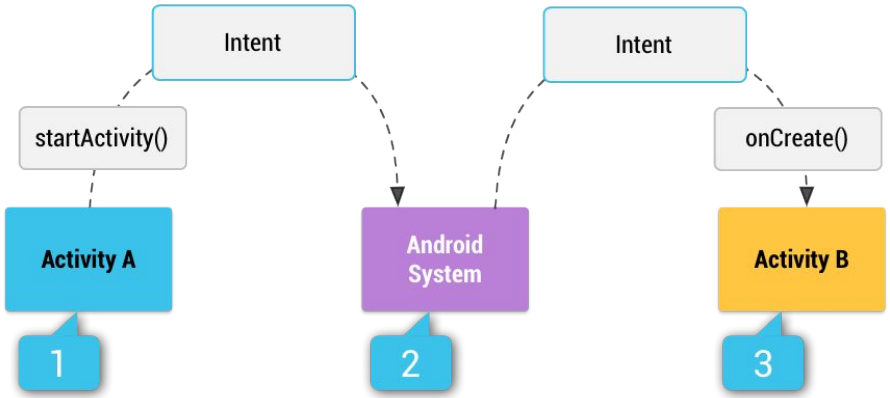
```
<activity android:name="ShareActivity">
   <intent-filter>
      <action android:name="android.intent.action.SEND"/>
      <category android:name="android.intent.category.DEFAULT"/>
      <data android:mimeType="text/plain"/>
   </intent-filter>
</activity>
```

Pas conseillé pour les services à cause de problèmes de sécurité car sans retour utilisateur.

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.

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

Déclarer 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>.

C'est une description du type d'activité recherchée.

Exemple de déclaration

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>
```

Appeler une autre activité

startActivity() avec pour paramètre
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.

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

Pour appeler une activité à partir de l'action, par exemple, un envoi de mail.

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

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

Recevoir 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().

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...
        }
    }
}
```

onActivityResult() method in order to handle an activity result.
The first condition checks whether the request was successful—if it was, then the resultCode will be RESULT_OK—and whether the request to which this result is responding is known—

in this case, the requestCode matches the second parameter sent with startActivityForResult().

From there, the code handles the activity result by querying the data returned in an Intent (the data parameter).

What happens is, a ContentResolver performs a query against a content provider, which returns a Cursor that allows the queried data to be read. For more information, see the Content Providers document.

Passage de parametres 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
}
```

Exemple d’appels

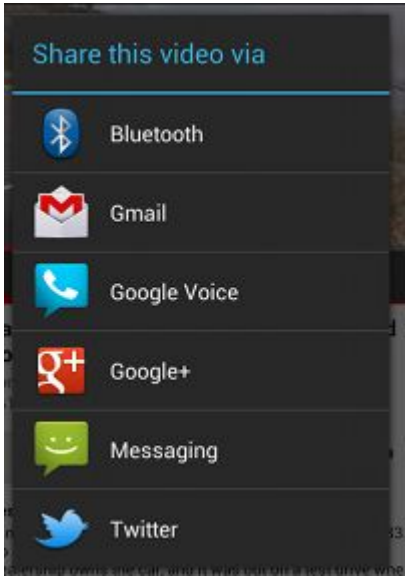
```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
...

// 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);
}
```



Autres exemples d’appels

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"

Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl))
startService(downloadIntent);
```

```
// Create the text message with a string

Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Aperçu sur les AsyncTask et les services

Même problématique : lancer des tâches en tâche de fond pour faire des opérations longues et non IHM.
Cette prise en charge dans un Thread indépendant permet de ne pas affecter la qualité de l'interface (temps de réponse).

Lors du développement d'une application, il faut bien avoir en tête que toutes les tâches consommatrices de ressources (requêtes http, calculs lourds, …) doivent se faire dans un Thread séparé. En effet, le système affiche un message d'erreur et ferme l'application lorsque le Thread principal (appelé UI Thread) est bloqué trop longtemps.

Fonctionnement en asynchrone

Problème de priorité…

AsyncTasK

Pour réaliser des tâches de manière asynchrone, à la manière de la classe Thread de java.

Simple d'utilisation et d'implémentation
Le Thread secondaire est créé automatiquement et la communication entre les Thread est simplifiée.

Une Tâche Asynchrone hérite du la classe AsyncTask

Les trois paramètres attendus lors de la déclaration sont des types génériques :

- Le premier est le type des paramètres fournis à la tâche
- Le second est le type de données transmises durant la progression du traitement
- Enfin le troisième est le type du résultat de la tâche

http://www.tutos-android.com/asynctask-android-traitement-asynchrone-background

Fonctionnement

Une AsyncTask doit obligatoirement implémenter la méthode *doInBackground.* C'est elle qui réalisera le traitement de manière asynchrone dans un Thread séparé.

Les méthodes *onPreExecute* (appelée avant le traitement), *onProgressUpdate* (appelée pour afficher sa progression) et *onPostExecute* (appelée après le traitement) sont optionnelles.

Un appel à la méthode *publishProgress* permet la mise à jour de la progression. On ne doit pas appeler la méthode *onProgressUpdate* directement.

Les trois méthodes (onPreExecute, onProgressUpdate et onPostExecute) s'exécutent depuis l'UI Thread ! C'est d'ailleurs grâce à cela qu'elles peuvent modifier l'interface. On ne doit donc pas y effectuer de traitements lourds.

Service vs AsyncTask

La classe **IntentService** permet d'exécuter une opération dans un thread en background.
Un *IntentService* continue de s'exécuter alors qu'une AsyncTask est interrompue parce qu'il n'est pas lié à une Activité de la même façon

MAIS un IntentService
ne peut pas interagir directement avec l'interface utilisateur (exemple de la progressbar)
Il faut explicitement envoyer le résultat à l'activité

Les demandes de travaux sont séquentiels : une opération doit être terminée avant d'en commencer une autre.
On ne peut pas interrompre l'opération