Android

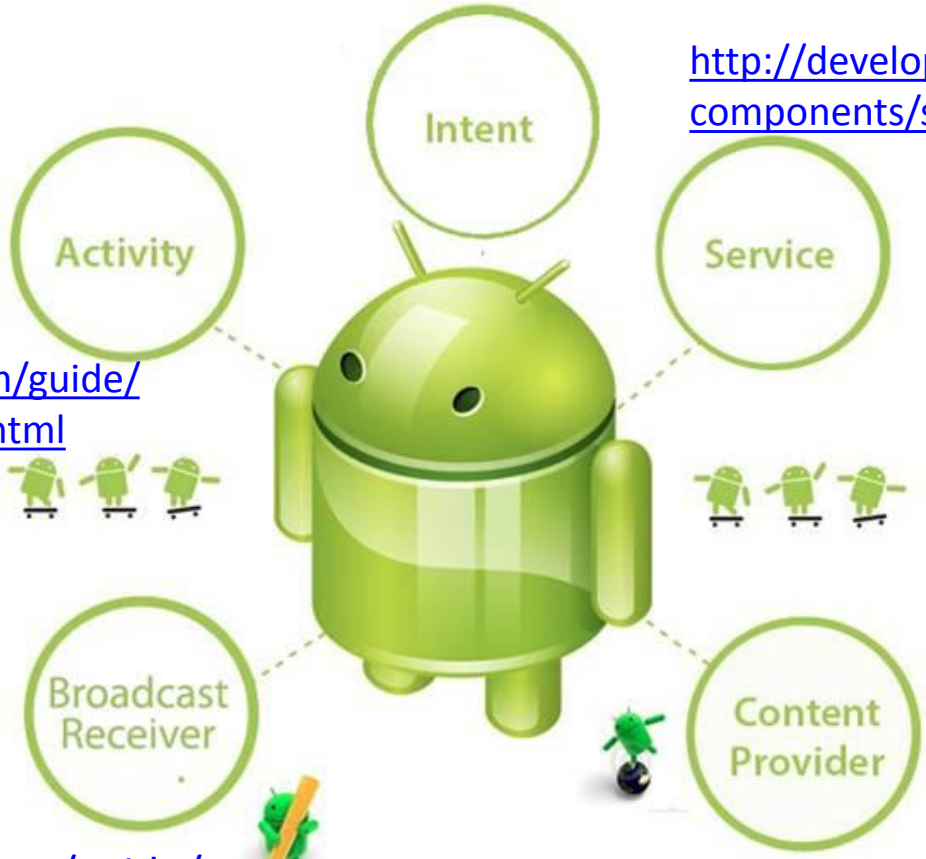

Cupcake
Donut
Eclair
Froyo
Gingerbread
Honeycomb
ICE Cream-Sandwitch
Jelly Bean
Kitkat
lollipop
Lollipop
Marshmallow
Nougat

/

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

Autres notions utiles

Plus d’informations sur le Material Design

L’appel à des services

L’accès aux données via des bases SQL

MATERIAL DESIGN

Material Design : c'est quoi ?

Facilité d'utilisation : simples, épurées et intuitives - « user friendly ».

Objectif : **design contemporain** càd efficace sur tout support, auto-adaptatif, interactif et intuitif **pour tous les utilisateurs**.

C*onception matérielle*, **basé sur le papier et l'encre**, vise à unifier avec un **design unique et homogène** l'ensemble des services et matériels de Google.

« Contrairement au vrai papier, le matériau digital peut s'étirer et se modifier de manière intelligente. Le matériau contextuel a une surface physique et des bords. Les superpositions et les ombres donnent des informations sur ce que vous pouvez toucher ». Matias Duarte, Designer et concepteur d'interface Google.

.

Conférence Google I/O du 25 juin 2014

Comment ?

Règles s'appliquant à l'interface graphique utilisées depuis la version 5,0

Une **palette** et une **typographie** proche de l'écriture : **encre et papier**.
Roboto et Noto Sans disponibles en téléchargement sur Google Font : optimisées pour le web pour une bonne lisibilité.

Agencement de l'espace : feuille et encre où la « feuille » peut s'adapter comme on le souhaite.

Animations pour capter l'attention de l'utilisateur *(micro-interactions)*.

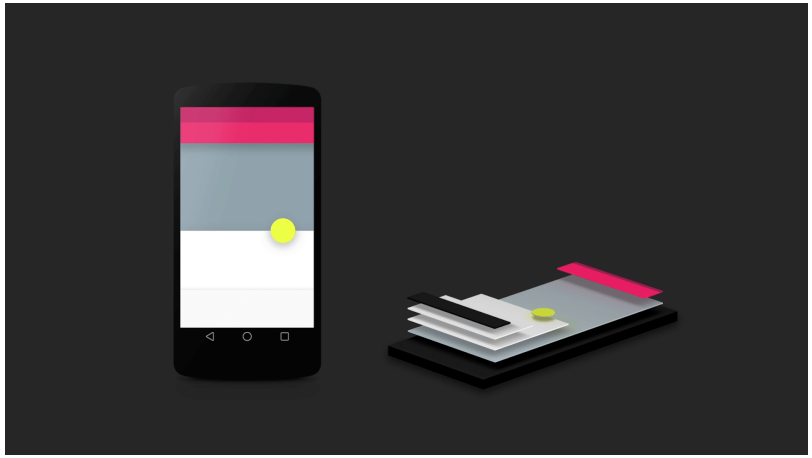
Grilles, typographie, couleurs et pictographie orientent l'utilisateur, visuellement guidé.

Le mouvement est le moteur des actions, il part du point d'interaction sans rupture de la continuité de l'expérience utilisateur (ux design). Il donne du sens, assure la cohérence et la continuité, tout en donnant des informations subconscientes sur les objets et leurs transformations.

Material vs Flat design

Le Material Design

- Répond à une action de l'utilisateur par un mouvement
- Unit le monde réel au monde digital,
- s'appuie sur le réalisme et la physique.

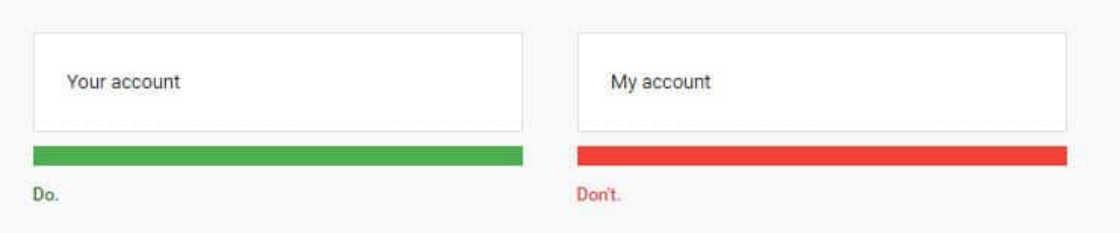


Le Flat Design

- Aplats de couleurs
- Formes géométriques *(boutons, icônes…)*
- Polices « sans-sérifs »
- Utiliser un minimum de 6 couleurs vives
- Pour les puristes, le design ne doit pas contenir d'ombre portée ou de texture.

Exemples et références

https://www.anthedesign.fr/autour-du-web/logos-apple-google//



Typographies
http://www.dafont.com/fr/roboto.font
https://www.google.com/get/noto/#sans-lgc
https://www.fontsquirrel.com/fonts/list/classification/serif

Micro animations
http://www.testapic.com/informations-pratiques/actualites/best-practices/7-secrets-pour-ameliorer-son-ux-avec-des-micro-interactions/

https://www.anthedesign.fr/webdesign-2/material-design-google-flat-design/
https://material.io/guidelines/material-design/introduction.html

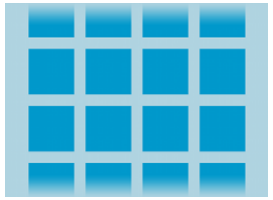
LAYOUT & ADAPTATEURS & VIEWHOLDER

Adaptateurs et Layout

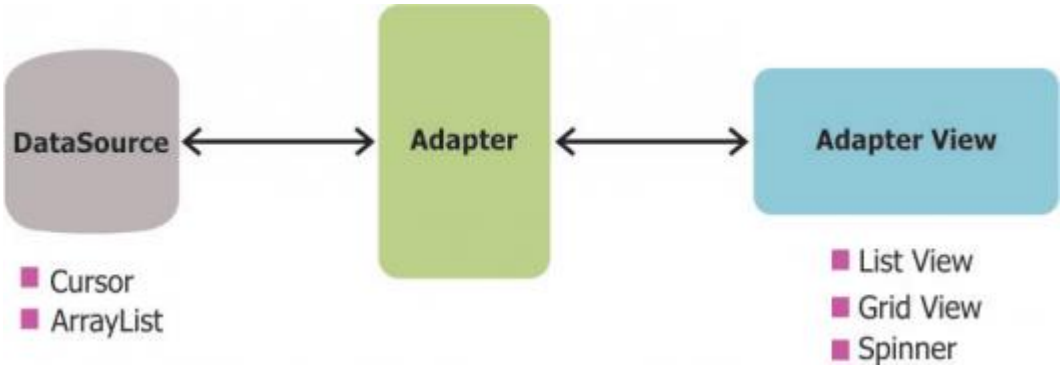
Pour un contenu dynamique non prédéfini,
AdapterView (par spécialisation…) permet de placer les vues dans le layout à l'éxécution.
Il faut un Adaptateur pour relier les données au layout. .

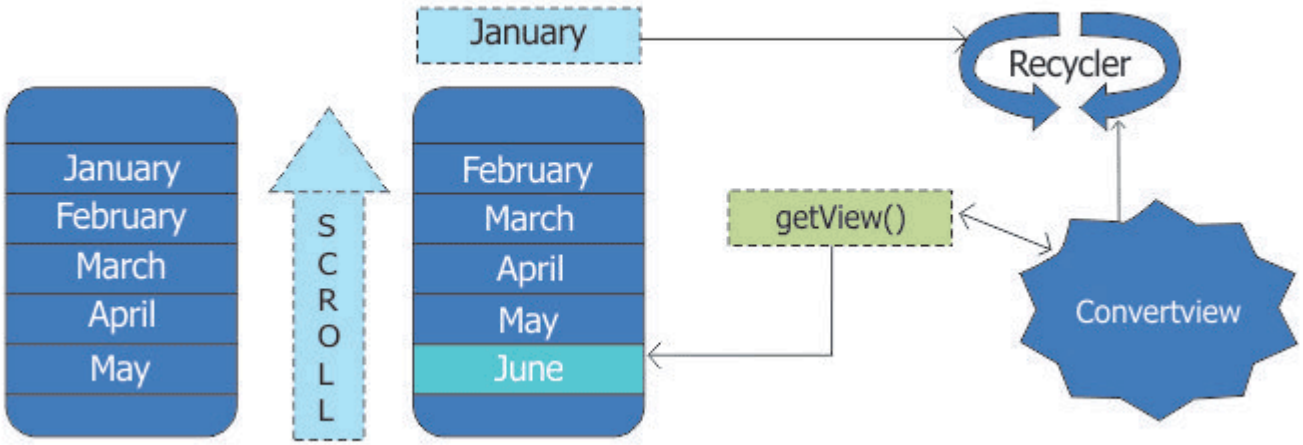
Par exemple **ListView** permet d'avoir des items scrollables. Les items sont automatiquement insérés dans la liste via un **Adaptateur** qui met les items d'un tableau ou d'une base de données.

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



Pourquoi des Adapters ?





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

Un peu plus sur les adaptateurs

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

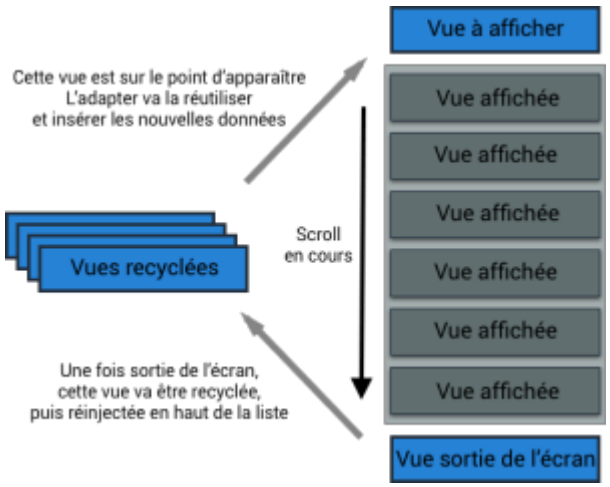
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, myStringArray)
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);

param 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)

Fonctionnement de ListView : recyclage et fluidité

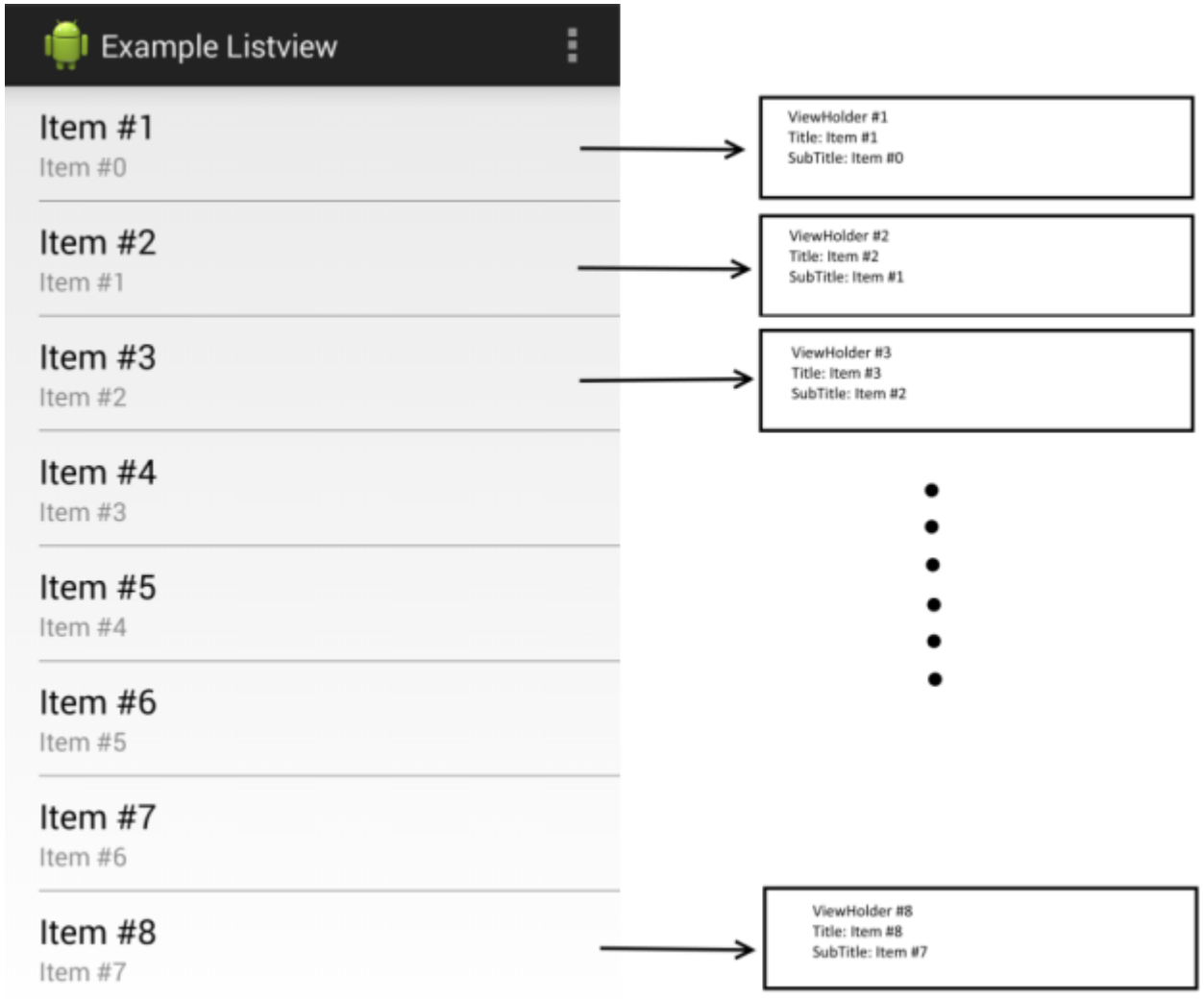


Pour réduire la consommation en mémoire la ListView stocke seulement les vues qu'elle a la capacité d'afficher,
Lorsqu'une vue sort de l'écran (scroll) elle est réutilisée pour la nouvelle vue à apparaître.

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);
}
```

Nouveau : RecyclerView

Motivations : grande quantité d'information, besoin de fluidité

Ex: 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 la position 0 à 9 et aussi celle en position 10

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

Quid des RecyclerView

Sans

Créer une classe **Holder** dans l'Adaptateur afin de garder en mémoire un pointeur sur chaque vue pour ne pas reparcourir l'arbre des vues pour chaque cellule.

Avec

RecyclerView oblige à utiliser ce même mécanisme directement dans L'Adapter.

RecyclerView, Recycler.Adapter<T> avec comme template, une classe qui hérite de **RecyclerView.ViewHolder** :

Recycler View

RecyclerView doit étendre *la classe abstraite* RecyclerView.ViewHolder

Il utilise le *layout manager* fourni pour organiser les items gérés par des instances de sous classes de RecyclerViewHolder
Chaque ViewHolder 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 ViewHolder 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,
création d'autant de nouveaux view holders que nécessaire

+

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

Si on poursuit le scroll descendant les plus anciens ViewHolders peuvent être réaffectés à de nouvelles données.

Il y a seulement 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

Encore plus de fluidité

https://developer.android.com/training/improving-layouts/smooth-scrolling.html

Objectif : l'application principale doit seulement gérer l'UI sans prendre en compte des processus lourds : accès au disque, au réseau, à une base de données.

Bonne pratique : utiliser un processus en background.
Les AsyncTask sont un moyen simple qui évitent la gestion du processus et permettent d'agir sur l'UI.

On peut définir également des services pour paralléliser les taches

SERVICES

Un service : c’est quoi ?

Un **service** est nécessaire lorsque votre application souhaite effectuer des opérations ou des calculs en dehors de l’interaction utilisateur.

un **service** ne dispose pas d’interface graphique.

Il existe deux types de services :

LocalService : Services qui s’exécutent dans **le même processus** que votre application.

RemoteService : Ils s’exécutent dans **des processus indépendants** de votre application.

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

Les 2 types de service

- Un service qui prend en charge une opération ne renvoiyant pas de résultat au composant appelant.
  - Le service est démarré quand un composant d'une application (telle qu'une activité) appelle **startService().**
  - Il s'exécute en background indépendamment du composant d'appel : même si l'activité est détruite
  - Lorsque l'opération est terminée le service se stoppe.

- Un service vu comme un serveur : une communication Client Serveur (IPC).
  - Le service est lié (bound) si il y a un appel à **bindService()**
  - Il offre une interface qui permet aux composants d'interagir avec le service en envoyant des requêtes et recevoir des résultats entre processus
  - Ne s'exécute que tant qu'il existe une application qui lui est liée (appel au destroyed).

Cycle de vie d'un service

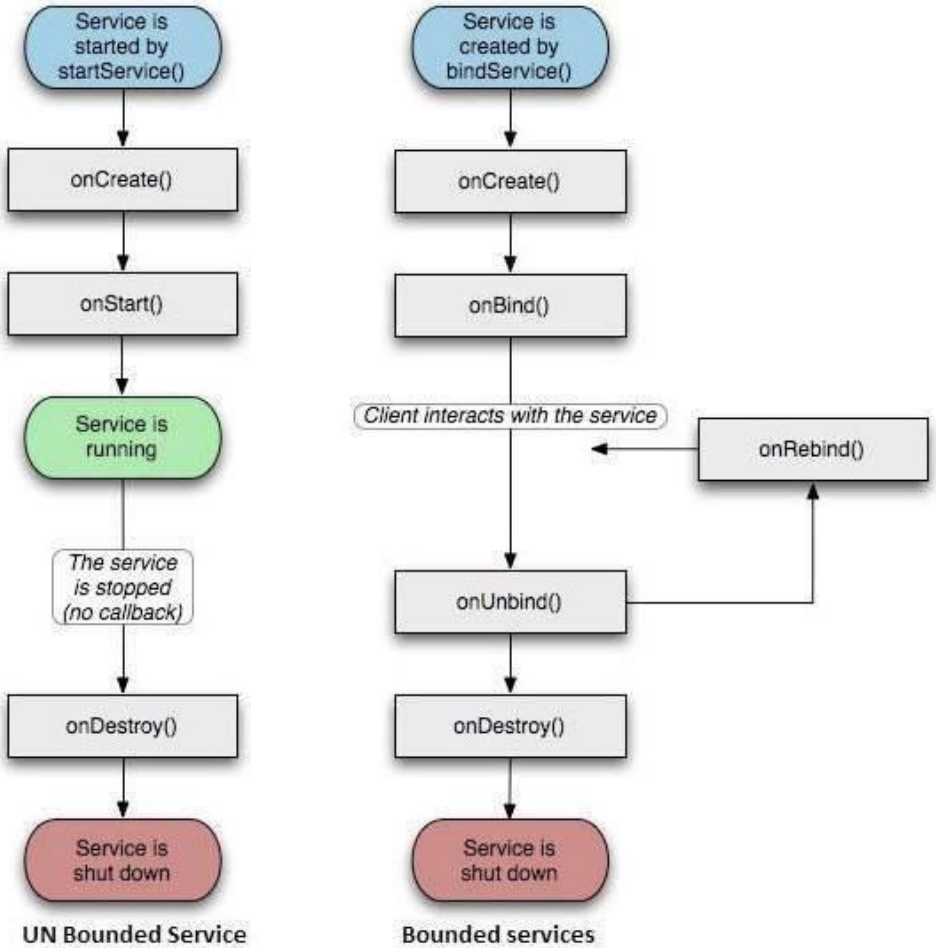
Un **service** n'a pas de durée définie , il est là pour exécuter sa tâche et il fonctionnera tant que c'est nécessaire.

onCreate() : méthode appelée à la création du service et est en général utilisée pour initialiser ce qui est nécessaire au service.

onStartCommand() : lorsque le service démarre (nécessite de gérer le stop…).

onDestroy() : appelée à la fermeture du **service**.

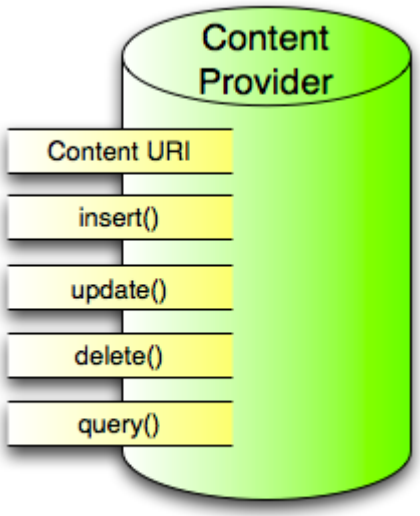
onBind() : appelée quand un composant se connecte à un service



https://www.tutorialspoint.com/android/android_services.htm

ACCES AUX DONNEES

Content Provider



Stocker et récupérer des données

Partager des données entre différentes applications.

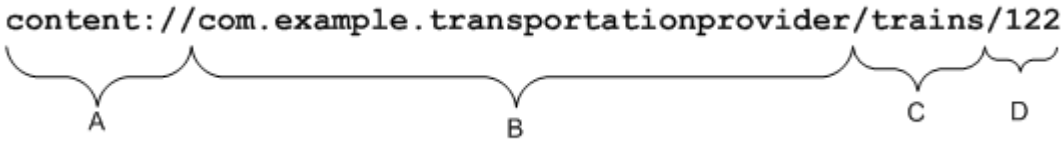
Utiliser un **contentProvider** :
- Uri
- Méthodes (Insert, Update, Delete, Query).

*Android propose plusieurs **ContentProviders** basiques (audio, vidéo, images, informations sur les contacts du téléphone…).*

http://www.tutos-android.com/contentprovider-android

e

Accès aux données



•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.

•D : L'id de la donnée qu'on souhaite récupérer. (optionnel)

Comment procéder ?

Mettre en place un système pour stocker vos données (les contents providers utilisent généralement le **SQLite**, la classe ***SQLiteOpenHelper*** vous facilite la création de votre base).

- Créer un **ContentProvider** : étendre la classe ***ContentProvider***.
- Déclarer le **Content Provider** dans le manifest (***AndroidManifest.xml***).
- Surcharger les 6 méthodes suivantes :
  - ***query()*** : retourne un objet **Cursor** sur lequel itérer pour récupérer les données.
  - ***insert()*** : rajoute des données au ContentProvider.
  - ***update()*** : met à jour une données déjà existante dans le Content Provider.
  - ***delete()*** : supprime une donnée du Content Provider.
  - ***getType()*** : Retourne le type MIME des données contenues dans le Content Provider.
  - ***onCreate()*** : initialise le Content Provider

http://www.tutos-android.com/contentprovider-android

Cursor : utiliser els données

Une requête retourne un objet **Cursor**
pointe généralement vers une ligne de ce résultat.

Android peut gérer les résultats de la requête sans avoir à charger toutes les données en mémoire.

getCount() : obtenir le nombre d'éléments résultant de la requête.

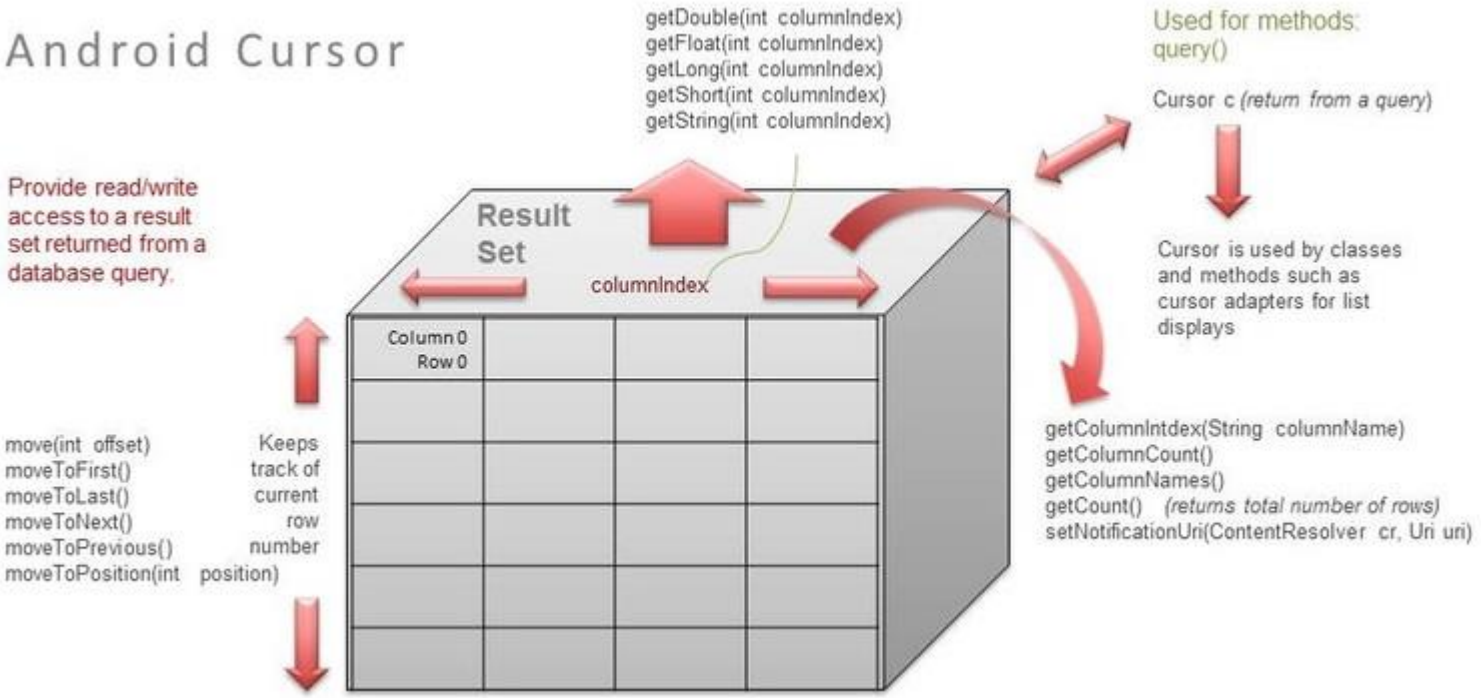
moveToFirst() et moveToNext() : se déplacer entre les données
isAfterLast() : vérifier si la fin du résultat de la requête a été atteinte.

get*() par type de données : getLong(columnIndex), getString(columnIndex), pour accéder aux données d'une colonne de la position courante du résultat.

getColumnIndexOrThrow(String) : obtenir l'index d'une colonne à partir de son nom passé en paramètre.

close() : fermer un Cursor

http://vogella.developpez.com/tutoriels/android/utilisation-base-donnees-sqlite/



http://donkcowan.com/blog/2013/4/21/android-cursor-class-and-objects

EXEMPLES D’UTILISATION

LISTVIEW ET DONNEES EN BASE

Adaptateurs : accès aux contacts

SimpleCursorAdapter données issues de Cursor

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);
    }
```

INTENTS : PLUS DE DETAILS

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);
  }
```

Récupérer un contact dans la liste des contacts

```
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";
```

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>
```