Formation Android

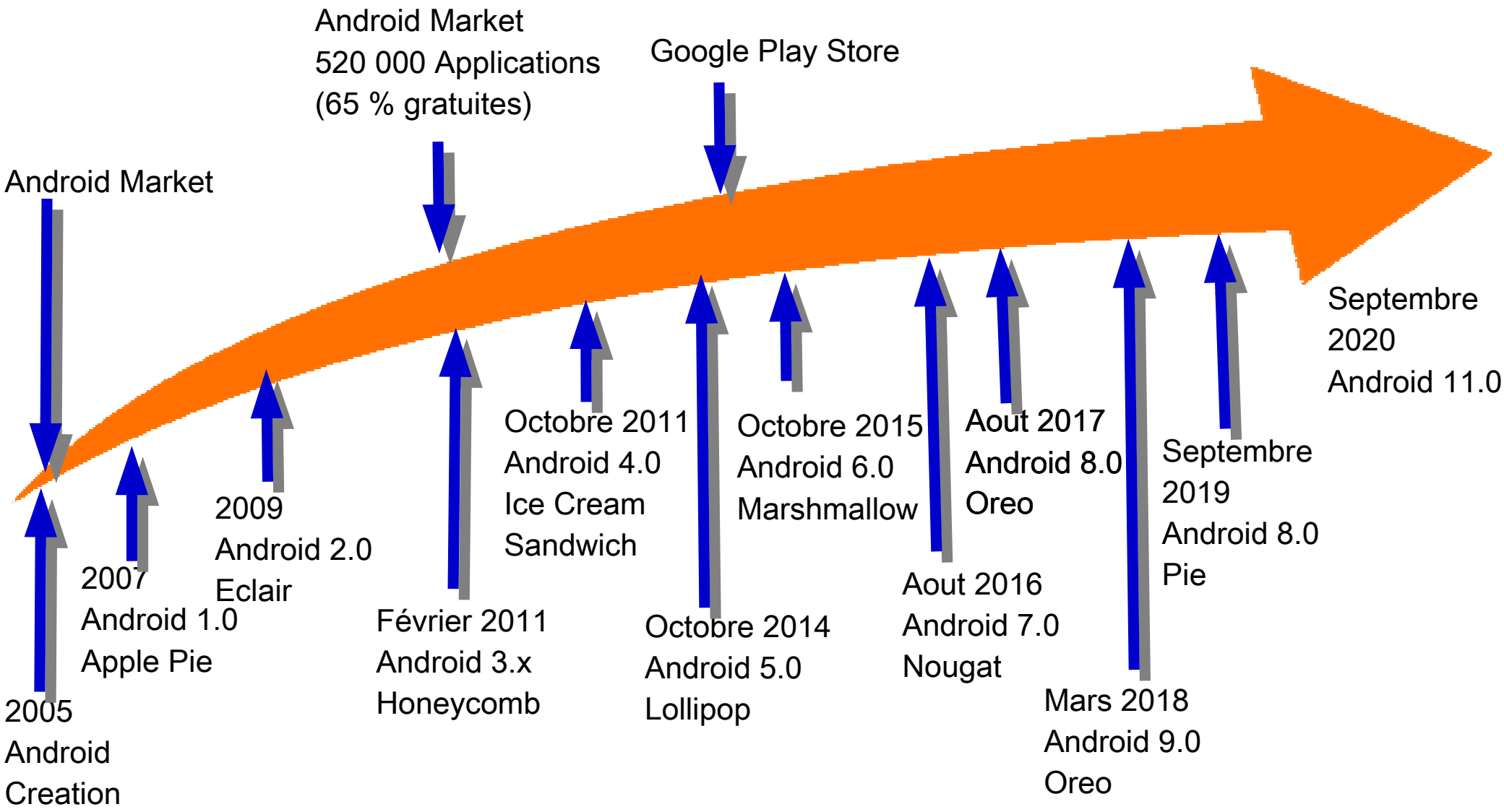
Georges Francisco
Ingénieur logiciel - AL Enterprise

Plan

- Présentation
- Historique Android
- Caractéristiques Android
- TP1 ; Création d'une première application

Présentation

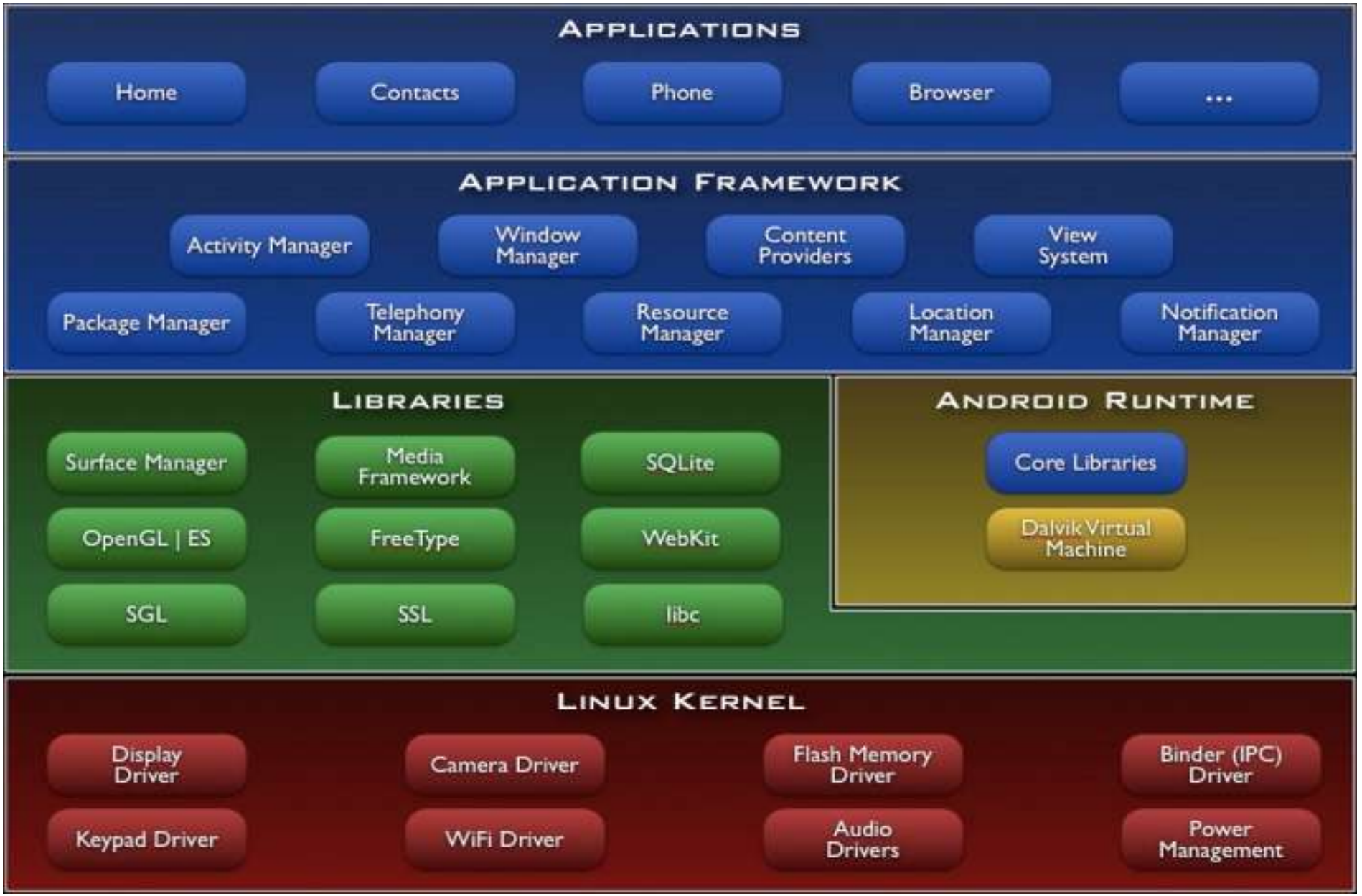
Historique


2005
Android
Creation
Android Market
2007
Android 1.0
Apple Pie
2009
Android 2.0
Eclair
Février 2011
Android 3.x
Honeycomb
Octobre 2011
Android 4.0
Ice Cream
Sandwich
Android Market
520 000 Applications
(65 % gratuites)
Octobre 2014
Android 5.0
Lollipop
Google Play Store
Octobre 2015
Android 6.0
Marshmallow
Aout 2016
Android 7.0
Nougat
Aout 2017
Android 8.0
Oreo
Mars 2018
Android 9.0
Oreo
Septembre
2019
Android 8.0
Pie
Septembre
2020
Android 11.0

Caractéristiques Android

- Système d'exploitation Open-Source basé sur le noyau Linux
- Système conçu initialement pour smartphone et tablettes tactiles, et plus tard pour les téléviseurs android, les smartwatch, ordinateurs ...
- Il incorpore de base une bibliothèque logicielle (webkit, OpenGL, SQLite, FreeType)
- API proposant la gestion non bridée de tous les périphériques, tels que ; GSM/UMTS, Bluetooth, WIFI, l'audio, les images, les sensors, caméra, boussole, récepteur GPS, l'écran tactile, les bases de données, affichage 2D/3D, les SMS ...

Architecture



Linux

Android est basé sur un kernel linux 2.6 mais ce n'est pas linux. Il ne possède pas de système de fenêtrage natif (X window system), la glibc n'est pas supporté, Android utilise une libc customisé appelé Bionic libc.
Enfin Android utilise un kernel avec différents patches pour la gestion de l'alimentation, le partage mémoire, etc. permettant une meilleurs gestion de ces caractéristiques pour les appareils mobiles.

Android n'est pas linux mais il est basé sur un kernel linux. Pourquoi sur un kernel linux ?

Le kernel linux a un système de gestion mémoire et de processus reconnu pour sa stabilité et ses performances.
Le model de sécurité utilisé par linux, basé sur un système de permission, connu pour être robuste et performant. Il n'a pas changé depuis les années 70
Le kernel linux fournit un système de driver permettant une abstraction avec le matériel. Il permet également le partage de librairies entre différent processus, le chargement et le déchargement de modules à chaud.
le kernel linux est entièrement open source et il y a une communauté de développeurs qui l'améliorèrent et rajoute des drivers.

Librairies

Comme cela a été dit précédemment, Android ne supporte pas la glibc, donc les ingénieurs d'Android ont développé une librairie C (libc) nommé Bionic libc . Elle est optimisée pour les appareils mobiles et a été développé spécialement pour Android.

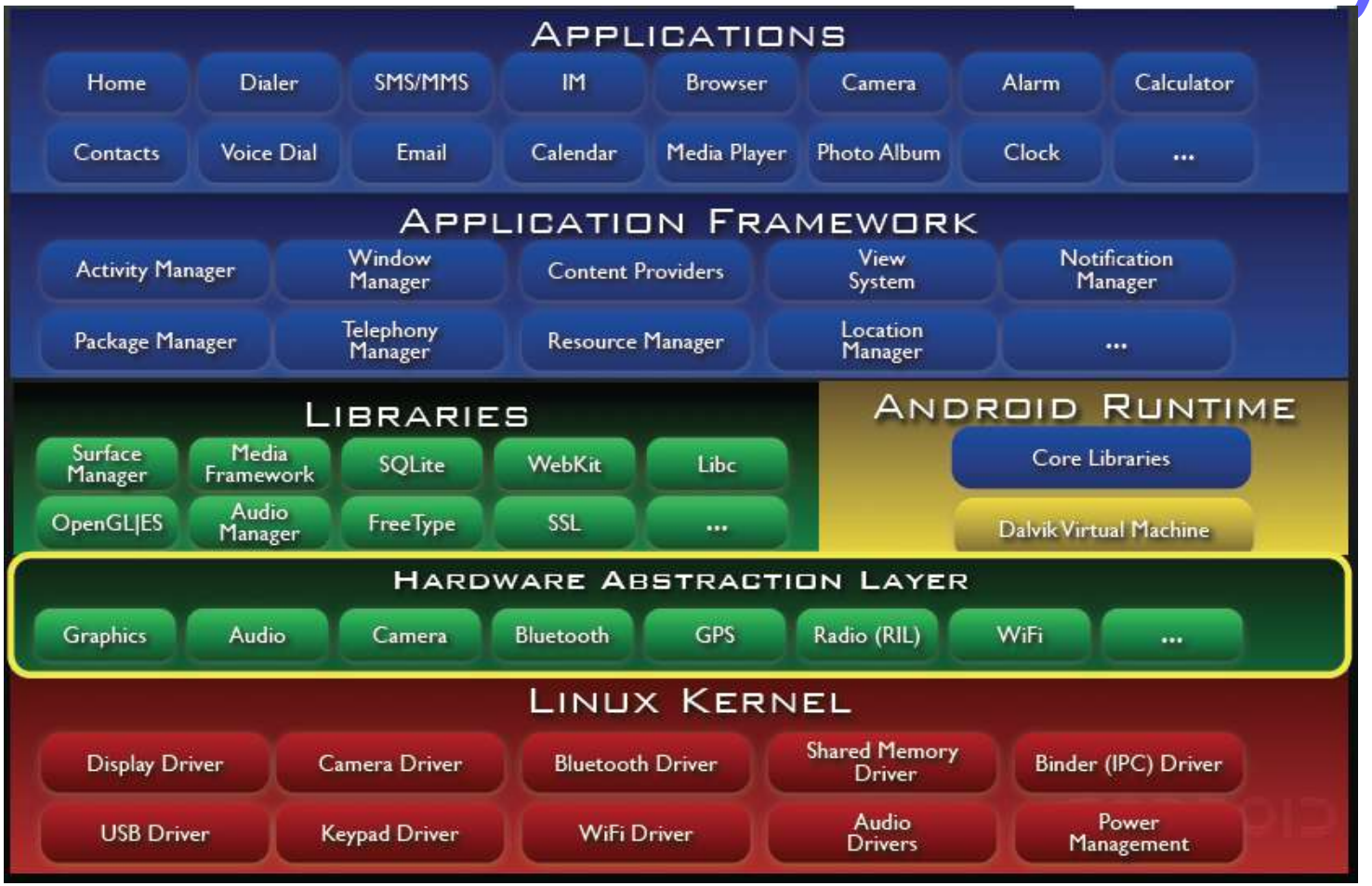
Les ingénieurs d'Android ont décidé de développer une libc propre à la plateforme Android car ils avaient besoin d'une libc légère (la libc sera chargé dans chaque processus) et rapide (les appareils mobiles ne disposent de CPU puissant).
La Bionic libc a été écrit pour supporter les CPU ARM, bien que le support x86 est présent. Il n'y pas de support pour les autres architecture CPU tel que PowerPC ou MIPS. Néanmoins, pour le marché des appareils mobiles, seulement l'architecture ARM est importante.

Cette libc est sous licence BSD, elle reprend une grande partie du code des glibc issue d'OpenBSD, FreeBSD et NetBSD.

Caractéristique importante :

- Elle pèse environ 200Ko soit la moitié de la glibc
- L'implémentation des pthreads (POSIX thread) a été complètement réécrit pour supporter les threads de la machine virtuelle Dalvik. De ce fait la Bionic libc ne supporte les threads POSIX
- Les exceptions C++ et les "wide char" ne sont pas supportés
- Il n'y a pas de "Standard Template Library" (STI)

Hardware Abstraction Layer



Cette couche se situe entre les librairies et le kernel linux, elle fournit les interfaces que doivent implémenter les drivers kernel. Cette couche sépare la plateforme logique des interfaces matérielles. Le but de cette couche est de faciliter le portage des librairies sur différents matériels.

Les ingénieurs d'Android ont décidé de faire cette couche car :

- pas tous les drivers kernel n'ont des interfaces standardisées.
- les drivers kernel sont sous licence GPL ce qui exposerait les interfaces propriétaires des fabricants. Les fabricants veulent pouvoir garder ces interfaces en "closed source"
- Android a des besoins spécifiques pour les drivers kernel.

Android Runtime

La machine virtuelle Dalvik est basée sur une architecture de registre à l'instar de beaucoup de machine virtuel et de la machine virtuel Java qui ont une architecture de pile. Utilisé une architecture de pile ou de registre dépends des stratégies de compilation et d'interprétation choisit. Généralement, les machines basées sur une architecture de pile, doivent utiliser des instructions pour charger les données sur la pile et manipuler ces données. Ce qui rajoute des instructions dans le code machine, et donc il y a plus de code que pour une machine basé sur une architecture de registre. Cependant, les instructions pour une machine basé sur une architecture de registre doivent être encodé pour les registres sources et destinations, ce qui prend également de la place dans le code machine résultant. La différence est essentiellement importante suivant l'interpréteur de code machine présent dans la VM.

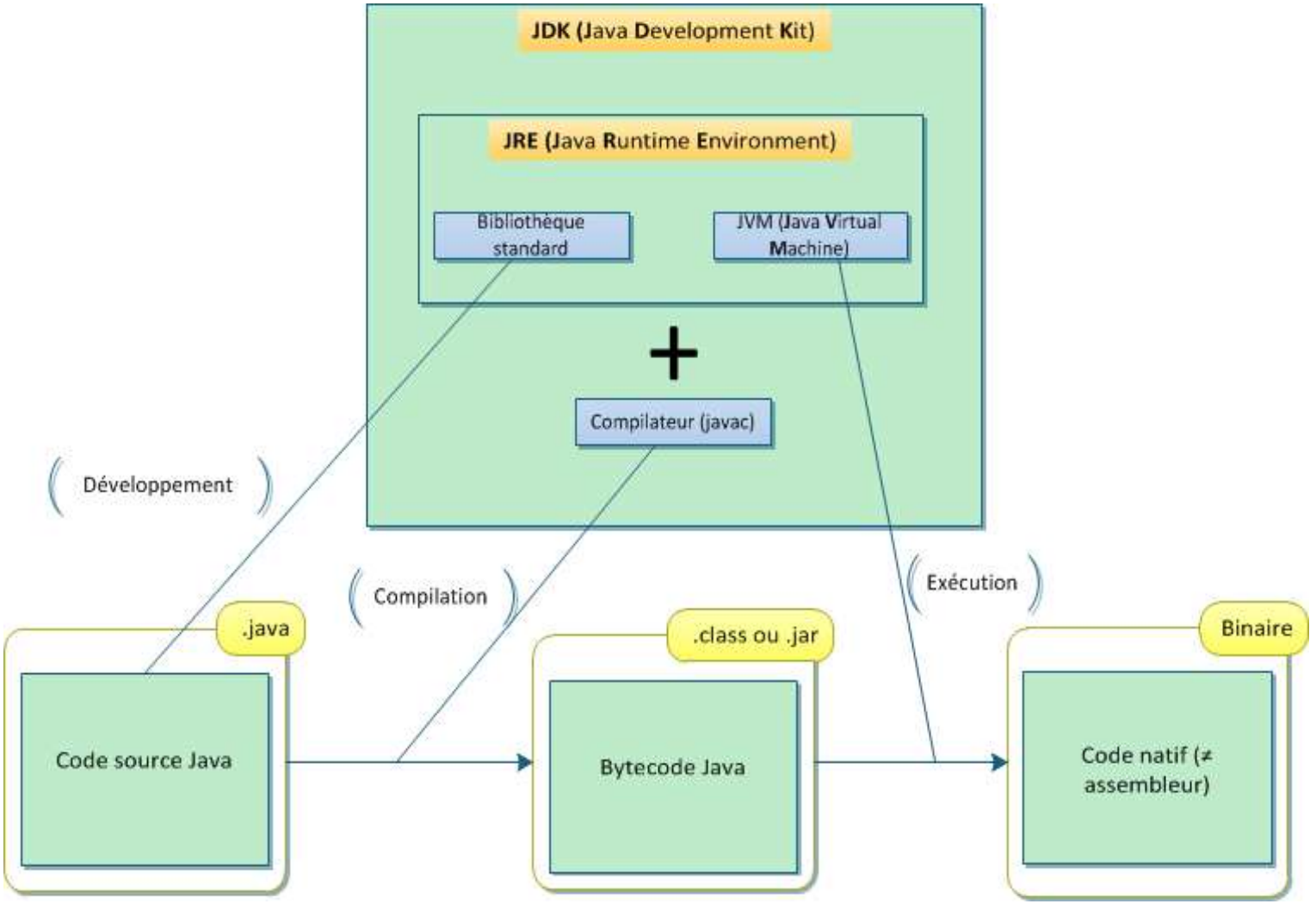
Les applications Java développées pour Android doivent être compilées au format dalvik exécutable (.dex) avec l'outil dx. Cet outil compile les .java en .class et ensuite il convertit ces .class en .dex. Un .dex peut contenir plusieurs classes. Les strings dupliquées et autre constantes utilisées dans de multiples classes sont regroupées dans un .dex. Le bytecode utilisé dans les .dex est le Dalvik bytecode et non le java Bytecode.

Pour comparaison un .dex décompressé est un peu plus petit en taille qu'un .jar compressé dérivé des même fichiers .class.

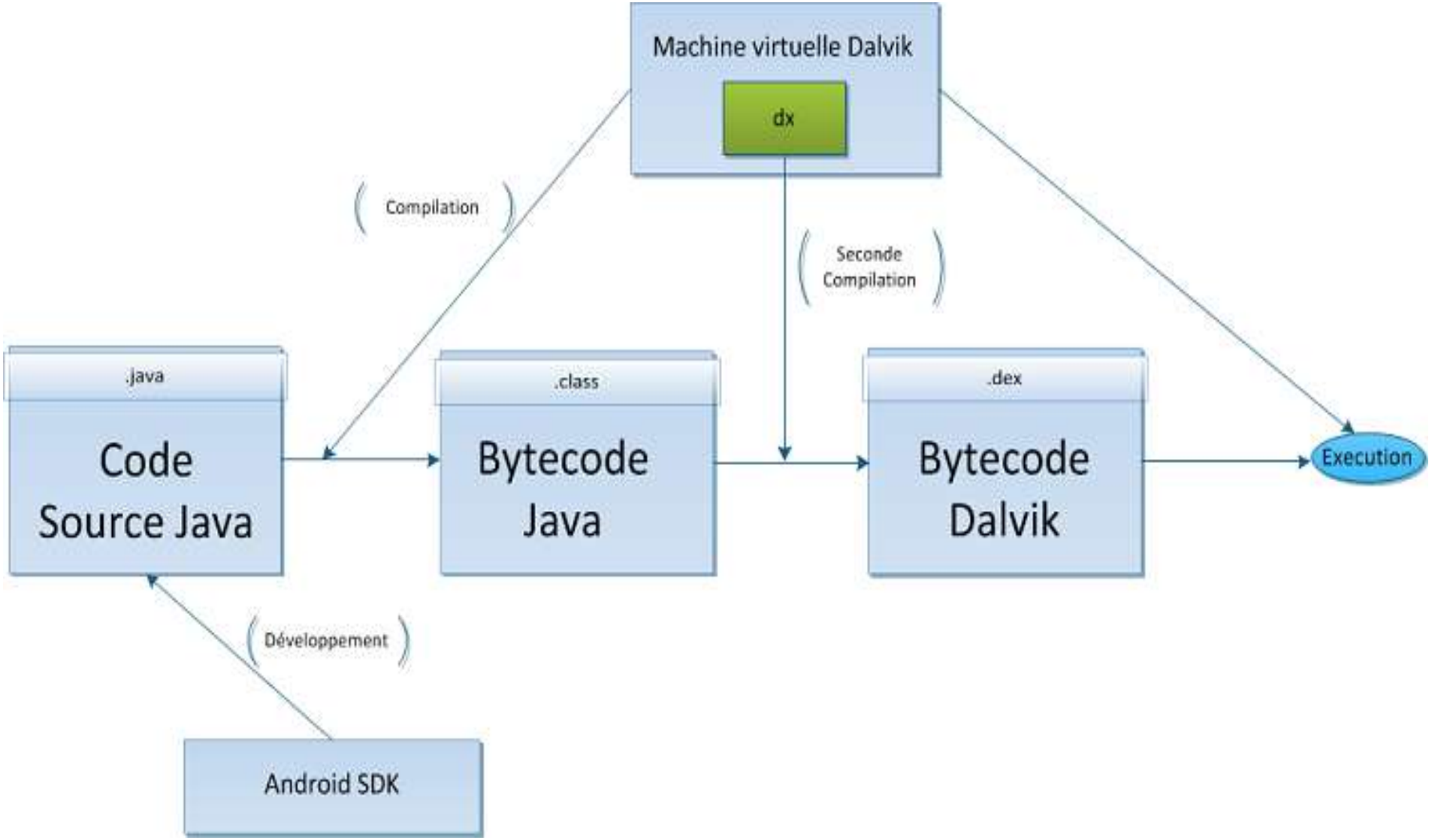
Etant optimisé pour utiliser une quantité de mémoire minimale, la VM Dalvik a quelques caractéristiques spécifiques par rapport aux autres VM:

- la VM a été "dégraissée" pour utiliser moins d'espace mémoire
- pas compilation à la volé (JIT)
- Elle utilise sont propre bytecode et pas le Java bytecode
- La table des constantes a été modifié pour n'utiliser que des indexes de 32 bit afin de simplifier l'interpréteur.

Moteur d'execution



Machine virtuelle Dalvik



Framework

Hardware Services

Les services matériels (Hardware Services) founissent un accès vers les API matérielles de bas niveau :

- **Telephony Service** : permet d'accéder aux interfaces "téléphonique" (gsm, 3G, etc.)
- **Location Service** : permet d'accéder au GPS.
- **Bluetooth Service** : permet d'accéder à l'interface bluetooth.
- **WiFi Service** : permet d'accéder à l'interface Wifi.
- **USB Service** : permet d'accéder aux interfaces USB.
- **Sensor Service** : permet d'accéder aux détecteurs (détecteurs de luminosité, etc.)

Framework

Hardware Services

Les services matériels (Hardware Services) founissent un accès vers les API matérielles de bas niveau :

- **Telephony Service** : permet d'accéder aux interfaces "téléphonique" (gsm, 3G, etc.)
- **Location Service** : permet d'accéder au GPS.
- **Bluetooth Service** : permet d'accéder à l'interface bluetooth.
- **WiFi Service** : permet d'accéder à l'interface Wifi.
- **USB Service** : permet d'accéder aux interfaces USB.
- **Sensor Service** : permet d'accéder aux détecteurs (détecteurs de luminosité, etc.)

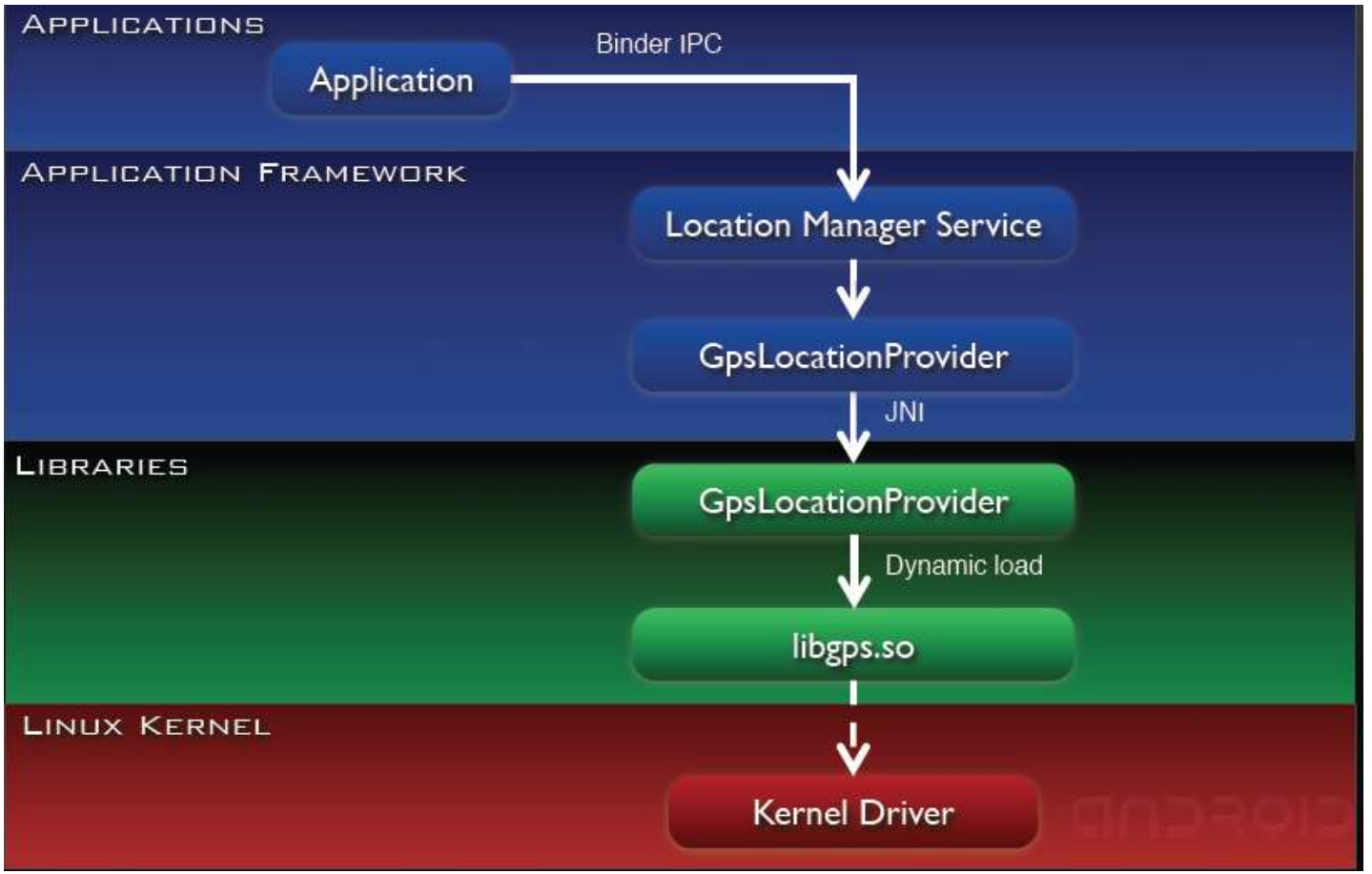
Framework

Core Plateform Services

Android introduit la notion de services. Un service est une application qui n'a aucune interaction avec l'utilisateur et qui tourne en arrière plan pendant un temps indéfini.
Les services cœurs de la plateforme (Core Plateform Services) fournissent des services essentiels au fonctionnement de la plateforme :

- **Activity Manager** : gère le cycle de vie des applications et maintient une "pile de navigation" (navigation backstack) permettant d'aller d'une application à une autre et de revenir à la précédente quand la dernière application ouverte est fermée.
- **Package Manager** : utilisé par l'Activity Manager pour charger les informations provenant des fichiers .apk (android package file)
- **Window Manager** : juste au dessus du Surface Flinger (lien), il gère les fenêtres des applications --> quelle fenêtre doit être afficher devant une autre à l'écran.
- **Resource Manager** : gère tous ce qui n'est pas du code, toutes les ressources --> images, fichier audio, etc.
- **Content Provider** : gère le partage de données entre applications, comme par exemple la base de données de contact, qui peut être consultée par d'autres applications que l'application Contact. Les Données peuvent partager à travers une base de données (SQLite), des fichiers, le réseau, etc.
- **View System** : fournit tous les composants graphiques : listes, grille, text box, buttons et même un navigateur web embarqué.

Exemple d'interaction



L'application utilisateur récupère Location Manager Service en utilisant le Context Manager. Ensuite le Location Manager interroge le GpsLocationProvider qui lui même interroge en utilisant JNI (Java Native Interface) la librairie C/C++ GpsLocationProvider qui va charger la librairie dynamique libgps.so.

Accès aux Applications Sous GITHUB

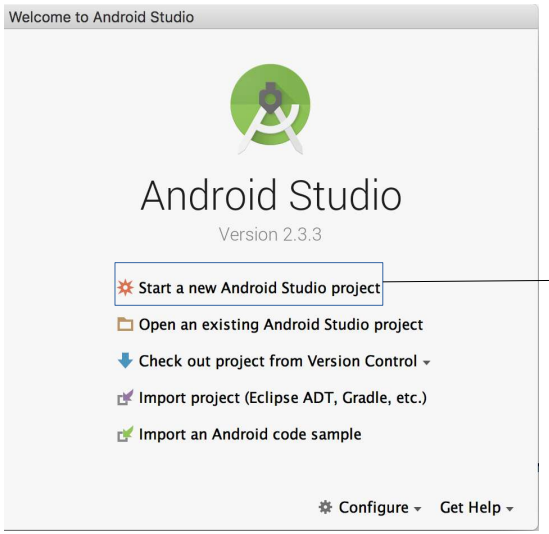
Projet GITHUB sous l'URL : https://github.com/Georges67/Android-Formation.git

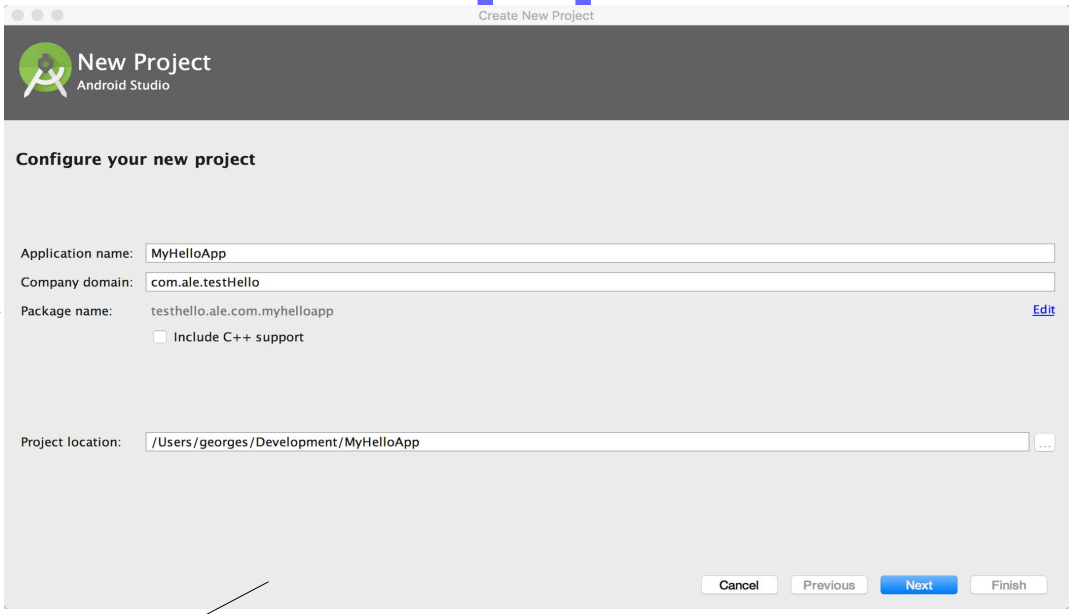
Commande : git clone https://github.com/Georges67/Android-Formation.git

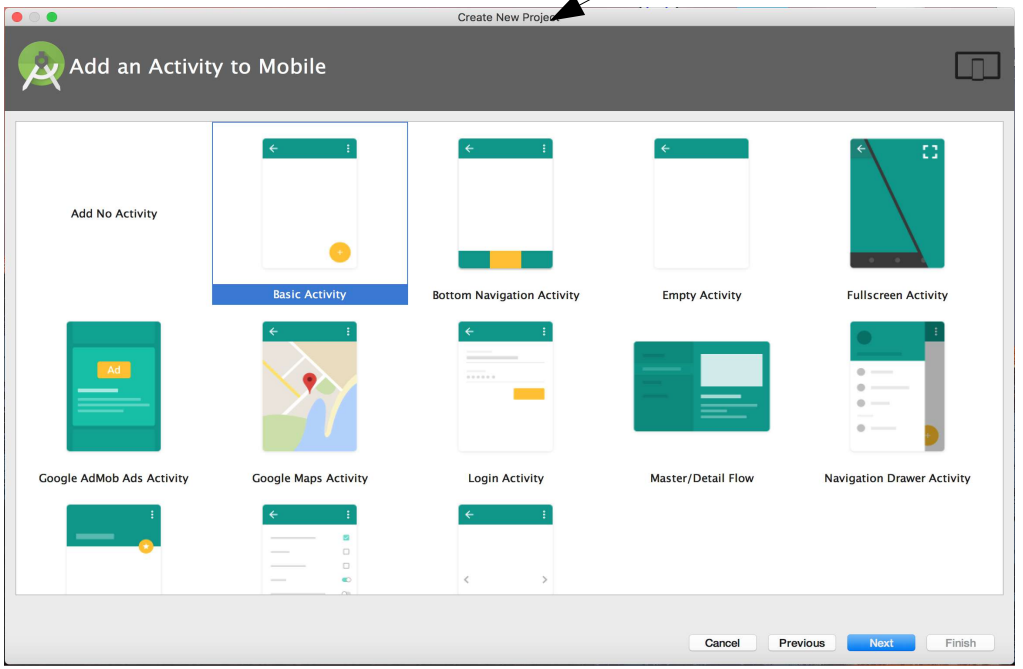
Compte GITHUB : AndroidStudents67

Mot de passe : @ndroid67

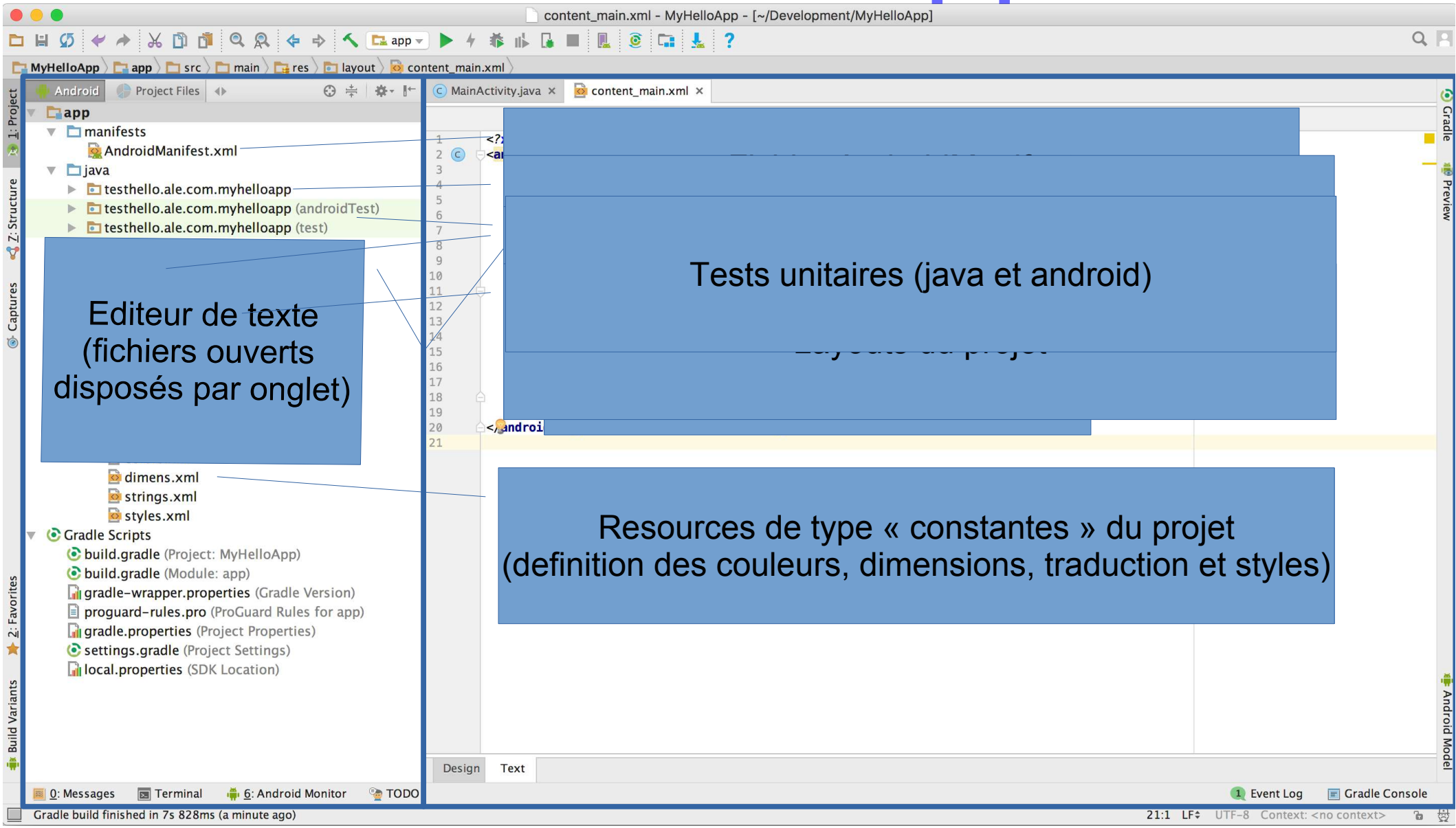
Création d'une Application







Création d'une Application

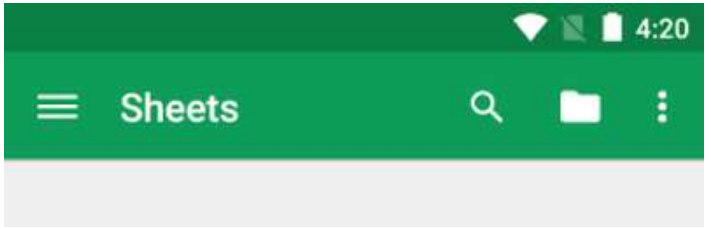


Installation du driver OEM

Voir site officiel https://developer.android.com/studio/run/oem-usb

Si votre device n'est pas reconnu, l'installation du driver correspondant à votre device est nécéssaire.

Création d'une ActionBar



Pour intégrer une ActionBar à Android, vous avez deux choix possibles :

- Utiliser un thème d’activité qui propose une ActionBar.
- Utiliser une vue nommée Toolbar, apparue à partir d’Android 5.0

Langage XML

- Le XML, pour Extensible Markup Language, désigne un langage informatique (ou métalangage pour être plus précis) utilisé, entre autres, dans la conception des sites Web et pour faciliter les échanges d'informations sur Internet. Ce langage de description a pour mission de formaliser des données textuelles. Il s'agit, en quelque sorte, d'une version améliorée du langage HTML avec la création illimitée de nouvelles balises.

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Langage XML

- **Value :** <price>29.99</price>
- **Attributes** : <person gender="female">
- Exemple :

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

XML Tree Structure

- Exemple :

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Les Layouts

- **LinearLayout** permet d'afficher plusieurs vues sur une même ligne de manière horizontale ou verticale. Il est possible d'attribuer un poids aux vues pour effectuer des placements précis.
- **RelativeLayout** permet d'afficher des vues les unes en fonction des autres.
- **TableLayout** permet d'organiser les éléments en tableau.
- **FrameLayout** permet d'afficher une vue à l'écran ou d'en superposer plusieurs les unes au-dessus des autres.
- **ScrollView** permet de rendre « scrollable » la vue qu'elle contient. Attention de ne lui donner qu'un fils et de ne pas fournir des vues déjà « scrollable » sinon il y aura des conflits.

Les Layouts

- **LinearLayout** (orientation : vertical ou horizontal)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <Button
    android:id="@+id/premier"
    android:layout_width="50dip"
    android:layout_height="wrap_content"
    android:text="Click" />
  <Button
    android:id="@+id/second"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Second bouton" />
</LinearLayout>
```

Les Layouts - TP

Les Layouts - TP

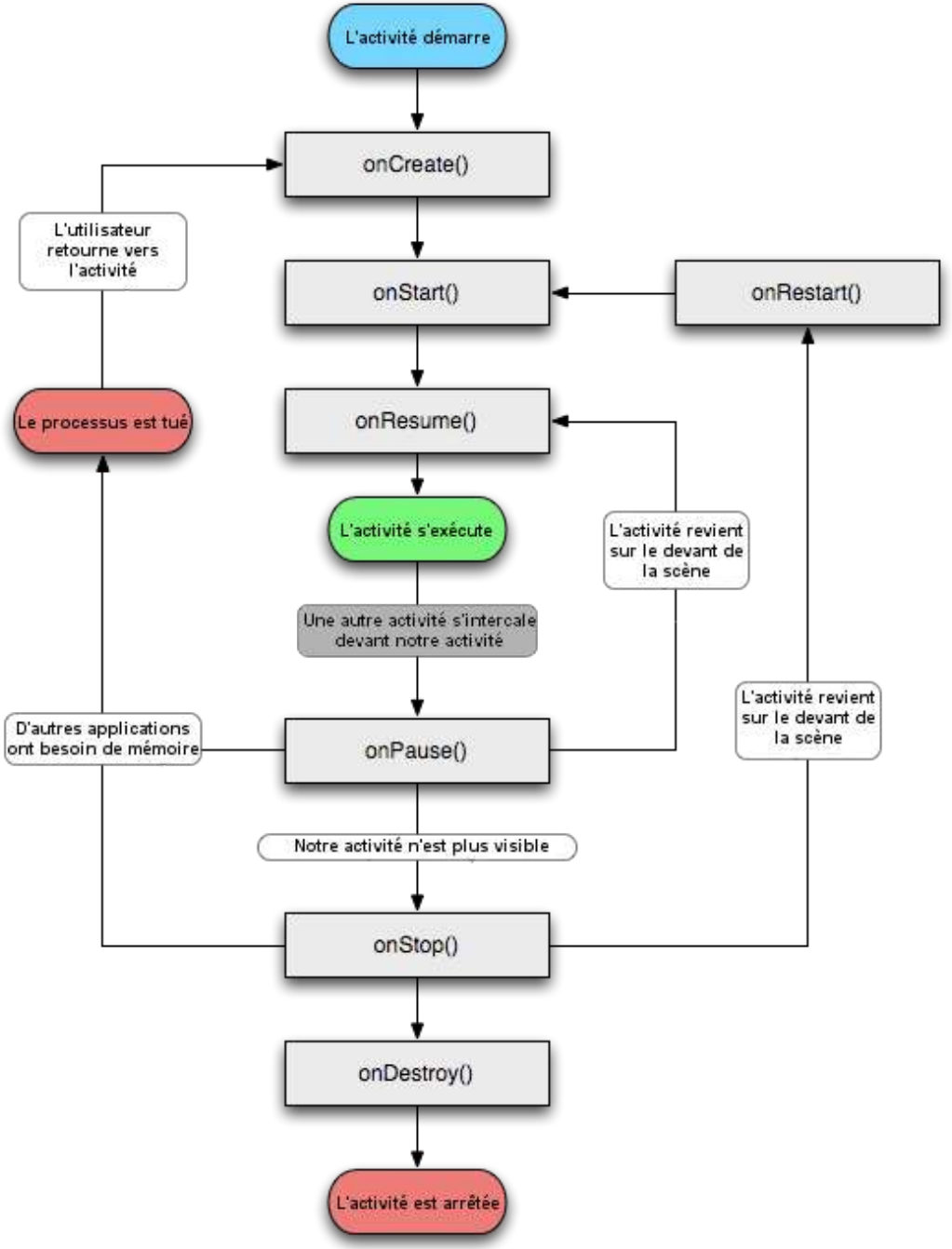
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/
res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:orientation="vertical">
 <LinearLayout
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="horizontal"
 >
  <TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Poids : "
   android:textStyle="bold"
   android:textColor="#FF0000"
   android:gravity="center"
  />
  <EditText
   android:id="@+id/poids"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:hint="Poids"
   android:inputType="numberDecimal"
   android:layout_weight="1"
  />
 </LinearLayout>
 <LinearLayout
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="horizontal"
 >
  <TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Taille : "
   android:textStyle="bold"
   android:textColor="#FF0000"
   android:gravity="center"
  />
  <EditText
   android:id="@+id/taille"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:hint="Taille"
   android:inputType="numberDecimal"
   android:layout_weight="1"
  />
 </LinearLayout>
 <RadioGroup
  android:id="@+id/group"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:checkedButton="@+id/radio2"
  android:orientation="horizontal"
 >
  <RadioButton
   android:id="@+id/radio1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Mètre"
  />
  <RadioButton
   android:id="@+id/radio2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Centimètre"
  />
 </RadioGroup>
 <CheckBox
  android:id="@+id/mega"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Mega fonction !"
 />
 <LinearLayout
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="horizontal"
 >
  <Button
   android:id="@+id/calcul"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Calculer l'IMC"
   android:layout_weight="1"
   android:layout_marginLeft="25dip"
   android:layout_marginRight="25dip"
  />
  <Button
   android:id="@+id/raz"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="RAZ"
   android:layout_weight="1"
   android:layout_marginLeft="25dip"
   android:layout_marginRight="25dip"
  />
 </LinearLayout>
 <TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Résultat:"
 />
 <TextView
  android:id="@+id/result"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:text="Vous devez cliquer sur le bouton « Calculer l'IM
» pour obtenir un résultat."
  />
</LinearLayout>
```

Les Activités

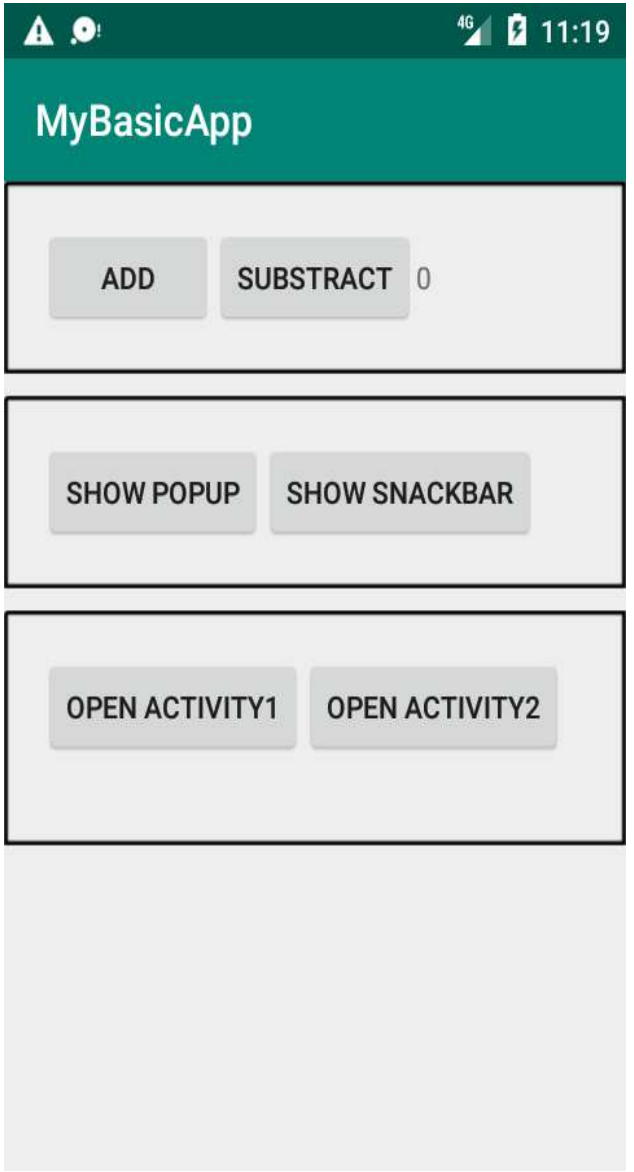
An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View). While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with windowIsFloating set) or embedded inside of another activity (using ActivityGroup). There are two methods almost all subclasses of Activity will implement:

- onCreate(Bundle) is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI, and using findViewById(int) to retrieve the widgets in that UI that you need to interact with programmatically.
- onPause() is where you deal with the user leaving your activity. Most importantly, any changes made by the user should at this point be committed (usually to the ContentProvider holding the data).

Activité – cycle de vie



My Basic Application



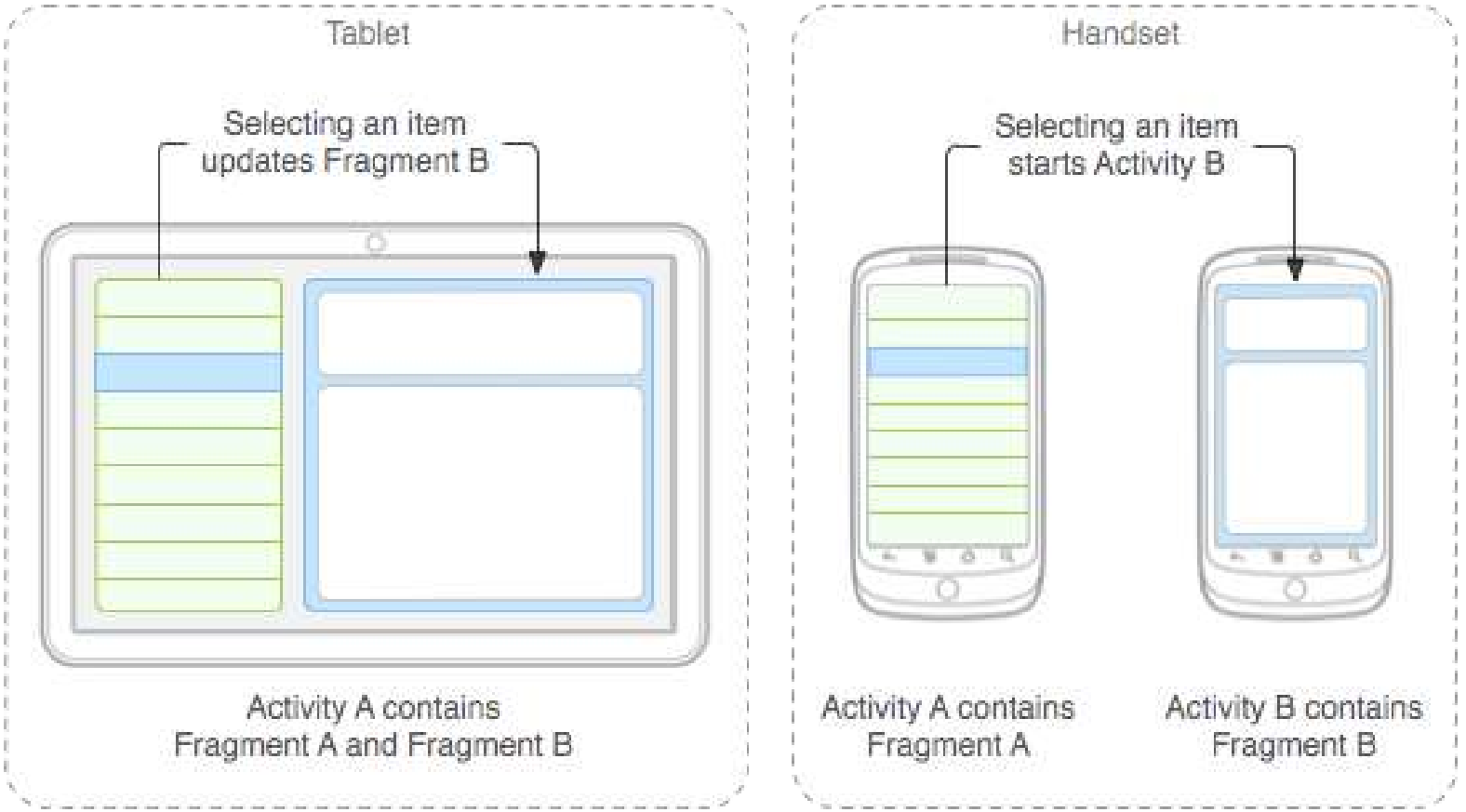
Les Fragments

- A Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).
- A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle. For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments. However, while an activity is running (it is in the resumed lifecycle state), you can manipulate each fragment independently, such as add or remove them. When you perform such a fragment transaction, you can also add it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred. The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the Back button.

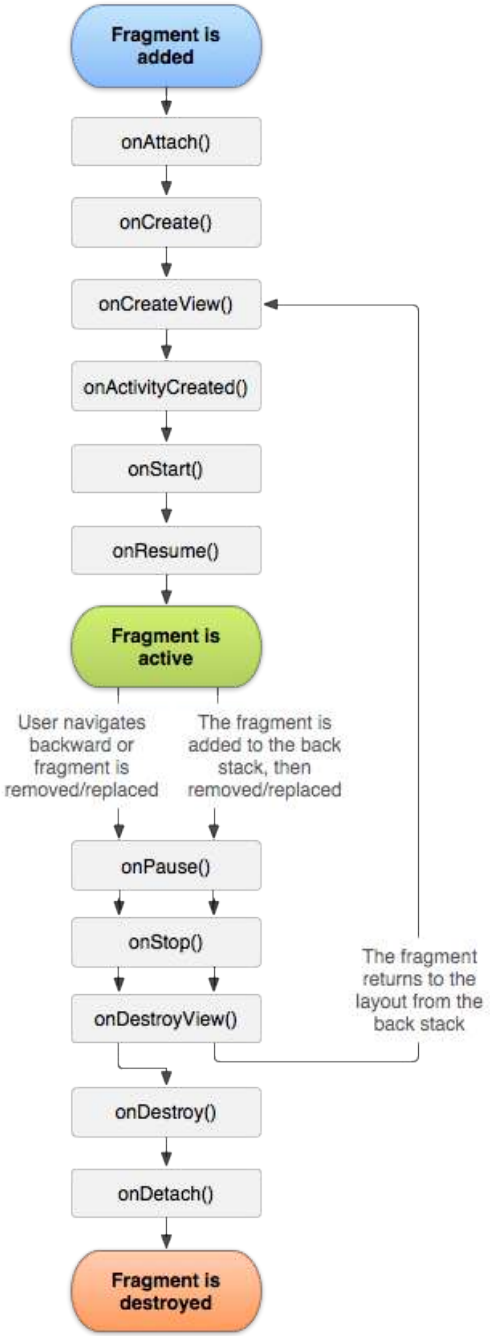
Fragment – Design Philosophy

- Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens, such as tablets. Because a tablet's screen is much larger than that of a handset, there's more room to combine and interchange UI components. Fragments allow such designs without the need for you to manage complex changes to the view hierarchy. By dividing the layout of an activity into fragments, you become able to modify the activity's appearance at runtime and preserve those changes in a back stack that's managed by the activity.

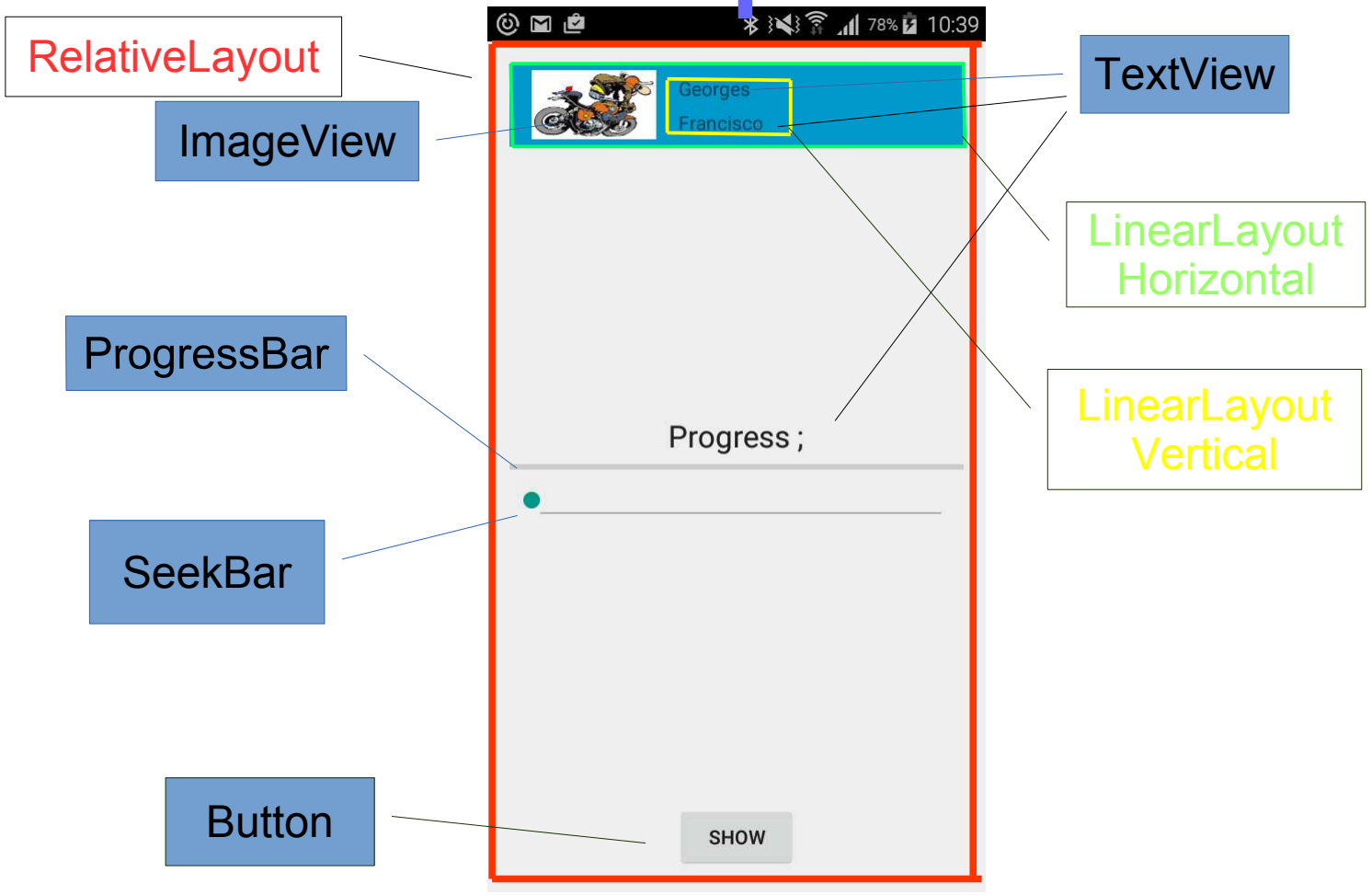
Fragment - example



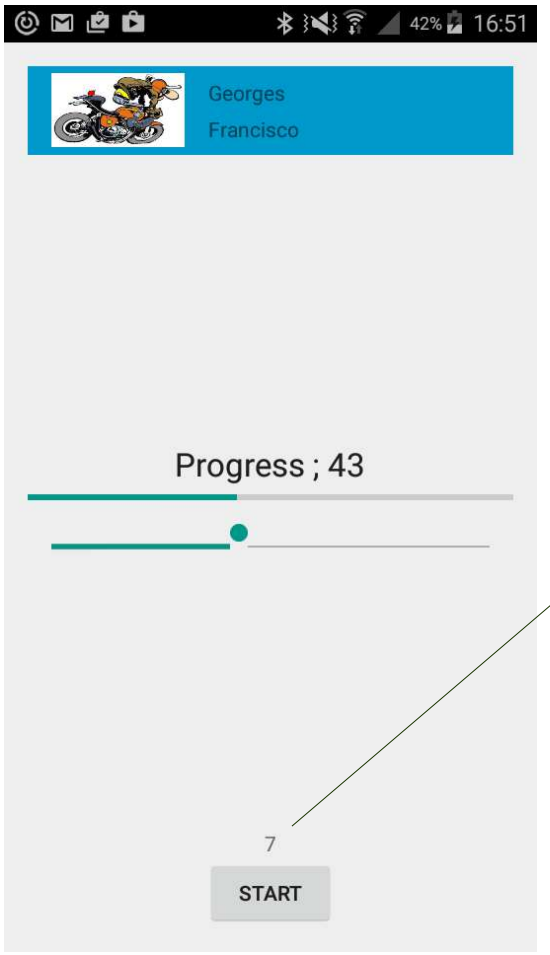
Fragment – life cycle



TP2 ; Création d'un compteur

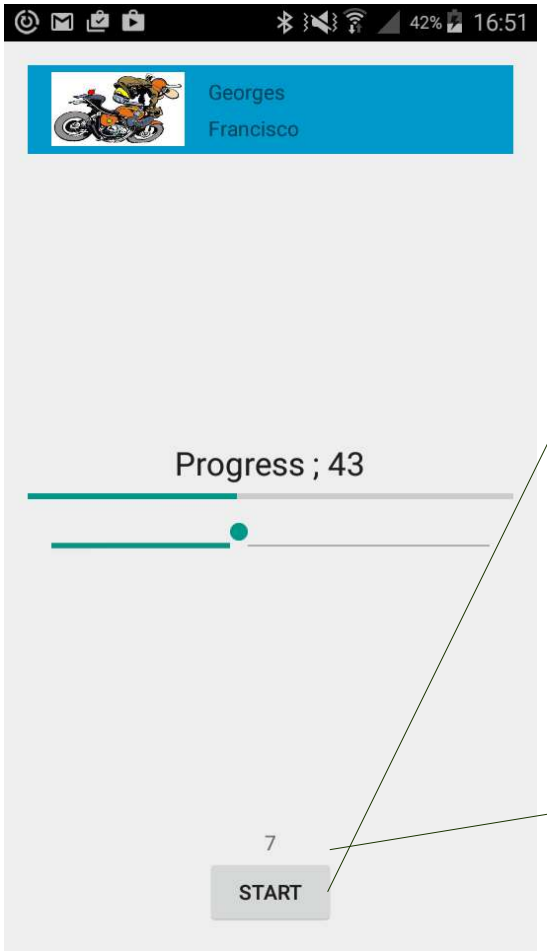


TP2; affichage d'un compteur



Le compteur s'incrémente
à chaque clique
du bouton START

TP3; comptage automatique




```
while(m_counter< 100) {
    m_counterView.setText(String.valueOf(m_counter));
    m_counter += 1;
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Sur clique du bouton START
le compteur s'incrémente automatiquement
Jusqu'à 1000 toutes les secondes

Les Threads

- L'API Thread d'android est l'API Thread de Java ;
  - Implémentation par dérivation de la classe Thread.
  - Implémentation de l'interface Runnable

Les Threads

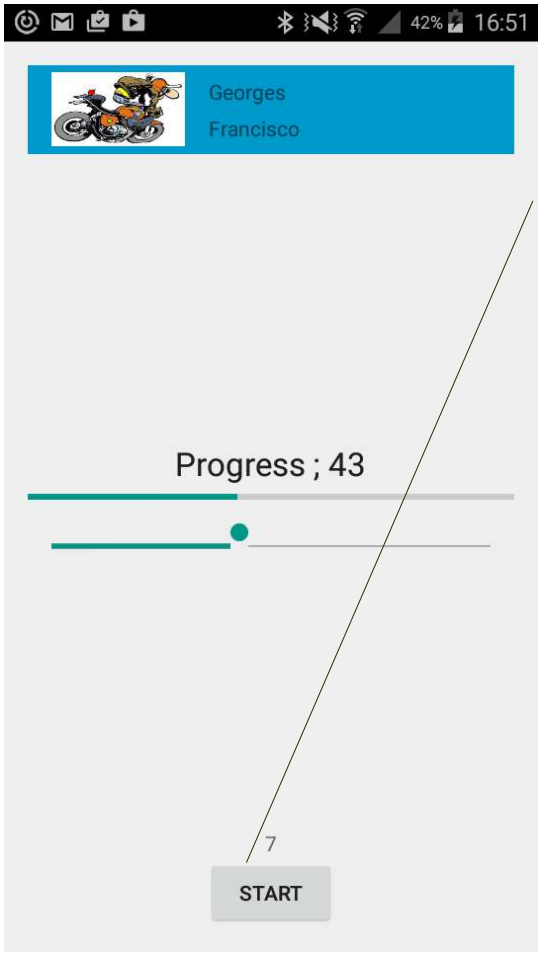
- Une application Android est associée à un processus et à un Thread unique.
- Tous les actions de l'application seront exécutées par le Thread Principal.
- Android impose un temps maximum sur l'exécution d'un événement utilisateur.
- Cette contrainte impose l'utilisation de Thread supplémentaires.
- La complexité de l'utilisation des Threads Android est lié à l'accès aux composants graphiques.
- Un composant graphique est détenu exclusivement par le Thread créateur

Les Threads

Sur Android, nous pouvons gérer les Taches asynchrones de 4 manières ;

1. Utilisation des Threads Java
2. Utilisation des AsyncTasks
3. Utilisation des Timers
4. Utilisation des Handlers

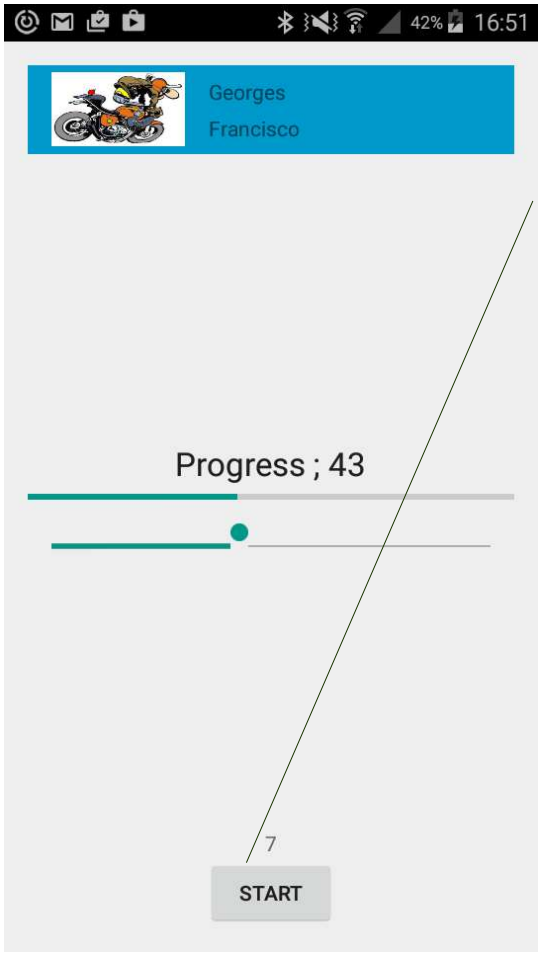
Les Threads JAVA



Le compteur s'incrémente
Sur clique du bouton START
Qui devient Stop
Permettant ainsi d'arrêter
le comptage

La progressBar évolue
également jusqu'à 100
(par pas de 5)

Les Threads JAVA



Le compteur s'incrémente
Sur clique du bouton START
Qui devient Stop
Permettant ainsi d'arrêter
le comptage

La progressBar évolue
également jusqu'à 100
(par pas de 5)

Les AsyncTasks

Android 1,5 introduit les AsyncTasks ;

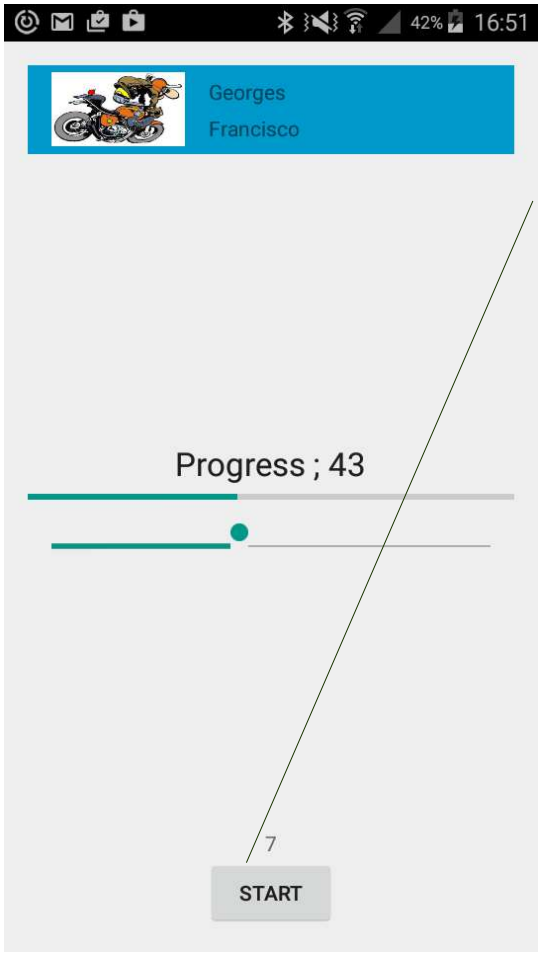
- Permet une gestion plus simple afin d'effectuer des opérations en background et en interagissant avec le GUI (UIThread)
- Android se charge seul de l'allocation et destruction du thread background
- Il est recommandé d'utiliser les AsyncTask pour des opérations de courte durée

Les AsyncTasks

La classe AsyncTasks offre les méthodes suivantes ;

- OnPreExecute ; lancé dans l'UIThread avant l'exécution de la tache
- DoInBackground ; lancé dans un thread de background immédiatement après onPreExecute. Cette étape est utilisée pour exécuter des taches de fond qui prennent beaucoup de temps. Pendant cette étape, nous pouvons utiliser publishProgress pour publier une ou plusieurs infos intermédiaires. Ces infos seront publiées alors dans le UI Thread au travers de onProgressUpdate.
- OnProgressUpdate ; lancé dans l'UIThread suite à un appel de publishProgress
- OnPostExecute ; lancé dans l'UIThread à la fin du traitement de la tache de fond

Les AsyncTasks



Sur clique du bouton START
Le compteur s'incrémente
Toutes les 500ms

La progressBar évolue
également jusqu'à 100
(par pas de 5)

Les Timers

- Les Timers Java permettent le lancement de taches programmées de façon ponctuelle ou répétitive.
- Chaque Timer possède son Thread dans lequel est exécutée sa tache.
- Si un timer n'est plus nécessaire, vous pouvez utiliser la méthode cancel(), qui va se charger de libérer son thread et les autres resources

Les Timers

La classe Timer offre les méthodes suivantes ;

- Cancel ; annule toute programmation du timer
- Schedule ; permet la programmation d'une tache à un moment donné
- ScheduleAtFixedRate ; permet la programmation d'une tache à un moment donné avec une période donnée

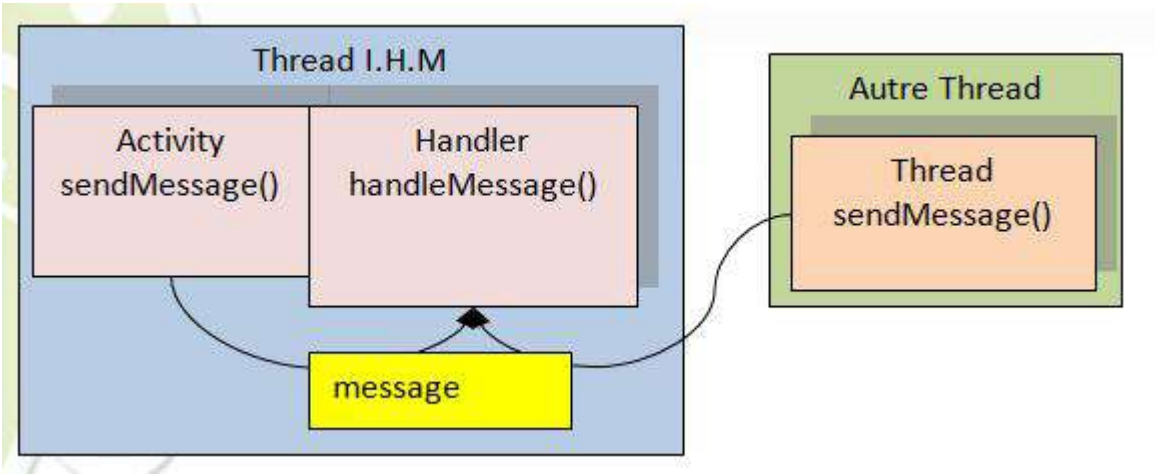
Les Handlers

Le Handler est associé à l'activité qui le déclare et travaille au sein du Thread d'IHM, ce qui signifie que tout traitement effectué par le Handler gèle l'IHM le temps qu'il soit effectué.

Il faut donc considérer le Handler comme celui qui met à jour l'IHM, le Thread (qui appelle le Handler) a la charge du traitement.

Le Handler ne doit que mettre à jour l'IHM, tout autre comportement est une erreur de conception.

Les Handlers



Un Thread communique avec un Handler au moyen de messages. Pour cela :

- Le Thread récupère l'objet Message du pool du Handler.
- Le Thread envoie le message au Handler en utilisant l'une des méthodes suivantes ;
  - sendMessage (envoie le message et le place à la fin de la queue)
  - sendMessageAtFrontOfQueue (envoie le message et le place au début de la queue)
  - sendMessageAtTime (envoie le message au moment donné en paramètre et le place à la fin de la queue)
  - sendMessageDelayed (envoie le message après un temps d'attente passé en paramètre et le place à la fin de la queue)
- Le Handler doit surcharger sa méthode handleMessage pour répondre aux messages qui lui sont envoyés. Il a la charge de mettre à jour l'IHM en fonction de ces données.

Les Services

Il y a 2 types de services :

- Les services de base (unbounded service).
  - Ils ne sont pas accessibles directement.
  - Ils ne donnent pas une référence aux modules initiateurs.

- Les services bindable (bounded service).
  - Ils sont accessibles via une interface.
  - Ils permettent un accès remote.
  - Ils sont la base de l'utilisation de mode IPC* d'android.

* : Interprocess communication

Les Services

Qu'est-ce qu'un service :

Le plus déroutant à propos de la classe Service est ;

- Un Service n'est pas un process séparé. L'objet Service ne tourne pas dans son processus spécif que, il tourne dans le même processus de l'application dont il appartient.

- Un Service n'est pas un thread. Cela ne veut pas dire qu'il n'effectue pas son travail en dehors du thread principal.

Les Services

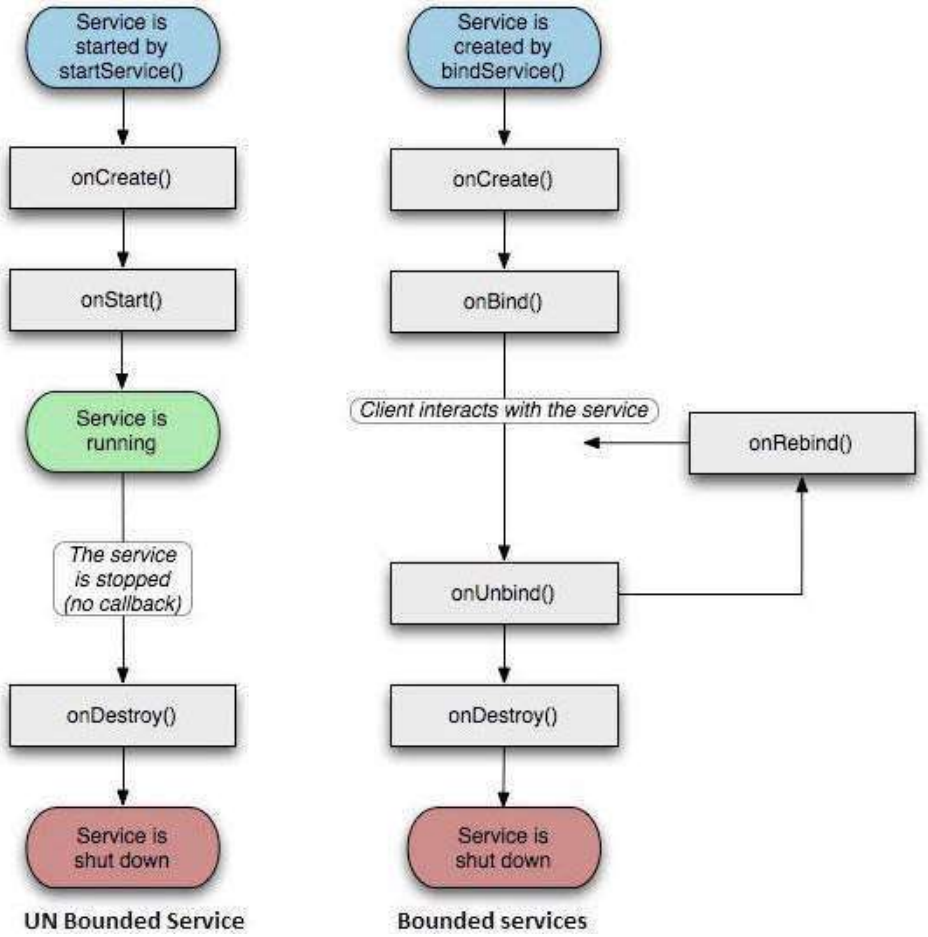
Donc un Service est très simple, fournissant 2 caractéristiques principales :

- Une facilité pour l'application pour dire au système d'effectuer quelque chose en background (même lorsque l'utilisateur n'utilise pas l'interface graphique)
- Une facilité pour l'application d'exposer certaines de ses fonctionnalités/données à d'autres applications.

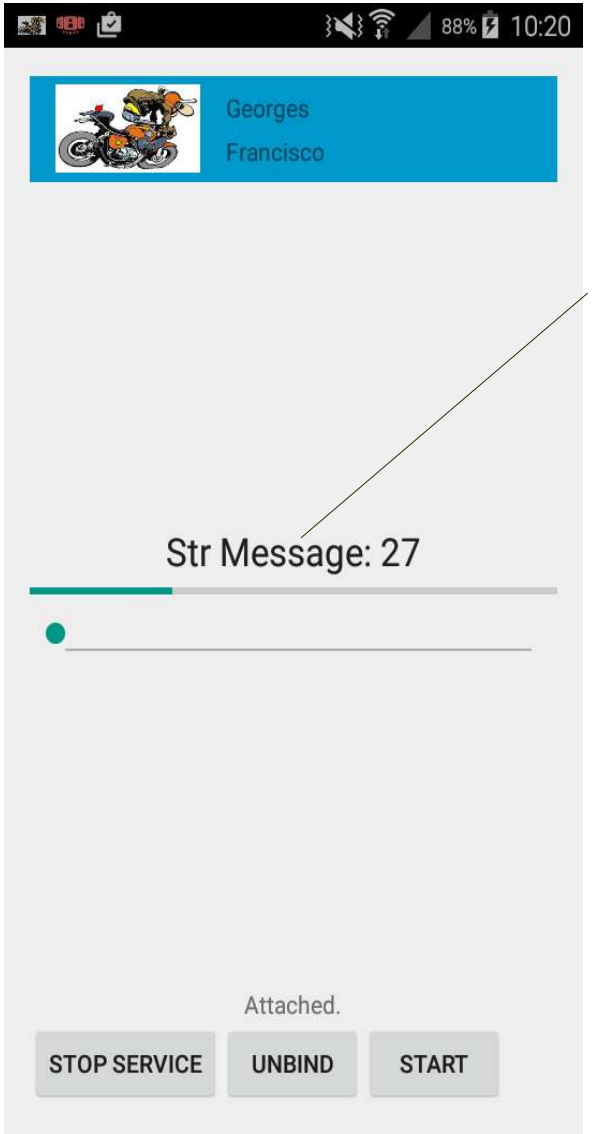
Lorsqu'un service est créé, le système effectue une instanciation et appelle la méthode OnCreate().

Cycle de Vie des Services

Les services peuvent être démarrés de 2 manières ;



TP6; Service avec Timer



Le compteur s'incrémente sur clique
Du bouton Start service,
Puis Bind et enfin Start

Repo GIT

- Cloner le repository GIT :

  git clone git@github.com:Georges67/CnamEntryProjects.git

- Credentials :

  AndroidStudents67 / @ndroid67

How to store data locally in an Android app

- Ways to store data
  - Shared Preferences
  - Internal storage
  - External storage
  - Saving cache files
  - SQLite databases

Shared Preferences

Shared Preferences is the way to go if you're saving primitive data as key-value pairs. It requires a key, which is a String, and the corresponding value for the said key. The value can be any of the following: a boolean, float, int, long, or another string.

Your Android device stores each app's Shared Preferences inside of an XML file in a private directory. Apps can also have more than one Shared Preferences file, and they're ideally used to store app preferences.

Before you can store data with shared preferences, you must first get a SharedPreferences object. There are two Context methods that you can use to retrieve a SharedPreferences object.

Shared Preferences

- Lorsque votre application n'a qu'un seul fichier de preference :

```
SharedPreferences sharedPreferences =
getPreferences(MODE_PRIVATE);
```

- Lorsque votre application possede plusieurs fichier de preference:

```
SharedPreferences sharedPreferences =
getSharedPreferences(fileNameString, MODE_PRIVATE);
```

Shared Preferences

- On getting the SharedPreferences object, you then access its Editor using the edit() method. To actually add a value, use the Editor's putXXX() method, where XXX is one of Boolean, String, Float, Long, Int, or StringSet. You can also remove a key-value preference pair with remove().
- Exemple :

```
SharedPreferences.Editor editor = sharedPreferences.edit();
editor.putString(keyString, valueString);
editor.commit();
```

Cache Files

- Android also provides a means to cache some data rather than store it permanently. You can cache data in either internal storage or external storage. Cache files may be deleted by the Android system when the device is low on space.
- To get the internal storage cache directory, use the **getCacheDir**() method. This returns a File object that represents your app's internal storage directory. You can access the external cache directory with the similarly named **getExternalCacheDir**().
- Although the Android device can delete your cache files if needed, you should not rely on this behavior. Instead, you should maintain the size of your cache files yourself and always try to keep your cache within a reasonable limit, like the recommended 1MB.

SQLite database

- Finally, Android provides support for apps to use SQLite databases for data storage. The databases you create remain specific to your app and can only be accessed inside your app.
- Android provides complete support for SQLite databases. The recommended way of creating SQLite databases is to subclass the **SQLiteOpenHelper** class and override the **onCreate()** method.

SQLite database

```
public class SampleSQLiteDBHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 2;
    public static final String DATABASE_NAME = "myDb";
    public static final String PERSON_TABLE_NAME = "personne";
    public static final String PERSON_COLUMN_ID = "_id";
    public static final String PERSON_COLUMN_NAME = "nom";

    public SampleSQLiteDBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        sqLiteDatabase.execSQL("CREATE TABLE " + PERSON_TABLE_NAME + " (" +
                PERSON_COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                PERSON_COLUMN_NAME + " TEXT)");
    }
    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + PERSON_TABLE_NAME);
        onCreate(sqLiteDatabase);
    }
}
```

SQLite database

- Ajouter de données :

```
private void saveToDB() {
        SQLiteDatabase database = new
SampleSQLiteDBHelper(this).getWritableDatabase();

        ContentValues values = new ContentValues();
        values.put(SampleSQLiteDBHelper.PERSON_COLUMN_NAME,
« Dupont »);
        long newRowId =
database.insert(SampleSQLiteDBHelper.PERSON_TABLE_NAME, null,
values);
    }
```

TP SQLite