

# INF4042, build awesome app

let's start with a basic one...



# Présentation

- Présentation du professeur Lalande | Fabrigli
- Présentation du cours :
  - Mise en place (découverte d'Android Studio, création d'un projet, utiliser git)
  - Les Ressources (gestion et utilisations)
  - L 'Activity
  - Interaction basic (toast, notification et dialog)
  - L 'ActionBar
  - Les Intents
  - Le Multithreading
  - La RecyclerView

# Présentation

- Évaluation par projet :
  - Projet par binôme
  - Livraison avant le **20/12/2015** par Mail aux professeurs ([jean-francois.lalancette@octip.fr](mailto:jean-francois.lalancette@octip.fr) )
    - Objet: "Rendu de projet Binôme1 Binôme2"
    - Message: L'adresse HTTP du repository GitHub
    - Pièce-jointe: un APK de l'application créée (app/build/outputs/apk/app-debug.apk)
  - Conditions de validation
    - L'application ne doit pas crasher et doit être identique à celle qui peut-être compilée depuis le code fournit sur GitHub
    - L'application doit correctement récupérer et afficher les données d'une interface REST
    - Implémenter au moins 6 des standards Android présentés en cours (recyclerview, intent, toast, intentservice, broadcastreceiver et notification )
    - Les layouts doivent utiliser les ressources pour afficher et traduire les chaînes de caractères (langue en **et fr**)

# Appli cible

L'application que nous allons construire contiendra:

- une interface graphique de base (bouton, textview, gabarits)
- des éléments graphiques dynamiques (toasts, boite de dialogue, notifications, action bar)
- plusieurs écrans pour cette application
- un service de téléchargement munie d'une notification de fin de téléchargement
- le traitement des données téléchargées (JSON) et l'affichage dynamique dans une liste

# Mise en place : Lancer l'IDE

- Démarrer votre machine sous Ubuntu
- Dans “montage-camus”, chercher le dossier partagé “fabrigili”  
(*oui il y a une faute*)
  - Naviguer le dossier “android-studio” -> “bin”
- Changer le comportement de votre navigateur de fichier :
  - Fichier -> Préférences
  - Onglet “Comportement”, dans la partie “Fichiers texte exécutables” choisir “Lancer les fichiers textes exécutables lorsqu’ils sont ouverts”, fermer la fenêtre des préférences
- Double cliquer sur “studio.sh”

# Mise en place : configurer l'IDE

- Lors de l'ouverture de la première fenêtre, choisissez de ne pas importer de réglages.
- Après plusieurs fenêtres d'attente, vous arrivez sur le "wizard"
  - next -> choisir "Custom" -> next -> Changer la localisation du SDK pour "home/local/ETD-P/VotreLogin/montage-camus/fabrigili/android-sdk", puis cliquez next -> next -> finish -> wait -> finish
- Choisissez "Start a new (...) project" ou "Open an existing (...) project"

# Mise en place : Créer son premier projet

- Première fenêtre :
  - Choisissez un nom pour votre projet (le nom importe peu)
  - Choisissez un “company domaine” respectant: nom\_binome1\_nom\_binome2.esiea.org
- Seconde fenêtre (form factor) :
  - Sélectionnez uniquement “Phone and Tablet” & minimum **SDK API:15**
- Troisième fenêtre (Activity) :
  - Sélectionnez “Blank activity”
- Quatrième fenêtre (Customize Activity) :
  - Laissez tous et choisir finish
- Après le démarrage de Android Studio :
  - Sélectionnez sur la gauche à la verticale “project” pour voir votre projet ‘façon Android’

# Mise en place : Historique et collaboration

*Créer son git et le partager sur GitHub.*

- Créer un compte GitHub puis un repository nommé :  
**"INF4041\_binome1Name\_binome2Name"**
- Ajouter le projet sous GitHub :
  - En console: (en console '.' est un raccourcis pour le dossier local)
    - git init /loc => crée un repo local **ou** git clone <http://adresse.git> => pour le récupérer
    - git status
    - git add . => ajouter tous les éléments du dossier local
    - git commit => enregistrer les modifications ajouter par 'add' comme une version
    - git remote add origin user@adresse.com:user\_namespace/nom\_du\_projet.git
    - git push / pull => Envoyé ou récupérer les versions
  - Via android studio (menu VCS)

Pour aller plus loin avec git & github: [ici](#) et [ici](#)



# Découverte d'une application : l'architecture

*Enfin l'architecture "façon Android studio"*

- App (tous ce qui est liée au code personnel)
  - Manifests (format xml)
    - Les permissions, les activity (nom, intent, orientation, etc), les services
  - java (format java): le code source et le code de test
  - res (format xml et png)
    - les values, les layouts, les icones, les styles, les effets
- Gradle script
  - Les fichiers de réglages de build
  - Les fichiers build.gradle, définissent notamment l'id unique, le sdk minimum, le type de build, la version du code, les librairies utilisées et les repositories.


# Découverte d'une application : build & run

- Compiler et déployer en mode debug :
  - clic sur le bouton : 
  - ou raccourci clavier : ctrl + r
- Lancement sur un émulateur :
  - choisir "Launch émulator" -> OK
  - Attendre le lancement de l'émulateur, laisser tourner l'émulateur entre deux déploiements.
- Lancement sur son téléphone Android :
  - Version d'Android entre 4.0.3 et supérieur (donc entre ICS et Marshmallow)
  - Cable Micro-USB branché
  - Mode développeur activé : (paramètres -> à propos -> 5 click sur "numéro de build", puis paramètres -> options développeur -> activer le débogage usb)

# Apparté : Sandbox & Antivirus



**InfoSec Taylor Swift**  
@SwiftOnSecurity

 Suivre

My favorite part about "Android antivirus" apps is that they run sandboxed.



# Les Ressources : vue global

- Le dossiers 'res' en détail:
  - Les définitions de tailles relatives : dpi, sp.
  - drawable & mipmap : les icônes en différentes tailles pour répondre aux multiples tailles d'écrans
  - layout : les définitions de vues (leurs éléments et leurs positions)
  - values/dimens : définition de taille (margin, padding, h/w...)
  - values/strings : définition de texte
  - values/styles : définition globale de style (couleur, transparence, visibilité d'éléments)
  - Les automatismes : '-land', '/fr', '-wdpi', 'mdpi/hdpi/xhdpi'
- la classe Java générée 'R'
  - Généré par le SDK après chaque modification d'un élément du dossier res
  - Attribut un int à chaque élément (**public static final int *actionBarDivider*=0x7f010064;**)
  - Accès depuis le java via cette référence : getString(R.string.msg\_toast\_not\_ready);

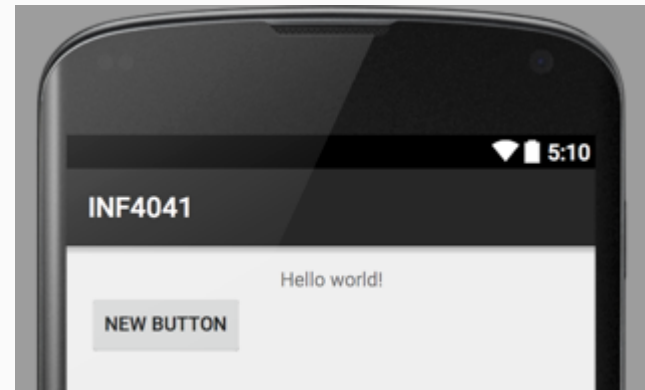
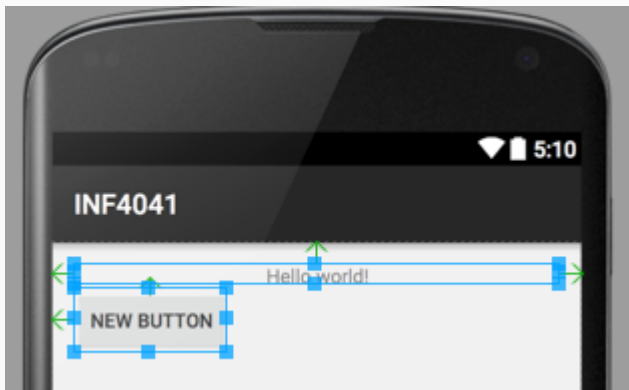
# Les Ressources : le RelativeLayout

Dans un RelativeLayout les éléments sont placés les uns par rapport aux autres, ou par rapport à leurs parents.

- Positionnement:
  - `layout_alignParentRight="true"` & **Left/End/Start/Bottom/top**
  - `layout_alignRight="@id/resource_id"` & **Left/End/Start/Bottom/top**
  - `layout_below` & **above**
  - `layout_toRightOf` & **Left/Start/End**

# Les Ressources : Ajouter un élément

- Ouvrir le fichier layout de notre activité générée `res/layout/activity_main`
  - choisir l'onglet 'design' en bas a gauche du fichier
  - cliquer glisser un "button" dans la vue en dessous du textView
  - agrandir le TextView 'Hello world !' pour qu'il prenne toute la largeur de la vue
  - Sélectionner la TextView et dans l'encadré 'properties' à droite chercher "gravity" et sélectionner "center"
  - vous devez obtenir ceci :



# Les Ressources : Le LinearLayout

Dans le LinearLayout les éléments sont juste disposés les uns après les autres, horizontalement ou verticalement en fonction de l'orientation, MAIS :

- On peut distribuer l'espace avec un weight pour chaque élément
- On peut demander à chaque élément de ne prendre que l'espace nécessaire avec wrap\_content
- On peut combiner wrap\_content et weight pour demander à certains éléments de remplir l'espace restant
- On peut utiliser la gravité pour placer les éléments au sein de leurs zones
- On peut mettre des LinearLayout dans des LinearLayout

# Les Ressources : crée un layout

- Crée un fichier activity\_main.xml dans res/layout-land
  - clic-droit sur res > new > Android resource file
    - file name : "activity\_main.xml"
    - Resource type : layout
    - Root Element : LinearLayout
    - Available qualifier > clic orientation > clic ">>" > Screen Orientation choisir 'landscape'
- Modifier ce nouveau layout
  - choisir l'onglet 'Text' en bas a gauche du fichier
  - créer un TextView (taper "<Tex" et utilisez la complétion avec ctrl+espace)
    - utiliser "wrap\_content" pour définir height & width
    - ajouter un argument "text" avec "hello world !" comme valeur
    - ajouter un argument 'gravity' pr centrer le text.
  - créer un Button de la même manière
  - changer l'orientation du LinearLayout en 'horizontal'
  - Remplir l'espace avec le TextView (width="0dp" & weight="1")



# Les Ressources : mutualiser les valeurs

- Ouvrir res > values > strings.xml
  - Ajouter un élément nommé "button\_text" ayant pour valeur "Click"
- Retourner dans les activity\_main portrait et paysage
  - Remplacer les textes en dur par les références aux ressources string (@string/nom\_elt)
- Internationaliser son application : créer un fichier de strings par langue
  - clic droit sur "res" > new > Android resource file
    - file name : strings
    - ressources type : values
    - Available qualifier : locale > fr > rFR
  - Dans le nouveau fichier de traduction en français redéfinir "hello\_world", "action\_settings" et "button\_text"

# Les Ressources : @runtime

- Dans les layouts, redéfinir les id des éléments en “tv\_hello\_world” et “btn\_hello\_world” (ex: android:id="@+id/tv\_hello\_world")
- Dans le code java de MainActivity dans la fonction onCreate :
  - Récupérer les deux éléments graphiques en objet java.
    - ex : `TextView tv_hw = (TextView)findViewById(R.id.tv_hello_world);`
  - Ajouter la date dans le TextView hello world
    - Pour accéder à strings ou fr-rFR/strings : `getString(R.string.hello_world)`
    - Pour changer le text : `tv_hw.setText(...)`
    - Pour récupérer la date formatée correctement en texte selon la langue :
      - `String now = DateUtils.formatDateTime(getApplicationContext(), (new Date()).getTime(), DateFormat.FULL);`

# Activity

L'Activity est la classe de base dont héritent toutes les 'vues' complètes d'une application

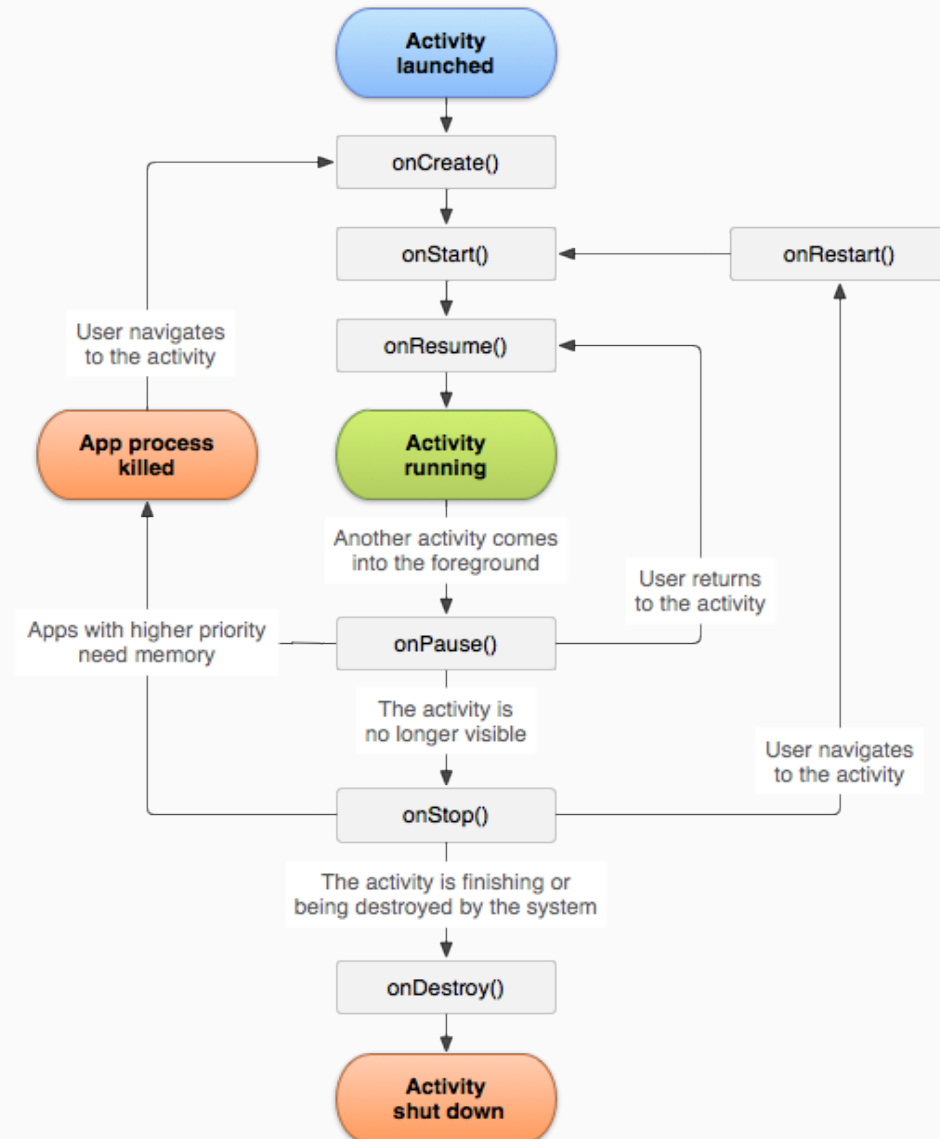
- Chaque Activity st définie dans le manifest de l'app :

- \*Dans la balise application
- \*Le nom = le nom complet de la classe java
- parentActivityName
- screenOrientation
- theme
- intent-filter
  - action MAIN
  - category LAUNCHER

- A un cycle de vie complet ⇒

- A un Context :

- Donne accès aux ressources
- Donne accès à la sandbox (fichier, db, etc)
- Permet de lancer des Intents (Donc de lancer une autre Activity)



# Activity : Définir des Actions

Tous les objets graphiques Android héritent de View, l'objet View à une fonction onClick permettant d'enregistrer une action à faire lors d'un clic. Ainsi tous les objets graphiques Android sont cliquables !

- Deux manières d'implémenter le onClick :
  - Full Java -> rapide
    - Définir un listener sur le bouton avec `btn_hw.setOnClickListener(new OnCli...);`
  - Mixte Java/XML -> plus propre
    - Dans le layout ajouter un argument `onClick=leNomDeMaFct` sur le button
    - Crée la fonction correspondante dans l'activité en public sans sortie et avec une View en entrée: `public void leNomDeMaFct(View v){...}`
- Ajouter un Toast sur le clic : `Toast.makeText(getApplicationContext(),getString(R.string.msg),Toast.LENGTH_LONG).show();`

# Interaction : Toast, notification et Dialog

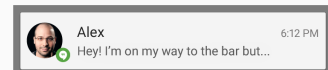
- Un Toast sert à informer l'utilisateur
  - ultra simple à implémenter, toujours utiliser les durées d'affichage std Toast.LENGTH\_
- Un Dialog (aka : popup) sert à
  - demander un choix immédiat
  - valider une action dangereuse
- Une Notification sert à informer l'utilisateur de la fin d'une longue action ou d'un état.
  - A deux états d'affichage (basic ou déplié)
  - A une actions au clic
  - A une action rapide (ex : play/pause de Google Music)
  - A une ou plusieurs actions (ex : terminer et répondre de Inbox)
  - Si la notification représente un état elle doit être fixe. (Non glissable)
  - A une priorité

Live long and prosper!

## Phone Ringtone

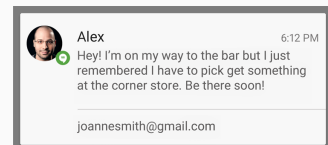
- ☐ None
- ☐ Callisto
- ☐ Ganymede
- ☒ Luna

CANCEL OK



EXPAND

CONTRACT



TEXT

# Dialog, utiliser les standards

- Utilisation du DatePickerDialog
  - Ajouter une variable de class 'dpd' de type DatePickerDialog
  - Dans le onCreate on construit le fragment
    - on attribut à dpd un nouveau DatePickerDialog, son constructeur prend :
      - Le context via 'this'
      - Un 'OnDateSetListener' que l'on doit construire:
        - Dans la fonction 'onDateSet' on renseigne le texte de tv\_hw avec la date renvoyée
      - 3 *integer* représentant la date actuelle : année, mois, jour
  - Dans la fonction onClick de tv\_hw on appelle la méthode 'show()' de dpd
- Pour aller plus loin : (<https://developer.android.com/guide/topics/ui/dialogs.html>)
  - Créer une popup simple avec AlertDialog
  - Définir son propre layout pour une popup

# Notification, un test rapide

- Créer une fonction “notification\_test” dans votre Activity
  - Créer un Notification builder : ‘NotificationCompat.Builder’, lui donner ‘this’ en entrée.
    - Lui donner une petite icône, un titre et un texte (via les set method du builder, pour l’icône, importer une icône dans votre projet ou utiliser ic\_launcher)
  - Récupérer un ‘NotificationManager’ via ‘getSystemService(Context.NOTIFICATION\_SERVICE)’
  - Envoyer la notification au système en utilisant la méthode ‘notify’ du manager
    - L’id est un *int* qui vous permettra de récupérer la notification. (Mettre n’importe quelle valeur)
    - Le deuxième argument attend un objet de type ‘Notification’ que vous pouvez obtenir en appelant la methode .build() de votre builder
- Appeler votre méthode lors du clic et tester
- Aller plus loin : (<https://developer.android.com/guide/topics/ui/notifiers/notifications.html>)
  - Ajouter une action lors du clic
  - Ajouter une priorité

# ActionBar



- Standard permettant de créer une continuité dans l'expérience utilisateur
  - Affiche l'identité de l'application ouverte via le nom de l'application (par défaut `R.strings.app_name`)
  - Permet de lister les actions particulières de l'application (par défaut Settings)
  - Peut aussi afficher le logo de l'application qui permet souvent d'ouvrir un drawer ou de revenir en arrière
  - Peut être splitté pour permettre un accès plus rapide aux actions
  - Peut inclure un champs de recherche
- Ajouter une action simplement
  - Dans `res > menu > menu_main`: ajouter un item nommé "toast me", définir comme toujours visible, et ajouter un attribut icon (`@android:drawable/ic_menu_...`)
  - Dans votre activity, ajouter une toast lors du clic sur notre nouvelle action en complétant le code existant de la méthode `onOptionsItemSelected`.
- Pour aller plus loin : <https://developer.android.com/guide/topics/ui/actionbar.html>



# Les intents : ouvrir une nouvelle activity

Les intents sont des “messages à tous faire” d’Android. Ils permettent d’ouvrir une nouvelle activity, de lancer une autre application, de demander qui veut traiter une action au système, d’afficher des notifications à distance, d’échanger des informations entre vos services et vos activity, ...

- Ouvrir une nouvelle Activity :
  - Créer une Activity (clic droit sur java > new > Activity > BlankActivity)
    - Appeler la “SecondeActivity” et cliquer sur ‘finish’  
(3 fichiers sont créés et un seul édité, lequel ?)
  - Créer une fonction pour créer et lancer l’intent dans MainActivity
    - Créer un intent en utilisant le constructeur Intent (Context packageContext, Class<?> cls), utiliser ‘this’ pour le contexte et SecondeActivity.class pour la classe.
    - Lancer l’intent avec startActivity
  - Appeler cette fonction lors d’un clic via un bouton, un textview, ou un élément de menu
- Les actions : un lien entre les applications, même si elles ne se connaissent pas !
  - essayer : `startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse("geo:0,0?q=Londre")));`
  - <http://developer.android.com/guide/components/intents-common.html>

# Apparté : 'battery saver' ...

- Tuer les applications est contre productif
  - L'espace libre de la RAM est de l'espace gaspillé
  - Rappelez vous le cycle de vie !
  - Tenter d'optimiser c'est bien souvent faire moins bien que le système en pompant des ressources pour le faire ...
- Que faire ?
  - Éviter les apps qui se lancent au démarrage, c'est souvent signe d'une app mal codée ! (ou d'un malware, par exemple écoutant BOOT\_COMPLETED)
  - Utiliser les fonctionnalité système de surveillance de la consommation pour repérer les mauvais élèves.
  - Désinstallez l'application Facebook. (Sérieusement, faite le ! MAINTENANT ! ou bien achetez un iPhone ^^)

# Multithreading : le Thread Java standard

- La manière la plus simple de faire du multithreading
- Pas de synchronisation avec le thread principal, attention au accès concurrent.
- Pas de lien avec le thread principal, vous ne pouvez pas modifier la UI !
- Tourne sur son propre thread
- Tourne sur un thread à priorité identique à celui qui la appelé par défaut

```
new Thread(new Runnable() {  
    @Override  
    public void run() {  
        Log.d(TAG, "Thread worker name:" + Thread.currentThread().getName());  
    }  
}).start();
```

# Multithreading : L'AsyncTask

- La manière la plus facile (et un peu normée) de faire du multithreading
- Méthode standard fournie pour suivre l'évolution de la tâche, ces méthodes s'exécutent automatiquement sur le bon thread.
- Tous les méthodes sont synchronisées, pas de soucis d'accès concurrent
- Ne peut être lancé que depuis le thread UI
- Pool d'AsyncTask => Garantie sans exécution parallèle
- Peut-être tué si l'Activity est en arrière plan. (Appel, retour au launcher, chgmt d'app, etc)
- Pour aller plus loin: <http://www.univ-orleans.fr/lifo/Members/Jean-Francois.Lalande/enseignement/android/presentation-android.html#slide100>

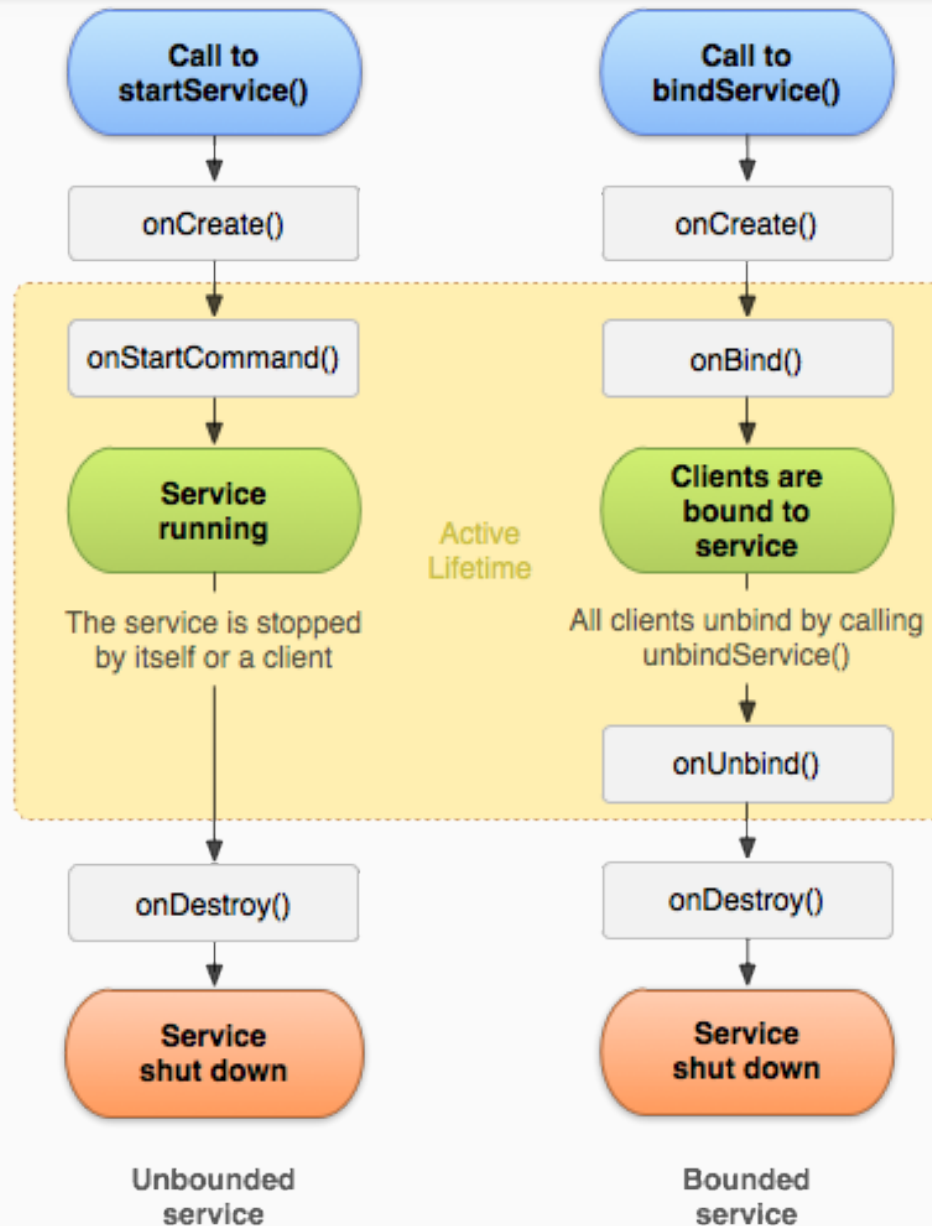
```
//launch by, new TestAsyncTask().execute("I", "Can", "put", "Several object");
private class TestAsyncTask extends AsyncTask<String, Integer, String> {
    protected String doInBackground(String... args) {
        publishProgress(args.length);
        Log.d(TAG, "Thread async doInBackground name:" + Thread.currentThread().getName());
        return "onPostExecute";
    }

    protected void onProgressUpdate(Integer... progress) {
        Log.d(TAG, "Thread async "+progress[0]+" name:" + Thread.currentThread().getName());
    }

    protected void onPostExecute(String result) {
        Log.d(TAG, "Thread async "+ result +" name:" + Thread.currentThread().getName());
    }
}
```

# Service : pour les tâches nécessitant un autre cycle de vie

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



# Service & Threading : IntentService

“The IntentService class provides a straightforward structure for running an operation on a **single background thread**. This allows it to **handle long-running** operations without affecting your user interface's responsiveness. Also, an IntentService **isn't affected by most user interface lifecycle events**, so it continues to run in circumstances that would shut down an AsyncTask” pompé ici: [developer.android.com/.../create-service.html](https://developer.android.com/.../create-service.html)

- Crée la classe java :
  - clic-droit sur 'java' > new > Services > IntentServices (créer la classe et ajouter le tag dans le manifest)
  - Nommez le : 'GetBiersServices'
  - Effectuer les //TODO :
    - delete params1, params2, et bar ainsi que toutes références et fonctions liées
    - remplacer foo par get\_all\_biers
    - Ajouter un log dans la fonction handleActionBiers
- Appeler le service grace au helper depuis l'activity
  - GetBiersServices.startActionBiers(this);

# Service & Threading : téléchargement !

- Pour commencer il nous faut la permission d'accéder à internet :
  - Ajouter "<uses-permission android:name='android.permission.INTERNET' />" dans le manifest avant la balise 'application'
- Implémenter une requête HTTP depuis le service (handleActionBiers)

```
private void handleActionBiers() {
    Log.d(TAG, "Thread service name:" + Thread.currentThread().getName());
    URL url = null;
    try {
        url = new URL("http://binouze.fabrigli.fr/bieres.json");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.connect();
        if(HttpURLConnection.HTTP_OK == conn.getResponseCode()){
            copyInputStreamToFile(conn.getInputStream(),
                                new File(getCacheDir(), "bieres.json"));
            Log.d(TAG, "Bieres json downloaded !");
        }
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
private void copyInputStreamToFile(InputStream in, File file){
    try {
        OutputStream out = new FileOutputStream(file);
        byte[] buf = new byte[1024];
        int len;
        while((len=in.read(buf))>0){
            out.write(buf,0,len);
        }
        out.close();
        in.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

# Threading : BroadcastReceiver

Objectif: notifier l'activité que le service à terminé le téléchargement.

- Créer une sous classe BierUpdate dans votre Activity qui 'extends' BroadcastReceiver. (Utiliser le générateur pour implémenter la méthode) et créer un String unique pour l'action de l'intent

```
public static final String BIERS_UPDATE = "com.octip.cours.inf4042_11.BIERS_UPDATE";  
public class BierUpdate extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Log.d(TAG,getIntent().getAction()); // prévoir une action de notification ici }}
```

- Dans le onCreate de l'activité, on crée un intentFilter et on load/unload le receiver

```
IntentFilter intentFilter = new IntentFilter(BIERS_UPDATE);  
LocalBroadcastManager.getInstance(this).registerReceiver(new BierUpdate(),intentFilter);
```

- Enfin on lance un intent depuis le service pour signaler l'update :

```
LocalBroadcastManager.getInstance(this).sendBroadcast(new Intent(MainActivity.BIERS_UPDATE));
```



# Lire un fichier JSON contenant un tableau

```
public JSONArray getBiersFromFile(){
    try {
        InputStream is = new FileInputStream(getCacheDir() + "/" + "bieres.json");
        byte[] buffer = new byte[is.available()];
        is.read(buffer);
        is.close();
        return new JSONArray(new String(buffer, "UTF-8")); // construction du tableau
    } catch (IOException e) {
        e.printStackTrace();
        return new JSONArray();
    } catch (JSONException e) {
        e.printStackTrace();
        return new JSONArray();
    }
}
```

# RecyclerView

- Anciennement ListView ou GridView
- Constitué de :
  - Une 'view' qui contient la liste, il est définie dans le layout de l'Activity
    - Et généralement une ou plusieurs vues éléments définis dans d'autres xml
  - Un Adapter chargé d'adapter les données aux vues, aka de remplir les vues éléments des données correspondantes à une position dans la liste.
  - Un layoutManager chargé de définir comment les éléments apparaissent les uns par rapport aux autres (liste verticale, liste horizontale, staggered grille, etc ). Il est aussi chargé du recyclage.
- Plus de détails sur : <https://developer.android.com/training/material/lists-cards.html>

# RecyclerView : créer la vue et le manager

- Ajouter recyclerView dans les dépendance du projet
  - gradle script > build.gradle App : ajouter "compile 'com.android.support:recyclerview-v7:22.2.1'" au paragraphe 'dependencies'
  - Enregistrer et synchroniser votre gradle (le bouton apparait après la sauvegarde)
- Ajouter la vue au layout de MainActivity
  - Ajouter une balise android.support.v7.widget.RecyclerView
  - Définir layout\_height en "0dp" et width en match\_parent
  - Aligner en bas du parent et en dessous de bouton hello world
  - Ajouter un id "rv\_biere"
- Appliquer un LayoutManager et un Adapter, dans l'Activity onCreate :
  - Récupérer le RecyclerView du layout avec un findViewById
  - Définir un LayoutManager standart pour notre recyclerView:  
`rv.setLayoutManager(new LinearLayoutManager(this,LinearLayoutManager.HORIZONTAL,false));`
  - Définir un Adapter ⇒ prochain slide !

# RecyclerView > Adapter

- Créer le layout d'un élément: res > layout: click droit > new > Layout ressource file, nom "rv\_bier\_element":
  - Ajouter un TextView en wrap\_content/wrap\_content
  - définir un id "rv\_bier\_element\_name" pour le TextView
- Créer notre propre Adapter comme une sous classe de notre Activity:
  - créer la classe privée "BiersAdapter" qui 'extends' RecyclerView.Adapter<BiersAdapter.BierHolder>
  - crée une sous-classe "BierHolder" qui 'extends' RecyclerView.ViewHolder dans 'BiersAdapter'
  - Définir une variable privée "biers" de type JSONArray.
  - Définir le constructeur de BiersAdapter
    - en entrée prendre un JSONArray
    - définir la variable locale bieres avec l'entrée du constructeur (utilisé this.bieres)
  - implémenter les méthodes abstraites de RecyclerView.Adapter (slides suivant, mais vous pouvez utiliser le générateur d'Android Studio pour créer le squelette des 3 méthodes abstraites)
- Dans le onCreate de l'Activity on peut maintenant ajouter un adapter à notre recyclerView
  - En ajoutant le BiersAdapter on lui donne le JSONArray grâce à notre fonction getBiersFromFile

# RV > Adapter > Crée une vue élément

onCreateViewHolder sert à créer la vue d'un élément. On ne crée pas une vue par éléments, puisque qu'on recycle les vues qui ne sont plus affichées. Aussi cette fonction ne remplit pas les données, elle ne sera donc appelé que lorsque le recycler aura besoin d'une nouvelle vue. Soit au début pour créer le nombre de vues affichées, lors d'un changement de taille de la RecyclerView, ou si une vue est marquée comme 'dirty'.

- Créer une 'View' avec LayoutInflater
  - utiliser la méthode static from, qui prends le context de la vues parent en entrée, pour obtenir une instance de LayoutInflater.
  - Avec cette instance créée, on gonfle une vue avec la méthode inflate. Elle prend en entrée l'id de ressource de votre vue élément, la vue parent et un boolean (false);
- Créer un BierHolder en lui donnant en entrée cette vue nouvellement créée
- Renvoyer ce BierHolder en fin de fonction.

L'argument viewType que la fonction prends en entrée sert à définir des cases différentes pour créer différents types de vues en fonction des éléments.

# RV > Adapter > ViewHolder

Un ViewHolder sert à simplifier l'accès aux données de la vue. L'idée est donc de traduire les éléments de la vue en objets java. Cette classe évite de ré-excuter les findViewByIds à chaque fois qu'on veut remplir la vue.

- Créer une variable publique "name" de type TextView dans BierHolder
- Dans le constructeur de BierHolder, on récupère la vue gonflée par onCreateViewHolder
- A partir de cette vue, on cherche l'élément graphique qui sera mis à jour et on le stocke dans name
  - Si l'objet View est complexe, on peut utiliser un appel à findViewById avec l'id du sous élément graphique que l'on souhaitera mettre à jour plus tard (Attention à bien utiliser la méthode d'itemView et pas celle de l'activity dans laquelle on se trouve !)

# RV > Adapter > Remplir une vue

onBindViewHolder: sert à remplir la vue des données d'un élément, il est donc appelé lors de l'arrivée d'un élément dans l'espace visible de la RecyclerView. Cette vue peut-être une nouvelle vue juste créée par onCreateViewHolder ou une vue recyclée. Une vue recyclée est une vue qui a disparue de l'espace visible et qu'on va réutiliser, aussi il faut redéfinir TOUS les textes/images de la vue pour éviter de se retrouver avec des données d'une ancienne vue.

- Récupérer le nom d'un élément
  - récupérer un objet json avec la méthode getObject de notre JSONArray biers en lui donnant la position que nous donne notre méthode
  - Avec cet objet json qui représente donc l'ensemble des informations d'une bière on peut récupérer le nom avec la méthode getString qui prend en entrée le nom de la clef. Dans notre exemple c'est "name".
- Remplir la vue avec ce nom
  - En utilisant le holder que nous donne notre méthode, on peut définir le nom de la bière en utilisant "setText"
  - Enfin on entoure le tout d'un try/catch pour gérer les exceptions JSON. (pas d'élément en

# RV > Adapter > Les derniers détails

La classe mère de notre Adapter s'occupe de gérer les mécanismes de de la RecyclerView, elle utilise notamment la notion de position dans la liste. Elle part du principe que notre liste de position part de zéro et ne comporte pas de trou. Cependant elle à encore besoin d'une dernière information: combien y a t'il d'éléments en tout !

- Dans la classe BiersAdapter, définir ou modifier la méthode getItemCount pour qu'elle retourne le nombre d'éléments.
  - Récupérer le nombre d'éléments de notre JSONArray avec la méthode length

Lorsque l'activity vas ce lancer, elle vas créer la recylcervView et lui attribuer un adapter. Cette adapter prends en entrée un JSONArray créé à partir du fichier téléchargé par le service. Hors, si c'est le premier lancement, le fichier n'a jamais été téléchargé. Il faut donc mettre à jour notre adapter après chaque mise à jour du fichier.

- Passer le RecyclerView en variable de classe de notre Activity
- Dans l'adapter, on crée une methode public setNewBiere qui prend update le JSONArray et qui appel notifyDataSetChange
- Dans le onReceive de BierUpdate, appeler la méthode setNewBiere (utiliser la méthode getAdapter du recylcervView pour récupérer le BierAdapter, caster le pour pouvoir appeler la méthode)



## Apparté : puissance vs optimisation

✕ Motorola Moto X 2014



✕ Apple iPhone 6 Plus



## Quelques librairies

- Pour faire de jolie transition avec le FAB
  - <https://github.com/truizlop/FABRevealLayout>
- Pour des champs plus Material
  - <https://github.com/florent37/MaterialTextField>
  - <https://github.com/rengwuxian/MaterialEditText>
- Pour gérer une base de donnée depuis des model
  - <https://github.com/satyan/sugar>
- Pour faire des images circulaire
  - <https://github.com/Pkmmte/CircularImageView>
  - <https://github.com/hdodenhof/CircleImageView>
- Pour cropper une image:
  - <https://github.com/jdamcd/android-crop>
- Un drawer plus complet et plus Material que celui de base
  - <https://github.com/mikepenz/MaterialDrawer>
- Des Toast plus complexe:
  - <https://github.com/code-mc/loadtoast>
- Helper pr le traitement d'image distante
  - <https://github.com/bumptech/glide>
- Affichage pour les erreurs de formulaire
  - <https://github.com/thyrlian/AwesomeValidation>

# Api REST utilisable pour le projets

- CKiKiPaye
  - index <https://ckikipaye.octip.co/:model>
  - show
- BugZilla
  - index: [https://bugzilla.mozilla.org/rest/bug?assigned\\_to=lhenry@mozilla.com-](https://bugzilla.mozilla.org/rest/bug?assigned_to=lhenry@mozilla.com-)
  - show: <https://bugzilla.mozilla.org/rest/bug/:id>
- Binouze
  - index: <http://binouze.fabrigli.fr/bieres.json>
  - show <http://binouze.fabrigli.fr/bieres/:id.json>
- Betaseries
  -