



# Android : OS pour « objets » communicants

## Équipe :

Nicolas Stouls <[nicolas.stouls@insa-lyon.fr](mailto:nicolas.stouls@insa-lyon.fr)>

Oscar Carrillo <[oscar.carrillo@cpe.fr](mailto:oscar.carrillo@cpe.fr)>

Anthony Chomienne <[anthony@mob-dev.fr](mailto:anthony@mob-dev.fr)>

membre de  UNIVERSITÉ DE LYON

  
**CPE**  
LYON  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON

- Durée : 8h
- Fil directeur de l'intervention (pour demain) :
  - Réaliser un jeu connecté avec une bande de LEDs et 2 smartphones
- Compétences visées aujourd'hui :
  - Découverte de l'environnement de développement
  - Réaliser une interface graphique simple
  - Utiliser des capteurs
  - Développer un système communicant
- Démarche :
  - 4h de cours/TD guidés
  - 4h de TP

# Pourquoi Android ?

- Internet des objets :
  - Capteurs
  - Crowd Sensing
  - Mobile edge computing

- La page du cours
- La bible contenant « presque tout » :
  - <http://developer.android.com/index.html>
- Un MOOC Kivabien :
  - <https://openclassrooms.com/fr/courses/4428411-developpez-des-applications-android-connectees>
- La bible contenant « tout » :
  - <http://www.google.fr>
- La bible de secours pour les cas critiques :
  - <http://www.bing.fr>



# Android

## *Une introduction*

*Nicolas Stouls*  
*nicolas.stouls@insa-lyon.fr*

membre de UNIVERSITÉ DE LYON





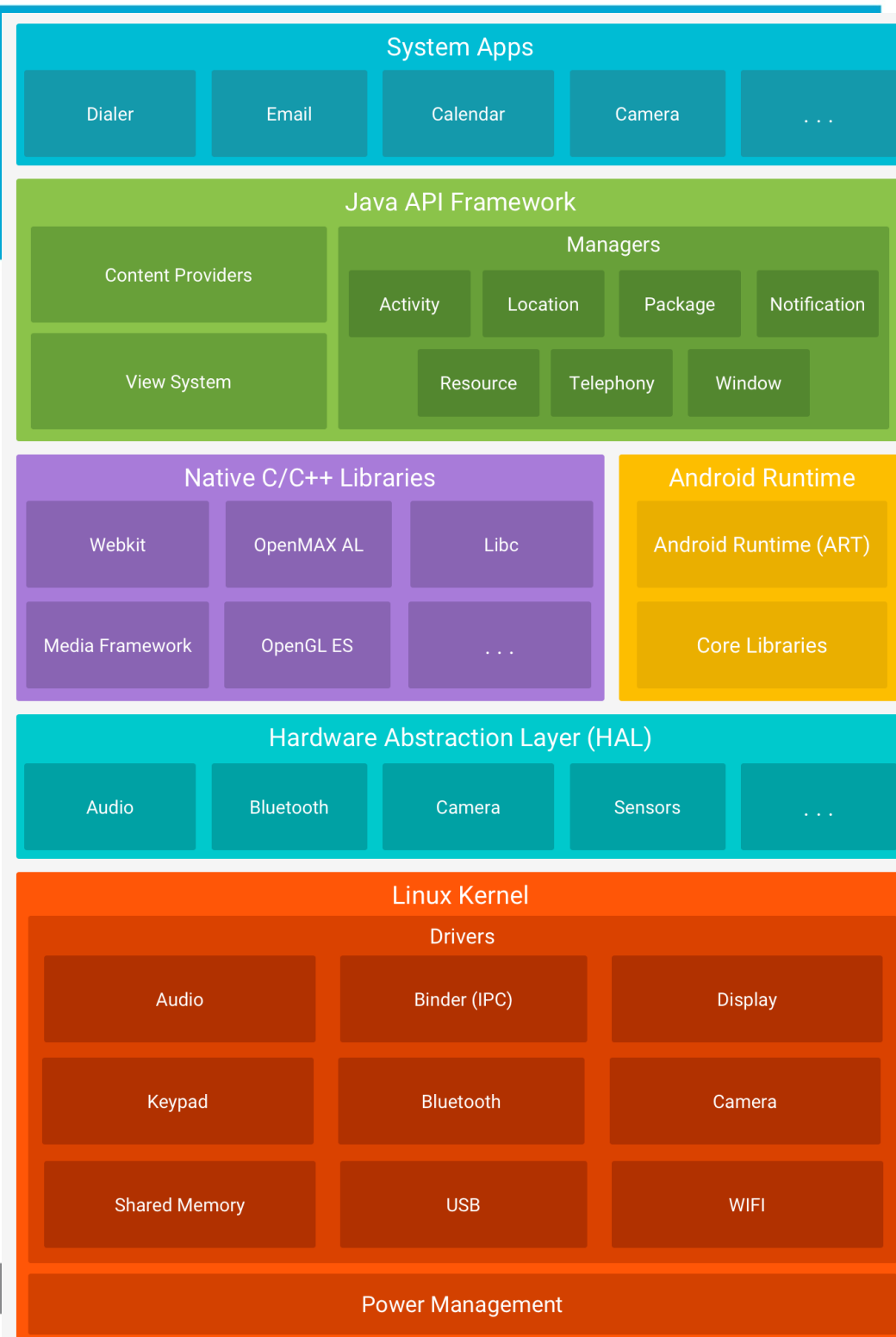
# Caractéristiques des objets visés par Android

## *Un OS pour « objets » communicants*

- Proche de l'utilisateur  
*(Pour mieux le traquer)*
- Autonomes en énergie  
*(Traquer longtemps)*
- Bardés de capteurs  
*(Savoir ce que fait l'utilisateur)*
- Intelligents  
*(Comprendre ce que fait l'utilisateur)*
- Hyper-communicants  
*(Pour transmettre les infos à qui de droit)*
- Faciles d'interactions avec un humain  
*(Recueillir les infos qu'il veut donner spontanément et créer le besoin)*
- Pas autonomes du tout sur la mobilité  
*(Rester au plus proche de l'utilisateur)*

# Architecture

- OS basé sur Linux
- Langage de dev :
  - Java+XML
- Bytecode  $\leadsto$  natif :
  - Java : JIT (@runtime)
  - Android : ART (@install)
- Sécurité :
  - 1 appli = 1 UID
  - Exécution sandboxée
- Interactions :
  - via système (Intent)
  - Permissions



- Portage d'une application Java sous Android  $\approx$ 
  - Définition / création d'une nouvelle IHM
  - Copié / collé de tout le reste
- L'accès aux périphériques fait parti de l'API
  - Java : machine virtuelle générique. Ex : pas d'accès USB
  - Android : machine virtuelle avec contraintes sur le matériel
    - *Wifi,*
    - *bluetooth,*
    - *USB,*
    - *GPS,*
    - *Accéléromètre,*
    - *téléphone,*
    - *SMS,*
    - *etc.*





- Java :
  - Code source
- XML :
  - Manifest
  - Interfaces graphiques
  - Données de l'application
  - Constantes textuelles multilingues
- Ressources complémentaires
  - Images (Mipmap ou Drawable)
  - Bibliothèques externes
- Unité de déploiement :
  - .apk signé

- Composant :

- un ensemble d'objets
- Sous Android :

- *Activité, ~~Service~~, ~~BroadcastReceiver~~, ~~ContentProvider~~*

Seulement *Activité* dans le cadre  
de cette introduction

- Orienté composant

- Appel explicite (com.android.chrome)
- Appel implicite (Action + données et/ou catégorie)
  - *Ex : Ouverture d'une URL = Action VIEW + URL en donnée*

- Notion d'activité

- Activité  $\approx$  fenêtre
- Ouvrir une fenêtre :
  - ***empiler** une activité au dessus*
  - *activités masquées mises en **pause***



# Hello world

membre de UNIVERSITÉ DE LYON

LYON  
**CPE**  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON



# Démo : que fallait il faire ?

- Grandes lignes :
  - New : Android project
    - *Application name -> Mon premier programme*
    - *Company Domain -> nstouls.cpe.fr (nom du package)*
    - *SDK -> 19 (KitKat)*
    - *Empty Activity + generate layout file*
  - Run
    - *Génération d'un fichier .apk*
    - *Chargement du fichier dans le simulateur ou le device*
- Oh que c'est bô ! ;)

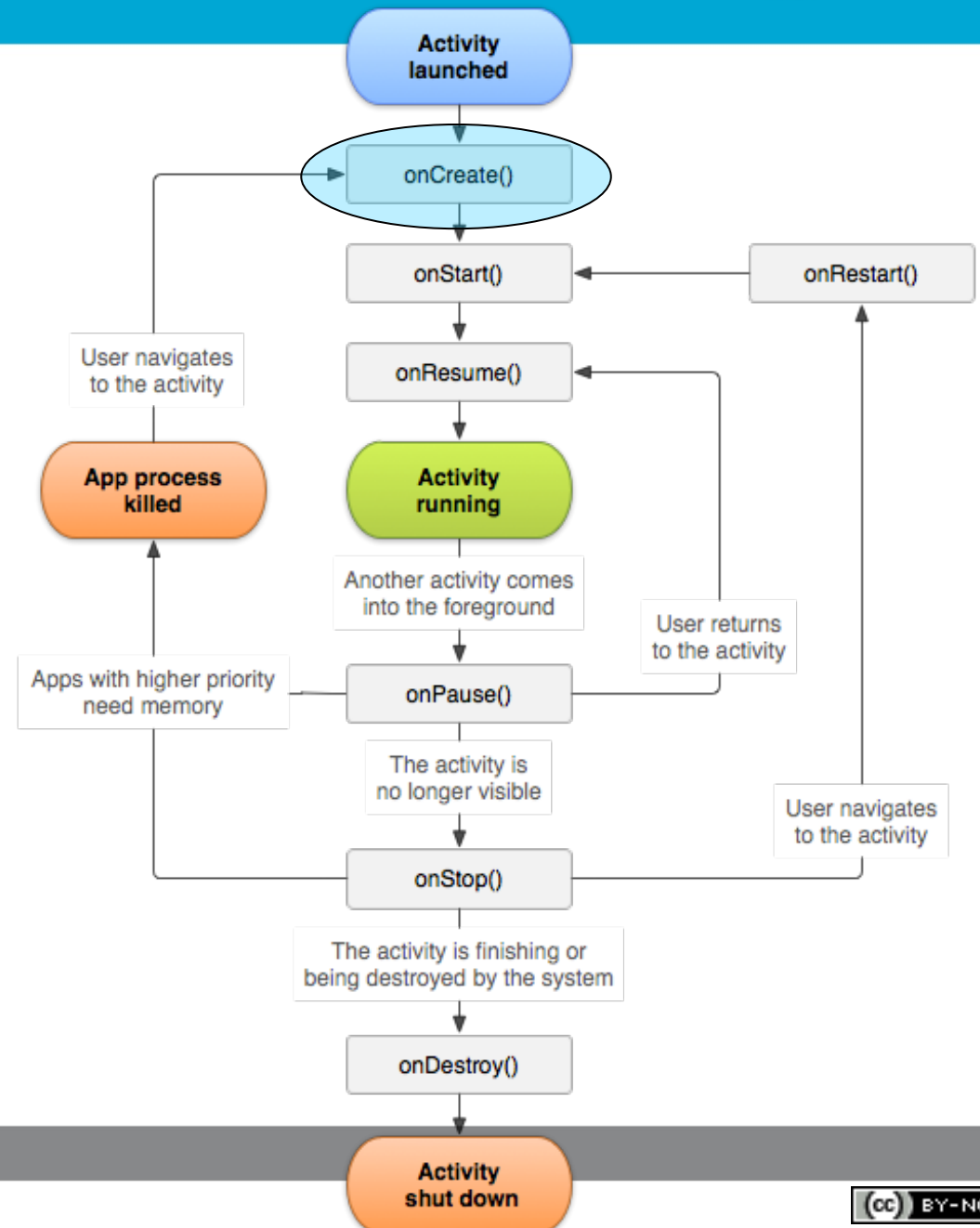
# Démo : Qu'est ce que cela a généré ?

- MainActivity.java
  - Activité initiale avec son « constructeur »
    - *public void onCreate(Bundle savedInstanceState) {...}*
- res/values/strings.xml
  - Chaînes de caractères constantes utilisables par le programme
- res/layout/main.xml
  - Descripteur de l'interface graphique
- res/mipmap/ic\_launcher.png
  - Icône de l'application en multiples résolutions
- AndroidManifest.xml
  - Description de l'application



# Cycle de vie d'une activité Android

- Programme = 1 Activité principale
- 1 Activité  $\approx$  1 écran
- onCreate :
  - *remplace le main*
  - *méthode à redéfinir*
- 1 activité a le focus  
*Si plusieurs activités, alors elles s'empilent et seule la dernière est visible.*



@Override

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
}
```

- **savedInstanceState** : informations sur l'état mémorisées lors d'un onPause() ou onStop().
- **setContentView** : choisit le XML d'habillage de l'activité



# Construction d'une IHM simple

membre de UNIVERSITÉ DE LYON

LYON  
**CPE**  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON

# Interfaces graphiques : 2 parties

## Construire le visuel



## Interagir

```
package oc.mooc.demos.myapplication;

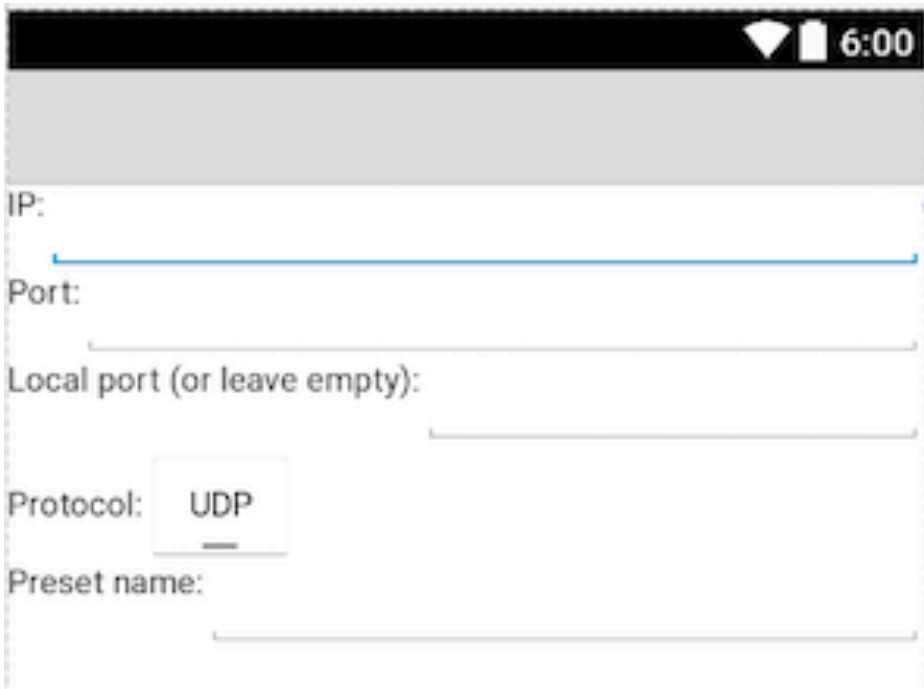
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view DragEvent;
import android.view MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    private Button btn;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btn = (Button)findViewById(R.id.clearbtn);
        btn.setText("Clear");
        btn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                effaceur((Button)v);
            }
        });
        btn.setOnDragListener(new View.OnDragListener() {
            @Override
            public boolean onDrag(View v, DragEvent event) {
                return false;
            }
        });
    }
}
```

# IHM : construire le visuel *XML*



The screenshot shows an Android application interface with a status bar at the top displaying signal strength, battery, and time (6:00). The interface contains several input fields and a dropdown menu:

- IP:** A text input field.
- Port:** A text input field.
- Local port (or leave empty):** A text input field.
- Protocol:** A dropdown menu currently showing "UDP".
- Preset name:** A text input field.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:text="IP:" />

        <EditText
            android:id="@+id/IP"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_alignParentTop="true"
            android:layout_toRightOf="@+id/textView1">

            <requestFocus />
        </EditText>

    </RelativeLayout>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
```

## Widgets

Zone de saisie de texte ....

**BUTTON**



**CheckBox**

## Structure de la vue

- Conteneurs :
  - « Visibles »
  - Dédiés

- RadioGroup
- ListView
- GridView
- ExpandableListView
- ScrollView
- HorizontalScrollView
- TabHost
- WebView
- SearchView

- Gabarits (Layout)
  - Invisible
  - ConstraintLayout / LinearLayout

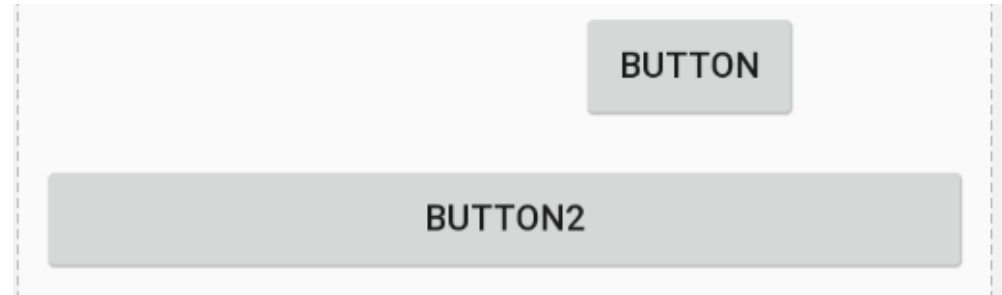
- ConstraintLayout
- GridLayout
- FrameLayout
- LinearLayout (horizontal)
- LinearLayout (vertical)
- RelativeLayout
- TableLayout
- TableRow



# Paramètre obligatoire : taille

- Principalement :
  - wrap\_content
  - match\_parent (fill\_parent)
  - Taille en pixels

```
<Button android:text="Button"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"/>
```



```
<Button android:text="Button2"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"/>
```

# Paramètre important : identifiant

- Propriété « id » :  
    <Button android:id="@+id/myButton" .... />
- Usages :
  - Placement des objets en relatif
  - Interaction Java - XML

- ConstraintLayout :
  - Conteneur d'organisation « libre » d'objets
- LinearLayout
  - Conteneur pour organiser sur une ligne ou colonne
  - Paramètre *android:orientation* obligatoire (horizontal / vertical)
- TextView
  - Affichage d'un texte
- Button :
  - Bouton
- EditText
  - Champ de saisie mono-ligne d'une chaîne de caractères

# Exemple de structuration d'un visuel

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent">
```

```
<TextView
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="Ceci est un texte affiché"
```

```
    android:id="@+id/monIdentifiant"/>
```

```
<TextView
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="@string/stringDemandee"/>
```

```
</LinearLayout>
```

# Comment interagir avec son IHM ?

1. Récupérer une référence vers un objet graphique
2. Associer un code à un événement

# Récupérer une référence vers un objet graphique

- Nommage des entités XML :
  - `<Button android:id="@+id/myButton" .... />`
- Méthode : `View findViewById(<identification objet>)`
  - Fournie par Activity
  - Renvoie l'adresse d'un objet graphique
- Classe R :
  - Générée automatiquement
  - Permet de désigner des ressources
    - *R.layout(...)* : accès aux layouts
    - *R.drawable(...)* : accès aux images
    - *R.id(...)* : accès aux objets de l'IHM



# Récupérer une référence vers un objet graphique

```
private Button btn;  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    btn = (Button)findViewById(R.id.btnStart);  
    btn.setText("Texte, change toi !");  
  
}
```

- Objectif :
  - Associer une méthode à un événement
  - Quand l'événement survient le système doit appeler la méthode
- Comment fournir une méthode en paramètre ?
  - La placer dans un objet (un écouteur)
  - Type de l'écouteur : `OnClickListener`
  - Méthode à redéfinir : `public void onClick(View v)`
  - Setter à utiliser : `setOnClickListener(...)`

# Exemple d'association action / événement

```
private Button btn;  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    btn = (Button)findViewById(R.id.btnStart);  
    btn.setOnClickListener(new OnClickListener() {  
        public void onClick(View v) {  
            out.setText(in1.getText().toString());  
        }  
    });  
}
```

- Exercice 1
- Configuration de la cible d'exécution (vos téléphones)
  - Activer le mode développeur du téléphone
    - *Menu préférences -> À propos du tel*
    - *7-8 clics sur numéro de build*
    - *Menu préférences -> Options développeur*
    - *Activer le débogage USB*
    - *Brancher le tel à l'ordi*
    - *(Possible demande d'acceptation de communication depuis le tel)*
  - Téléverser :
    - *cliquer sur « run »*
    - *choisir la cible*



# Utilisation des sondes

membre de  UNIVERSITÉ DE LYON

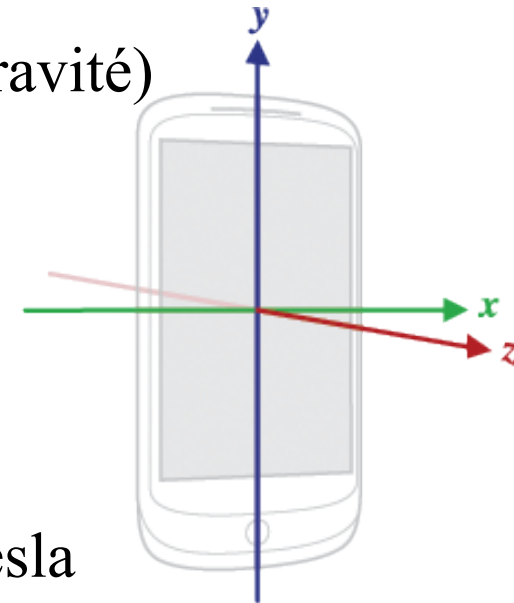
  
**CPE**  
LYON  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON

# Des capteurs ?

- 4 familles :
  - environnementaux (température, luminosité, ...)
  - de position (boussole, géolocalisation, ...)
  - de mouvement (accéléromètre, gyroscope, ...)
  - corporels (rythme cardiaque, nombre de pas, ...)
- Accessible via :
  - SensorManager
- Exception :
  - Géolocalisation
- Si empiète sur vie privée :
  - Permissions utilisateur nécessaires



- ACCELEROMETER
  - Capteur **physique** ou **simulé**
  - Mesure de l'accélération linéaire, en  $\text{m.s}^{-2}$  (dont la gravité)
- GYROSCOPE
  - Capteur **physique** ou **simulé**
  - Mesure de la vitesse angulaire, en  $\text{rad.s}^{-1}$
- MAGNETIC\_FIELD
  - Capteur **physique**
  - Mesure le champs magnétique ambiant, en micro-Tesla
- ROTATION\_VECTOR (*API 9*) :
  - Capteur **virtuel** à partir de plusieurs capteurs :
    - *Mesure des champs magnétiques*
    - *Accélérations*
  - Obtenir des infos d'orientation tangentielle à la terre en notre position



(Source : <https://developer.android.com/reference/android/hardware/SensorEvent.html>)

# Tous capteurs pas toujours dispos

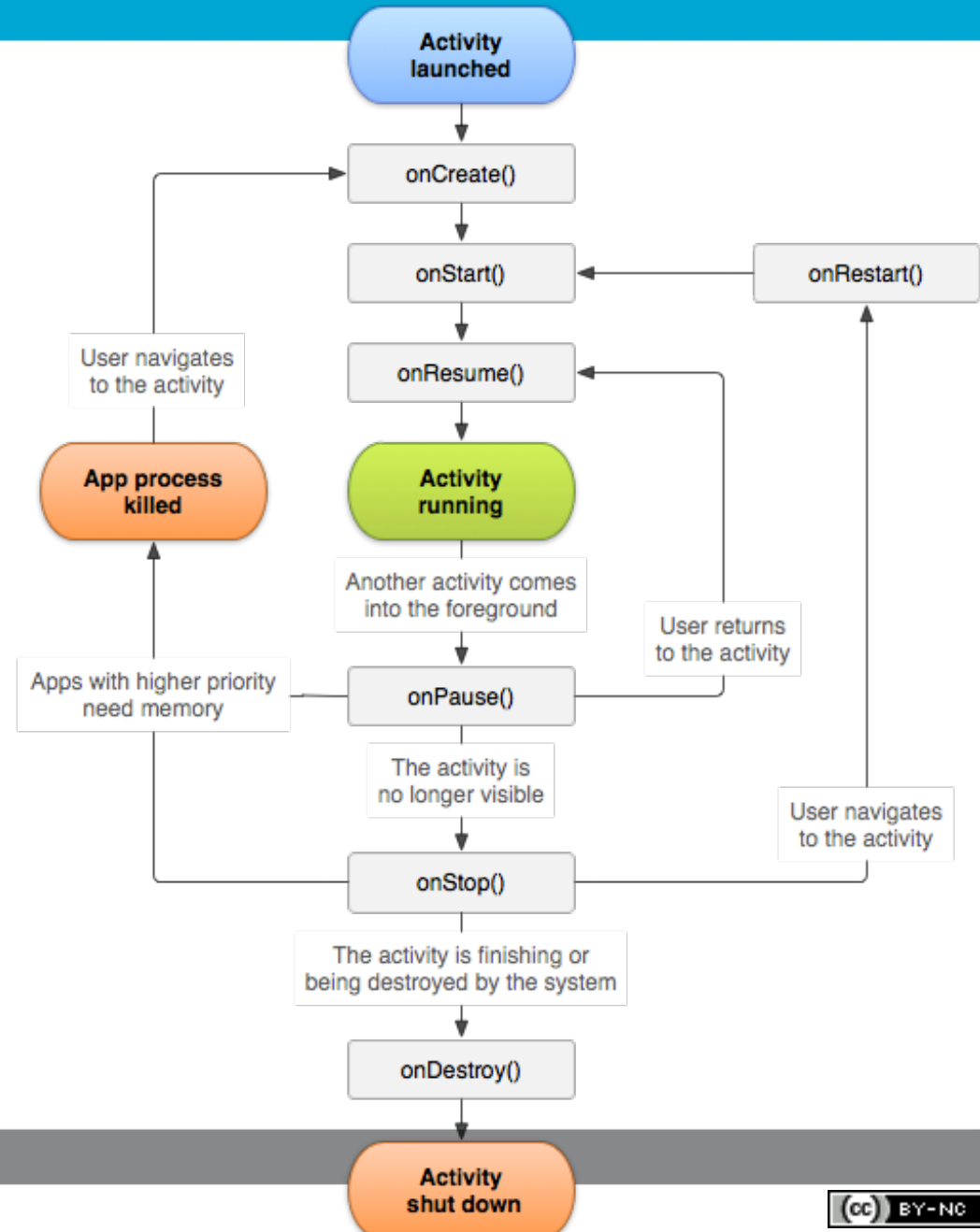
- ACCELEROMETER
  - Physique ou Virtuel
- LINEAR\_ACCELERATION
  - Virtuel : ACCELEROMETER sans la gravité
- GRAVITY
  - Virtuel : que la gravité

$$\text{ACCELEROMETER} = \text{LINEAR\_ACCELERATION} + \text{GRAVITY}$$

- Pas d'attente active
  - *(Pas de boucle de lecture de l'état du capteur)*
- Enregistrement de d'un écouteur auprès du système
  - Le système notifie régulièrement l'écouteur
  - Être notifié : fournir une méthode appelée par le système
    - *(Répondre à l'interface `SensorEventListener`)*
- Penser à dé-enregistrer l'application

# Cycle de vie d'une activité (rappel)

- onCreate : méthode à redéfinir pour le lancement  
*≈ le main de l'activité*
- D'autres méthodes existent, que l'on peut redéfinir
  - onResume : au lancement ET après la réapparition
  - onPause : masquage



# Exemple concret

```
package nst.exemple.capteurs;
public class SensorsActivity extends Activity implements SensorEventListener {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout....);
    }

    private SensorManager sm;
    protected void onResume() {
        super.onResume();
        sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        sm.registerListener(this, sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                           SensorManager.SENSOR_DELAY_UI);
    }
    protected void onPause() {
        super.onPause();
        sm.unregisterListener(this);
        sm = null;
    }
    public void onAccuracyChanged(Sensor sensor, int accuracy) {}
    public void onSensorChanged(SensorEvent event) {...}
}
```

# À vous !

- Exo 2



# Programmation réseau en Android

*Focus sur UDP*

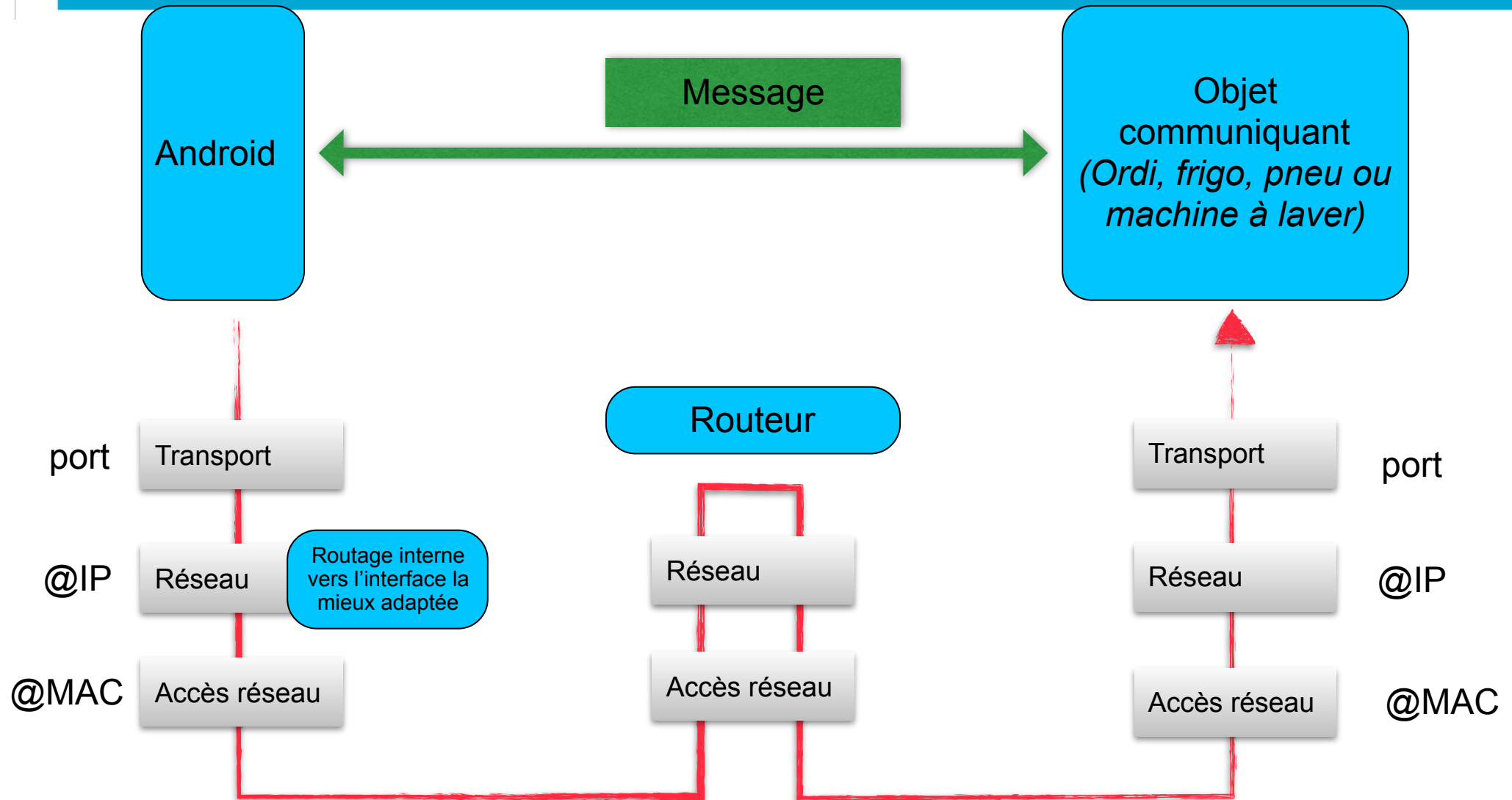
membre de UNIVERSITÉ DE LYON

LYON  
**CPE**  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON

- Principe général
  - UDP en Java
  - Primitives
  - Exceptions
- Complexité 1 : Permissions
- Complexité 2 : Processus séparé



# Réseaux : quelques rappels



| Éléments | Rôle  | Objet Java<br><i>(Cas d'UDP)</i>            |
|----------|---|---|
| @IP      | Identification de la machine                              | InetAddress                                 |
| Port     | Identification du service                                 | int   |
| Packet   | Message à envoyer   | DatagramPacket<br><i>(Données + entête)</i> |
| Socket   | Point d'accès au réseau<br><i>(Identifié par IP+Port)</i> | DatagramSocket                              |

# Instanciation et exceptions

## *Quelques remarques*

- Tout accès au réseau peut générer des erreurs
  - Ex : Permission non demandée, port déjà utilisé, connexion refusée, hôte non atteignable, etc
  - Prendre en compte les IOException à chaque fois
- Socket :
  - Penser à libérer la ressource
  - Réserver/libérer dans onResume/onPause ?
- InetAddress :
  - Classe sans constructeur, mais avec une Factory
  - Ex : InetAddress **address** = InetAddress.getByName("192.168.0.254");

- Initialisation du réseau

```
try { // Choix du port local laissé à la discrétion de la plateforme
    UDPocket = new DatagramSocket();
    address = InetAddress.getByName("192.168.0.254");
} catch (IOException e) {
    e.printStackTrace();
}
```

- Émission d'un message

```
try{
    byte[] data = {42}; // Pour simplifier, nous n'envoyons qu'1 octet.
    DatagramPacket packet = new DatagramPacket(data, data.length, address, port);
    UDPocket.send(packet);
} catch (IOException e) { e.printStackTrace(); }
```

- Réception d'un message (*devrait être dans un processus à part*)

```
try{
    byte[] data = new byte [1024]; // Espace de réception des données.
    DatagramPacket packet = new DatagramPacket(data, data.length);
    UDPocket.receive(packet);
    int size = packet.getLength();
} catch (IOException e) { e.printStackTrace(); }
```

- Certaines permissions doivent être demandées explicitement
  - Accès au réseau
  - Accès à la localisation
  - ...
- Demandées dans le Manifest
  - *Depuis Android 6 : **ET** au runtime*
    - <https://openclassrooms.com/fr/courses/4428411-developpez-des-applications-android-connectees/4559221-configurez-les-dependances-et-permissions-dans-le-manifest>
- Exemple pour l'accès réseau :

```
<uses-permission android:name="android.permission.INTERNET" />
```

# Complexité 2 : Problème d'expérience utilisateur

- Pas de réception ou d'émission autorisée de manière synchrone avec l'IHM.
- Nécessité de créer des processus pour les différentes tâches
- AsyncTask : très pratique pour les réceptions
- Pour les émissions, plusieurs alternatives :
  - *Créer un nouveau processus pour chaque émission*
  - *Rester en API 8 (pas de thread nécessaire)*
  - *Utiliser une file bloquante (BlockingQueue, ... )*

# Exemple simplifié en UDP (Émission)

- Émission d'un message

```
(new Thread() {  
    public void run() {  
        try{  
            byte[] data = {42};  
            DatagramPacket packet = new DatagramPacket(data, data.length, address, port);  
            UDPSocket.send(packet);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}).start();
```

# Exemple simplifié en UDP

## (Réception)

- Réception d'un message

```
private class ReceiverTask extends AsyncTask<Void, byte[], Void> {
    protected Void doInBackground(Void... rien) {
        while(true){
            byte[] data = new byte [1024]; // Espace de réception des données.
            DatagramPacket packet = new DatagramPacket(data, data.length);
            UDPSocket.receive(packet);
            int size = packet.getLength();
            publishProgress(java.util.Arrays.copyOf(data, size));
        }
    }

    protected void onProgressUpdate(byte[]... data) {
        ... // Appelé de manière asynchrone, mais synchronisé avec l'UI
    }
}

(new ReceiverTask()).execute();
```



- Exo 3
- Pour vos tests :
  - Un serveur écho tourne sur l'ordi prof
  - Vidéo projection des messages reçus
- À fournir :
  - SSID
  - IP du prof
  - port du serveur echo



# Faire une vue custom

membre de  UNIVERSITÉ DE LYON

  
**CPE**  
LYON  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON

# Créer une vue perso 1/3

- Créer une nouvelle classe, héritant de `android.view.View`

```
package ot.nst.tp2;  
//...  
public class TraceCourbe extends View {
```

- Intégrer la vue dans l'IHM

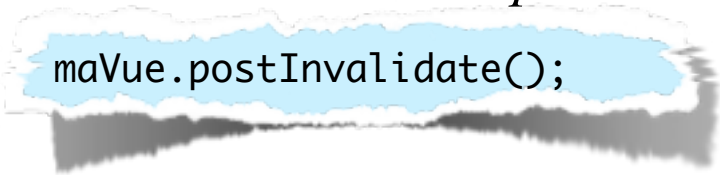
```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent">  
  
    <ot.nst.tp2.TraceCourbe  
        android:id="@+id/tracageCourbe"  
        android:layout_width="fill_parent"  
        android:layout_height="fill_parent" />  
  
</LinearLayout>
```

- Redéfinir la méthode onDraw

```
package ot.nst.tp2;
//...
public class TraceCourbe extends View {
    //...
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        Paint crayon=new Paint();
        crayon.setColor(Color.BLUE);
        canvas.drawText("Alors !? Alors!? Ça marche ?", 30, 30, crayon);
    }
}
```

- Classes utilisées :

- Paint :
  - *Description du « crayon » à utiliser pour dessiner*
- Canvas :
  - *Zone de dessin*
  - *Fournit les primitives de dessins*

- Pourquoi ça marche ? (ou pas)
  - À chaque re-calculation de l'affichage :
    - OS appelle la méthode *onDraw* de chaque Vue
    - Canvas donné en paramètre dimensionné pour chaque vue
    - L'OS gère lui même les superpositions
  - Quand est re-calculé l'affichage ?
    - À la première apparition et à chaque ré-apparition d'une vue
    - Quand on le demande explicitement :

```
maVue.postInvalidate();
```
  - Quand forcer le rafraichissement ?
    - JAMAIS dans *onDraw* (recursion infinie)
    - Lorsque une valeur d'entrée a changé (SeekBar, valeur saisie, etc.)





# Orienté composant

membre de UNIVERSITÉ DE LYON

LYON  
**CPE**  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON

# Orienté composant : appel explicite d'une activité

- Exemple d'appel d'une activité nécessitant un paramètre (Para) :

```
Intent intent = new Intent(this, ConnectedActivity.class);  
intent.putExtra("fr.insa.Para", ValeurATransmettre);  
startActivity(intent);
```

- Déclaration de l'activité :

```
<activity android:name="ConnectedActivity">  
  <intent-filter>  
    <action android:name="fr.insa.START" />  
    <data android:name="fr.insa.Para" android:mimeType="text/plain" />  
  </intent-filter>  
</activity>
```

- Récupération du paramètre par l'activité appelée :

```
Bundle b = getIntent().getExtras();  
String parametre = b.get("fr.insa.Para").toString();
```

- Exemple d'appel d'une activité de recherche sur google :

```
String requete = "http://www.google.fr/search?q=" + maRequete;  
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(requete));  
startActivity(intent);
```

- Déclaration pouvant répondre à cet appel :

```
<activity android:name="GoogleSearch">  
  <intent-filter>  
    <action android:name="android.intent.action.VIEW" />  
    <data android:name="uri" android:mimeType="text/plain" />  
  </intent-filter>  
</activity>
```

Source : <http://www.pointgphone.com/tutoriel-android-introduction-intents-7779>



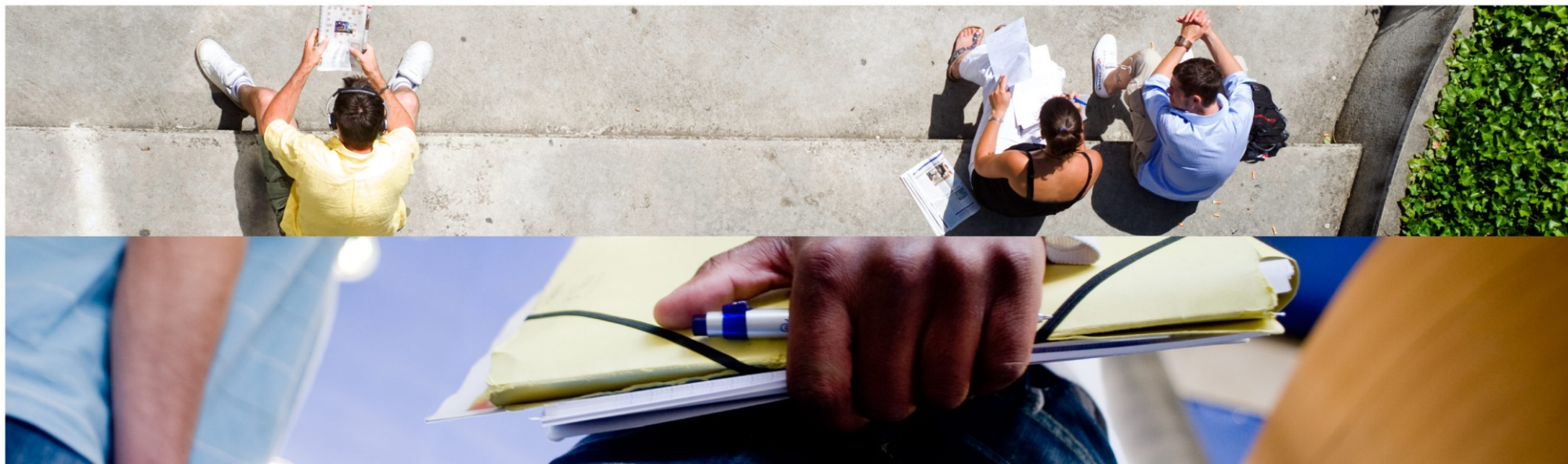
- Exemple d'appel d'une activité de recherche sur google :

```
private void pickContact() {  
    Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);  
    startActivityForResult(intent, PICK_CONTACT_REQUEST);  
}
```

- Déclaration pouvant répondre à cet appel :

```
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (resultCode == Activity.RESULT_OK && requestCode == PICK_CONTACT_REQUEST) {  
        // ....  
    }  
}
```

- Paramètre lors de l'appel d'une activité
- Communication réseau sur le 127.0.0.1
- Données statiques publiques (attention aux accès concurrents)
- Gestion des préférences d'application
- Fichier
- SQLite DB



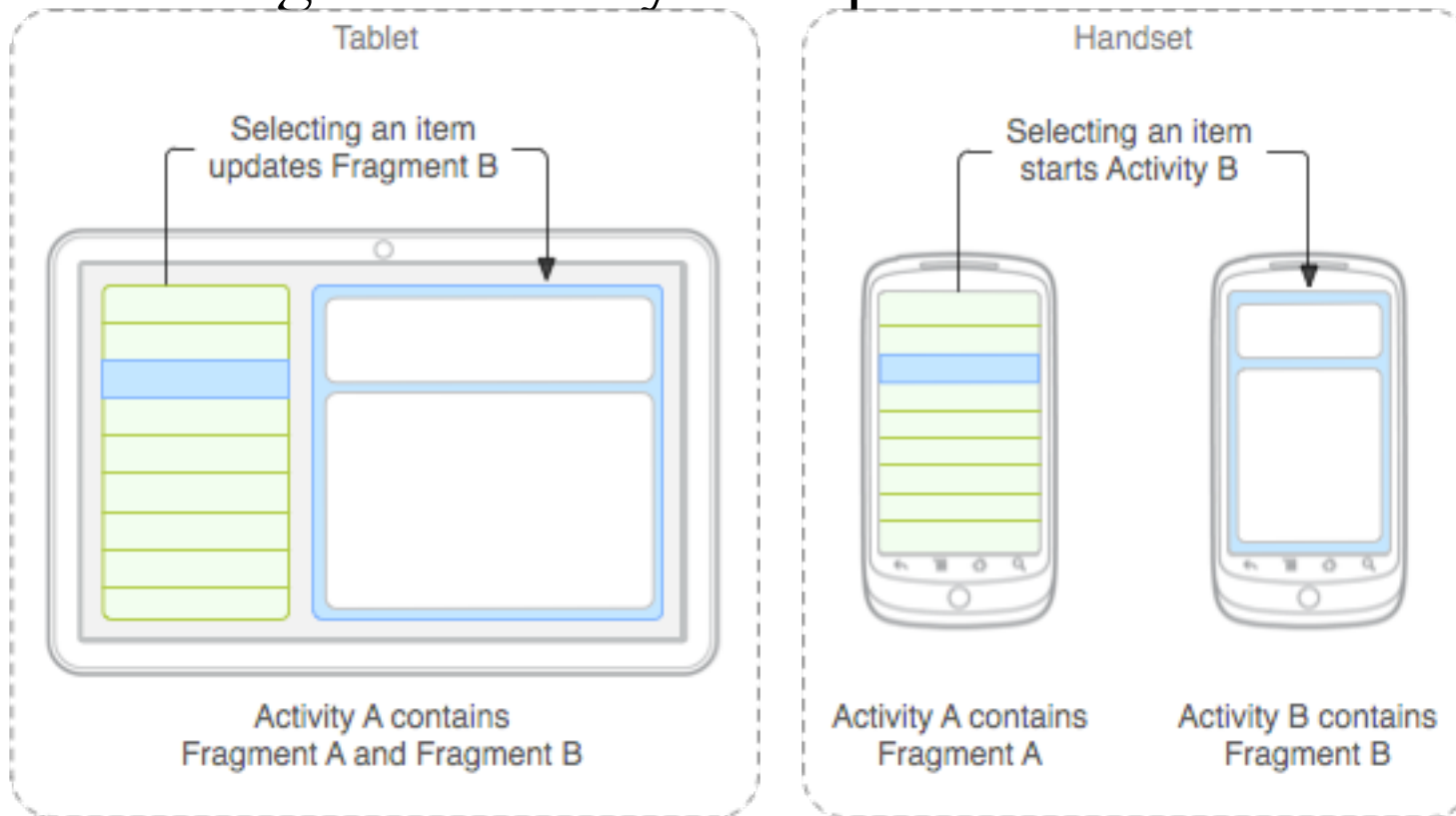
# Fragments

membre de UNIVERSITÉ DE LYON

LYON  
**CPE**  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON

# Concept de fragment

- Dépend d'une activité
- Portion d'une interface graphique
- Permet une organisation dynamique de l'IHM

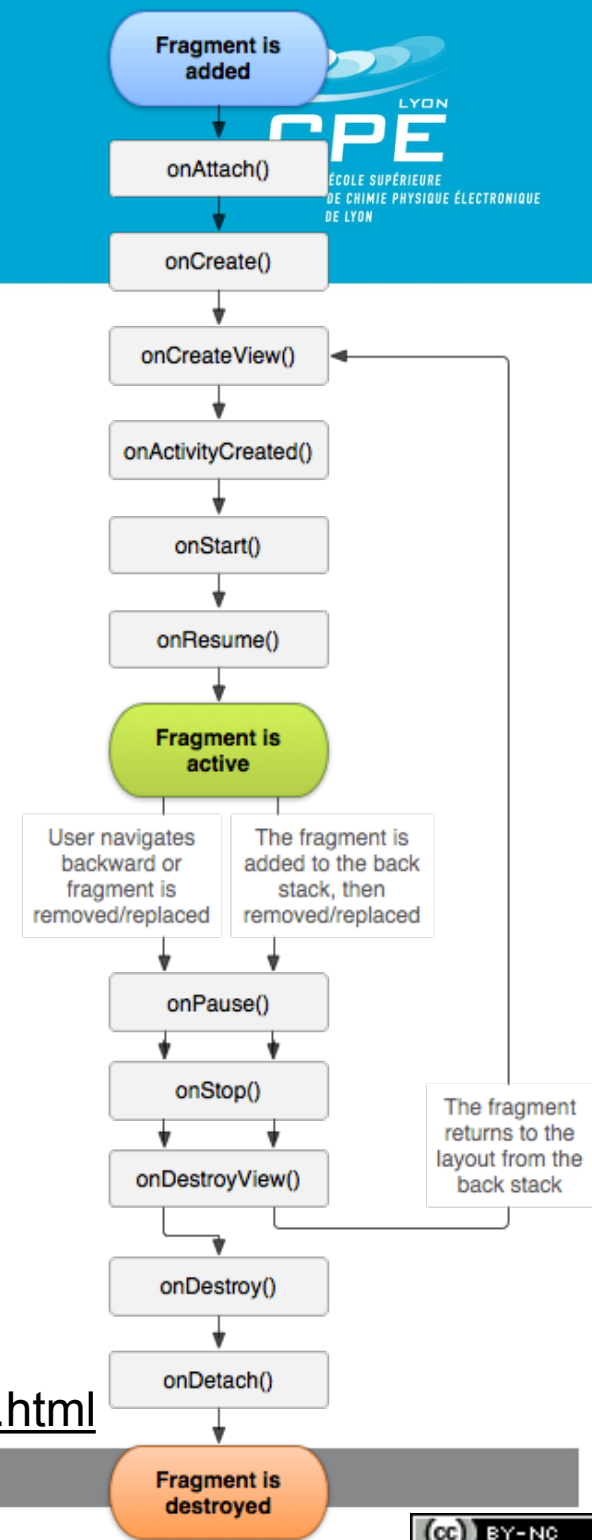


Source : <http://developer.android.com/guide/components/fragments.html>

- Par défaut :
  - 1 fragment est affiché
  - les autres sont en pause
  - Pas d'appel direct
  - Faire des appels à l'activité, qui fera le nécessaire
    - *Si le second fragment est stoppé, alors l'action doit être stockée en attente du réveil de l'autre Fragment*
    - *Permet d'éviter les liens directs vers des occurrences qui peuvent disparaître*

# Cycle de vie d'un fragment

- onCreateView : initialisation et habillage
- Suivant le FragmentManager, peut être détruit à chaque fois que masqué



Source : <http://developer.android.com/guide/components/fragments.html>





# Demande de permission au runtime

membre de  UNIVERSITÉ DE LYON

  
**CPE**  
LYON  
ÉCOLE SUPÉRIEURE  
DE CHIMIE PHYSIQUE ÉLECTRONIQUE  
DE LYON

# Demande de permission au runtime

## *(Principe)*

Faire une action avec permission :

Ai-je déjà le droit ?

Oui : faire l'action

Non : Déjà refusé par le passé ?

Oui : argumenter (ou ne rien faire)

Non : demander la permission → asynchrone

retour de demande de permission :

Suivant l'action en cours :

Si autorisation(s) accordée(s) :

Faire l'action



# Demande de permission au runtime

## *(Code de la demande)*

```
final static int MY_PERMISSION_REQUEST_CODE=1; // valeur arbitraire
final static String MY_PERMISSION_NAME=Manifest.permission./*...*/;

public void myActionWithPermission(){
    if (ActivityCompat.checkSelfPermission(MainActivity.this,
        MY_PERMISSION_NAME) == PackageManager.PERMISSION_GRANTED) {
        Toast.makeText(MainActivity.this,"Go", Toast.LENGTH_LONG).show();
    } else {
        if (ActivityCompat.shouldShowRequestPermissionRationale(this,
            MY_PERMISSION)) {
            // Affichage d'un dialogue de justification ou rien
        } else {
            ActivityCompat.requestPermissions(MainActivity.this,
                new String[]{MY_PERMISSION_NAME},
                MY_PERMISSION_REQUEST_CODE);
        }
    }
}
```

# Demande de permission au runtime (Callback)

```
/* ActivityCompat.requestPermissions(MainActivity.this,  
                                     new String[]{MY_PERMISSION_NAME},  
                                     MY_PERMISSION_REQUEST_CODE); */  
  
@Override  
public void onRequestPermissionsResult(int requestCode,  
                                     String permissions[],  
                                     int[] grantResults) {  
    switch (requestCode) {  
        case MY_PERMISSION_REQUEST_CODE: {  
            if (grantResults.length > 0 &&  
                grantResults[0] == PackageManager.PERMISSION_GRANTED) {  
                myActionWithPermission(); // On a l'autorisation.  
            } else { /* On abandonne notre super fonctionnalité. */  
            }  
            return;  
        }  
    }  
}
```