



IN01

# Programmation Android

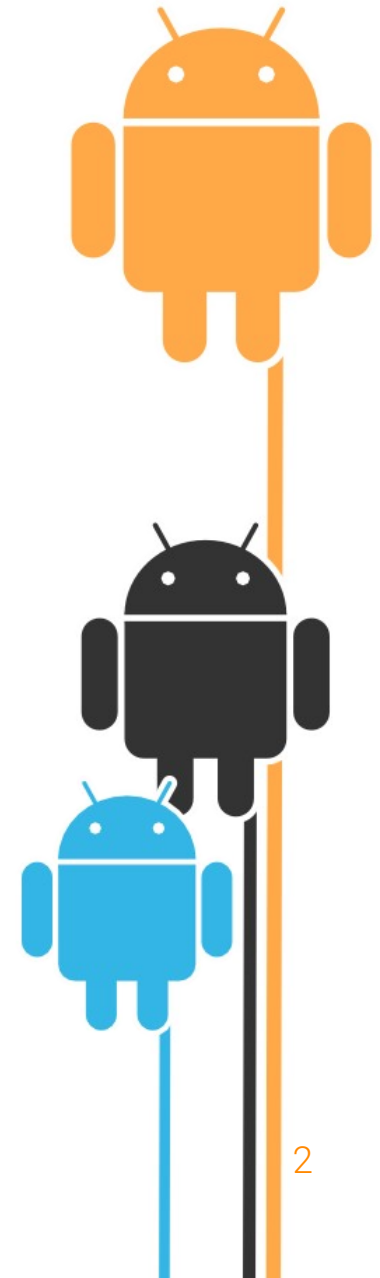
07 – Techniques avancées

Yann Caron

**ENSG**  
Géomatique

# Sommaire - Séance 07

- Programmation avancée
  - ➔ Capteurs
  - ➔ Optimisations
  - ➔ Tests unitaires
  - ➔ Concurrency
- IHM avancées
  - ➔ Vues personnalisées
  - ➔ Fragments
- Autres
  - ➔ Stratégies et alternatives
  - ➔ Jeux vidéo mobiles



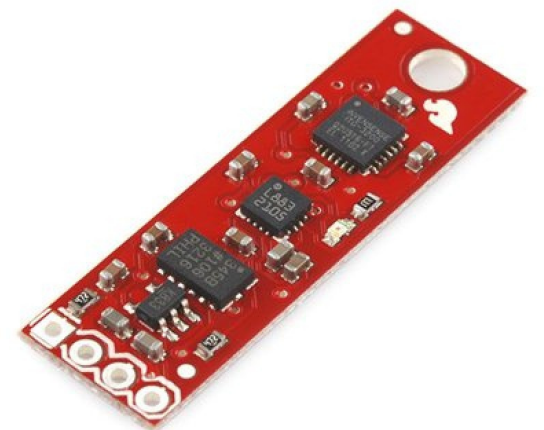
# IN01 – Séance 07

## Programmation avancée Capteurs



# Généralités

- La miniaturisation et l'industrialisation rendent aujourd'hui accessibles des capteurs précis au grand public
- La majorité des appareils Android embarquent au moins un accéléromètre, un gyroscope et un magnétomètre
- Les trois tiennent sur ce circuit :



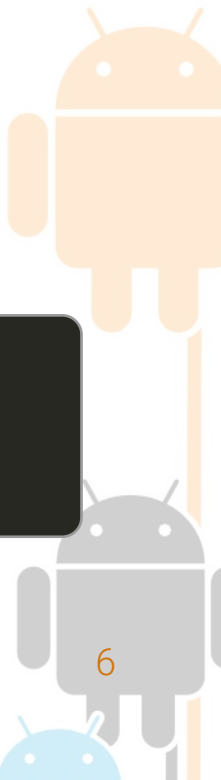
# Différents capteurs Android

|                     |   |                  |  |   |
|---------------------|---|------------------|--|---|
| TYPE_ACCELEROMETER  | 3 | m/s <sup>2</sup> | Mesure de l'accélération (gravité incluse)                   | [0] axe x [1] axe y [2] axe z   |
| TYPE_GYROSCOPE      | 3 | Rad / s          | Mesure la rotation en termes de vitesse autour de chaque axe | [0] vitesse x<br>[1] vitesse y<br>[2] vitesse z                             |
| TYPE_LIGHT          | 1 | Lux              | Mesure de la luminosité                                      | [0]valeur   |
| TYPE_MAGNETIC_FIELD | 3 | μTesla           | Mesure du champ magnétique                                   | [0] axe x [1] axe y [2] axe z   |
| TYPE_ORIENTATION    | 3 | degrés           | Mesure l'angle entre le nord magnétique                      | [0] Azimut y / nord<br>[1] Rotation x (-180,180)<br>[2] Rotation y (-90,90) |
| TYPE_PRESSURE       | 1 | KPas             | Mesure la pression   | [0]valeur   |
| TYPE_PROXIMITY      | 1 | mètre            | Mesure la distance entre l'appareil et un objet cible        | [0]valeur   |
| TYPE_TEMPERATURE    | 1 | Celsius          | Mesure la température  | [0]valeur   |

# Présence du capteur

- On peut interdire (ou pas) l'utilisation de l'application aux appareils qui ne possèdent pas un certain capteur
- Utile si votre application ne peut pas fonctionner sans de façon correcte
- Dans le manifest (toujours !!) :

```
<uses-feature  
  android:name="android.hardware.sensor.accelerometer"  
  android:required="true" />
```



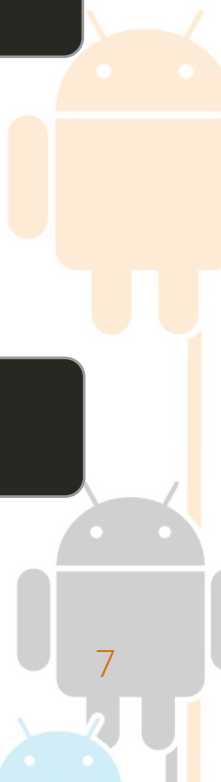
# Sensor Manager

- Pour accéder aux senseurs, on appelle une méthode de l'Activity qui renvoie un objet de type `SensorManager`

```
SensorManager sensorManager =  
    (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

- On peut récupérer la liste des capteurs disponibles sur l'appareil

```
ArrayList<Sensor> liste = (ArrayList<Sensor>)  
    sensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
```



# Obtenir une instance de capteur

- On peut maintenant obtenir une instance du capteur souhaité

```
Sensor accelerometer =  
    sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

- Attention !! Si le capteur n'est pas disponible, l'objet renvoie la valeur **null**. Si le capteur n'est pas requis, il faut tester et gérer s'il est absent.





# Évènement

- Les capteurs fonctionnent aussi sur un modèle évènementiel

```
final SensorEventListener mSensorEventListener = new SensorEventListener() {  
    public void onAccuracyChanged(Sensor sensor, int accuracy) {  
        // Que faire en cas de changement de précision ?  
    }  
  
    public void onSensorChanged(SensorEvent sensorEvent) {  
        // Que faire en cas d'évènements sur le capteur ?  
    }  
};
```

Si la précision  
a changé

Si les valeurs  
ont changé

# Évènement

- Attention, il est préférable d'enregistrer l'évènement lorsque l'application est active et de le désenregistrer lorsque celle-ci est inactive
- Sinon, il continue de s'exécuter

```
@Override
protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(mSensorEventListener, mAccelerometer,
        SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(mSensorEventListener, mAccelerometer);
}
```

# Évènement – Valeurs

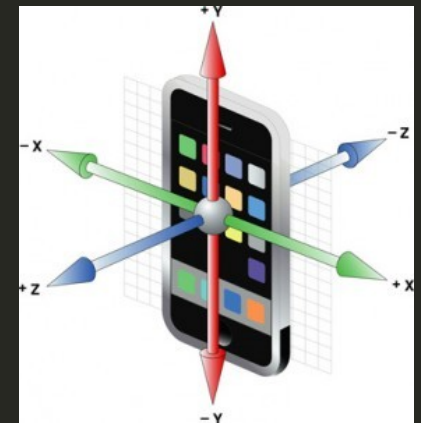
- L'évènement du capteur renvoie un tableau de valeurs float
- Leur nombre et leur signification varient selon les capteurs (voir tableau précédent)

```
public void onSensorChanged(SensorEvent sensorEvent) {  
    float[] values = sensorEvent.values;  
  
    Log.d("Sensors", "Accelération sur l'axe x : " + values[0]);  
    Log.d("Sensors", "Accelération sur l'axe y : " + values[1]);  
    Log.d("Sensors", "Accelération sur l'axe z : " + values[2]);  
}
```

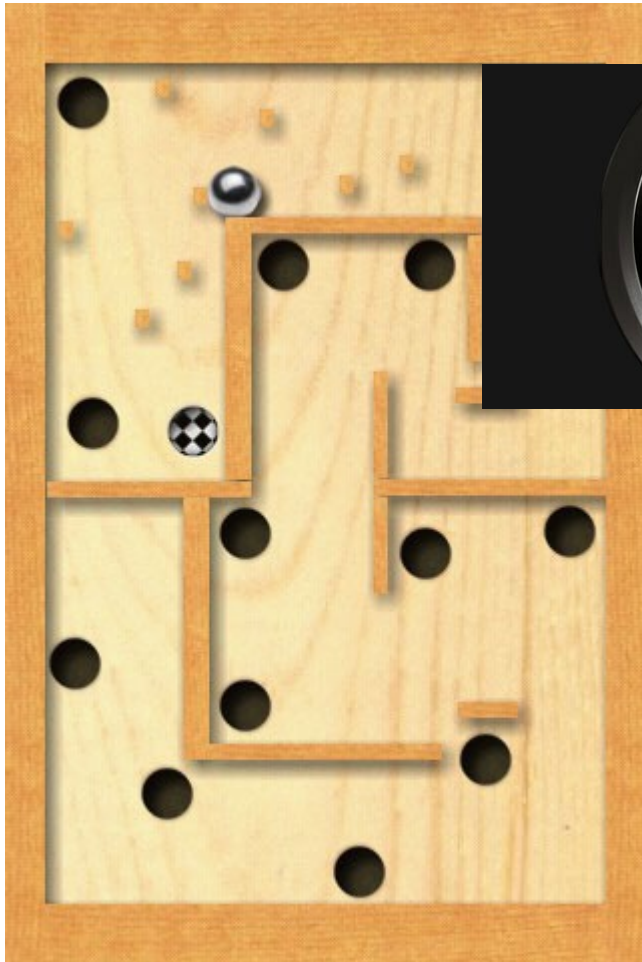
# Rotation

- Doit être calculée avec les valeurs de l'accéléromètre et du magnétomètre combinées à l'aide de formules complexes
- Deux méthodes existent pour faciliter ce calcul :  
`getRotationMatrix()` et `getOrientation()`

```
float[] values = new float[3];  
float[] R = new float[9];  
  
SensorManager.getRotationMatrix(R, null,  
    accelerometerValues, magnetometerValues);  
  
SensorManager.getOrientation(R, values);  
  
Log.d("Sensors", "Rotation sur l'axe z : " + values[0]);  
Log.d("Sensors", "Rotation sur l'axe x : " + values[1]);  
Log.d("Sensors", "Rotation sur l'axe y : " + values[2]);
```



# Quelques applications



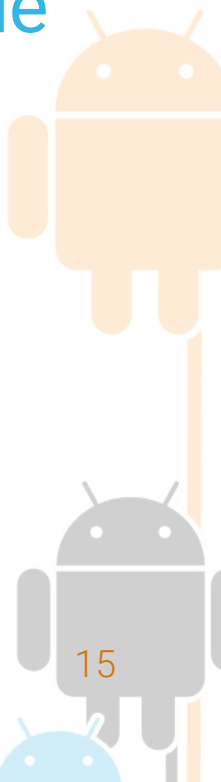
# IN01 – Séance 07

Programmation avancée  
Optimisation



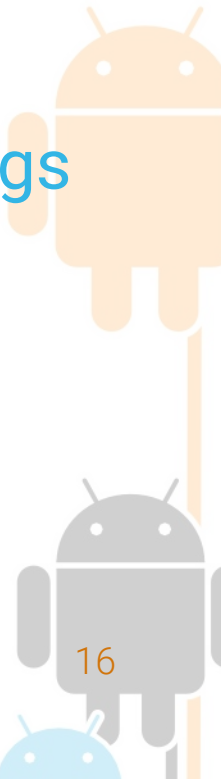
# Optimisation sous Android

- Pourquoi ??
  - ➔ Pour de meilleures performances !!
- Oui, mais pour qui ??
  - ➔ Les applications de bas niveau (un compilateur de langage comme Algoid par exemple ;-))
  - ➔ Des applications gourmandes en ressources (les jeux vidéo, les applications graphiques)



# Règles – Les objets

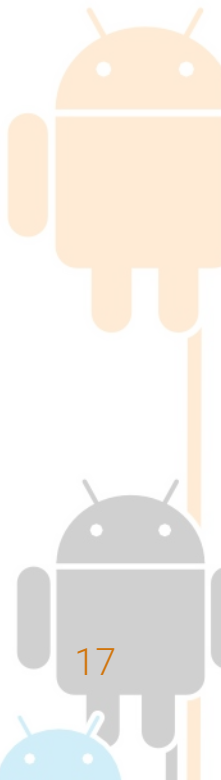
- Éviter de créer des objets non nécessaires
- En règle générale, éviter de créer trop d'objets !
  - ➔ Parce que la création d'objets c'est gourmand
  - ➔ Et la destruction encore pire : la dalvikVM bloque tout processus lors du GC (GC dit **Stop the World**)
  - ➔ Préférer les StringBuffer aux concaténations de strings à outrance
  - ➔ Un tableau d'**int** est meilleur qu'un tableau d'integer (utilisation des types primitifs)





# Règles – Les listes

- Un tableau est meilleur qu'une ArrayList, même règle que précédemment sur les types primitifs
- Une ArrayList est moins coûteuse à lire qu'une LinkedList (création d'objets entry qui seront à gérer par le GC)



# Règles – Les boucles

- Idéal :

```
int size = myArray.length);  
for (int i = 0; i < size; i ++ ) {  
}
```

La taille est évaluée à chaque itération

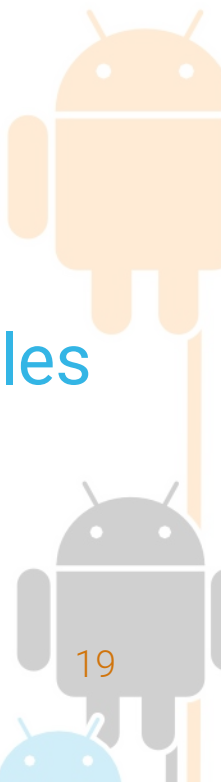
- Catastrophique :

```
Map<Integer, String> myMap = new HashMap();  
for (Entry<Integer, String> item : myMap.entrySet()) {  
}
```

Un objet Entry<Key, Value> créé à chaque itération

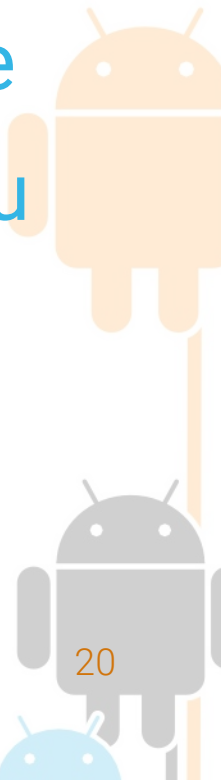
# Règles – Méthodes et variables

- Préférer les méthodes statiques aux méthodes de classe, si aucun attribut de la classe n'est nécessaire
- Utiliser `static final` pour les constantes
- Éviter les inner class, surtout si elles ne sont pas statiques
- Éviter les getter et setter (antipattern)
- Les `float` 2x plus lents que les `int`
- Les `double` sont 2x plus gourmands en mémoire que les `float`



# Règles - Conclusion

- Attention toutefois à ne pas abuser des règles d'optimisation
- Évaluer si l'application nécessite une optimisation
- Toujours mettre en relation le gain de performances (CPU/Mem) et la lisibilité du code
- Car attention ! Ces règles dégradent la qualité du code et donc la lisibilité et la réutilisabilité



# Optimisation - Outils

- Toujours mesurer les gains par des micro-benchmarks :

```
long time = System.currentTimeMillis();
for (int i=0; i<10000000; i++) {
    // le code à tester
}
long timeSpent = System.currentTimeMillis() - time;
System.out.println("Time spent " + timeSpent);
```

- Et le JIT (Just In Time Compiler) ?
- Profilers netbeans ou Eclipse sont nos amis
- Résultat : Algoid Language aussi rapide que Python est plus rapide que JavaScript sur Android :-) Pas mal pour un langage seulement interprété force brute



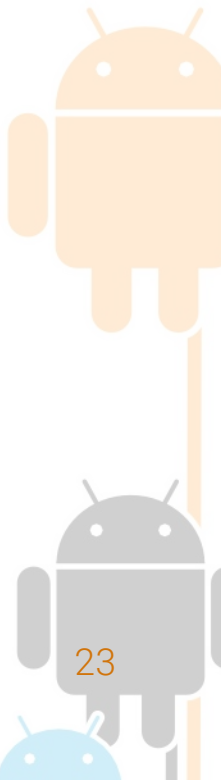
# IN01 – Séance 07

Programmation avancée  
Tests unitaires



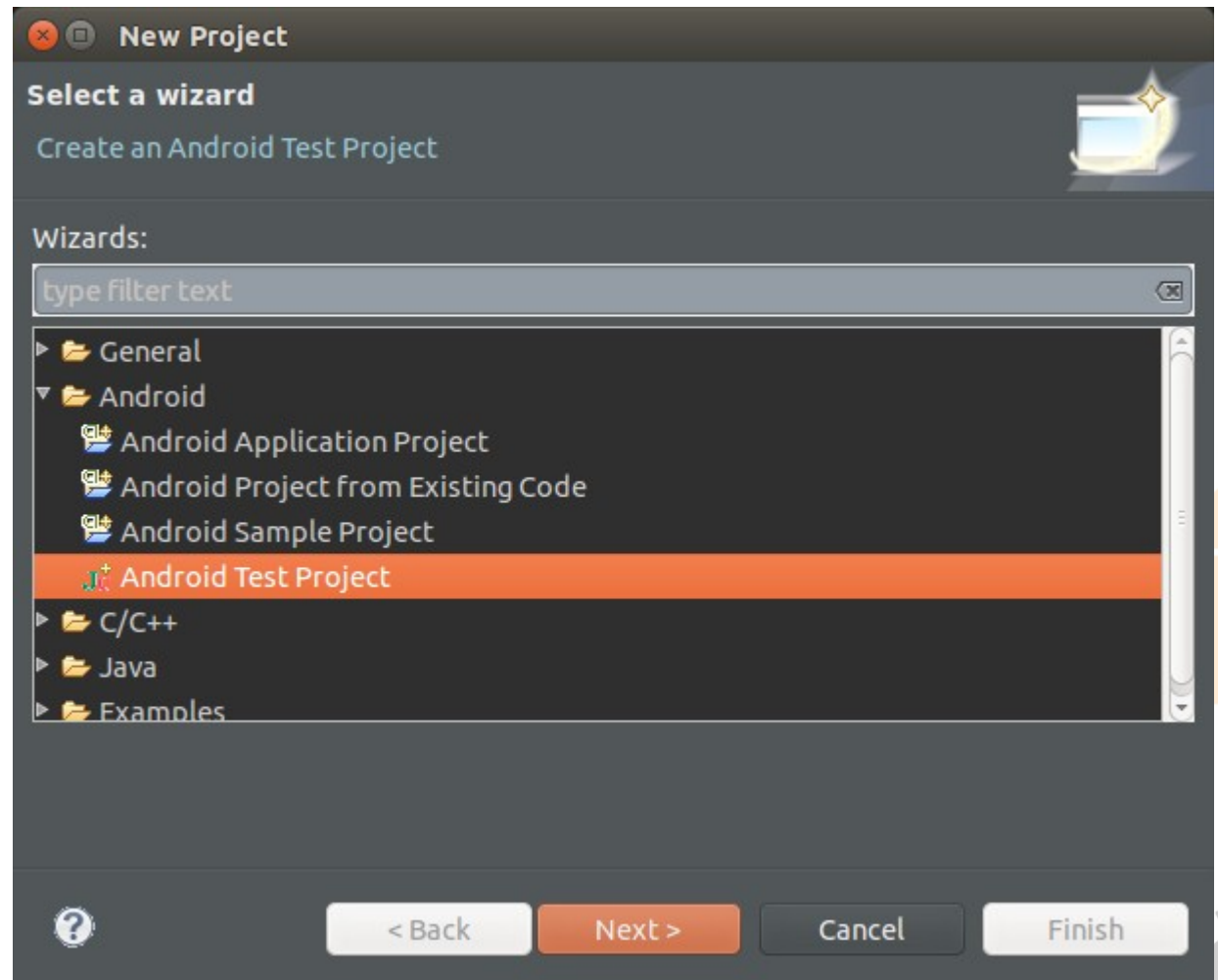
# Vue d'ensemble

- Créer des tests unitaires pour tester des apps Android, c'est possible
- Ça fonctionne avec JUnit
- On peut tester l'interface utilisateur (manipulation)



# Création du projet de test

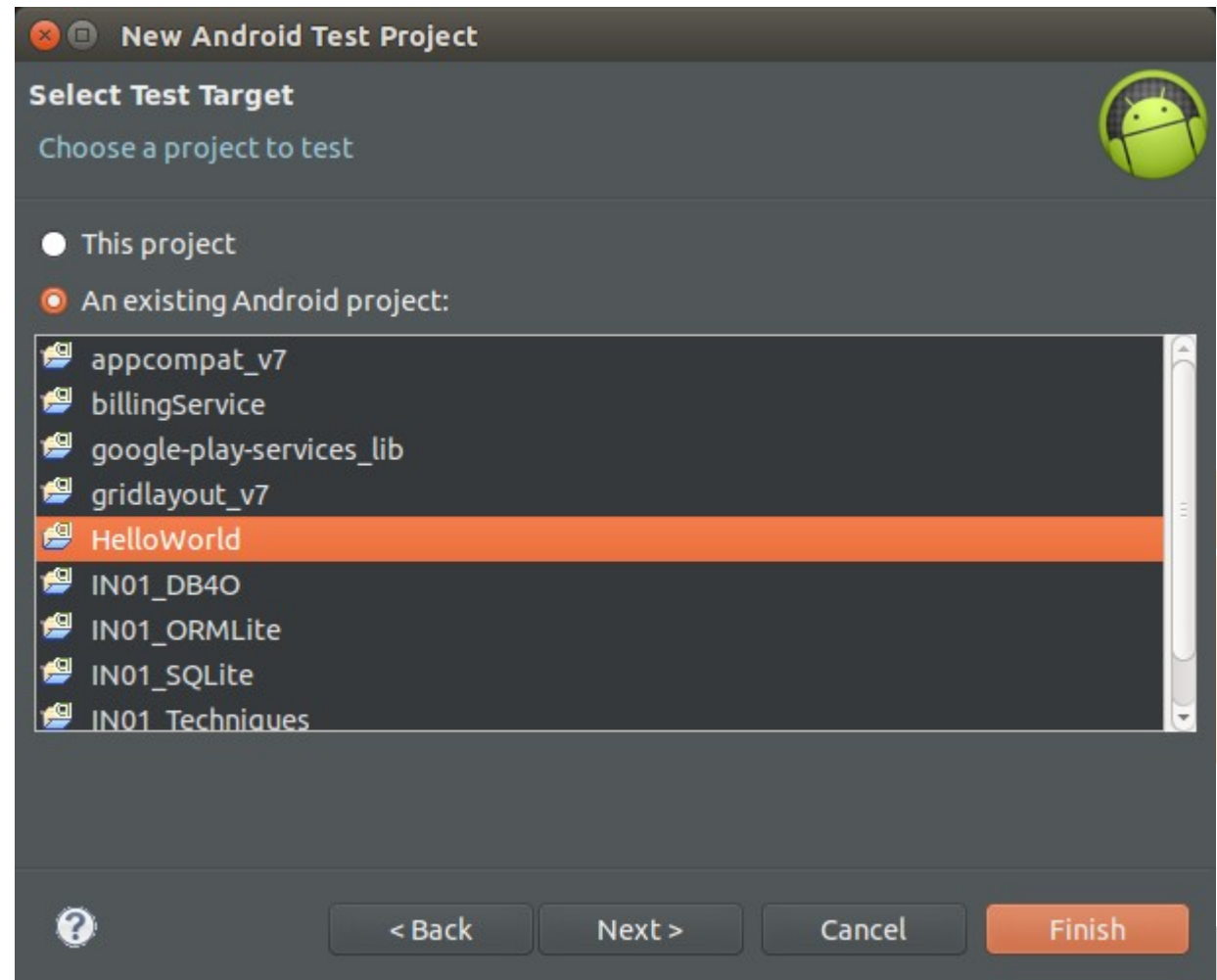
- Depuis Eclipse





# Création du projet de test

- On choisit le projet à tester




# Test Fixture

- Pour créer un test case, il suffit de créer une nouvelle classe de test
- Celle-ci devra hériter de la classe `ActivityInstrumentationTestCase2`
- Et fournir la classe à tester dans son constructeur

```
public class ActivityTest
    extends ActivityInstrumentationTestCase2<MainActivity>{

    public ActivityTest() {
        super(MainActivity.class);
    }
}
```



# SetUp

- Lors du setUp, on récupère les instances de l'activity et des composants à tester

```
private MainActivity activity = null;
private TextView myTextView = null;
private Button myButton = null;

@Override
protected void setUp() throws Exception {
    super.setUp();
    activity = getActivity();

    myTextView = (TextView) activity.findViewById(
        fr.cnam.helloworld.R.id.myTextView);

    myButton = (Button) activity.findViewById(
        fr.cnam.helloworld.R.id.myButton);
}
```

R (ressources)  
du projet à tester

# Préconditions

- Une bonne pratique consiste à tester que le setUp s'est bien déroulé

```
@SmallTest  
public void testPreconditions() {  
    assertNotNull("activity is null", activity);  
    assertNotNull("myTextView is null", myTextView);  
    assertNotNull("myButton is null", myButton);  
}
```

# Layout test

- Tester le layout des composants

```
@MediumTest
public void testMyTextView_layout() {
    final View decorView = activity.getWindow().getDecorView();

    ViewAsserts.assertOnScreen(decorView, myTextView);

    final ViewGroup.LayoutParams layoutParams =
        myTextView.getLayoutParams();

    assertNotNull(layoutParams);
    assertEquals(layoutParams.width,
        WindowManager.LayoutParams.WRAP_CONTENT);

    assertEquals(layoutParams.height,
        WindowManager.LayoutParams.WRAP_CONTENT);
}
```

Test que le composant  
est présent

# Tester les comportements souhaités

- Utilisation de TouchUtils pour simuler les actions utilisateur

```
@MediumTest
public void testMyButton_click() {
    String initialText = activity.getString(
        fr.cnam.helloworld.R.string.hello_world);

    String expectedText = activity.getString(
        fr.cnam.helloworld.R.string.after_click);

    assertEquals(initialText, myTextView.getText());

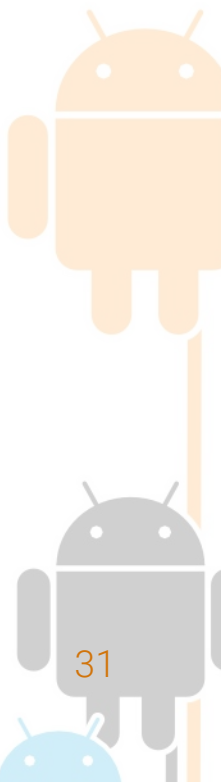
    TouchUtils.clickView(this, myButton);

    assertEquals(expectedText, myTextView.getText());
}
```

Clic sur le bouton

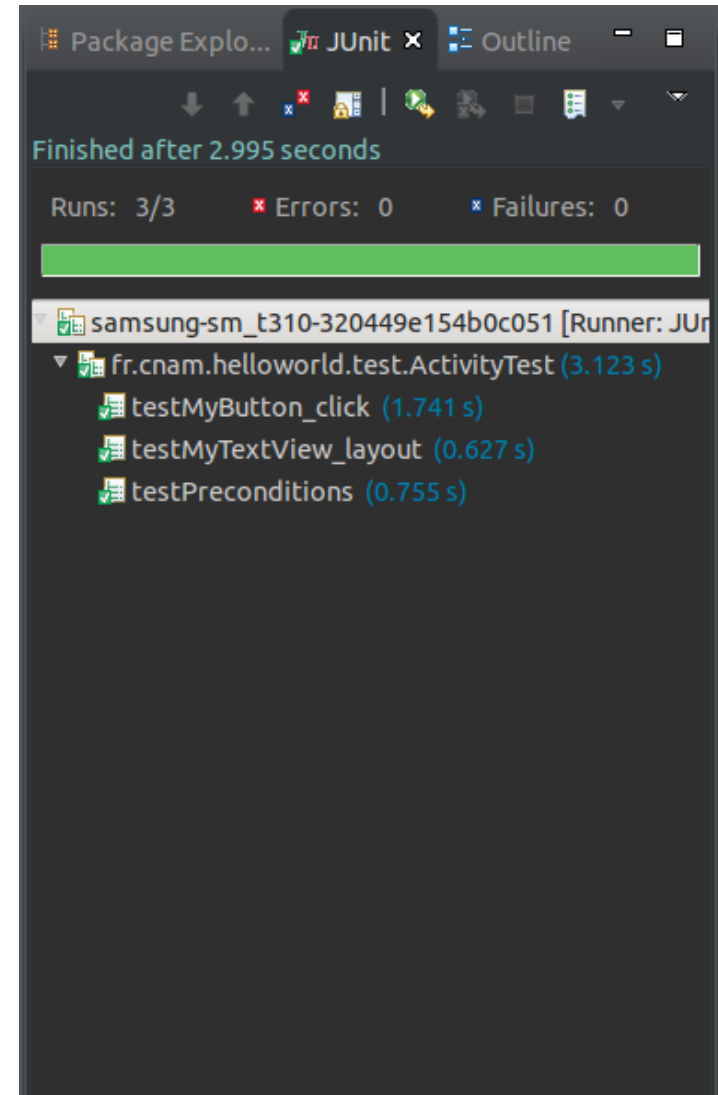
# Test Annotations

- **@SmallTest :**
  - ➔ Un test court, moins de 100 ms
- **@MediumTest :**
  - ➔ Un test de durée moyenne, supérieure à 100 ms
- **@LargeTest :**
  - ➔ Idem que le mediumTest, avec un accès plus important aux ressources



# Résultats

- Et voilà le travail !!
- Les sources complètes sont dans le projet du cours
- Bons tests !!





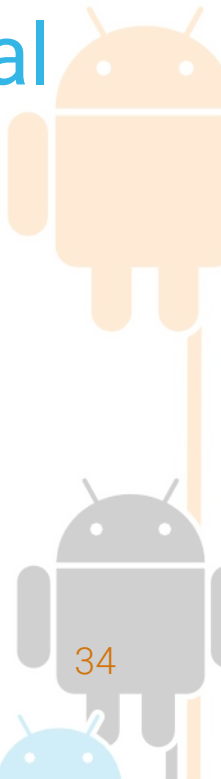
# IN01 – Séance 07

## Programmation avancée Concurrence



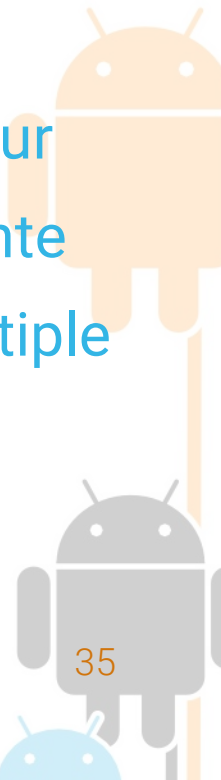
# Généralités

- Pour gérer des traitements et pour ne pas bloquer l'interface utilisateur, on utilise des Threads
- Problème !! Il nous est interdit de modifier l'IHM depuis un autre Thread que le Thread principal
- Deux outils existent sur Android : Handler et AsyncTask
- cf. SwingWorker de Swing



# Handler - Pattern

- Son but : synchroniser de petites tâches et les ordonner (comme un ordonnanceur)
- Un handler particulier attaché à l'IHM
- C'est une queue de tâches
- Le consommateur se met en attente sur la queue
- Lorsqu'une tâche est ajoutée par un des multiples producteurs (threads), la queue est notifiée, débloquent ainsi le consommateur
- Lorsque le consommateur a tout consommé, il se remet en attente
- Ça ressemble au patron Reader – Writer lock, mais, inversé (multiple writer et mono reader)
- Utilisation massive dans Alroid (le script emploie son propre thread)



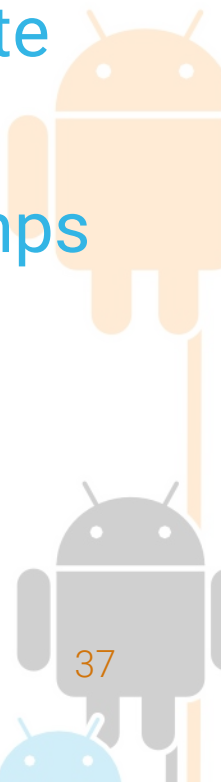
# Handler - Utilisation

- Il faut créer un handler (ou en récupérer un d'une vue)
- Puis lui envoyer des tâches

```
Handler handler = new Handler(Looper.getMainLooper());  
  
handler.post(new Runnable() {  
  
    @Override  
    public void run() {  
        // TODO Auto-generated method stub  
    }  
});
```

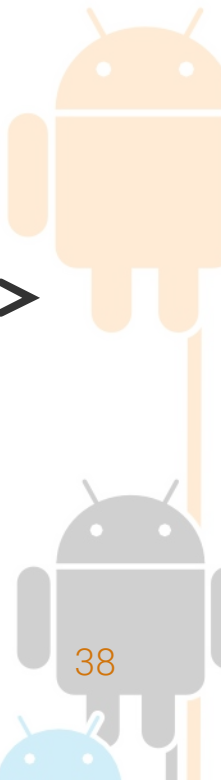
# Handler - Les méthodes

- Plusieurs méthodes pour poster une tâche sont disponibles :
  - ➔ `post()` : place la tâche à la fin de la queue (sera exécutée après toutes les autres)
  - ➔ `postAtFrontOfQueue()` : place la tâche au début, juste après celle en cours d'exécution
  - ➔ `postAtTime()` : place la tâche avec un marqueur temps
  - ➔ `postDelayed()` : place la tâche avec un marqueur de délai



# AsyncTask

- Utilisé pour des tâches généralement plus longues
- Et/ou, pour gérer l'affichage de la progression dans l'IHM
- On utilise la classe **AsyncTask**  
`<TypeDesParamètres,  
TypeDeProgression, TypeDuRésultat>`



# AsyncTask - Création

```
class MyAsyncTask extends AsyncTask<String, Integer, String> {  
    @Override  
    protected String doInBackground(String... params) {  
        return null;  
    }  
  
    @Override  
    protected void onCancelled(String result) {  
    }  
  
    @Override  
    protected void onPostExecute(String result) {  
    }  
  
    @Override  
    protected void onProgressUpdate(Integer... values) {  
        super.onProgressUpdate(values);  
    }  
}
```

Tâche principale

Si la tâche est annulée

Une fois finie

Gérer la progression

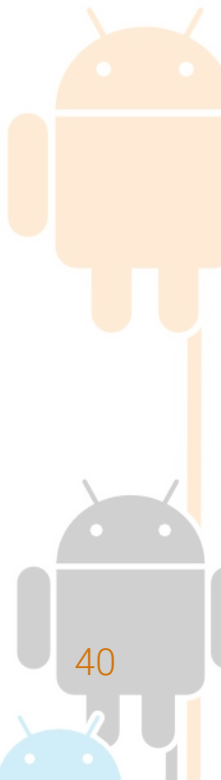
# AsyncTask - Utilisation

- Il suffit maintenant de l'instancier et de la lancer

```
MyAsyncTask task = new MyAsyncTask();  
task.execute(new String[] { "my String" });
```

paramètres de la méthode

`doInBackground(String... params)`





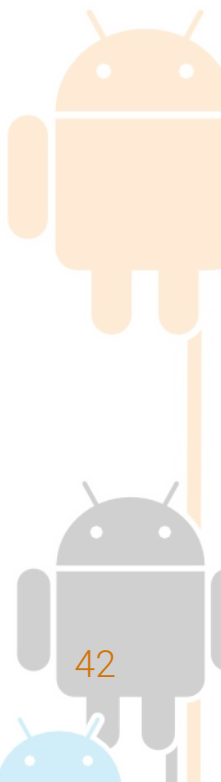
# IN01 – Séance 07

IHM avancées  
Vues personnalisées



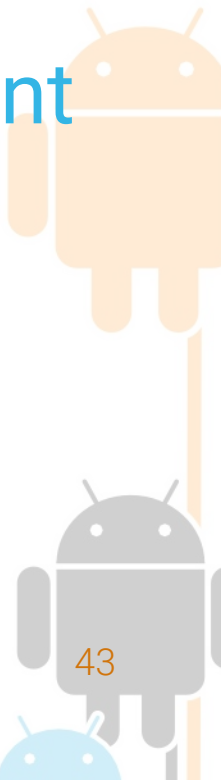
# Vue d'ensemble

- Les Activities sont utilisées pour agréger des vues ensemble (traitement d'un ensemble de données)
- Une vue permet de traiter une donnée en particulier
- Il existe quatre façons de créer des vues personnalisées :
  - ➔ Une vue composée (héritage d'une sous-classe de **ViewGroup**)
  - ➔ Un layout (héritage de ViewGroup)
  - ➔ Spécialisation d'une vue existante (héritage d'une sous-classe de **View**)
  - ➔ Création d'une vue de bout en bout (hérite de **View**)
- Possibilité de combiner les approches



# Vue composée

- On doit créer une classe qui hérite d'une classe elle-même héritant d'un ViewGroup
- Par exemple LinearLayout
- Ensuite, dans le constructeur ou dans une méthode spécifique, on ajoute dynamiquement des vues enfants !!



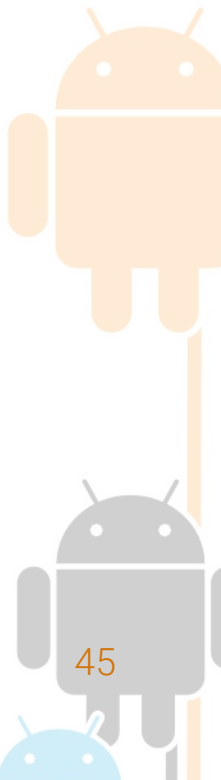
# Vue composée

- Deux façons d'ajouter des composants :
  - ➔ Par programmation grâce à la méthode **addView** (héritée de view)
  - ➔ Par XML avec la méthode **LayoutInflater**

```
public class MyView extends LinearLayout {  
  
    public MyView(Context context, AttributeSet attrs) {  
        super(context, attrs);  
  
        EditText txt1 = new EditText(context);  
        EditText txt2 = new EditText(context);  
  
        super.addView(txt1);  
        super.addView(txt2);  
    }  
}
```

# Vue composée - Utilisation

- Utile pour afficher des listes de données, des tableaux
- Dans Algoid j'ai utilisé cette technique pour :
  - ➔ Le File Manager
    - Construction dynamique de la vue en fonction des fichiers
  - ➔ Invite de commande
    - Entrées sorties utilisateur pilotées par programme
  - ➔ Scope View
    - Construction d'une vue hiérarchique en fonction de l'état de la mémoire



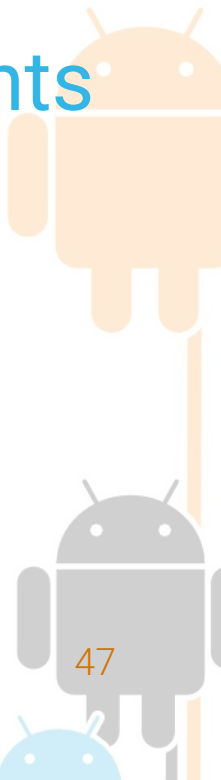
# Création d'un layout

- Principe : hériter de la classe **ViewGroup**
- Surcharger la méthode **onLayout** et placer les vues enfants (**getChildCount()** et **getChildAt()**)

```
public class MyLayout extends ViewGroup{  
  
    @Override  
    protected void onLayout(boolean changed, int l,  
        int t, int r, int b) {  
  
        for (int i = 0; i < getChildCount(); i++) {  
            View child = this.getChildAt(i);  
  
            child.setX(10);  
            child.setY(10 * i);  
        }  
    }  
}
```

# Spécialisation

- Principe : hériter d'une vue existante (elle-même spécialisation de la classe **View**)
- Ajouter des comportements en surchargeant des méthodes (il faut bien lire la documentation)
- Ajouter des comportements sur des évènements (**onTouch**, **onLongClick**, **onKey**)
- Ajouter des dessins (surcharge de la méthode **onDraw**)



# Spécialisation – Exemple

- L'IDE d'Algoid par exemple : AutoCompletion, coloration syntaxique et breakpoints

```
public class SourceEditorTextView extends MultiAutoCompleteTextView {  
  
    public SourceEditorTextView(Context context, AttributeSet attrs) {  
        super(context, attrs);  
  
        setOnLongClickListener(new OnLongClickListener() {  
            // ajouter les breakpoints  
        })  
    }  
  
    @Override  
    protected void onDraw(Canvas canvas) {  
        // dessiner ici les breakpoints  
  
        super.onDraw(canvas);  
    }  
}
```

Ajoute un breakpoint

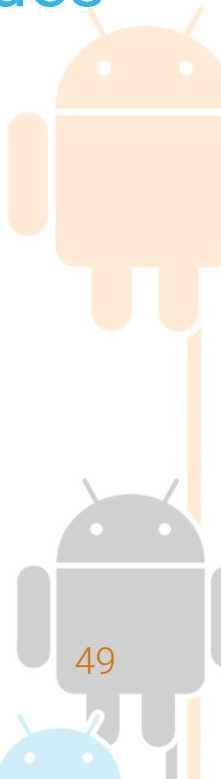
Dessine les breakpoints

En réalité,  
3 constructeurs



# Création de bout en bout

- Principe : hériter de la classe View
- Comme une vue spécialisée, mais sans comportement préalable
  - ➔ Gérer des évènements et implémenter les méthodes nécessaires comme onDraw
- Peut être long et fastidieux, il vaut mieux bien évaluer si la vue n'existe pas au préalable
- Exemple : Algo la petite tortue



# Utilisation de la vue

- Une fois la vue créée, il est possible de l'ajouter à un conteneur :
  - Activity, ViewGroup, Fragment
- Toujours de deux façons :
  - En Java

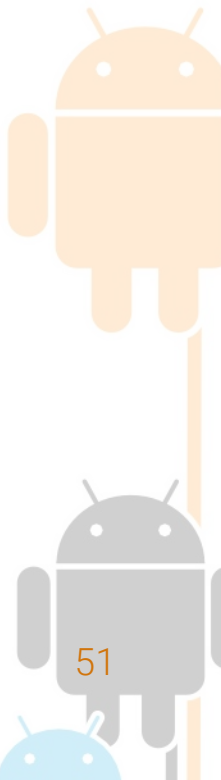
```
addView(new MyView(context, attrs));
```

- En XML

```
<fr.cnam.in01.techniques.MyView  
    android:id="@+id/myView"  
    myView:text1="myText1" myView:text2="myText2"/>
```

# Paramètres personnalisés

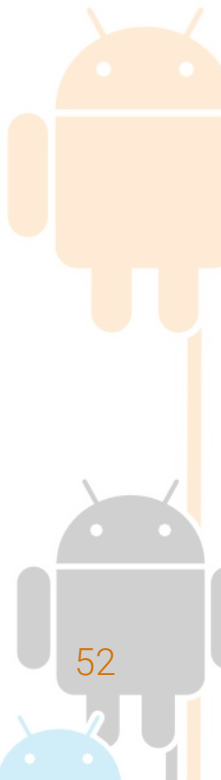
- Plusieurs étapes pour créer un attribut personnalisé
- But : pouvoir paramétrer la vue depuis des valeurs passées dans l'XML
- Un “Vrai” composant graphique !!



# Paramètres personnalisés

- Pour chaque attribut (ici text1 et text2), il faut créer une paire getter/setter (setText1 etc.)
- Il faut déclarer ces attributs dans une ressource **sous** `res/values/attrs.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <declare-styleable name="MyView">
    <attr name="text1" format="string" />
    <attr name="text2" format="string" />
  </declare-styleable>
</resources>
```



# Paramètres personnalisés

- Il faut ensuite déclarer le nouveau Namespace dans le fichier XML de l'activity (là où est utilisé MyView)

```
xmlns:myview="http://schemas.android.com/apk/res-auto"
```

- Puis utiliser les paramètres dans la déclaration du composant

```
<fr.cnam.in01.techniques.MyView  
    android:id="@+id/myView"  
    myview:text1="myText one"  
    myview:text2="myText two"/>
```

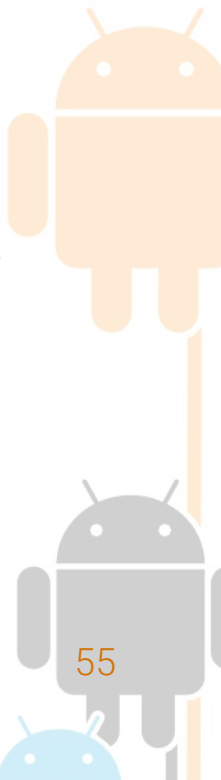
# Paramètres personnalisés

- Et enfin, récupérer les valeurs dans le constructeur de la vue

```
public MyView(Context context, AttributeSet attrs) {  
    super(context, attrs);  
  
    TypedArray typedAttrs = getContext()  
        .obtainStyledAttributes(attrs, R.styleable.MyView);  
  
    String text1 = typedAttrs.getString(R.styleable.MyView_text1);  
    txt1.setText(text1);  
  
    String text2 = typedAttrs.getString(R.styleable.MyView_text2);  
    txt2.setText(text2);  
}
```

# Conclusion

- De quoi créer des bibliothèques de composants entièrement personnalisées
- Idéal pour la réutilisabilité du code
- Ou la distribution de celui-ci
- Pourtant, encore assez peu de bibliothèques graphiques ne sont disponibles sur Android !!



# IN01 – Séance 07

IHM avancées  
Fragments



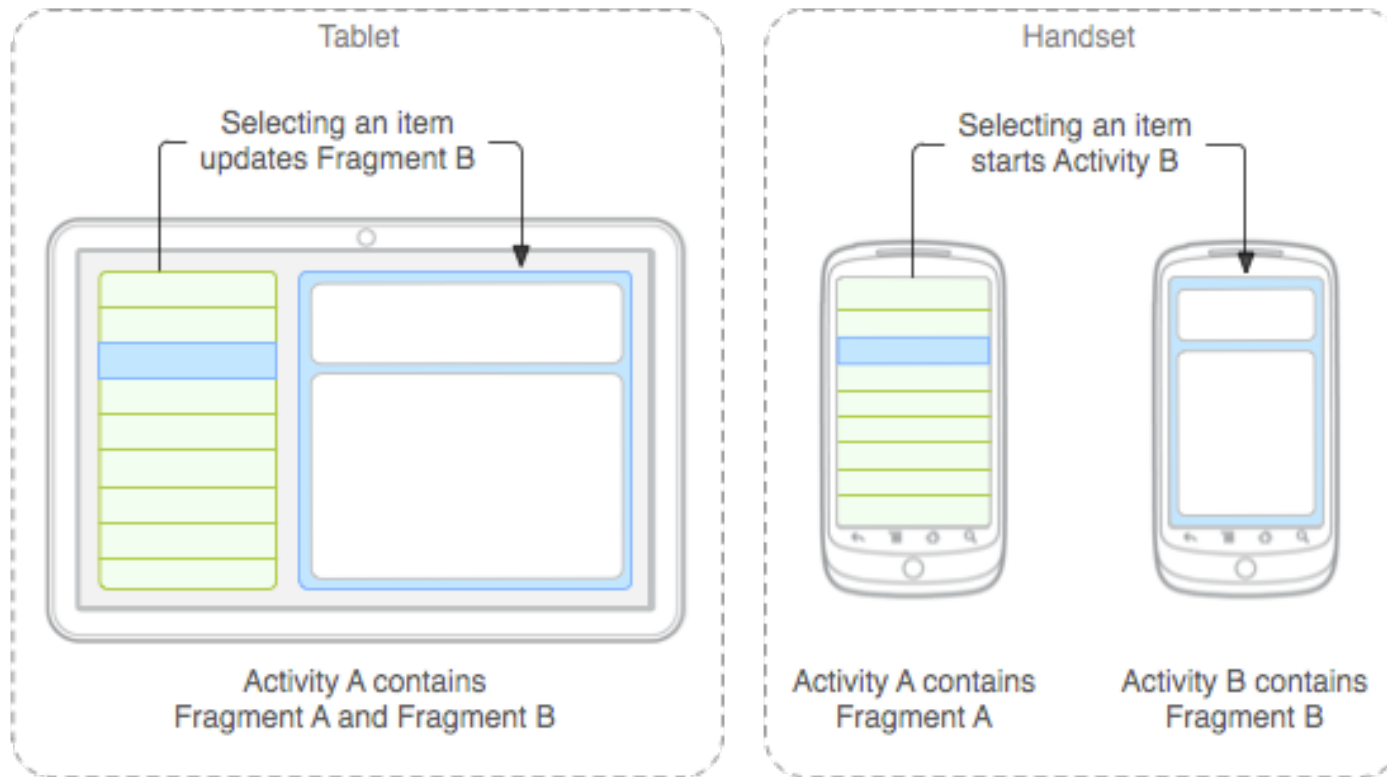


# Vue d'ensemble

- Un fragment est un composant graphique qui se situe entre la vue et l'activity
- Il est lié à une activity
- Il peut être graphique ou logique
- Il peut être chargé/déchargé dynamiquement (par programmation)
- Ou être utilisé de façon statique (en XML)
- Il est disponible depuis la version 11 de l'API (HoneyComb), mais grâce à l'app compat générée avec un projet Android, il peut être utilisé sur des API plus anciennes
- Il sert à adapter les interfaces aux différentes résolutions des appareils

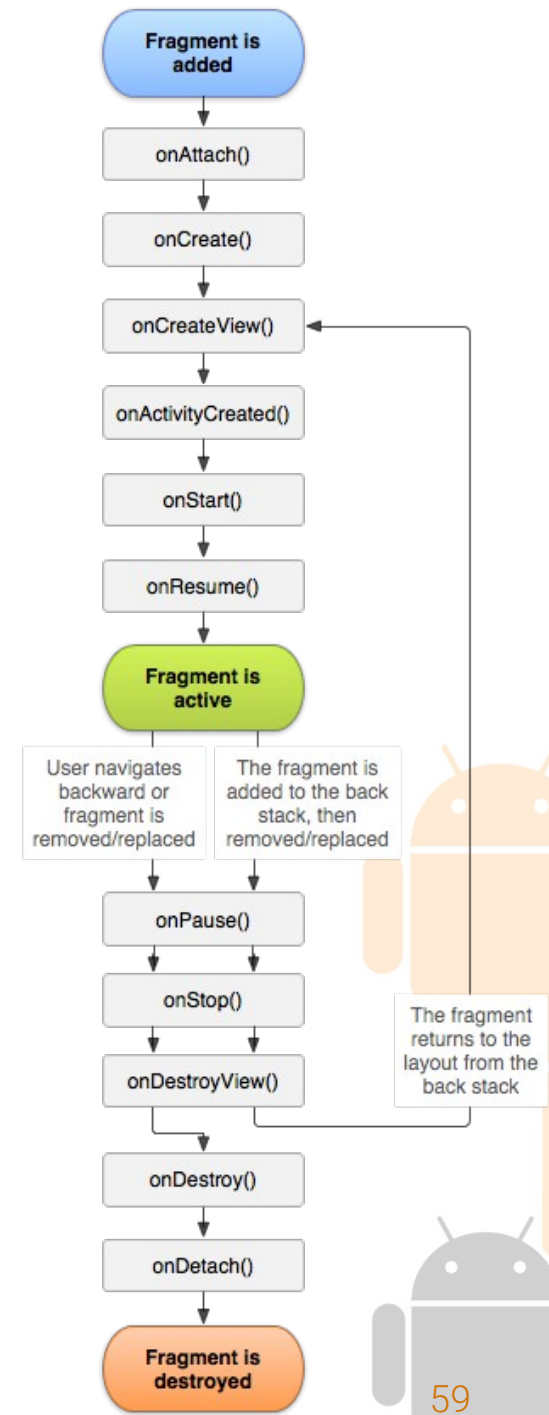
# Vue d'ensemble

- Les activities peuvent être composées d'un ou plusieurs fragments en fonction de la place à disposition



# Cycle de vie

- Il a son propre cycle de vie
- Cela permet d'isoler son comportement et ainsi de rendre plus "léger" le code de l'activity
- **onAttach** : sert à récupérer l'instance de l'activity
- **onCreate** : sert à instancier les objets non graphiques
- **onCreateView** : idem, mais graphiques
- **onStart** : lancer les traitements
- **onResume** : s'abonner aux événements, récupérer le contexte
- **onPause** : s'y désabonner et sauver le contexte
- **onStop**, **onDestroyView**, **onDestroy**, **onDetach** : on désalloue les ressources créées ci-dessus



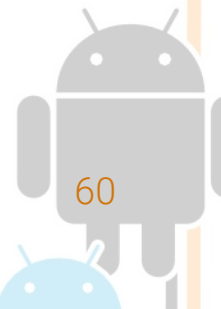
# Création d'un fragment

- Un fichier XML, comme pour une activity

```
<?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" >

    <!-- content -->

</LinearLayout>
```



# Création d'un fragment

- Et la classe associée (qui hérite de Fragment)

```
public class MenuFragment extends Fragment
```

- Laquelle implémente l'évènement onCreateView

```
@Override  
public View onCreateView(LayoutInflater inflater,  
    ViewGroup container, Bundle savedInstanceState) {  
  
    View view = inflater.inflate(R.layout.menu_fragment,  
        container, false);  
  
    return view;  
}
```

# layout/main\_activity

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="fr.cnam.in01.in01_fragments.MainActivity"
    tools:ignore="MergeRootFrame" >

    <fragment
        android:id="@+id/menu"
        android:name="fr.cnam.in01.in01_fragments.MenuFragment"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="horizontal" />

</FrameLayout>
```

Un seul fragment

# layout-land/main\_activity

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal">

  <fragment
    android:name="fr.cnam.in01.in01_fragments.MenuFragment"
    android:layout_height="match_parent"
    android:layout_width="0dp"
    android:layout_weight="1" />

  <fragment
    android:name="fr.cnam.in01.in01_fragments.MainFragment"
    android:layout_height="match_parent"
    android:layout_width="0dp"
    android:layout_weight="2" />

</LinearLayout>
```

Plusieurs fragments

# Instance du fragment

- Dans la méthode `onCreate()` de l'activity, on a accès à l'instance du/des fragments
- Attention, ils peuvent ne pas exister dans le XML, leur référence est `null` dans ce cas !!
- On utilise la méthode  
`Activity.getFragmentManager()` ou  
`Activity.getSupportFragmentManager()`

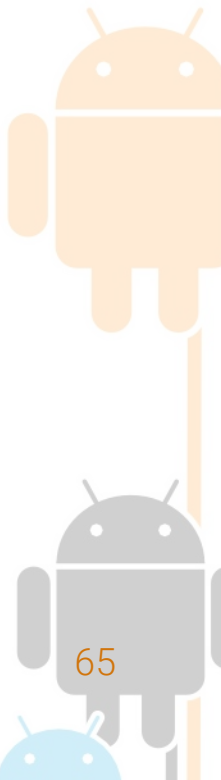
```
MenuFragment fragment = (MenuFragment) getFragmentManager()  
    .findFragmentById(R.id.menu);
```







# Du fragment vers l'activity

- Pour que les fragments soient réutilisables, il ne faut pas qu'ils communiquent directement avec l'activity
- On utilise une inversion de dépendance
- Par exemple un template method (design pattern, peu utilisé dans les IHM)
- Ou un observer/observable



# Évènements

- On définit une interface et un mécanisme d'évènement

```
public class MenuFragment extends Fragment {  
    private MenuChangeEvent menuChanged;  
  
    public interface MenuChangeEvent {  
        void menuChanger(int id);  
    }  
  
    public void setOnMenuChange(MenuChangeEvent menuChanged) {  
        this.menuChanged = menuChanged;   
    }  
  
    private void fireMenuChanged(int id) {  
        if (this.menuChanged != null) {  
            this.menuChanged.menuChanger(id);   
        }  
    }  
}
```

Abonnement

Inversion de dépendance

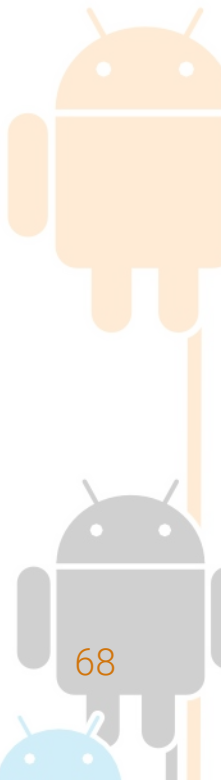
# Évènements

- Dans l'activity, lorsque le fragment est chargé, on s'abonne à l'évènement

```
menuFragment.setOnMenuChange(new MenuChangeEvent() {  
  
    @Override  
    public void menuChanger(int id) {  
        switch (id) {  
            case R.id.button1:  
                navigate(MenuFragment.class);  
                break;  
            case R.id.button2:  
                navigate(MainFragment.class);  
                break;  
        }  
    }  
});
```

# Placeholder

- Les fragments peuvent être placés de façon statique dans le XML, comme on l'a vu
- Ils peuvent également être chargés dynamiquement (par programmation) dans des placeholders
- En général, on utilise des **FrameLayout**



# Placeholder

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <FrameLayout
        android:id="@+id/placeholder1"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/placeholder2"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2" />

</LinearLayout>
```

# Placeholder

- On peut ensuite, charger les fragments à l'aide d'un objet de type **FragmentManager**

```
FragmentManager ft = getSupportFragmentManager().beginTransaction();  
ft.replace(placeholder, fragment);  
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);  
ft.addToBackStack(null);  
ft.commit();
```

- La méthode **addToBackStack** permet de gérer le bouton back (conserve un historique des fragments chargés, comme pour les activities)

# Navigation

- Ensuite, selon le nombre de placeholder à disposition, on choisit quel type de navigation l'on souhaite :
  - ➔ Une seule, on peut lancer une nouvelle activity ou remplacer le placeholder 1
  - ➔ Deux, on peut choisir le placeholder à charger et remplacer le précédent
- Le mieux étant de toujours passer le nom de la classe du fragment de façon à l'instancier par introspection

```
Class<? extends Fragment> frClass = MenuFragment.class;  
Fragment fragment = frClass.newInstance();
```

# Persistance des états du fragment

- On l'a vu, les Fragments sont des objets autonomes avec leur cycle de vie propre
- L'idée initiale est de décharger l'activity des tâches propres aux sous-éléments graphiques qui se chargent et se déchargent de façon dynamique
- De cette façon, il est possible de gérer la sauvegarde des états dans un Bundle
- Comme pour les activity, on utilise les évènements du cycle de vie du fragment : **onCreateView()** et **onSaveInstanceState()**





# Menus

- Les fragments apportent leur propre menu lorsqu'ils sont chargés

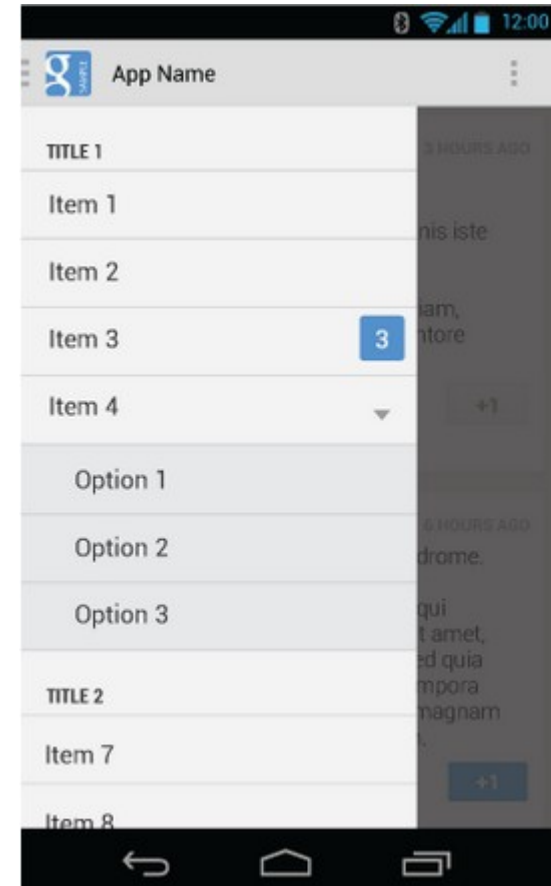
```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
}

public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    inflater.inflate(R.menu.fragment_menu, menu);
    super.onCreateOptionsMenu(menu, inflater);
}

public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.fragment_menu_item1:
            // Action 1
            return true;
        default:
            // Quelqu'un d'autre peut gérer ce menu
            return super.onOptionsItemSelected(item);
    }
}
```

# Navigation drawer

- Un menu qui glisse quand on en a besoin
- Source :  
<http://developer.android.com/training/implementing-navigation/nav-drawer.html>



# Navigation drawer

## ■ Création de l'activity

```
<android.support.v4.widget.DrawerLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/drawer_layout"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">
```

```
    <FrameLayout  
        android:id="@+id/content_frame"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent" />
```

Contenu

```
    <ListView android:id="@+id/left_drawer"  
        android:layout_width="240dp"  
        android:layout_height="match_parent"  
        android:layout_gravity="start"  
        android:choiceMode="singleChoice"  
        android:divider="@android:color/transparent"/>
```

Menu sous forme  
de liste

```
</android.support.v4.widget.DrawerLayout>
```

# Navigation drawer

## ■ Chargement du menu dans l'activity

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    ListView menu = (ListView) findViewById(R.id.menu);

    String[] menuItems = getResources().getStringArray(R.array.menu_items_array);

    // Set the adapter for the list view
    menu.setAdapter(new ArrayAdapter<String>(this, R.layout.drawer_list_item, menuItems));

    // Set the list's click listener
    menu.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            // TODO Auto-generated method stub
        }
    });
}
```

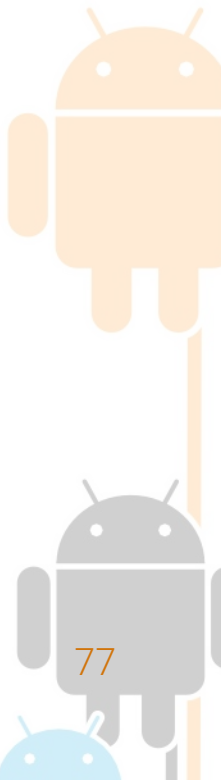
Items dans des tableaux res/values

Icône + texte

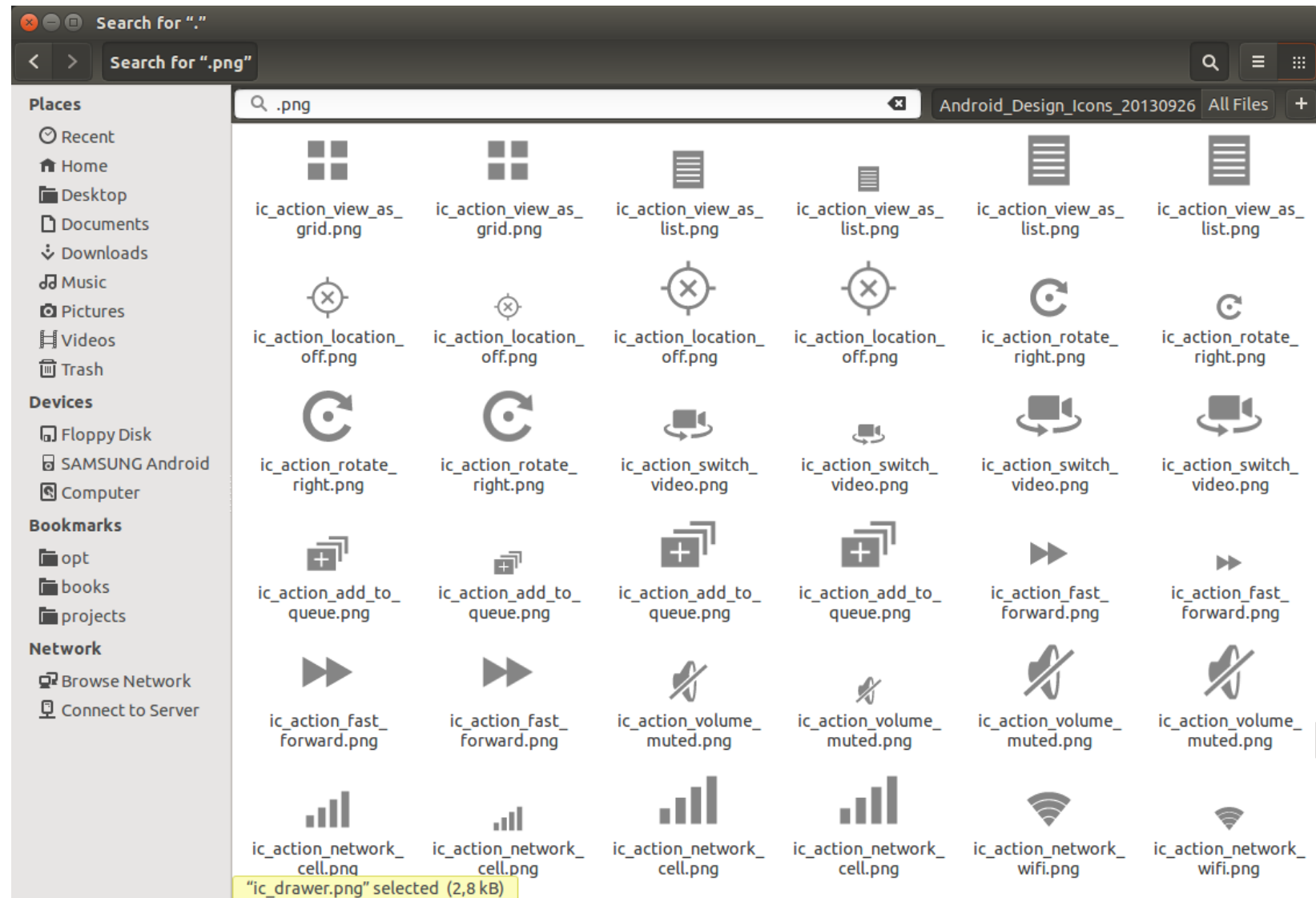
Gestion des événements

# Navigation drawer

- Idéal avec les fragments
- Google fournit un ensemble d'icônes
- [http://developer.android.com/downloads/design/Android\\_Design\\_Icons\\_20130926.zip](http://developer.android.com/downloads/design/Android_Design_Icons_20130926.zip)

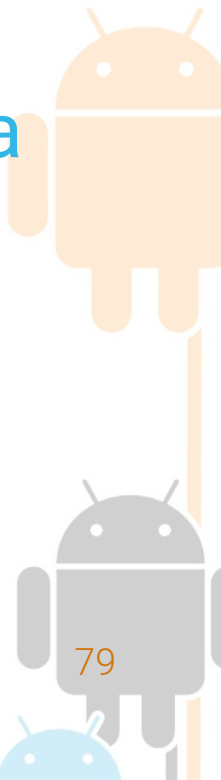


# Navigation drawer



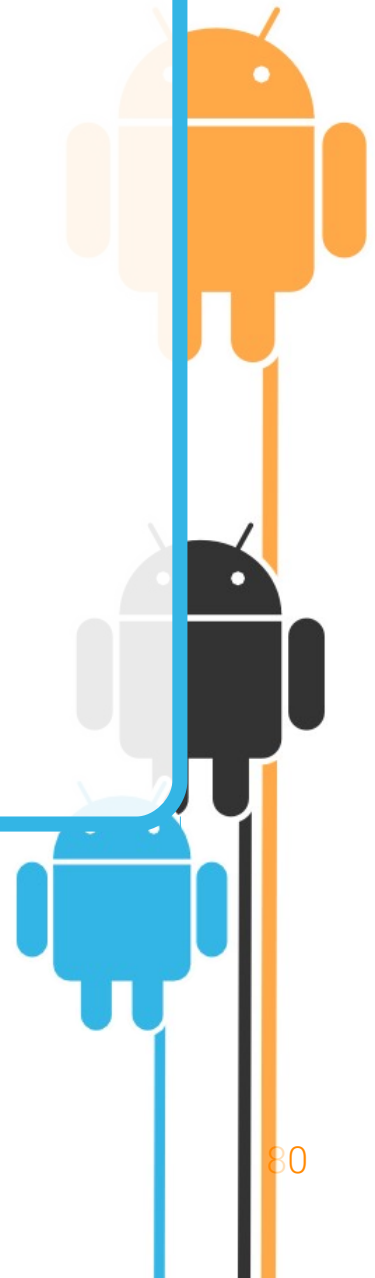
# Conclusion

- Les fragments offrent une réelle souplesse dans l'organisation des IHM en fonction des résolutions
- Une bonne pratique consiste à toujours concevoir ses IHM selon des fragments, même pour des apps ne nécessitant que peu d'écran
- Une navigation plus riche pourra être conçue par la suite
- Les composants de l'application bénéficieront également d'une plus grande réutilisabilité



# IN01 – Séance 07

Autres  
Stratégies et alternatives





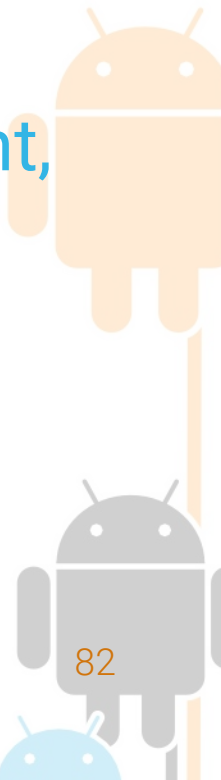
# Stratégie – Choix de l'application

- Red Ocean Strategy :
  - Compléter un marché existant
  - Exploiter une demande existante
  - Battre la concurrence
- Exemples : Pet rescue, Points and clicks
- Avantage : Le marché existe déjà
- Inconvénient : Pas d'innovation, pas passionnant



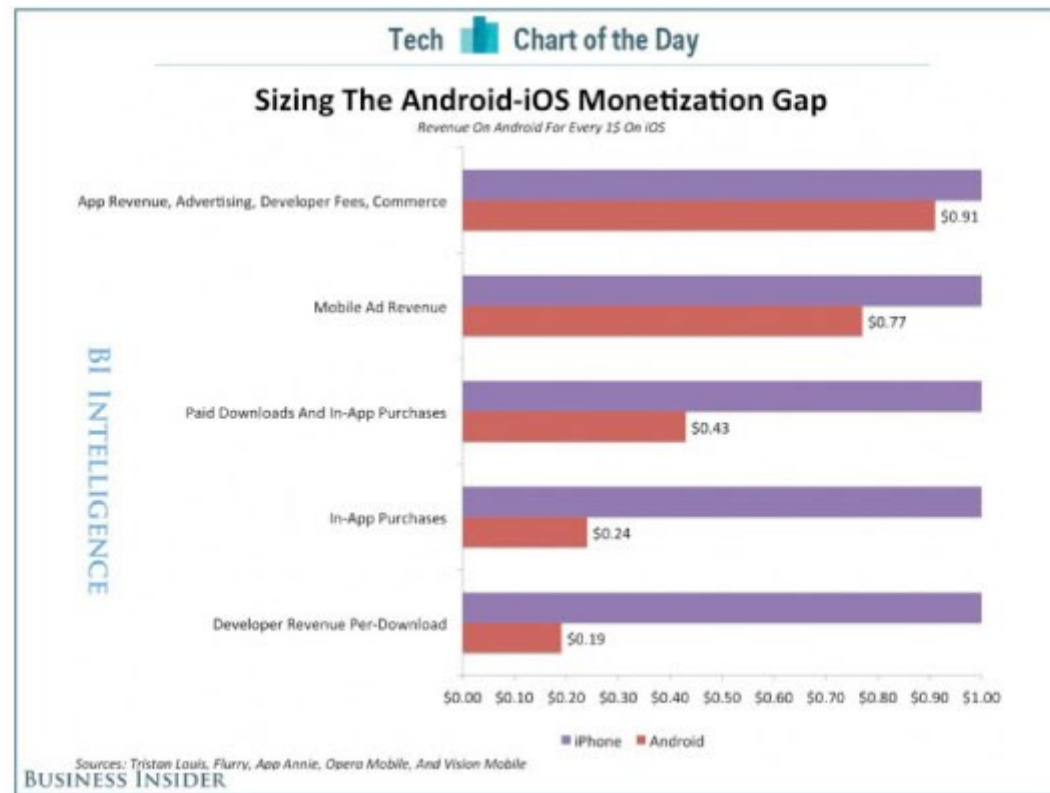
# Stratégie – Choix de l'application

- Blue Ocean Strategy :
  - Créer un marché inexploité
  - Créer ou capter de nouveaux besoins
  - Pas de concurrence
- Exemple : AIDE
- Avantages : Avoir une longueur d'avance, être innovant, développement motivant
- Difficultés : Identifier le besoin, faire connaître son application



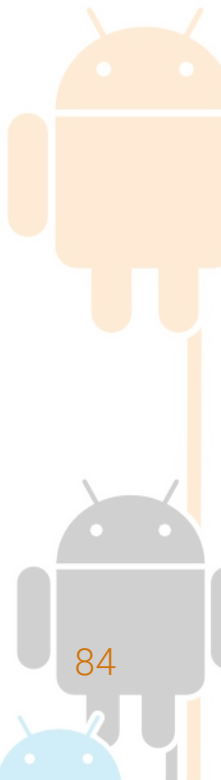
# Dégager du revenu

- Android une plateforme où il est difficile de dégager du revenu



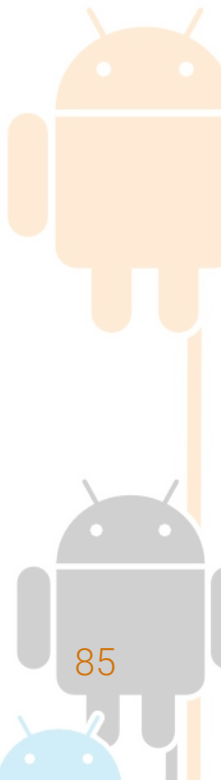
# Enjeux du multiplateforme

- De plus, il existe beaucoup d'applications sur le Play Store (phénomène de flooding, noyade)
- Les utilisateurs n'ont pas l'habitude d'acheter sur cette plateforme
- Il vaut mieux viser la publicité
- Une meilleure stratégie consiste à viser plusieurs plateformes



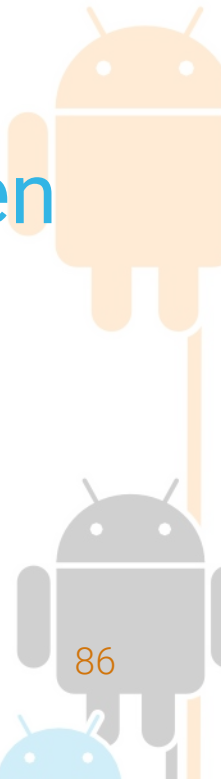
# Enjeux du multiplateforme

- Double avantage :
  - Doubler potentiellement l'audience
  - Gagner en crédibilité
- Inconvénient :
  - Il faut programmer plusieurs fois la même chose
  - Dans des langages différents (Java – Objective C)
- Ce n'est pas une obligation, des alternatives existent



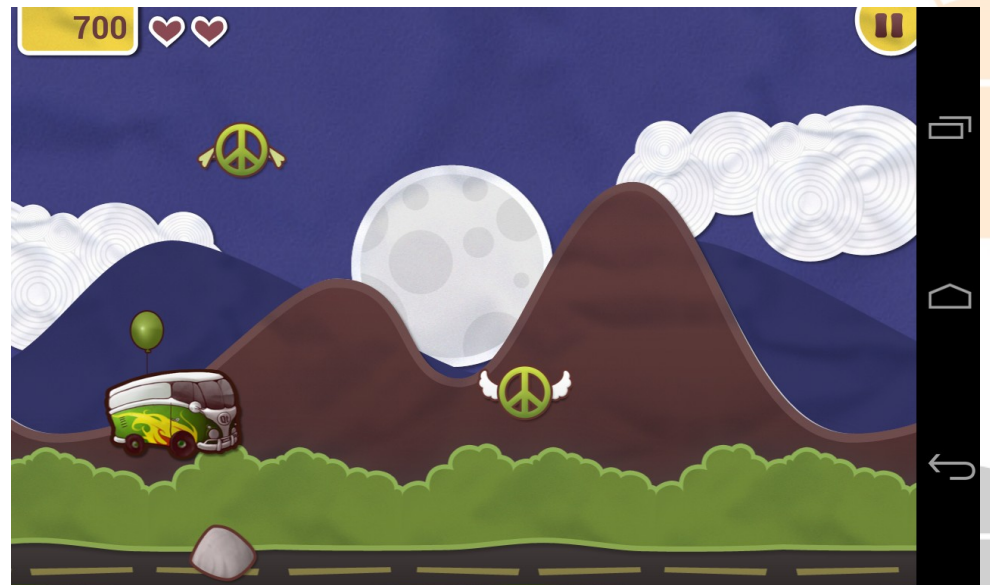
# C++

- Un tronc commun, le C++ finalement le langage le plus multiplateforme (Cross-compilation vs Interpretation VM)
- Une stratégie : écrire le middleware en C++ et les UI en natif sur les deux plateformes
- Inconvénients : compliqué, difficile à mettre en place (database ?, bibliothèque ?)



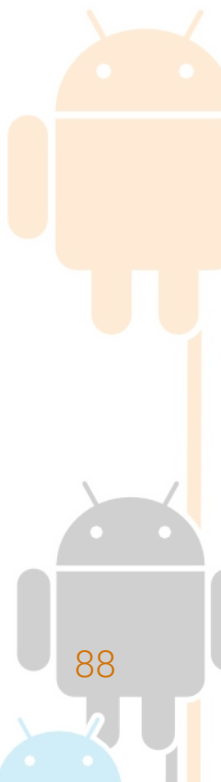
# C++/Qt

- Une alternative : Qt 5.2
- La version 5 annonçait les prémices, la 5.2 officialise le portage iOS et Android
- Une UI non native, mais complètement portable (la force de Qt)
- C++ plus rapide que Java
- Pas de Garbage collector
- On peut faire des jeux
- Exemple : FlyingBus de Digia



# Xaramin/Mono

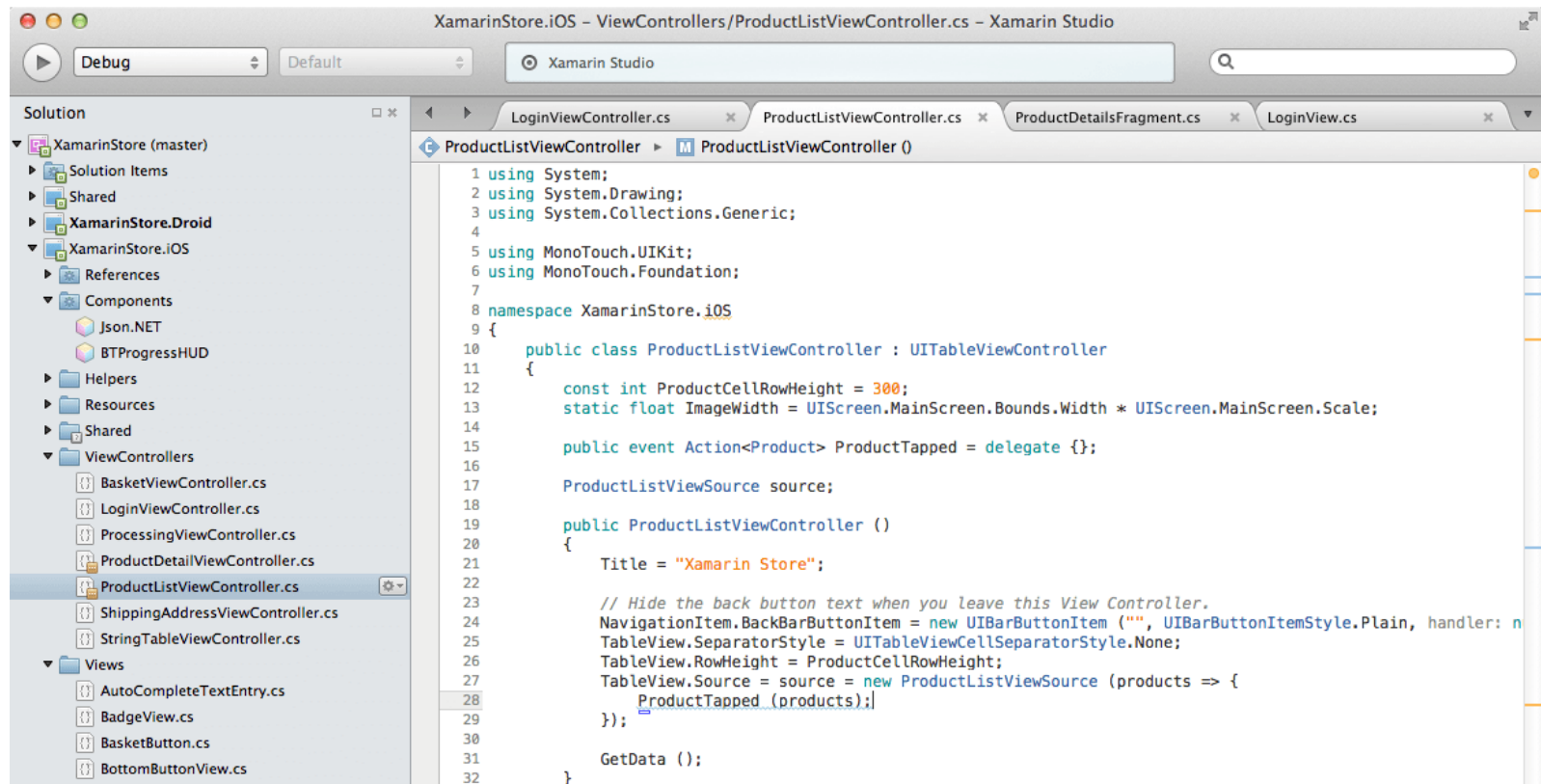
- Mono est une implémentation de la plateforme .net sur UNIX
- Xaramin est une startup des créateurs de Mono pour viser les plateformes mobiles
- Solution payante (démon puis 299 \$/année)
- Mais un gain de temps
- Et C# est vraiment un langage puissant
  - ➔ Objet/Fonctionnel (Linq)
  - ➔ Nombreux sucres syntaxiques : propriétés, événements, lambdas, extension methods
  - ➔ Un framework riche : Reflexion, Emit, Lightweight AOP (LinFu)





# Xamarin

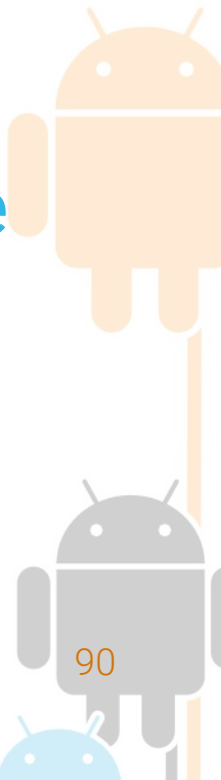
- Un IDE multiplateforme (Win, MacOS), mais pas Linux



# RobotVM

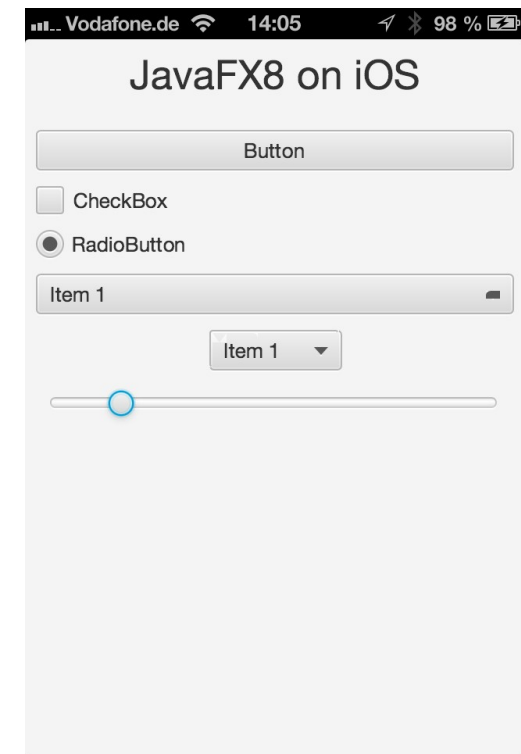


- RobotVM compile le bytecode Java en Assembleur à destination des processeurs ARM ou X86
- Bridge avec les API Objective C comme CocoaTouch
- Open source (GPLv2)
- Support Eclipse (plugin) et Maven
- Stratégie : développement en couche, middleware commun en Java et deux applications UI natives
- Le langage AL et JASI (le parser) fonctionnent



# JavaFX

- Expérimental
- Utilisation de l'OpenJFX 8, projet non officiel :  
<https://bitbucket.org/javafxports/android/wiki/Home>
- Utilisation de RobotVM pour viser iOS
- Il existe même un plugin NetBeans



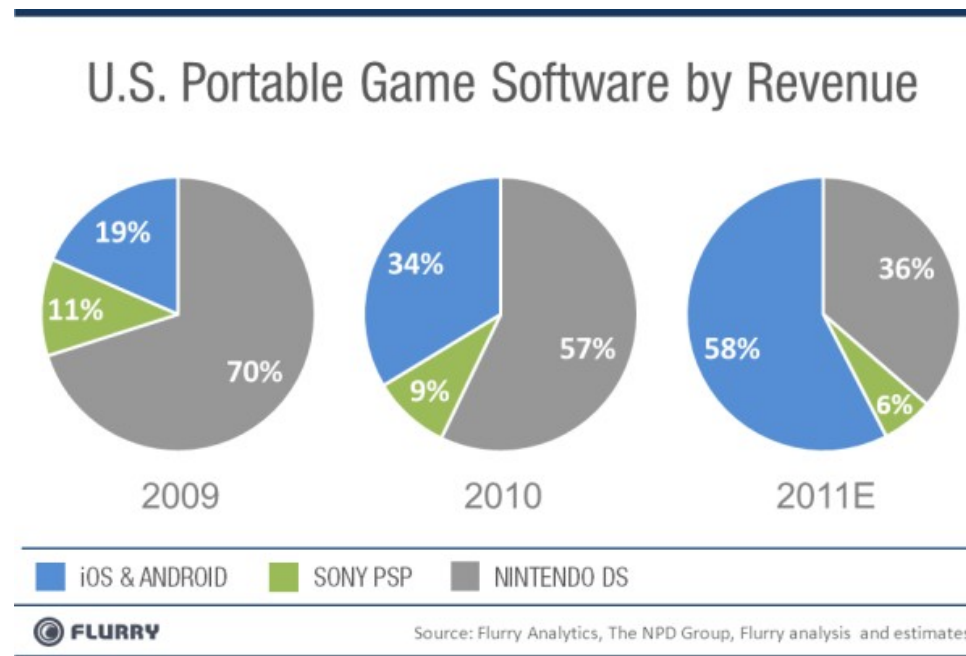
# IN01 – Séance 07

## Autres Jeux Vidéo Mobiles



# Parts de marché

- Le jeu vidéo mobile progresse de façon spectaculaire sur le marché



Source : Flurry Analytics (un concurrent de Google Analytics)

# Game engines

- AndEngine :

- 2D – OpenGL ES - Physic
- Android
- Java based – simple d'utilisation
- Gratuit et Open Source



- JmonkeyEngine :

- 3D – OpenGL ES
- Java based
- Open Source free BSD



# Multiplateforme - Java

- LibGDX :

**libGDX**

- 2D / 3D – OpenGL ES – Physics
- Multiplateforme (Android, iOS, Sybian, Win, Linux, MacOS, html 5)
- RobotVM ©
  - cross compilation
- Java
- Open source



# OpenFl

- OpenFl :
  - 2d uniquement
  - massivement multiplateforme
  - Fonctionne sur le langage Haxe (multiparadigme)
    - Créé par un Français
  - Sur une VM NekoVM dont l'intermediate language est un langage de script (et pas du byteCode)
  - Initialement inspiré par Flash

Open**FL**

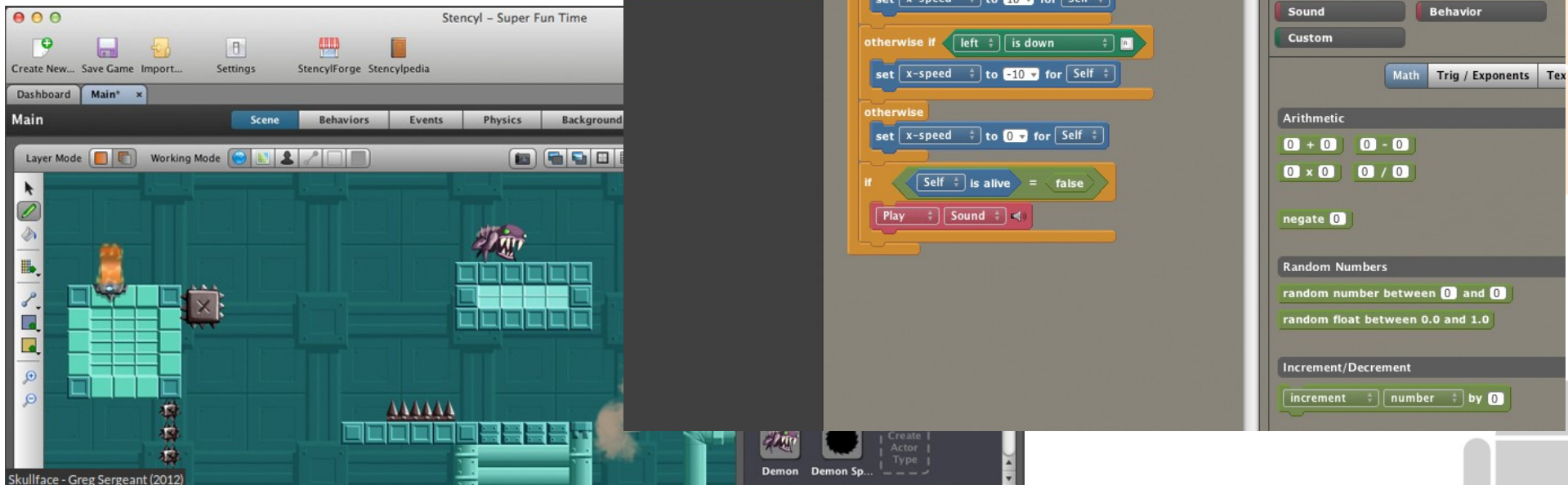




# NoJava – Alternative 2D

- Sencyl :

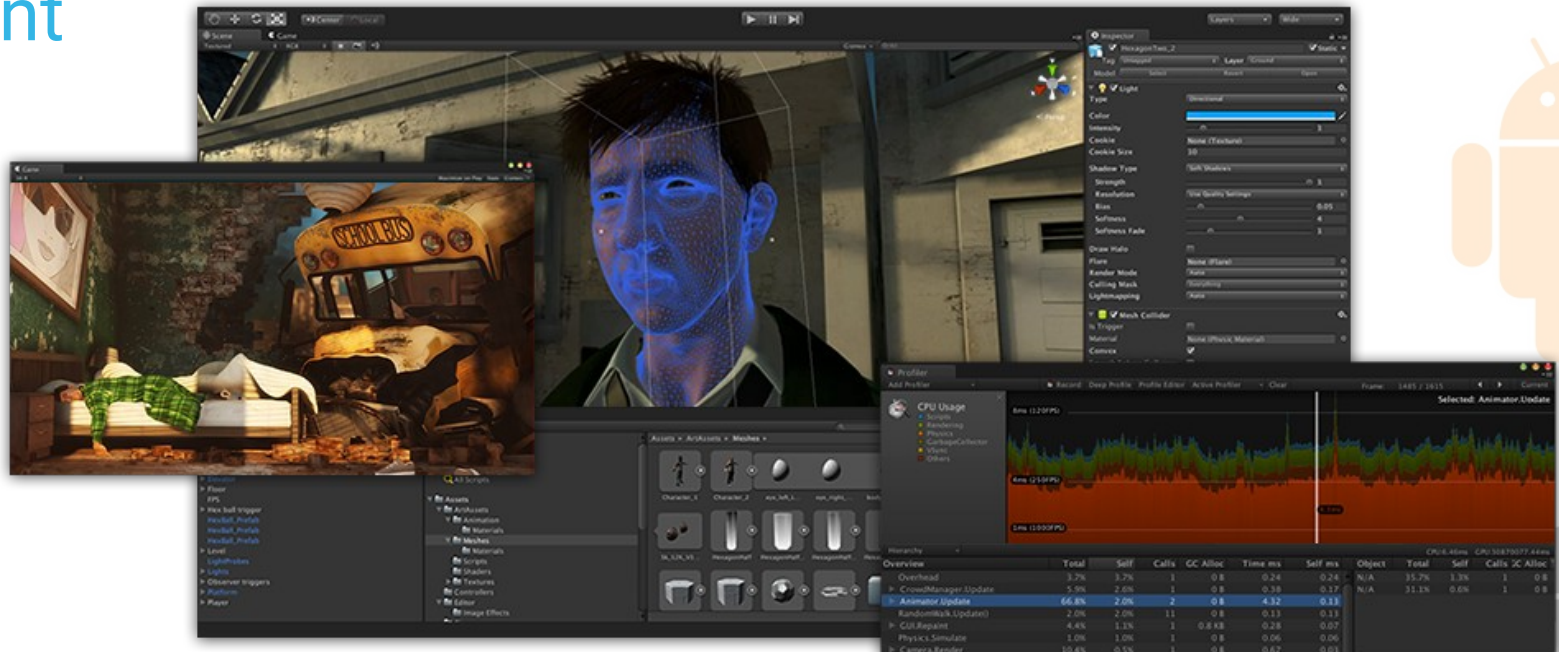
- 2D, multiplatforme (iOS, Android, Flash, etc.)
- Basé sur OpenFL
- NoCode - Payant



# No Java – Alternative 3D

- Unity 3D :

- 2D / 3D
- WYSIWYG Lua scripting
- Payant



# NoJava – Alternative 3D

- ShiVa 3D :



- 3D multiplateforme
- WYSIWYD – Lua scripting
- Server MMO
- Gratuit



# Fin

- Merci de votre attention
- Des questions ?

