
Android View, onClick, Activity, Modèle Vue Contrôleur

jean-michel Douin, douin au cnam point fr
version : 28 Octobre 2012

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Notes de cours

Bibliographie utilisée

<http://developer.android.com/resources/index.html>

<http://jfod.cnam.fr/NFP121/>

<http://www.oracle.com/technetwork/java/mvc-detailed-136062.html>

<http://csis.pace.edu/~bergin/mvc/mvcgui.html>

Sommaire

- **MVC,**
 - un rappel

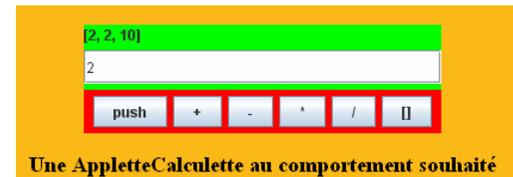
- **Mise en œuvre avec Android**
 - Une activité
 - Une IHM
 - Un modèle

- **Exemple d'utilisation de code j2se existant**

Avertissement, pré-requis et question

- **Pré requis indispensable**

- Avoir réalisé le tp mvc,
- Un tp utilisant une calculette à pile



- **Question ?**

- Comment assurer un couplage faible des classes
- Observable/Observateur
- IHM/View et Listener/Contrôleur

Pré requis, rappel

- **Pré requis**

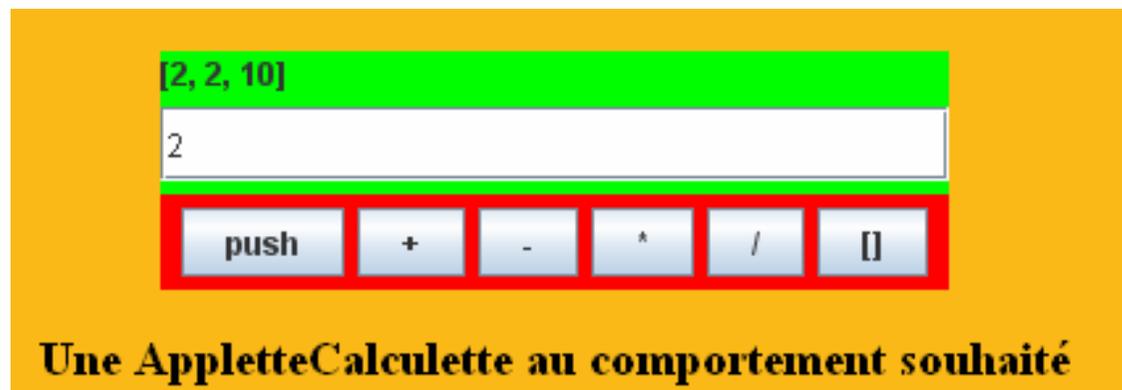
- TP Calculette à pile déjà réalisé J2SE

- **Thème : Modèle Vue Contrôleur**

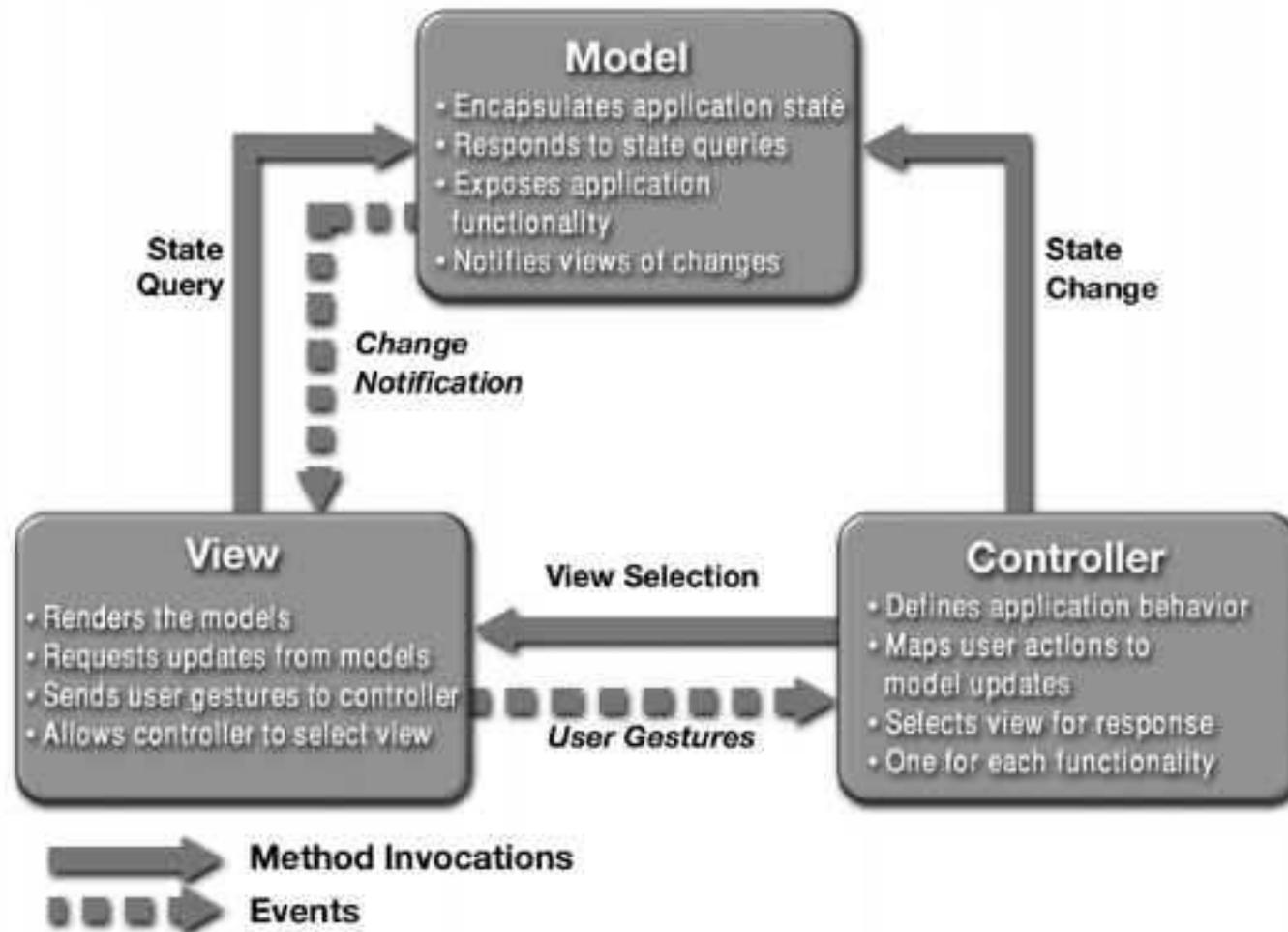
- http://lmi92.cnam.fr/progAvancee/tp4_2012.jar

- **Un usage de cette applette à cette URL est fortement conseillé**

- >appletviewer http://jfod.cnam.fr/eicnam/tp_mvc/tp_mvc.html

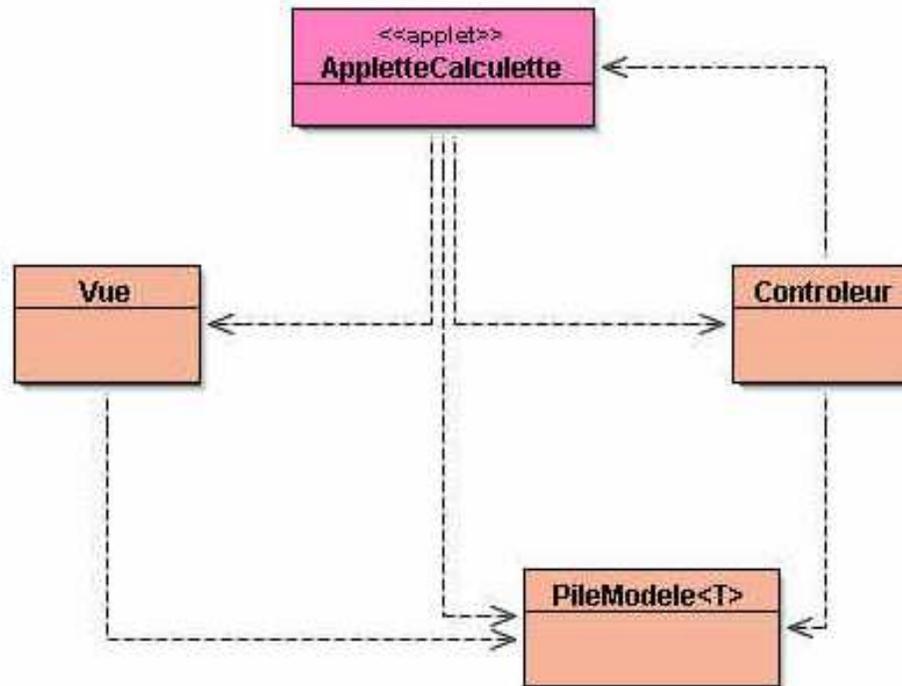


Pré requis, MVC



- <http://www.oracle.com/technetwork/java/mvc-detailed-136062.html>

Pré requis, l'architecture retenue pour le TP



- Le Modèle est une pile (classe **PileModele<T>**).
- La Vue correspond à l'affichage de l'état de la pile (classe **Vue**).
- Le Contrôleur gère les événements issus des boutons +, -, *, /, [] (classe **Controleur**).
- L'applette crée, assemble le modèle, la vue et le contrôle (classe **AppletteCalcullette**).

Cette architecture engendre des discussions

- Le **Modèle** est ici une pile (classe **PileModele<T>**).
- La **Vue** correspond à l'affichage de l'état de la pile (classe **Vue**).

[2, 3]

- Le **Contrôleur** gère les événements issus des boutons +, -, *, /, []



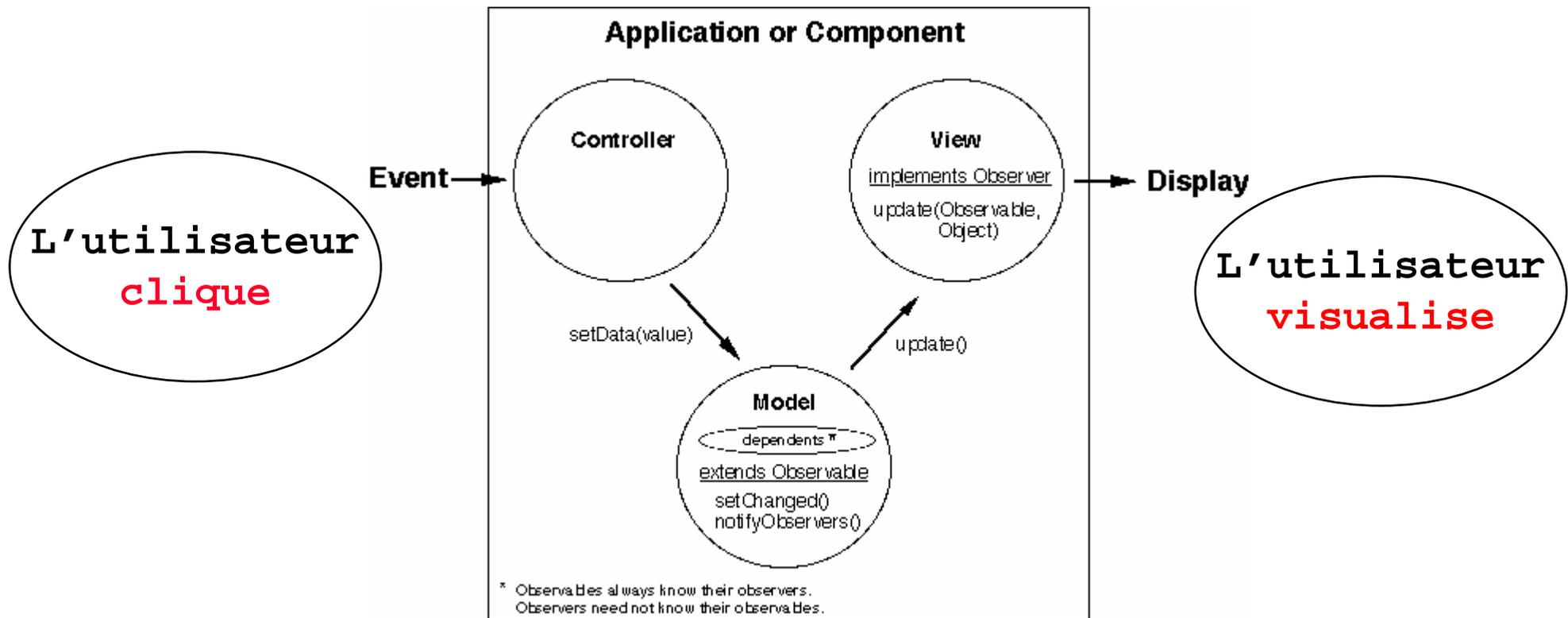
- L'applette crée, assemble le modèle, la vue et le contrôle (classe **AppletteCalculette**).



Discussions ... entre nous

- *Le modèle pourrait être la calculette constituée pour ses calculs internes d'une pile,*
- *Pourquoi les "listeners" des boutons sont-ils locaux au contrôleur ?*
- *Pourquoi un JPanel pour le contrôleur ?*
- *Ce choix de découpage MVC vous parait-il réaliste ?*
 - *Discussion, blabla, blabla, blabla*

Architecture classique ... une valeur sûre

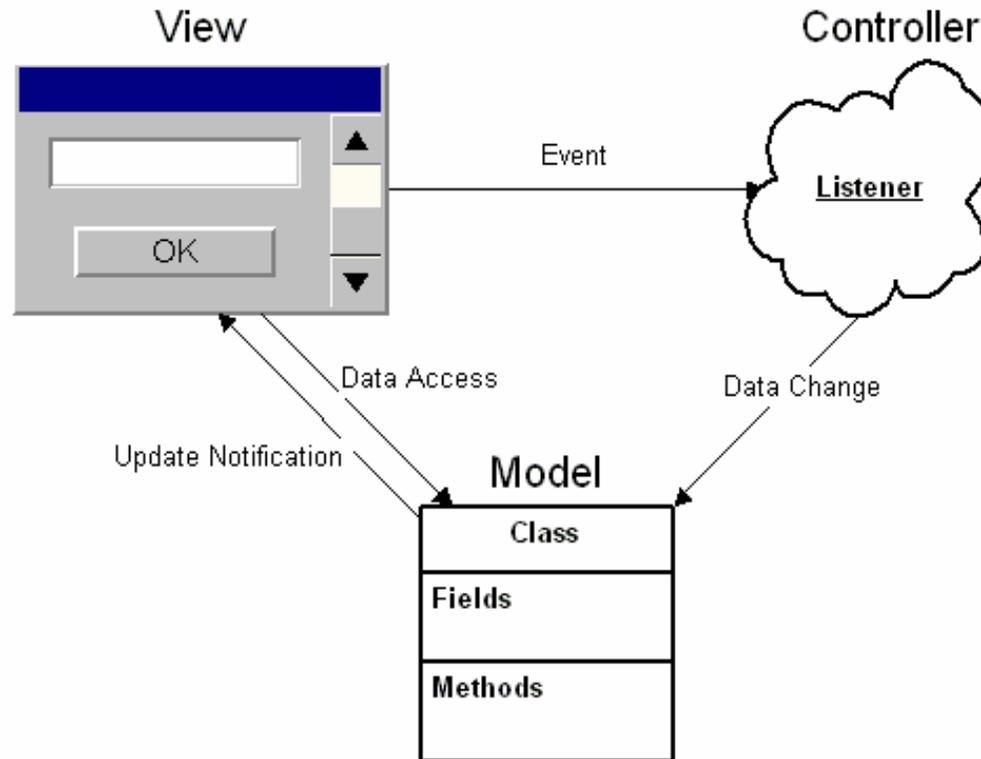


www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

- Ici le Modèle hérite de `java.util.Observable`
- La Vue implémente `java.util.Observer`

MVC encore

Model-View-Controller Architecture



- **Model** extends `java.util.Observable`
- **View** implements `java.util.Observer`
- **Controller** implements `XXXXListener`, `YYYYListener`

Nouvelle architecture

En conséquence

Au tp

- Le Modèle est une pile (classe **PileModele<T>**).
- La Vue correspond à l'affichage de l'état de la pile (classe **Vue**).
- Le Contrôleur gère les évènements issus des boutons +, -, *, /, [].

Architecture retenue

- Le Modèle est une calculette
- La Vue correspond à l'IHM (au complet).
- Le Contrôleur gère les évènements issus des boutons +, -, *, /, []

Architecture retenue

- **Le Modèle**

- La calculette munie de ses opérations (+,-,/,*,...)
 - Hérite de la classe `java.util.Observable`
 - Les sources du modèle sont ici
 - <http://douin.free.fr/tp4Calculette/>

- **La Vue**

- L'IHM affichage, zone de saisie, boutons ...
 - Implémente `java.util.Observer`

- **Le Contrôleur**

- Réalisation, implémentation des listeners, (le comportement de l'IHM)
 - Implémente plusieurs `ActionListener`

-> *pour Android, quel découpage ?, quelles classes ?*

Architecture pour Android

- **Le Modèle est inchangé**
 - La calculatrice munie de ses opérations (+,-,/,*,...)
 - <http://douin.free.fr/tp4Calculatrice/>
- **L'activity est associée à un écran, elle est constituée**
 - **La Vue**
 - L'IHM affichage, zone de saisie, boutons ... description XML
 - **Le Contrôleur**
 - Réalisation, implémentation des listeners, (le comportement de l'IHM)
 - Implémente plusieurs OnClickListener

Android, la classe Activity

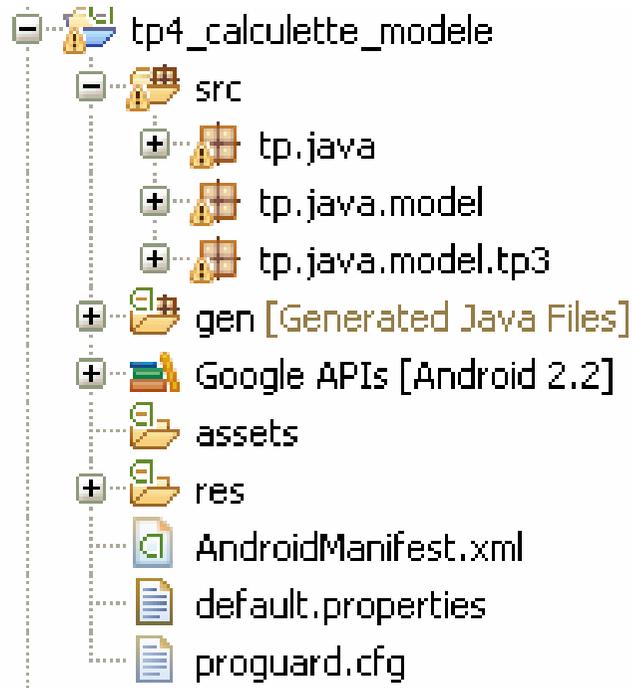
- **Activité comme application élémentaire**
 - À cette activité lui correspond une IHM, ce que l'on voit ...
public class Calculette extends Activity {
 - Cette IHM est décrite par un fichier XML (la vue)
 - L'activité réagit aux sollicitations de l'utilisateur (le contrôleur)

L'IHM de la calculette



- **Un fichier XML décrit complètement cette interface**
- **L'activité Calculette affiche, présente cette interface**

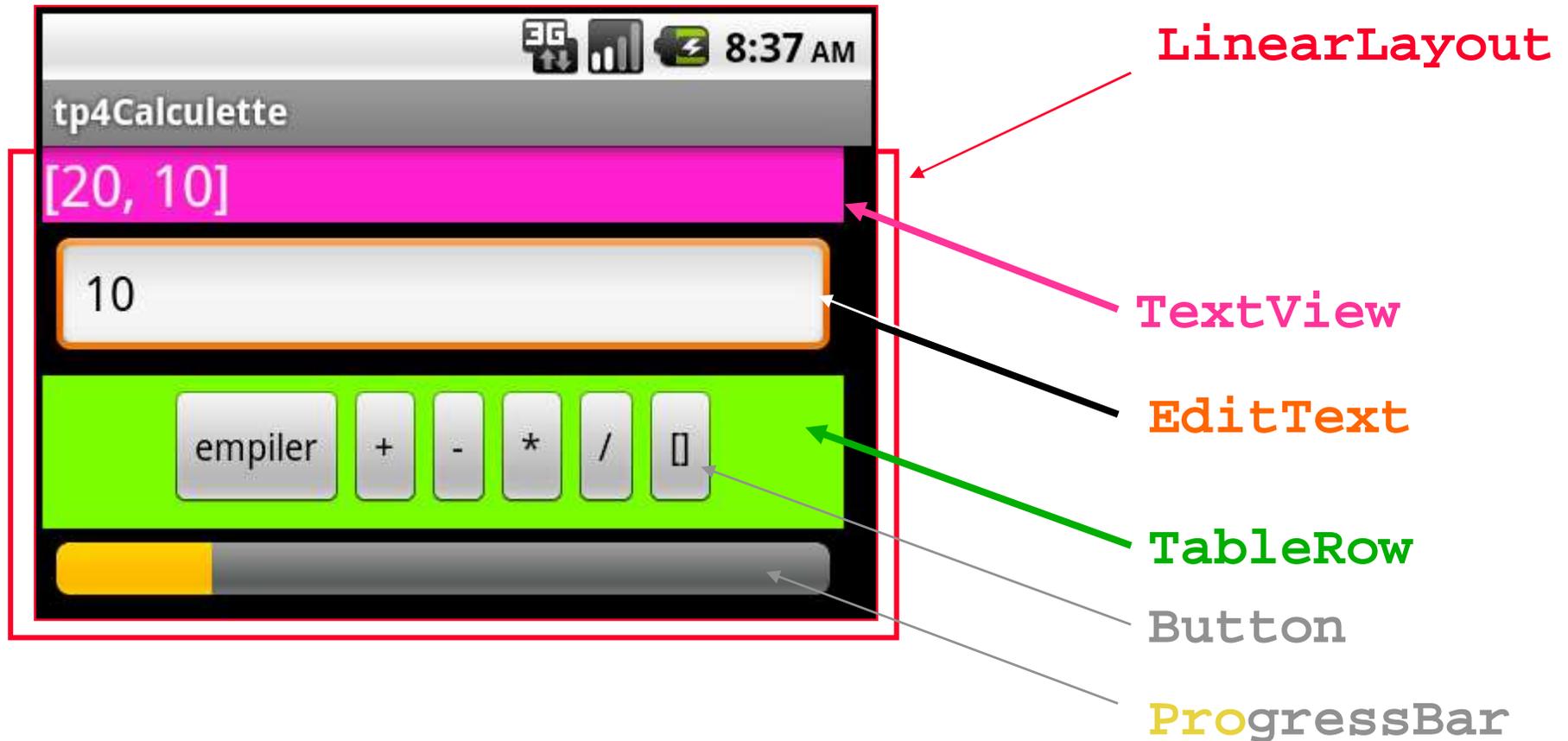
Android, la calculette



- **L'activité affiche l'IHM**
- **Le modèle ne change pas**
 - Cf. le TP

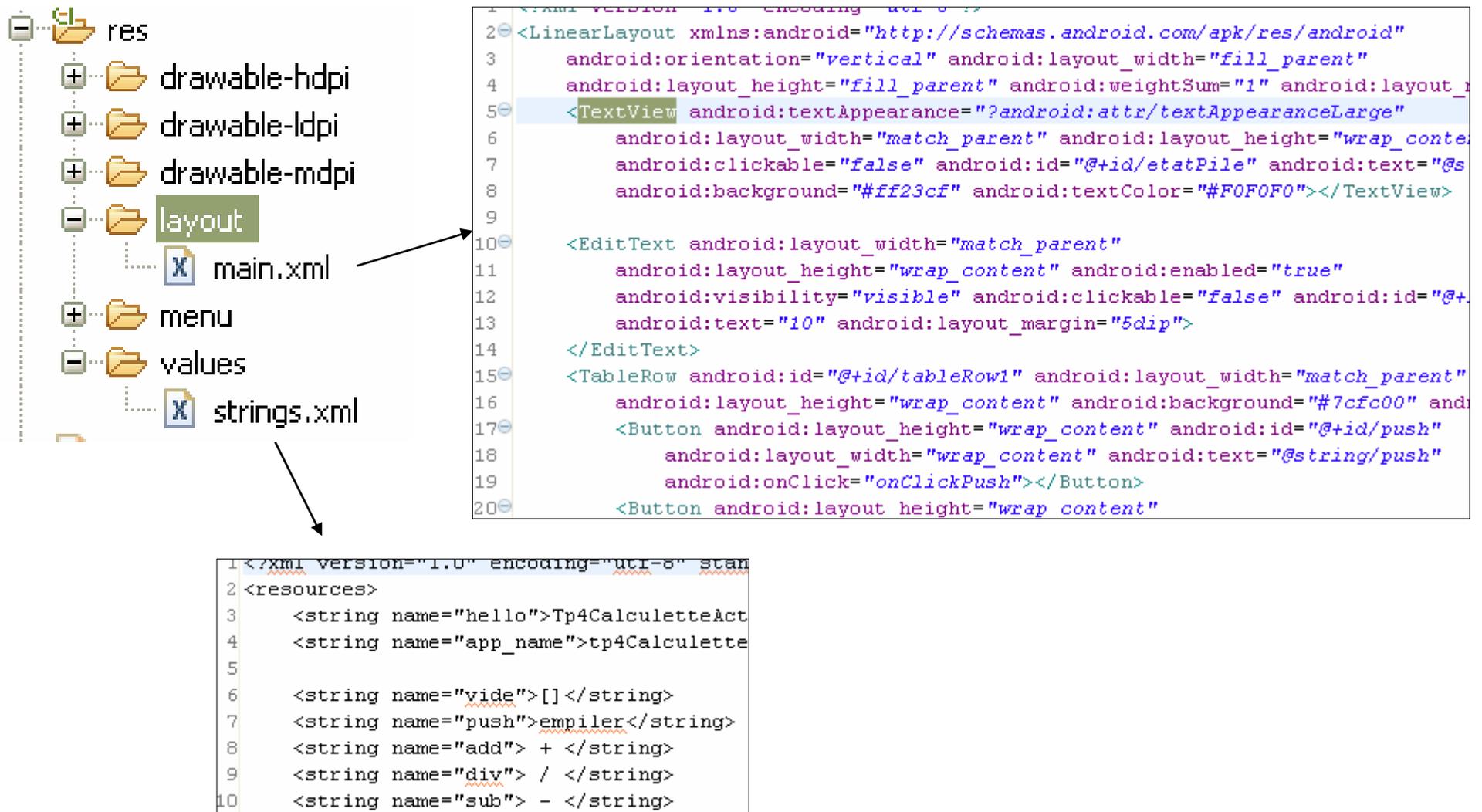
- **Android : Démonstration ...**

IHM : Layout, View , Button...



- **Description de cette interface en XML**
 - Fichier `res/layout/main.xml`

Interface, IHM : Approche déclarative ./res/



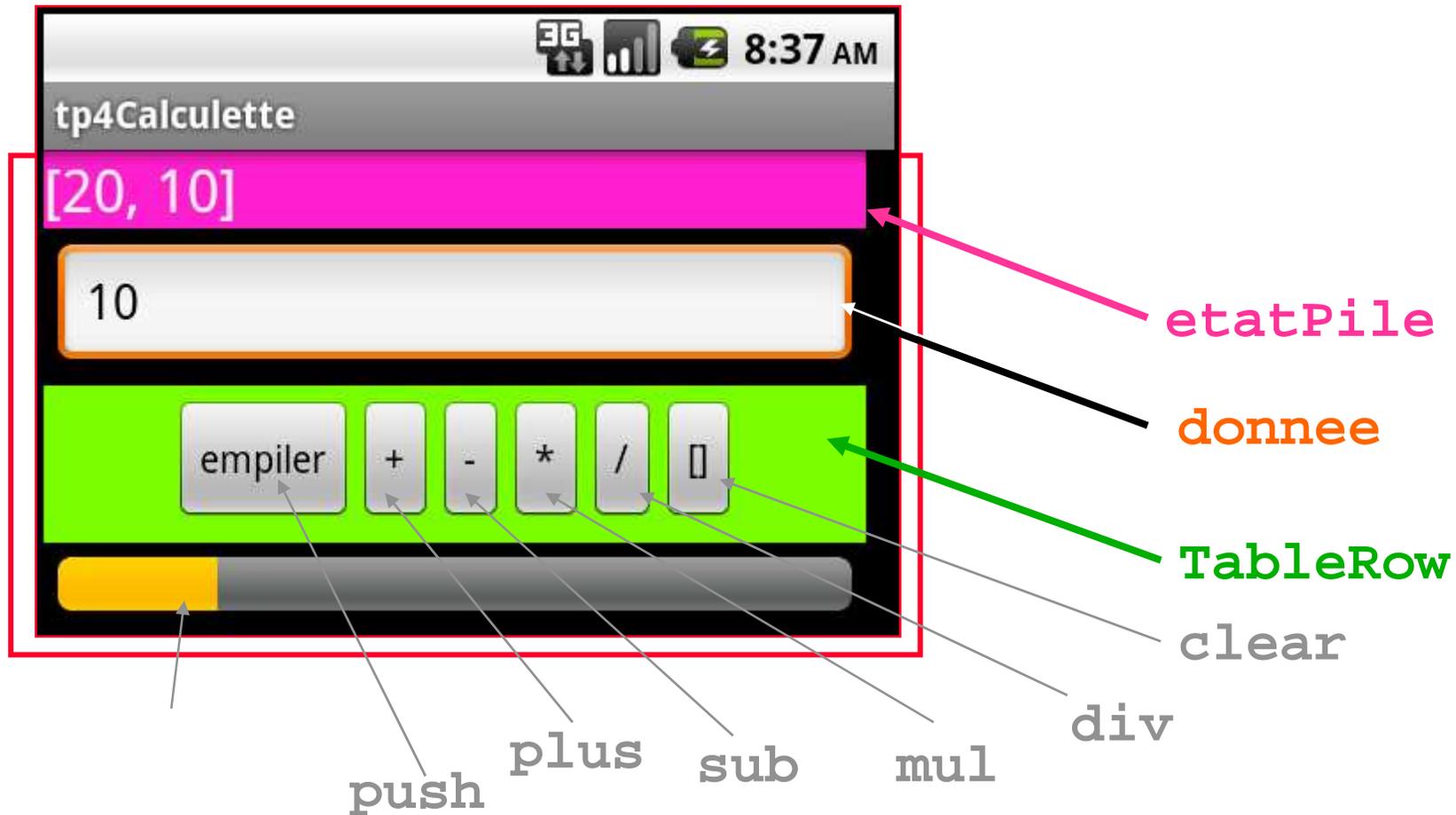
The image displays the 'res' directory structure in an Android IDE. The 'layout' folder contains 'main.xml', and the 'values' folder contains 'strings.xml'. Arrows point from these files to their respective XML code snippets.

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:orientation="vertical" android:layout_width="fill_parent"
4     android:layout_height="fill_parent" android:weightSum="1" android:layout_
5 <TextView android:textAppearance="?android:attr/textAppearanceLarge"
6     android:layout_width="match_parent" android:layout_height="wrap_conte
7     android:clickable="false" android:id="@+id/etatPile" android:text="@s
8     android:background="#ff23cf" android:textColor="#FOFOFO"></TextView>
9
10 <EditText android:layout_width="match_parent"
11     android:layout_height="wrap_content" android:enabled="true"
12     android:visibility="visible" android:clickable="false" android:id="@+
13     android:text="10" android:layout_margin="5dip">
14 </EditText>
15 <TableRow android:id="@+id/tableRow1" android:layout_width="match_parent"
16     android:layout_height="wrap_content" android:background="#7cfc00" and
17 <Button android:layout_height="wrap_content" android:id="@+id/push"
18     android:layout_width="wrap_content" android:text="@string/push"
19     android:onClick="onClickPush"></Button>
20 <Button android:layout height="wrap content"
```

```
1 <?xml version="1.0" encoding="utf-8" stan
2 <resources>
3     <string name="hello">Tp4CalculetteAct
4     <string name="app_name">tp4Calculette
5
6     <string name="vide">[]</string>
7     <string name="push">empiler</string>
8     <string name="add"> + </string>
9     <string name="div"> / </string>
10    <string name="sub"> - </string>
```

- Chaque composant possède un id (android:id= "@+id/push")

Chaque composant a son id



- `<Button android:id="@+id/push">`

IHM, un outil de conception sous eclipse

- **Item comme classe**

- **Nom de la balise**

comme

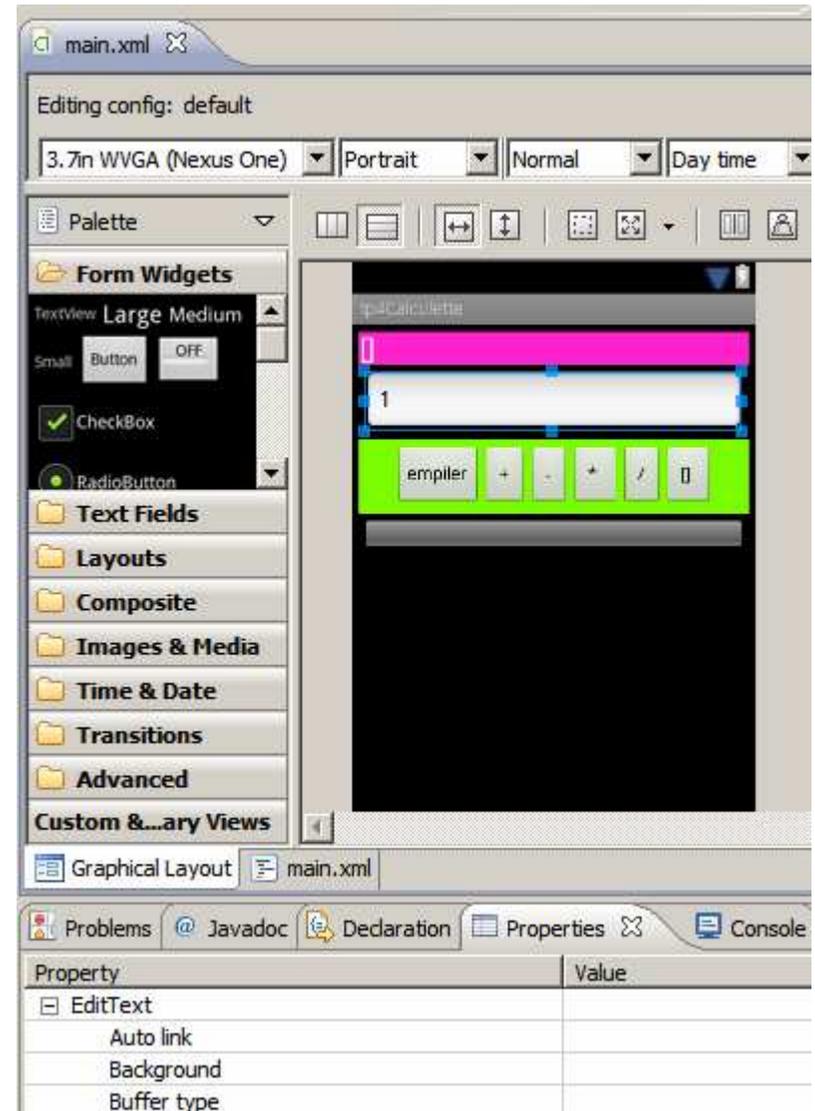
- **Nom de la classe**

- **Properties de chaque item**

- **Attribut XML, cf. Properties**

comme

- **Mais aussi comme Attribut de la classe,**



Adéquation XML <-> java, le fichier R

- En XML
- **<EditText**

```
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:enabled="true"  
android:visibility="visible"  
android:clickable="false"  
android:layout_margin="5dip"  
android:numeric="decimal"  
    android:inputType="number"  
  
android:id="@+id/donnee"  
android:text="1" />
```

...

```
<Button android:id="@+id/push"
```

En Java

```
EditText donnee = (EditText) findViewById(R.id.donnee);  
Button empiler = (Button) findViewById (R.id.push);
```

Intégration de l'IHM, R.layout.main

- **Au sein d'une Activity**

- **XML : accès en java via R**

- **Les ressources XML sont accessibles via le fichier R**
 - **R.java est généré par Android**
 - **Convention : /layout/main -> R.layout.main, R.id. R.string. ...**
»R cf. dossier /gen/

Intégration de l'IHM, R.layout.main

- **Au sein d'une Activity**

- Au préalable l'affectation de l'interface par l'appel de
 - **setContentView(R.layout.main);**

- **Les composants de l'IHM deviennent accessibles**

- **Button empiler = (Button) findViewById(R.id.push);**

- **findViewById est une méthode héritée de la classe Activity**

- Le source java ne manquera pas de R, ni d'appels de R (facile ...)

Une Activity, « démarrage » par onCreate

```
public class TPCalculatrice extends Activity {
```

```
    @Override
```

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

```
        // Affectation de l'IHM issue des fichiers XML
```

```
        setContentView(R.layout.main);
```

Une Activity accès aux composants

```
public class TPCalculatrice extends Activity {
```

```
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
```

```
// accès aux composants de l'IHM
```

```
Button empiler = (Button) findViewById(R.id.push);
```

```
ProgressBar jauge = (ProgressBar) findViewById(R.id.jauge);
```

```
// accès aux chaînes de caractères ( plusieurs langues)
```

```
String str = getString(R.string.app_name);
```

Comportement : OnClickListener et plus

- Un seul « Listener » par composant
- **Button empiler = (Button) findViewById(*R.id.push*);**

```
empiler.setOnClickListener(new View.OnClickListener(){  
    public void onClick(View v){  
        // traitement associé  
    }  
})
```

Ou bien usage de l'Attribut onClick

- **Extrait de layout/main.xml**

- `<Button android:layout_height="wrap_content" android:id="@+id/push"`
- `android:layout_width="wrap_content" android:text="@string/push"`
- `android:onClick="onClickEmpiler">`
- `</Button>`

- **Dans la classe de l'activity**

```
public void onClickEmpiler(View v){  
    // traitement associé  
}
```

Démonstration

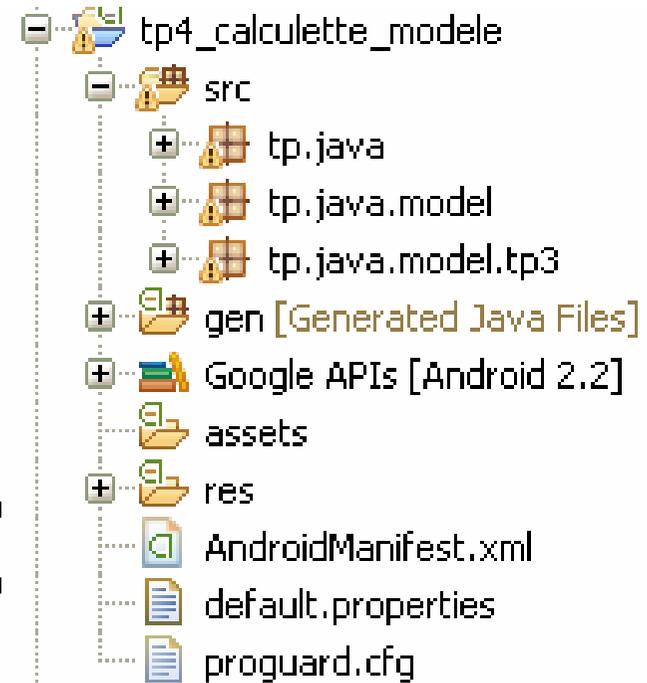
Une démonstration

- Avec au moins un bouton ... de bienvenue
 - À chaque clic la chaîne hello world est modifiée

Android : la description de l'application

- **Description de l'application**

- Quelles activités ?
 - Plusieurs activités pour une application
- Quelles permissions ?
 - SMS, Réseau, GPS, ...
- Quelles librairies ?



- **Dans un fichier XML**

- **AndroidManifest.xml**

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

AndroidManifest.xml, tp4Calcullette

- La description de l'application,
 - destinée à l'hébergeur

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

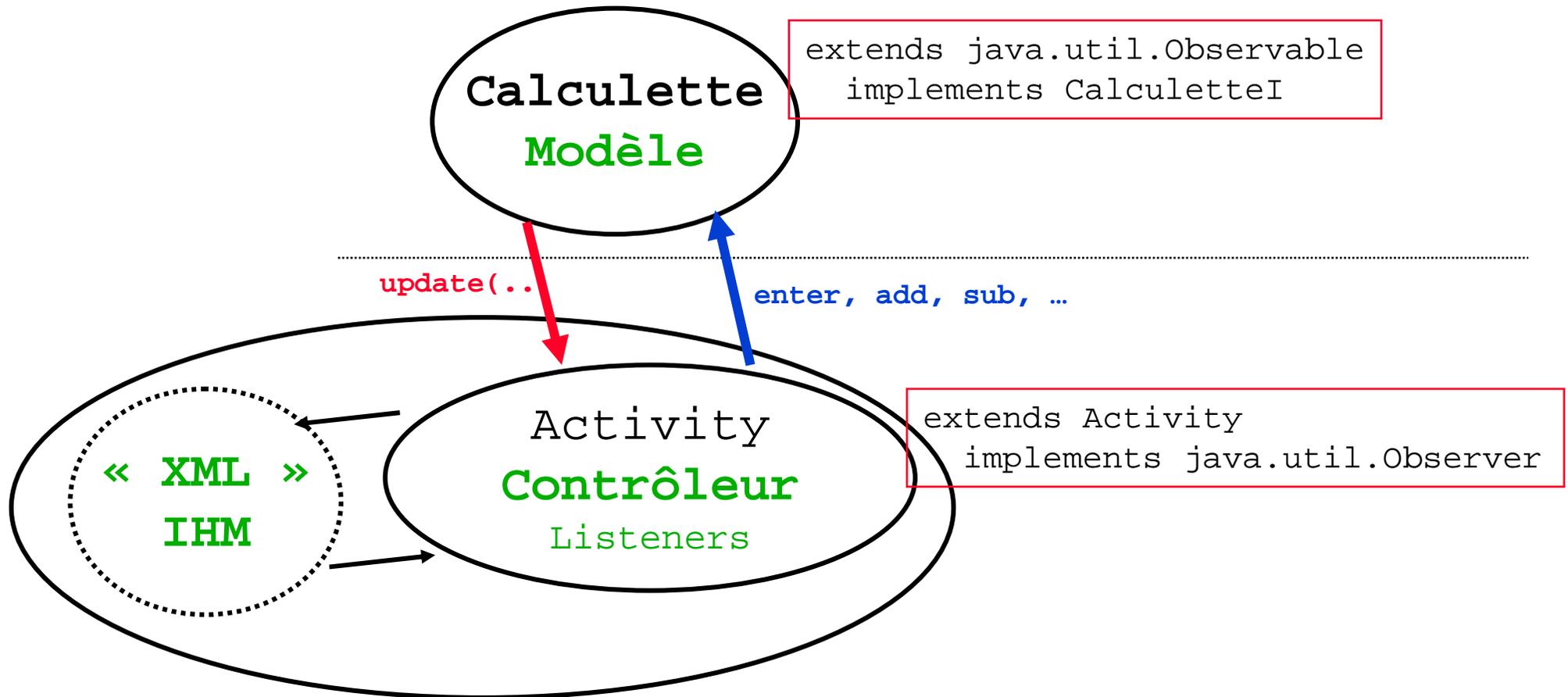
- ```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="tp.java" android:versionCode="1" android:versionName="1.0">
 <uses-sdk android:minSdkVersion="8" />
```
- ```
<application android:icon="@drawable/icon_calculator"
  android:label="@string/app_name">
  <uses-library android:name="android.test.runner" />
```
- ```
<activity android:name=".Tp4CalculletteActivity"
 android:label="@string/app_name">
 - <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
 </intent-filter>
 </activity>
</application>
</manifest>
```

# MVC et Android

---

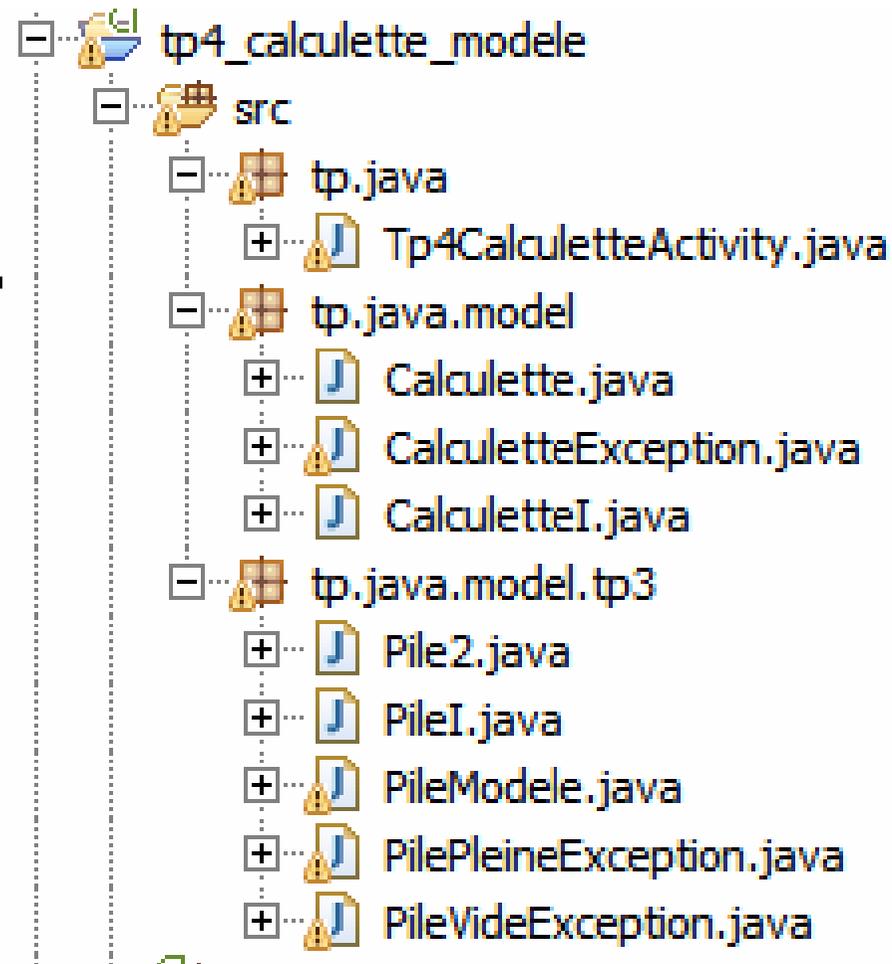
- **Nous avons :**
  - Une IHM décrite en XML
  - Une activité qui implémente le comportement
- **Alors**
  - L'activité est le contrôleur de la vue, de l'IHM décrite en XML
  - L'activité met l'IHM au premier plan,
    - l'activité a accès aux composants graphiques
- **MVC ?**
  - Le modèle est une calculatrice
  - La vue du modèle est l'activité

# MVC, Mise en Pratique : discussions



- L'activity Android est une vue du Modèle Calculette (implements Observer)
- L'activity Android est le contrôleur de l'IHM décrite en XML (extends Activity)

# packages



**modèle**

# Le Modèle : la Calculette

---

```
public interface CalculetteI {
 // operations
 void enter(int i) throws CalculetteException;

 void add() throws CalculetteException;
 void sub() throws CalculetteException;
 void div() throws CalculetteException;
 void mul() throws CalculetteException;

 void clear();
 int pop() throws CalculetteException;

 // interrogations
 int result() throws CalculetteException;
 boolean isEmpty();
 boolean isFull();
 int size();
 int capacity();
}
```

```
public class Calculette
 extends java.util.Observable
 implements CalculetteI
```

## TP4CalculletteActivity, la vue du modèle + le Contrôleur de l'IHM

---

```
public class Tp4CalculletteActivity extends Activity implements Observer{
 private Calcullette calcullette;

 public void onCreate(Bundle savedInstanceState) { // appelée par Android
 super.onCreate(savedInstanceState);
 this.calcullette = new Calcullette(); // une calcullette est créée
 this.calcullette.addObserver(this); // c'est une vue du modèle calcullette
 setContentView(R.layout.main); // l'IHM est associée à cette activité

 }

 ...
 public void onClickEmpiler(View v){ // attribut onClick balise <Button
 }

 ...
 public void update(Observable arg0, Object arg1) { // à chaque notification
 }
}
```

## TP4CalculletteActivity, le Contrôleur de l'IHM

```
public class Tp4CalculletteActivity extends Activity implements Observer{
 private Calcullette calcullette;

 public void onCreate(Bundle savedInstanceState) { // appelée par Android
 ...

 }

 ...

 public void onClickEmpiler(View v){ // attribut onClick balise <Button
 try{
 int operande = ...
 this.calcullette.empiler(operande); // opération empiler sur le modèle
 }catch(CalculletteException nfe){}

 }

 public void onClickAdd(View v){
 ...

 public void update(Observable arg0, Object arg1) { // à chaque notification
 }
}
```

## TP4CalculletteActivity, est une vue du modèle

---

```
public class Tp4CalculletteActivity extends Activity implements Observer{
 private Calcullette calcullette;

 public void onCreate(Bundle savedInstanceState) { // appelée par Android

 }

 ...
 public void onClickEmpiler(View v){ // attribut onClick
 }

 ...

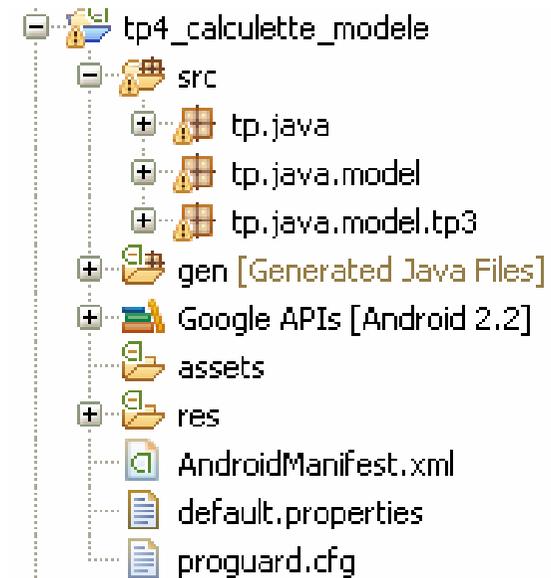
 public void update(Observable arg0, Object arg1) { // à chaque notification du
 TextView etat = (TextView) findViewById(R.id.etatPile); // modèle
 etat.setText(calcullette.toString());
 ProgressBar jauge = (ProgressBar) findViewById(R.id.jauge);
 jauge.setProgress(calcullette.size());
 actualiserInterface();
 }
}
```

## Architecture retenue

---

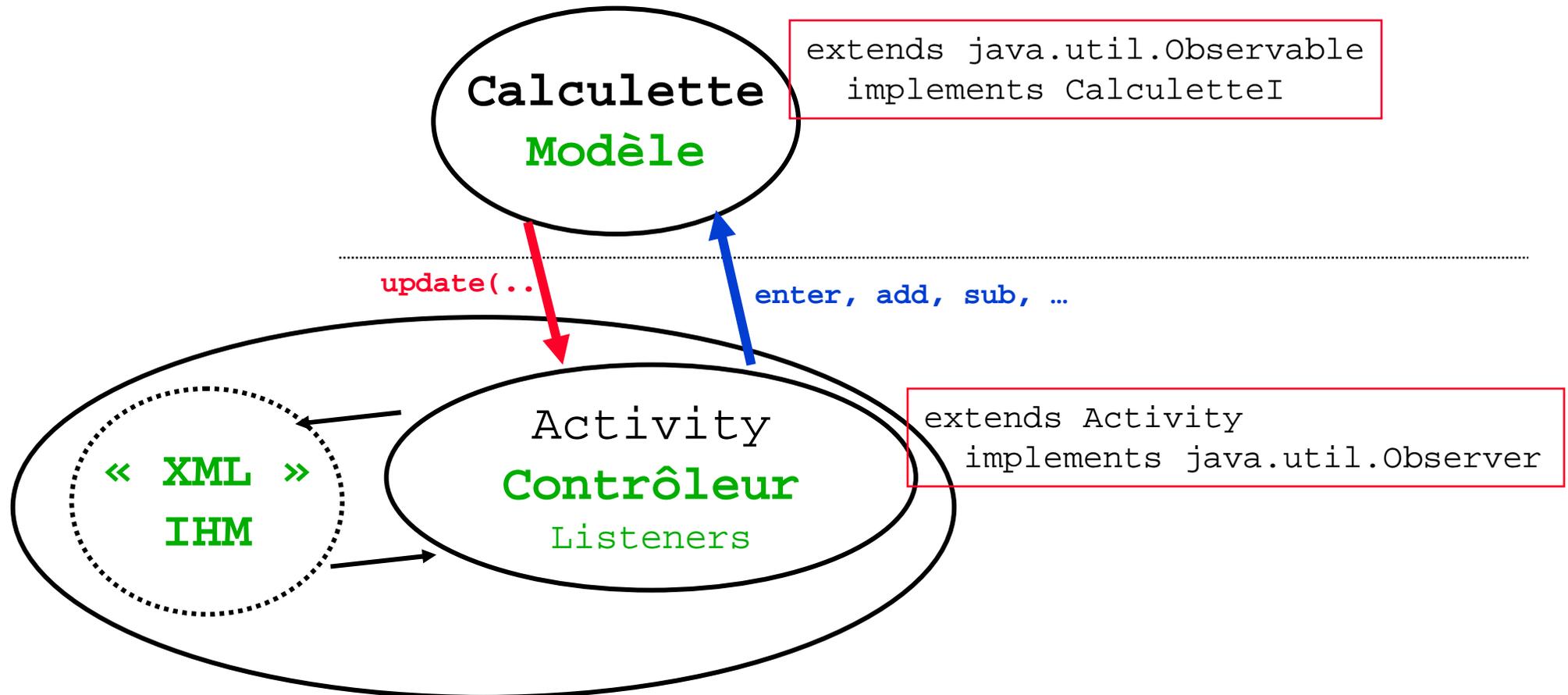
- **Application Calculette en résumé**
  - **Observable** : Le modèle, la calculette
  - **Observer** : l'Activity, mise à jour de l'interface
  
  - **View** : le fichier XML (l'IHM)
  - **Controller** : l'Activity

# Android, la calculette



- Discussion, réalisation ...

# Android et MVC, discussion



```
public class CalcuetteActivity
 extends Activity
 implements Observer{
```

# Comportements attendus, cycle de vie

---

- **3 + 2 == 5 ?**

- **Appui sur la touche « retour »**
  - Fin de l'activité



- **Appui sur la touche « HOME »**
  - L'activité est en Pause ...

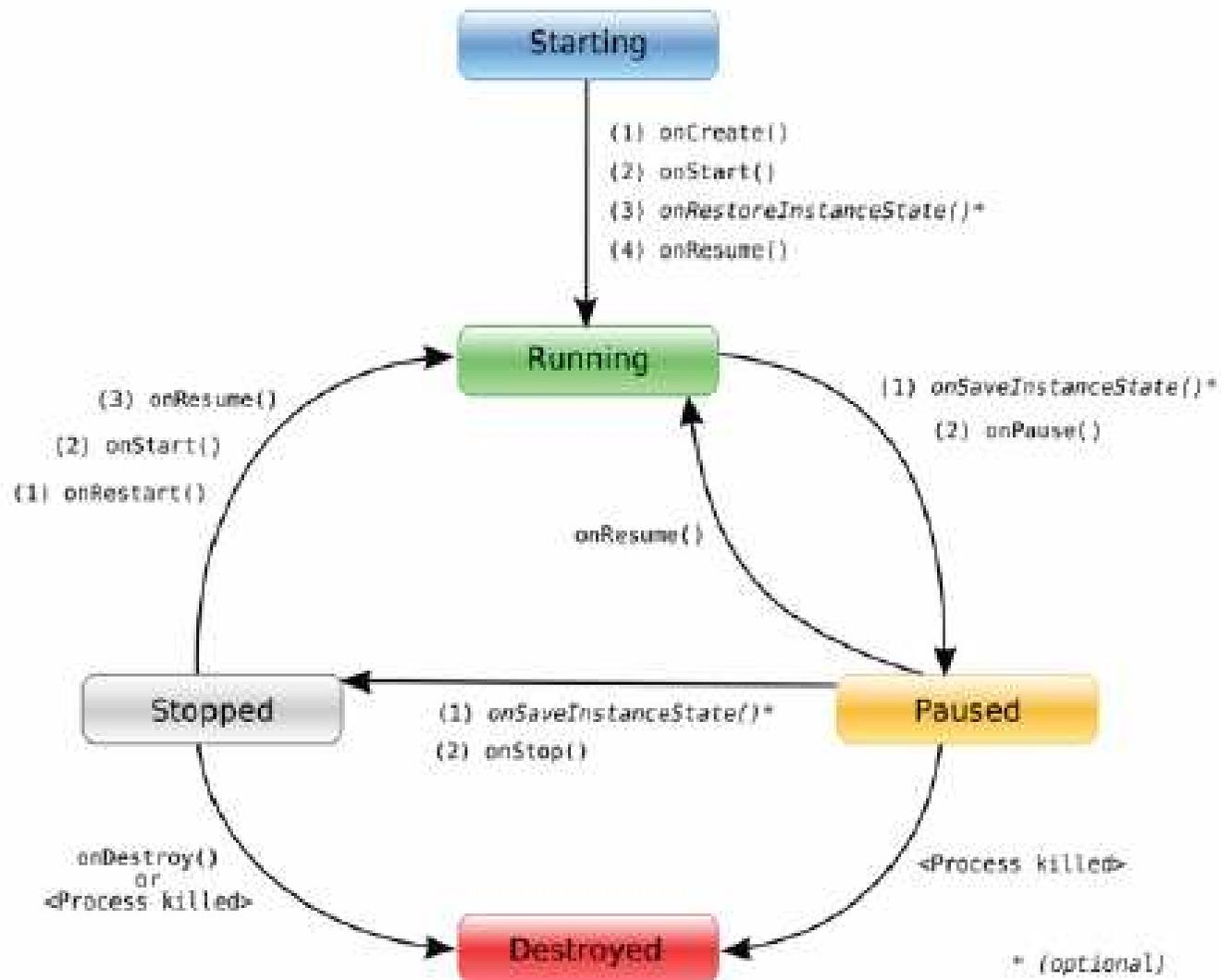


- **Pendant que je calcule 3 + 2, je reçois un urgent appel téléphonique**
  - telnet localhost 5554
  - gsm call 5554



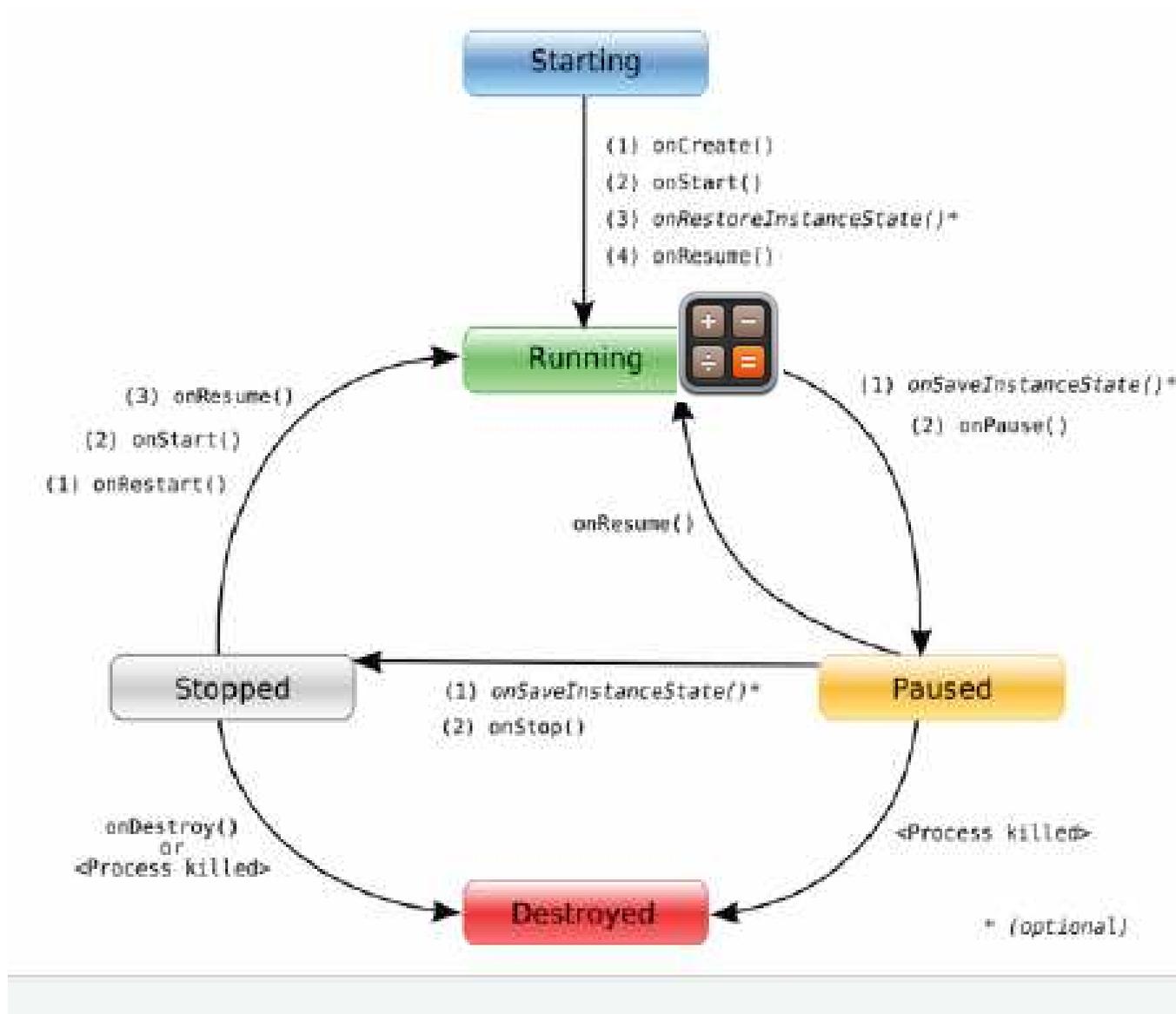
- **Une rotation de l'écran a lieu**
  - <http://developer.android.com/guide/developing/tools/emulator.html>
    - Ctrl-F11, Ctrl-F12

# Cycle de vie d'une activité

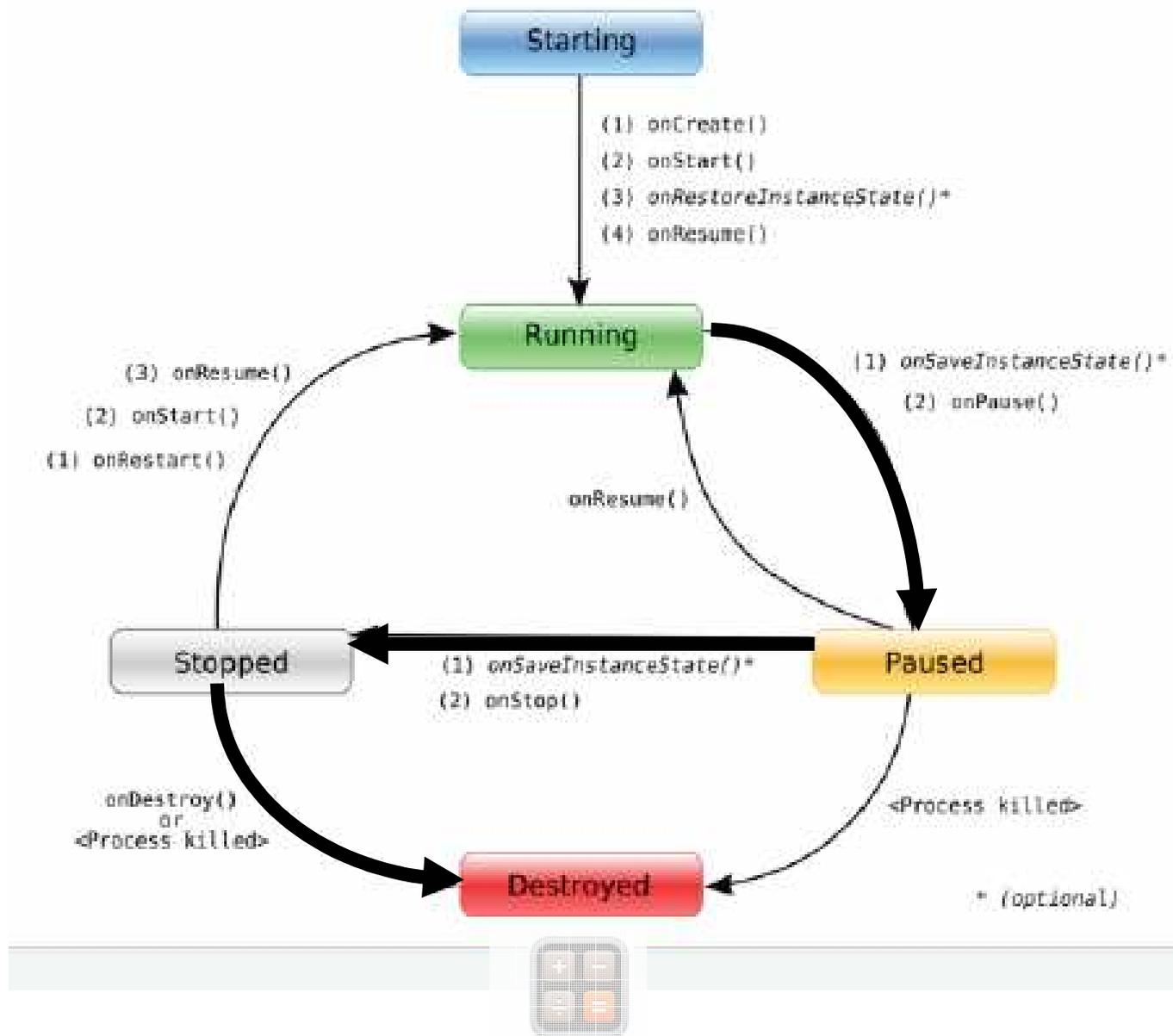


- Une première approche, par l'exemple

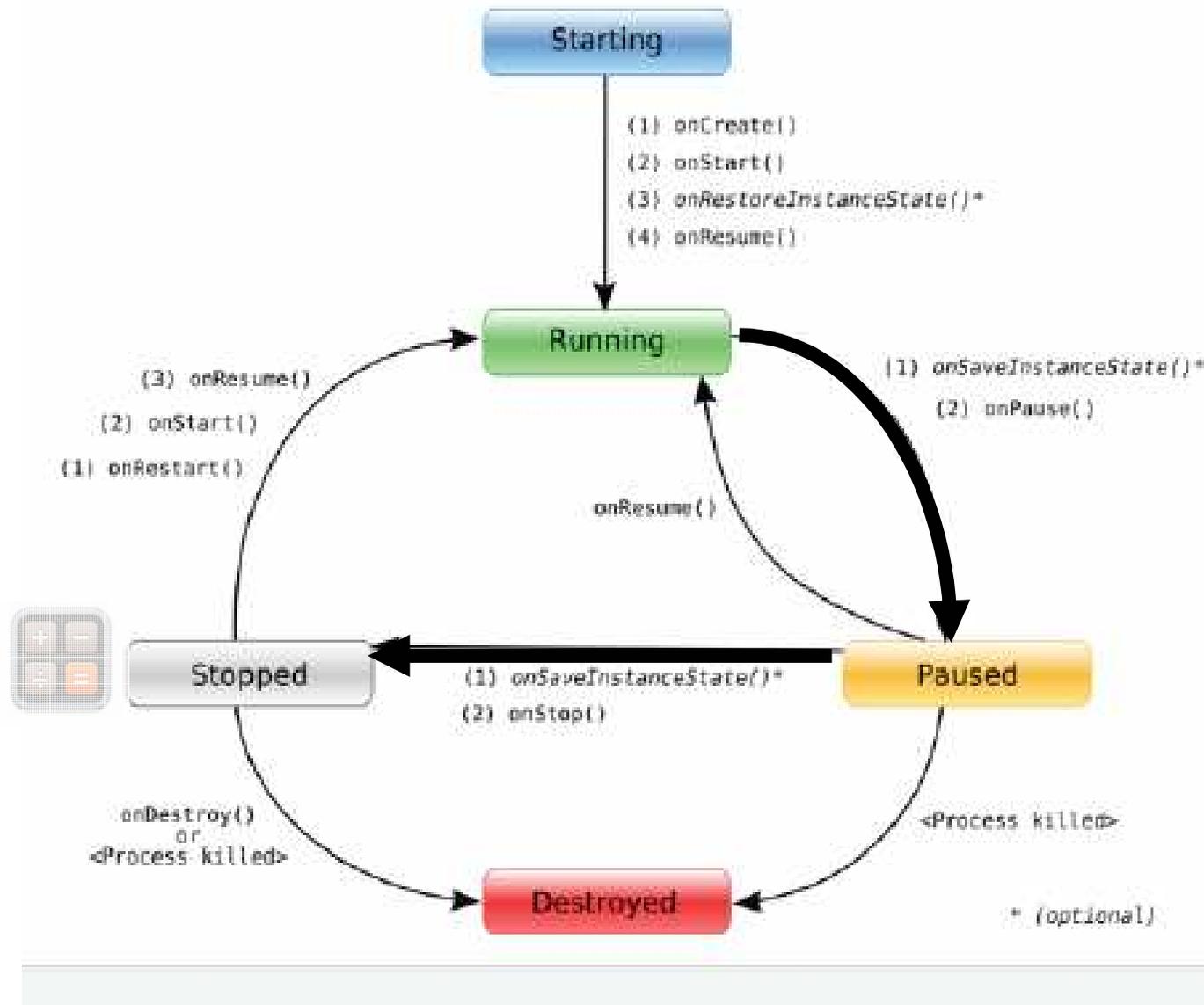
# Cycle de vie d'une activity, je calcule 3 + 2



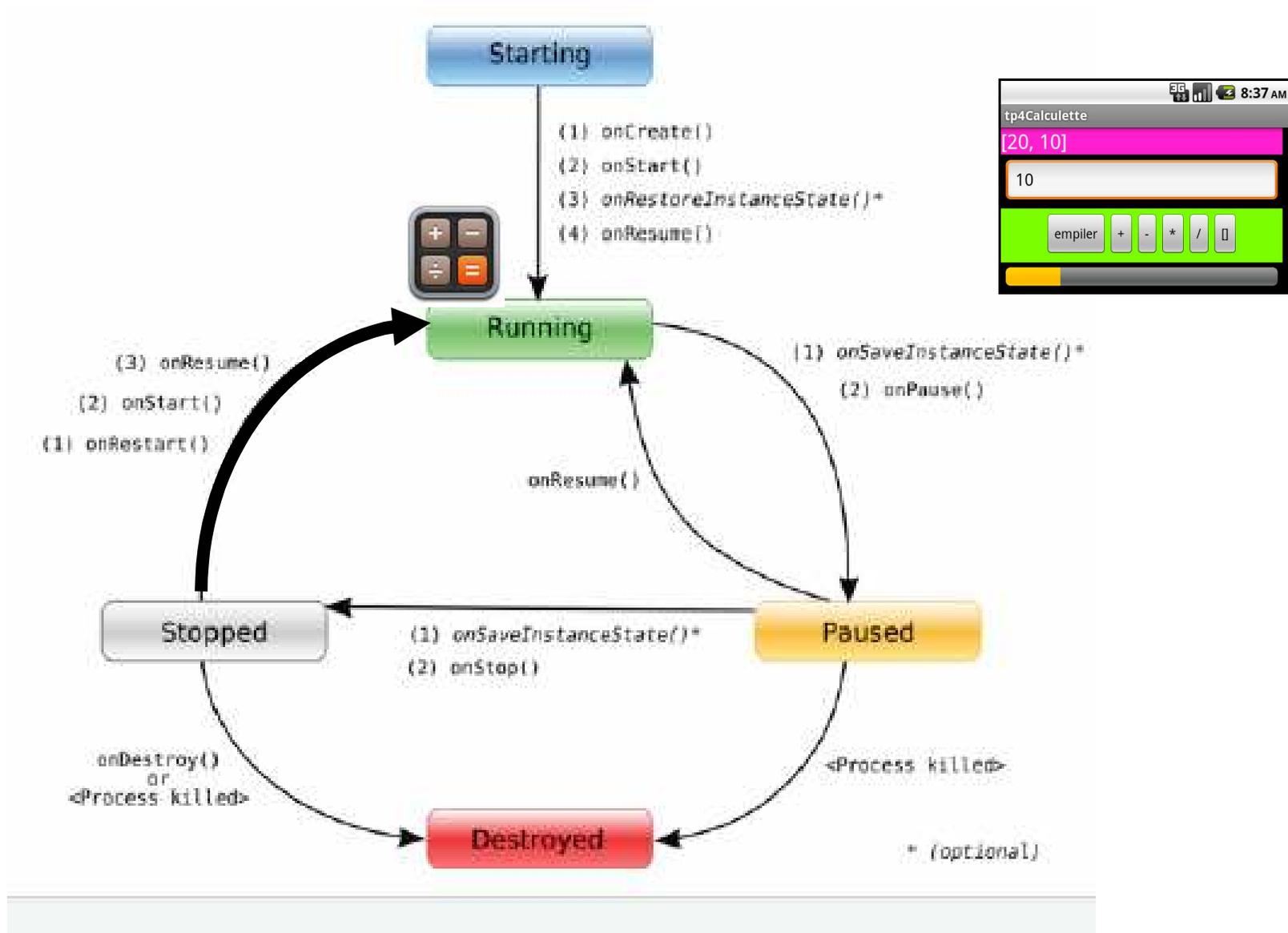
# Cycle de vie d'une activity, j'ai fini



# Cycle de vie d'une activity, je lance une autre activity

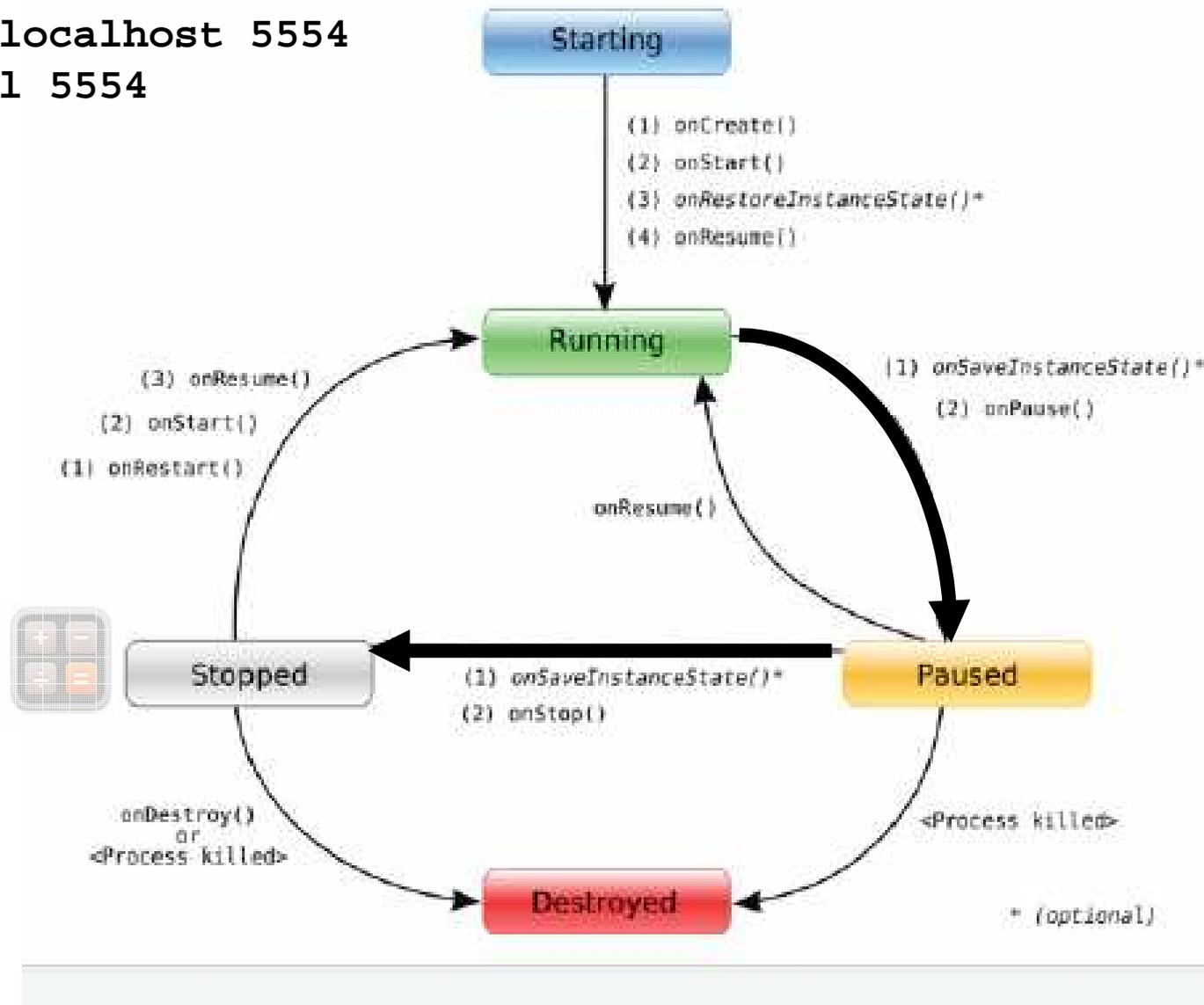


# Cycle de vie d'une activity, je calcule de nouveau

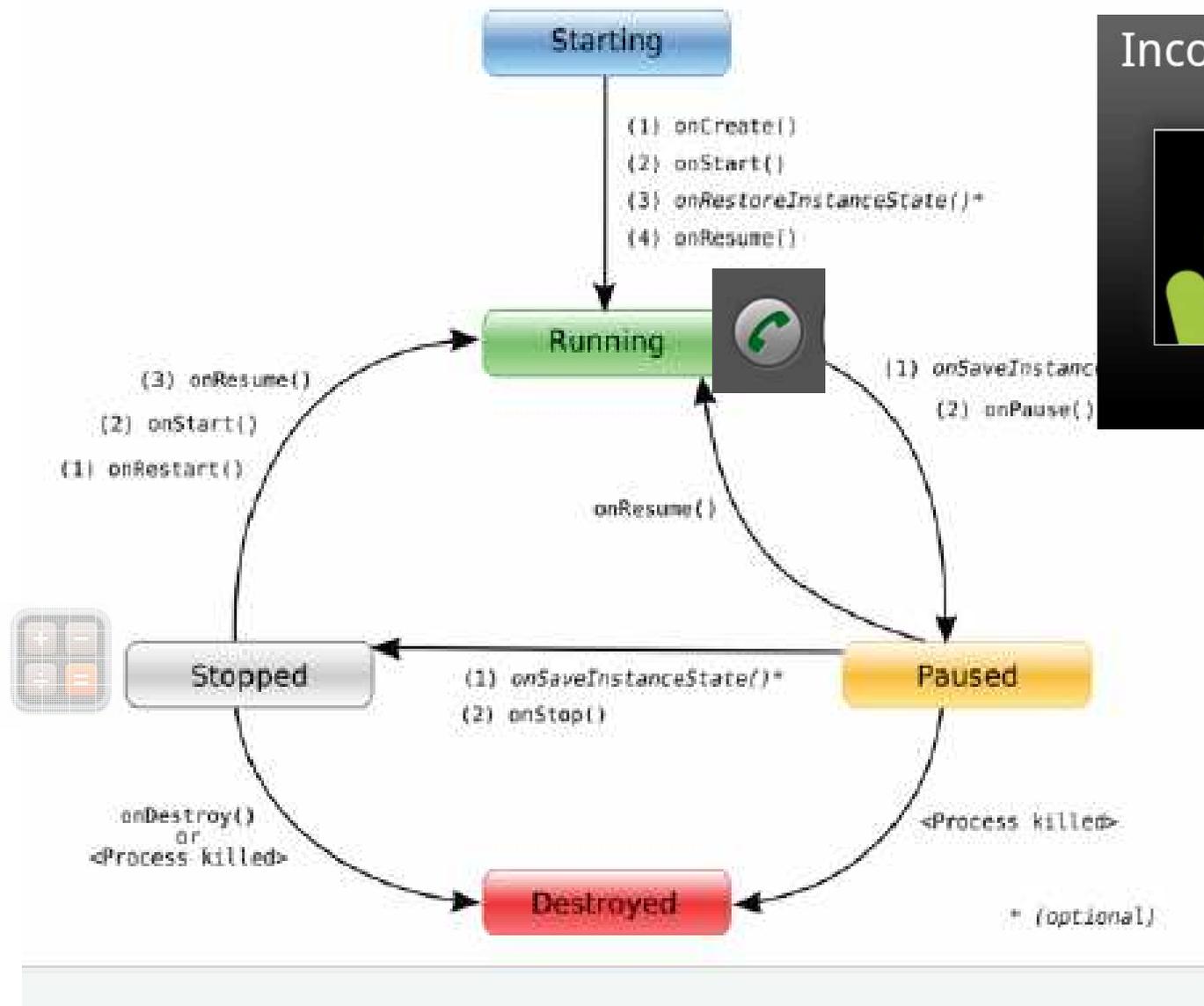


# Cycle de vie d'une activity, un appel urgent

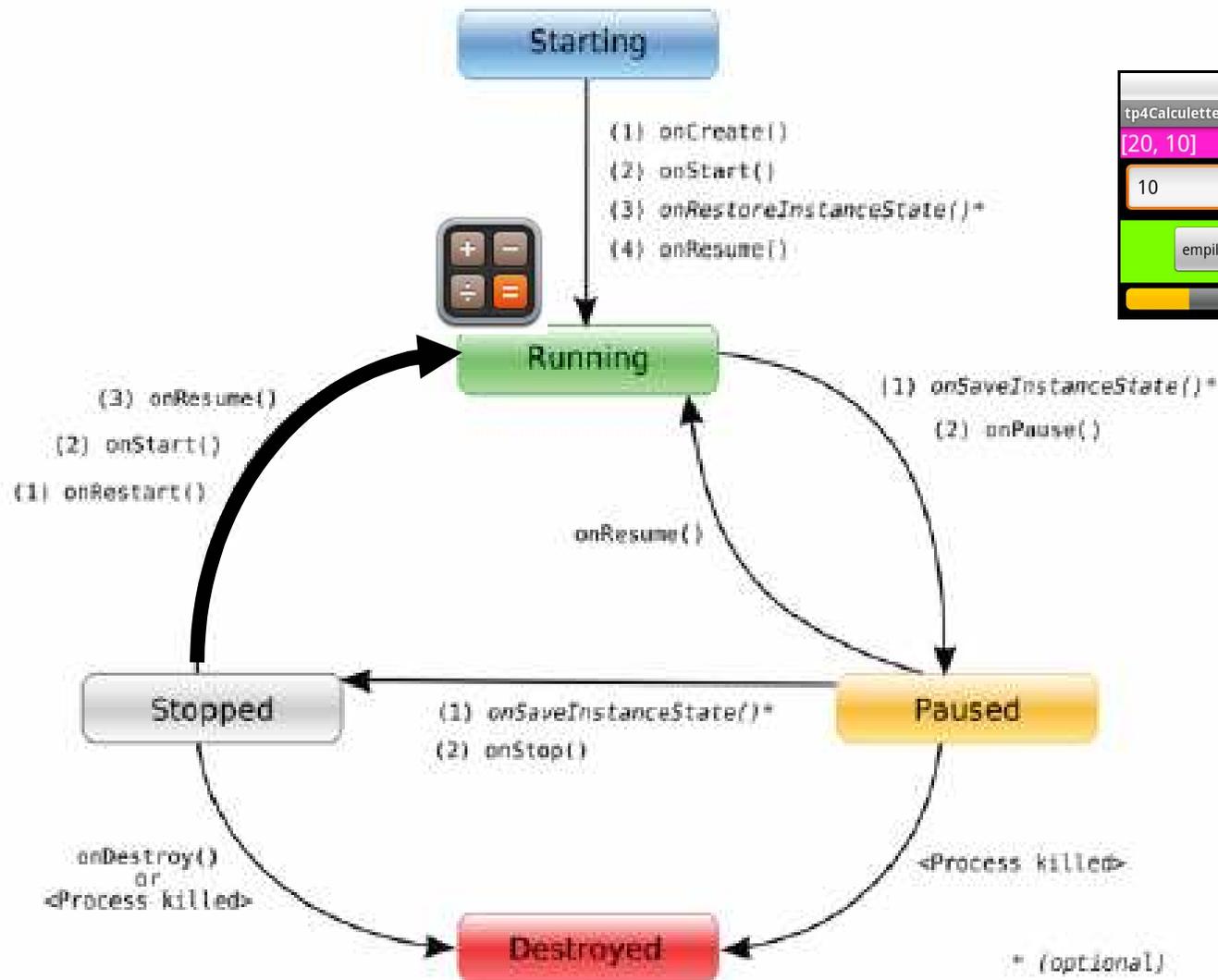
```
telnet localhost 5554
gsm call 5554
```



# Cycle de vie d'une activity, je réponds



# Cycle de vie d'une activity, je raccroche

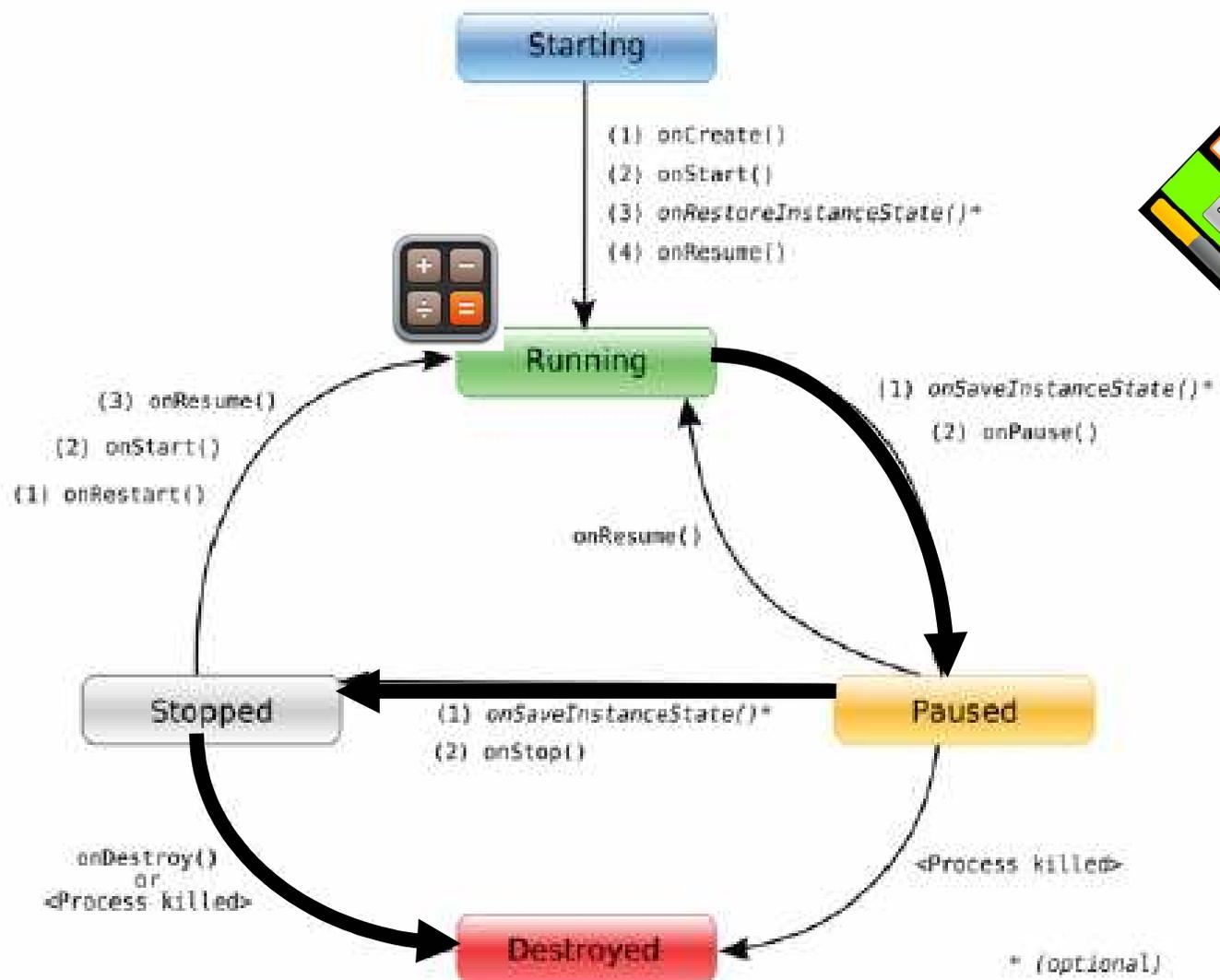


# Illustration du cycle de vie

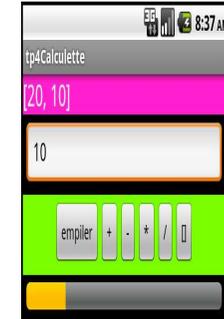
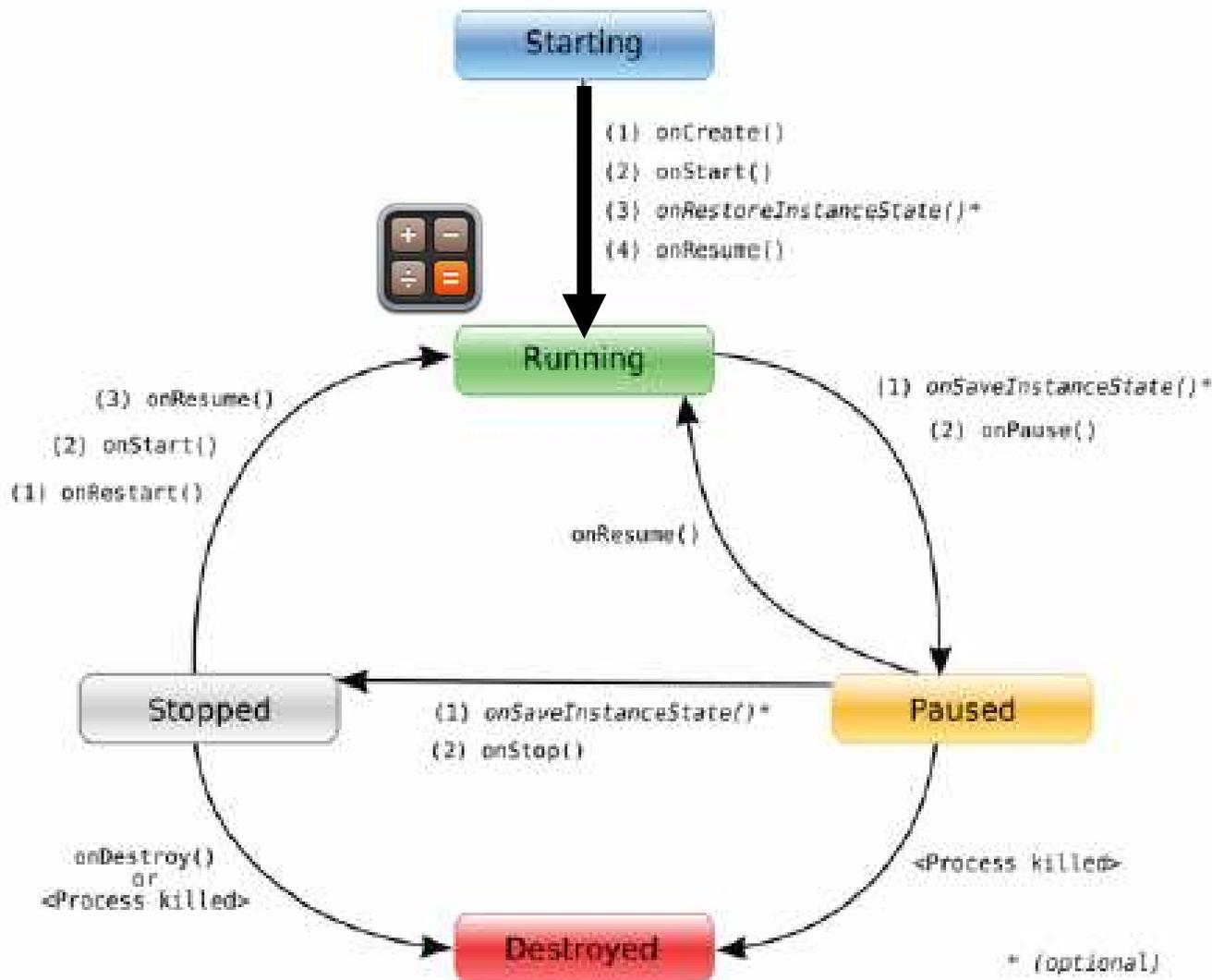
---

- **Illustration du cycle de vie**
  - Démonstration
- **Touche « Retour »**
  - onPause, onDestroy ....ok
- **Touche Menu**
  - onSaveInstanceState, onPause, onStop ...ok
  - Sauvegarde par défaut ...
- **Un appel idem**
- **Rotation de l'écran Ctrl-F11, Ctrl-F12**
  - ... !!!
    - Android détruit votre vue en cas de rotation pour construire la nouvelle vue

# Cycle de vie d'une activity, onSaveInstanceState



# Cycle de vie d'une activity, `onRestoreInstanceState`



# Mise en Pratique, suite

---

## – Suite du TP :

- Sauvegarde de l'état de la calculette
- Cf. cycle de vie

```
protected void onSaveInstanceState(Bundle out){
 out.putInt("taille",calculette.size());
 ...
}
```

```
protected void onRestoreInstanceState(Bundle in){
 int taille = in.getInt("taille");
 ...
}
```

## – Sauvegarde et restitution via le bundle

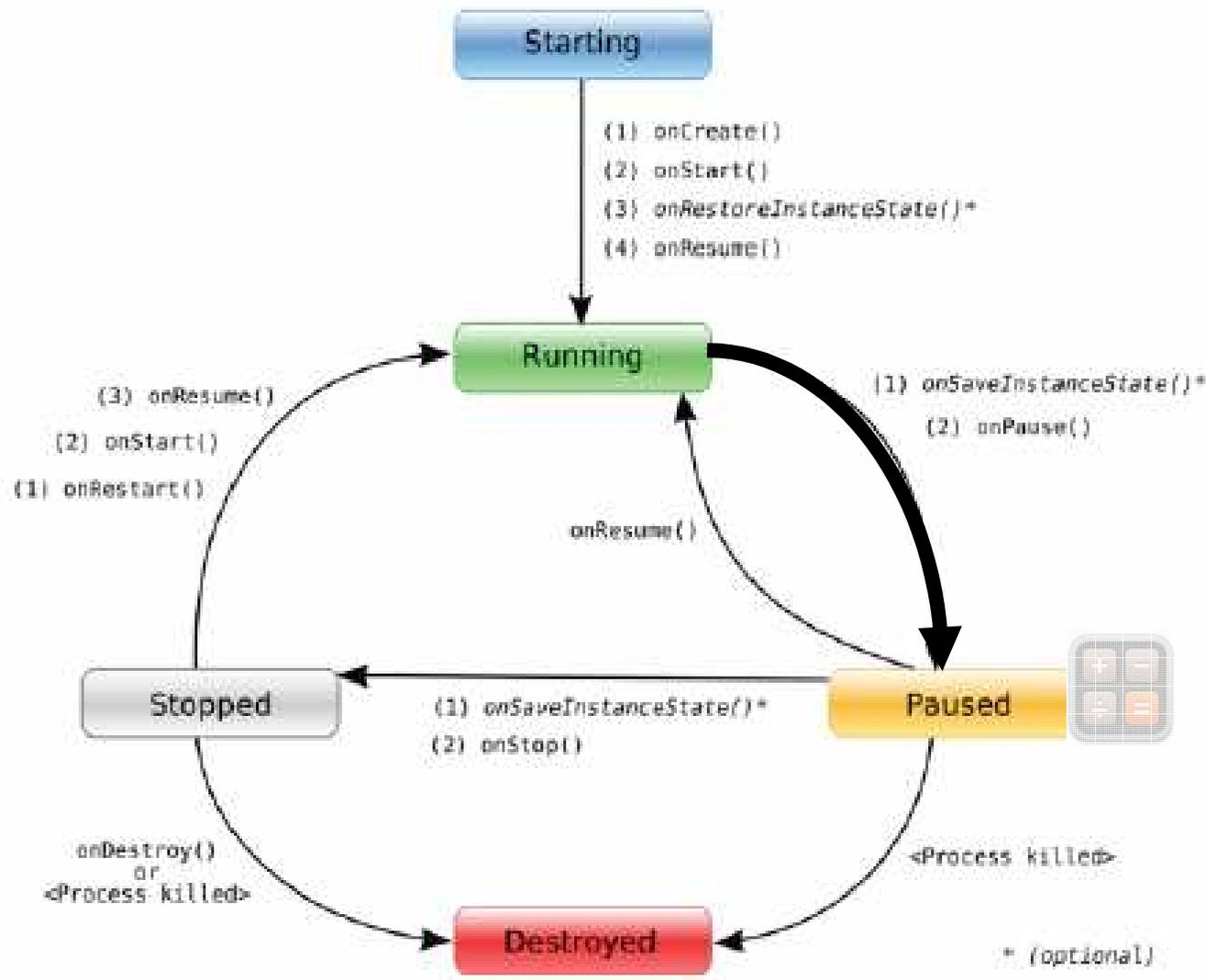
- <http://developer.android.com/reference/android/os/Bundle.html>

# Illustration du cycle de vie

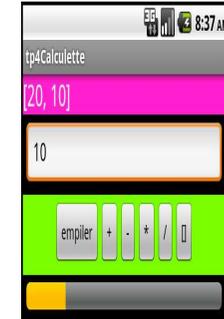
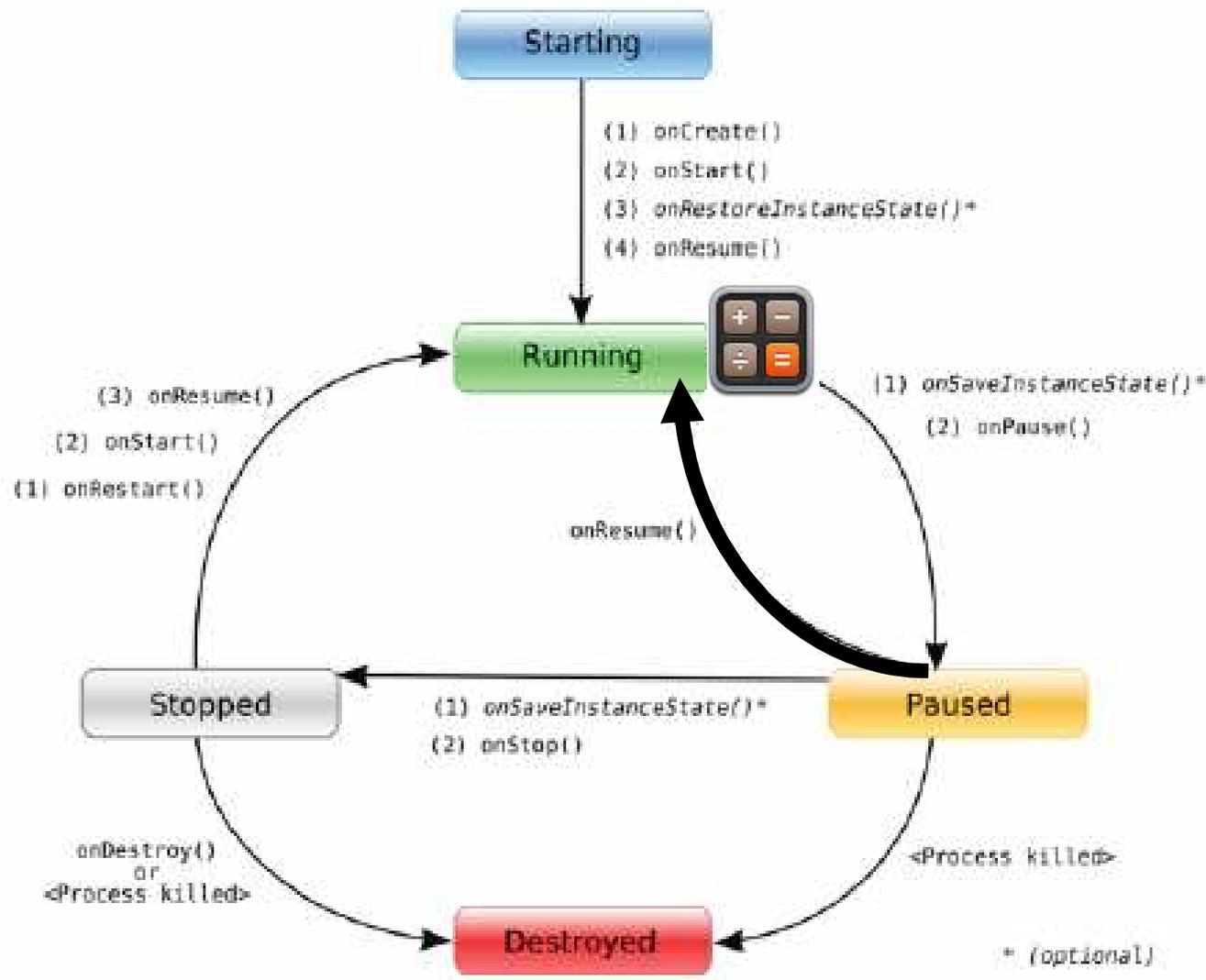
---

- **Ecran de veille alors que la calculette était au premier plan**
  - **onPause**
  - **Noté semi-visible dans la biblio ...**

# Cycle de vie d'une activity, écran de veille, touche F7



# Cycle de vie d'une activity, au réveil, touche F7



# Un extra

---

- **Ou une utilisation d'un service existant**
  - **Envoi d'un SMS**
    - **Depuis la calculette**
      - Une permission est à installer
    - **Depuis l'application prédéfinie**
      - Sélection d'une application avec certains critères

# Un extra pour la calculette

---

- **A chaque mauvais format du nombre, un sms est envoyé !**
  - Une exception Over The Air ...
- **Deux possibilités**
  - 1) l'activité Calculette envoie le SMS  
SmsManager sm = SmsManager.getDefault();  
**Si j'ai la permission**
  - 2) L'activité standard d'envoi de SMS est appelée
    - Intent sendIntent = new Intent(Intent.ACTION\_VIEW);
    - // le message ...
    - sendIntent.setType("vnd.android-dir/mms-sms");
    - startActivity(sendIntent);

# Au sein de votre application

---

```
private void sendSMS(String msg){
 try{
 SmsManager sm = SmsManager.getDefault();
 String body = getString(R.string.app_name) + " : " + msg + "\n";

 sm.sendTextMessage(getString(R.string.numero_tel), null, body,
 null, null);

// ou bien un Toast de simulation
Toast.makeText(getBaseContext(),
" envoi d'un sms " + msg, Toast.LENGTH_LONG).show();

 }catch(Exception e){
 Toast.makeText(getBaseContext(),getString(R.string.erreur),
 Toast.LENGTH_LONG).show();
 }
}
```

- Mais avez-vous la permission ? `<AndroidManifest`

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

# Une Activity en démarre une autre

---

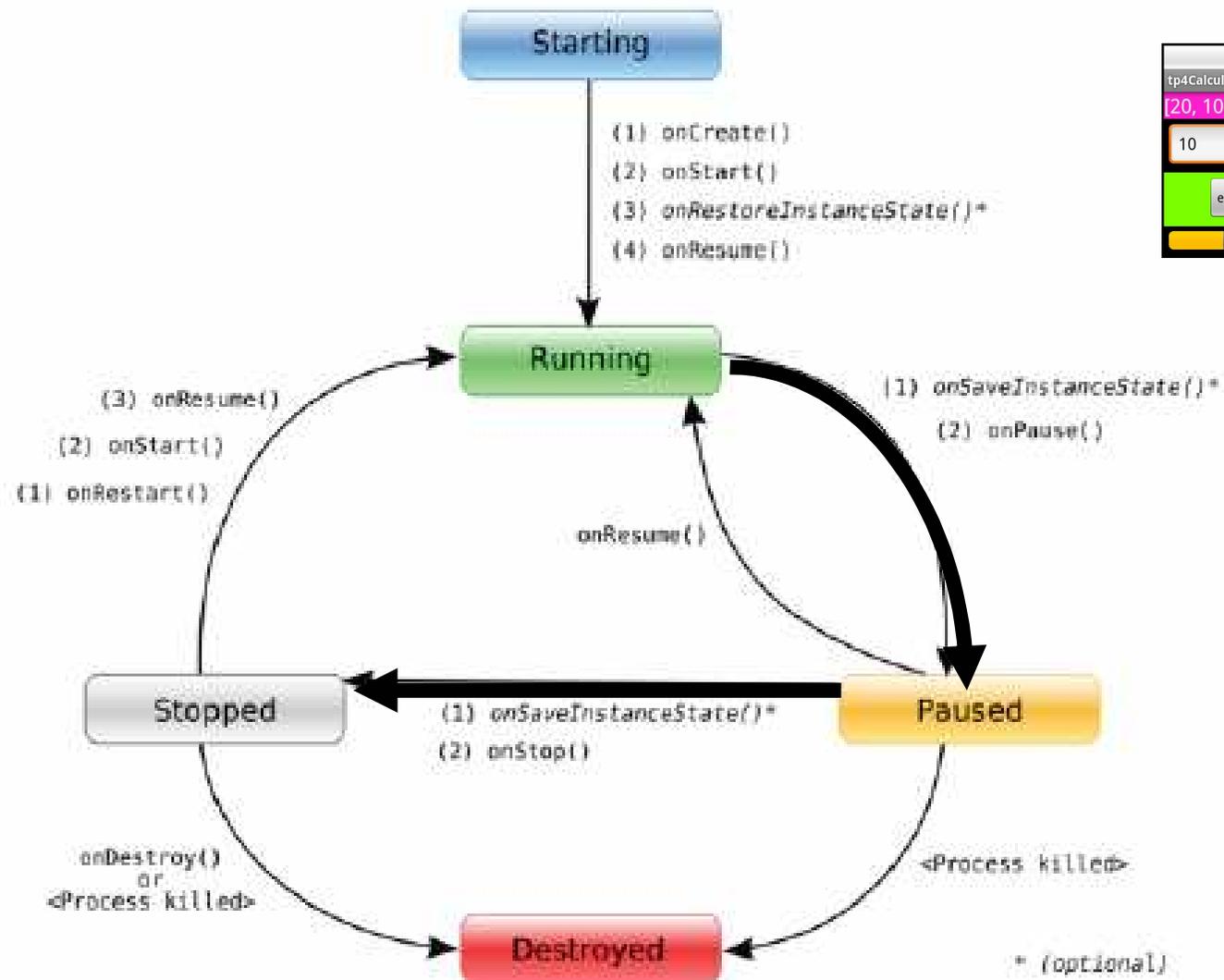
- Intent

ou comment transmettre des paramètres à une activité

- `Intent sendIntent = new Intent(Intent.ACTION_VIEW);`
- `sendIntent.putExtra("sms_body", "The SMS text");`
- `sendIntent.setType("vnd.android-dir/mms-sms");`
- `startActivity(sendIntent);`

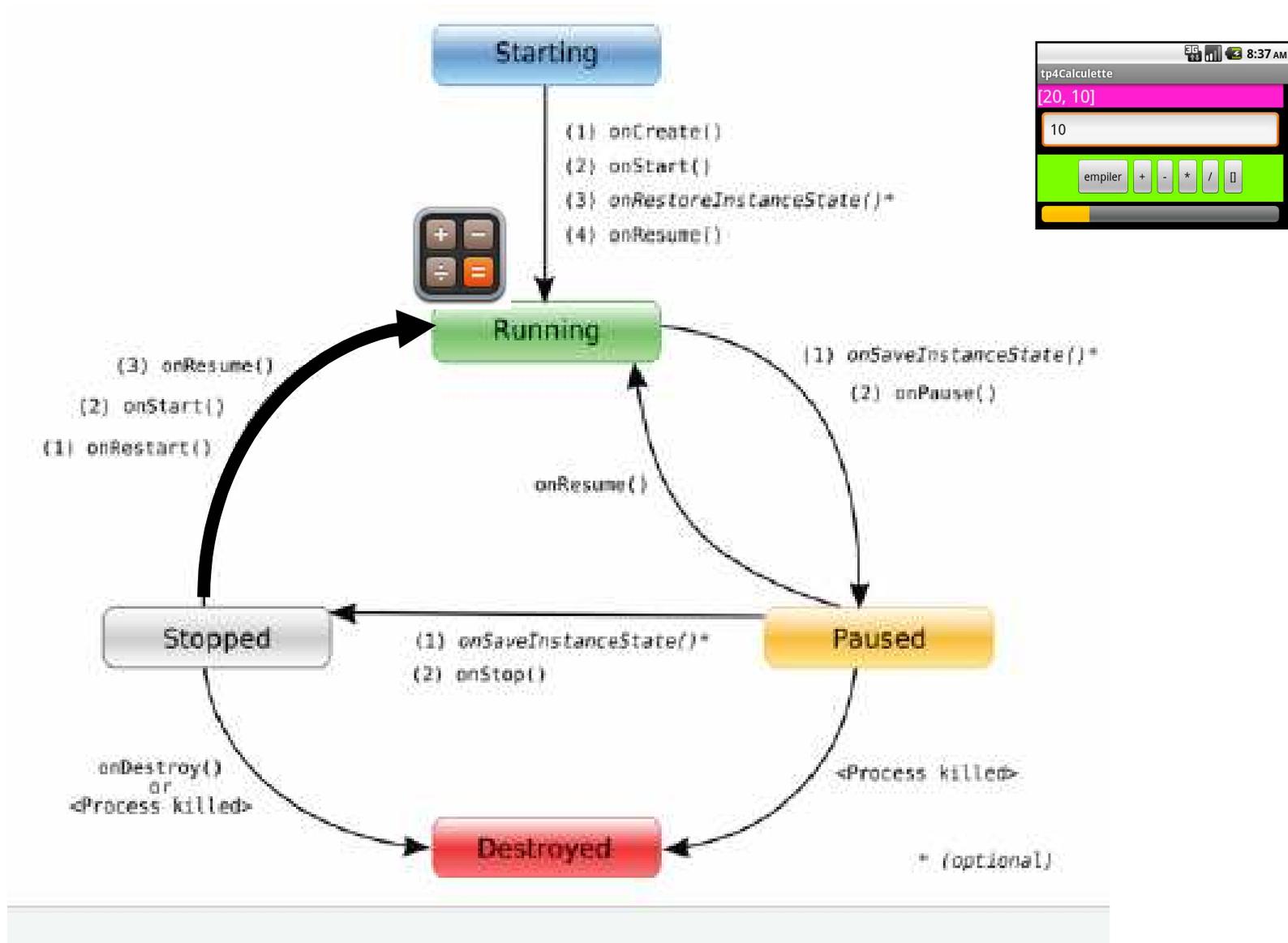
- **Une pile d'activité en interne**
- **Cycle de vie**

# Cycle de vie d'une activity, `startActivity(sendIntent)`:



- <http://inandroid.in/archives/tag/activity-lifecycle>

# Cycle de vie d'une activity, sms envoyé +

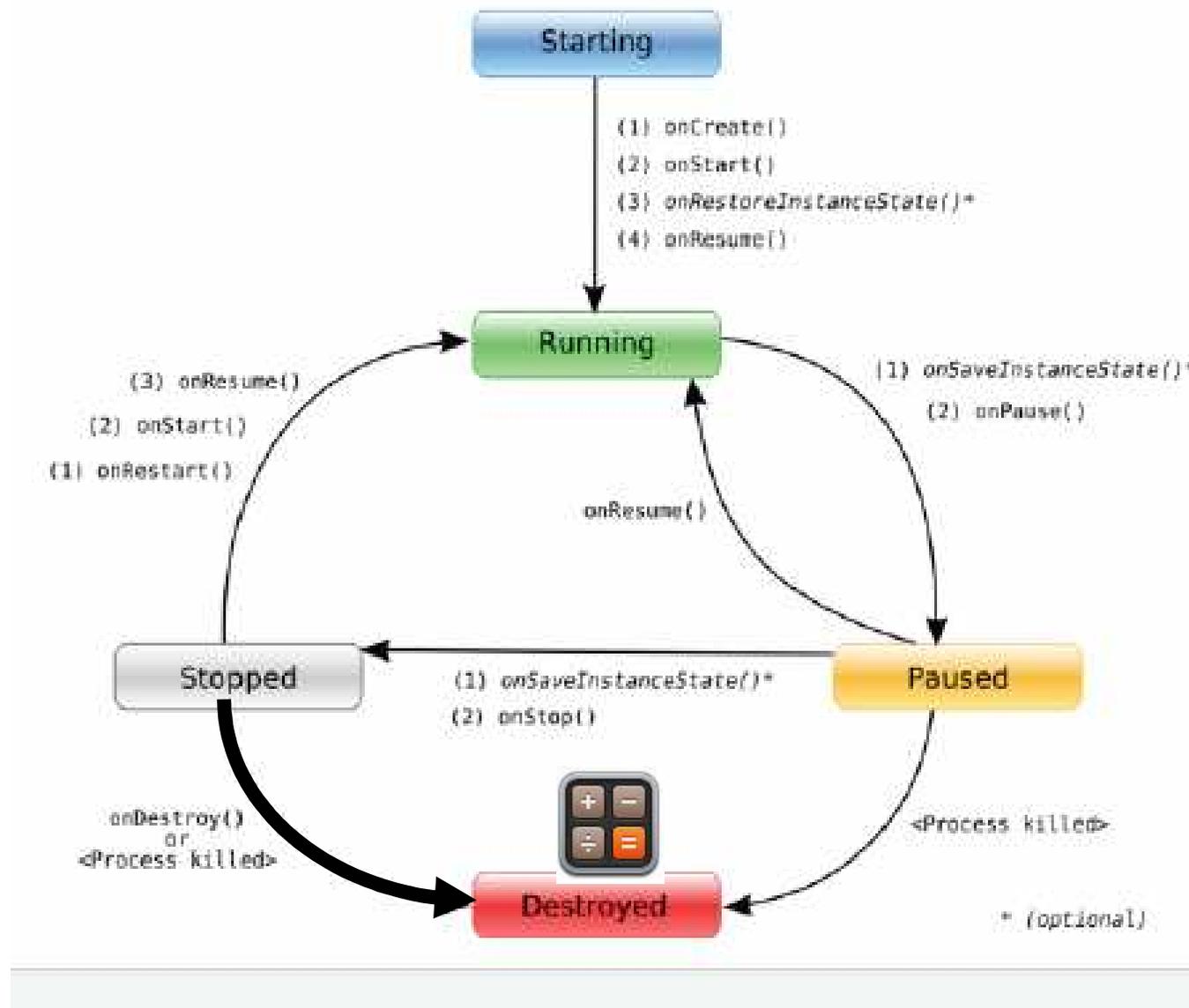


## Quelques remarques

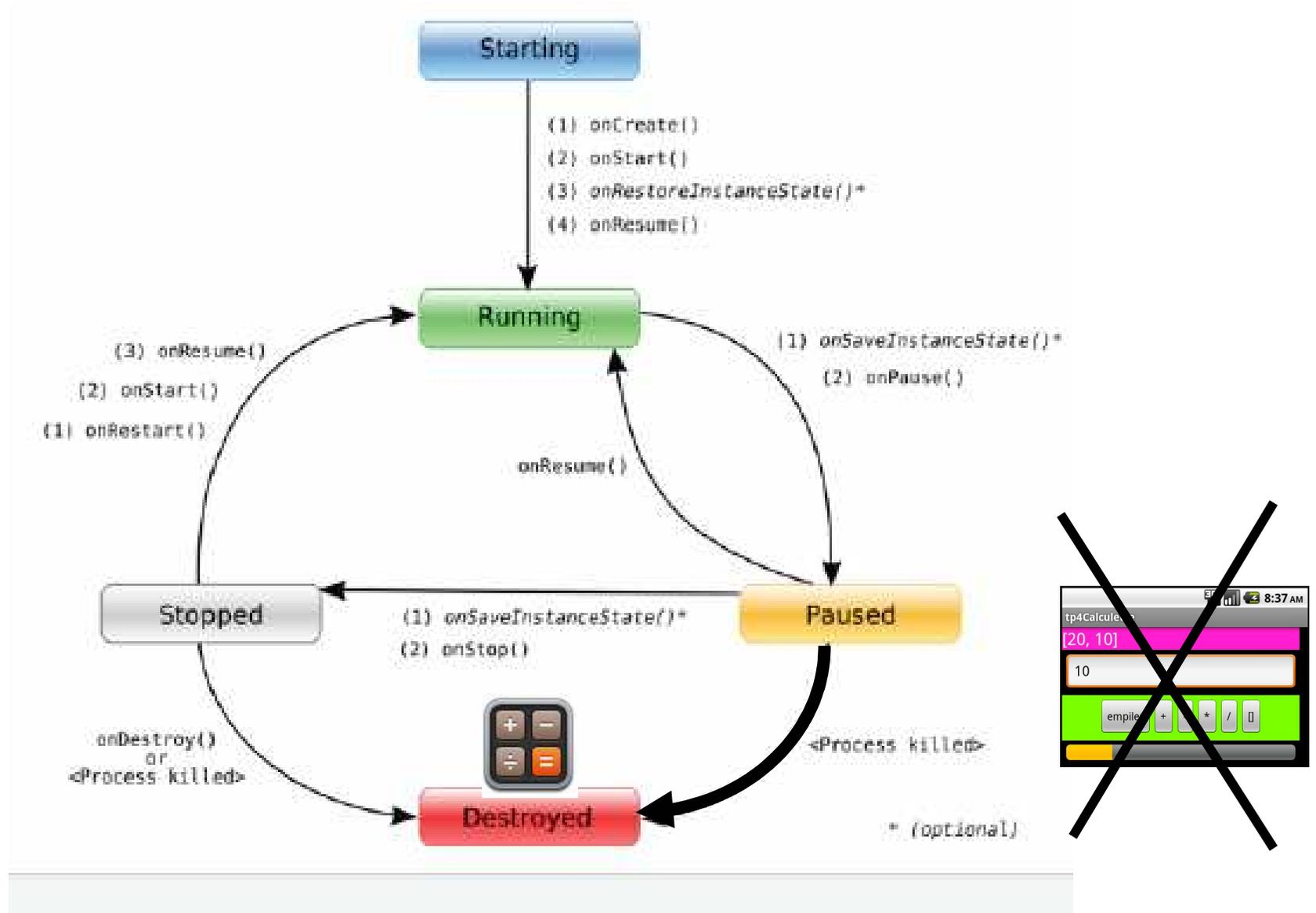
---

- **Attention à la persistance**
- **Lorsque l'activité est en Pause ou Stoppée**
  - **Android peut décider de supprimer l'activité**

# Cycle de vie d'une activity, android killer

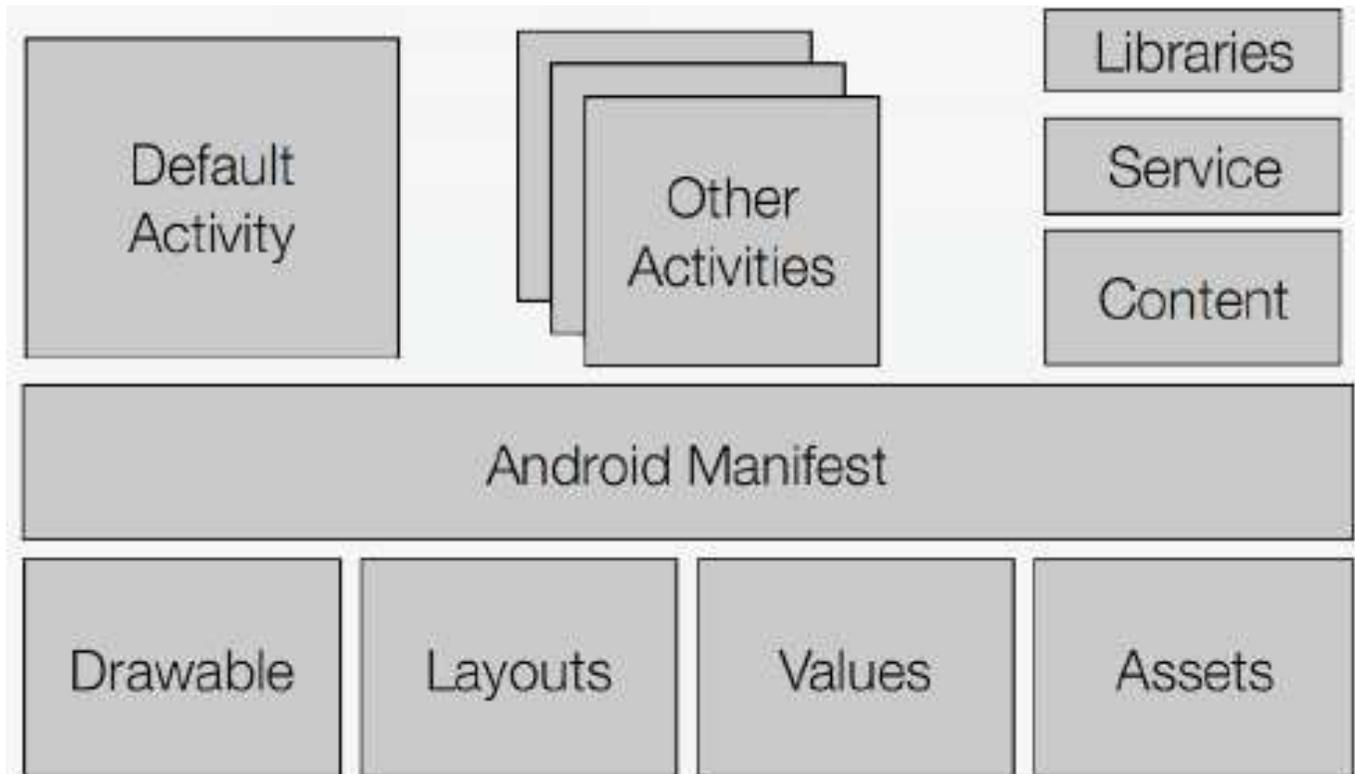


# Cycle de vie d'une activity, android killer

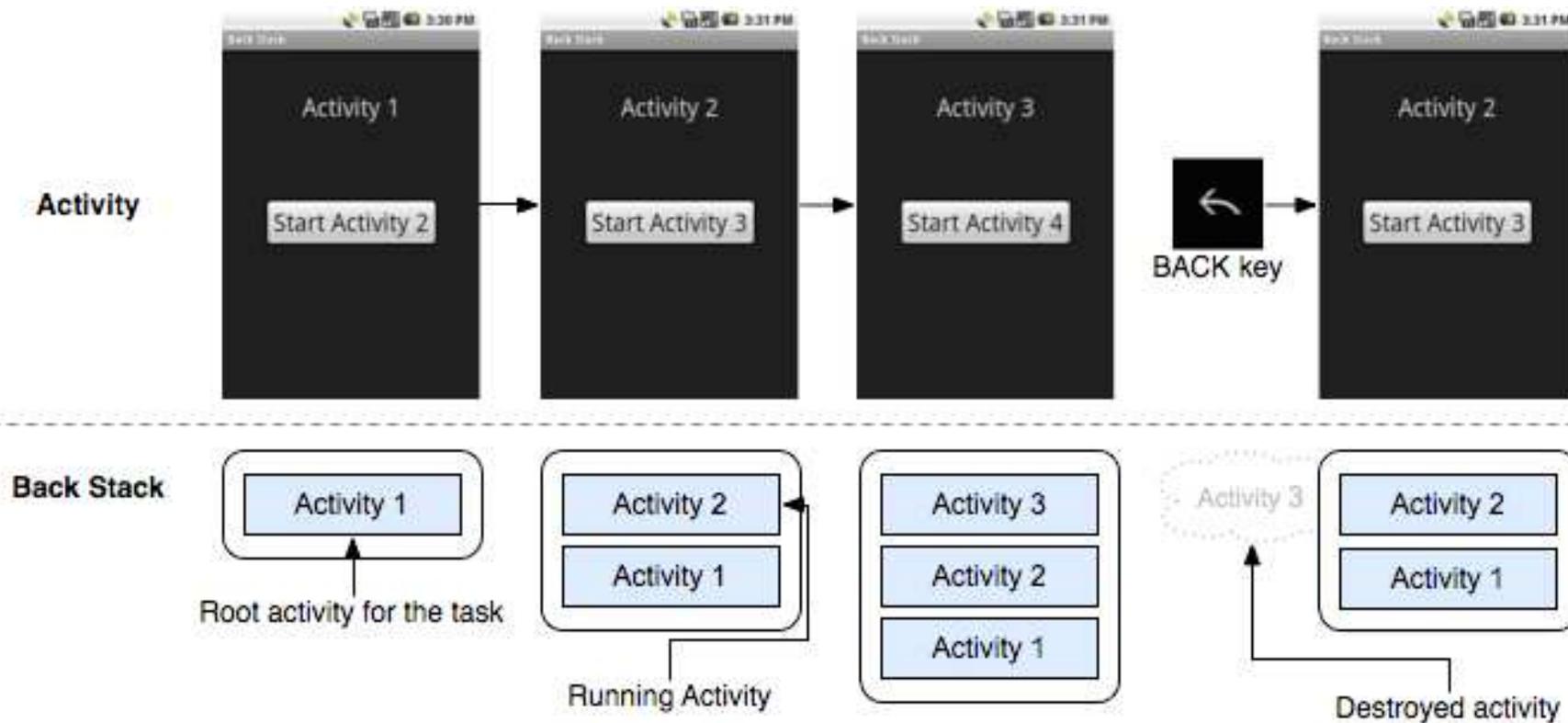


# En résumé, l'architecture se précise

- **Une Activity peut en déclencher une autre**
  - A chaque activity son écran (son fichier XML)
  - Nécessaire gestion de ces activités, android utilise une pile



# La pile des Activity



- <http://www.vineetgupta.com/2011/03/mobile-platforms-part-1-android/>
- <http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html>

## Extension possible : à chaque exception un sms !

---

- **Trop de sms ....**
- **Un mauvais format de Nombre → Contrôle de la saisie**

**android:numeric="decimal"**  
**android:inputType="number"**

# Annexes

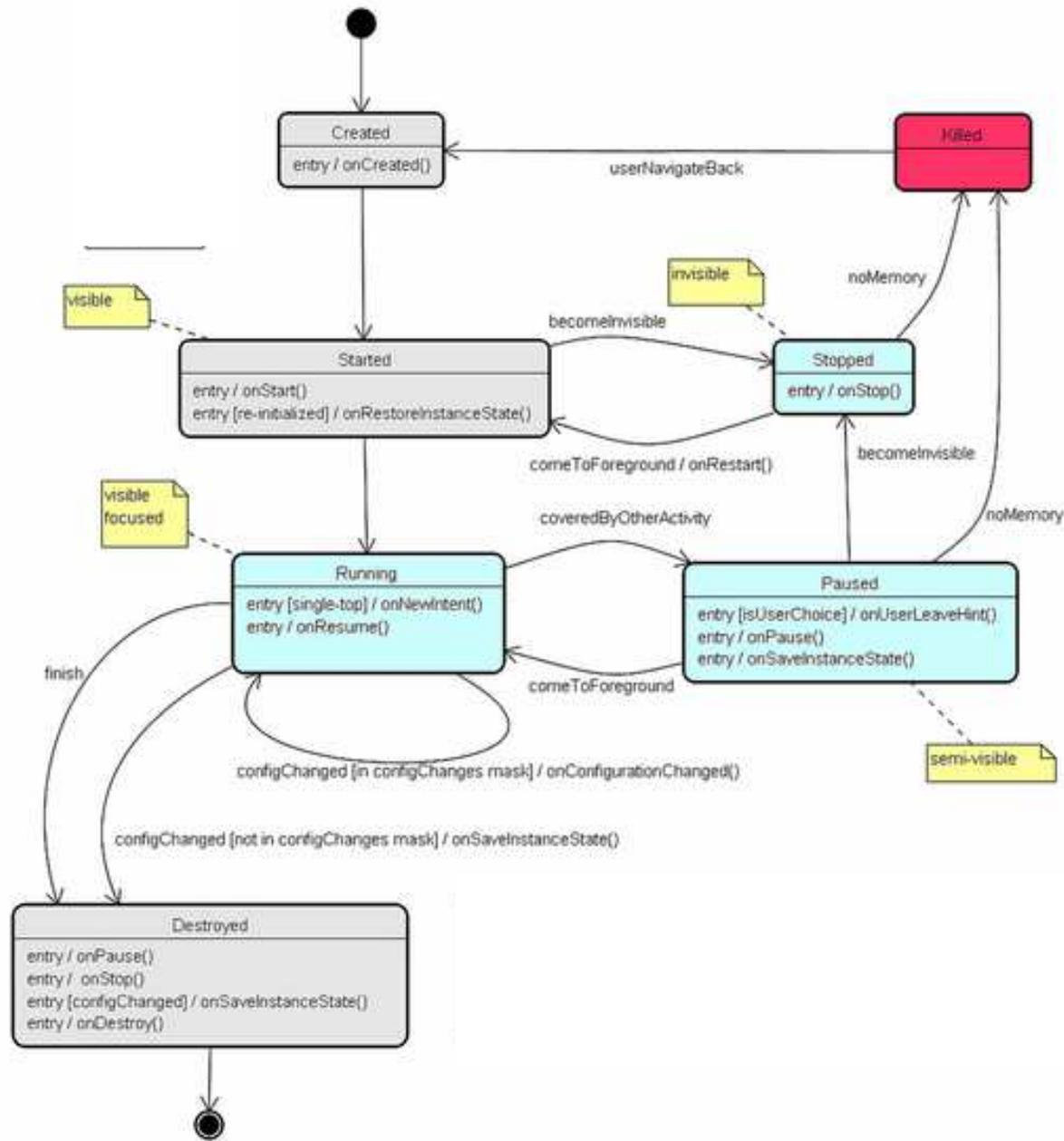
---

- **Un tutorial sur l'approche déclarative**
- **Un statechart pour le diagramme d'états d'une activité**
- **Des exemples de composant graphiques**
  - La liste n'est pas exhaustive

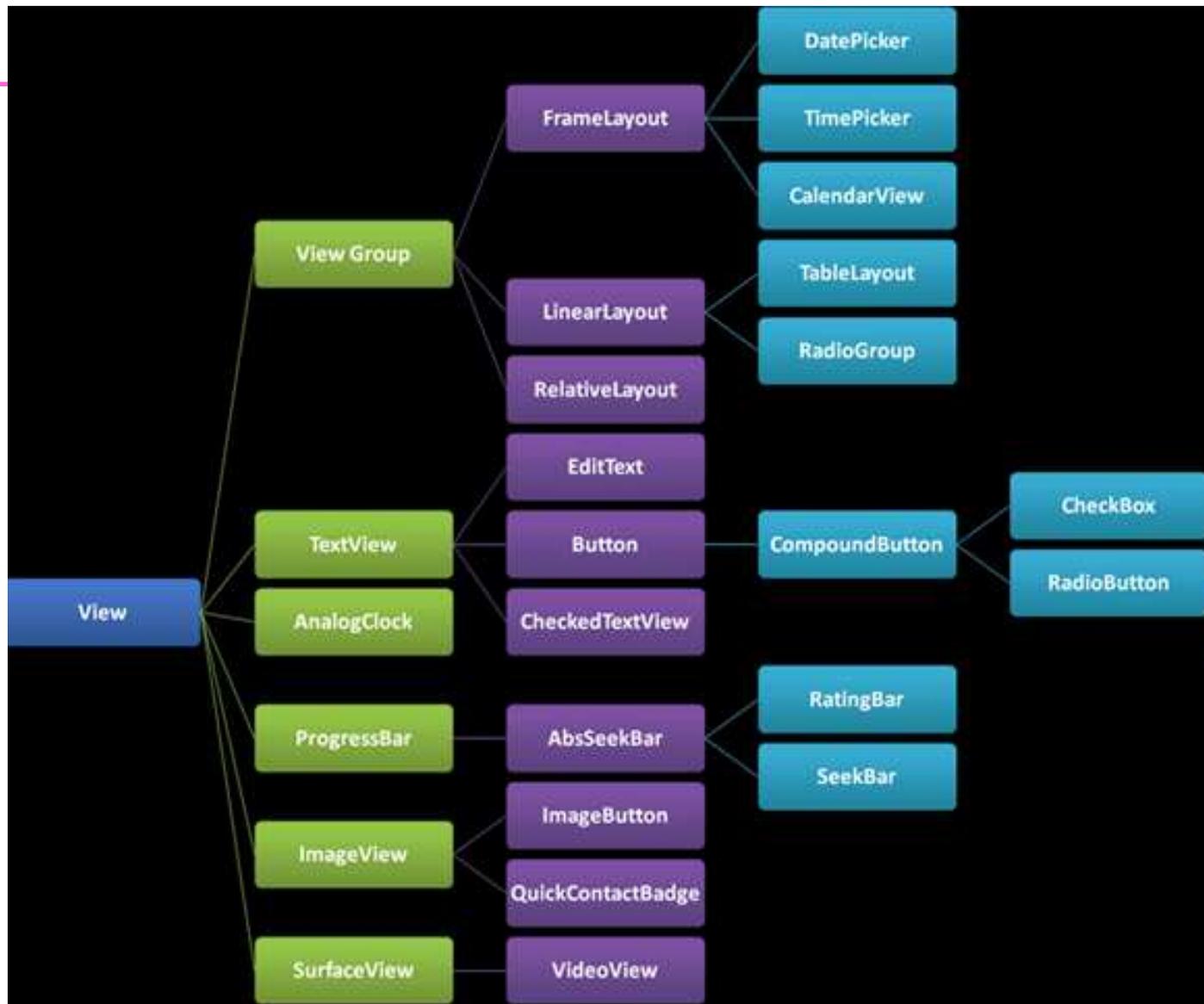
## Annexe : un tutorial

---

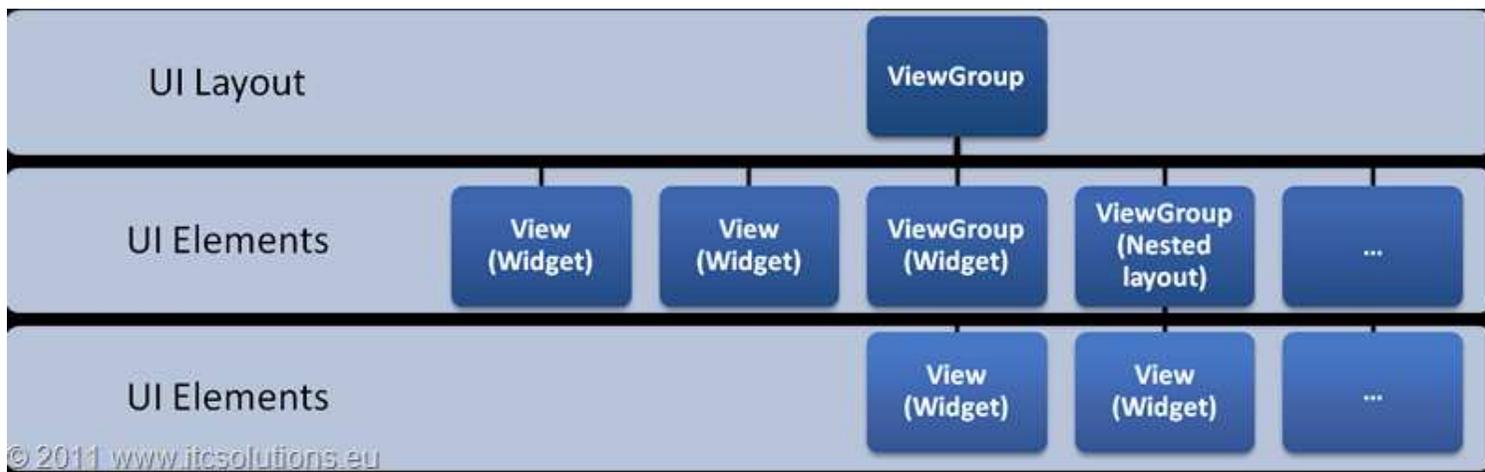
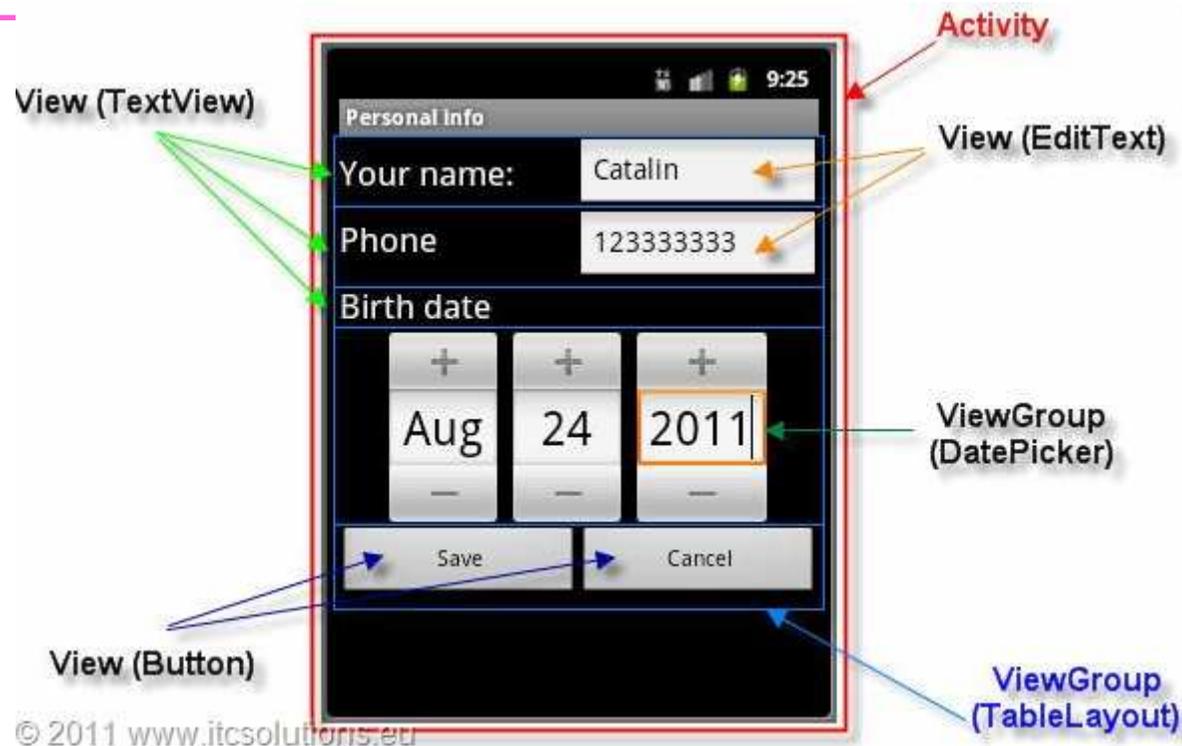
- <http://www.itcsolutions.eu/2011/08/26/android-tutorial-overview-and-contents/>
- <http://www.itcsolutions.eu/2011/08/27/android-tutorial-4-procedural-vs-declarative-design-of-user-interfaces/>



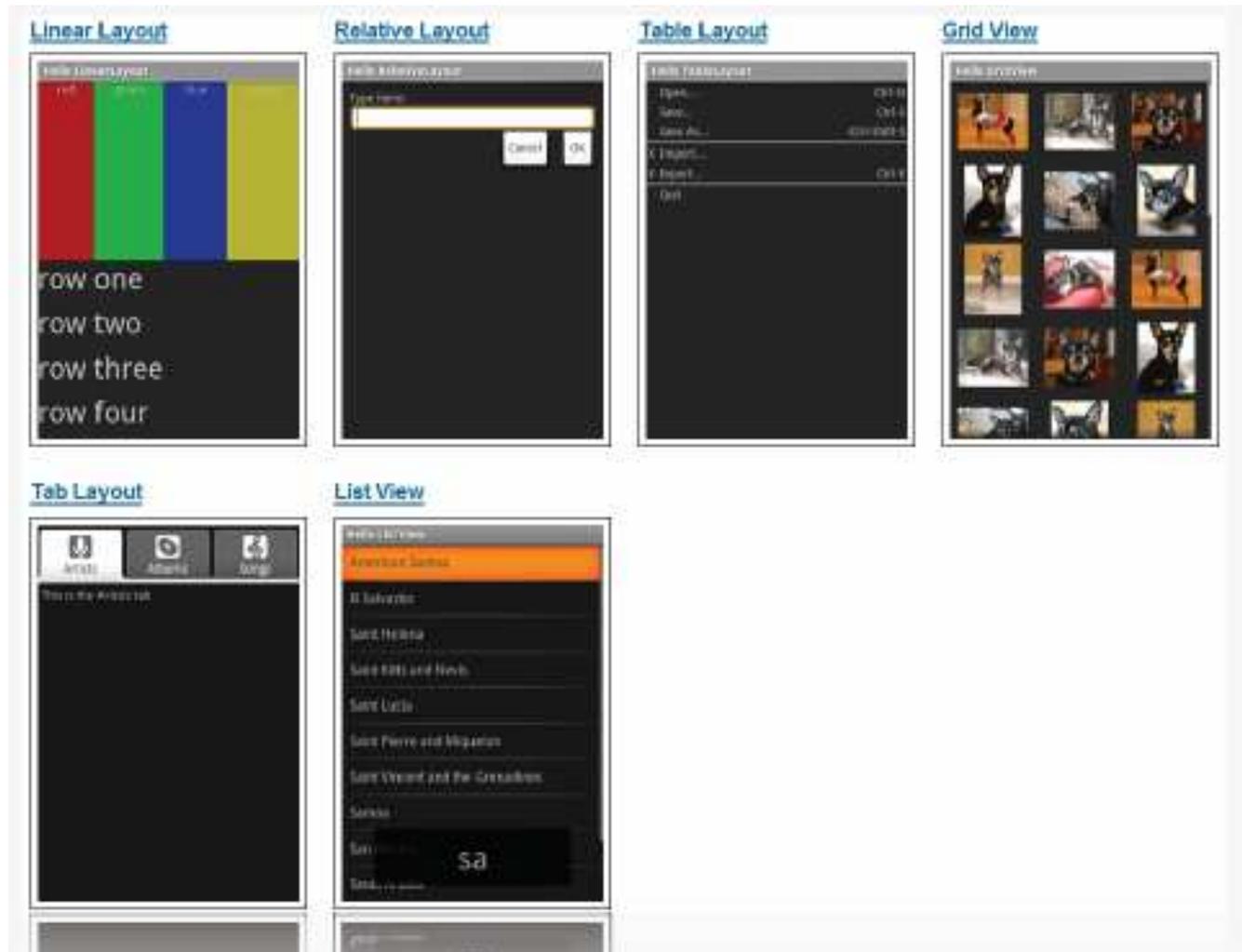
- <http://good-good-study.appspot.com/blog/posts/103001>



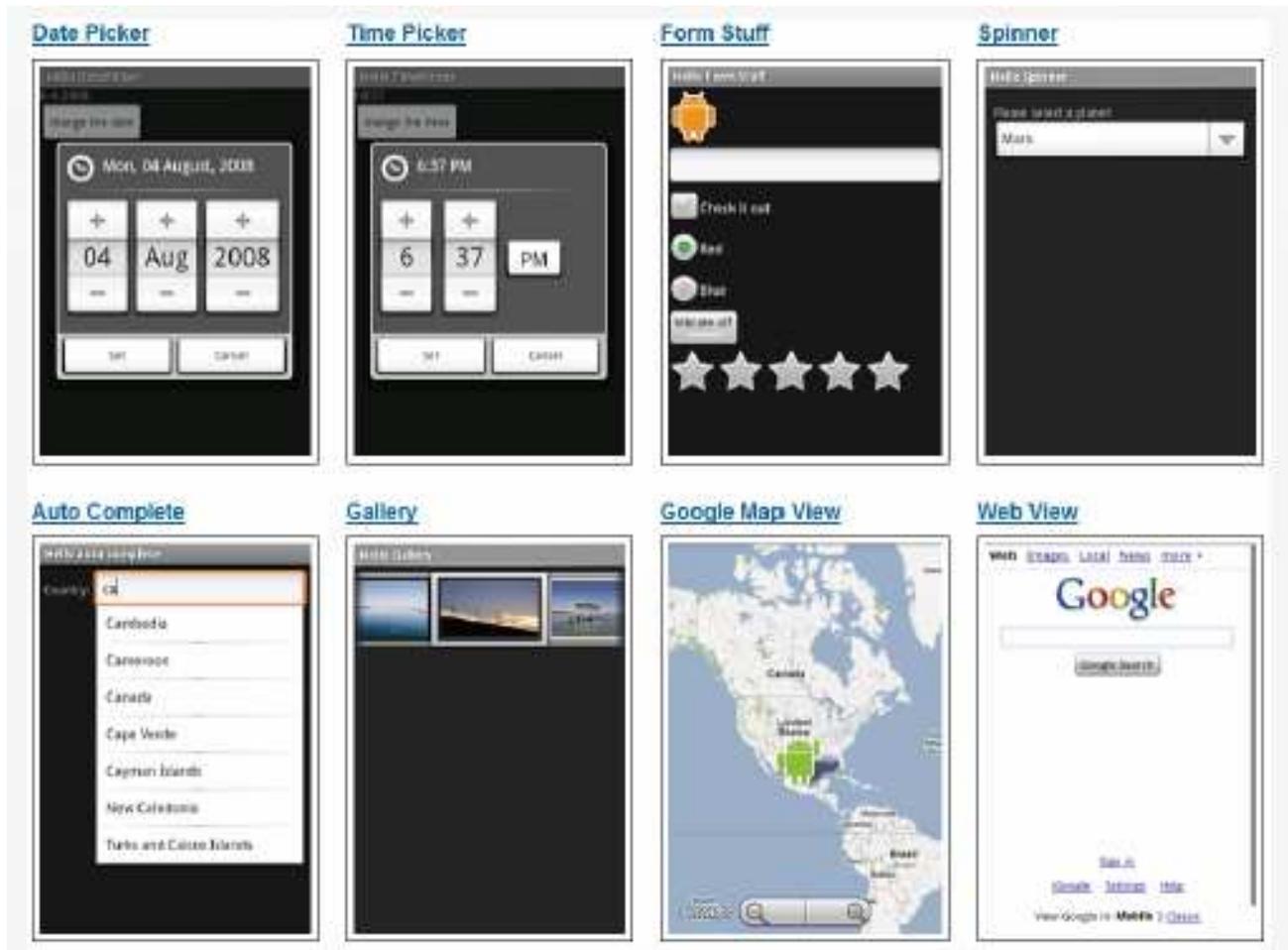
- <http://www.itcsolutions.eu/2011/08/27/android-tutorial-4-procedural-vs-declarative-design-of-user-interfaces/>



# Vocable



# Vocable



[www.Mcours.com](http://www.Mcours.com)  
Site N°1 des Cours et Exercices Email: [contact@mcours.com](mailto:contact@mcours.com)