tp8

Lectures préalables :	Thèmes du TP :
• Tutoriel Intallation PLN • Les notes de cours	• Plateforme Androïd • Le modèle MVC • Le refactoring

---

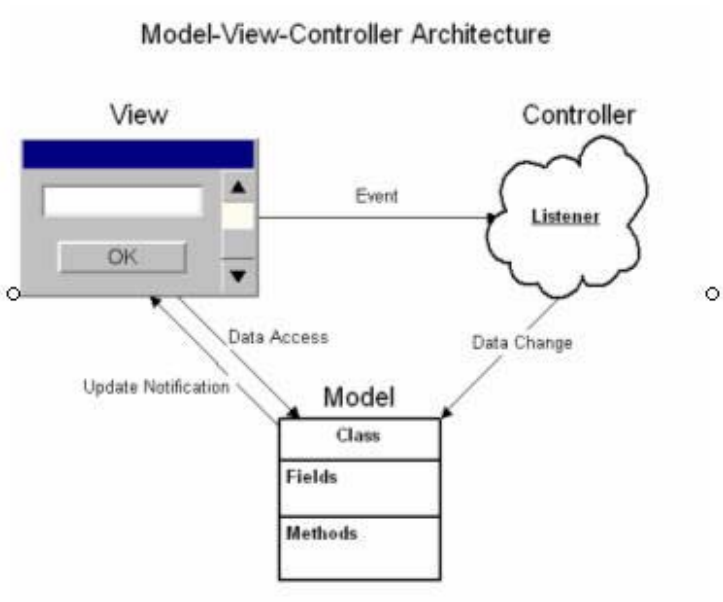
Outils

- JDK 6
- Android SDK
- Eclipse IDE
- ADT plugin pour eclipse

Préambule : Refactoring et déploiement

La question3 du tp_mvc contient plusieurs "erreurs" de conception :

1. Le Modèle est une pile (classe PileModele<T>).
   Le modèle devrait être une calculette qui utilise en interne une pile.
2. Le Contrôleur est un composant graphique qui implémente les actions de l'utilisateur.
   Le contrôleur devrait implémenter les actions de l'utilisateur et uniquement.
3. La Vue ne contient pas toute l'Interface Homme/Machine.
   La vue, ce que l'on voit, devrait contenir toute l'interface.



Une architecture type selon le paradigme MVC

Remaniement (refactoring) :

1. Le Modèle est une calculette.
2. Le Contrôleur implémente toutes les actions de l'utilisateur.
3. La vue contient toute l'interface.

En Java :

1. Le Modèle est observable, ***Calculette extends java.util.Observable***
2. Le Contrôleur ***implements tous les "Listener"***
3. La vue est un observateur, ***implements java.util.Observer***

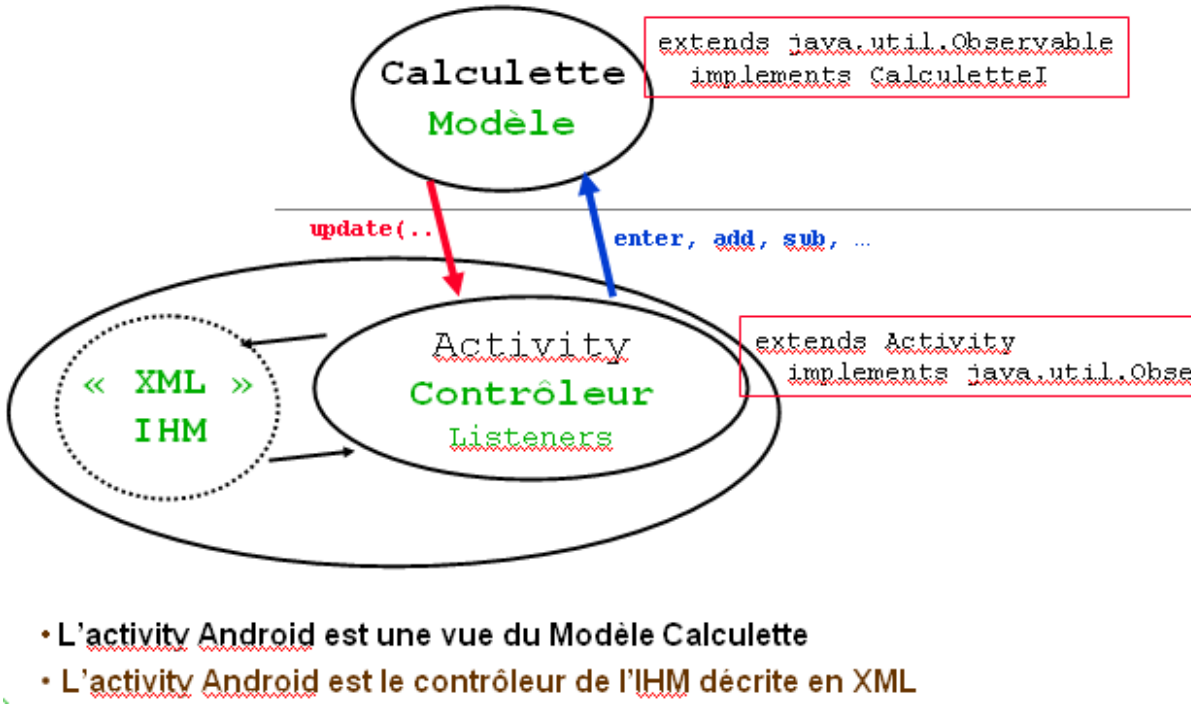
Les sources du modèle Calculette sont ici : http://douin.free.fr/tp4Calculette/

Sur la plate forme Android, à un écran est associé une *"activity"*, assimilable à la **vue** selon le découpage MVC.
Les composants graphiques de l'interface décrits en XML sont accessibles depuis cette activité.
C'est au sein de cette activité que les "listeners" sont associés au comportement des composants graphiques. L'activité devient le **contrôleur** des actions de l'utilisateur.

Sous Android :

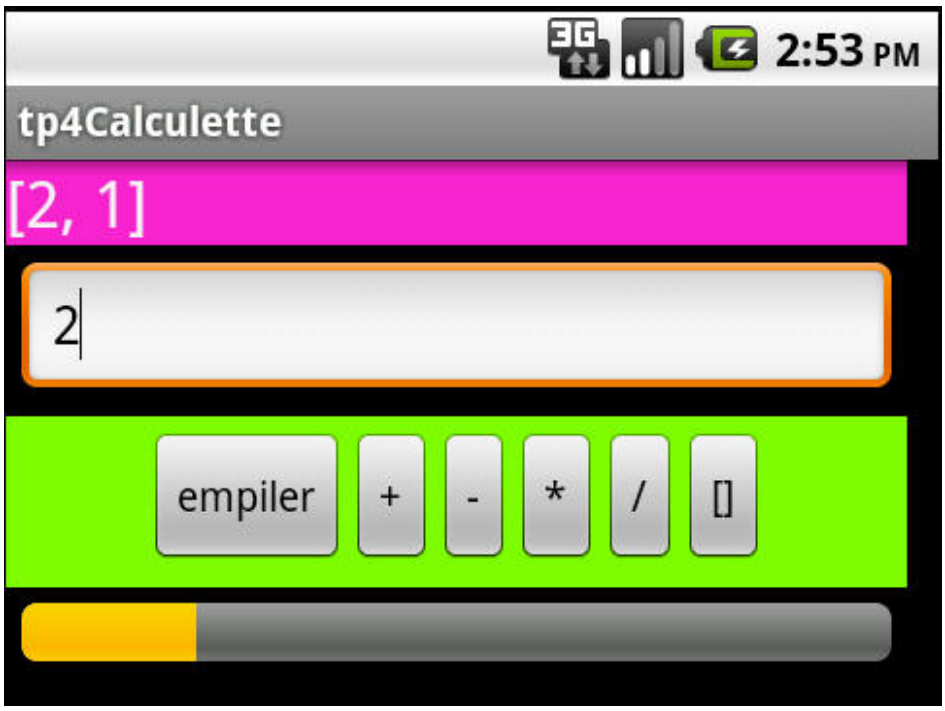
1. Le Modèle est observable => ***extends java.util.Observable,***
2. L'activité est le Contrôleur => ***implements les "Listener"***
3. L'activité est la vue du modèle => ***implements java.util.Observer***

Déploiement sur plate forme Android :

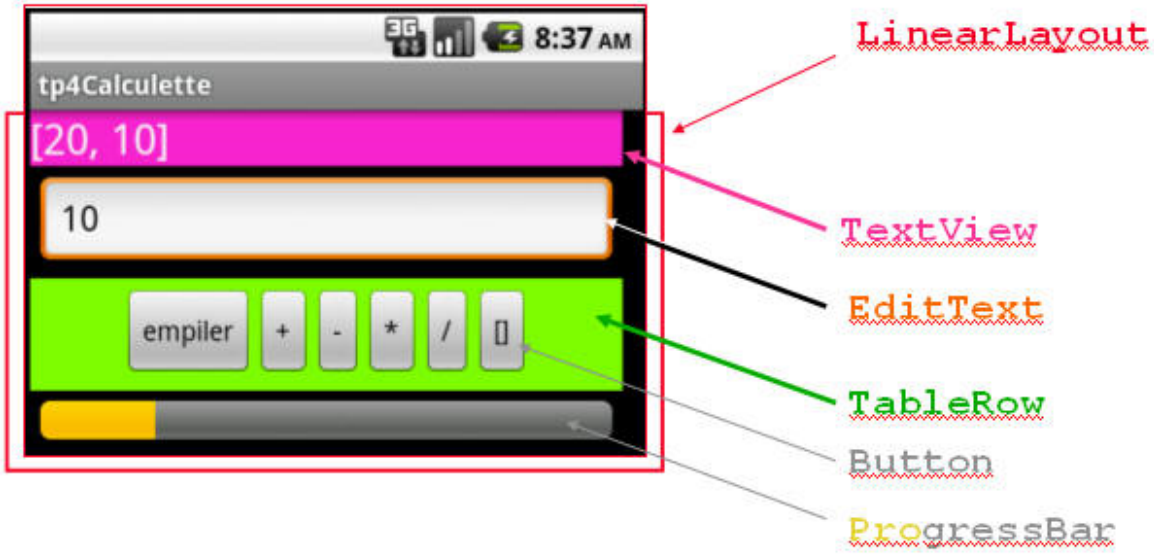


Développez l'activité Calculette pour Android

Une idée d'interface pourrait être :



soit l'usage par exemple de ces composants graphiques,



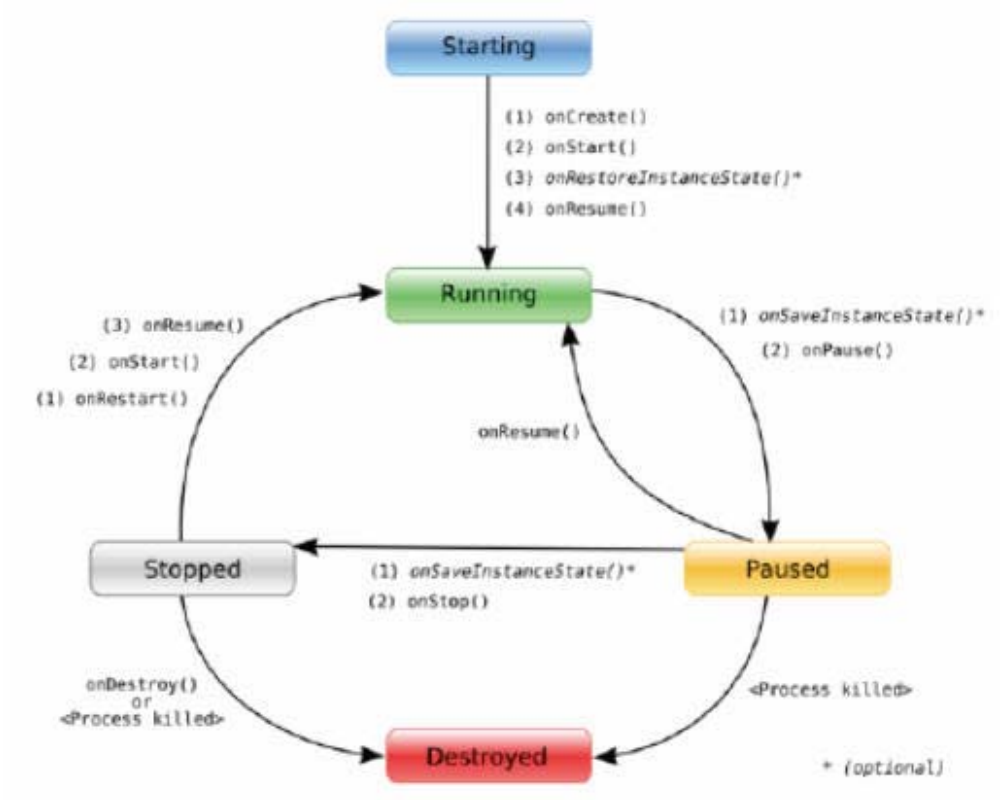
ou tout autre interface ...

Par exemple le *bouton*  **(*Button*) pourrait être remplacé par**  **(*ImageButton*) (www.iconfinder.com pour les images)**

Le code source de la méthode onCreate, redéfinie au sein de l'activité pourrait être :

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

Cette méthode onCreate est appelée au démarrage de l'activité, comme le montre ce diagramme d'états d'une activité sous Android :



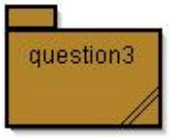
Persistance à l'aide des *Bundle*

La rotation de l'écran entraîne une nouvelle création de l'activité, la méthode *onSaveInstanceState* est appelée avant sa destruction, *onRestoreState* lorsque cette activité est de nouveau créée, au premier plan. Une solution serait d'utiliser le Bundle en paramètre :

http://developer.android.com/reference/android/os/Bundle.html
http://stackoverflow.com/questions/151777/how-do-i-save-an-android-applications-state

Prenez en compte cette rotation de l'écran dans votre activité.

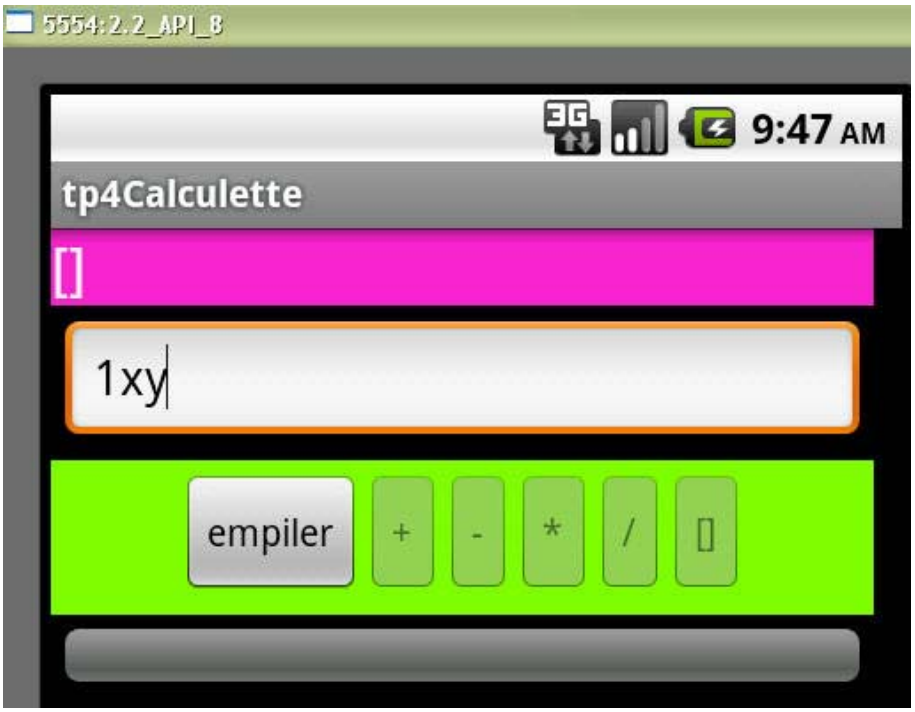
Note : la commande de rotation de l'écran de l'émulateur s'effectue avec Ctrl + F11,
http://developer.android.com/guide/developing/tools/emulator.html

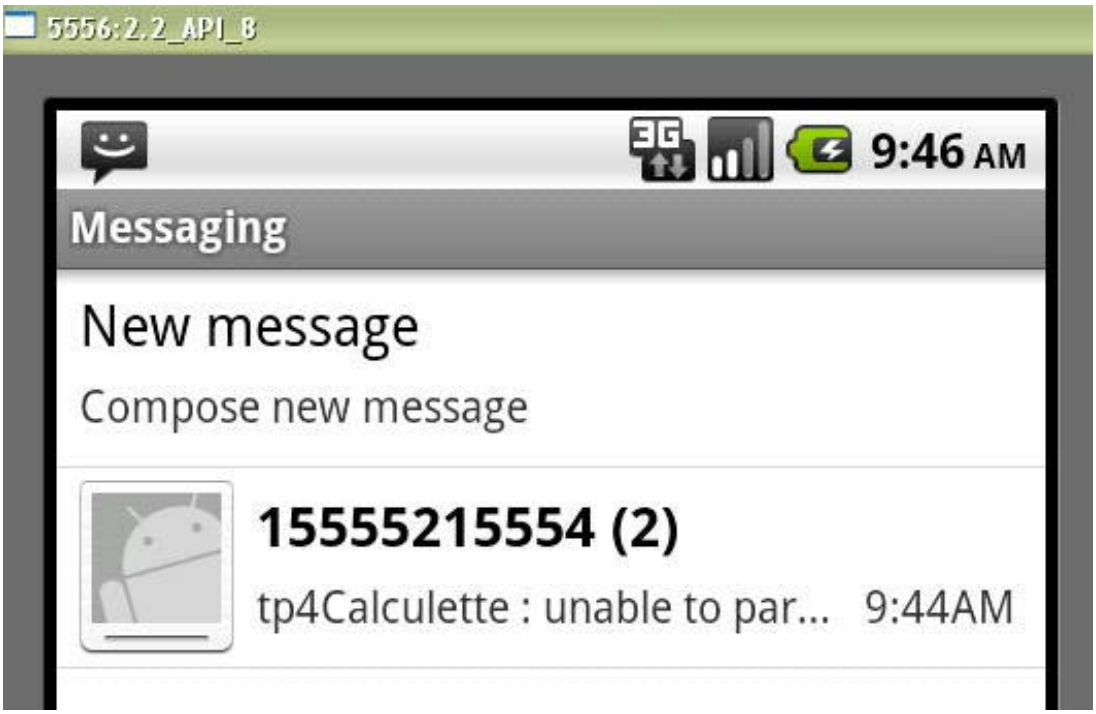


Envoi de SMS

A chaque mauvaise entrée de nombre, envoyez un sms au deuxième émulateur installé sur le même poste, le contenu contient le message de l'exception ...

(selon les deux façons présentées sur les transparents 44, 45 et 46 du support de cours)





Une exception est levée => un sms est envoyé au second émulateur (ici numéro de tél. 5556)

http://developer.android.com/guide/developing/devices/emulator.html