android

An Open Handset Alliance Project

Le patron de conception : observer

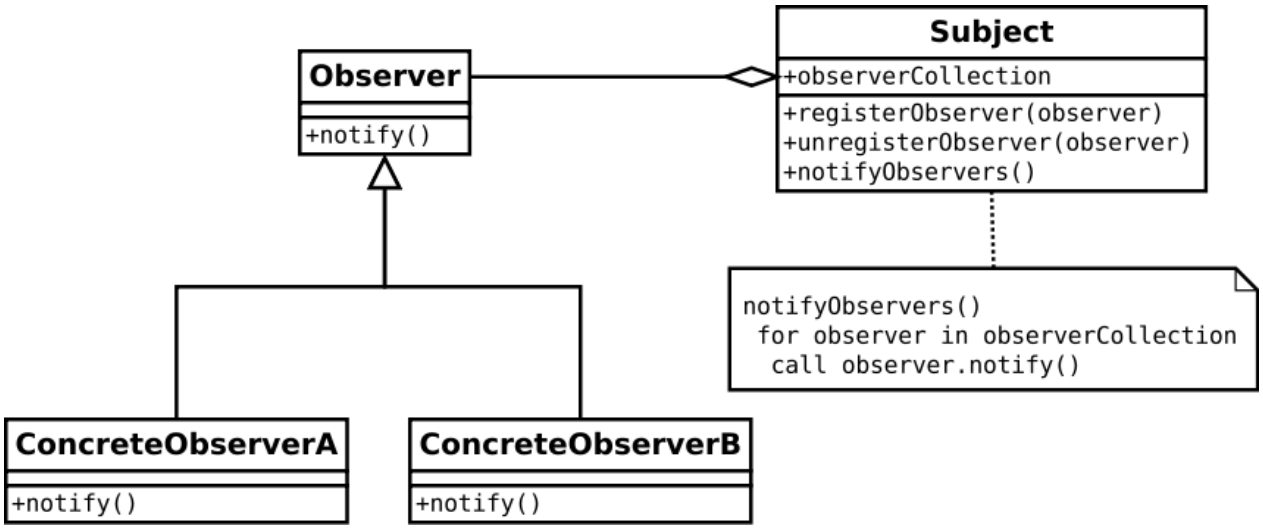
G.Seront

Interfaces Homme-Machine

- C’est souvent difficile de développer une bonne IHM.
- En particulier, il est trop simple de s’embourber dans du code qui part dans tous les sens.
- On mitige ce problème en séparant les événements de leur(s) traitement(s) : patron de conception Observer.
  - On enregistre sur l’événement le ou les observeurs intéressés.
  - Lorsque l’événement survient, tous les observeurs sont interpelés.

https://fr.wikipedia.org/wiki/Observateur_(patron_de_conception)

Observer



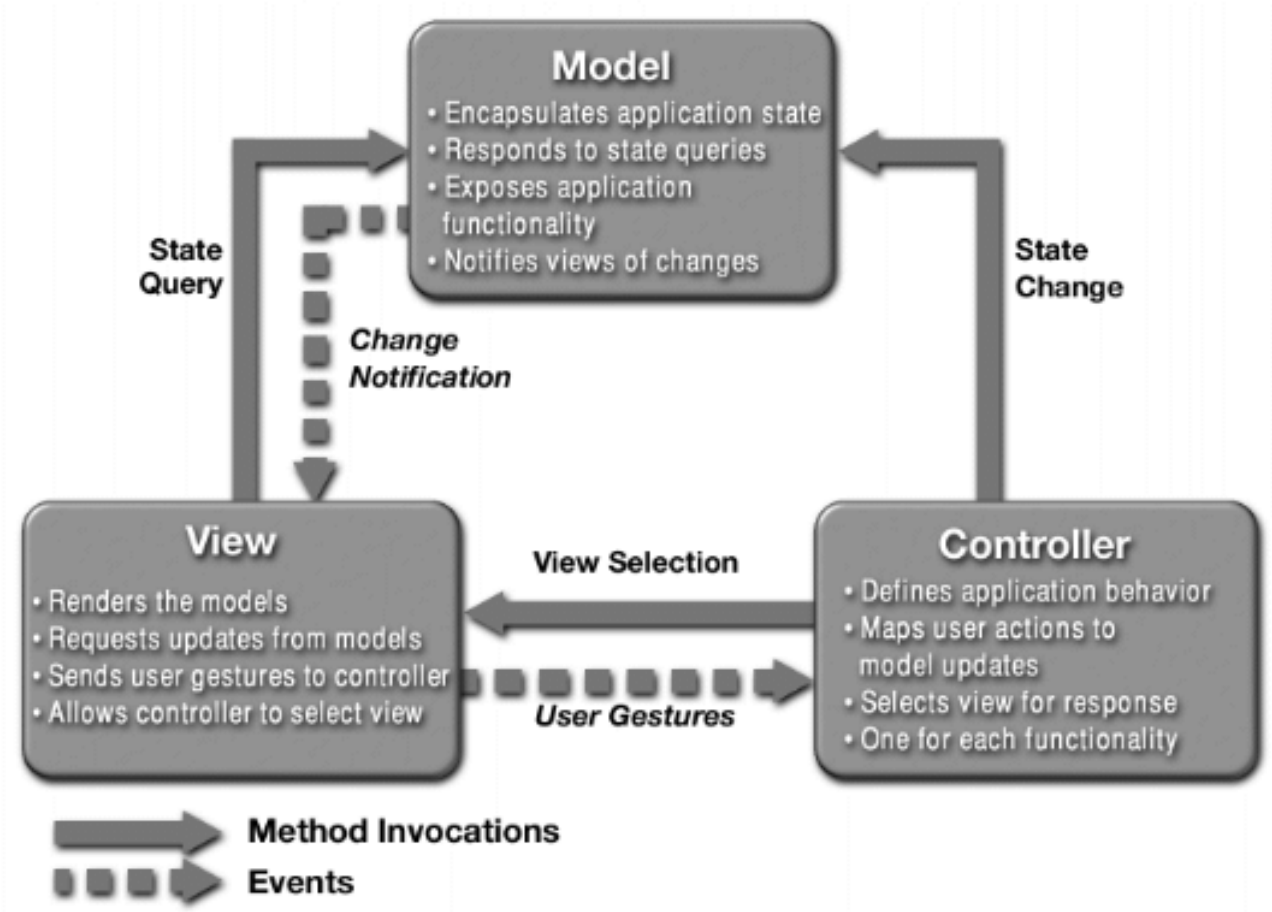
- Le sujet retient le(s) observeur(s) intéressés.
- Lorsque l'événement survient (p.ex. un click sur un bouton), les observeurs sont notifés.

Model View Controller

- Structuration du code en 3 composants :
  1. Le ou les modèles.
     - Responsable des données, de l'état de l'application.
     - Implémente un sujet du patron observeur. Notifie tous les observeurs lorsque les données changent.
  2. La ou les vues.
     - Responsable d'afficher tout ou partie des données à l'utilisateur.
     - Implémenté en tant qu'observeur enregistré au modèle.
  3. Le ou les controlleurs.
     - Responsable de capturer les entrées de l'utilisateur, afin de modifier le modèle.

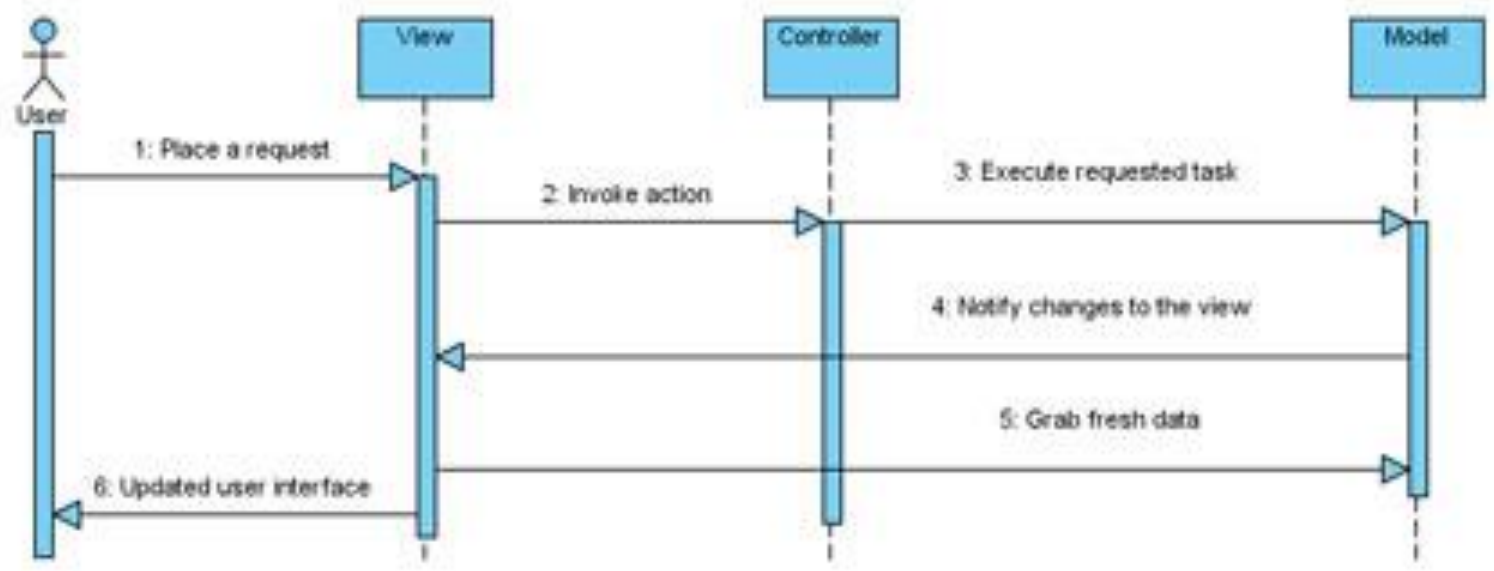
https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leurl

Model View Controller



http://www.bogotobogo.com/DesignPatterns/mvc_model_view_controller_pattern.php

Model View Controller



http://www.bogotobogo.com/DesignPatterns/mvc_model_view_controller_pattern.php

Code business en MVC

- Mais où va le code métier de l'application ?
  - P.ex. la mise à jour de la DB, le calcul de statistiques, etc…
- Le code métier ne va pas dans la vue, sa responsabilité est limitée à de l'affichage.
- Le code métier peut aller dans le contrôleur ou dans le modèle, les deux sont viables.

MVC en PHP

- En PHP, le code métier va dans le contrôleur :
  - La vue est la page à afficher
  - La modèle contient les données nécessaires à la vue pour s'afficher correctement
  - Le contrôleur est appelé par une requête HTTP :
    - Il effectue le traitement (métier) de cette requête
    - Il calcule le modèle nécessaire à la vue
    - Il renvoie la vue générée au front-end

https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur

MVC en Android

- En Android la découpe sera différente :
  - La vue = l'activité qui s'affiche.
  - Le modèle = les données et l'état du système + les méthodes qui manipulent tout cela.
    C'est dans ces méthodes que le traitement métier sera réalisé.
  - Le contrôleur = ce qui appelle les méthodes du modèle pour manipuler les données = des listeners configurés dans l'activité.

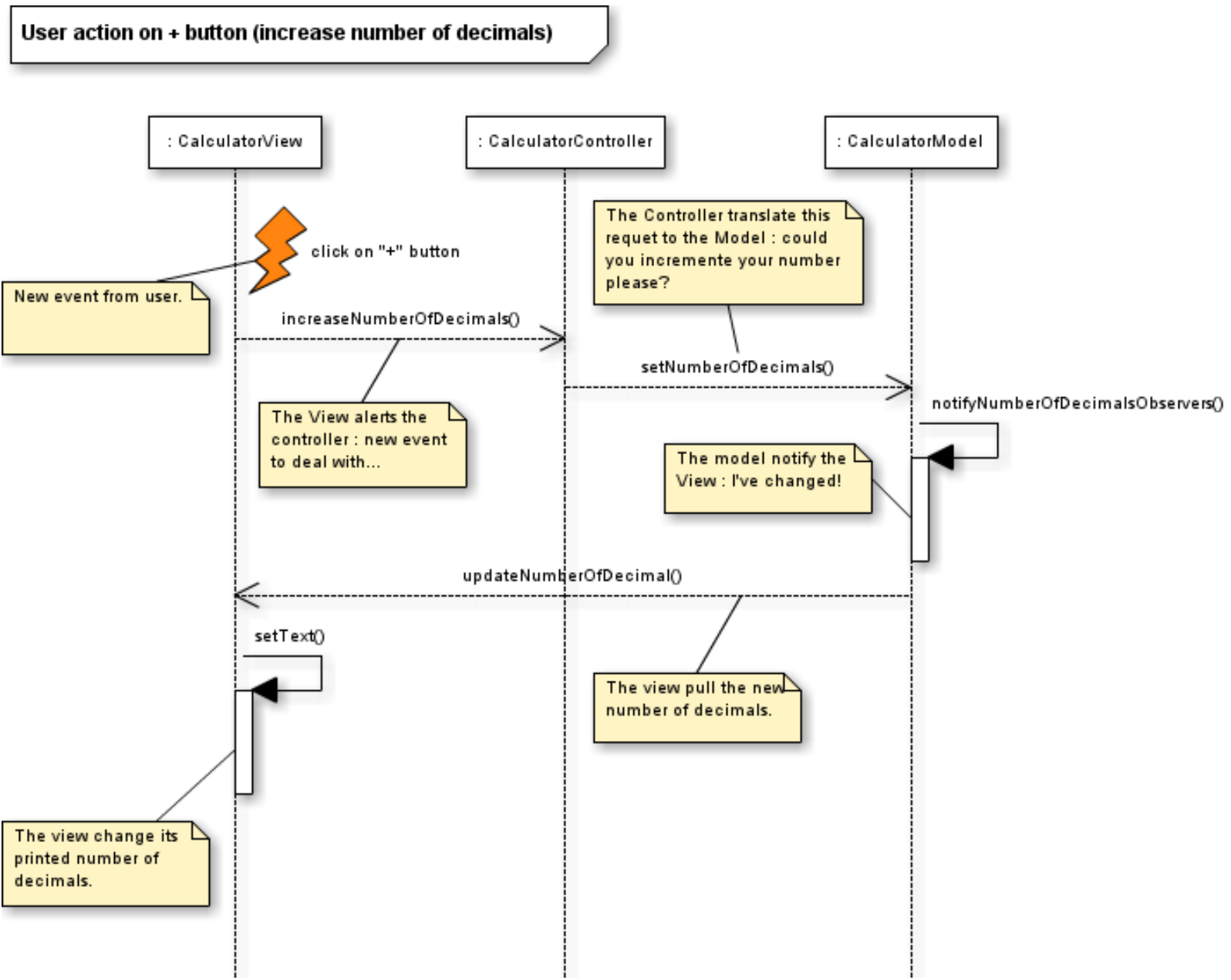
=> Les activités serviront de vue et de contrôleur sur le modèle.
Le code métier sera directement embarqué sur le modèle.

https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur

Exemple: Calculatrice

- On a une calculatrice avec un bouton pour augmenter le nombre de décimales affichées.
- Que se passe-t-il quand l'utilisateur appuie sur le bouton?

http://code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc/

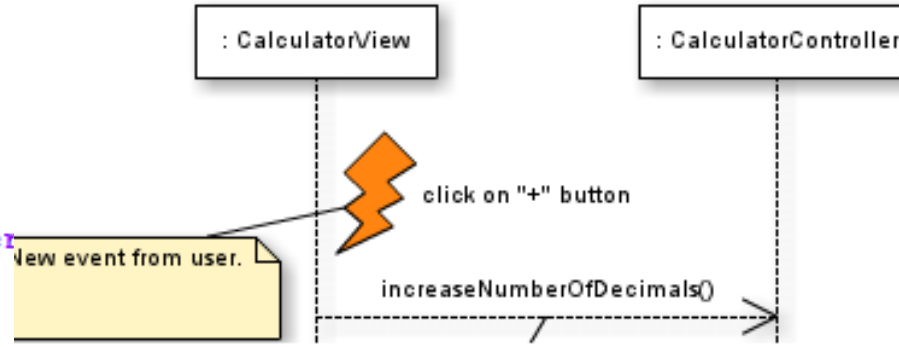


http://code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc/

View => Controller

- Dans le layout XML

```
<Button android:id="@+id/increaseDecimalNumber
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="+"
        android:onClick="onClickIncreaseDecimalNumber"
        />
```

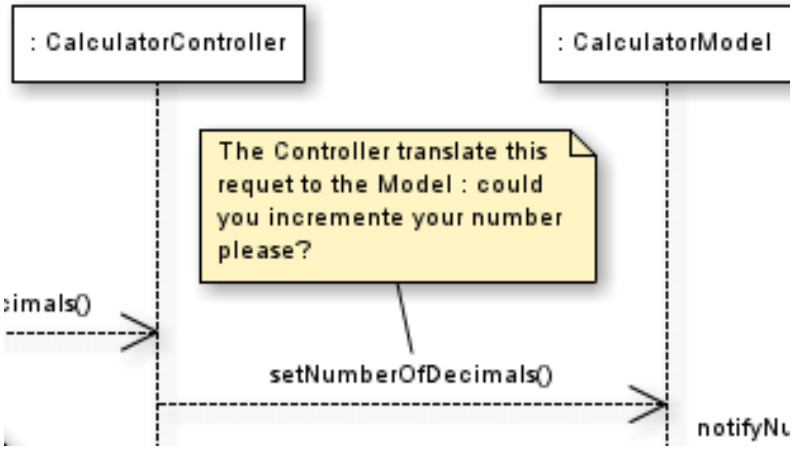


- Dans CalcView.java (l'activité)

```
public void onClickIncreaseDecimalNumber(View view) {
        mController.increaseNumberOfDecimals();
}
```

http://code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc/

Controller => Model

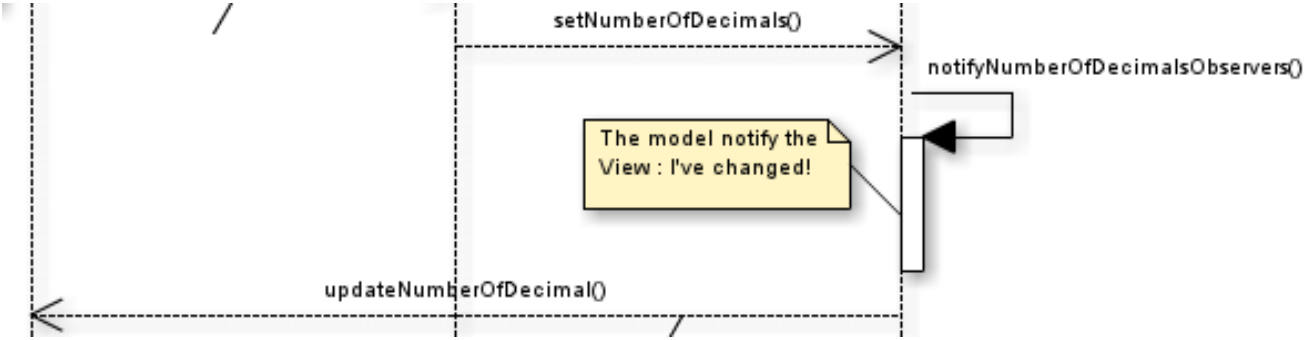


- Dans CalcController.java

```
public void increaseNumberOfDecimals() {
    int newNumber = mModel.getNumberOfDecimals() + 1;
    mModel.setNumberOfDecimals( newNumber );
}
```

http://code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc/

Model => View

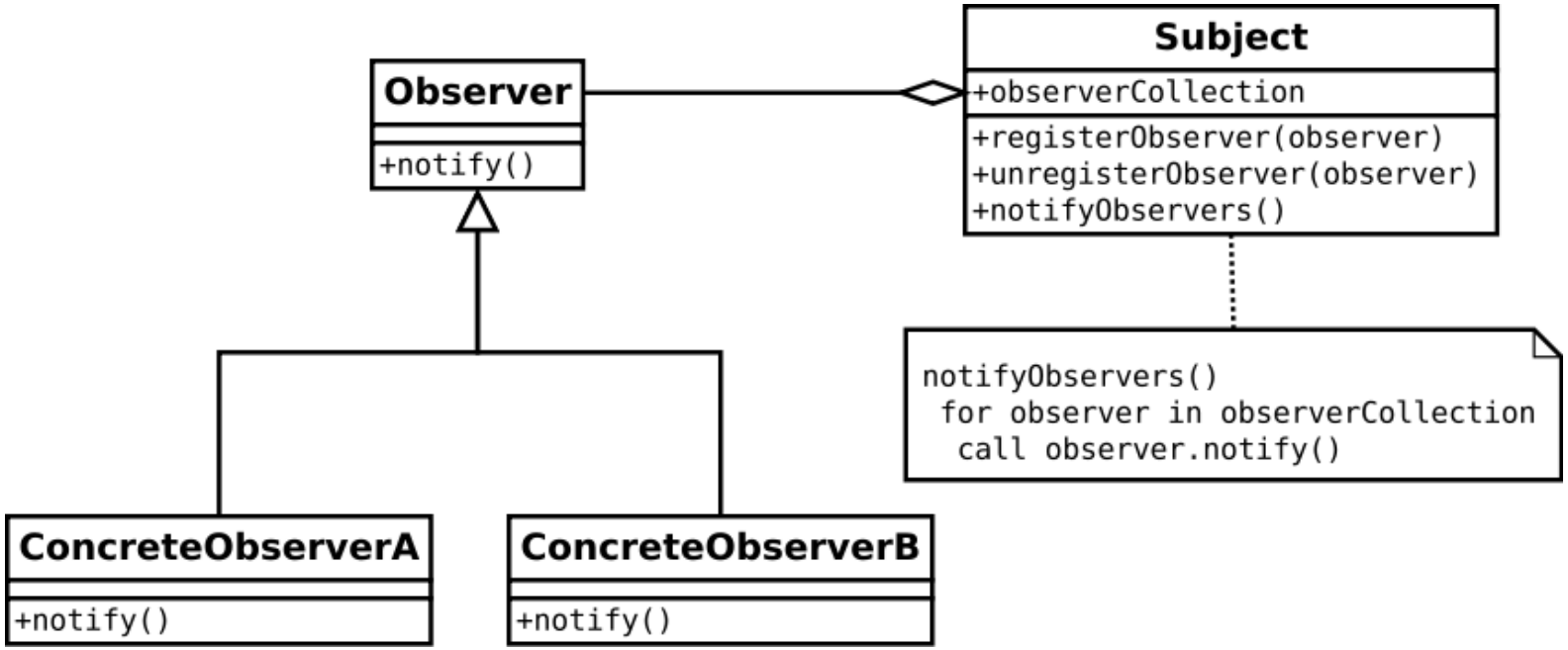


- Dans CalcModel.java

```
public void setNumberOfDecimals( int numberOfDecimals ) {
    this.mNumberOfDecimals = numberOfDecimals;
    notifyNumberOfDecimalsObservers();
}
public void notifyNumberOfDecimalsObservers() {
    for ( int i = 0; i < mNumberOfDecimalsObservers.size(); i++ ) {
        NumberOfDecimalObserver observer =
            (NumberOfDecimalObserver ) mNumberOfDecimalsObservers.get( i );
        observer.updateNumberOfDecimal();
    }
}
```

http://code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc/

Observer Pattern



https://en.wikipedia.org/wiki/Observer_pattern

Enregistrement observateur

- Dans le *onCreate()* de la vue

```
// We register observers to be notified by the model when it has changes
mModel.registerObserver( (NumberOfDecimalObserver) this );
mModel.registerObserver( (ExpressionObserver) this );
```

- Dans le modèle

```
ArrayList<NumberOfDecimalObserver> mNumberOfDecimalsObservers;
public void registerObserver( NumberOfDecimalObserver observer ) {
    mNumberOfDecimalsObservers.add( observer );
}
```

- Interface implémentée par la vue

```
public interface NumberOfDecimalObserver {
    void updateNumberOfDecimal();
}
```

https:// code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc

Mise à jour Vue

- Interface implémentée par la vue

```
public interface NumberOfDecimalObserver {
    void updateNumberOfDecimal();
}
```

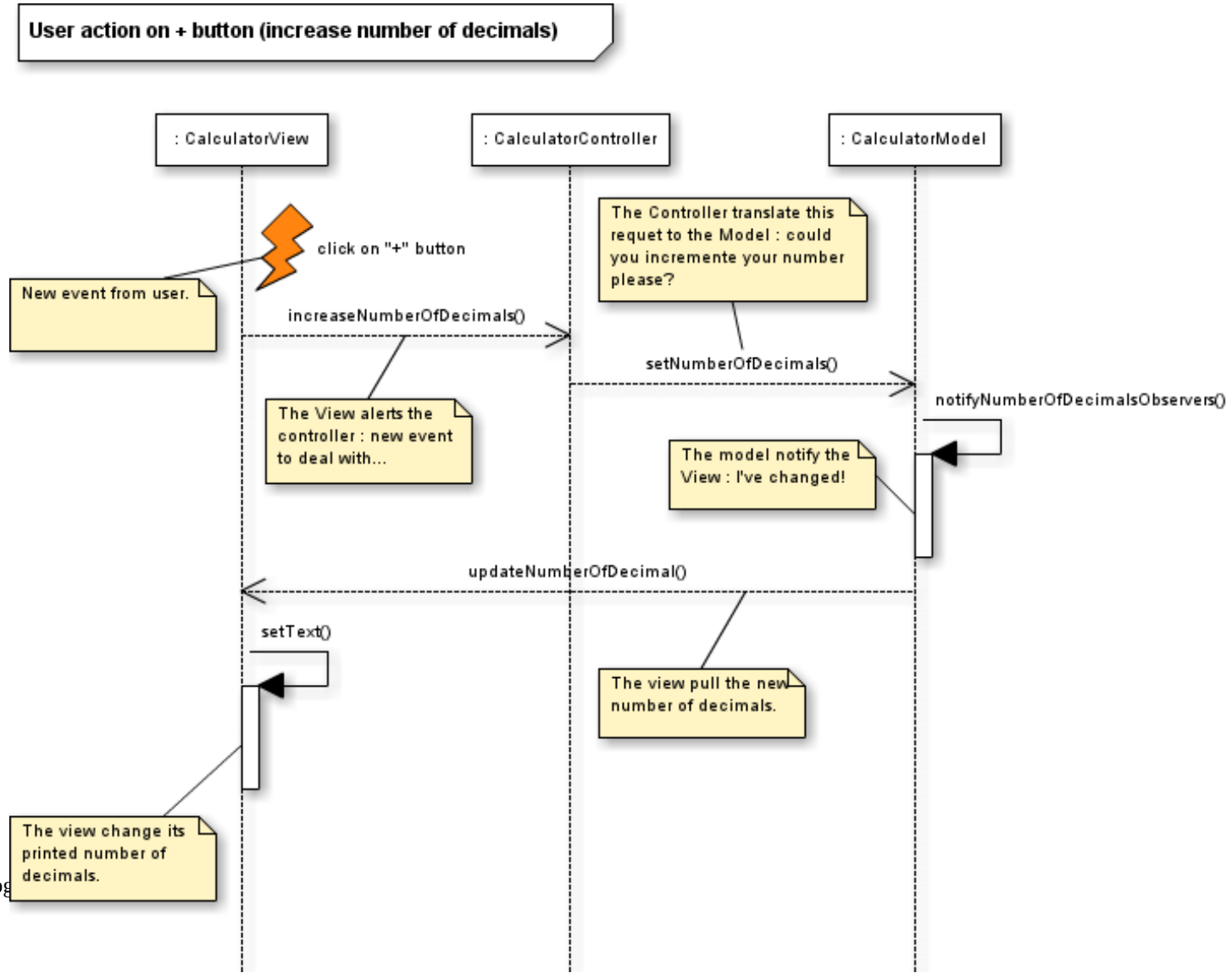
- Dans CalcView.java

```
public void updateNumberOfDecimal() {
    int numberOfDecimals = mModel.getNumberOfDecimals();
    mNumberOfDecimalsWidget.setText( new String( String.valueOf( numberOfDecimals ) ) );

    // We actualize the decrease widget if the model change its number of decimal
    if ( numberOfDecimals <= 0 )
        disableDecreaseButton();
    else
        enableDecreaseButton();
}
```

https:// code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc

C’est clair?



http://code-weblog

MVC: la poule et l'oeuf

- Qui crée qui?
- Normalement on crée d'abord le modèle
- En Android la première chose créé est l'activité.
- On pourrait créer le contrôleur et modèle dans l'activté
- Mais quid si même modèle pour plus d'une vue?
- Classe application à la rescousse!
- Méthode plus avancée: Dependency Injection
  - Voir cours de Laurent Leleux

http://code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc/

Classe Application

- Un seul objet application par application
- Créé avant tout le monde

```
public class Builder extends Application{
    JoueurDetailsModel mModel;
    @Override
    public void onCreate() {
        super.onCreate();
        mModel = new JoueurDetailsModel();
    }

    public JoueurDetailsModel getModel();
```

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme"
    android:name=".Builder"
    >
```

- + création dans le onCreate de l'activity

http://code-weblog.com/developpement-dune-appli-android-basee-sur-un-mvc/