

TD4 : MVC, modèle, vue, contrôleur

Ce TD est une introduction et une première approche du design pattern MVC.

Ce TD sera également l'occasion de faire connaissance avec l'outil de développement d'interfaces graphiques, Matisse, fourni avec Netbeans. N'hésitez pas à (re)lire le « Guide de l'utilisateur Swing ».

Avant de commencer l'exercice, nous vous recommandons la lecture du « Guide de designer MVC ».

ADDITIONNEUR MULTI-VUES

1 Préalables

Afin d'appliquer le design pattern MVC, nous allons écrire un **additionner** disposant de plusieurs vues. Le **modèle**, très simple, permettra de faire l'addition de deux nombres. L'application disposera quant-à elle de plusieurs vues différentes de ce modèle.

Le **modèle** vous est donné dans un package *additionneur.modele* comprenant les classes:

- AdditionneurModele, une interface spécifiant le modèle
- Additionneur, l'implémentation du modèle
- AdditionneurVue, une interface définissant de la vue
- AdditionneurOverflowException, une exception
- Etat, du modèle et Termes, les opérands de l'addition, classes particulières au modèle

Les classes que vous aller écrire sont des **composants** qui pourront être ajoutés à un *container* (JPanel ou un JFrame), ces classes hériteront de JComponent. Par facilité, une classe *AbstractAdditionneurVue* implémentant *AdditionneurVue* et héritant de JComponent vous est fournie dans le package *additionneur.vue*.

La figure suivante montre un exemple de mise en œuvre utilisant une classe *AdditionneurVueSaisieCurseur* et une classe *AdditionneurVueSimple*. Ces deux classes héritent de *AbstractAdditionneurVue* et sont ajoutées comme composant d'une classe principale héritant de JFrame (ce JFrame contient les deux composants).



Le code de la classe principale est simpliste dans ce cas,

```
package additionneur.vue;

import ...

public class MainGUISimple extends javax.swing.JFrame {
    private AdditionneurModele modele ;

    public MainGUISimple() {
        modele = new Additionneur() ;
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Additionneur");
        add(new AdditionneurVueSaisieCurseur(modele),
            BorderLayout.NORTH) ;
        add(new AdditionneurVueSimple(modele),
            BorderLayout.CENTER) ;
        pack() ;
    }

    public static void main(String args[]) {
        ...
    }
}
```

2 Exercice

Nous vous demandons d'écrire un composant `AdditionneurVueSaisieCurseur`. Ce composant est une vue du modèle ne s'intéressant qu'à la modification du modèle (par appel à la méthode `calcule` qui se chargera d'informer ces écouteurs) en fonction des valeurs des deux curseurs (*slider*). Cette classe hérite de `AbstractAdditionneurVue`, ce qui lui permet d'être un `JComponent` en passant.

Vous utiliserez l'outil de conception d'interfaces graphiques de Netbeans mais vous éviterez d'utiliser le `GroupLayout` (Java 1.6) comme *layout manager*.

Nous vous demandons d'écrire une classe principale **MainGUI** affichant le composant `AdditionneurVueSaisieCurseur`.

A ce stade, rien ne se passe. Écrivez une classe `AdditionneurVueSimple` permettant de voir le résultat d'une addition de manière « simple », comme ci-dessous



Ajoutez un menu à votre classe principale et un élément de menu permettant d'afficher une vue. Cet affichage crée un `JFrame` contenant votre composant « additionneur simple ».

Il reste maintenant à écrire quelques autres vues ...

- une vue simple en hexadécimal, se basant sur votre première vue simple,
- une vue binaire, à base de leds¹

¹ Les revoilà, je vous l'avais dit.

La vue simple en hexadécimal ne devrait pas poser de problèmes. Pour l'autre vue (en binaire à base de leds), vous aurez probablement besoin d'écrire un composant `NombreBinaireLeds` – basé sur le travail du TD sur les java bean— représentant un nombre binaire de n leds. Lorsque ce composant est écrit, en disposer trois dans une vue n'est plus difficile.

Le résultat pourrait ressembler à ceci.

