Master Informatique — **Approche Objet – 4TIN706U**
Université Bordeaux — **2016 – 2017**

TD N^o5 Model, View Controller

Une calculette

Le but de cet exercice est de réaliser une application java pour une petit calculette telle que celle-ci :

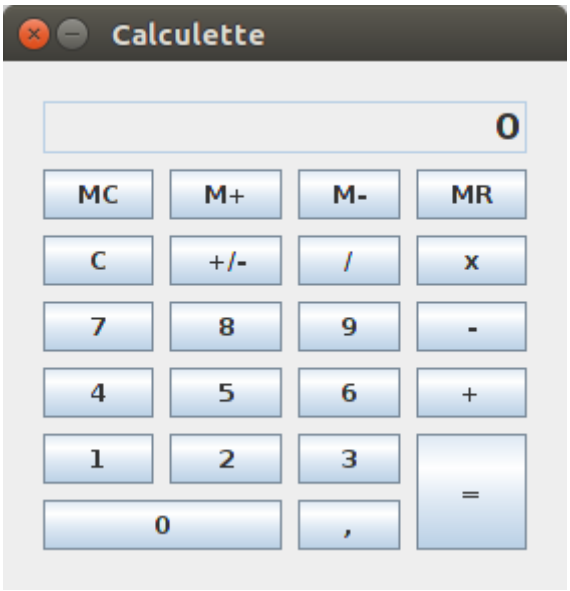

FIGURE 1 – Calculette

Vous pourrez trouver l'exécutable ici :
`http://www.labri.fr/perso/clement/enseignements/ao/`.

C'est un prétexte pour écrire convenablement un projet en respectant le *design patern Model-View-Controller*.

Nous invitons l'étudiant à suivre pas-à-pas cette feuille de TD pour arriver au résultat.

1 Créer une interface homme-machine vide

1. Créer un *package* `calculette.view` et y ajouter une classe `View` pour réaliser la présentation graphique de la calculette. La laisser vide pour l'instant.
2. Créer un *package* `calculette.model` et une classe `Model` pour réaliser le module fonctionnel de la calculette. La laisser vide pour l'instant.
3. Créer un *package* `calculette.controller` et une classe `Controller` pour réaliser l'interface de la calculette. La laisser vide pour l'instant.
4. Créer la classe `Main` dont le code est le suivant :

Listing 1 – Main.java

```
import calculette.controller.Controller;

public class Main {
        public static void main(String[] args) {
                new Controller();
        }
}
```

2 Appliquer le *design pattern* `Singleton` aux trois classes `Model`, `View` et `Controller`

On appliquera le *design pattern* `Singleton` pour s'assurer que chacune de leurs instances sont uniques.

Le code de la classe `Main` sera dorénavant le suivant :

Listing 2 – Main.java

```
import calculette.controller.Controller;

public class Main {
        public static void main(String[] args) {
                Controller.getInstance();
        }
}
```

3 Créer une interface graphique

Le but est de produire maintenant la fenêtre présentée en 1.

Dans le constructeur de la classe `View`, créer l'ensemble de l'application graphique

- Fenêtre principale : `frame`
  - Le titre est "Calculette" `setTitleframe("Calculette")`
  - Non redimensionnable `setResizable(false)`
  - Centrée `setLocationRelativeTo(null)`
  - Qui arrête l'application quand on ferme le fenêtre `setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`
  - Calcule sa mise en page `pack()`
  - S'affiche `setVisible(true)`
- Contenant principal : `panel`
  - Sa mise en page (*layout*) est un `FlowLayout`
  - Le marges horizontales et verticales sont de 16 pixels `setHgap(16)`, `setVgap(16)`
- Contenant secondaire : `component`
  - Sa mise en page (*layout*) est un `GridBagLayout`
  - Il est placé au centre du contenant principal :

```
component = new JPanel(new GridBagLayout());
panel.add(component, BorderLayout.CENTER);
```

  - Les contraintes de mise en page sont exercées par `constraints`

```
GridBagConstraints constraints = new GridBagConstraints();
```

Ces contraintes sont les suivantes :
  - `gridX`, `gridY` : position relative de la cellule du tableau en cours d'édition – $(0, 0)$ étant en haut à gauche
  - `gridWidth`, `gridHeight` : nombre de colonnes (resp. lignes) qu'occupe la cellule
  - `fill` : Si l'objet remplit l'espace libre horizontalement, verticalement ou les deux (`GridBagConstraints.BOTH`)
  - `anchor` : Position de l'objet dans la cellule en haut, à gauche, ... (`GridBagConstraints.CENTER`)
  - `insets` : Marges autour de la cellule
- Différents objets graphiques
  - `JTextField` : Zone de texte
    - Éditable : `setEditable(false)`
    - Aligné à droite : `setHorizontalAlignment(JTextField.RIGHT)`
    - Avec une police de caractères choisie : `setFont(new Font("SansSerif", Font.BOLD, 18))`
  - `JButton` : Boutons
    - Texte affiché sur le bouton : `setText("M+")`
    - Icône affichée sur le bouton : `setIcon(...)`
    - Commande associée au bouton : `setActionCommand("M+")`

  Ajouter les éléments graphiques dans le contenant `component` en exploitant les contraintes :

```
component.add(textField, constraint);
```

4 Créer une interface homme-machine

4.1 Faire communiquer la vue et le modèle avec le contrôleur

- Vue :
  Ajouter une méthode `void addActionListener(Controller controller)` à la classe `View`. Cette méthode se contentera d'appeler les méthodes `addActionListener(controller)` des différents objets graphiques interactifs (boutons).
  Appeler cette méthode lors de la construction de l'instance du contrôleur.
- Contrôleur :
  La classe `Controller` implémente maintenant l'interface `java.awt.event.ActionListener` qui contient la seule méthode `void actionPerformed(ActionEvent e)`. La méthode est évoquée quand un évènement survient (typiquement quand l'utilisateur appuie sur un bouton).
  Implémenter cette méthode `actionPerformed` comme ceci :

Listing 3 – actionPerformed

```
@Override
public void actionPerformed(ActionEvent e) {
        model.executeCommand(e.getActionCommand());
        view.drawData(model.getData());
}
```

Implémenter les méthodes `void executeCommand(String command)`, `String getData()`, `void drawData(String data)` en réalisant simplement un affichage en sortie standard des éléments passés pour l'instant.
Remarquer que le modèle et la vue sont en capacité de recevoir des données par l'intermédiaire des méthodes `void executeCommand(String command)` et `void drawData(String data)`, mais qu'il n'y a aucune communication directe entre les deux.

4.2 Rendre la vue interactive

Implémenter la méthode `void drawData(String data)` en faisant afficher les données dans le `textField`.

L'effet des boutons est déjà réalisé par l'intermédiaire de la méthode `addActionListener`.

4.3 Rendre le modèle interactif

Créer une propriété `String data` dans le modèle qui sera l'élément retourné par le *getter* `String getData()`.

Le but de la suite sera d'affecter une valeur correcte à `data` en fonction des données passées à la méthode `void executeCommand(String command)`.

5 Écrire le modèle

Bon, c'est bien beau tout ça, mais on n'a toujours aucune fonctionnalité de la calculette. Nous allons les ajouter.

5.1 Écrire les expressions arithmétiques sous la forme d'arbres binaires

— Une feuille est une opérande
— Un nœud interne est un opérateur

1. Créer la classe `Node` dans le *package* `calculette.model` qui contient :
   — Le type de nœud : (`enum UNARY_OPERATOR, BINARY_OPERATOR, OPERAND, ERROR`)
   — L'opérateur : (`enum PLUS, MINUS, MUL, DIV, UMINUS, NOP`)
   — L'opérande : (`String`) qui sert à conserver l'écriture d'un nombre
   — La valeur calculée : (`double`)
   — les fils gauche et droite
2. Implémenter les méthodes
   — `void addDigit(String digit)` qui ajoute un chiffre à un nœud existant. Astuce : on se servira de la valeur de l'opérande (`String`) pour concaténer un chiffre à sa droite.
   — `Node addBinaryOperator(String operator)` et `Node addUnaryOperator(String operator)` qui renvoie un nouveau nœud étiqueté par l'opérateur passé en paramètre.
   — `void addDot()` qui ajoute la virgule d'un nombre.
   — `void eval()` qui évalue l'arbre.
   — `String toString()` qui renvoie une chaîne de caractères (celle qui sera affichée dans l'afficheur de la calculette).
   — `reset` qui initialise l'arbre à la valeur zéro.
   — `error` qui affecte le type `ERROR` à l'arbre.
3. Ajouter la propriété `private Node root` à la classe `Model`.
4. Implémenter la méthode `void executeCommand(String command)` pour réaliser l'action attendue par la touche pressée et affecter à `data` la valeur de `root.toString()`.
5. Supplément : implémenter les éléments permettant de gérer la mémoire.