TP4 : Préparer une IHM pour le simulateur d'ascenseur

Une première version

Si vous n'avez jamais réalisé d'IHM en java, vous pouvez vous inspirer de l'exemple de la calculatrice placé à la fin de ce document.

Votre interface devra offrir (pour *N* niveaux) :

- Les *N* boutons de la cabine.
- Les *N-1* boutons *UP*.
- Les *N-1* boutons *DOWN*.
- Le bouton *HALT*.
- Le bouton *RESET*.
- Une visualisation du message envoyé à l'utilisateur (l'étage en général).
- Une visualisation de la position de l'ascenseur.
- Une visualisation des requêtes à traiter.

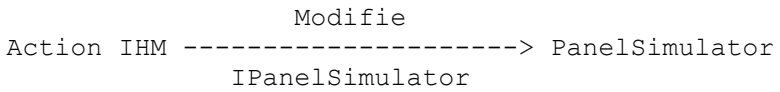
Architecture :

- L'IHM devra avoir des références vers le simulateur, le contrôle-commande ainsi que l'ordonnanceur.
- IHM se chargera de créer et d'agencer ces composants.

Une deuxième version

Dans la version précédente le contrôle-commande reçoit directement les requêtes de l'IHM qui simule les boutons. Ce n'est pas très réaliste.

- Ajouter à votre IHM une référence vers le simulateur de panneau et modifier votre IHM afin que les actions sur les boutons soient renvoyées vers le simulateur de panneau.



- Introduire un nouveau composant (le PanelControlCommand) qui a pour objectifs de scruter le panneau et de retransmettre les évènements détectés au CC.

Une troisième version

Nous devons maintenant nous occuper des lumières.

- Modifier votre CC afin qu'il allume/éteigne les boutons du panneau en fonction des requêtes enregistrées ou traitées.

- Modifier votre IHM afin qu'elle détecte les changements de lumières et qu'elle répercute ces changements sur la couleur des boutons.

```
          allume/éteint                          est scruté par
CC ------------------> PanelSimulator ---------------------> IHM
          IPanel                                 IPanelSimualtor
```

Une quatrième version

Modifier votre CC afin de générer des messages qui seront scrutés et affichés par l'IHM.

```
        message                            est scruté par
CC -------------> PanelSimulator ---------------------> IHM
        IPanel                            IPanelSimualtor
```

Une calculatrice

```
package calculator;

import java.awt.BorderLayout;
import java.awt.GridLayout;
```

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;

public class Calculator {

    JTextArea expression;

    /*
     * Créer un bouton et lui associer une action
     */
    private JButton newButton(String name, ActionListener action) {
        JButton button = new JButton(name);
        button.addActionListener(action);
        return button;
    }

    /*
     * Construire l'IHM de la calculatrice
     */
    public void run() {
        var frame = new JFrame();
        expression = new JTextArea();
        var panelNumber = new JPanel(new GridLayout(4, 3));
        var panelOp = new JPanel(new GridLayout(4, 1));
```

```
    panelOp.setSize(10, 100);
    var panel = new JPanel();

    frame.getContentPane().add(expression, BorderLayout.NORTH);

    for (int i = 1; i < 10; i++) {
        panelNumber.add(newButton(i + "", this::actionBuild));
    }

    panelNumber.add(newButton("C", this::actionClear));
    panelNumber.add(newButton("0", this::actionBuild));
    panelNumber.add(newButton("=", this::actionCalculate));

    panelOp.add(newButton("+", this::actionBuild));
    panelOp.add(newButton("-", this::actionBuild));
    panelOp.add(newButton("*", this::actionBuild));
    panelOp.add(newButton("/", this::actionBuild));

    panel.add(panelNumber, BorderLayout.CENTER);
    panel.add(panelOp, BorderLayout.EAST);
    frame.getContentPane().add(panel, BorderLayout.CENTER);

    frame.setSize(200, 200);
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
}

/*
 * Effacer l'expression
```

```
     */
    private void actionClear(ActionEvent event) {
        expression.setText("");
    }

    /*
     * Mémoriser un chiffre ou une opération
     */
    private void actionBuild(ActionEvent event) {
        if (expression.getText().equals("Erreur")) {
            expression.setText("");
        }
        String str = event.getActionCommand();
        expression.setText(expression.getText() + str);
    }

    /*
     * Evaluer l'expression
     */
    private void actionCalculate(ActionEvent event) {
        String syntax = "[0-9]+[-+*/][0-9]+"; // so the delimiters are: + - * /
        String delims = "[-+*/]"; // so the delimiters are: + - * /

        var exp = expression.getText();
        if (!exp.matches(syntax)) {
            expression.setText("Erreur");
            return;
        }
        var tokens = exp.split(delims);
```

```
        var op1 = Integer.parseInt(tokens[0]);
        var op2 = Integer.parseInt(tokens[1]);
        var res = 0;
        if (exp.contains("+")) {
            res = op1 + op2;
        } else if (exp.contains("-")) {
            res = op1 - op2;
        } else if (exp.contains("/")) {
            res = op1 / op2;
        } else if (exp.contains("*")) {
            res = op1 * op2;
        } else {
            expression.setText("Erreur");
            return;
        }
        expression.setText(exp + "=" + res);
    }

    /*
     * Lancement de la calculatrice
     */
    public static void main(String[] args) {
        var cal = new Calculator();
        cal.run();
    }

}
```