TD PAC - Java n° 7
(Correction)

# Conception d'interfaces graphiques avec Java (1)

L'objectif de ce TD est de vous familiariser avec les principaux composants graphiques des librairies SWING, afin de pouvoir concevoir des interfaces graphiques en JAVA. Etant donné la richesse des librairies, il est impossible de tout connaître par coeur... Ce travail est donc aussi l'occasion d'apprendre à se familiariser avec la documentation de java.

## I) Un mot sur la documentation de java

**Exercice 1** La documentation de java est disponible en ligne sur le site de `Sun Microsystems` à l'url : `http://java.sun.com/j2se/1.4.2/docs/` (penser à consulter la doc adéquate à la version du `jdk` installée sur votre machine). Pour un accès plus rapide, il est aussi possible que la doc soit aussi installée localement sur votre site. Renseignez vous auprès de votre administrateur local.

La page d'accueil de la documentation donne un certain nombre de pointeurs vers des documents utiles mais ce qui nous intéresse le plus ici est le lien appellé `Java 2 Platform API Specification`, qui donne accès à la documentation de l'ensemble des classes existantes du `jdk`. Cette documentation a été générée automatiquement à l'aide de `javadoc`. Cet outil est capable d'exploiter un format spécial de commentaires dans des sources java, pour générer automatiquement une documentation au format `html`, consultable à l'aide de n'importe quel navigateur[1].

Vous noterez que la page principale est découpée en trois zones. Dans la première (en haut à gauche) se trouvent la liste des packages, dans le seconde (en bas à gauche) une liste de classes et interfaces et dans la troisième, la page principale de description d'une classe, interface, package, ....

1. Familiarisez vous avec l'organisation de la doc. Que se passe-t-il si l'on clique sur un nom de package dans la fenêtre des packages ? Que se passe-t-il si l'on clique sur le nom du package dans la fenêtre des classes et interfaces ?
2. Retrouver les pages décrivant les classes JFrame, JPanel et JButton.
3. Pour chacune de ces classes, trouver à quel package elles appartiennent, quels sont leurs constructeurs, de quelles classes elles dérivent (i.e. leur position dans la hierarchie des classes) et les méthodes qui leurs sont applicables (notamment par héritage)

**Correction :** Rien de particulier à dire pour cet exercice... consacrer environ 10-15mn. Peut-être insister un peu plus sur la notion de package dont nous avons assez peu parlé jusqu'à présent.

## II) Conception d'une interface graphique simple

Nous nous proposons ici de réaliser une interface pour une calculette simple, dont nous enrichissons progressivement l'interface. La calculette elle même n'est pas à réaliser. Nous vous fournissons une classe `Calculette` déjà toute prête. De façon générale, lors de la mise en oeuvre d'une application informatique, il est préférable de ne pas mélanger les fonctionalités principales de l'application, avec les aspects liés à l'interface utilisateur. Il vaut mieux concevoir l'interface séparément et l'associer à l'application centrale. Ceci nécessite naturellement de bien comprendre et spécifier au préalable, comment les deux composantes communiquent.

[1] Vous aussi, lors de l'écriture de vos classes, avez tout intérêt à prendre rapidement l'habitude d'écrire vos commentaires de façon à ce qu'ils soient exploitables par `javadoc`. Vous devrez notamment l'utiliser pour documenter votre travail de TER

Dans le cas présent, vous ne disposez pas du source de la classe Calculette mais seulement du fichier `Calculette.class`. Vous pouvez néanmoins vous faire une idée de ce que cette classe met à disposition à l'aide de la commande :

```
javap <NomDeClasse>
```

Cette commande vous indiquera la liste des constantes et méthodes publiques définies dans cette classe. Utiliser cette commande pour inspecter la classe `Calculette`.

La calculette dont nous disposons ici est supposée disposer de 16 touches, correspondant aux 10 chiffres, aux quatre opérations de base, et aux touches `[=]` et `[C/CE]`. La méthode `input(n)` sert ici à répercuter sur la calculette ce qui se passe lorsque l'on appuie sur l'une des touches. La méthode `getAff()` permet de récupérer le contenu de l'écran de la calculette, qui peut être affiché à l'aide de la méthode `affiche()`.

**Exercice 2** La plupart des interfaces graphiques se définissent en dérivant une nouvelle classe à partir de la classe `JFrame`.

1. Définir une classe `IGCalculette`, comme une extension de `JFrame`. Cette classe devra contenir un attribut lui permettant d'interagir avec une calculette. Cependant cette calculette devra également pouvoir transmettre les résultats de ses calculs à l'interface Graphique. Par conséquent, définir parallèlement une classe `CalculetteGraphique` comme une extension de la classe `Calculette` et comportant un attribut de la classe `IGCalculette`.

   **Correction :** Insister sur la necessité que chaque classe ait un attribut correspondant à l'autre classe.
   Préciser comment effectuer le référencement mutuel, en créant l'autre objet de l'autre classe en lui passant `this` dans le constructeur pour que lui aussi puisse avoir la référence vers l'objet qui l'a créé. Dans le corrigé, j'ai mis deux constructeurs dans chaque classe afin de mettre en évidence le rôle symétrique des deux classes.
   Difficulté possible : l'appel au constructeur parent (avec `super(...)`) doit absolument s'effectuer avant que l'on parle de `this`.

   Le code est dans le dossier `exo1_1`.

2. Il est généralement préférable de regrouper préalablement les différents composants de l'interface dans différents `JPanel` que l'on intègre au Container de la fenêtre principale (JFrame). (voir la méthode de la classe JFrame *Container getContentPane()* pour récupérer le Container d'une JFrame). Définir deux JPanels. Le premier servira pour le clavier de la calculette. Le second servira pour l'écran et éventuellement des composants permettant de configurer l'affichage sur l'ecran. Afin de pouvoir visualiser ces deux JPanels, leur donner des couleurs de fond différentes..

   **Correction :**
   Pour cette question les étudiants doivent :
   - comprendre comment créer des JPanels,
   - comprendre comment modifier leur couleur de fond (avec la méthode `setBackground`) ce qui va les obliger aussi à aller voir la classe `Color`,
   - comprendre qu'il faut préalablement définir le `LayoutManager` d'un Container avant d'y ajouter des composants graphiques.

   Pour ne pas perdre de temps on peut les guider vers le choix d'un BorderLayout pour le Container principal. Le code est dans le dossier `exo1_2`.

```
private void initIG() {
    setSize(280,350);
  // Creation des JPannel
    JPanel clavier = new JPanel();
    clavier.setBackground(Color.gray);
    JPanel ecran = new JPanel();
    ecran.setBackground(Color.lightGray);
  // Disposition sur le JFrame
    Container c = getContentPane();
    c.setLayout(new BorderLayout());
```

```
        c.add(ecran,BorderLayout.NORTH);
        c.add(clavier,BorderLayout.CENTER);
        setVisible(true);
    }
```

3. On pourra définir l'écran comme un `JTextField`, que l'on ajoutera dans le JPanel de l'écran. Cependant (au moins dans un premier temps) il ne doit pas être directement éditable.

   Dans la mesure où une `CalculetteGraphique` doit pouvoir actualiser la valeur de cet écran, on le définira comme un attribut de l'interface graphique. Définir dans la classe `IGCalculette` une methode publique d'accès permettant de modifier la valeur du texte de cet attribut puis, dans la classe `CalculetteGraphique`, surcharger la méthode `affiche()` afin qu'elle modifie dans l'interface graphique associée la valeur de l'écran.
   Tester que cela fonctionne bien en réalisant une opération simple dans une méthode `main(...)` de la classe `CalculetteGraphique`.

   **Correction :**
   Il faut penser à associer un LayoutManager sur JPanel correspondant a l'écran. On peut éventuellement toucher un mot sur les fontes, puisque dans la suite on sera amené a les modifier. Dans le fichier `IGCalculette` on rajoute :

```
public class IGCalculette extends JFrame{
 ...
    /** La zone d'affichage correspondant a l'ecran de la calculette */
    JTextField _affichage;
...
    private void initIG() {
        setSize(280,350);

      // Zone d'affichage de la calculette
        _affichage = new JTextField("0"); //
        _affichage.setEditable(false);         // L'ecran de doit pas etre editable
        _affichage.setHorizontalAlignment(JTextField.RIGHT);
                                                // L'ecran doit etre justifie a droite
    // Optionnel - le choix de la fonte de l'ecran
       _affichage.setFont(new Font("Times",Font.BOLD,36));

      // Creation des JPannel
        JPanel clavier = new JPanel();
        clavier.setBackground(Color.gray);
        JPanel ecran = new JPanel();
      //  ecran.setBackground(Color.lightGray);

      // Ajout des composants sur l'ecran
        ecran.setLayout(new BorderLayout());
        ecran.add(_affichage,BorderLayout.CENTER);
        ecran.add(_affichage);

      // Disposition sur le JFrame
        Container c = getContentPane();
        c.setLayout(new BorderLayout());
        c.add(ecran,BorderLayout.NORTH);
        c.add(clavier,BorderLayout.CENTER);

     // Visualisation
        setVisible(true);
    }

   public void setAffichage(String s){
        _affichage.setText(s);
   }
```

Dans le fichier `CalculetteGraphique` on rajoute :

```
    /**
     * Affiche la valeur de l'ecran de la calculette
     */
     public void affiche() {
        super.affiche();
        _ig.setAffichage(""+getAff());
    }



//----------------------
//   Pour le test
//----------------------

  public static void main(String args[]){
    CalculetteGraphique c = new CalculetteGraphique();
    c.input(2);
    c.input(Calculette.PLUS);
    c.input(3);
    c.input(Calculette.EQUAL);
    c.affiche();
  }
```

Le code est dans le dossier `exo1_3`.

**Exercice 3** Pour réaliser le clavier de la calculette, nous allons utiliser des composants de type JButton. Ces composants étant utilisés pour le contrôle de l'interface, il est nécessaire de les déclarer comme attributs de la classe `IGCalculette`.

1. Créer un nouvel attribut pour l'interface graphique, correspondant à un tableau de boutons pour le clavier (nb : on pourra de façon utile créer des constantes de classe pour référencer les boutons qui ne correspondent pas à des chiffres par des entiers).

   **Correction :**

```
public class IGCalculette extends JFrame{

// Constantes de la classe
//-------------------------
    private static final int NB_BOUTONS = 16;
 /** Codes pour les boutons */
    private static final int PLUS = 10;
    private static final int MOINS = 11;
    private static final int MULT = 12;
    private static final int DIV =  13;
    private static final int EQUAL = 14;
    private static final int CLEAR = 15;

// Attributs de la classe
//-------------------------

    /** La calculette graphique associee a l'interface */
    CalculetteGraphique _cg;
    /** La zone d'affichage correspondant a l'ecran de la calculette */
    JTextField _affichage;
    /** Les boutons du clavier de la calculette */
    JButton[] _btclavier;
```

2. Lors de l'initialisation de l'interface, créer les 16 boutons , chacun d'entre eux étant étiqueté par le symbole approprié, et disposer le tout de façon adéquate sur le Panel correspondant au

clavier.
Quel est le Layout manager qui vous semble le plus approprié ?

**Correction :** Il faut ici penser à utiliser un LayoutManager de type GridLayout. Une matrice $4 * 4$ fait parfaitement l'affaire

```
...
  // Boutons pour le clavier
        _btclavier = new JButton[NB_BOUTONS];
        // Creation des boutons
        for (int i = 0; i < 10 ; i++)
        _btclavier[i] = new JButton(""+i);
        _btclavier[PLUS] = new JButton("+");
        _btclavier[MOINS] = new JButton("-");
        _btclavier[MULT] = new JButton("*");
        _btclavier[DIV] = new JButton("/");
        _btclavier[EQUAL] = new JButton("=");
        _btclavier[CLEAR] = new JButton("C/CE");
.....
      // Ajout des boutons sur le panel de clavier
      // ligne par ligne
        clavier.setLayout(new GridLayout(4,4));

        clavier.add(_btclavier[1]);
        clavier.add(_btclavier[2]);
        clavier.add(_btclavier[3]);
        clavier.add(_btclavier[MULT]);

        clavier.add(_btclavier[4]);
        clavier.add(_btclavier[5]);
        clavier.add(_btclavier[6]);
        clavier.add(_btclavier[DIV]);

        clavier.add(_btclavier[7]);
        clavier.add(_btclavier[8]);
        clavier.add(_btclavier[9]);
        clavier.add(_btclavier[MOINS]);

        clavier.add(_btclavier[0]);
        clavier.add(_btclavier[CLEAR]);
        clavier.add(_btclavier[EQUAL]);
        clavier.add(_btclavier[PLUS]);
```

Le code est dans le repertoire `exo3_2`.

### Exercice 4

**Correction :** Dans cet exercice surtout on des composants swing de différentes sortes pour illustrer dans l'exercice suivant que selon le type de composant, on utilise le Listener approprié.

Nous souhaitons illustrer l'utilisation d'autres composants SWING en rajoutant quelques outils de contrôle de l'affichage de l'écran.

1. Créer un nouveau JPanel pour recevoir ces outils de configuration et le disposer au dessus de la zone d'affichage dans le pannel correspondant à l'écran.

   **Correction :** Rien de particulier :

   ```
   // Panel Pour les options de configuration
         JPanel configOptions = new JPanel();
         configOptions.setLayout(new FlowLayout());
         ....
         ecran.add(configOptions,BorderLayout.NORTH);
   ```

2. Ajouter successivement sur ce panel
   (a) un `JLabel` étiqueté par "Fonte"
   (b) un `JComboBox` auquel on associera des noms de fontes
   (c) deux `JCheckBox` auquelles on associera les chaînes "gras" et "italique"

   Ces outils seront utilisés pour contrôler le choix de la fonte utilisée pour la zone d'affichage.

**Correction :**

On a tout intérêt à définir un tableau constant pour les noms des fontes.

Ceci sera utile pour retrouver le nom de la fonte capturée par le listener par son indice dans le tableau. NB : j'ai agrandi un tout petit peu la taille de la fenêtre sinon la barre de configuration ne tient pas entièrement Le code est dans le repertoire `exo4_2`.

```
// Constantes de la classe
//------------------------
  ...
 /** Liste de noms de fontes */
    private static final String[] FONT_NAMES =
        {"Helvetica","Arial","Courier","LucidaBright","Symbol","Verdana","TimesRoman",};
...
// Attributs de la classe
//------------------------
 ...
     // Controle de l'affichage sur l'ecran */
    /** Controle du style de l'affichage sur l'ecran */
    JCheckBox _cbBold, _cbItalic;
    /** Controle de la fonte de l'affichage sur l'ecran */
    JComboBox _comboFonts;
```

Et dans `initIG` :

```
...
       // CheckBox pour le style
        _cbBold = new JCheckBox("Gras");
        _cbItalic = new JCheckBox("Italique");

        // JComboBox pour le choix de la police
       _comboFonts = new JComboBox(FONT_NAMES);

        configOptions.add(new JLabel("Fonte "));
        configOptions.add(_comboFonts);
        configOptions.add(_cbBold);
        configOptions.add(_cbItalic);
. ....
```

# III) Conception d'une l'interface réactive

**Exercice 5** Pour l'instant notre interface graphique est quasiment *passive*. Nous avons pu constater que la calculette graphique était capable d'agir sur l'écran de l'interface, mais l'interface n'est pas capable d'agir sur la calculette. Pour cela, il est nécessaire d'associer à chacun des composants graphiques un gestionnaire d'événements.

1. Les JButtons sont généralement associés à des gestionnaires de type `ActionListener` pour réagir aux clics de souris. Ici chaque bouton du clavier a un effet différent. On peut imaginer créer un gestionnaire différent pour chaque bouton... mais avec 16 boutons, cela risque d'être un peu lourd !. Une autre faç on de faire est de ne créer qu'un seul gestionnaire, que l'on associe à chaque bouton, mais dont la méthode `ActionPerformed` qu'il implémente, est capable d'adapter son action en fonction du bouton qui a été cliqué. Pour connaître le bouton à l'origine

de l'événement, on peut utiliser la méthode `getSource` de la classe `EventObject`.

Implémenter une *classe interne* pour un tel gestionnaire d'événements et associer ce gestionnaire à l'ensemble des boutons du clavier.
Tester maintenant la réactivité de votre interface graphique aux événements concernant le clavier de votre calculette graphique.

**Correction :**
Penser à rajouter au début :

```
import java.awt.event.*;      // pour gerer les evenements
```

Ici on peut rajouter une classe interne, mais pas anonyme (on utilise des classes anonymes lorsque l'on n'a besoin que d'une instance de celle-ci or, ici, il nous en faut 16).

```
/**
 * gestionnaire d'evenements associe aux boutons du clavier
 */
private class BoutonClavierHandler implements ActionListener {
 public void actionListener(ActionEvent e) {
     Object objetClique =  e.getSource();
     int i = 0 ;
     while (objetClique != _btclavier[i])
         i++ ;
     if (i < NB_BOUTONS) {
        switch (i) {
          case PLUS :
                 _cg.input(Calculette.PLUS);
                 break;
          case MOINS :
                 _cg.input(Calculette.MOINS);
                 break;
          case MULT :
                 _cg.input(Calculette.MULT);
                 break;
          case DIV :
                  _cg.input(Calculette.DIV);
                 break;
          case EQUAL :
                  _cg.input(Calculette.EQUAL);
                 break;
          case CLEAR :
                 _cg.input(Calculette.CLEAR);
                 break;
          default :
                 _cg.input(i);
        }
      }
      else
     System.err.println("Action Listener : Mauvais click ! ");
     }
}
```

et après la création des boutons lors de l'initialisation du clavier :

```
// Gestion de l'interaction utilisateur sur les boutons du clavier
BoutonClavierHandler hbc = new BoutonClavierHandler();
for (int i = 0; i < NB_BOUTONS ; i++)
    _btclavier[i].addMouseListener(hbc);
```

2. De manière similaire, associer un gestionnaire d'événements aux deux JCheckBoxes permettant de contrôler le style de l'affichage de l'écran (ici, c'est l'interface `ItemListener` qui doit être implémentée).

**Correction :** Le processus est similaire avec une classe interne de gestionnaire pour ces checkboxes :

```
/**
 * gestionnaire d'evenements associe aux JCheckBox permettant
 * de controler le style de la police utilisee pour l'affichage
 */
 private class FontStyleCheckBoxHandler implements ItemListener {

   private int _bold;
   private int _italic;

 public void itemStateChanged(ItemEvent e) {
     Object cboxClique =  e.getSource();
     if (cboxClique == _cbBold) {
        if (e.getStateChange() == ItemEvent.SELECTED)
             _bold = Font.BOLD;
        else
             _bold = Font.PLAIN;
     }
     if (cboxClique == _cbItalic) {
        if (e.getStateChange() == ItemEvent.SELECTED)
             _italic = Font.ITALIC;
        else
             _italic = Font.PLAIN;
     }
     _affichage.setFont(_affichage.getFont().deriveFont(_bold+_italic));
     _affichage.repaint();
   }
}
```

et l'ajout d'un handler sur les JCheckBox concernés lors de l'initialisation :

```
// gestionnaire d'evenements pour les boutons JCheckBox
FontStyleCheckBoxHandler fsch = new FontStyleCheckBoxHandler();
_cbBold.addItemListener(fsch);
_cbItalic.addItemListener(fsch);
```

3. De manière similaire, associer un gestionnaire d'événements à la JComboBox permettant de contrôler la police de l'affichage de l'affichage. Dans la mesure un tel gestionnaire ne concerne qu'un seul composant, on pourra le définir directement comme une classe anonyme apparaissant comme paramètre de la méthode qui ajoute le gestionnaire approprié à ce type de composant.

**Correction :** Ici il faut dire un mot sur la syntaxe des objets de classes anonymes. En particulier il est bon de préciser que que la syntaxe : `new ItemListener(){...}` ne crée pas d'objet de la classe ItemListener (puisqu'il s'agit en fait d'une interface). C'est juste un raccourci pour dire :

```
Crée un objet d'une classe (anonyme) qui impl émente l'interface ItemListener
```

```
// gestionnaire d'evenements pour la JComboBox
_comboFonts.addItemListener(
    new ItemListener(){
        public void itemStateChanged(ItemEvent e){
            _affichage.setFont(new Font(FONT_NAMES[_comboFonts.getSelectedIndex()],
                                _affichage.getFont().getStyle(),
                                _affichage.getFont().getSize())
                               );
            _affichage.repaint();
        }
    }
    );
```

4. Vous avez surement remarqué que la fermeture de la fenêtre correspondant à votre interface graphique de calculette ne provoque pas pour autant la terminaison du processus java. Pour cela il faut aussi associer un gestionnaire d'événements, à votre objet qui étend la classe JFrame. Associer un tel gestionnaire permettant de terminer le processus lorsque se produit la fermeture de la fenêtre (On aura intérêt à utiliser un `WindowAdapter`).

   **Correction :** De même on pourra rappeler le principe des *Adapters*, qui sont des classes déclarées abstraites (et dont on ne peut donc pas créer d'instance) qui implémentent toutes les méthodes de l'interface correspondantes mais avec des corps vides. Ici encore la notation `new WindowAdapter(){...}` est un raccourci pour dire :

   *Crée un objet d'une classe (anonyme) qui étend la classe WindowAdapter*

   Rappel : ceci est utile lors que l'interface comporte plusieurs méthodes mais que l'on souhaite seulement en définir un petit nombre.

```
addWindowListener(
new WindowAdapter() {
   public void windowClosing( WindowEvent e )
   {
      System.exit( 0 );
   }
}
);
```

   Le code de tout l'exercice est dans le dossier `exo5_4`.