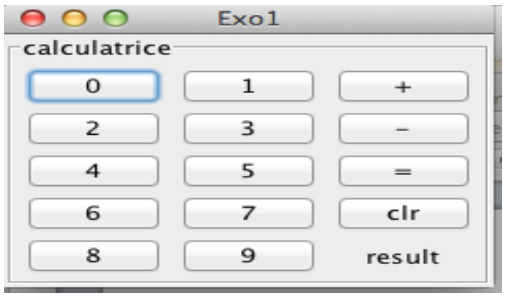
# TD/TP PO2 Licence Informatique 3 (2015 – 2016)

## Exercice 1

Ecrire une interface graphique qui contient :

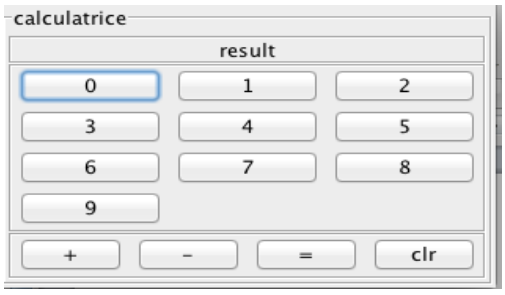
- 10 boutons correspondant aux chiffres : 0, 1, …,9
- 4 boutons correspondant aux opérations : + - = clr
- 1 label pour afficher le résultat




## Exercice 2

Idem avec 3 zones :

- une zone supérieure pour le résultat
- une zone au milieu pour des chiffres
- une zone inférieur pour des opérations




## Exercice 3

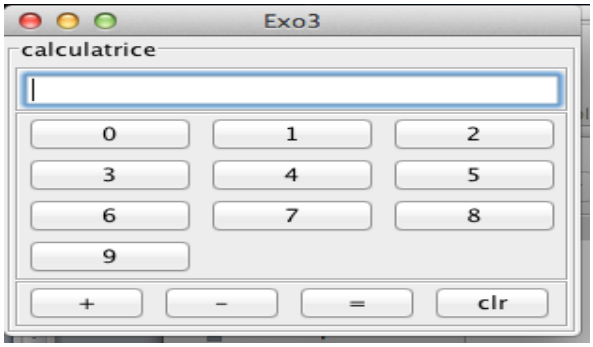
Idem avec 3 zones :

- une zone supérieure pour afficher le résultat
- une zone au milieu pour des chiffres
  - nous vous donnons l'écouteur avec classe interne pour les boutons chiffres, et vous créez un seul écouteur partagés par tous les boutons chiffres :

```
class NumberBtnListener implements ActionListener {
      public void actionPerformed(ActionEvent evt) {
         numberInStr += evt.getActionCommand();
         tfDisplay.setText(numberInStr);
      }
   }
```

- une zone inférieur pour des opérations

- réaliser un seul écouter avec classe interne pour les boutons opérateurs « + », « - », « = ». Vous avez besoin de deux opérandes et deux opérateurs pour faire « + » et « - »: previousNb, currentNb, previousOpr, currentOpr.
- réaliser un écouteur avec classe anonyme pour le bouton « clr » qui effcace le résultat ancien




## Exercice 4

1. Ecrire une interface graphique permettant de choisir une couleur parmi 5 à l'aide de bouton radios et de l'afficher dans une zone rectangulaire. (Fig.1)
2. Enrichir l'interface par une liste de choix des couleurs. (Fig. 2)
3. Enrichir l'interface par un label qui affiche, en plus, la valeur hexadécimale de la couleur. (Fig. 3)
   Ci-dessous le code convertissant un entier en String de 2 chiffres hexa :

```
private String toHex(int val) {
        String result = Integer.toHexString(val).toUpperCase();
        if (result.length() == 1)
                return "0"+result ;
        else
                return result;
}
```

   Dans cet exercice, il devient difficile de gérer les interactions entre contrôleur et afficheurs, …

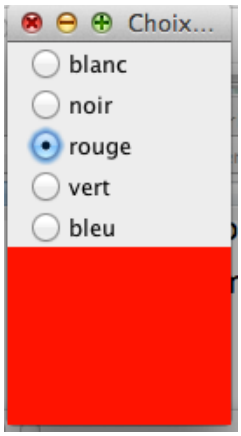



Fig. 1

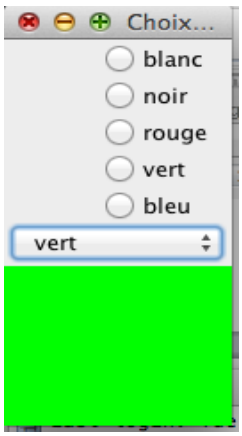



Fig. 2

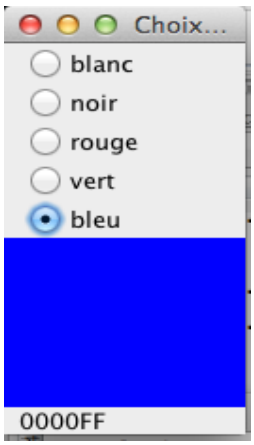
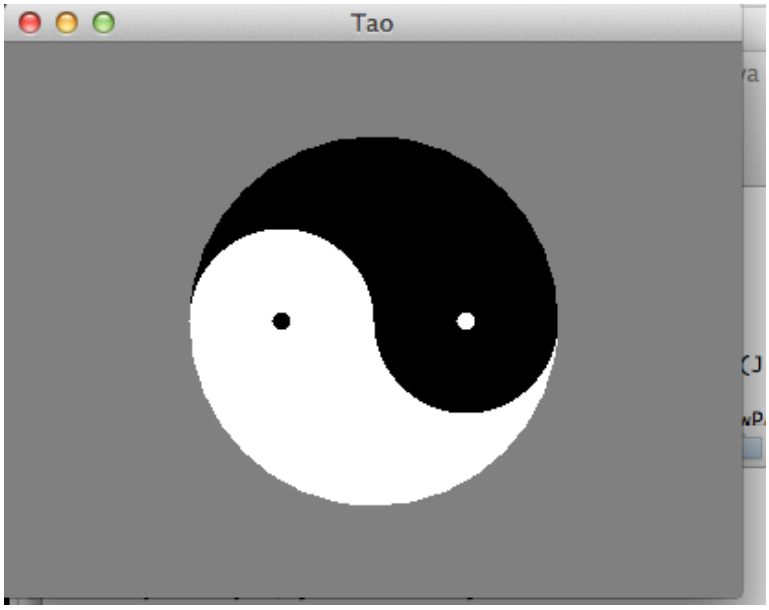



Fig. 3

## Exercice 5

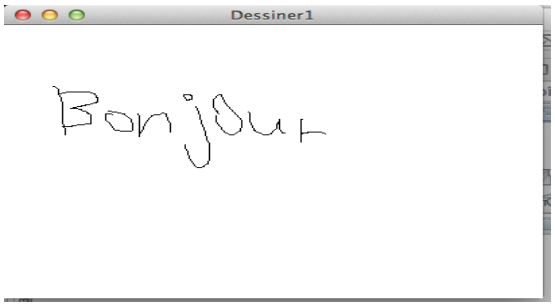
Dessiner le symbole « Yin-Yang » dans une interface :




Pour cela, décomposer le symbole en :
1. deux demi-cercles noir et blanc avec rayon r, r=100 par exemple ;
2. deux demi-cercles noir et blanc avec rayon r/2 ;
3. deux cercle avec un petit rayon r0, r0=5 par exemple.

## Exercice 6

Réaliser une interface graphique pour dessiner à la souris :




1. Dans cette 1ère version, le masquage de la fenêtre provoquera la perte du dessin. Compléter la classe Dessiner1.java en s'inspirant de l'exemple du cours MouseEvent :

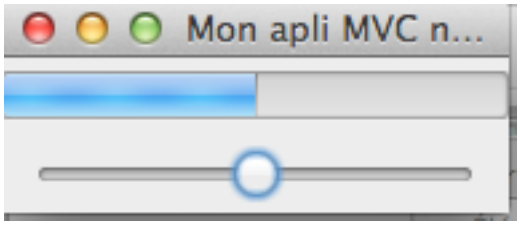
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Dessiner1 extends JFrame {
  private int lastX, lastY;
  private JPanel panelDessin;
  public Dessiner1() {
    super("Dessiner1");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

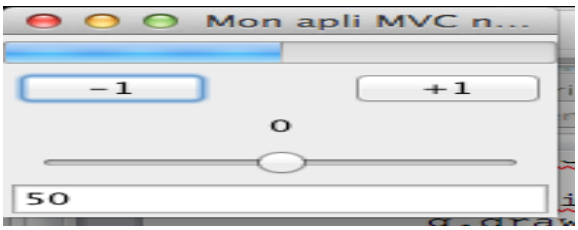
```
    panelDessin = new JPanel();
    panelDessin.setPreferredSize(new Dimension(400,300));
    panelDessin.setBackground(Color.WHITE);
....
    this.getContentPane().add(panelDessin);
    this.pack();
    this.setVisible(true);
  }
 public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
           new Dessiner1();
        }
    });
  }
}
```

2. Améliorer l'interface précédente, sans perte, le masquage de la fenêtre en définissant une classe PolyLine qui consiste en deux listes pour stocker les coordonnées X et Y de chaque point d'une trace.

## Exercice 7

On vous fournit une interface graphique qui permet de contrôler et visualiser un niveau entre 2 bornes limites : ici, un entier compris entre 0 et 100.




Le modèle est défini dans la classe Niveau, le contrôle dans la classe Tirette, et la vue dans la classe Curseur.

Vous disposez aussi d'une classe MonAppliMVC1 qui utilise le contrôle d'une tirette et l'affichage d'un curseur.

Maintenant, vous enrichissez cette interface par l'ajoute de :

- Un contrôle par boutons +1 et -1
- Un affichage de la valeur du niveau (grâce à un JLabel)
- Un contrôle et affichage par une zone de saisie

```
import java.util.*;
import javax.swing.*;
import java.awt.*;
public class MonAppliMVC1 extends JFrame {
  public MonAppliMVC1() {
super("Mon apli MVC numero 1");
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    Niveau modele = new Niveau(0,50,100);
    Tirette tirette = new Tirette(modele);
    Curseur curseur = new Curseur(0,50,100);
```

```
    modele.addObserver(curseur);

    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(2,1));
    panel.add(curseur);
    panel.add(tirette);
    this.getContentPane().add(panel);
    this.pack();
    this.setVisible(true);
  }

  public static void main(String[] args) {
    SwingUtilities.invokeLater( new Runnable() {
        public void run() {
          new MonAppliMVC1();
        }
      });
  }
}

------------------------
import java.util.Observable;
public class Niveau extends Observable {
  private int niveau, min, max;
  public Niveau(int mi, int n, int ma) {
    niveau=n; min=mi; max=ma;
  }
  public int getNiveau() { return niveau; }
  public int getMin() { return min; }
  public int getMax() { return max; }
  public void setNiveau(int nouveau) {
    if ((nouveau != niveau)  && (nouveau>=min) && (nouveau<=max))
    { niveau=nouveau;
      setChanged();
      notifyObservers(new Integer(niveau));
    }
  }
  public String toString() { return niveau+" in ["+min+","+max+"]"; }
}

----------------------------

import javax.swing.*;
import javax.swing.event .*;
public class Tirette extends JSlider {
  private Niveau modele;
  public Tirette(Niveau model) {
    super(model.getMin(),model.getMax(),model.getNiveau());
         this.modele=model;
    this.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
          modele.setNiveau(Tirette.this.getValue() );
    }} );
  }
}


----------------------------

import java.util.*;
import javax.swing.*;
public class Curseur extends JProgressBar  implements Observer {
```

```
    public Curseur(int min, int niv, int max) {
      super(min, max);
      setValue(niv);
    }
    public void update(Observable source, Object donnees) {
      if ((source instanceof Niveau) && (donnees != null)
          && (donnees instanceof Integer))
      { int niv= (Integer)donnees;
        this.setValue(niv);
      }
    }
  }
```

## Exercice 8

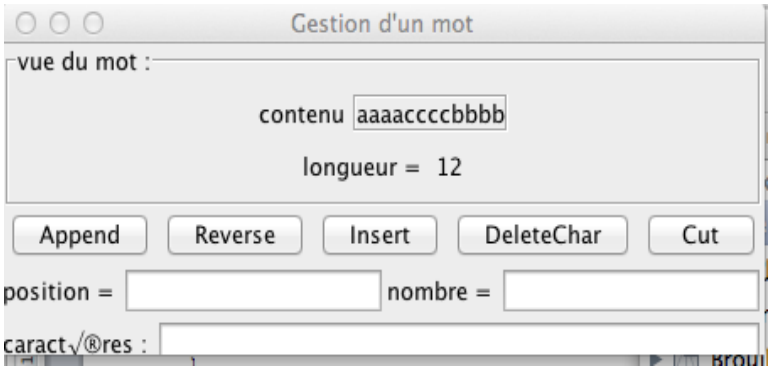
A partir de l'interface de l'exercice 4, réaliser une nouvelle interface avec MVC qui permet à l'utilisateur de

- choisir une couleur dans la palette en haut de l'interface, ou bien définir une nouvelle couleur par saisir ses intensités de rouge, vert et bleu en bas de l'interface ;
- voir la couleur obtenue au milieu de l'interface ainsi que ses intensités de rouge, vert et bleu en bas de l'interface.


## Exercice 9

Ecrire une interface graphique avec MVC qui permet de gérer un mot à la console : ajout à la fin, insertion, suppression d'un caractère ou d'un morceau, renverser, … Le modèle (Mot.java) est fourni. Commencer par un contrôleur et une vue en console, puis ajouter une vue graphique, puis un contrôleur graphique.

Ci-dessous le fichier Mot.java, une capture d'écran de l'interface graphique et de la console :




```
----------

----------
app aaaa
----------
aaaa
----------
app bbbb
----------
aaaabbbb
----------
ins 4 cccc
----------
aaaaccccbbbb
----------
```

```
import java.util.*;
/** gestion d'un mot
 */
public class Mot extends Observable
{
  /** le mot*/
  private StringBuffer mot;
  /** constructeur sans parametre */
  public Mot() {
    mot = new StringBuffer("");
  }
  /** ajoute un mot à la fin */
  public void append(String ajout) {
    if (ajout != null) {
      mot.append(ajout);
      this.setChanged();
      this.notifyObservers(mot.toString());
    }
  }
  /** insert un mot  */
  public void insert(int pos, String ajout) {
    if ((ajout != null)&&(pos >= 0)&&(pos < mot.length())) {
      mot.insert(pos, ajout);
      this.setChanged();
      this.notifyObservers(mot.toString());
    }
  }
  /** supprime une partie du mot*/
  public void cut(int pos, int nombre) {
    if ((pos >= 0)&&((pos + nombre -1)  < mot.length())) {
        mot.delete(pos, pos + nombre -1);
        this.setChanged();
        this.notifyObservers(mot.toString());
      }
  }
  /** supprime une lettre du mot*/
  public void deleteChar(int pos) {
    if ((pos >= 0)&&(pos < mot.length())) {
        mot.deleteCharAt(pos);
        this.setChanged();
        this.notifyObservers(mot.toString());
      }
  }
  /** inverse les lettres du mot*/
  public void reverse() {
      mot.reverse();
      this.setChanged();
      this.notifyObservers(mot.toString());
```

```
  }
  /** retourne le contenu du mot*/
  public String toString() {
    return mot.toString();
  }
}
```