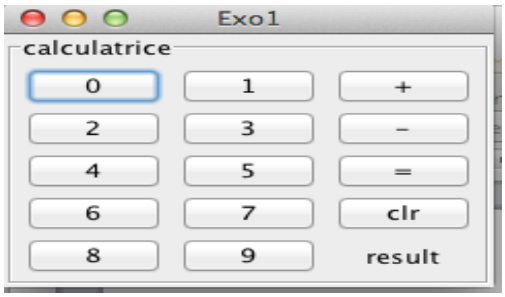
TD/TP PO2 Licence Informatique 3 (2016 – 2017)

Exercice 1

Ecrire une interface graphique qui contient :

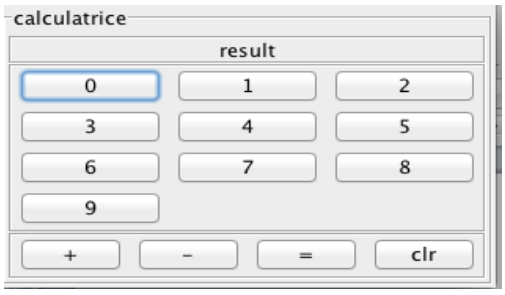
- 10 boutons correspondant aux chiffres : 0, 1, …,9
- 4 boutons correspondant aux opérations : + - = clr
- 1 label pour afficher le résultat



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

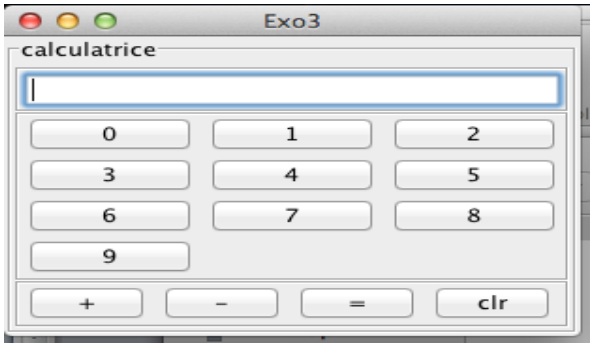
Idem avec 3 zones :

- 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 efface 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, …

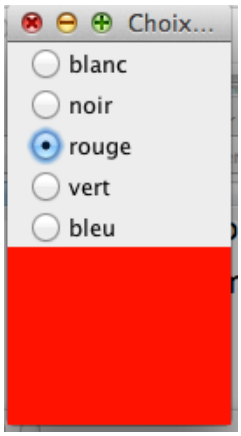


Fig. 1

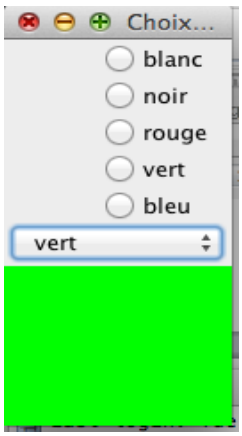


Fig. 2

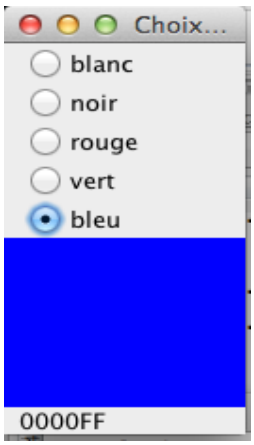


Fig. 3

Exercice 5

Afin d'étudier la relation qui pourrait exister entre l'âge et la pression sanguine, écrire une interface graphique permettant de saisir manuellement ces données, et de les représenter comme « nuage de points » dans un plan à deux dimensions.

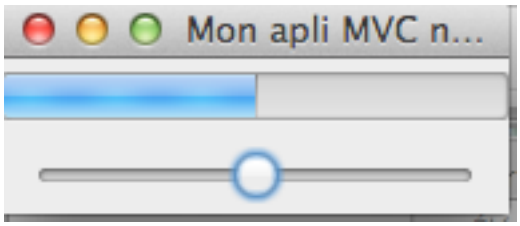
Voici un jeu de données venant d'un médecin qui mesure sur 12 femmes d'âges (x) différents la pression sanguine systolique (y).

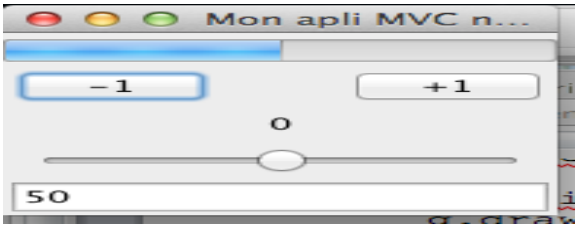
X (ans) :	56	42	72	36	63	47	55	49	38	42	68	60
Y (mm Hg) :	147	125	160	118	149	128	150	145	115	140	152	155

1. Saisir la donnée (x, y) d'un individu et l'afficher dans l'interface.
2. Proposer des solutions s'il y a des problèmes dans l'interface précédente.

Exercice 6

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));
```

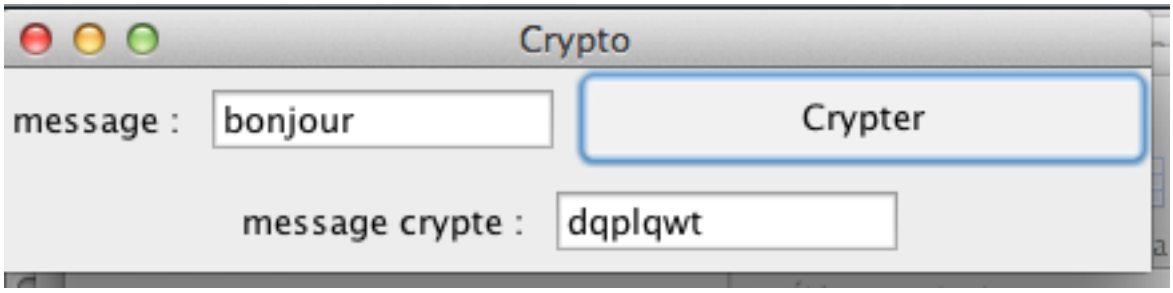
```
    panel.add(curseur);
    panel.add(tirette);
    this.getContentPane().add(panel);
    this.pack();
    this.setVisible(true);
  }

  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 7

L'objectif est de créer une interface permettant de crypter un message. Le principe de cryptographie utilisé ici est de remplacer chaque lettre d'un message par une autre à l'aide d'un tableau de chiffrement. Par exemple, un tel tableau avec le décalage circulaire de 2 :
- l1: abcdefghijklmnopqrstuvwxyz
- l2: cdefghijklmnopqrstuvwxyzab

Ainsi, le message « bonjour » sera crypté « dqplqwt ». L'interface est donnée au-dessous :



Concevoir une application avec MVC.