

T.P SWING



Pour Java et Swing

Découvrir :

Vous allez construire des applications informatiques fenêtrées réagissant aux actions utilisateur en vous appuyant sur l'existence de composants graphiques.




Cette expérience vous permettra, en outre, de bien assimiler la notion d'interface (au sens objet).

Réaliser :

- Étudiez les chapitres suivants de la ressource "*interface graphique avec Swing*" :
 - > bibliothèque graphique de l'API,
 - > architecture d'une application Swing,
 - > les composants *SWING*,
 - > agencement des contrôles,
 - > règles de conception de l'interface utilisateur.
- Comprendre le modèle *M.V.C.* avec Swing

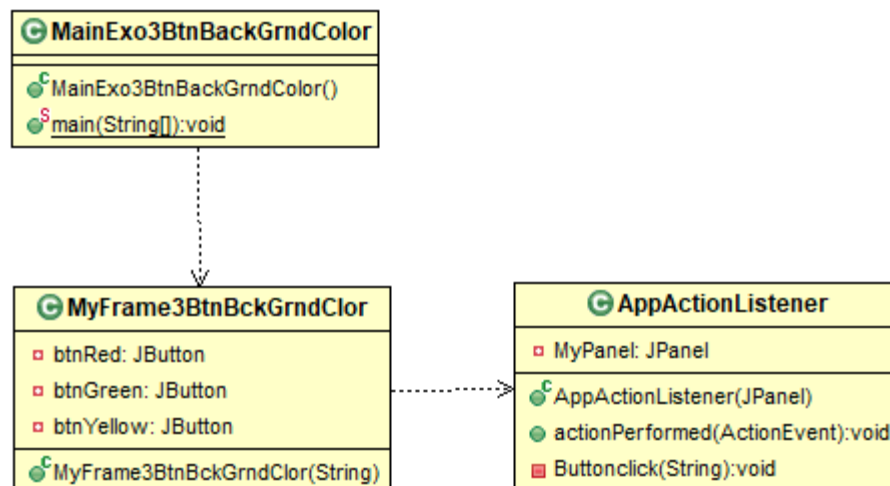
exo 1: changer la couleur de fond

le top: utiliser 3 classes pour une plus grande modularité

- ▷  ActionListener.java
- ▷  MainExo3BtnBackGrndColor.java
- ▷  MyFrame3BtnBckGrndClor.java

stand alone application correspondante:exo3BtnBackGrndColor.jar

conseil: inspirez vous du code présent dans le support de cours correspondant



- **Exo 2: objet à 2 états**

- Exo 2.1: radio button :

Alternier affichage de 2 images avec un radio button
au démarrage du programme l'image 'bird' est chargée
la frame courante implémente le listener `ItemListener`
les étapes:

```
//dans un JPanel
//Create the radio buttons.
//Group the radio buttons
//Register a listener for the radio buttons
birdButton.addItemListener(this);
// ItemListener pourrait être remplacé par un action listener puisque
//l'exo utilise 2 images (ce qui ne sera pas le cas dans exo suivant)
//les image sont transformés en icône
imgIconBird = new ImageIcon(RadioButtonExo.class.getResource("img/Bird.gif"));
//par défaut l'image bird est chargé
LblPicture = new JLabel(imgIconBird);

//coder le changement d'item (click sur un radio) dans la méthode fournie
//par l'interface
LblPicture.setIcon(imgIconCat);
```



```
//pour la disposition des objets graphiques dans la JFrame
JPanel radioPanel = new JPanel(new GridLayout(0, 1));
radioPanel.add(birdButton);
radioPanel.add(catButton);
add(radioPanel, BorderLayout.WEST);
add(LblPicture, BorderLayout.EAST);
```

• Exo 2 : objet à 2 états

- Exo2.2: GeekCheckBox

Using Swing Component: How to Use Check Boxes

The [JCheckBox](#) class provides support for check box buttons. Because JCheckBox inherit from AbstractButton, Swing check boxes have all the usual button characteristics, For example, you can specify images to be used in check boxes.

Check boxes are similar to radio buttons but their selection model is different, by convention. Any number of check boxes in a group — none, some, or all — can be selected.



Utiliser obligatoirement l'interface ItemListener car

il fournit la methode itemStateChanged pour les objets avec 2 etats (coche , decoche ex : checkbox)

JcheckBox implements **implements ItemListener**

```
public void itemStateChanged(ItemEvent e) {  
    ...  
    Object source = e.getItemSelectable();  
  
    if (source == glassesButton) {  
        //.....  
    } else if (source == hairButton) {  
        //.....  
    } else if (source == teethButton) {  
        //.....  
    }  
  
    if (e.getStateChange() == ItemEvent.DESELECTED)  
        //.....  
    ...  
    updatePicture();  
}
```

La stand alone application correspondante: exoGeekCheckBox.jar

exo 3: Ajouter et supprimer des Items d'une liste

- Comprendre le modèle *M.V.C.* avec Swing

Dans une liste insérer et supprimer des éléments provenant d'une source de données

Ajout ou retrait d'éléments

La liste est une instance de la classe `DefaultListModel`.

Enlever/supprimer un élément de la liste aura un effet sur le contrôle `JList` associé.

Création de la liste et de son modèle. Le modèle est connecté à la vue: l'ajout d'un élément au modèle sera visualisé dans la liste:

```
listModel = new DefaultListModel<Personne>();
```

Comme la liste est une vue du modèle.

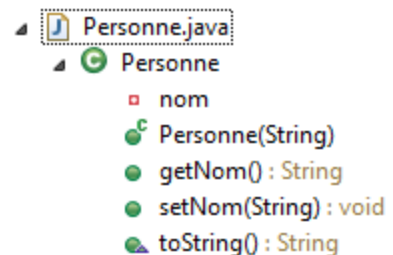
associer le modèle à la liste avec le code ci dessous:

```
list = new JList<Personne>(listModel)
```

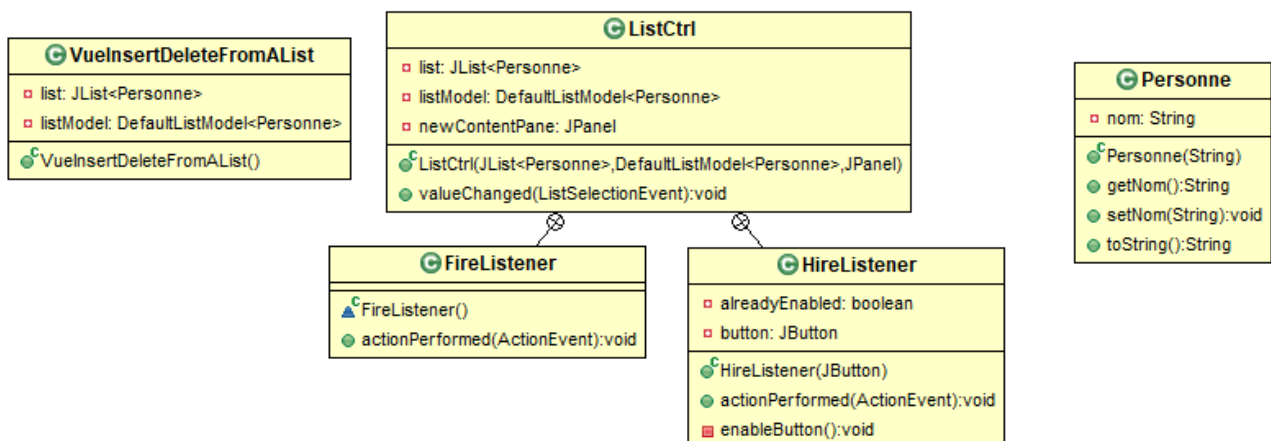
coder 3 objets de type `Personne` dans la liste pour éviter une liste vide au départ

```
Personne pers1 = new Personne("Jane, Doe");
```

implémentation du modèle: la classe `Personne`



remarque: l'ajout d'objets de type `Personne` à la liste se fait dans le contrôleur
respecter le pattern M.V.C ci dessous de gauche à droite: vue, contrôleur, modèle



stand alone application correspondante: `exoInsertDeleteFromAList.jar`

exo 4: Peindre & repeindre

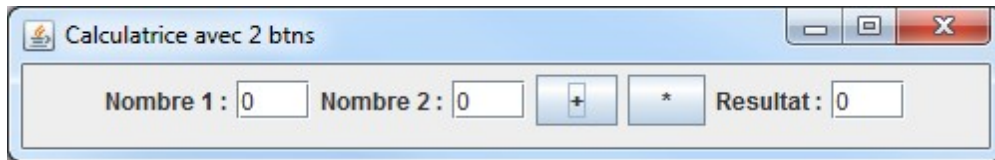
L'application permet de dessiner un polygone

L'utilisateur clique dans la fenêtre pour définir les points qu'il souhaite placer comme sommets des segments la figure.

*stand alone application correspondante : **exoDrawLinePtClick.jar***

exo 5 une calculatrice graphique en plusieurs étapes.

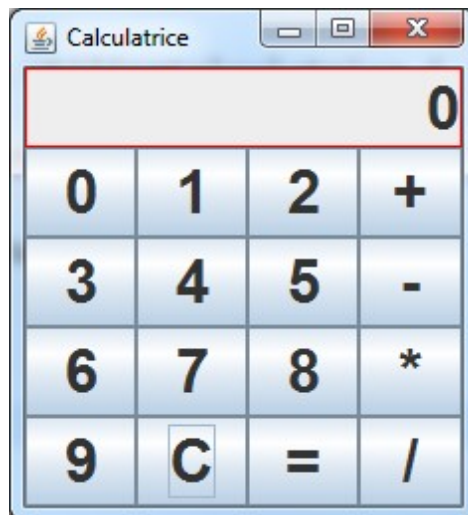
Etape 1: création d'une calculatrice simple capable d'opérations (* +) sur 2 nombres, le résultat s'affiche dans la fenêtre correspondante.



stand alone application correspondante : calculatrice2Btns.jar

Etape 2: Calculatrice réservée aux entiers relatifs

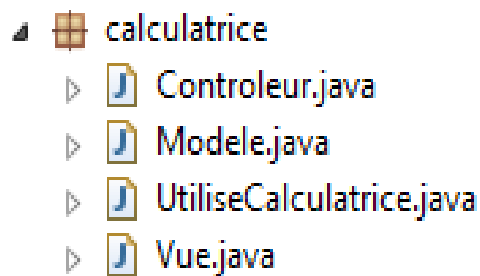
A l'aide des gestionnaires de disposition, nous allons améliorer son apparence..



Gestion des exceptions :

Essaie de diviser un nombre par zéro à l'aide de notre calculatrice – le champ textuel affiche un signe ∞ qui veut dire "infini". Modifiez le code pour afficher le message "Impossible de diviser par zéro" si l'utilisateur tente une division par zéro.

Respecter le pattern M.V.C

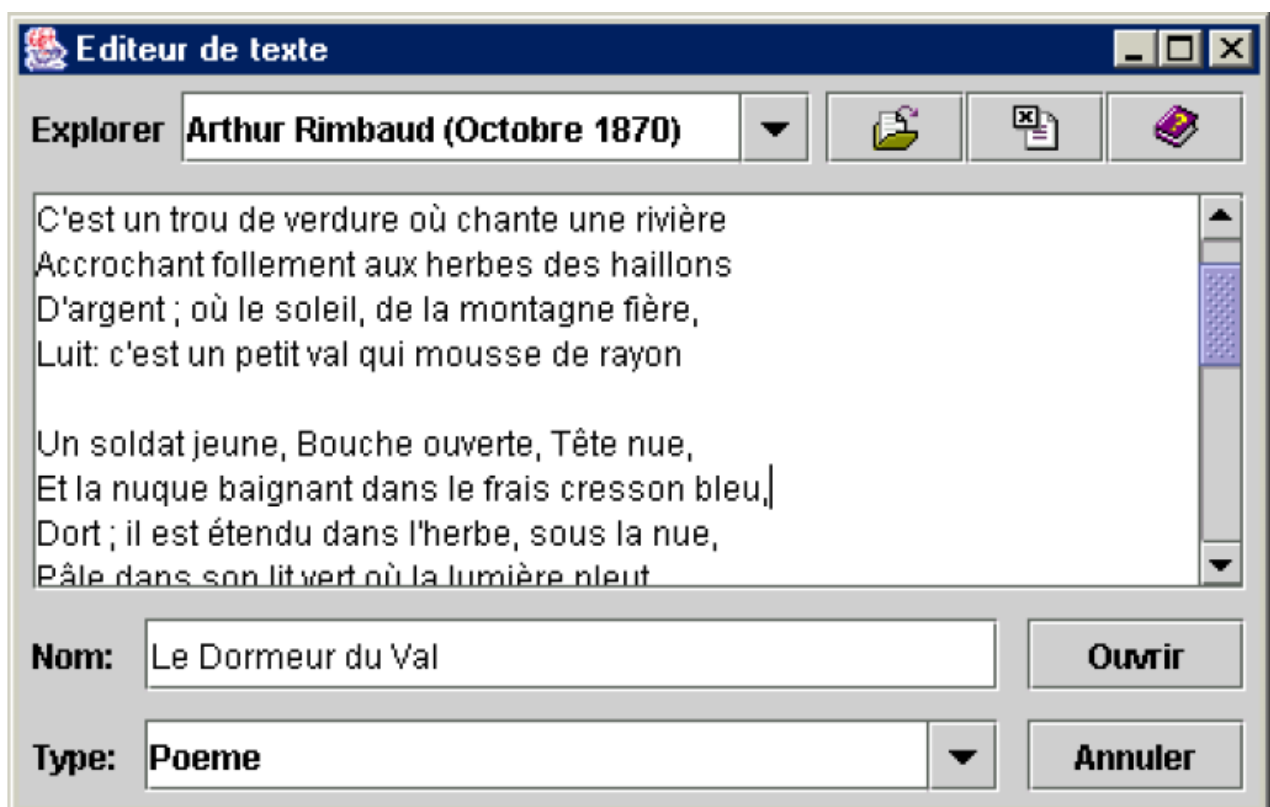
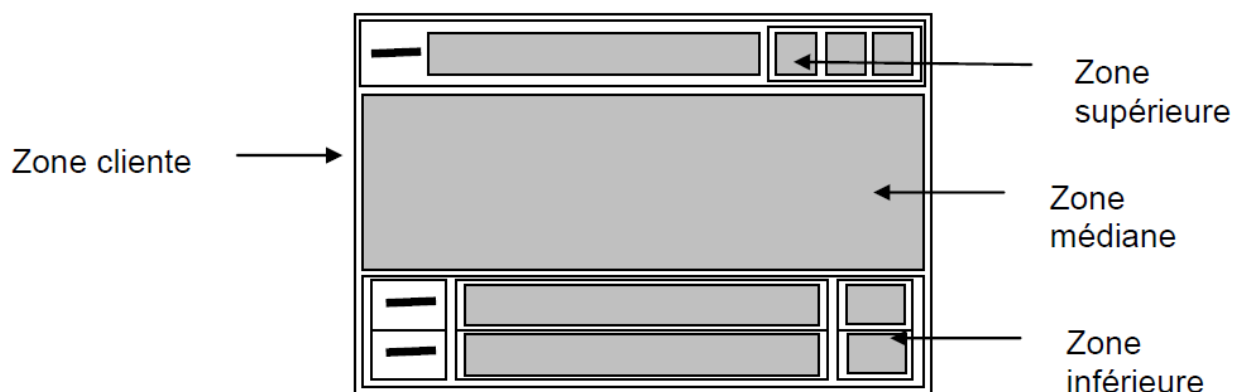


stand alone app : calcMVC.jar

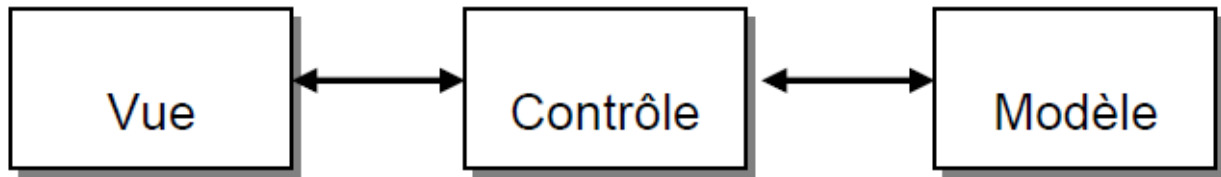
Exercice gestion de la disposition des objets graphiques (voire support AFPA dans le dossier InterfaceGraphique)

Ecrire une application ayant l'interface utilisateur décrit ci-dessous. La fenêtre application comporte 3 zones:

- La partie supérieure de hauteur constante comprend un label, une boîte combo qui occupe le maximum de largeur et une barre d'outils à 3 boutons graphiques
- La partie médiane est une zone de texte multi-lignes qui occupe le maximum d'espace
- La partie inférieure est constituée elle-même de 3 zones, la zone gauche avec 2 labels, la zone droite avec 2 boutons et la zone centrale qui occupe le maximum de largeur.



M.V.C (présentation succincte)



Chacun de ces composants tient un rôle bien défini.

La vue correspond à l'interface avec laquelle l'utilisateur interagit. Dans notre cas il s'agit d'une ou plusieurs classes UI utilisant des composants Swing. Aucun traitement n'est effectué dans la vue (hormis ceux correspondant à la "mécanique" de l'interface) elle sert uniquement à afficher les données et permettre à l'utilisateur d'agir sur ces données.

Le deuxième composant, le modèle, représente les données et les règles métier. C'est là que s'effectuent les traitements. Les bases de données en font partie, de même que des objets métier. Les données renvoyées par le modèle sont indépendantes de la présentation, c'est-à-dire que le modèle ne réalise aucune mise en forme. Les données d'un seul modèle peuvent ainsi être affichées dans plusieurs vues.

Enfin, le contrôleur interprète les requêtes de l'utilisateur et appelle les méthodes du modèle et de la vue, nécessaires pour répondre à la requête. Ainsi, lorsque l'utilisateur clique sur un contrôle, le contrôleur intercepte la requête et détermine quelle portion du modèle et quelle portion de la ou des vues doivent être associées.

Le Contrôleur décide également quelle est la prochaine vue à présenter au client en fonction de sa dernière action et des résultats des actions correspondantes au niveau du Modèle.

exemple de 2 vues partageant le même modèle:

exécutable: demoMVC2VuesTable.jar

code source: dossier demoMVC2VuesTableSrc