Swing

Sujet de Travaux Pratiques

Table des matières

1. Chapitre 2 : Premiers pas

1.1. Exercice n°1

- Créez une classe `TestFenetre` avec un `main` identique à l'exemple précédent.

```
public class TestFenetre {
 public static void main(String[] args){
                JFrame fenetre = new JFrame();
                fenetre.setVisible(true);
        }
}
```

- Exécutez, que constatez-vous ?

1.2. Exercice n°2

- Modifiez la classe `TestFenetre`.

- La fenêtre doit désormais s'afficher au milieu de l'écran avec le titre « Une JFrame », de taille 200 x 200 et qui termine le programme à se fermeture.

1.3. Exercice n°3

- Ajoutez maintenant à la fenêtre de l'exercice précédent 2 boutons.

- Le 1er aura le libellé « Bouton 1 » le second « Bouton 2 ».

2. Chapitre 3 : Gestionnaires de mise en forme

2.1. Exercice n°1

- Modifier l'exercice précédent.

- Faites en sorte que les boutons s'affichent de haut en bas, répartis avec un espacement fixe de 50px.

2.2. Exercice n°2

- Modifier l'exercice précédent.

- Faites en sorte que s'affiche au Nord le bouton 1 et au sud le bouton 2.

- Ajouter au centre un libellé (`JLabel`) « Ceci est un test ».

2.3. Exercice n°3

- Modifier l'exercice précédent.

- Placez un `Layout` de type `GridLayout` sur le panel principal.

Le `layout` sera composé de 2 lignes et 2 colonnes.

- Faites en sorte que s'affiche

  – à la première ligne première colonne le bouton 1

  – à la première ligne deuxième colonne le bouton 2

  – à la deuxième ligne première colonne le libellé.

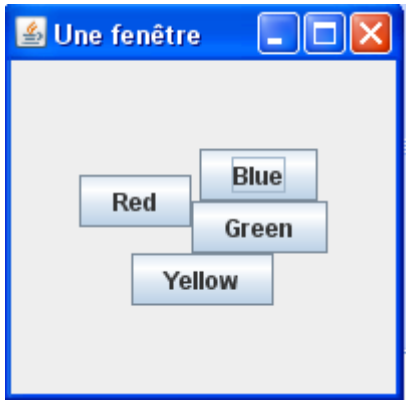
Le libellé ne dispose alors que d'au maximum la demie largeur de la fenêtre (100 px moins la bordure de la fenêtre).

- Modifiez le libellé afin que celui si soit « Ceci est un test de longueur ». Que constatez vous ?

2.4. Exercice n°4

- Reproduisez l'exemple précédent en considérant que chaque composant est un bouton.

Vous devez constater que les boutons se positionnent correctement mais qu'ils ne prennent pas toute la surface qu'ils ont à leur disposition :



2.5. Exercice n°5

- Reprenez l'exercice afin d'obtenir le résultat souhaité initialement.

- Chaque bouton doit occuper toute la place qui lui est offerte.

- Modifier ensuite le premier exercice composé de 2 boutons et un libellé.

- Placer maintenant le libellé sur toute la largeur de la 2ème ligne.

Le libellé à maintenant assez de place pour afficher tout son texte.

2.6. Exercice n°6

- Placez 3 `JLabel` dans un `CardLayout`.

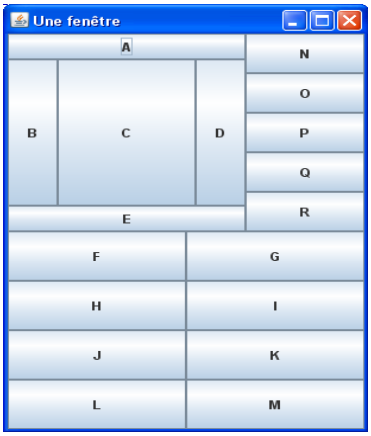
- Toutes les 5 secondes, passez au libellé suivant.

Note : Pour faire une pause de 5 secondes, vous pourrez utiliser la méthode `sleep` de la classe `Thread` comme suit :

```
try {
      Thread.sleep(5000);
} catch (InterruptedException e) {
      e.printStackTrace();
}
```

2.7. Exercice n°7 [Groupe en avance]

- Combinez les panneaux et les `layouts` pour réaliser l'exemple suivant :



Vous procéderez à un auto-dimensionnement de la fenêtre pour respecter les tailles imposées.

3. Chapitre 3.1 : Gestionnaires de mise en forme personnalisés

3.1. Exercice n°1

- Sur la base de l'exemple du `TwoPanesLayout` fourni, mettez en œuvre un `FourPanesLayout` composé de 4 zones :

- FourPanesLayout.TOPLEFT
- FourPanesLayout.TOPRIGHT
- FourPanesLayout.BOTTOMLEFT
- FourPanesLayout.BOTTOMRIGHT

4. Chapitre 4 : Look And Feel

4.1. Exercice n°1

- Reprenez l'exercice avec les 2 boutons et le libellé (Chapitre 3 : Exercice 2).

- Afficher les composants avec le look and feel de votre système d'exploitation.

4.2. Exercice n°2

- Reprenez l'exercice avec les 2 boutons et le libellé (Chapitre 3 : Exercice 2).

- Modifier la couleur de fond par défaut des boutons dans le look and feel actuel (propriété `Button.background`).

5. Chapitre 5 : Gestion événementielle

5.1. Exercice n°1

- Créez les classes `Pompier` et `Urgence` avec le code suivant :

```
public class Pompier {
        private String nomPompier;

        public Pompier(String nom){
                this.nomPompier=nom;
        }
        public void intervientSurUrgence(){
                System.out.println("Je m'appelle "+nomPompier+"
et j'intervient sur l'urgence");
        }
}


public class Urgence {

        private List<Pompier> lesPompiersDeService=new ArrayList<Pompier>();

        public void ajoutePompier(Pompier p){
                lesPompiersDeService.add(p);
        }

        public void caUrge(){
                for (Pompier p : lesPompiersDeService){
                        p.intervientSurUrgence();
                }
        }
}
```

- Créez une classe `SimulUrgence` avec un main qui instancie une urgence et plusieurs pompiers.

- Dans une classe avec un `main()`, abonnez les pompiers à l'urgence puis déclenchez l'urgence via la méthode `caUrge` de l'urgence.

5.2. Exercice n°2

- Reprenez les classes précédentes.

- La classe `SimulUrgence` doit maintenant déclenchez une urgence « incendie » puis une urgence « accident ».

5.3. Exercice n°3

- Créez la classe `UrgenceEvent` pour l'événement et l'interface `UrgenceListener` pour le listener.

- La classe `UrgenceEvent` aura la propriété `nature` de type `String` indiquant la nature de l'événement. Cette classe aura bien évidemment un constructeur ayant en paramètre la source de l'événement, et un

getter/setter pour la nature.

- L'interface `UrgenceListener` héritée de `EventListener` aura la méthode `urgenceSurvenue` prenant en paramètre un `UrgenceEvent`.

- La classe `Pompier` doit désormais hériter de `UrgenceListener`.

- Ajoutez à la classe `Urgence` une méthode de désabonnement d'un Pompier qui prend en paramètre le nom du pompier à désabonner.

- Modifiez `SimulUrgence` et déclenchez successivement une urgence incendie puis une urgence accident.

5.4. Exercice n°4

- Reprendre l'exercice de la fenêtre à deux boutons et un libellé (Chapitre 3 : Exercice 2).

Faites en sorte que le texte du `JLabel` change lorsque l'on appuie sur le bouton 1.

5.5. Exercice n°5

- Faites en sorte qu'un click sur le 2ème bouton déclenche une pause de 30 secondes (pour simuler une opération longue).

```
try {
        Thread.sleep(30000);
} catch (InterruptedException e) {
        e.printStackTrace();
}
```

- Pendant ce temps, essayer de cliquer sur le 1er bouton.

5.6. Exercice n°6

Améliorez l'exercice en introduisant le traitement long dans un nouveau Thread, conformément à l'exemple suivant :

```
Runnable r=new Runnable() {
        public void run() {
                try {
                        Thread.sleep(30000);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
};

Thread t=new Thread(r);
t.start();
```

- Cliquer sur le 2ème bouton puis sur le 1er.

L'exécution de la section longue ne vous empêche plus de cliquer sur le 1er bouton.

5.7. Exercice n°7

- Faites en sorte que dans l'exercice précédent, le click sur le 2ème permette également de modifier le texte du libellé « Traitement long terminé ».

- Appuyer sur le bouton 2 puis sur le bouton 1.

5.8. Exercice n°8

- Adaptez l'exercice précédent afin d'effectuer toutes les modifications graphiques au sein de l'Event Dispatch Thread.

5.9. Exercice n°9

- Adaptez l'exercice précédent afin d'utiliser un `SwingWorker`.

5.10. Exercice n°10

- A la manière de l'exemple du cours, modifier le libellé central durant la progression de l'opération longue. Chaque progression de 10% doit être annoncée « `X% du traitement long` ».

5.11. Exercice n°11

- Abonnez le bouton 2 à un événement souris qui permet, lorsque l'utilisateur survole le bouton, de modifier le libellé du bouton par « Déclenche une action longue ».

6. Chapitre 6 : Composants basiques

6.1. Exercice n°1

- Reprendre l'exercice de la fenêtre à deux boutons et un libellé (Chapitre 3 : Exercice 2).

- Faites en sorte que le texte du `JLabel` soit précédé de l'icône eye.png fourni.

6.2. Exercice n°2

- Créer une classe `TestOrderForm`.

Cette classe doit produire une fenêtre contenant

- un `JTextField` permettant de saisir un login,
- un `JPassword` permettant de saisir un mot de passe
- un bouton OK et
- un bouton Reset.

- Sur le click du bouton OK, vérifiez que le login vaut 'toto' et le mot de passe 'admin'.

- Si oui, affichez dans la console standard « Order OK », sinon affichez «Order failed ».

- Sur le click du bouton Reset, remettez les champs à vide.

6.3. Exercice n°3

- Modifier la classe `TestOrderForm`.

- Ajouter un `JFormattedTextField` permettant par exemple de saisir une date.

Premier constat, le format de saisie de la date est très contraignant. Impossible de saisir un format court.

Deuxième constat, le champ attend une date au format français, ce n'est peut-être pas le comportement attendu.

6.4. Exercice n°4

- Modifier la classe `TestOrderForm`.

- Faites en sorte de pouvoir saisir la date au format court MM/AA.

6.5. Exercice n°5

- Modifier la classe `TestOrderForm`.

- Ajoutez-y un champ de saisie permettant d'indiquer de saisir un numéro de carte de crédit (espaces inclus).

6.6. Exercice n°6

- Reprenez l'exercice qui affiche 2 boutons et un libelle (Chapitre 3 : Exercice 2).

- Ajoutez la possibilité de faire un click droit sur le libellé pour afficher un `JPopupMenu` avec un item (« Reset »).

Lorsque l'utilisateur choisit cet item, le libellé doit revenir à son texte d'origine.

6.7. Exercice n°7

- Reprenez l'exercice précédent et placez les 2 boutons dans une barre d'outils.

- Tentez de déplacer la barre d'outils hors de la fenêtre.

6.8. Exercice n°8

- Reprenez l'exercice précédent et fusionner les 2 `Listeners`.

6.9. Exercice n°9

- Reprenez l'exercice précédent et affectez 1 instance d'Action à chaque bouton.

6.10. Exercice n°10

- Traduisez tous les éléments texte de notre exercice en anglais et en français.

- Ajouter un bouton qui déclenche le renommage des libellés en anglais et un bouton qui déclenche le renommage en français.

7. Chapitre 7 : Composants évolués

7.1. Exercice n°1

- Créez une classe `TestCombo` qui affiche via le main un menu déroulant.

- Ce menu contient un ensemble d'objets `Stagiaire` ayant les propriétés nom, prénom et société.

Ne seront affichés que les noms et les prénoms.

L'élément sélectionné sera en blanc sur fond rouge et précédé de l'icône dont nous nous sommes déjà servi précédemment.

Ajoutez un bouton « supprimer ».

Ce bouton permettra de supprimer de la liste l'élément sélectionné.

7.2. Exerckce n°2

- Créez une classe `TestList` qui propose la même fonctionnalité que `TestCombo`.

- Le panneau principal doit pouvoir gérer le défilement.

Lorsque vous aurez démarré l'application, redimensionnez manuellement la taille de la fenêtre pour vous assurer de la capacité de défilement.

7.3. Exercice n°3

- Créez une classe `Loto` qui affiche un tableau permettant de lister les 49 numéros du loto (`Integer`) sur 10 colonnes et 5 lignes.

7.4. Exercice n°4

- Modifiez la classe `Loto` de manière à ce que tous le numéros se situent dans une seule colonne. Vous ajouterez une deuxième colonne que vous laisserez vide pour l'instant.

La première colonne doit avoir comme titre « Numéro », la seconde « Cocher ».

Créez une classe `NumeroLoto` qui encapsule l'information du numéro et affecter des instance de `NumeroLoto` au modèle.

7.5. Exercice n°5

- Modifiez la classe `Loto` de manière à ce que la deuxième colonne affiche une case à cocher indiquant si le numéro à été sélectionné par le joueur ou non.

5 de ces numéros doivent apparaître sélectionnés.

Note : Vous devrez donc rajouter un attribut à `NumeroLoto`.

7.6. Exercice n°6

- Modifiez la classe `Loto` et faites en sorte que, par le biais d'un « renderer », le numéro s'affiche dans un `JLabel` et systématiquement précédé de « n° ».

7.7. Exercice n°7

- Modifiez la classe `Loto` et faites en sorte que, par le biais d'un « editor », les numéros puissent être sélectionnés à l'aide d'une case à cochée.

Attention, il s'agit de la case à cochée de l' « éditor », pas celle du « renderer ». Nous verrons juste après comment réutiliser celle du « renderer ».

Attention : Encore une fois, la fin de l'édition est reconnue par Swing comme un changement de focus ou la touche entrée.

Or dans le cas d'un `Checkbox`, l'utilisateur peut très bien cocher à la suite plusieurs cases, le changement de focus correspond donc également l'édition d'une autre case à cocher. Des effets de bords vont donc se produire, la méthode `getCellEditorValue` pouvant être appelée après l'édition de la donnée de la case à cocher (Faites l'expérience).

Pour palier au problème, il va falloir explicitement déclencher l'événement de fin d'édition sur changement de valeur de la case à cocher. (plus besoin en 16_018 ou 17)

```
private JCheckBox tfEnCours = new JCheckBox();

public LotoEditor() {
	ItemListener itemListener = new ItemListener() {
		public void itemStateChanged(ItemEvent itemEvent) {
			fireEditingStopped();
		}
	};
	tfEnCours.addItemListener(itemListener);
}
```

7.8. Exercice n°8

- Modifiez la classe `Loto` et ajoutez un bouton qui permette l'ajout d'une nouvelle ligne ayant comme numéro le dernier numéro + 1.

7.9. Exercice n°9

- Dans une classe `TestArbre` reprenez la liste des stagiaires que vous afficherez sous forme d'arbre, chaque stagiaire étant au même niveau sous la racine.

- La racine aura le libellé « Cours Swing ».

- On affichera la racine accompagnée de son maneton.

- Vous réaliserez votre propre modèle bénéficiant d'une méthode `addStagiaire(Stagiaire set)`.

7.10. Exercice n°10

- Modifier `TestArbre`.

Faites en sorte que chaque nœud « stagiaire » dispose de 2 enfants ayant le libellé :

- Nom : <nom du stagiaire>
- Prénom : <prénom du stagiaire>

Ces deux nœuds doivent être éditables.

Vous devez cependant constater 2 effets indésirables. Premièrement un simple click suffit pour éditer le nœud, comportement inhabituel.

Deuxièmement, vous avez probablement placé dans tous les nœuds (hormis la racine), un objet de type Stagiaire.

Aussi le nœud parent à nom / prénom est également éditable.

- C'est en surchargeant la méthode `isCellEditable` que l'on va résoudre les 2 problèmes :

```
@Override
public boolean isCellEditable(EventObject evt) {

        JTree tree=(JTree)evt.getSource();

        if(evt instanceof MouseEvent){
                MouseEvent mevt = (MouseEvent) evt;

                if (mevt.getClickCount()==1) return false;

                TreePath path=tree.getPathForLocation(mevt.getX(), mevt.getY());
                DefaultMutableTreeNode lastNode=(DefaultMutableTreeNode)
                                path.getLastPathComponent();
                if (lastNode.isLeaf()) return true;
        }
        return false;
}
```

8. Chapitre 7.1 : Autres composants

8.1. Exercice n°1

- Créer une classe `OpenDialog` avec un `main()`.

- La classe devra ouvrir une `JFrame` avec un bouton « Ouvrir la boîte de dialogue ».

- Le click sur le bouton ouvrira la boite de dialogue ci-dessous.

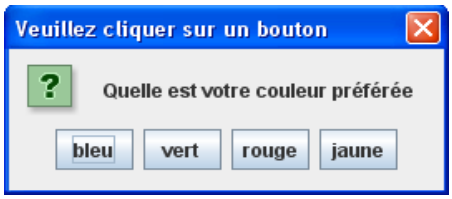
- Le bouton « OK » doit fermer « proprement » la boîte de dialogue.

8.2. Exercice n°2

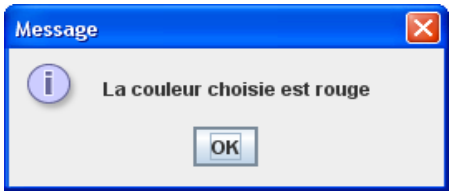
- Reprendre l'exercice précédent en utilisant la méthode `showMessageDialog`.

8.3. Exercice n°3

- Reproduire la boite de dialogue suivante à l'aide de `showOptionDialog`



Suivi de :



8.4. Exercice n°4

- Créer une frame avec un bouton. Au clic, ouvrir le `JColorChooser` et récupérer la couleur sélectionnée (par défaut, la couleur est rouge).

Lorsque la couleur est choisie, modifier la couleur du bouton qui a permis l'ouverture de la boite de dialogue.

9. Chapitre 8 : Drag and Drop

9.1. Exercice n°1

- Créez un classe `TestDnD` qui affiche 2 `JTextField`.

- L'utilisateur pourra effectuer du DnD du premier `JTextField` au second mais pas l'inverse.

Ajouter également la possibilité d'effectuer un DnD entre 2 `JColorChooser`.

9.2. Exercice n°2

- Modifiez la classe `TestDnD` afin de permettre également le DnD entre 2 `JColorChooser`.

9.3. Exercice n°3

- Modifiez la classe `TestDnD` afin que seule la copie de texte du premier `JTextField` vers le second soit autorisée.

9.4. Exercice n°4

- Modifiez la classe `TestDnD` afin que soit possible la copie à partir d'un `JcolorChooser` vers tous les autres composants.

9.5. Exercice n°5

- Créer une classe `ColorScanner` qui affiche un `JColorChooser` et 3 `JTextField`.

- L'utilisateur doit pouvoir déposer une couleur dans n'importe quel champ texte.

  – Le 1er champ doit afficher la composante rouge de la couleur : `getRed()`

  – Le 2ème champ doit afficher la composante verte de la couleur : `getGreen()`

  – Le 3ème champ doit afficher la composante bleue de la couleur : `getBlue()`

9.6. Exercice n°6

- Créer une classe `TreeDnD` qui affiche un arbre dans lequel chaque nœud peut être déplacé entre 2 autres nœuds existants situés au même niveau (même parent).

Note : Il s'agit à peu de choses près du code de l'exemple.

10. Chapitre 9 : Le binding - JGoodies

10.1. Exercice n°1

- Créez un Bean « `Car` » ayant 3 propriétés :

  – nbPortes – `int`,

  – couleur - `String` et

  – mmatriculation – `String`.

- Ajoutez un `Listener` capable de notifier le changement de l'attribut immatriculation.

Le `Listener` devra afficher dans la console les modifications effectuées (valeur précédente, nouvelle valeur).

Dans un programme principal (`main`), vous instancierez `Car` et vous mettrez à jour la propriété écoutée.

10.2. Exercice n°2

- Utilisez la librairie de Binding de `JGoodies` afin de mettre en place le Binding entre l'objet métier « `Car` » et un formulaire d'édition graphique.

- Vérifiez que le Binding fonctionne dans les 2 sens:

  – Toute mise à jour directe de l'objet métier est immédiatement reportée dans l'interface graphique

  – Tout changement dans l'interface graphique est immédiatement reportée sur le modèle métier.

11. Chapitre 9.2 : Améliorer un composant existant

11.1. Exercice n°1

- Créez une case à cocher personnalisée

- Créez 2 classes UI chargées de dessiner la case à cocher :
  - L'une dessinant un carré qui passe alternativement du vert et au rouge lorsqu'il est cliqué.
  - L'autre dessinant un rond qui passe alternativement du vert au rouge lorsqu'il est cliqué.

- Créez 2 look and feel :
  - Style « rond » qui utilisera automatiquement l'UI dessinant votre `checkbox` personnalisée ronde.
  - Style « carré » qui utilisera automatiquement l'UI dessinant votre `checkbox` personnalisée carré.

- Testez successivement sur les 2 look and feel.

12. Chapitre 10 : Déploiement

12.1. Exercice n°1

Nous allons décrire une de nos précédente applications dans un fichier .jnlp, puis l'exécuter à l'aide de Java Web Start.

- Il faudra avant tout créer une librairie contenant les classes de notre application.

- Créez ensuite le fichier jnlp.

Nous allons mettre en ligne notre application grâce à un serveur Tomcat disponible sur vos machines.

- Créez une application Web `testapp` contenant un la librairie, le jnlp et un fichier WEB-INF/web.xml (fourni).

- Le fichier jnlp devra donc déclarer un codebase : `http://localhost:8080/testapp`

- Déposez le répertoire dans « webapps » et démarrer le serveur.

- Ouvrez votre navigateur web et demandez la ressource : `http://localhost:8080/testapp/monfichierjnlp.jnlp`