Comprendre les principes de Swing
Corrigé

L'objectif de ces exercices est de comprendre les bases de Swing sur un exemple fil rouge. Ils sont à faire sur machine ! Le point de départ est la classe `ComprendreSwing`.

Exercice 1 : Les constituants
Commençons par comprendre cette application.
1.1. Lire le texte de la classe `ComprendreSwing` et expliquer les éléments qui la composent, en particulier (par ordre d'apparition dans le texte) :

1. `JPanel`
   Solution : C'est un conteneur qui permet de regrouper plusieurs composants.
2. `JLabel`
   Solution : Ce composant permet de visualiser un texte, une image ou les deux.
3. `JTextField`
   Solution : Ce composant correspond à une zone de saisie d'un texte sur une seule ligne.
4. `JButton`
   Solution : Ce composant correspond à un bouton sur lequel l'utilisateur pourra cliquer pour déclencher une action. Comme un JLabel, il contient un texte, une image ou les deux.
5. `setLayout`
   Solution : Cette méthode permet de positionner un gestionnaire de placement sur un conteneur (`Container`), par exemple un `JPanel`.
6. `FlowLayout`
   Solution : C'est un gestionnaire de placement qui place les différents composants les uns par rapport aux autres.
7. `addActionListener`
   Solution : Cette méthode permet d'associer une action à un bouton.
8. `addMouseListener`
   Solution : Cette méthode permet d'associer une réaction à des événements émis par un composant, par exemple un `JLabel`.
9. `ActionListener`
   Solution : C'est l'interface qui spécifie une action qui sera exécutée quand un bouton sera cliqué (actionné).
10. `actionPerformed`
    Solution : La méthode de l'interface `ActionListener`. C'est cette méthode qui sera exécutée quand le bouton sera cliqué.

11. `ActionEvent`

    Solution : Les informations sur l'événement qui s'est produit (clic sur un bouton), par exemple quel bouton (`getSource()`), les coordonnées de la souris, le bouton cliqué, la nature du clic (simple, double...)...

12. `MouseAdapter`

    Solution : L'interface qui modélise les événements qui correspondent aux déplacements de la souris vis à vis d'un composant Swing.

13. `mouseCliked`, `mouseEntered` et `mouseExited`

    Solution : Les événements en question.

14. `MouseEvent`

    Solution : Les informations liées au déplacement de la souris.

15. `JFrame`

    Solution : Ce composant Swing correspond à une fenêtre qui est gérée par le système de gestion de fenêtres du système d'exploitation.

16. `getContentPane`

    Solution : Méthode de `JFrame` qui donne accès au conteneur du volet principal de la fenêtre.

17. `add`

    Solution : Méthode qui permet d'ajouter un composant dans un conteneur.

18. `pack`

    Solution : Méthode de `JFrame` qui permet de calculer la dimension optimale de la fenêtre en fonction des composants qu'elle contient et des gestionnaires de placement associés aux conteneurs.

19. `setDefaultCloseOperation`

    Solution : Méthode de `JFrame` qui définit l'action à exécuter quand l'utilisateur quittera la fenêtre.

20. `JFrame.EXIT_ON_CLOSE`

    Solution : Valeur qui indique que l'action consiste à quitter le programme Java (appel à `System.exit()`).

21. `setLocation`

    Solution : Positionner la fenêtre.

22. `setVisible`

    Solution : Rendre la fenêtre visible.

23. `EventQueue`

    Solution : Le fil d'exécution (l'activité ou *thread*) associé à Swing, qui gère les interactions de l'utilisateur sur les composants Swing. C'est en particulier lui qui identifie le composant sur lequel l'utilisateur agit, crée l'événement correspondant et appelle la méthode `fireXListener` pour que tous les écouteurs inscrits soient notifiés.

24. `invokeLater`

    Solution : Méthode de classe de `EventQueue` qui permet de demander à l'activité `EventQueue` d'exécuter la méthode (en fait le `Runnable`) passé en paramètre.

1.2. Cette application crée deux fenêtres identiques (au titre près). Dessiner l'apparence que devrait avoir ces fenêtres.
Solution : Elle est donnée à la figure 1

Exercice 2 : Exécuter l'application
2.1. Exécuter `ComprendreSwing` pour vérifier si l'apparence des fenêtres est celle attendue.
Solution : À vous de vérifier !
2.2. Jouer avec cette application. Ne pas hésiter à déplacer et redimensionner les fenêtres.
Solution : Il faut le faire...
2.3. Le programme principal s'est terminé. Pourquoi l'application s'exécute-t-elle toujours ?
Solution : Car nous avons créé des composants Swing et avons donc activé le fil d'exécution (l'activité / *thread*) de Swing.

Exercice 3 : Le placement des composantes graphiques
On envisage plusieurs autres aspects pour la présentation de cette application. Ici, on s'intéresse à l'agencement des composants et pas à la barre haute de la fenêtre. Celle-ci dépend du système de gestion de fenêtres du système d'exploitation (Ubuntu/GNOME, Windows, MacOS, etc.).
3.1. Modifier l'apparence de l'application pour qu'elle corresponde à celle de la figure 1.

FIGURE 1 – Nouvelle apparence souhaitée pour l'application

Solution : C'est ce qu'on a déjà.
3.2. Modifier l'apparence de l'application pour qu'elle corresponde à celle de la figure 2.

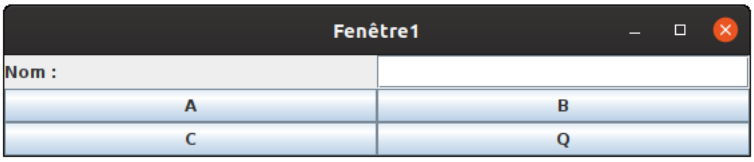

FIGURE 2 – Nouvelle apparence souhaitée pour l'application

Solution : On remplace FlowLayout par GridLayout(3, 2).
Voir la classe `ComprendreSwingGridLayout`.
3.3. Modifier l'apparence de l'application pour qu'elle corresponde à celle de la figure 3.
Solution : Plusieurs conteneurs (JPanel) : avec un FlowLayout au Nord pour les boutons A, B et C, un GridLayout au centre et un le bouton Q au Sud (dans un JPanel si on ne veut pas qu'il occupe toute la place).

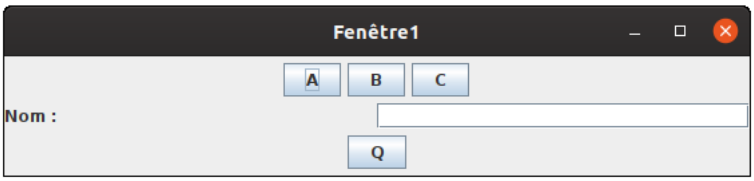

FIGURE 3 – Nouvelle apparence souhaitée pour l'application

```
JPanel -- BorderLayout
    |
    \--NORTH-- JPanel -- FlowLayout
    |            |
    |            \--- bA
    |            \--- bB
    |            \--- bC
    |
    \--CENTER-- JPanel -- GridLayout(1, 2)
    |            |
    |            \--- nomTxt
    |            \--- nom
    |
    \--SOUTH-- JPanel -- FlowLayout
                 |
                 \--- bQ
```

Voir la classe `ComprendreSwingBorderLayout`.

3.4. Modifier l'apparence de l'application pour qu'elle corresponde à celle de la figure 4.

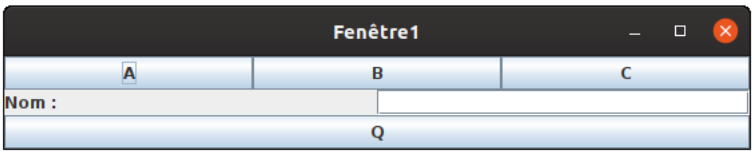

FIGURE 4 – Nouvelle apparence souhaitée pour l'application

Solution : Dans le JPanel du haut, on utilise un GridLayout(1, 3). En bas, on peut directement mettre le bouton.

```
JPanel -- BorderLayout
    |
    \--NORTH-- JPanel -- GridLayout(1, 3)
    |            |
    |            \--- bA
    |            \--- bB
    |            \--- bC
    |
    \--CENTER-- JPanel -- GridLayout(1, 2)
```

```
10      |            |
11      |            \--- nomTxt
12      |            \--- nom
13      |
14      \--SOUTH-- bQ
```

Voir la classe `ComprendreSwingBorderLayoutLarge`.

Exercice 4 : Aligner le texte des JLabel

Le texte « Nom : » est aligné à gauche. Ce n'est pas très esthétique dans les dernières apparences envisagées. Modifier l'application pour qu'il soit aligné à droite. On pourra utiliser la méthode `setHorizontalAlignment` de `JLabel`.

Solution : Il suffit d'ajouter l'instruction :

```
nomTxt.setHorizontalAlignment(JLabel.RIGHT);
```

Elle peut être ajoutée en début du constructeur ou dans un initialiseur (du code entre accolades au même niveau que les attributs, les constructeurs et méthodes) juste après la déclaration de l'attribut (c'est la solution retenue dans la solution proposée). Les initialiseurs sont exécutés après la réservation de la mémoire d'un objet et avant l'exécution du constructeur.

Exercice 5 : Fermer une fenêtre

L'application crée plusieurs fenêtres. Regardons ce qui se passe quand on ferme une fenêtre.

5.1. Que se passe-t-il si on ferme une fenêtre (par exemple en cliquant sur la croix en haut à droite sous Ubuntu/GNOME) ?

Solution : Le programme s'arrête !

5.2. Qu'est ce qui provoque ce comportement ?

Solution : C'est l'appel à `setDefaultCloseOperation(EXIT_ON_CLOSE)` qui arrête l'application quand la fenêtre est fermée.

5.3. Terminer l'application quand on ferme l'une de ses fenêtre n'est généralement pas souhaitable. Comment ne fermer que la fenêtre concernée et laisser l'application s'exécuter ? On pourra utiliser `DISPOSE_ON_CLOSE` plutôt que `EXIT_ON_CLOSE`.

Solution : Il suffit donc de remplacer `EXIT_ON_CLOSE` par `DISPOSE_ON_CLOSE`. On constate alors que l'application ne s'arrête que quand les deux fenêtres sont fermées.

Remarque : Si on n'avait pas utilisé `setDefaultCloseOperation` ou si on lui avait donné comme paramètre `HIDE_ON_CLOSE` (valeur par défaut), le programme ne se serait pas arrêté même après avoir « fermé » les deux fenêtres car elles auraient juste été cachées (rendues non visibles) : les fenêtres existant toujours, l'activité de Swing aurait continué à être active.

On peut le voir avec la classe `ComprendreSwingTimer` qui toutes les 5 secondes rend visible la fenêtre. On peut « fermer » la fenêtre mais elle réapparaitra (au plus tard dans les 5 secondes). Au passage, c'est une illustration de la classe `Timer` de Swing.

Exercice 6 : Le bouton « Q »

Considérons le bouton Q et l'effet qu'il a.

6.1. Que se passe-t-il quand on clique sur le bouton « Q » ?

Solution : Le bouton s'enfonce et le message « Quitter » s'affiche dans le terminal où l'application a été lancée.

6.2. Expliquer les mécanismes de Swing qui sont mis en œuvre.

Solution :

1. Définir un ActionListener et sa méthode actionPerformed (classe `ActionQuitter`).
2. Inscrire une instance de cet écouteur auprès du bouton concerné (méthode `addActionListener`).

6.3. Faire que l'application s'arrête quand on clique sur « Q ». On utilisera `System.exit()`.

Solution : On ajoute la ligne suivante à la fin de la méthode `actionPerformed` de la classe `ActionQuitter` :

```
System.exit(0);
```

Exercice 7 : Le bouton « C »

Faire que le texte de la zone de saisie (`JTextField`) soit effacé quand on clique sur le bouton « C ». On pourra utiliser la méthode `setText` de la classe `JTextField`.

Solution : Comme pour le bouton « Q », il s'agit de définir une classe qui réalise l'interface `ActionListener` et définir sa méthode `actionPerformed` puis d'en enregistrer une instance auprès du bouton « C ».

Voir la classe `ComprendreSwingEffacer`.

Exercice 8 : Listener et Adapter

Les gestionnaires d'événements de Swing (Listener) proposent souvent des « adaptateurs ». C'est par exemple le cas de `MouseListener` avec `MouseAdapter`.

8.1. Commençons par répondre à quelques questions rapides.

1. Combien y a-t-il de méthodes dans l'interface `MouseListener` ?
   Solution : 5
2. Combien y a-t-il de méthodes abstraites dans la classe `MouseAdapter` ?
   Solution : Aucune.
3. Combien de méthodes sont définies dans la classe `ActionTrace` ?
   Solution : 3

8.2. Pourquoi les « adaptateurs » sont-ils des classes abstraites ?

Solution : Car toutes les méthodes sont définies avec un code vide. Elle ne font donc rien t'intéressant. Il est donc inutile de pouvoir en créer des instances. Il faut en hériter et redéfinir certaines des méthodes.

8.3. Quel est l'intérêt des « adaptateurs » ?

Solution : Ils permettent de ne redéfinir que les méthodes qui nous intéressent (en laissant les autres avec leur code vide). Passer par l'interface nous obligerait à toutes les définir.

8.4. Quel est le risque des « adaptateurs » ?

Solution : Se tromper en redéfinissant une méthode. Il faut donc penser à mettre `@Override` pour éviter ce risque.

8.5. Qu'est ce qui devrait se passer quand on clique sur un bouton A ou B ?

Solution : On devrait avoir un message qui dit « Appui sur » suivi du nom du bouton.

Que se passe-t-il en réalité ?

Solution : Rien ne s'affiche !

Pourquoi ?

Solution : Parce qu'on a commis une faute de frappe : mouseCliked, sans le « c » (mouseClicked !).

Comment éviter ce type de problème ?

Solution : Ne pas oublier de mettre @Override quand on redéfinit une méthode.

8.6. Ajouter les @Override qu'il est conseillé de mettre systématiquement quand on définit ou redéfinit une méthode. Conclusion ?

Solution : Il faut vraiment penser à les mettre !

Exercice 9 : Le bouton Q (suite)

Faire en sorte que le bouton Q ferme la fenêtre mais pas l'application. Ceci signifie qu'il ne faut plus utiliser `System.exit` mais appeler la méthode `dispose()` de la fenêtre (`JFrame`).

Solution : Il faut donc avoir accès à la fenêtre. Elle n'est pas directement accessible du JPanel ComprendreSwing. Nous allons donc la transmettre au constructeur et la conserver dans un attribut pour pouvoir ensuite l'utiliser.

Voir la classe `ComprendreSwingComplet`.