Programmation Orientée Objet
IG2I, LE3 – 2024-2025

TP2 – Interfaces graphique avancée

Maxime Ogier

Contexte et objectif du TP

Ce TP est la suite du TP 1, et il propose de mettre en place une interface graphique pour la gestion de la bibliothèque de Triffouilly-les-Oies. Les fonctionnalités principales de cette interfaces seront les suivantes :

- lister les livres ;
- savoir si un livre est actuellement emprunté ou non ;
- rechercher un livre ;
- rendre un livre ;
- emprunter un livre.

Ce TP permet ainsi d'illustrer les notions suivantes :

- l'utilisation des requêtes développées lors du TP 1 ;
- l'API Swing ;
- l'utilisation de différents objets graphique : JFrame, JPanel, JLabel, JList, JTextArea, JButton, JComboBox ;
- l'utilisation avancée des objets pour le rendu visuel des objets graphiques ;
- les gestionnaires de mise en page ;
- les écouteurs et la programmation événementielle.

Bonnes pratiques à adopter

- Testez votre code au fur et à mesure de votre avancement.
- Respectez les conventions de nommage Java.
- Faites des indentations de manière correcte.
- Donnez des noms compréhensibles et pertinents à vos variables, attributs, classes, méthodes.
- Respectez autant que possible le principe d'encapsulation.
- Utilisez des structures de données adaptées.
- Faites des méthodes courtes avec peu de niveaux d'indentation.
- Ne faites pas de duplication de code, mais privilégiez le code réutilisable.
- Utilisez les outils de votre IDE pour faire du *refactoring* (clic-droit puis '*Refactor*' sur IntelliJ IDEA).
- Utilisez la Javadoc pour comprendre le fonctionnement des méthodes que vous appelez, en particulier celles de l'API JDBC.
- Quand c'est nécessaire, commentez votre propre code de façon pertinente au format Javadoc (commentaire commençant par `"/**"` au-dessus des définitions des attributs et méthodes).

1 Mise en place de la fenêtre principale

À la fin de ce sujet, vous devriez afficher une fenêtre principale qui ressemble à celle de la Figure 1.

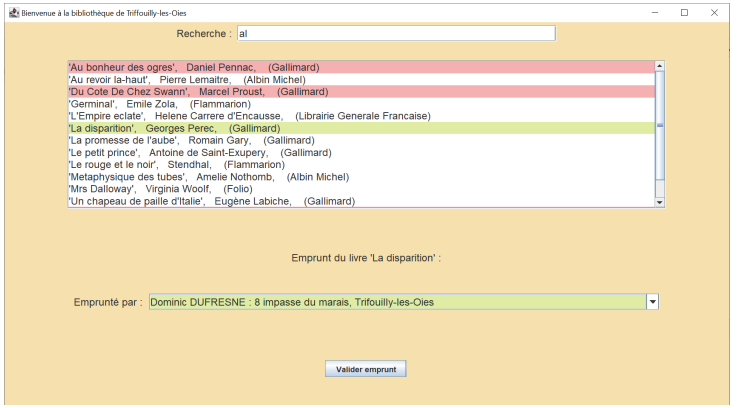

Figure 1: Fenêtre principale de l'application de gestion de la bibliothèque.

La fenêtre sera séparée en trois composants :

- une barre de recherche pour filtrer la liste des livres de la bibliothèque ;
- la liste des livres de la bibliothèque ;
- un panneau pour gérer les nouveaux emprunts, avec un menu déroulant pour sélectionner l'usager qui emprunte le livre, et un bouton pour valider le nouvel emprunt.

Voici les fonctionnalités principales qui sont attendues :

- à l'ouverture de l'application seules la barre de recherche et la liste des livres sont visibles ;
- dans la liste des livres, ceux qui sont actuellement empruntés sont identifiables avec une couleur spéciale (en rouge dans l'exemple) ;
- si on double clique sur un livre déjà emprunté, alors un menu pop-up permet de rendre le livre ;
- si on double clique sur un livre qui n'est pas emprunté, alors la panneau du bas s'affiche, et on peut choisir dans la liste déroulante l'usager qui emprunte le livre ; on clique ensuite sur le bouton valider pour réaliser l'emprunt, et une boîte de dialogue apparait pour confirmer l'emprunt.

1.1 Création de la fenêtre principale

Question 1. Reprenez le projet créé lors de la séance précédente. Ajoutez un nouveau paquetage **vue** dans lequel on regroupera toutes les classes liées à l'interface graphique de notre application. Dans ce paquetage, créez une fenêtre nommée **FenetreBiblio** qui sera la fenêtre principale de l'application.

Faites en sorte que :

- la fenêtre ait un titre ;
- l'application termine sont exécution lorsque l'on ferme la fenêtre ;
- la fenêtre apparaisse en plein écran ;
- si on clique sur l'icône pour réduire la fenêtre, la fenêtre ait une taille "raisonnable".

Testez que tout fonctionne correctement.

Pour rappel, les fenêtres graphiques sont des objets qui héritent de la classe **JFrame**. Quelques méthodes utiles de la classe **JFrame** sont : **setTitle**, **setDefaultCloseOperation**, **setExtendedState** et **setSize**.

1.2 Création des panneaux

Question 2. Ajoutez une classe **PanneauEmprunt** qui représentera le panneau bas de la fenêtre. Pour le moment, le panneau sera vide.

Les panneaux sont des objets qui héritent de la classe **JPanel**.

Question 3. Ajoutez une classe **PanneauLivres** qui représentera le panneau haut de la fenêtre sur lequel sont présents :

- une étiquette "Recherche : " ;
- un champ de texte à remplir par l'utilisateur ;
- une liste qui contiendra les livres de la bibliothèque.

Les types d'objet qui peuvent vous être utiles sont : **JLabel**, **JTextField** et **JList**.
Notez que la classe **JTextField** possède un constructeur qui prend en paramètre un entier qui représente le nombre maximum de caractères qui peuvent être vus dans le champ de texte.
Notez que la classe **JList** est une classe paramétrée, et on peut donc spécifier le type d'objets qui sont contenues dans la liste (de manière similaire à ce qu'on fait avec des **ArrayList** ou **LinkedList**).
Enfin, notez que la classe **JList** ne propose pas directement le défilement (avec une barre pour naviguer dans la liste). Pour créer une liste qui défile, il faut créer un objet de type **JScrollPane** qui montrera une vue de la liste. Voici un exemple pour ajouter une liste qui défile sur un panneau :

```
1  JPanel panel = new JPanel();
2  JList<TypeObjet> list = new JList<>();
3  JScrollPane scrollPane = new JScrollPane(list);
4  panel.add(scrollPane);
```

Question 4. Modifiez la classe **FenetreBiblio** de telle sorte que les deux panneaux soient affichés sur la fenêtre. Testez que tout fonctionne correctement.

Pour le moment, on ne cherche pas à avoir un affichage joli. Ne vous étonnez pas si vous ne voyez rien !

1.3 Théorie : les gestionnaires de mise en page

1.3.1 Intérêt d'un gestionnaire de mise en page

Les objets de type **JPanel** disposent d'une méthode **add** qui permet d'ajouter des objets graphiques (qui héritent de **JComponent**) sur le panneau. Mais cet ajoute se fait de telle sorte que les objets sont disposés sur le panneau les uns après les autres. Or cette disposition n'est pas toujours celle qui est recherchée.

La manière la plus élégante de disposer des objets sur un panneau est de déléguer la gestion de la mise en page des composants graphiques à un objet qu'on appelle *layout manager*. Il existe différents types d'objet qui permettent de faire différents types de mise en page. Ces objets implémentent l'interface **LayoutManager**. La classe **JPanel** possède une méthode **setLayout** qui permet de définir le gestionnaire de mise en page à utiliser.

1.3.2 Gestionnaire de type GridLayout

La classe **GridLayout** permet de disposer les composants graphiques dans une grille. Le constructeur prend en paramètre le nombre de lignes et le nombre de colonnes de la grille. Avec un tel gestionnaire de mise en page, chaque appel à la méthode **add** de la classe **JPanel** va positionner les objets dans la grille de haut en bas et de gauche vers la droite. Voici un exemple :

```
1  JPanel panel = new JPanel();
2  GridLayout layout = new GridLayout(3, 2); // 3 lignes et 2 colonnes
3  panel.setLayout(layout);
4  panel.add(new JLabel("A")); // label positionne en haut a gauche
5  panel.add(new JLabel("B")); // label positionne en haut a droite
6  panel.add(new JLabel("C")); // label positionne au milieu a gauche
7  panel.add(new JLabel("D")); // label positionne au milieu a droite
8  panel.add(new JLabel("E")); // label positionne en bas a gauche
9  panel.add(new JLabel("F")); // label positionne en bas a droite
```

Notez que le panneau est divisé en cases qui ont toutes la même taille, il n'est pas possible de modifier la taille d'une ligne ou d'une colonne.

1.3.3 Gestionnaire de type BorderLayout

La classe **BorderLayout** permet de disposer les composants graphiques de telle sorte qu'il y ait un composant central entouré de 4 composants au-dessus, en-dessous, à gauche et à droite. Lors de l'ajout de chaque composant avec la méthode **add**, un second paramètre permet de préciser la localisation du composant. Voici un exemple :

```
1  JPanel panel = new JPanel();
2  BorderLayout layout = new BorderLayout();
3  panel.setLayout(layout);
4  panel.add(new JLabel("A"), BorderLayout.NORTH);
5  panel.add(new JLabel("B"), BorderLayout.SOUTH);
6  panel.add(new JLabel("C"), BorderLayout.EAST);
7  panel.add(new JLabel("D"), BorderLayout.WEST);
8  panel.add(new JLabel("E"), BorderLayout.CENTER);
```

Notez qu'il est possible de créer des composants horizontaux ou verticaux de taille fixe et qui sont invisible. Cela permet par exemple de définir des marges autour d'un composant central. Voici un exemple pour mettre un bouton avec un espace vide de 50 pixels autour :

```
1  JPanel panel = new JPanel();
2  BorderLayout layout = new BorderLayout();
3  panel.setLayout(layout);
4  panel.add(new JButton("Bouton central", BorderLayout.CENTER);
5  panel.add(Box.createHorizontalStrut(50), BorderLayout.NORTH);
6  panel.add(Box.createHorizontalStrut(50), BorderLayout.SOUTH);
7  panel.add(Box.createVerticalStrut(50), BorderLayout.EAST);
8  panel.add(Box.createVerticalStrut(50), BorderLayout.WEST);
```

1.3.4 Le cas des JFrame

Par défaut, le gestionnaire de mise en page d'une **JFrame** est de type **BoderLayout**. Évidemment, il est possible de changer le gestionnaire de mise en page en appelant la méthode **setLayout**.

1.4 Mise en page de l'application

Question 5. Modifier votre code en ajoutant des gestionnaires de mise en page qui permettront d'avoir un rendu propre de votre interface graphique. Testez que tout fonctionne correctement.

Choisissez bien le type de gestionnaire de mise en page que vous souhaitez utiliser. Notez qu'il est possible de créer des "sous-pages" en ajoutant des objets de type **JPanel**.

Question 6. Modifier votre code afin de définir la couleur de fond des panneaux, ainsi que la fonte utilisée pour objets graphiques. Testez que tout fonctionne correctement.

Vous pouvez utiliser la méthode **setBackground** de la classe **JPanel** ainsi que la méthode **setFont** des différents composants graphiques.
Assurez-vous d'avoir une uniformité des couleurs et des fontes.

2 Connexion à la base de données

Avant de mettre en place la logique d'utilisation de l'interface graphique, nous allons mettre en place la connexion avec la base de données. En effet, pour alimenter les objets graphiques de l'interface, il nous faudra effectuer des requêtes sur la base de données.

Question 7. Dans la classe **FenetreBiblio**, ajoutez un attribut de type **RequeteBiblio** qui sera initialisé dans le constructeur par défaut.

Assurez-vous de gérer de façon propre, du point de vue de l'utilisateur, le cas où la connexion avec la base de données a échoué.

Testez que tout fonctionne correctement, en particulier si la connexion avec la base de données n'a pas été possible.

Pour indiquer à l'utilisateur qu'il y a eu une erreur, on peut utiliser la méthode statique **showMessageDialog** de la classe **JOptionPane**.

Question 8. Modifiez les classes **PanneauLivres** et **PanneauEmprunt** de telle sorte qu'elles aient un attribut de type **RequeteBiblio**, ainsi qu'un constructeur par donnée de cet attribut (qui remplacera le constructeur par défaut).

Notez ici que nous proposons de créer une fois un objet de type **RequeteBibio** dans la classe **FenetreBiblio**, et la référence de cet objet est ensuite passé aux autres classes. Ceci permet d'éviter d'avoir plusieurs connexions en parallèle sur la base de données.

3 Liste des livres

Dans cette section, nous souhaitons mettre en place la logique d'utilisation pour lister les livres dans le panneau haut de l'application. Ceci regroupe les éléments suivants :

- lister les livres dans l'objet de type **Jlist** ;
- faire du filtrage avec la barre de recherche ;
- avoir un affichage spécifique selon que le livre soit actuellement emprunté ou non.

3.1 Théorie : remplir une JList avec un modèle de données

Une manière propre d'indiquer quels sont les éléments à afficher sur un objet graphique de type **JList** est d'utiliser un modèle de données. Il suffit donc dans un premier temps de remplir le modèle de données, puis dans un second temps d'indiquer à la liste quel est le modèle à utiliser.

Pour une **JList**, un modèle de données doit être une classe qui implémente l'interface **ListModel**. Pour un modèle de données standard, on peut utiliser la classe **DefautListModel**. Si on souhaite créer un modèle plus spécifique, on peut alors créer une nouvelle classe qui hérite de **AbstractListModel**.

Voici un petit exemple de code pour définir le modèle d'une **JList** à partir d'un objet de type **DefaultListModel** :

```
1  List<TypeObjet> elements = .... // elements a afficher dans la JList
2  JList<TypeObjet> list = new JList<>();
3  DefaultListModel<TypeObjet> model = new DefaultListModel<>();
4  for(TypeObjet obj : elements) {
5     model.addElement(obj);
6  }
7  list.setModel(model);
```

3.2 Afficher la liste des livres

Question 9. Modifiez votre sorte de telle sorte qu'au lancement de l'application la liste de tous les livres soit affichée.

Testez que tout fonctionne correctement.

Notez que par défaut la **JList** affiche chaque élément en faisant appel à la méthode **toString** du type des objets de la liste.
Pour le moment, nous nous contenterons de cet affichage basique. Nous améliorerons cela plus tard.

3.3 Théorie : écouteur d'événement

Ce que l'on appelle *événements de bas niveau* sont des actions physiques de l'utilisateur sur le clavier ou la souris ; c'est par exemple le cas d'un clic dans une fenêtre. Les événements de bas niveau peuvent être classifiés en quatre catégories :

- les événements liés à la souris : appui sur un bouton, relâchement d'un bouton, double clic, glisser de la souris, etc. ;
- les événements liés au clavier : appui sur une touche, relâchement d'une touche, etc. ;

- les événements de focalisation ;
- les événements de gestion des fenêtres.

Lorsqu'il y a un événement sur un objet graphique, cela peut déclencher une action exécutée par d'autres objets. L'objet graphique qui crée des événements est appelé *source*. Cet objet *source* délègue le traitement de l'événement à un objet *écouteur* (*listener*) qui traite l'événement. L'objet *écouteur* doit "s'enregistrer" auprès de l'objet *source* qui peut déclencher ces événements.

En Java, chaque *écouteur* implémente une interface particulière correspondant à une catégorie d'événements. On parle bien ici d'interface en tant que concept de la POO et non pas d'interface graphique. Chaque méthode proposée par l'interface correspond à un événement de la catégorie.

Par exemple, il existe une catégorie d'événements clavier qu'on peut traiter avec un écouteur qui est un objet d'une classe implémentant l'interface **KeyListener**. Cette interface comporte trois méthodes correspondant chacune à un événement particulier : **keyTyped**, **keyPressed**, et **keyReleased**.

Ainsi, pour créer un écouteur spécifique pour traiter des événements clavier, il suffit de créer une classe qui implémente l'interface **KeyListener**, comme dans l'exemple suivant :

```
1  public class MyListener extends KeyListener {
2     @Override
3     public void keyTyped(KeyEvent e) {
4        System.out.println("appui sur une touche");
5     }
6
7     @Override
8     public void keyPressed(KeyEvent e) {
9        System.out.println("une touche est enfoncee");
10    }
11
12    @Override
13    public void keyReleased(KeyEvent e) {
14       System.out.println("relachement d'une touche");
15    }
16 }
```

Ensuite, on peut créer une instance de cette classe que l'on "enregistrera" auprès d'un composant graphique, en appelant une méthode dont le nom est **addKeyListener**. L'exemple suivant montre comment enregistrer un tel écouteur sur un champ de texte :

```
1  MyListener ecouteur = new MyListener();
2  JTextField champTexte = new JTextField(50);
3  champTexte.addKeyListener(ecouteur);
```

Enfin, notez que dans l'exemple précédent si la classe **MyListener** a été crée pour être utilisée une seule fois par un seul composant graphique, on pourrait tout à fait utiliser une classe anonyme.

3.4 Filtrer la liste des livres

Question 10. Modifiez votre code de telle sorte que lorsque que l'utilisateur tape du texte dans la barre de recherche, la liste soit filtrée et affiche seulement les livres dont le titre, l'auteur ou l'éditeur contiennent le texte tapé dans la barre.

Testez le bon fonctionnement de votre code.

Pour avoir accès au texte actuellement écrit dans un objet de type **JTextField**, vous pouvez utiliser la méthode **getText**.

3.5 Théorie : objets pour définir le rendu graphique

Il est possible de définir la manière dont sont affichés les éléments les éléments d'un composants graphique qui contient une liste d'items. Nous nous intéresserons ici en particulier au cas de composants de type **JList**.

La classe **JList** possède une méthode **setCellRenderer** qui prend en paramètre un objet qui doit implémenter l'interface **ListCellRenderer<TypeObjet>** où **TypeObjet** est le type d'objet contenus dans la **JList**.

L'interface **ListCellRenderer** déclare une seule méthode **getListCellRendererComponent** qui possède 5 paramètres : la **JList** sur laquelle on veut définir le rendu, une valeur de la liste (donc de type **TypeObjet**), l'indice dans la liste de cette valeur, un booléen pour indiquer si la valeur est actuellement sélectionnée, et un booléen pour indiquer si la valeur a le focus. Par ailleurs, cette méthode renvoie un objet de type **Component**, dont la méthode **paint** sera appelée pour faire le rendu de la cellule. En général, on utilise un objet de type **JLabel** qui est un sous-type de **Component**.

Voici un exemple de définition d'une classe qui implémente l'interface **ListCellRenderer** :

```
1  public class MyRenderer implements ListCellRenderer<TypeObjet> {
2     @Override
3     public Component getListCellRendererComponent(
4                JList<? extends TypeObjet> list,
5                TypeObjet value,
```

```
                int index,
                boolean isSelected,
                boolean cellHasFocus) {
        JLabel label = new JLabel();
        label.setText(value.textToDisplay());
        if(cellHasFocus) {
           label.setForeground(Color.BLACK);
           label.setOpaque(true);
           label.setBackground(Color.RED);
        } else {
           label.setForeground(Color.GREEN);
        }
        return label;
    }
}
```

Ensuite, pour indiquer à une **JList** la lanière de faire le rendu des éléments, on peut utiliser le code suivant :

```
JList<TypeObjet> list = new JList<>();
MyRenderer rendu = new MyRenderer();
list.setCellRenderer(rendu);
```

3.6 Rendu graphique des éléments de la liste

Question 11. Modifiez votre code de telle sorte que lorsque que les livres qui sont actuellement empruntés apparaissent avec une couleur différente dans la liste. Faites aussi en sorte qu'on puisse voir facilement si un item de la liste est sélectionné ou a le focus. Définissez également la fonte des items de la liste, et faites en sorte d'afficher un texte propre pour chaque item.

Testez le bon fonctionnement de votre code.

4 Rendre un livre

Dans cette section, nous souhaitons gérer au travers de l'interface graphique le rendu d'un livre qui est actuellement emprunté. Nous souhaitons avoir le fonctionnement suivant :

- si on fait un double clic avec la souris sur un élément de la liste des livres, et que le livre sélectionné est actuellement emprunté, alors un menu popup s'ouvre avec une seule possibilité : rendre le livre sélectionné ;
- si on clique sur ce seul item du menu, alors le livre est rendu, et donc l'affichage de la liste est mis à jour en conséquence.

Question 12. Modifiez votre code de telle sorte que si on réalise un double-clic sur un livre de la liste qui est actuellement emprunté, alors on affiche sur la console le titre de ce livre.

Testez que tout fonctionne correctement.

Ici on fait une première étape pour vérifier qu'on arrive à avoir un déclenchement seulement en cas de double clic sur un livre emprunté. Notez que comme on s'intéresse à un événement lié au clic sur la souris, un écouteur de type **MouseListener** semble approprié.
Par ailleurs, notez que la classe **MouseEvent** possède une méthode **getClickCount** qui permet de savoir combien de clics consécutifs ont été réalisés.

Question 13. Modifiez votre code de telle sorte qu'à présent au lieu d'un affichage console, un menu popup s'ouvre avec un item qui propose de rendre le livre, et si on clic sur cet item, alors le livre est rendu, et l'affichage dans la liste des livres est mis à jour en conséquence.

Testez que tout fonctionne correctement.

Les menus popup sont de type **JPopupMenu**. Les items de ces menus sont de type **JMenuItem**. On ajoute un item sur un menu popup avec la méthode **add**. On affiche le menu popup en appelant la méthode **show**.
On affecte un menu popup à un objet graphique avec la méthode **setComponentPopupMenu**.
On peut ajouter un écouteur sur un item de menu popup pour indiquer ce qu'il se passe en cas d'action sur l'item (par exemple après avoir relâché le bouton de souris).

5 Emprunter un livre

Dans cette dernière section, nous souhaitons mettre en place l'emprunt d'un livre. On attend le fonctionnement suivant du point de vue de l'utilisateur :

- lors du double clic sur un livre de la liste qui n'est pas actuellement emprunté, sur le panneau d'emprunt s'affichent : un label avec le nom du livre à emprunter, une **JComboBox** avec les utilisateurs de la bibliothèque, un bouton pour valider l'emprunt ;
- on sélectionne dans la **JComboBox** l'utilisateur qui réalise l'emprunt ;
- on appuie sur le bouton de validation ;
- une boîte de dialogue s'ouvre pour indiquer que le livre a bien été emprunté, et la liste est mise à jour en conséquence ;
- le panneau d'emprunt redevient vide.

Question 14. Modifiez votre code de telle sorte que l'on puisse réaliser des emprunts depuis l'interface graphique selon la procédure décrite ci-dessus.

Testez que votre code fonctionne correctement.