POO – Programmation Orientée Objet en Java

Fiche de TP numéro 4

Introduction aux fenêtres

On veut afficher une petite fenêtre au centre de l'écran, contenant un message indiquant la taille de l'écran. On utilise un objet JFrame pour réaliser la fenêtre.

Étape 1 : *Préparations*

À l'intérieur de votre répertoire `Java`, créez un répertoire `TP4`. Créez un fichier pour la classe `Exo1`.

Gardez aussi une fenêtre de navigateur (éventuellement sur un autre bureau) avec la documentation API. Dans le module `java.desktop`, dans le paquetage `javax.swing`, cherchez les informations sur la classe `JFrame`.

Étape 2 : *Importations*

Pour réaliser cette application vous devez d'abord importer les paquetages de swing. Mettez ces instructions avant de déclarer une classe.

```
import javax.swing.*;
import java.awt.*;
```

Étape 3 : *Déclaration et affichage*

La classe que vous écrivez ne contient que l'instruction `main`. Nous y déclarons dans un premier temps un objet `fenetre` de classe `JFrame` et nous en créons une instance dont le titre est *Une fenêtre*. Nous affichons la fenêtre en invoquant la méthode `setVisible()`.

Reportez-vous à la documentation pour trouver comment appeler le constructeur et comment utiliser la méthode `setVisible()`. De quelle classe la méthode `setVisible()` est-elle héritée ?

Étape 4 : *Pour fermer la fenêtre*

Conformément aux indications données dans l'entête de la documentation API sur `JFrame`, nous ajoutons :
`fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
avant l'appel à la méthode `setVisible()`.

Cette instruction associera un clic sur la croix de la fenêtre à un arrêt du programme.

Étape 5 : *Exécution !*

Compilez et lancez cette première application. Normalement vous ne devez pas voir grand chose se passer. La fenêtre apparaît en haut à gauche de l'écran. Vous pouvez la re-dimensionner. L'appui sur la case de fermeture ferme la fenêtre et quitte le programme.

Étape 6 : *Réglons le problème de la position et taille*

Utilisez la méthode `setBounds` avec les 4 valeurs entières suivantes 50,100,500,300 pour modifier la taille et la position de la fenêtre.

Remarquez dans l'aide que la méthode `setBounds` n'est pas dans la classe `JFrame` mais plutôt dans la classe `Component` dont `JFrame` hérite. Que signifient les quatre valeurs passées en paramètres de `setBounds()` ?

Étape 7 : *La résolution de l'écran*

Nous affichons maintenant la résolution de l'écran dans cette fenêtre. Pour cela, nous créons un objet de la classe `Dimension` que nous pouvons appeler `dimEcran`. Nous y plaçons la valeur retournée par cette expression :

`fenetre.getToolkit().getScreenSize()`

Cherchez ensuite dans la documentation API la classe `Dimension` et cherchez comment obtenir la hauteur et la largeur.

Affichez la valeur de dimEcran sur la console (à l'aide de `System.out.println()`).

Étape 8 : *Créons un JLabel*

On prefère le plus souvent afficher les informations directement dans la fenêtre. On utilise pour cela un composant graphique qui permet d'afficher du texte ou une image : le `JLabel`.

Pour ajouter un composant graphique dans la `JFrame`, utilisez le code suivant :

```
JLabel label = new JLabel("Taille de l'écran : "+dimEcran);
fenetre.getContentPane().add(label);
```

Que se passe-t-il ?

Utilisez `fenêtre.pack();` après avoir ajouté le composant graphique dans la fenêtre. Cela permet d'ajuster la taille de la fenêtre en fonction des composants qu'elle contient.

Étape 9 : *Placement au centre*

Utilisez la méthode `setLocation` pour remplacer l'instruction `setBounds` et poser maintenant la fenêtre au centre de l'écran (comment pouvez-vous accéder à la largeur et à la hauteur de la fenêtre ?).

Étape 10 : *Avec dix bons points, une image...*

Vous trouverez sur Moodle des images que vous pouvez copier chez vous, dans le répertoire `bin`. Nous allons remplacer notre message par une image. Copiez votre fichier `Exo1.java` sous le nom `Exo1b.java`, puis changez le nom de la classe en `Exo1b`. Remplacez le message `"Taille de l'ecran ..."` par

`new ImageIcon("images/Bird.gif").`

Compilez, interprétez...

Si vous avez des problèmes pour lire votre image, vous pouvez utiliser la solution suivante, qui permet aussi d'accéder à des images qui se trouvent dans une archive Java.

```
import java.net.URL;
...
URL url = Exo1b.class.getResource("images/Bird2.gif");
JLabel label = new JLabel(new ImageIcon(url));
```

Bravo, vous êtes prêt pour la partie suivante.

Introduction aux Layout Managers

On veut créer une interface graphique pour un jeu du nombre mystérieux. Nous adopterons plusieurs formes pour étudier comment Java gère le placement des objets dans les fenêtres.

Étape 11 : *Importations*

Créez un fichier pour la classe `Exo2`.

Pour réaliser cette application vous devez d'abord importer les paquetages de swing. Mettez cette instruction avant de déclarer une classe.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

Étape 12 : *Déclaration, affichage et fermeture de la fenêtre*

La classe que vous écrivez ne contient que l'instruction `main`. Nous y déclarons dans un premier temps un objet `fenetre` de classe `JFrame` et nous en créons une instance dont le titre est *Nombre mystérieux*. Nous affichons la fenêtre en invoquant la méthode `setVisible()`. N'oubliez pas de préciser le comportement de votre fenêtre lors d'un click sur la croix :

```
fenêtre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Étape 13 : *Déclarer les objets*

Nous avons besoin de :

- deux boutons (JButton), que nous nommerons `btnFin` et `btnCommencer`, et qui auront comme intitulé, respectivement, `"Fin"` et `"Commencer"` ;
- deux labels (JLabel), que nous nommerons `lblResultat` et `lblNbCoups`. Initialisez `lblResultat` avec le message suivant `"Trouvez un nbre entre 0 et 99"`, et `lblNbCoups` avec le message `"0"`. Vous pouvez centrer horizontalement vos messages dans le `JLabel`. Regardez dans la documentation de l'API, aussi bien la classe `JLabel` que `SwingConstants` (attention, cette dernière est dans la catégorie des interfaces) ;
- une zone de saisie (JTextField),que nous nommerons `txtSaisie`. Initialisez-la pour cinq caractères maximum.

Faites-en la déclaration.

Étape 14 : *Placer les composants*

Dans les applications graphiques, basiquement, on différencie les composants de bases, ou *contrôles* (comme les boutons, labels et autre zone de saisie que vous venez de déclarer), et les *conteneurs*, qui sont les "boîtes" à l'intérieur desquelles on pose les contrôles, ou d'autres conteneurs. Une `JFrame` est un conteneur, par exemple. Elle possède un conteneur par défaut dans lequel on va déposer tous les éléments que nous venons de déclarer et créer. On peut se simplifier la tâche en déclarant une référence sur le conteneur par défaut de la fenêtre par :

```
Container cont = fenêtre.getContentPane();
```

Il suffira ensuite d'invoquer la méthode `add` de l'objet `cont` de classe `Container`. La méthode `add` que nous utilisons demande deux paramètres : l'objet à placer et la position. On utilise les constantes `BorderLayout.CENTER`, `BorderLayout.EAST`, `BorderLayout.WEST`, `BorderLayout.SOUTH`, `BorderLayout.NORTH`.

Étape 15 : *Liaison avec le jeu*

Le développement du jeu a été effectué, la documentation et un `.jar` sont disponibles sur Moodle (`myst.jar`). La liaison entre l'interface graphique et le jeu se fait par les méthodes de la classe `LiaisonMystIG`. Ici, le `.jar` ne nécessite aucun `import`, il suffit juste que vous l'ajoutiez dans le `classpath` de `javac` et de `java`.

```
moi@mamachine:~/java/tp4$ javac -d bin -cp src:bin:bin/myst.jar src/Exo2.java
moi@mamachine:~/java/tp4$ java -cp bin:bin/myst.jar Exo2
```

Et dans le contructeur de la classe `Exo2` vous faites directement appel aux méthodes de `LaisonMystIG` :

```
// associer les contrôles graphiques au jeu du nbre mystérieux
LiaisonMystIG.setBoutonFin(btnFin);
LiaisonMystIG.setBoutonCommencer(btnCommencer);
LiaisonMystIG.setLabelResultat(lblResultat);
LiaisonMystIG.setLabelNbCoups(lblNbCoups);
LiaisonMystIG.setZoneTexte(txtSaisie);
```

Étape 16 : *Finition*

N'oubliez pas de redimensionner la fenêtre et affichez le tout !

Étape 17 : *Et autrement ? Horizontalement*

Nous avons utilisé un gestionnaire de positionnement des composants pour poser les contrôles graphiques dans le conteneur. Ce gestionnaire était un `BorderLayout`. Si ce n'est déjà fait, allez voir cette classe dans la documentation de l'API.

Ce gestionnaire est le gestionnaire par défaut du conteneur associé à une `JFrame`. Il est possible d'en choisir un autre, en modifiant le gestionnaire de positionnement associé à un conteneur par la méthode `setLayout()` de la classe `Container`. Sauvegardez votre premier programme, puis enregistrez-le sous le nom `Exo2b.java` (n'oubliez pas de changer le nom de la classe). Créez et installez un gestionnaire de type `FlowLayout` pour le conteneur par défaut de `fenêtre`. Pour ajouter les contrôles graphiques dans la fenêtre, vous utiliserez une méthode `add` à un seul argument.

Centrez les composants. Installez-les horizontalement, centrés, puis alignés à droite, faites-varier les espaces verticaux et horizontaux.

Étape 18 : *Verticalement*

Dans la classe `Exo2c`, créez et installez un gestionnaire `BoxLayout`. Alignez vos composants verticalement.

Étape 19 : *En grille*

Dans la classe `Exo2d`, créez et installer un gestionnaire `GridLayout`. Essayez avec 3 lignes et 2 colonnes, puis avec 3 colonnes et 2 lignes, variez l'ordre d'ajout de vos composants.

Introduction à la gestion des évènements

Nous allons écrire une application qui affiche une nouvelle image à chaque clic de l'utilisateur.

Étape 20 : *Déclaration, affichage et fermeture de la fenêtre*

La classe `MaPremiereAppliGraphique` contient l'instruction `main`. Nous y déclarons dans un premier temps un objet `fenêtre` de la classe `JFrame` et nous créons une instance de `JFrame` dont le titre est *Les images*. Nous affichons la fenêtre en invoquant la méthode `show` (ou `setVisible(true)`). N'oubliez pas de préciser le comportement de votre fenêtre lors d'un clic sur la croix.

Étape 21 : *Les images*

Créez un répertoire `images` dans votre répertoire `java` et copiez-y, si ce n'est déjà fait, les images `Bird.gif, Bird2.gif, Cat.gif, Cat2.gif, Dog.gif, Rabbit.gif, Pig.gif` (disponibles sur Moodle).

De retour dans votre classe `MaPremiereAppliGraphique`, créez comme attribut public de classe un tableau `TABIMAGES` de 7 objets de type `ImageIcon` contenant chacun une des images que vous venez de copier.

Étape 22 : *Création des contrôles graphiques*

Nous allons avoir besoin simplement d'un bouton `leBtn` de type `JButton`. Un `JButton`, comme un `JLabel`, peut afficher une image, mais il peut en plus réagir aux actions de l'utilisateur ! C'est ce qu'il nous faut. Déclarez `leBtn` comme une variable locale dans la méthode `main`, et faites-le afficher la première image de `TABIMAGES`.

Vous pouvez aussi demander à afficher aléatoirement une de ces images (voir la documentation de `java.util.Random`).

Étape 23 : *Gestion des évènements – la classe `GereClickBouton`*

Nous voulons maintenant qu'un click sur le bouton provoque l'affichage de l'image suivante dans le tableau des images (on veut un fonctionnement en boucle : lorsque la dernière image est affichée, on revient à la première).

Dans un premier temps, nous avons besoin d'un objet qui soit capable de recevoir l'information *l'utilisateur a cliqué*, et d'effectuer l'action correspondante. les classes qui sont capables de recevoir ce genre d'informations sont des *Listener*, et, dans le cas qui nous intéresse (un bouton actionné), il s'agit d'un `ActionListener`. Créez une classe `GereClickBouton` qui implémente l'interface `ActionListener`.

C'est ici, dans la méthode `actionPerformed`, que vous allez écrire le code pour votre application. Que doit faire `GereClickBouton` ? A chaque click de l'utilisateur, le message `actionPerformed` va lui être envoyé. Son travail consiste donc à demander au bouton de modifier son affichage. Il est donc nécessaire que :

- la classe `GereClickBouton` possède un attribut d'instance de type `int` que nous nommerons `compteur` initialisé à `0` ;
- la classe `GereClickBouton` possède un attribut d'instance de type `JButton`, que nous nommerons `btn` ;
- cet attribut (privé, bien sûr), soit initialisé dans le constructeur de `GereClickBouton` et ne puisse pas être modifié ni accédé depuis l'extérieur (aucun accesseur ou modificateur à écrire).

Déclarez ces attributs et écrivez le constructeur de `GereClickBouton`.

Étape 24 : *Gestion des évènements – implémentation*

Vous êtes maintenant prêt à écrire le contenu de la méthode `actionPerformed`. Celle-ci se contente d'incrémenter `compteur` puis de demander à `btn` de modifier son affichage. Regardez dans la documentation de l'APIquelle est la méthode de la classe `JButton` à utiliser.

Étape 25 : *Gestion des évènements – liaison entre les objets*

La dernière étape est la liaison entre le bouton `leBtn` et l'instance de la classe `GereClickBouton` qui va gérer les évènements. Ceci se fait, dans la méthode `main` de la classe `MaPremiereAppliGraphique`, par les lignes suivantes :

```
// on gère les évènements sur les boutons
leBtn.addActionListener(new GereClickBouton(leBtn));
```

Par cette instruction, `leBtn` enregistre une instance de la classe `GereClickBouton` qu'il préviendra (appel de la méthode `actionPerformed`) chaque fois qu'il recevra un clic de l'utilisateur.

Lancez votre application !

Étape 26 : *On complique un peu...*

Nous allons maintenant réutiliser cette première application pour en construire une deuxième. La seconde application est constituée d'une grille d'images qui se découvrent lorsque l'utilisateur passe la souris sur les images. Ce genre de jeu est utilisé pour développer la motricité des jeunes enfants, nous nous en servirons pour développer la gestion des évènements chez de jeunes programmeurs.

Créez une classe `DecouvreImage` qui contient seulement la méthode `main`. Dans la méthode `main`, créez une instance de `JFrame`, et choisissez comme gestionnaire de positionnement un `GridLayout` (avec 2 lignes et 2 colonnes) pour son conteneur principal. Vous écrirez, comme d'habitude, toutes les instructions nécessaires pour quitter facilement l'application, pour qu'elle soit correctement dimensionnée et qu'elle soit visible.

Dans un premier temps, pour vérifier votre code, ajoutez dans le conteneur principal quatre instances de `JButton` initialisés avec une image de votre choix.

Lancez votre application.

Étape 27 : *Les composants `UneImage`*

Quelles doivent être les propriétés des composants graphiques de base de votre application ? Ils doivent :

— pouvoir afficher une image ;
— écouter les évènements provenant de la souris ;
— mémoriser l'indice de l'image à afficher lorsque l'utilisateur aura correctement déplacé la souris.

Pour afficher une image, on pourrait se contenter d'un composant de type `JButton`, mais si on veut mémoriser un indice, il va falloir écrire une classe spécifiqueé : une sous-classe de `JButton` pour votre application.

Créez une classe `UneImage` qui hérite de `JButton`. Cette classe possède en plus
— comme attribut de classe, un tableau d'images (cf. classe `GereClickBouton`) ;
— comme attribut d'instance, un entier `indice` qui lui permet de connaitre quelle image il doit afficher.

Ecrivez la classe, avec ses attributs et un constructeur. Ce constructeur prend un indice en paramètre. Dans un premier temps, l'image sera directement affichée sur le composant (n'oubliez pas l'appel au constructeur de la super-classe). Modifiez la méthode `main` de `DecouvreImage` afin de poser des composants `UneImage` et non plus des `JButton`. Lancez votre application.

Étape 28 : *Gestion des évènements*

Les évènements qui nous intéressent sont des évènements gérés par des `MouseListener`. Cette interface propose la méthode `mouseEntered` qui répond à nos besoins. Les composants doivent être initialisés avec une image "vide", et lorsque le message `mouseEntered` sera envoyé, ils afficheront l'image qui est donnée par `indice`. Le plus simple est donc que ce soit les composants eux-mêmes qui gèrent les évènements.

Déclarez maintenant que la classe `UneImage` implémente l'interface `MouseListener`. Implémentez toutes les méthodes de l'interface avec un corps vide.

Ajoutez, à votre tableau d'images, `rien.gif`.

Dans le constructeur d'`UneImage`, initialisez maintenant l'affichage avec `rien.gif`, puis déclarez votre composant comme écoutant les évènements de type `MouseEvent` (par `addMouseListener`), et se déléguant à lui-même leur gestion.

Dans la méthode `mouseEntered`, faites afficher l'image donnée par l'attribut `indice`.

Lancez votre application.

Étape 29 : *Un peu d'animation...*

Dernier point, lorsque l'utilisateur clique sur une image, celle-ci doit se mettre à clignoter. Lorsque la souris sort du champ du composant, l'image s'arrête de clignoter.

Pour effectuer ce genre d'animation, le paquetage `javax.swing` propose la classe `Timer`. Un `Timer` est un objet qui envoit à intervalles réguliers un message `actionPerformed`. Une instance de `Timer` est créée en donnant deux informations : le délai, en millisecondes, entre deux messages, et l'objet destinataire des messages.

Ajoutez un attribut d'instance de type `Timer` à `UneImage`. Cet attribut est créé dans le constructeur. Pour le délai, prenez une valeur au moins égale à `200`. C'est l'instance courante d'`UneImage` qui sera destinataire des messages du `Timer`.

Déclarez qu'`UneImage` implémente également l'interface `ActionListener`. Dans la méthode `actionPerformed`, faites basculer l'affichage entre `rien.gif` et `TABIMAGES[indice]`. Cette méthode doit fonctionner comme un commutateur, vous aurez peut-être besoin d'ajouter un attribut d'instance.

Pour démarrer un `Timer`, il faut lui envoyer un message `start()`, pour l'arrêter un message `stop()`. Le `Timer` démarre lorsque la souris est cliquée sur le composant, arrêté lorsque la souris sort du champ du composant.