IHM – TP2

L'objectif de ce deuxième TP est de se familiariser avec la réalisation d'écouteurs. Il s'agit d'écrire le code réagissant aux divers évènements créés par les composants de l'interface afin de rendre l'IHM pleinement fonctionnelle.

Exercice 1

Nous proposons de réaliser un compteur sous la forme d'un bouton qui incrémente une valeur (initialement 0), comme illustré à la figure 1.

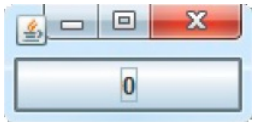

FIGURE 1 – Un compteur

1. Créer une classe `Compteur` constituée d'un unique bouton ayant 0 pour texte initial.
2. Ajouter au bouton un écouteur sous la forme d'une classe interne et anonyme. Cet écouteur incrémentera la valeur indiquée sur le bouton.

Un compteur c'est bien, trois c'est encore mieux, comme illustré à la figure 2.

FIGURE 2 – Trois compteurs

3. Créer une classe `CompteursCIA` constituée de trois boutons ayant 0 pour texte initial.
4. Ajouter à chacun des boutons un écouteur sous la forme d'une classe interne et anonyme, chargé d'incrémenter la valeur du bouton.

Utiliser des classes internes et anonymes entraîne de la duplication de code...

5. Créer une classe `CompteursCI` constituée de trois boutons ayant 0 pour texte initial.
6. Ajouter à chacun des boutons un écouteur sous la forme d'une classe interne, chargé d'incrémenter la valeur du bouton.

Exercice 2

Nous proposons de réaliser une version simple d'un lecteur audio, permettant d'écouter un seul morceau de musique prédéfini. L'interface du lecteur sera réduite, dans un premier temps, à

un unique bouton, permettant à l'utilisateur de lancer/reprendre la lecture du morceau ou encore de mettre sur pause, comme illustré à la figure 3. L'aspect du bouton alternera donc entre deux icônes, symbolisant respectivement que le morceau n'a pas encore démarré ou est en pause, ou qu'il est en cours de lecture.

FIGURE 3 – Lecteur audio 1 bouton

1. Créer une classe `Player1Bouton` constituée d'un unique bouton ayant "play.jpg" pour icône et "play_pressed.jpg" lorsqu'il est pressé (*i.e.*, `pressedIcon`).
2. Ajouter un écouteur au bouton sous la forme d'une classe interne qui changera son icône à chaque clic, alternant entre "play.jpg" (et "play_pressed.jpg") et "pause.jpg" (et "pause_pressed.jpg").

Nous souhaitons faire évoluer l'interface en passant à trois boutons : marche, pause et arrêt, comme illustré à la figure 4. Ainsi, chaque bouton a son propre rôle et n'a plus besoin de changer d'aspect (ou presque !). En effet, initialement la lecture n'a pas démarrée, et par conséquent les boutons de pause et d'arrêt ne devraient pas être actifs (*i.e.*, `enabled`) et donc avoir un aspect grisé. De plus, si la lecture est en cours, seuls les boutons de pause et d'arrêt devraient être actifs, alors que si elle est en pause, seuls les boutons de marche et d'arrêt devraient l'être.

FIGURE 4 – Lecteur audio 3 boutons

3. Créer une classe `Player3Boutons` constituée de trois boutons ayant respectivement "play.jpg", "pause.jpg" et "stop.jpg" pour icône, "play_pressed.jpg", "pause_pressed.jpg" et "stop_pressed.jpg" lorsqu'ils sont pressés, et "play_disabled.jpg", "pause_disabled.jpg" et "stop_disabled.jpg" lorsqu'ils sont inactifs.
4. Ajouter un écouteur à chacun des boutons afin de respecter les changements d'aspect du bouton.

Jusqu'à présent, nous ne gérions pas la lecture, la mise en pause et la reprise du morceau de musique.

5. En utilisant la classe `Son`[1] fournie, compléter votre code pour correctement gérer la lecture d'un morceau de musique.

Exercice 3

Au TP précédent, nous avons réalisé l'interface graphique du jeu *Le mot le plus long*. Nous proposons maintenant de le rendre opérationnel.

1. La classe `Son` dispose des méthodes `start`, `suspend`, `resume` et `stop` permettant de respectivement lancer la lecture d'un morceau de musique, de le mettre sur pause, de le reprendre et de l'arrêter.

1. Compléter votre classe `MotLePlusLong` en rajoutant un écouteur au `JTextField` de sorte qu'à chaque saisie d'un mot m dans ce champs, le contenu de la `JList` soit mis à jour avec la liste des mots qu'il est possible de former à partir des lettres de m.
2. Écrire une méthode de classe `motsRealisables` qui retourne un tableau d'`Object` constitué de tous les mots réalisables à partir des lettres du mot (`String`) passé en paramètre. Pour tester, vous pouvez utiliser les dictionnaires fournis.

Exercice 4

Nous proposons de réaliser un album photo. L'application initialisera l'album à partir de photos situées dans un répertoire donné et disposera d'un menu permettant d'ajouter d'autres photos. Une `JList` à gauche permettra de sélectionner le nom de la photo à afficher dans le panneau central. Il sera également possible de sélectionner la photo à afficher en cliquant sur l'une des icônes du bandeau bas. De plus, l'utilisateur pourra passer d'une photo à la précédente ou à la suivante en cliquant sur les boutons du panneau haut. Enfin, un `JSlider` à droite permettra de redimensionner la photo couramment affichée. Le noyau applicatif (*i.e.*, le modèle) de cette application est fourni. Il est constitué des classes `Photo` (qui mémorise non seulement l'image correspondant à la photo mais aussi un facteur de zoom) et `Album` (dont l'état est une `ArrayList` de `Photo`). Un visuel de l'application est donné à la figure 5.

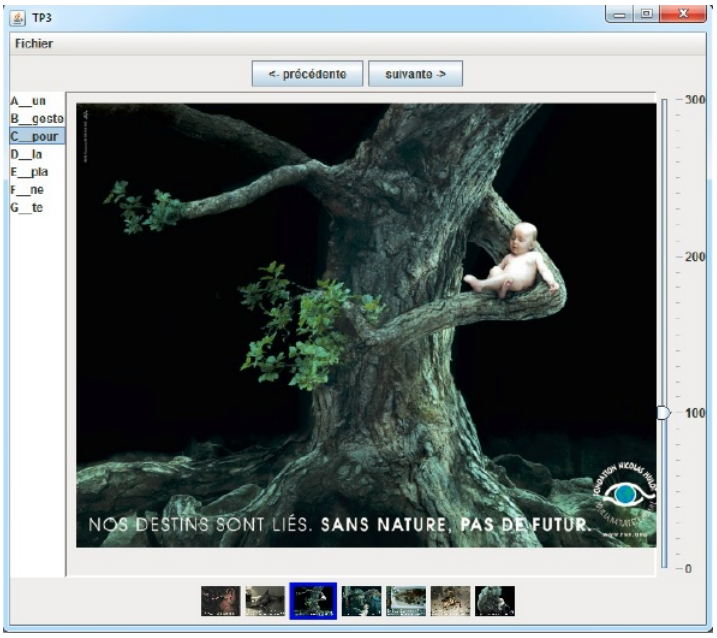

FIGURE 5 – Album photo

1. Créer une classe `AlbumPhoto` dont le constructeur fait appel aux différentes méthodes à implémenter ci-après, afin de construire l'IHM de l'application.
2. Écrire une méthode `creerCentre` permettant de créer le panneau central de taille préférée 600x500. Ce dernier est constitué d'un `JPanel` qui contient l'image (`JLabel`) courante.
3. Le curseur à droite permet de modifier le facteur zoom de l'image courante.
   (a) Écrire une méthode `creerSlider` permettant de créer le `JSlider`. Consulter la documentation de cette classe pour afficher et espacer les tirets principaux de 100 et les petits de 10, ainsi que pour afficher les valeurs des unités.
   (b) Ajouter un écouteur au `JSlider` qui met à jour le facteur zoom de l'image courante.

4. La liste à gauche permet de sélectionner une seule image à partir de son nom, qui devient alors l'image courante affichée dans le panneau central.
   (a) Écrire une méthode `creerListe` permettant de créer la liste [2] à gauche de l'interface, initialisée avec les noms des photos de l'album.
   (b) Ajouter un écouteur à la liste qui (1) met à jour le modèle en faisant de la photo choisie la photo courante, (2) affiche la photo sélectionnée dans le panneau central et (3) met à jour le curseur du `JSlider` afin qu'il corresponde au facteur de zoom de la photo sélectionnée.
5. Le bandeau bas est constitué de `JButton`.
   (a) Écrire une méthode `creerBandeauBas` permettant de créer un bandeau (ici un `JPanel`) composé d'autant de `JButton` qu'il y a de photos dans l'album. L'icône de ces boutons peut être obtenue *via* la méthode `getIcone` de la classe `Photo`. Les boutons n'ont pas de bordure et sont ajoutés à l'`ArrayList lesBoutons` afin que les autres composants puissent y accéder facilement.
   (b) Ajouter à chacun de ces boutons un écouteur qui (1) met à jour le modèle en faisant de la photo choisie la photo courante, (2) rajoute une bordure bleue de 4 pixels autour de l'icône sélectionnée (les autres icônes n'ont pas de bordure), (3) affiche la photo sélectionnée dans le panneau central, (4) met à jour le curseur du `JSlider` afin qu'il corresponde au facteur de zoom de la photo sélectionnée et (5) met à jour la sélection de la `JList` pour qu'elle corresponde à la photo sélectionnée.
   (c) Modifier l'écouteur de la `JList` de sorte à ce que l'icône sélectionnée dans le bandeau bas (celle avec une bordure bleue) corresponde à la sélection de la liste.
6. Le bandeau haut est constitué de deux `JButton`.
   (a) Écrire une méthode `creerBandeauHaut` afin de créer un `JPanel` composé de deux `JButton` «précédente» et «suivante».
   (b) Ajouter aux boutons «précédente» et «suivante» un écouteur qui (1) met à jour le modèle en faisant de la photo précédente/suivante (*i.e.*, d'index immédiatement inférieur/supérieur) la photo courante, (2) rajoute une bordure bleue de 4 pixels autour de l'icône de la photo courante (les autres icônes ne doivent pas avoir de bordure), (3) affiche la photosélectionnée dans le panneau central, (4) met à jour le curseur du `JSlider` afin qu'il corresponde au facteur de zoom de la photo sélectionnée, (5) met à jour la sélection de la `JList` pour qu'elle corresponde à la photo sélectionnée et (6) ajuste la disponibilité (*i.e.*, `setEnabled`) des boutons «précédente» et «suivante» selon que l'image courante soit la première (pas de précédente), la dernière (pas de suivante) ou au milieu d'autres photos.
   (c) Modifier les écouteurs de la `JList` et des boutons du bandeau bas pour qu'ils mettent à jour la disponibilité des boutons «précédente» et «suivante» en fonction de la position de la photo courante dans l'album.
7. L'application dispose d'un menu «Fichier» qui comporte deux items (séparés par un séparateur) : l'un pour ajouter une image à l'album et l'autre pour quitter l'application.
   (a) Écrire une méthode `creerMenu` permettant d'ajouter un menu à l'application constitué d'un `JMenuItem` «Ajouter une image», d'un séparateur et d'un `JMenuItem` «Quitter».
   (b) Ajouter un écouteur à «Quitter» afin qu'il ferme l'application (appel à `System.exit(0)`).
   (c) Ajouter un écouteur à «Ajouter une image» afin qu'il (1) ouvre une boite de dialogue permettant de choisir un nom de fichier, (2) ajoute la photo sélectionnée dans le modèle et (3) mette à jour tous les autres composants impactés par cet ajout.

2. La mise à jour du contenu d'une `JList` est plus simple si cette dernière est créée à partir d'une instance de `DefaultListModel`, remplie en utilisant la méthode `addComponent`. Ainsi, il est possible de rajouter facilement des éléments dans la `JList` en appelant `addComponent` sur son modèle (récupéré *via* `getModel()`) casté en `DefaultListModel`.