TP 3

Introduction à Swing et aux threads

1 – Présentation du TP

Objet du TP

Ce TP a pour objectif de se familiariser avec Swing et les threads.

Dans ce TP, vous allez concevoir votre première interface graphique et votre première animation.

Compétences travaillées dans ce TP

- Composants Swing
- Gestionnaires de placement
- Programmation événementielle : événements et écouteurs
- Threads
- Utilisation de la documentation de l'API Java

2 – Démarrage du TP

Suivez la même procédure de démarrage que dans les TPs précédents, en nommant votre projet « tp3 ».

De base, IntelliJ importe les classes de AWT et de Swing avec *. Pour désactiver ceci :

- Allez dans allez dans File > Settings > Editor > Code Style > Java > Imports
- Appuyez sur « - » dans la partie « Packages to Use Import with '*' » jusqu'à ce que la liste soit vide.

3 – Création d'une fenêtre

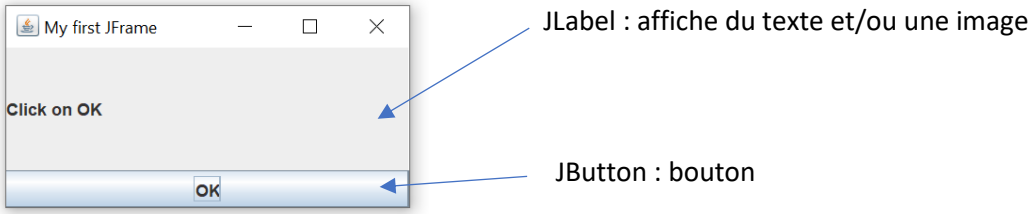
Créez une classe Window héritant de JFrame ayant les caractéristiques suivantes :

- Titre : « My first JFrame »
- Largeur : 300 px
- Hauteur : 100 px
- Le clic sur la croix ferme la fenêtre
- La fenêtre s'affiche à sa construction

Dans le main(), instanciez la classe Window et vérifiez qu'elle est conforme aux spécifications.

4 – Ajout des composants graphiques

Ajoutez les composants graphiques suivants à votre fenêtre :



Au besoin, consultez la documentation des classes JLabel et JButton pour comprendre comment les instancier et les paramétrer.

Rappel : BorderLayout est le gestionnaire de placement par défaut pour une JFrame.

5 – Clic sur le bouton

Affichage dans la console

Faites en sorte qu'un clic sur le bouton OK produise l'affichage « OK clicked ! » dans la console.

Rappel : Pour réagir au clic, il faut implémenter l'interface ActionListener.

Affichage dans le JLabel

Faites en sorte qu'un clic sur le bouton OK modifie le texte du JLabel en « OK clicked ! »

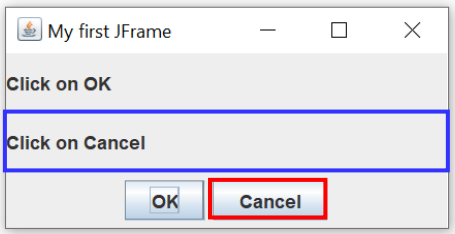
Indication : `j.setText(t)` permet d'afficher le texte `t` dans le JLabel `j`.

Rappel : La classe implémentant ActionListener doit avoir une référence sur le JLabel dans ses attributs pour le manipuler au moment du clic.

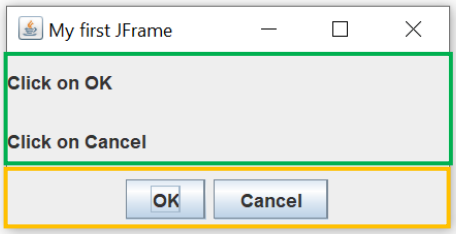
6 – Deux boutons

Ajout d'un deuxième bouton

Ajoutez un nouveau JButton (encadré en rouge) et un JLabel associé (encadré en bleu), et placez-les dans votre fenêtre comme illustré ci-dessous :



Indication : Il vous faudra créer deux JPanel (illustrés en vert et en orange) avec un gestionnaire de placement approprié.



Mise à jour du JLabel associé à chaque bouton au clic

Faites en sorte que :

- Un clic sur le bouton OK change le texte du JLabel « Click on OK » en « OK clicked ! »
- Un clic sur le bouton Cancel change le texte du JLabel « Click on Cancel » en « Cancel clicked ! »

Elément de réflexion : comment allez-vous différencier les deux boutons lorsqu'un ActionEvent sera émis ?

7 – Animation séquentielle de texte

On vous fournit une classe Animation, qui affiche un par un les caractères d'un texte dans un JLabel, en faisant une pause entre chaque caractère.

Récupérez cette classe sur la page Moodle du cours et intégrez-la à votre projet.

Regardez le code de cette classe. Dans la méthode animate(), on fait appel à la méthode statique `Thread.sleep(n)` pour faire une pause de `n` millisecondes. Cette méthode est susceptible de lancer une exception de type InterruptedException, que nous n'allons pas traiter ici. Les exceptions et leur traitement seront abordés dans un prochain cours. Les élèves qui le souhaitent peuvent lire l'encadré dédié pour plus de détails.

InterruptedException

Il s'agit d'une exception qui est levée quand un thread est interrompu par un autre thread. En principe, il faudrait gérer cette exception pour terminer le thread « proprement », par exemple libérer les ressources utilisées par ce thread. Ici il n'y a rien de particulier à faire, nous avons donc écrit un `catch` vide. Mais il faut avoir à l'esprit que l'écriture de `catch` vides est une mauvaise pratique en général, car elle stoppe la propagation de l'exception sans traiter l'erreur.

Dans le main(), ajoutez les lignes suivantes :

```
Animation a1 = new Animation(okLabel, "ABCDEF", 500);
a1.animate();

Animation a2 = new Animation(cancelLabel, "123456", 500);
a2.animate();
```

où `okLabel` et `cancelLabel` sont les références vers les JLabel associés aux boutons OK et Cancel (adaptez ces noms à votre code).

Lancez le programme.

Vous devriez voir le texte du premier JLabel s'animer, puis celui du deuxième : c'est ce qu'on appelle un comportement *séquentiel*.

Peut-on animer ces deux textes en même temps ? Oui, avec deux fils d'exécution différents !

8 – Animation de texte en parallèle

Nous allons modifier la classe Animation pour modéliser qu'une animation ***est un*** thread, ce qui se traduit naturellement par un héritage. Pour ce faire :

- Modifiez la classe Animation pour indiquer qu'elle hérite de la classe Thread (aucun import à faire, elle fait partie des classes de base du langage Java).
- Surchargez la méthode run(). Dans le corps de la méthode, appelez la méthode animate().

Voilà ! Nous n'avons plus qu'à instancier et lancer deux fils d'exécutions de type Animation dans le main(). Ceci revient simplement à remplacer les appels à animate() par des appels() à start() :

```
Animation a1 = new Animation(okLabel, "ABCDEF", 500);
a1.start();

Animation a2 = new Animation(cancelLabel, "123456", 500);
a2.start();
```

La méthode start() va appeler la méthode run() dans un fil d'exécution dédié.

Faites les modifications dans le main() et lancez le programme. Vous devriez voir les deux textes s'animer en parallèle, et se terminer en même temps.

Modifiez la vitesse de l'une des animations et observez le résultat.

Une alternative à l'héritage de Thread est d'*implémenter* l'interface Runnable. Ceci peut être utile voire nécessaire quand notre classe hérite déjà d'une autre classe (d'un composant Swing par exemple), Java ne permettant pas d'héritage multiple.

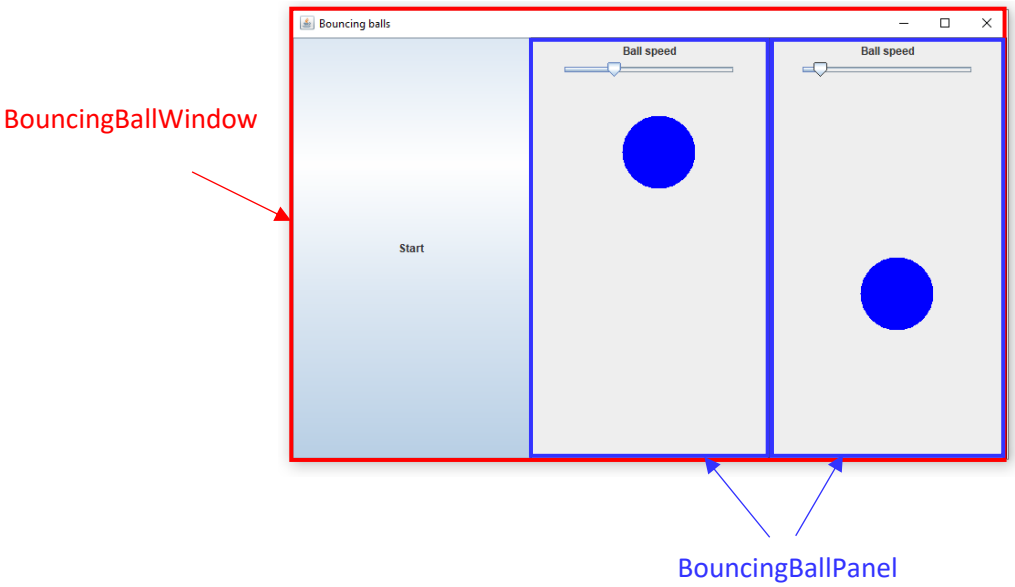
Cette alternative est illustrée dans la partie facultative ci-après.

9 – [Facultatif] Balles rebondissantes

Sur la page Moodle du cours, on vous fournit une archive balles_rebondissantes.zip qui contient une petite application Swing permettant d'animer des balles rebondissantes.

Cette archive contient deux classes :

- Une classe BouncingBallPanel, héritant de JPanel, qui fait bouger et dessine une balle rebondissant sur les bords horizontaux.
- Une classe BouncingBallWindow, héritant de JFrame, qui inclut un bouton et 2 BouncingBallPanel



L'objectif de cette partie est le suivant : faire en sorte que quand l'utilisateur appuie sur « Start », les deux balles rebondissent en parallèle, chacune à leur vitesse.

C'est la classe BouncingBallPanel qui se charge de l'animation. Comme elle hérite de JPanel, on ne peut pas la faire hériter de Thread. Pour créer un fil d'exécution qui se chargera de l'animation, il faut donc :

- Indiquer que BouncingBallPanel implémente l'interface Runnable
- Initialiser un attribut de type Thread de manière appropriée
- Démarrer ce thread dans la méthode animate().
  Attention : pour éviter de démarrer le thread plusieurs fois, il faut vérifier si le thread est déjà en cours d'exécution avec la méthode isAlive().

Bonus

Faites en sorte que l'utilisateur puisse changer la vitesse de la balle en manipulant le JSlider.

Indication : Pour écouter le changement d'état d'un JSlider, il faut implémenter l'interface ChangeListener