TP n°3 : les threads

1. Qu'est ce que c'est ?

Un « thread » est une tâche séquentielle, il possède un début, une fin, et à tout instant de son exécution correspond un point précis dans son flot d'instructions. Une application peut être « multi-threads », c'est à dire avoir plusieurs tâches actives simultanément. Par exemple, votre navigateur internet permet de visualiser une page WEB contenant une animation graphique; l'animation graphique se poursuit même si l'on consulte des menus du navigateur. Il y aurait alors au moins deux threads : l'un pour l'animation graphique, l'autre pour le navigateur et ses menus. Un « thread » est donc assimilable à un processus, sauf que les ressources qu'il utilise sont principalement allouées par le programme général de l'application (qui activera et arrêtera les threads selon ses besoins).

2. Héritons de la classe Thread préexistante

Première méthode pour utiliser les Threads, utiliser la classe **Thread** comme parente. Dans la classe **Thread**, la méthode **run ( )** contient toujours le corps de la séquence à exécuter. Cette méthode run dispose par défaut d'une séquence vide, donc rien n'est exécuté. Afin d'exécuter quelque chose dans un Thread, on va redéfinir la méthode run dans la classe qui en hérite. Définir la nouvelle classe A*ffichеTexte* qui hérite de Thread (**extends Thread**), possède un attribut ***texte*** de type String, et les deux méthodes suivantes :

- le constructeur AfficheTexte, avec pour paramètre une variable de type String : ce constructeur doit appeler le constructeur parent pour créer le Thread, puis recopier le paramètre dans l'attribut ***texte*** en utilisant le constructeur String.
- la méthode **void run ( )** qui affichera successivement chacune des lettres du texte transmis :

```
for (int c=0;c<texte.length();c++)
      System.out.print(texte.charAt(c));
System.out.println();
```

length() et charAt(position) sont des **méthodes** de la classe String donnant respectivement la longueur de la chaîne de caractères, et le caractère situé à une certaine position. Compilez cette nouvelle classe.

Créez une classe *Principale* contenant la méthode *main* à l'intérieur de laquelle vous testerez la construction d'un objet AfficheTexte. Puis vous appellerez la méthode start() sur cet objet pour tester le démarrage de votre thread. Compilez et exécutez *Principale*, ajoutez un second texte avec un second objet *AfficheTexte*, puis un troisième (en regardant l'exécution à chaque fois).

Essayez de relancer la même exécution plusieurs fois de suite. Vu, pour l'exécution en parallèle ?

3. « Implémentons » l'interface Runnable des Threads

Une autre méthode pour utiliser les Threads est de définir une classe qui réalise l'interface **Runnable**. Reprendre la classe AfficheTexte ci-dessus, et remplacer **extends Thread** par **implements Runnable**. Recompilez et réexécutez l'ensemble. Y a t-il un problème ? Oui, car notre classe AfficheTexte n'a plus de Thread, ni de méthode start (héritage du Thread remplacé par un certain nombre de méthodes d'interface). Pour corriger cela :

a) Définir une variable (attribut) de type Thread à l'intérieur de la classe AfficheTexte (par exemple, « ma_tache »).

b) Dans le constructeur de AfficheTexte, remplacer l'appel du constructeur de Thread **super()** par la création : **ma_tache=new Thread(this);** Il faut préciser la classe courante (avec this) comme Thread à construire, car sinon nous aurons un Thread par défaut, avec une méthode run( ) vide !

c) Enfin, créer la méthode start pour la classe AfficheTexte. A l'intérieur de cette méthode on appellera la méthode start() sur la variable ma_tache. (Étonnant, non ?) Réfléchissez-y. Si vous avez tout assimilé jusqu'ici, c'est que la programmation en Java, et dans les langages orientés objet, commence à venir...

4. Le cycle de vie d'un thread

Un Thread est avant tout un objet, il a besoin d'être créé :

- soit en appelant le constructeur Thread directement: **new Thread (...)**
- soit indirectement lorsqu'on hérite de la classe Thread : **super (...)**

Ensuite, le Thread doit être ***activé*** : c'est le rôle de la méthode **start ( ).** Aussitôt actif, le thread exécute automatiquement la méthode **run ( )** qui contient le corps de la tâche à exécuter. Comme le Thread est une tâche qui consomme du temps CPU, tout comme un processus, il est souvent utile de mettre le Thread dans l'état ***endormi*** lorsqu'il n'a pas quelque chose d'important à faire. Nous allons nous intéresser à deux méthodes pour endormir le Thread :

- **sleep(time)** où time est une valeur entière, endort le Thread pour *time* millisecondes
- **wait ( )** qui endort le Thread jusqu'à ce qu'il soit explicitement réveillé. (il existe aussi la forme wait (time) qui garantit son réveil au bout du temps spécifié).

Ces deux méthodes sont sujettes aux **InterruptedException** qui surviennent lorsque le Thread est réveillé : si le temps est écoulé (cas du sleep, ou du wait avec durée), ou si un message explicite de réveil lui parvient (cas de wait sans paramètre). Il faut donc les utiliser dans un bloc try – catch.

La forme la plus usuelle pour inclure une attente est donc la suivante :

```
public void run () {
      try {
      //attente pendant 1 seconde
      } catch (InterruptedException exception) { } // ne rien faire
      l'exception est rattrapée et on va continuer en séquence
```

Ceci a permis d'interrompre la tâche, et de laisser le processeur à d'autres tâches. Les méthodes de réveil explicite sont : **notify ( )** : réveille le Thread qui attend depuis le plus longtemps; et **notifyAll ()**.: réveille tous les Threads (endormis avec wait)

Faites en sorte que les caractères des threads s'affichent toutes les secondes. Vous devez pour cela utiliser la méthode sleep sur votre thread lors de l'affichage du texte.

5. Utilisation de threads pour faire des animations

Télécharger le code source correspondant à cette question. Il s'agit d'une application graphique où une balle rebondie sur les rebords horizontaux de votre fenêtre. La classe FenetreGraphique s'occupe de l'affichage de l'interface graphique : un bouton et deux objets PanelDessin qui permettent d'afficher 2 balles.

- dans la classe FenetreGraphique, ajoutez la fonctionnalité permettant d'appeler la méthode startAnim() sur vos objets PanelDessin lors du clic sur le bouton.
- Lancez l'interface et testez le clic sur le bouton. Plusieurs observations peuvent alors être effectuées :
    - l'affichage des balles ne s'effectue qu'à la fin (i.e. lorsque y a atteint une certaine valeur)
    - le déroulement est linéaire : on s'occupe d'abord du y de la première balle puisque lorsque la boucle est terminée, on passe à la seconde.

Ce comportement linéaire n'est pas acceptable dans le cadre d'animations dans une interface graphique. La solution à ce problème est l'utilisation de threads dans la classe PanelDessin.

- Implémentez l'interface Runnable dans votre classe PanelDessin, déclarez un attribut de type Thread dans votre classe. Initialisez-le dans le constructeur.
- Redéfinissez la méthode run de l'interface Runnable dans votre classe. Vous devez, dans cette méthode, endormir votre thread quelques millisesondes (40 ms pour avoir un affichage à 25 images par seconde) grâce à la méthode sleep, puis appeler la méthode bougeBalle.
- Faites en sorte que le clic sur le bouton de l'interface démarre le thread et observez alors le comportement de votre programme.

Vous allez enfin gérer l'utilisation des sliders de l'interface pour gérer la vitesse de chute de la balle. Pour cela, vous devez implémenter l'interface ChangeListener et écouter les évenements se produisant sur votre slider.