TP 1 : BallWorld

Nous allons commencer à partir d'un programme qui ouvre une fenêtre dans laquelle 4 balles colorées doivent bouger en rebondissant contre les murs.

Programme initial

Télécharger `BallWorld.zip` dans votre répertoire de travail et décompressez le fichier. Vous y trouverez trois classes :

- `Ball.java`
  Une instance de cette classe est une balle, qui vit dans un `BallWorld`. La balle a la capacité de se dessiner (avec la méthode `draw`) dans un contexte graphique. En outre, à la création (dans son constructeur), la balle s'enregistre elle-même dans le monde où elle habite en appelant la méthode `BallWorld.addBall`.
- `BallWorld.java`
  Une instance de cette classe est un monde, qui peut contenir plusieurs balles, stockés dans un `ArrayList`. Un monde est une sous-classe de `JPanel`, la classe Swing utilisée comme surface de dessin. Par conséquent, le monde peut se dessiner (méthode `paintComponent`) en demandant à toutes ses balles de se dessiner. Les détails graphiques de Swing sont de moindre importance pour les exercices, mais attention toutefois à l'utilisation de `SwingUtilities.invokeLater` qui permet d'exécuter un bout de code depuis le `Thread` de l'IHM Swing (ce qui, en occurence, évite les accès à l'array `balls` depuis plusieurs `Threads`).
- `Balls.java`
  Cette classe contient la méthode `main`, qui crée un monde et y insère 4 balles.

Compiler et exécuter le programme:

```
$ javac *.java
$ java Balls
```

Regardez bien le code des classes et assurez-vous que vous le comprenez.

Exercice 1: Faire bouger les balles

Comme vous constatez, les balles ne bougent pas encore. Votre première tâche est de les faire bouger indépendamment les unes des autres. Pour cela, vous allez devoir convertir la classe `Ball` en une sous-classe de `Thread` et définir sa méthode `run` comme une boucle infinie où le ballon met à jour sa position et dort pendant une période égale a `period`.

En sachant que la méthode `BallWorld.paintComponent` est appelée par le thread GUI de Swing, il est possible en principe qu'une balle soit dessinée à une position qui ne correspond pas à sa trajectoire (celle calculée par `move`), du fait que des attributs

partagés par plus d'un *thread* (notamment `xpos`, `ypos`). Pouvez-vous identifier dans quel cas cela arrive ? Modifier le code (en ajoutant éventuellement des variables) de manière que, chaque fois que cela se produit, la balle double sa taille.

Ajouter maintenant les protections nécessaires (clauses `synchronized`) pour les accès en lecture et en écriture de attributs partagés par plus d'un *thread*, de manière à ce que la situation décrite ci-dessus n'arrive plus.

Exercice 2: "Tuer" les balles

Ajouter la fonctionnalité suivante à votre application : quand l'utilisateur clique à l'intérieur de la fenêtre, un autre `Thread` est lancé qui tue les balles dans un ordre et à des intervalles de temps aléatoires.

Tuer une balle signifie que sa méthode `run` doit se terminer toute seule. Cela doit se faire en faisant la boucle se terminer par une condition qui devient fausse, et et non en appelant la méthode `stop` sur le `Thread` (cette méthode est obsolète) ! En outre, la balle doit être retiré de `BallWorld` (par un code thread-safe semblable à `addBalls`).