Programmation Orientée Objet
TP 3 : programmation événementielle.

https ://dpt-info.u-strasbg.fr/~kvanhoey

Introduction

Le but de ce TP est de manipuler des threads en Java. Avant de débuter ce TP, soyez au point sur la théorie des threads en Java.

1 Compte à rebours (threads indépendants)

Implémentez la classe `CompteARebours`, dérivant de `Thread`. Un objet `CompteARebours` a un identifiant unique, et une durée (en nombre d'étapes). Une fois un compte à rebours lancé (avec *start()*), il doit à chaque étape, attendre un certain temps (avec *sleep()*), et afficher son identifiant et le nombre d'étapes restantes.

Codez une fonction `main` créant plusieurs comptes à rebours, et lancez-les. Exécutez plusieurs fois votre fonction pour observer les différents comportements. Vous pouvez également essayer de modifier la priorité de chaque thread et les durées des comptes à rebours.

2 Producteur/Consommateur

Cet exercice est un exemple classique de synchronisation de ressources. D'un côté, nous avons un ensemble de producteurs qui produisent des ressources. D'un autre côté, nous avons bien évidemment les consommateurs, qui vont prendre de la ressource dans la réserve (quand il y en a). Des exemples classiques sont de la production/consommation de matières premières (blé, pétrole, ...) par exemple, ou encore la gestion d'accès à une imprimante. Il s'agit grosso modo de tout ce qui peut être considéré comme une ressource produite en quantité limitée et pour laquelle les clients doivent attendre pour s'en procurer.

Programmez une classe `Réserve`, qui stockera le nombre de ressources dans la réserve. Cette classe aura également deux méthodes *prendre* et *mettre*, qui respectivement ajoutent et retirent une ressource. Dans le cas où on essaie de prendre une ressource alors qu'il n'y en a pas, on doit attendre qu'un producteur en rajoute une. De même, la réserve n'a pas une taille illimitée. Les producteurs doivent également attendre s'ils veulent rajouter une réserve alors qu'elle est remplie.

Programmez enfin les classes `Producteur` et `Consommateur`, dérivant de `Thread`. La classe `Producteur`, une fois lancée, produit une ressource (appelle la méthode *mettre* de la réserve), attend un certain temps (aléatoire), et recommence (à l'infini). La classe `Consommateur`, une fois lancée, récupère une ressource (appelle la méthode *prendre*), attend un certain temps (aléatoire),

et recommence (à l'infini).

Écrivez un programme de test, avec une réserve et un certain nombre de producteurs/consommateurs. Modifiez vos méthodes pour afficher l'état de la réserve à chaque opération. Essayez de modifier vos différents paramètres (nombre de producteurs et de consommateurs, taille limite de la réserve) pour observer les différences de comportement.

3 Course dans un château

Un groupe d'amis se lancent un défi : entrer dans un château et en ressortir le plus rapidement possible. Les conditions sont les suivantes :

- Le château a plusieurs entrées, plusieurs sorties, plusieurs étages, des passages secrets... En d'autres termes, vous représenterez le château par un graphe, dont les nœuds sont les salles, et les arêtes sont les portes (et passages) entre les salles.
- Il ne peut y avoir qu'un seul joueur par salle à chaque instant.
- Lorsqu'un joueur entre dans une salle, il doit se reposer quelques instants.
- Pour passer d'une salle à une autre, un joueur doit d'abord décider (aléatoirement) quelle porte il prend, puis doit éventuellement attendre que la salle se libère s'il y a déjà un autre joueur.
- Lorsqu'un joueur atteint une salle de sortie, il s'arrête et sort du château (pour ne pas bloquer définitivement la sortie).