BUI Vu Minh Duc

INF2-A

**TP1 Mobile :**

**TP2 synchronized, sémaphore :**

**Exercice 1 :**

Dans un premier temps, nous utilisons la méthode synchronized() qui permet de faire l’attente d’un accès à une section critique (ref. diapo 9 cours 2).

Pour cela, nous avons besoin de la présence de l'objet d'exclusion mutuelle. Cet objet doit être static car à chaque instanciation, un nouvel objet de type Exclusion est créé.

**Exercice 2 :**

Ensuite, l’exercice 2 consiste à utiliser les méthodes wait() et signal() qui a pour but de faire le même traitement que synchronized.

Pour cela, on a besoin tout d’abord instancier 1 variable static de type semaphoreBinaire, même raison que la variable de type Exclusion. En revanche, on a besoin d’attribuer 1 valeur à cette variable (soit une variable supérieure à 0 soit 0).

Ensuite, nous allons faire l’attente l’accès grâce à la méthode syncWait(). Si la valeur = 0, c’est-à-dire qu’il y a un thread en cours d’exécution et inverse. C’est pour cette raison que si on commence par une valeur de 0, le programme sera bloqué. Une fois le thread est terminé, la valeur incrémentée de 1 pour dire qu’il est libre de lancer une nouvelle thread.

Pour aller plus dans le détail, nous analysons la classe sémaphore que la classe sémaphoreBinaire hérite. Dans cette classe, nous nous intéressons à 2 méthodes syncWait() et syncSignal().

* syncWait() est composée d’un try/catch. En général, si la valeur = 0, on fait l’attente. Si ce n’est pas le cas, on décrémente la valeur.
* syncSignal() : vu que la valeur = 0 à cause de la méthode précédente (syncWait) donc le travail de syncSignal est d'incrémenter la valeur pour que cette valeur revienne à l’état initial. Ce qui a pour but de notifier qu’il est libre de lancer une nouvelle thread aussi.
* 