IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP4 – Le tampon borné

Fondements théoriques du développement des logiciels concurrents

Objectifs

À la fin de l'activité, les élèves devront être capables de :

- définir et d'activer des *threads* en Java selon l'approche par composition en utilisant un IDE moderne (Eclipse) ;
- utiliser les fonctions basiques du débogueur d'un IDE moderne (Eclipse) pour analyser la trace d'un programme concurrent simple ;
- programmer l'exclusion mutuelle en utilisant un verrou.

Préambule

Afin de faciliter la mise en pratique des différents concepts, une application existante vous est fournie. Tout au long du TP, vous aurez à la modifier pour réaliser les exercices proposés. Le code source de l'application est accessible sur Moodle à l'adresse habituelle.

Contexte

Dans ce TP nous allons mettre en application les concepts de *thread* et d'exclusion mutuelle vus en cours et en séance de travaux dirigés. Le problème sur lequel nous le ferons sera celui du tampon borné déjà présenté en TD.

Pour rappel, dans le problème du tampon borné, deux types d'*entités*, les producteurs et les consommateurs, partagent un tampon de taille fixe. Les producteurs remplissent le tampon avec des données. Simultanément, les consommateurs retirent des données du tampon pour les traiter. Par exemple, un clavier produit des caractères qui sont consommés par le logiciel qui gère l'affichage à l'écran.

Des contraintes s'appliquent à un tel système. Les producteurs ne doivent pas ajouter de données lorsque le tampon est plein et les consommateurs ne doivent pas récupérer des données d'un tampon vide.

Exercice 1 (*Un modèle non concurrent*)

La figure 1 correspond au diagramme de classes de l'application qui vous est fournie. Le tampon borné est représenté par un objet de type `Stock`. Nous supposerons, comme lors du TD, que les consommateurs consomment les produits dans un ordre LIFO (*« Last In First Out »*).

▷ **Question 1.1 :**
Créez une classe `Main` qui dans sa méthode `main` crée deux producteurs (qui produisent un produit chacun) puis deux consommateurs qui consomment ces produits.

▷ **Question 1.2 :**
Exécutez votre classe `Main` en utilisant le débogueur d'Eclipse. Pour cela,

- **Créez un point d'arrêt avant la création du premier produit par un producteur (double click sur la barre à gauche de la ligne) ;**
- **Assurez vous que la perspective [1] de débogage est utilisée (icône ). Les onglets *Debug*, *Variables* et *Breakpoints* doivent être visibles ;**
- **Sélectionnez votre classe puis utilisez l'icône  pour lancer l'exécution.**

L'exécution est arrêtée sur le point d'arrêt que vous avez choisi. Sélectionnez le *main* dans l'onglet *Debug* et vérifiez la valeur des variables dans l'onglet *Variables*. Demander ensuite la reprise de l'exécution pas à pas (instruction par instruction) à l'aide de l'icône ).

Quelle est la trace d'exécution observée ?

Exercice 2 (*Des processus concurrents*)

Nous allons maintenant modifier l'application pour créer un système concurrent composé de producteurs et de consommateurs tels que vus dans le TD « tampon borné ».

▷ **Question 2.1 :**
Appliquer les modifications vues en TD pour les consommateurs et les producteurs deviennent des processus concurrents.

▷ **Question 2.2 :**
Proposer le code Java d'activation d'un consommateur et d'un producteur dans la méthode `main` de votre classe `Main`.

▷ **Question 2.3 :**
Exécuter votre classe `Main` en utilisant le débogueur d'Eclipse. Poser des points d'arrêt au début des méthodes `add` et `remove` dans la classe `Stock`. Lancer le débogueur et exécuter le programme pas à pas. Quelles sont ses traces d'exécution ?

Exercice 3 (*L'accès exclusif au tampon*)

Dans la solution précédente, les accès au tampon se font de manière indépendante : un producteur peut donc être en train d'exécuter sa méthode `produce` qui ajoute un produit dans le tampon, lorsque le consommateur exécute sa méthode `consume` qui récupère un produit du tampon.

1. Une perspective est un ensemble prédéfini de vues.

▷ **Question 3.1 :**
À l'aide du débogueur d'Eclipse, exécutez les traces :
- *`produce("banane")` en totalité ; ligne 38 [`if (p==null)`] de la méthode `add` de `Stock` ; `consume()` ; ligne 41 [`if (!isFull())`] de la méthode `add` ; ligne 42 [`this.content[t`] de la méthode `add`*
- *`produce("banane")` ; `produce("banane")` ; `consume()`*

Quel produit consomme le consommateur pour les deux traces ?

Pour éviter le problème précédent, l'ensemble des opérations de manipulation du tampon (section critique) doivent être exécutées en exclusion mutuelle.

▷ **Question 3.2 :**
Utiliser un verrou pour programmer l'accès exclusif à la section critique. Les verrous sont définis dans le paquetage `java.util.concurrent.locks`. L'interface est `Lock` et une réalisation suffisante ici est `ReentrantLock`.

▷ **Question 3.3 :**
Exécutez maintenant les deux traces précédentes. Que constatez-vous ? Comment expliquez vous ce comportement ?

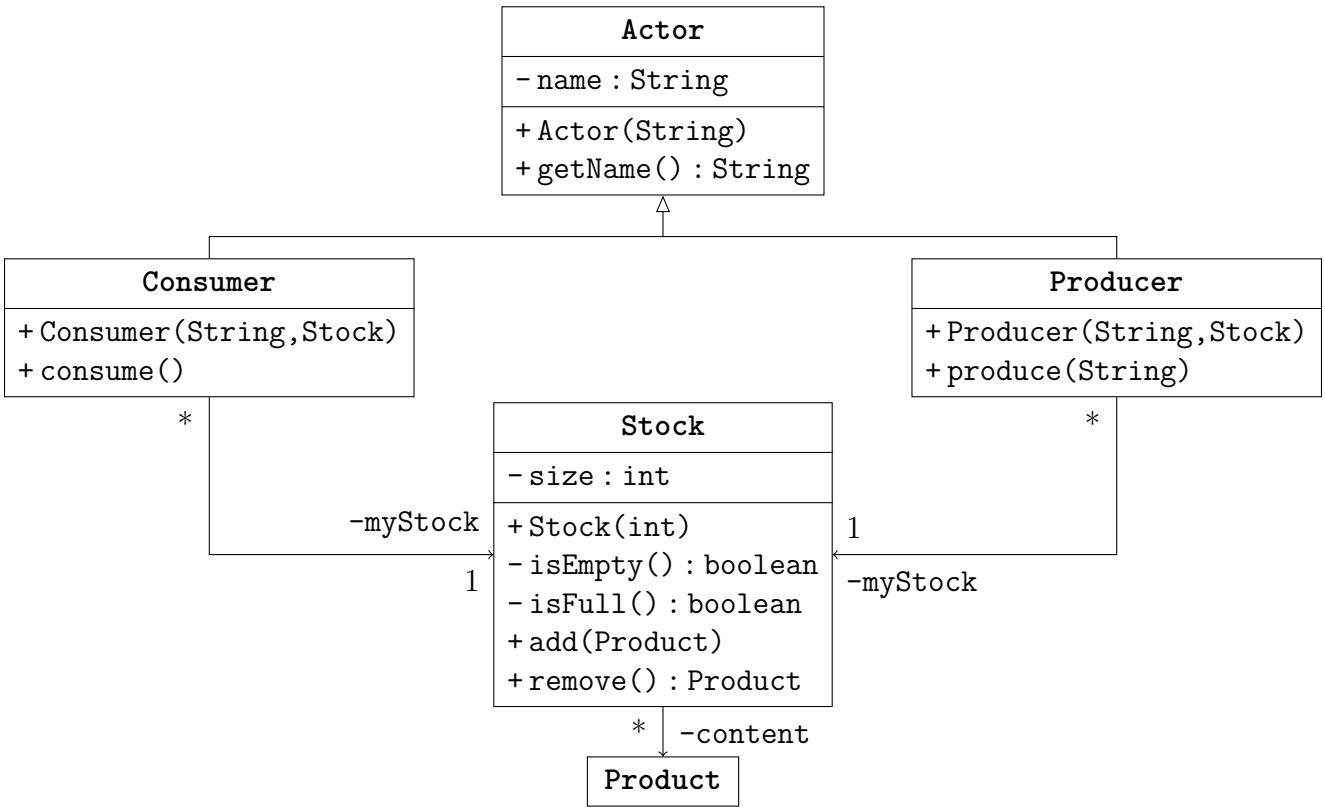

FIGURE 1 – Diagramme de classes d'une solution au problème du tampon borné