

# Bienvenue dans un monde parallèle

## 1 Construction d'une usine

Vous trouverez les sources de ce TP dans le canal CSE de votre groupe Teams.

### 1.1 Les stocks

La classe `Stock` représente un stock de pièces. Deux méthodes d'accès `stocker()` et `destocker()` permettent respectivement d'ajouter et de retirer une pièce du stock. La méthode `afficher()` affiche le nom du stock et le nombre de pièces actuellement en stock. Le constructeur prend le nom du stock et le nombre initial de pièces en paramètres.

### 1.2 Les ateliers

La classe `Atelier` représente un atelier de transformation. La méthode `transformer()` retire un élément d'un stock *A*, attend 100 ms<sup>1</sup> afin de simuler la transformation de la pièce, puis ajoute un élément au stock *B*. La méthode `travailler()` effectue *n* transformations successives. Le constructeur de la classe `Atelier` prend en paramètre des références sur les stocks *A* et *B*, ainsi que le nombre *n* de transformations à effectuer.

### 1.3 L'usine

La classe `Usine` représente une usine avec, initialement deux ateliers. Elle dispose d'un stock de pièces à transformer (doté initialement de 10 unités), ainsi que d'un stock de pièces finies (initialement vide). Chacun des deux ateliers transforme initialement 5 unités. La méthode `fonctionner()` fait travailler successivement les deux ateliers et affiche l'état des stocks à la fin des travaux.

- Observez la classe `Stock`
- Observez la classe `Atelier`
- Observez la classe `Usine`
- Ecrivez une fonction principale simple dans la classe `Usine` qui permette de créer une usine et de le faire fonctionner.
- Compilez, exécutez, vérifiez le résultat.

Jusqu'à la fin du TP, votre classe principale sera l'usine. Vous pourrez vous aider de schémas pour mieux prendre en main le sujet et surtout bien comprendre les différents problèmes de synchronisation à résoudre.

---

1. `Thread.sleep(100)`; en Java

## 2 Plus de productivité

### 2.1 Travail en parallèle

*Pour cette question, on ne traitera pas les problèmes de synchronisation dans un premier temps.*

Actuellement, il faut une seconde (environ) pour transformer les 10 pièces alors qu'une demi seconde, potentiellement, devrait y suffire si les deux ateliers travaillent en parallèle.

- Effectuez toutes les modifications nécessaires pour faire travailler les ateliers en parallèle.
- Identifiez les threads, et transformez ces objets en Threads
- Démarrez les de façon asynchrone dans l'usine
- Recompilez, exécutez, vérifiez le résultat.
- Trouvez le problème, expliquez le et corrigez le.

### 2.2 Plus vite, mais pas trop.

A ce stade, vous êtes peut-être convaincus que votre programme est correct... Peut être observez-vous déjà un nouveau problème, peut être pas. Cela dépend de votre ordonnanceur. Pour être sûr :

- Portez la capacité de production de votre usine à mille pièces (cinq cent par atelier),
- Pour accélérer l'exécution, commentez l'appel à `sleep()` dans la méthode `transformer()`,
- Recompilez, exécutez, observez le résultat.
- Trouvez le problème et corrigez le.

### 2.3 Parallélisme et non déterminisme

Parallélisme signifie

- Revenez à une production de 10 pièces et **décommentez** l'appel à `sleep()`,
- Faites les modifications nécessaires pour pouvoir suivre l'état des stocks en direct à l'aide de traces du style : `Le Thread-2 vient de dépiler. Le stock de départ contient 9 piece(s).`<sup>2</sup>,
- Recompilez, exécutez, observez le résultat.
- Est-on sûr d'obtenir la même trace d'exécution à chaque fois ? Pourquoi ?

## 3 Réorganisations

### 3.1 Réorganisation du travail

*Pour cette question, on ne traitera pas les problèmes de synchronisation dans un premier temps.*

On décide de spécialiser les ateliers. Nous allons donc modifier la répartition du travail entre les deux ateliers. Au lieu de faire tout le travail sur la moitié du stock, chaque atelier va devoir effectuer une moitié du travail sur la totalité des 10 pièces. En d'autres termes, chacune des dix pièces devra passer par les deux ateliers. Le premier atelier puise dans le stock de départ A et alimente un stock intermédiaire B. Le deuxième atelier puise dans ce stock intermédiaire B et alimente le stock des pièces finies C.

---

2. Le nom du thread courant est accessible grâce à l'appel `Thread.currentThread().getName()`.

- Effectuez les modifications nécessaires,
- Recompilez, exécutez plusieurs fois, vérifiez le résultat,
- Observez attentivement les traces.

### 3.2 Une petite sieste, ça ne fait pas de mal

Vous avez sûrement observé que le stock intermédiaire peut devenir négatif. En effet, le deuxième atelier va puiser une pièce dans ce stock avant même que le premier atelier ne l'alimente. En fait, si un stock est nul, celui qui vient y puiser devrait partir faire une petite sieste. À charge pour celui qui alimente le stock d'aller réveiller un des dormeurs.

- Insérez les synchronisations nécessaires. Réfléchissez. Si un `if` suffit, n'écrivez pas un `while`. De même, si un `notify()` simple suffit, n'écrivez pas un `notifyAll()`,
- Recompilez, exécutez, vérifiez les traces et le résultat.

### 3.3 Jamais deux sans trois

Une nouvelle fois, nous allons essayer de prendre le code en défaut. Pour cela, nous allons créer un troisième atelier.

- Donnez la moitié du travail de l'atelier 2 à un nouvel atelier,
- Lancez l'atelier 1 en dernier,
- Recompilez, exécutez, vérifiez les traces et le résultat,
- Trouvez le problème, expliquez-le et corrigez-le.

### 3.4 Toujours plus de productivité

Pour améliorer la rentabilité, la direction décide d'augmenter la cadence et de travailler en flux tendu. La transformation sera instantanée et le stock intermédiaire sera limité à une seule pièce. L'atelier 1 sera dédoublé.

- Pour rendre la transformation quasi instantanée, commentez l'appel à `sleep()` et supprimez tous les affichages intermédiaires,
- Ajoutez un nouveau paramètre au constructeur de la classe `Stock` représentant le nombre maximum de pièces qu'il est possible de stocker,
- Ajoutez les synchronisations nécessaires en cas de dépassement de cette valeur,
- Dédoubez l'atelier 1,
- Recompilez, exécutez, attendez... mais pas trop longtemps quand même!
- Trouvez le problème, expliquez-le et corrigez-le.