FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

Info-F202 : Projet Java – Une usine multi-tâche – seconde session

Vandy Berten

1 Préambule

Ceci est le projet Java pour l'examen de seconde session 2012–2013 pour le cours de Langages de Programmation II (INFO-F202). Il s'agit d'une version légèrement modifiée du projet de première session. Les principales modifications sont en ***gras italique***.

2 Idée générale

L'idée de ce projet est de simuler le comportement d'un atelier de construction d'un produit dont la réalisation nécessite un certain nombre d'étapes, dont certaines peuvent s'exécuter en parallèle, mais devant respecter des contraintes de précédence bien définies. Chacune de ces étapes consiste en le passage par un *poste de travail* (workstation) sur lequel sera effectuée une tâche définie. Le processus de fabrication sera défini par un graphe de postes de travail, qui seront manipulés par un groupe d'artisans agissant de façon partiellement coordonnée.

3 Exemple

Pour illustrer le processus, supposons un chaîne de production composée des postes de travail A, B, C, D et E, dont les dépendances sont représentées à la Figure 1. On considère un exemple avec trois artisans α_1, α_2 et α_3 (qui seront représentés par des *threads*), qui ont pour objectif de réaliser trois objets finis O_1, O_2 et O_3. Les artisans sont capables de manipuler chacun des outils et tant que l'objet qu'ils ont commencé n'est pas fini, ils ne passeront pas au suivant. Mais il se peut qu'un artisan ne réalise qu'une partie des étapes de la réalisation d'un objet. Par ailleurs, un poste de travail ne peut pas servir à la conception de l'objet O_i tant que l'objet O_{i-1} n'y est pas passé, les pièces étant numérotées et empilées en ordre inverse sur le poste. Par contre, un objet peut très bien être réalisé en même temps sur deux postes différents s'ils ne sont pas dépendants l'un de l'autre. On suppose dans ce cas qu'il s'agit d'un travail sur des parties distinctes de l'objet final.

Supposons que les artisans α_1 et α_2 sont chargés de la réalisation de O_1 et O_2 et que α_3 réalise O_3. Un exemple de déroulement pourrait être le suivant :

– L'artisan α_1 démarre avec l'objet O_1 au poste A, le termine puis démarre C ;

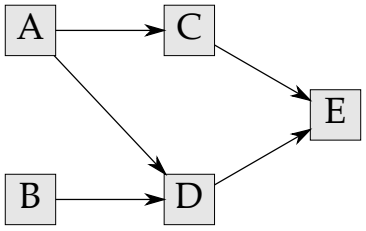

Figure 1 – Exemple de graphe de précédence.

- L'artisan α_3 démarre avec l'objet O_3 au poste A. Ce poste est libre, mais n'étant pas encore prêt pour lui (O_2 n'y est pas encore passé), il attend ;
- L'artisan α_2 démarre avec l'objet O_1 au poste A. Il se rend compte que le travail est déjà terminé, et passe au poste B, puis D ;
- L'artisan α_1 termine en C. Il veut réaliser B (mais le saute car l'objet y est déjà passé), puis D. Il est mis en attente, car le poste est occupé par α_2. Vous remarquerez qu'en termes de threads, il s'agit d'une attente différente que celle décrite plus haut ;
- L'artisan α_2 termine D, passe à C qu'il ignore, puis passe à E et le termine ;
- L'artisan α_1 se « réveille », se rend compte qu'il est trop tard pour D, ainsi que pour E ;
- L'artisan α_2, en grande forme, réalise sans être interrompu l'objet O_2 sur A, B, C, D puis E ;
- L'artisan α_3, qui attendait la réalisation de O_2 en A, est réveillé, et peut réaliser O_3, en passant, par exemple, par A, B, D, C, E ;
- L'artisan α_1 se rend compte qu'il n'a plus rien à faire pour O_2, et a donc terminé son travail.

4 Fonctionnement général

Chaque artisan maintient une liste $\mathcal{L}$ des postes par lesquels il doit encore passer pour la réalisation de l'objet en cours. Au départ, la liste contient tous les postes ne possédant pas de prédécesseurs (A et B dans l'exemple de la figure 1). Pour chaque objet O qu'il devra réaliser, les artisans procéderont de la façon suivante :

1. On commence par choisir un poste de travail $\mathcal{W}$ parmi sa liste $\mathcal{L}$ (soit le premier, soit au hasard).
2. Si l'objet qui précède O n'a pas encore été traité sur le poste $\mathcal{W}$, il conviendra d'attendre qu'il soit prêt à le traiter. ***Dans ce cas, l'artisan examinera les objets suivants dans la liste, jusqu'à en trouver un qu'il peut traiter. Pour éviter une attente active dans le cas où aucun objet de la liste $\mathcal{L}$ ne peut être traité directement, après trois tentatives infructueuses, l'artisan « s'endormira », en attendant qu'un autre artisan ait traité l'objet précédant le dernier objet considéré.***
3. Si $\mathcal{W}$ est prêt, c'est-à-dire que le dernier objet qui y a été traité est celui qui précède O, alors il effectue la tâche associée au poste. Il vérifie ensuite qu'il n'y a pas d'artisans faisant la « file » à ce poste, et les réveille le cas échéant. Il rajoute ensuite à sa liste $\mathcal{L}$ tous les successeurs de $\mathcal{W}$ dont l'ensemble de prédécesseurs a déjà traité l'objet O.
4. S'il s'avère que l'objet O a déjà été traité sur le poste de travail $\mathcal{W}$ (par un autre artisan chargé de la réalisation du même objet), il n'est plus nécessaire de le faire. On l'ignorera simplement, sans devoir mettre à jour la liste $\mathcal{L}$. On peut le faire sans risque : si la tâche a déjà été effectuée, c'est qu'un autre thread a placé les successeurs potentiels dans sa propre « todo list ».

Vous remarquerez qu'il s'agit en fait d'un tri topologique imbriqué dans une généralisation de l'exemple du « producteur-consommateur » vu au cours.

5 Les composants

Les sections suivantes définissent les différentes classes et interfaces que l'on vous demandera d'implémenter.

5.1 Classe `Article`

Cette classe représente les objets qui seront fabriqués. Les artisans (`Worker`) en recevront une liste (présumée triée) lors de leur initialisation, et passeront d'un poste de travail (`Workstation`) à l'autre. Pour les besoins de la simulation, seul un message sera ajouté aux articles (par exemple, « Worker 4 processed Workstation 3 », ou « Worker 1 skipped Workstation 2 »), qui possèdera donc une liste de String, en guise de « construction ». Ces articles se créeront par ailleurs un identifiant entier, unique, attribué automatiquement de façon croissante et continue à la construction, permettant de savoir si un article peut être traité à un poste de travail, c'est-à-dire que tous les articles d'indice inférieur y ont été traités.

5.2 Interface `Task`

L'interface `Task` contiendra une unique fonction, `process(Article article)`. Elle simulera le travail réel effectué à un poste de travail sur l'objet `article`. Il conviendra d'implémenter au moins une classe concrète sur base de cette interface, mais son code ne doit pas être pertinent : un affichage, un « `sleep()` » de quelques milli-secondes, ...

5.3 Classe `Workstation`

Cette classe représentera un poste de travail. Elle recevra à la construction un objet de type `Task`, et devra gérer un entier représentant le dernier identifiant d'objet traité sur ce poste. Un appel à la méthode `process(Article article)` d'un objet de cette classe invoquera la méthode du même nom de la tâche associée, et mettra à jour l'identifiant du dernier article traité.

5.4 Classe `Graph`

Cette classe, générique, contiendra essentiellement deux éléments : une liste d'objets génériques `Node` (tous d'un même type, grâce à la généricité), ainsi qu'une structure permettant de représenter les dépendances entres les objets. Une implémentation simple conviendra, on ne cherchera pas à tout prix à avoir une structure performante. On y trouvera également quelques méthodes du type `List<Node> getInitiators()` ou `List<Node> getSuccessors(Node ws)`.

Là où le graph sera créé, on instanciera donc un `Graph<WorkStation>`.

Notez que la méthode `boolean isReady(Workstation ws, Article article)`, vérifiant que tous les prédécesseurs de `ws` ont déjà traité `article` et présente dans cette classe dans la première version du projet, devra être déplacée ailleurs.

5.5 Classe `Worker`

On trouve dans cette classe le travail des artisans. Il s'agira d'un thread, qui recevra à la construction une liste d'article à réaliser (`List<Article>`), ainsi que le graphe de précédence, et appliquera l'algorithme défini plus haut. Sur chaque opération effectuée sur un article, il ajoutera un message dans son « log » (cf. description de `Article`). Par ailleurs, chaque artisan maintiendra également une sorte de « carnet de bord » (un String, ou une liste de String), où il y notera ses actions. Cela permettra, à la fin de l'exécution, d'afficher à la fois les messages des articles et ceux des artisans, sans qu'ils soient entrelacés et illisibles, à cause de l'exécution parallèle des threads.

5.6 Méthode `main()`

On ajoutera une méthode `main()` permettant de tester le code. L'emplacement de la méthode est laissé à la libre appréciation des étudiants.

6 Consignes

Le projet est à remettre une semaine avant l'examen oral, d'une part sur l'Université Virtuelle (en envoyant individuellement chaque fichier java, pas dans un fichier zip), d'autre part sur papier, au secrétariat étudiants du département d'Informatique.

Il servira de base à l'examen oral qui se déroulera en session. Les étudiants s'inscriront auprès du secrétariat étudiants.