# TD Étude d'une chaîne d'usinage

Pour mesurer les performances d'une chaîne d'usinage, il est moins coûteux de la simuler par programme que de la construire réellement. Le but du problème est donc de simuler le fonctionnement d'une telle chaîne.

## 1 Fonctionnement d'une chaîne

Une chaîne d'usinage est constituée d'une suite de **postes de travail** terminée par exactement un **poste de sortie**. Chaque poste réalise une partie du travail à effectuer sur les pièces. Pour être complètement traitée, une pièce passe successivement par chacun des postes de travail de la chaîne (du premier au dernier) puis est déposée sur le poste de sortie.

Chaque poste de travail traite au plus une pièce à la fois et les postes peuvent travailler en parallèle : si la chaîne comporte $N$ postes de travail, alors on pourra avoir jusqu'à $N$ pièces en cours de traitement. Tant qu'un poste de travail est occupé avec une pièce, il ne pourra pas commencer à en traiter une nouvelle ; par ailleurs, les pièces arrivent sur le poste de sortie dans l'ordre dans lequel elles sont entrées dans la chaîne : une pièce ne peut en doubler une autre.

Le temps de traitement d'une pièce par un poste de travail dépend soit du poste (ce temps alors le même pour toutes les pièces traitées par ce poste), soit de la pièce elle-même (c'est la pièce qui détermine le temps de traitement et ce temps est alors le même quelque soit le poste). Quand une pièce arrive sur un poste de travail, elle attend d'abord que les pièces arrivées avant elle aient été traitées. Cette attente, éventuelle, ne bloque pas le poste précédent qui peut continuer à traiter des pièces[1].

## 2 Mesures à faire

Pour une chaîne d'usinage, vide de pièces, et un certain nombre d'entrées datées de pièces dans cette chaîne, le responsable de l'usine souhaite connaître les informations suivantes :

1. Informations par poste de travail :
   - le nombre de pièces qu'il a traitées.
   - les durées d'activité et d'inactivité du poste[2] entre la date de début de traitement de la première pièce et celle de fin de traitement de la dernière pièce. Ces durées seront nulles si aucune pièce n'a été traitée.
2. Informations connues du poste de sortie :
   - le nombre de pièces complètement traitées,
   - et si ce nombre n'est pas nul :
     - le maximum des durées d'attentes des pièces ayant traversé cette chaîne,
     - la moyenne des attentes.

## 3 Réalisation

Les dates et les durées seront exprimées en nombres entiers d'unités de temps[3]. On appellera **mouvement** le fait qu'à une certaine date, une pièce arrive à un poste pour être traitée (cette date ne correspond pas forcément au début du traitement de la pièce par le poste, car il peut y avoir déjà d'autres pièces en attente de traitement pour ce poste). Évidemment, pour que la simulation donne des résultats corrects, il faut que le programme *joue* les mouvements dans leur ordre chronologique. A un instant donné, plusieurs pièces peuvent se trouver dans la même chaîne : pour respecter l'enchaînement de la simulation, il faudra toujours considérer le mouvement le plus ancien. On utilisera un **échéancier** unique pour gérer les mouvements qui n'ont pas encore été joués. Auprès de cet échéancier, on pourra obtenir, en l'en supprimant, le mouvement dont la date d'arrivée à un poste est la plus ancienne (c'est-à-dire le prochain mouvement à jouer), et on pourra y ajouter de nouveaux mouvements.

Nous avons alors les classes suivantes : `Piece`, `Poste`, `Mouvement` et `Echeancier`.

[1] On considérera, pour simplifier, que le nombre de pièces en attente d'un poste de travail n'est pas borné ; autrement dit, un poste lent ne peut pas empêcher ses prédécesseurs plus rapides de travailler.

[2] Un poste est actif quand il traite une pièce.

[3] L'unité de temps n'est pas précisée.

### 3.1 Les méthodes de `Piece`

`public int dureeAttente()` : Renvoie la durée totale d'attente de la pièce.

`public void attend(int duree)` : Augmente la durée totale d'attente de **duree**.

`public int dureeTraitement()` : renvoie la durée de traitement de la pièce, utile dans le cas d'un poste dont la durée de traitement dépend de la pièce traitée. La valeur de **dureeTraitement()** sera fixée à la construction de la pièce.

### 3.2 Les méthodes de `Mouvement`

`public int date()` : La date à laquelle ce mouvement doit être réalisé.

`public Piece piece()` : La pièce concernée par ce mouvement.

`public Poste poste()` : Le poste qui doit traiter la **piece ()**.

### 3.3 Les méthodes de `Echeancier`

`public boolean estVide()` : Renvoie vrai si cet échéancier ne contient aucun mouvement.

`public Mouvement prochain() throws NoSuchElementException` : Renvoie le plus ancien mouvement et le supprime de cet échéancier. Une exception est déclenchée si aucune pièce n'est disponible.

`public void enregistrer(Mouvement mouvement)` : Ajoute un nouveau mouvement à cet échéancier.

## 4 Les postes

Voici l'interface `Poste` :

```
public interface Poste {
    public void traiter(int date, Piece piece) ;
    public void printStat() ;
}
```

La méthode **printStat()** imprime les informations demandées par le responsable de l'usine.
La méthode **traiter()** correspond à demander au poste de traiter la **piece** à l'instant **date**.

**Cas des postes de travail :**
La méthode **printStat()** imprime les informations de ce poste de travail, puis appelle la méthode **printStat()** de son successeur.
La méthode **traiter()** doit mettre à jour les informations du poste, mais aussi, et surtout, calculer à quelle date le traitement de **piece** sera terminé (et donc à quelle date le poste sera de nouveau libre) :

- si le poste est libre, il peut facilement calculer à quelle date le traitement sera terminé.
- s'il est déjà occupé, ce n'est pas beaucoup plus compliqué, à condition que le poste connaisse la date à laquelle il aura terminé de traiter la pièce précédente.

Une fois la date de fin de traitement connue, il reste à la méthode **traiter()** à enregistrer le prochain mouvement de la pièce auprès de l'échéancier. Pour faire tout cela, le poste de travail doit connaître le poste qui le suit dans la chaîne ainsi que l'échéancier de la simulation.

Comme on l'a dit au début, il y a deux sortes de postes de travail qui ne diffèrent que par la manière de calculer la durée de traitement d'une pièce. Les postes *constants* ont une durée de traitement constante et fixée lors de leurs créations. Les postes *variables* ont une durée de traitement égale à la durée de traitement spécifique à la pièce considérée.

**Cas des postes de sortie :**
La méthode **printStat()** imprime les informations demandées.
La méthode **traiter()** met à jour les informations.

**Un exemple :** L'exécution du programme suivant :

```java
package usine;

public class ChaineUsinage {

   public static void main(String arg[]) throws Exception {
      Echeancier e = new Echeancier();
      Poste sortie = new PosteSortie();
      Poste p3 = new PosteConstant(e, sortie, 10);
      Poste p2 = new PosteVariable(e, p3);
      Poste p1 = new PosteConstant(e, p2, 6);

      e.enregistrer(new Mouvement(20, new Piece(3), p1));
      e.enregistrer(new Mouvement(21, new Piece(7), p1));
      e.enregistrer(new Mouvement(22, new Piece(5), p1));

      while (!e.estVide()) {
         Mouvement m = e.prochain();
         m.poste().traiter(m.date(), m.piece());
      }

      p1.printStat();
   }
}
```

pourra imprimer :

```
Travail (Constant-3 (6)) : Nb pieces traitees = 3, Activite = 18, Inactivite = 0
Travail (Variable-2) : Nb pieces traitees = 3, Activite = 15, Inactivite = 3
Travail (Constant-1 (10)) : Nb pieces traitees = 3, Activite = 30, Inactivite = 0
Sortie : 3 pieces traitees ; Attente max = 16, Moyenne = 7
```

si la période qui précède le traitement de la première pièce par un poste n'est pas comptabilisée comme période d'inactivité.

**Q 1 .** "Dérouler à la main" l'exemple correspondant à la méthode `main` de `ChaineUsinage` ci-dessus, notamment pour bien comprendre le fonctionnement de l'échéancier.
**Q 2 .** Donner l'algorithme de la méthode `traiter` des différents postes de travail.
**Q 3 .** Donner les codes des classes `Piece`, `Mouvement` et `Echeancier` (il faut gérer l'ordonnancement des mouvements au sein de l'échéancier).
**Q 4 .** Modéliser puis écrire les différentes classes qui implémentent les postes (de travail et de sortie) en factorisant quand c'est possible.