

SYSTEMES ET ALGORITHMES REPARTIS

Rapport du projet

GUO Hengyi – ONG Léa – PHAM Duc-Chinh

Table des matières

Producteurs – Consommateurs	3
Pseudo-code.....	3
Diagramme UML des classes	5
Justification des choix techniques	5
Documentation utilisateur	6
Copies écran de l'exécution	6

Producteurs – Consommateurs

Le contrôle de flux se réalise à partir d'un tampon dont dispose le site T, qui est partagé entre les producteurs et les consommateurs. Chaque site, producteur ou consommateur, communique avec le site T. Lorsqu'un producteur souhaite produire un message dans le tampon, le site enverra : soit une autorisation de production signifiant qu'une place est libre, soit un refus signifiant que le tampon est plein et qu'il faut attendre. De même avec un consommateur : lorsqu'il souhaite consommer un message, le site l'autorisera si le tampon n'est pas vide ou le refusera dans le cas contraire.

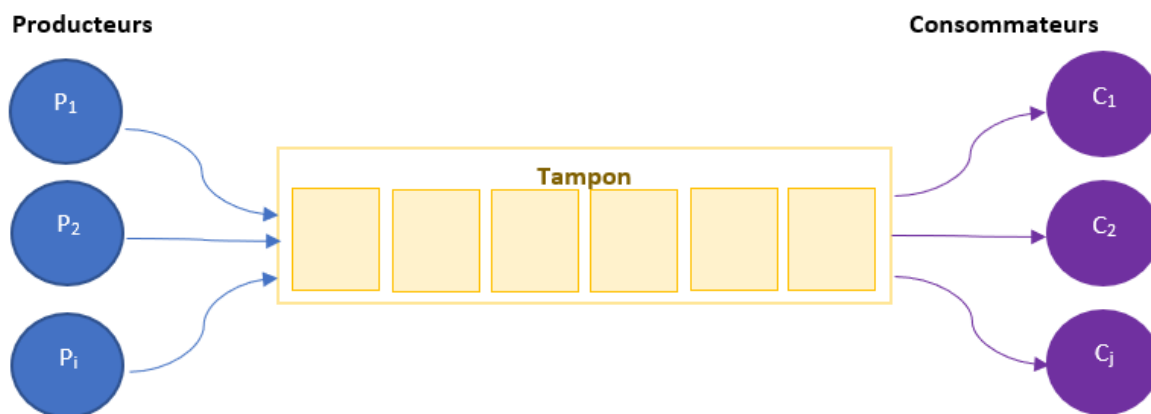


Schéma représentatif

Pseudo-code

Nous avons généralisé respectivement à une production/consommation à la fois pour chaque producteur/consommateur.

Variables :

- **Site T :**
 - $T[0..N-1]$: Tableau de taille N contenant les messages produits par les producteurs et à consommer par les consommateurs. Vide à l'initialisation.
 - In : Indice d'insertion dans le tampon. Initialisé à 0.
 - Out : Indice d'extraction dans le tampon. Initialisé à 0.
 - $Nbmess$: Nombre de messages encore non consommés stockés dans le tampon. Initialisé à 0.

Primitives :

- **Producteur :**

Facteur () :

Début

While (true)

Envoyer_à (T, Req) ;

Fin

Sur_réception_de (T, Rep) :

Début

If (Rep == Ack) Produire (m) ;

Fin

- **Consommateur :**

Facteur () :

Début

While (true)

Envoyer_à (T, Req) ;

Fin

Sur_réception_de (T, Rep) :

Début

If (Rep == Ack) Consommer () ;

Fin

- **Site T :**

Produire (m) :

Début

Attendre (Nbmess < N) ;

T[in] = m ;

in = (in + 1) % N ;

Nbmess++ ;

Fin

Consommer (m) :

Début

Attendre (Nbmess > 0) ;

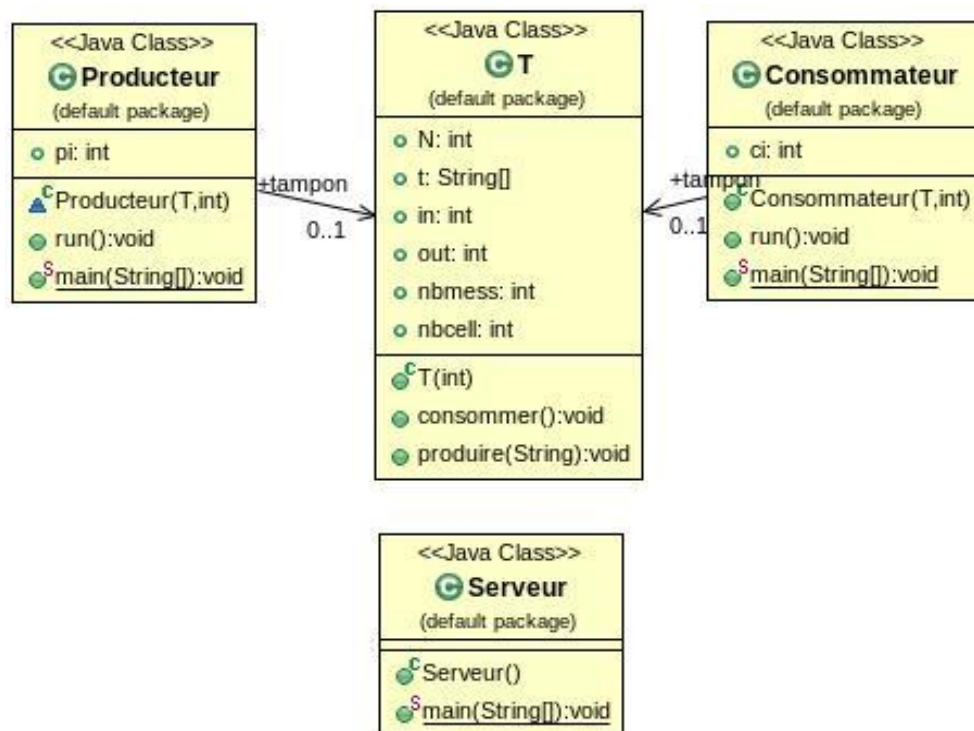
m = T[out] ;

out = (out + 1) % N ;

Nbmess-- ;

Fin

Diagramme UML des classes



Justification des choix techniques

Nous n'avons pas réussi à implémenter l'algorithme généralisé du cours avec les primitives *Sur_réception_de*, *Envoyer_à*... Nous avons donc implémenté les primitives *Produire (m)* et *Consommer ()* dans le site T qui est responsable de la gestion du tampon. De plus, étant donné la restriction à une seule autorisation de production/consommation à la fois, il n'y a donc pas de présence de jeton permettant l'asservissement des producteurs et consommateurs. Nous avons en effet utilisé les threads, notamment les méthodes *sleep ()* et *notifyAll ()* afin de laisser le temps aux autres sites de produire ou consommer.

Etapes de programmation :

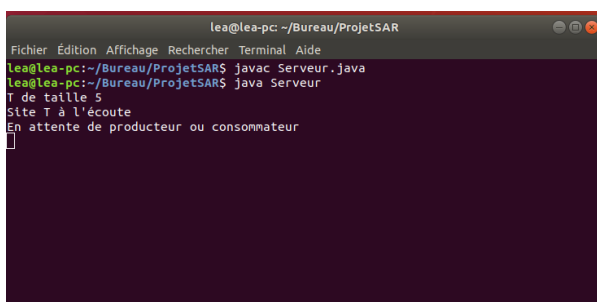
1. Définir la taille du tampon et lancement du serveur
2. Ecoute constante du serveur
3. Ajout des/du client(s) Producteur et/ou Consommateur au serveur
4. À la suite de l'ajout du Producteur/Consommateur, ce dernier commence à produire/consommer sur le site T
5. Il est possible de lancer plusieurs Producteurs et/ou Consommateurs
6. A chaque production ou consommation, le tampon est mis à jour. Si non, un message apparaît et les producteurs/consommateurs sont en attente
7. L'avancement est visible sur le terminal du serveur. Le partage de l'avancement sur les terminaux des clients n'a pas été possible.

Documentation utilisateur

1. Décompresser l'archive *ProjetSAR* contenant le code source (4 fichiers .java)
2. Ouvrir 2 terminaux et se placer dans le répertoire contenant les fichiers sources
3. Dans le 1^{er} terminal, compiler le fichier *Serveur.java* à l'aide de la commande `javac Serveur.java`
4. Exécuter le fichier avec la commande `java Serveur` afin de démarrer le serveur. *Note :* Il est possible de définir la taille du tampon du site *T* en argument, à entrer après *Serveur*. Si aucun argument n'est rentré, le site *T* aura par défaut un tampon de taille 5
5. Un message s'affiche sur le terminal, indiquant à l'utilisateur que le serveur est à l'écoute et qu'il est en attente des producteurs et/ou consommateurs
6. Le 2nd terminal servira à ajouter les producteurs/consommateurs dans le serveur
7. Dans le 2nd terminal, compiler les fichiers *Producteur.java* et *Consommateur.java* avec la commande `javac Producteur.java Consommateur.java`
8. Exécuter le fichier souhaité avec la commande `java [nomFichier]` (avec nomFichier : Producteur ou Consommateur)
9. Sur le 1^{er} terminal, on peut apercevoir les échanges entre les producteurs/consommateurs et le site *T*
10. Il est possible d'ajouter autant de producteurs et consommateurs sur le serveur. Pour cela, il suffit de réitérer les étapes à partir de la 8^{ème}

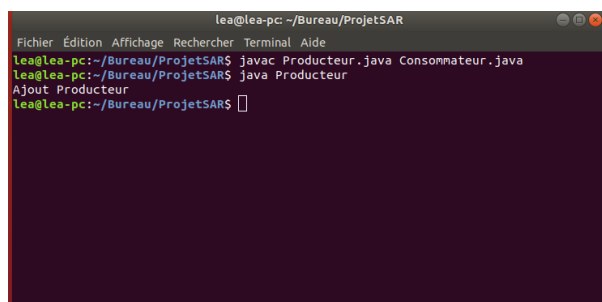
Copies écran de l'exécution

Exécution avec le tampon par défaut (de taille 5) avec au départ 1 seul producteur.



```
lea@lea-pc: ~/Bureau/ProjetSAR
Fichier Édition Affichage Rechercher Terminal Aide
lea@lea-pc:~/Bureau/ProjetSAR$ javac Serveur.java
lea@lea-pc:~/Bureau/ProjetSAR$ java Serveur
Site T à l'écoute
En attente de producteur ou consommateur
```

Serveur en attente de producteur/consommateur



```
lea@lea-pc: ~/Bureau/ProjetSAR
Fichier Édition Affichage Rechercher Terminal Aide
lea@lea-pc:~/Bureau/ProjetSAR$ javac Producteur.java Consommateur.java
lea@lea-pc:~/Bureau/ProjetSAR$ java Producteur
Ajout Producteur
lea@lea-pc:~/Bureau/ProjetSAR$
```

Producteur ajouté au Serveur et souhaite produire des messages

