Mail : yassine.elkhalki@outlook.fr
Nom : ELKHALKI Yassine
Groupe : INF3-FA

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES — IUT DE VÉLIZY

Dépôt Git : https://github.com/YassouSensai/yassine_prog_avancee.git

---

Ce rapport a été réalisé en se basant sur les différents CMs & TPs/TDs

Sommaire

Un README est présent sur le dépôt git pour retrouver le code des différents TPs

Notes de cours

CM1

Architectures matérielles que je ne connaissais pas ou de nom pour le HPC

- **HPC (High Performance Computing) :** Utilisation d'un cluster de processeurs qui travaillent en parallèle pour traiter des grandes quantités de données et résoudre des problèmes complexes (calcul intensifs, problématiques actuelles pour l'innovation, …)
- **FPGA (Field-Programmable Gate Array) :** Un type de circuit intégré qui est composé de blocs logiques utilisés pour effectuer beaucoups de fonctions (exemple du prof avec les feux de circulation)

Exemples, selon moi, de SR et d'AR

- **Système répartie :** Par exemple, il s'agit de l'ensemble des ressources matérielles (ou non), connectées entre elles et qui apparaissent comme une seule machine. Les ressources coopèrent afin d'atteindre un objectif commun.

- **Application répartie :** Par exemple, un logiciel fonctionnant sur un ensemble de machine, ce qui nécessite une coordination (gestion de la mémoire RAM, utilisation des CPU/GPU, …). => Il s'agit d'une application destinée à fonctionner sur un environnement de système répartie.

=> 5 principes de conception : Ici, le terme "Transparence" veut dire que l'utilisateur ne doit pas tenir compte de (selon moi) :

- **Transparence à la localisation :** la localisation des différentes ressources matérielles
- **Transparence d'accès :** L'accès aux différentes ressources. Elle se fait via une seule interface
- **Transparence à l'hétérogénéité :** les différences matérielles
- **Transparence aux pannes :** les pannes partielles (réseau, machines, logiciels, …)
- **Transparence à l'extension des ressources :** L'extension / La réduction du système

Contextes d'exécution :

- Mono Core :
- Multi Core :
- Multi Thread :

Thread = fil d'exécution : Plus petite unité d'exécution qu'un processeur peut gérer de manière indépendante (ChatGPT). => Meilleure parallélisation

CM2

Contexte multi Thread

Finalement : **Thread** = processus léger qui est caractérisé par son environnement (processus, actif/en attente, …). Il possède sa propre pile.

La classe Thread (en Java) permet de créer des processus (ou plutôt, des unités d'exécutions) qui peuvent communiquer entre eux et même être synchronisés (car ils utilisent le même espace mémoire).

Termes abordés :

- **Mot clé synchronized :** synchronisation de blocs de codes ou encore, de méthodes. Un verrou (lock) est mis en place interdisant l'accès au bloc à d'autres threads. => **Verrou MUTEX**

Dans ce contexte, synchroniser veut dire restreindre l'accès à un morceau de code pour qu'un seul thread aie l'accès. (Exemple sur l'avancement du Tp avec le déplacement des mobiles : un seul mobile peut se déplacer sur une section à un instant t)

- **Ressource critique :** ressource qui ne peut être utilisée que par un seul processus à la fois (comme un scanneur de document (exemple de l'imprimante et de la case mémoire dans le cours))
- **Section critique :** Il s'agit d'une partie du code qui va contenir des objets / informations susceptible d'être modifiées et accessibles par plusieurs threads. La protection de cette partie de code se fait par la synchronisation de celle-ci.
- **Semaphore :** Mécanisme de synchronisation qui permet l'accès à une ressource critique à un nombre limité de processus (un peu comme un pool en python)

CM3

Principalement parallélisme de tâches (2eme rapport ?)

(Parallélisme de données => + pour le calcul scientifique)

SIMD => résolution par le GPU
MMD => résolution par le CPU

<u>Structure d'algorithme (exécution sur une machine parallèle) :</u>

- parallélisme récursif : décomposition des tâches
- pipeline : nombre de tâche fois plus vite
- Maître/Esclave VS Client/Serveur

- SPMD : Single Programming Multiple Data

TPs/TDs

TD1

Architecture matérielle de l'ordinateur

Avec la commande msinfo32 (ligne de commande windows) : (Machine de l'IUT)

Nom de l'appareil G26g-9
Nom complet de l'appareil G26g-9.veli.iut-velizy.uvsq.fr
Processeur Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 3.60 GHz
Mémoire RAM installée 32,0 Go (31,9 Go utilisable)
ID de périphérique D50C37BE-9C65-4C2E-B9DD-75C9B9C4B650
ID de produit 00378-40000-00001-AA105
Type du système Système d'exploitation 64 bits, processeur x64
Stylet et fonction tactile La fonctionnalité d'entrée tactile ou avec un stylet n'est pas disponible sur cet écran

On peut également aller voir dans le gestionnaire des périphériques ou encore dans **Gestionnaire des tâches > Performance**

TP1

TP sur les mobiles (différentes évolutions)

1) Aller-retour avec un seul mobile : Très simple, modifier la boucle while dans la méthode run() de la classe UnMobile.

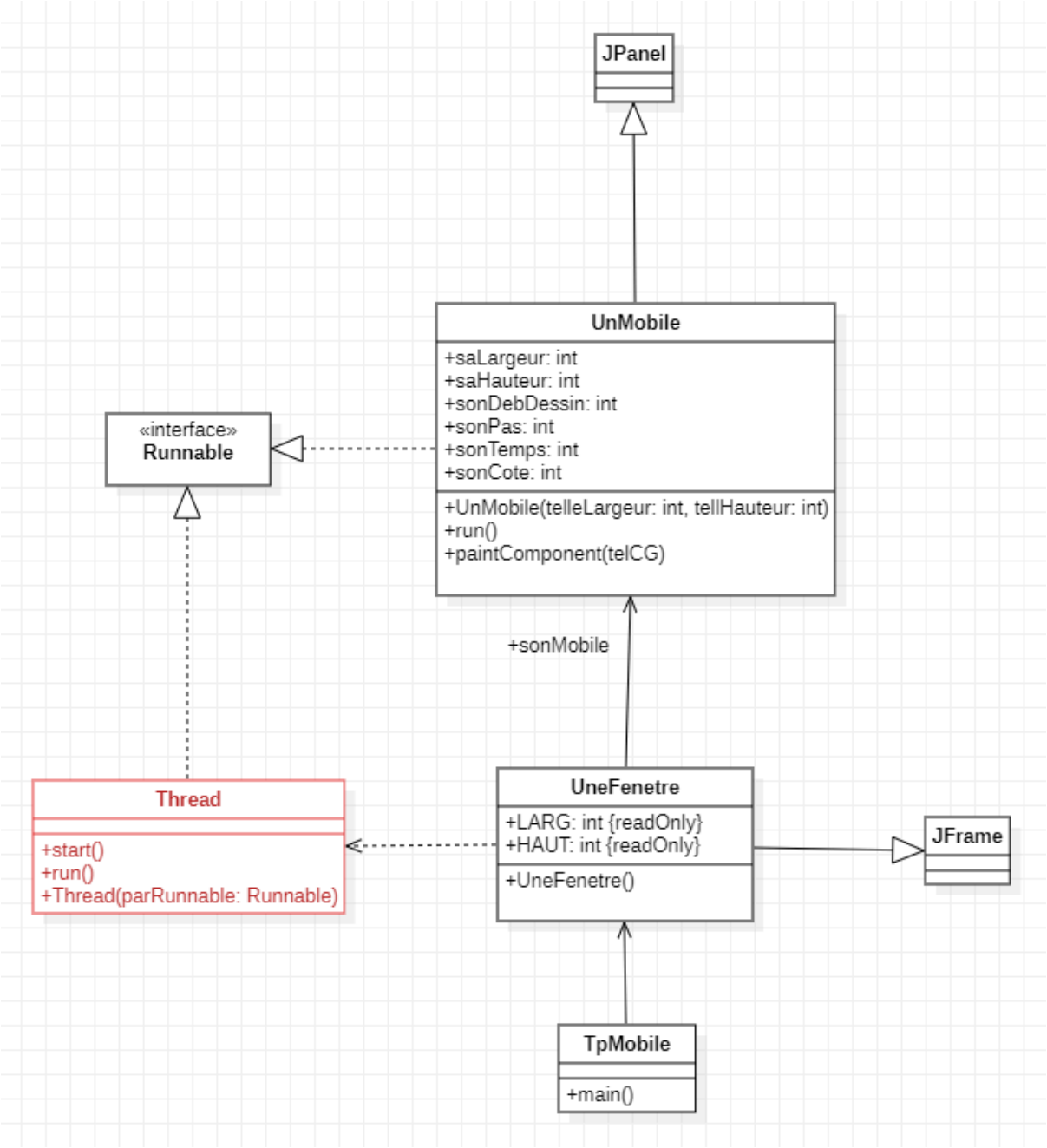
2) Version avec trois mobiles ou plus : Il faut modifier la classe UneFenetre de sorte à donner un nombre de lignes. Et dans une boucle for, il faut rajouter un objet de la classe UnMobile dans le conteneur et activer la méthode start().

3) Dernière version avec autant de mobiles que l'on veut, mais le conteneur est séparé en trois colonnes imaginaires qui constituent des sections critiques. Cette fois-ci, lors de l'activation des mobiles avec la méthode start, on utilisera la classe Thread pour qu'aucun mobile n'accède à une section critique qui est déjà utilisée.

   Dans la classe UnMobile, on utilisera les méthodes wait() et signal() du sémaphore qui a été implémenté lors du TP2 mais adapté au TP1.

Lors des TPs, je n'ai pas pensé à faire des tags pour pouvoir accéder aux différentes évolutions. Ainsi, seule la dernière version du TP est disponible dans le dépôt git.

Voici également le diagramme UML fait ensemble en séance de TP/TD/cours :



TP2

TP2 pour illustration du CM2

- Semaphore binaire : le code avait été donné entièrement. L'affichage des lettres se faisait bien l'un après l'autre.
- Semaphore : qui permet de signaller aux autres threads que la section est libre, mais en decrémentant une variable

TP3 + cours de l'autre prof

TP3 pour la boite aux lettres en se basant sur l'exemple de la boulangerie

=> Programmation concurrente avec les files d'attentes
=> On utilise une BloquingQueue (Buffer) qui a plusieurs méthodes - celles utilisées :

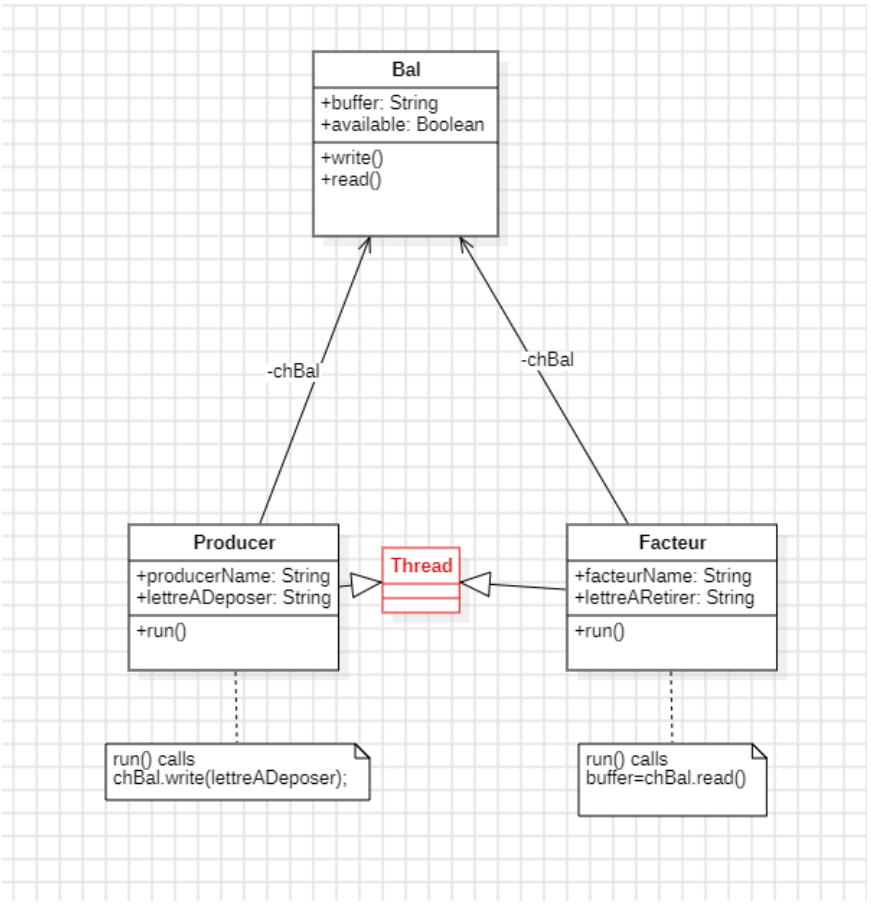
- offer : Ajoute un élément à la queue et renvoi false si la queue est pleine contrairement à la méthode add qui lève une exception (Méthode deposer())
- poll : pareil que offer mais pour retirer un élément (Méthode retirer()). (renvoie null et fonction similaire remove)

On détermine la taille de la queue lors de l'instanciation de la queue.

2 sections critiques (deposer() et retirer())

Par rapport à l'exemple de la boulangerie, Dans mon code, la classe Boulanger = Producteur, Boulangerie = BoiteALettre, Mangeur = Facteur

<u>Voici le diagramme UML pour la boite aux lettres (fait en cours):</u>
(Le diagramme n'a pas été mis à jour, mais le buffer est de type BloquingQueue)
(La méthode write() => deposer() dans la classe BoiteALettre)
(La méthode read() => retirer() dans la classe BoiteALettre)
(La classe Bal s'appelle BoiteALettre)
(Les Producers déposent les lettres)
(Les Facteurs les récupèrent)


Bal
+buffer: String
+available: Boolean
+write()
+read()
-chBal
-chBal
Producer
+producerName: String
+lettreADeposer: String
+run()
Thread
Facteur
+facteurName: String
+lettreARetirer: String
+run()
run() calls
chBal.write(lettreADeposer);
run() calls
buffer=chBal.read()