Faculté des Sciences
Université d'Etat D'Haïti

Réponses aux questions théoriques du TP ATM de programmation

orientée objet en

Java

BENOIT Ulrick Wadson
DAGOBERT Dually
PETIT-HOMME Ivenson
PIERRE Valendino
(Étudiants en Conception et Développement d'Applications Web & Mobile)

21 décembre 2025.
Professeur : Amosse Édouard.

Réponses aux questions théoriques de « Partie 3 - Mise en évidence d'une Race Condition »

1. Quel solde attend-on théoriquement ?

Théoriquement, avec un solde initial de 1000 € et deux retraits de 700 € chacun, on devrait obtenir :

- Solde final attendu : 300 € (seul le premier retrait devrait réussir).

- Le deuxième retrait devrait échouer avec une erreur "fonds insuffisants" car après le premier retrait, il ne reste que 300 €.

2. Quel solde observez-vous parfois ?

Sans synchronisation, on observe parfois un solde final de -400 € (si les deux retraits réussissent).

Parfois aussi on obtient 300 € (si un thread termine complètement avant que l'autre ne commence).

Le résultat est inconstant et imprévisible, c'est exactement le problème !

3. Pourquoi les deux retraits peuvent-ils réussir ?

Les deux retraits peuvent réussir à cause de la séquence d'exécution non atomique :

1. Thread 1 lit le solde : 1000 €
2. Thread 2 lit le solde : 1000 € (avant que Thread 1 n'ait modifié le solde)
3. Thread 1 soustrait 700 € => solde = 300 €
4. Thread 2 soustrait 700 € => solde = -400 €

Les deux threads exécutent les opérations *lire_solde()* et *modifier_solde()* dans un ordre entremêlé, car ces opérations ne sont pas atomiques.

4. Comment appelle-t-on ce type de problème ?

Ce type de problème s'appelle une *Race Condition* (Condition de Course en français).

C'est une situation où plusieurs threads accèdent et modifient une ressource partagée simultanément, et le résultat final dépend de l'ordre d'exécution des threads, ce qui conduit à un comportement imprévisible et erroné.

Réponses aux question théoriques de « Patie 4 – Synchronisation avec Synchronized »

1. Pourquoi *synchronized* empêche-t-il la *Race Condition* ?

synchronized empêche la race condition en garantissant l'exclusivité d'accès à la méthode ou au bloc de code :

1. Verrou (lock) sur l'objet : Quand un thread entre dans une méthode *synchronized*, il acquiert un verrou sur l'objet (*this*).

2. Exécution atomique : Les opérations *lire solde()* et *modifier solde()* deviennent atomiques - aucun autre thread ne peut interrompre entre ces deux opérations.

3. Visibilité des modifications : Grâce au mécanisme de *synchronized*, les modifications faites par un thread sont immédiatement visibles par les autres threads qui entrent ensuite dans la méthode.

En résumé : *synchronized* transforme l'accès concurrent en accès séquentiel pour la section critique.

2. Que se passe-t-il si plusieurs threads veulent entrer dans la méthode ?

Si plusieurs threads veulent entrer dans une méthode *synchronized* sur le même objet :

1. Un seul thread entre : Le premier thread qui arrive acquiert le verrou et entre dans la méthode.

2. Les autres threads sont bloqués : Ils sont mis en état d'attente (dans une file d'attente liée au verrou de l'objet).

3. Libération du verrou : Quand le thread sort de la méthode (normalement ou par exception), il libère le verrou.

4. Sélection du prochain thread : Le scheduler Java choisit un thread en attente pour lui donner accès au verrou.

5. Ordre d'accès : L'ordre d'accès n'est pas garanti (c'est le scheduler qui décide), mais chaque thread aura son tour.

Les threads qui attendent ne consomment pas de temps CPU - ils sont dans un état de veille jusqu'à ce que le verrou soit disponible.

Voyons un exemple concret de la différence.

Sans *synchronized* (Race Condition) :

Thread 1: lit 1000 €
Thread 2: lit 1000 €
Thread 1: retire 700 € => solde = 300 €
Thread 2: retire 700 € => solde = -400 €

Avec *synchronized* (Thread-safe) :

Thread 1: entre dans withdraw(), verrou acquis
Thread 1: lit 1000 €
Thread 1: retire 700 € => solde = 300 €
Thread 1: sort de withdraw(), verrou libéré

Thread 2: entre dans withdraw(), verrou acquis
Thread 2: lit 300€
Thread 2: erreur "fonds insuffisants"
Thread 2: sort de withdraw(), verrou libéré

Cette différence illustre parfaitement pourquoi la synchronisation est cruciale dans les applications multi-threads accédant à des ressources partagées.