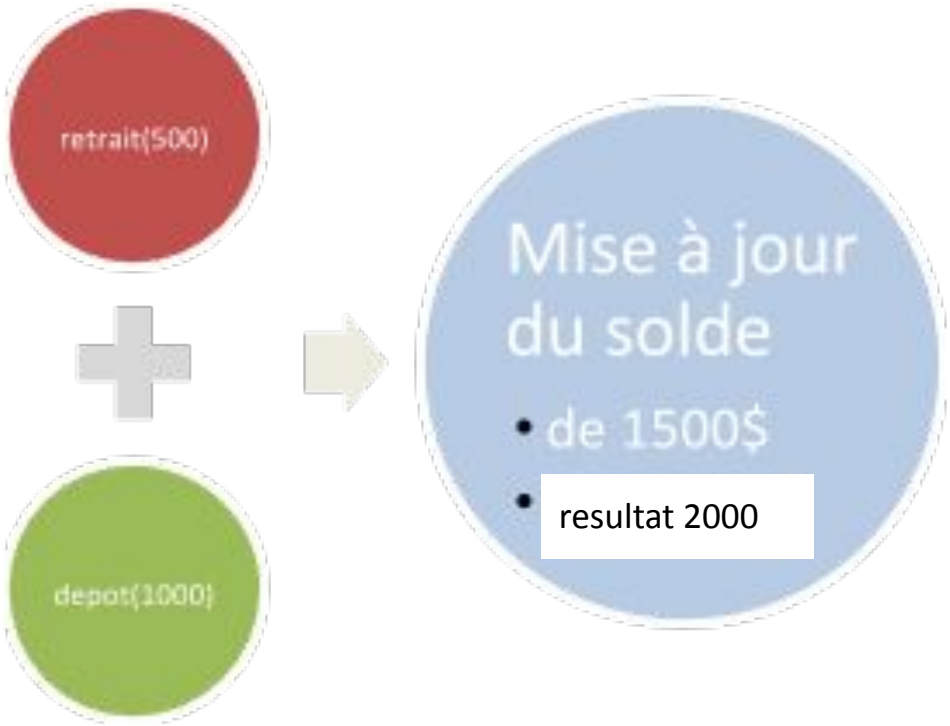


La Concurrence

Démonstration et évitement

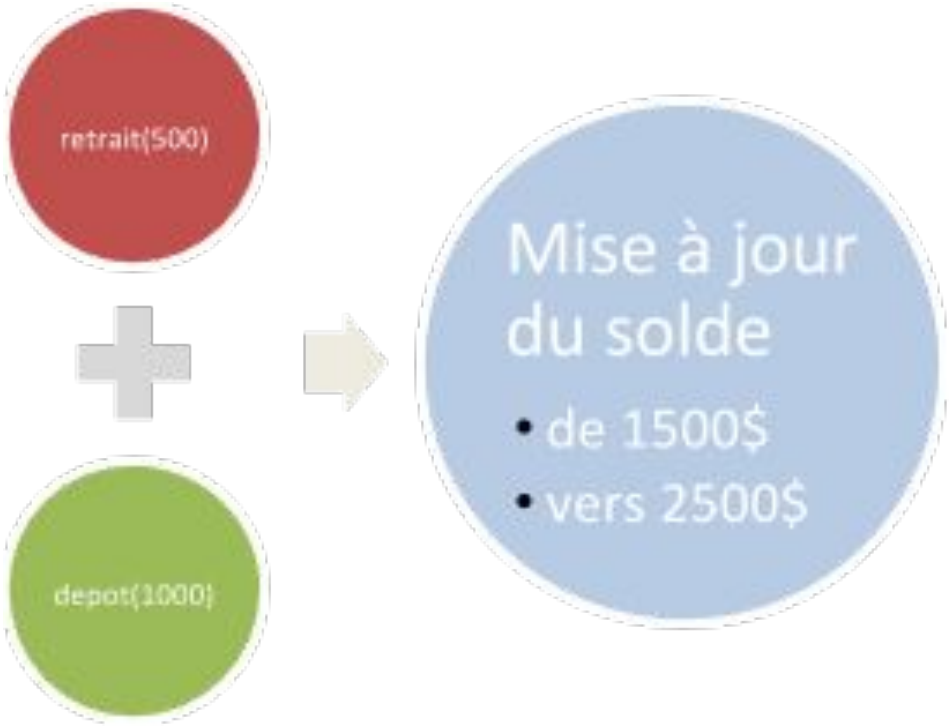
Opérations désirées

Supposons que vous avez un compte bancaire conjoint et le hasard veuille que vous et votre conjoint alliez effectuer une opération bancaire au même moment.



Opérations désirées

Supposons que vous avez un compte bancaire conjoint et le hasard veuille que vous et votre conjoint alliez effectuer une opération bancaire au même moment.





Code des opérations

Voici le code des deux procédures qui emploient une variable locale dans chaque cas.

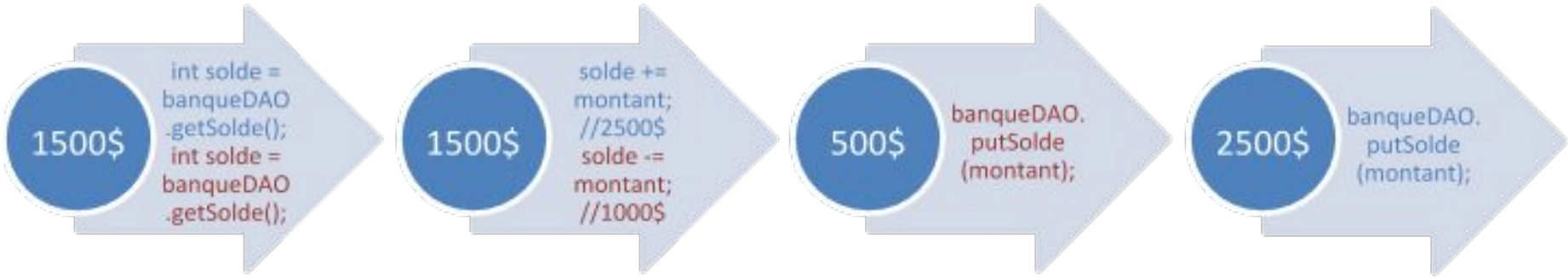
```
bool depot(int montant)
{
  int solde = banqueDAO.getSolde();
  solde += montant;
  banqueDAO.putSolde(montant);
  return true;
}
```

```
bool retrait(int montant)
{
  int solde = banqueDAO.getSolde();
  solde -= montant;
  banqueDAO.putSolde(montant);
  return true;
}
```

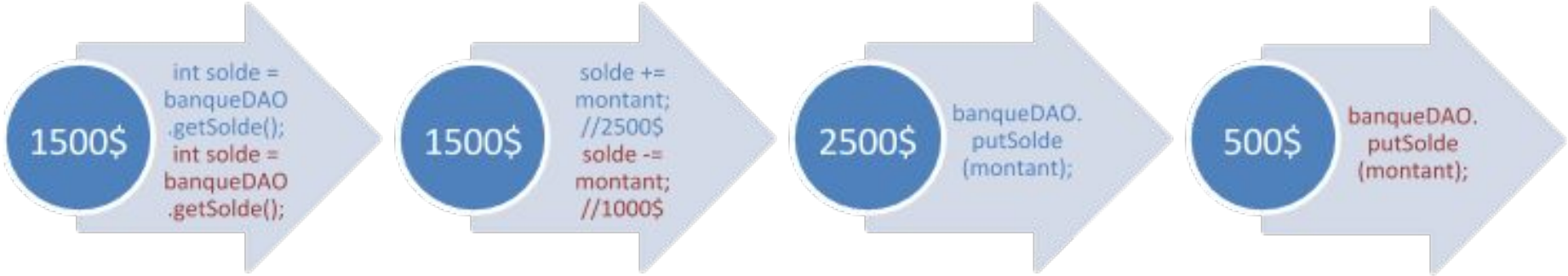


Problème possible

Si le dépôt se complète avant le retrait…



Si le retrait se complète avant le dépôt par contre…

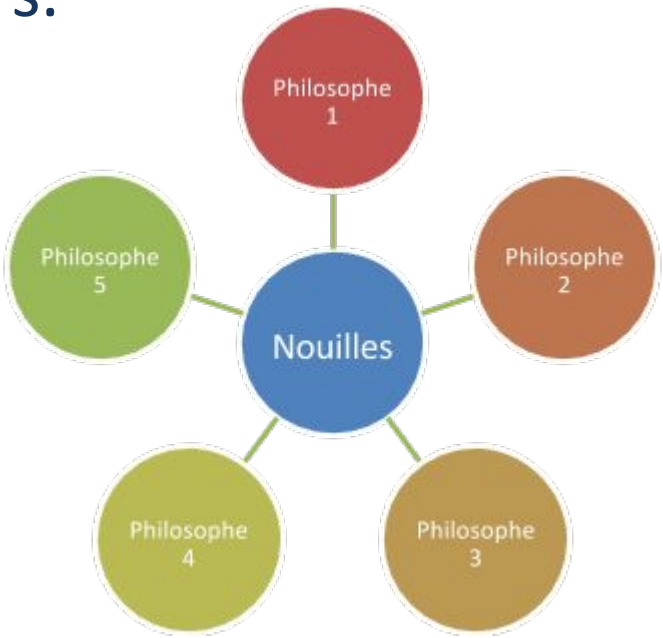


Paradigme producteur-consommateur

Il s'agit d'une solution générale au problème de la concurrence sur des ressources partagées lorsqu'il existe un certain nombre de ressources à consommer par un certain nombre de consommateurs.

*** Voir histoire

des 7 philosophes





Le diner des philosophes

Dans le problème classique des philosophes, il existe le même nombre d'ustensiles (disons des cuillères) que de philosophes. Chaque philosophe doit attendre d'avoir les deux ustensiles de chaque côté de lui avant de pouvoir manger un peu. Il a le droit d'accaparer un ustensile en attendant l'autre.
Le défi est de produire un algorithme qui ne pose pas l'un des problèmes de synchronisation suivants :

- Dead-lock : Si chaque philosophe possède une cuillère et seulement une, le système est bloqué pour toujours
- Non-progression : Certains algorithmes permettent d'éviter le dead-lock mais au prix de la priorité excessive de certains philosphes, amenant que d'autres vont mourir de faim.



Démos du Diner des philosophes

- http://java.com/fr/download/faq/exception_sitelist.xml
- http://www.java.com/fr/download/help/jcp_security.xml
- http://ptolemy.eecs.berkeley.edu/ptolemyII/ptII8.0/ptII/ptolemy/domains/csp/demo/DiningPhilosophers/DiningPhilosophers.htm
- http://users.erols.com/ziring/diningAppletDemo.html
- http://www.doc.ic.ac.uk/~jnm/concurrency/classes/Diners/Diners.html
- http://www-cs-students.stanford.edu/~dwhitney/java/DiningPhilosophers/DiningPhilsApplet.html
- https://web.duke.edu/cps001/labs/lab9/lab9.html



La concurrence en pratique

- Java Concurrency in Practice
  - http://jcip.net/
  - http://jcip.net/listings.html (code listing)
- Java Concurrency / Multithreading Tutorial
  - http://tutorials.jenkov.com/java-concurrency/index.html
  - http://www.vogella.com/tutorials/JavaConcurrency/article.html
- Articles DZone
  - https://dzone.com/articles/java-concurrency-in-depth-part-1
  - https://dzone.com/articles/java-concurrency-thread-pools
  - https://dzone.com/articles/3-synchronization-idioms

Mécanismes de synchronisation



- Lock = Mutex = Simple objet de type jeton qu'on associe à une ressource partagée afin de la protéger des opérations concurrentes.
- Sémaphore = Simple objet de type compteur qu'on associe à une ressource partagée afin de la surveiller et d'exclure les opérations concurrente dessus.
- Moniteur = Objet qui surveille et exclut activement un ensemble d'opérations sur des processus/threads différents. Il est souvent implémenté sous forme d'une paire mutex-condition.



Mécanisme et structure de synchronisation

La différence fondamentale entre le moniteur et les autres structures est que celui-ci prend charge de TOUT. Il ne doit pas être acquis et relâché, c'est lui qui fait tout ce travail étant donné des méthodes à protéger. De plus, le moniteur s'associe généralement à des opérations et non pas à des données, c'est notre tâche d'envelopper le bon groupe d'opérations de manière à encapsuler nos données de manière thread-safe.

Les autres structures (mutex et sémaphore) sont des constructions de bas-niveau, simples, qui sont passives.

Le mutex est un sémaphore spécialisé dans l'exclusion mutuelle pour la protection d'une ressource. Par déduction, les sémaphores non-mutex sont utilisés pour les autres usages de signalement…. certains auteurs vont jusqu'à dire que les sémaphores ne servent qu'à signaler (http://www.barrgroup.com/Embedded-Systems/How-To/RTOS-Mutex-Semaphore, http://www.geeksforgeeks.org/mutex-vs-semaphore/), en cela ils n'ont pas tort, mais le mutex serait-il ou non une forme de signalement de type boîte aux lettres simplifié. Sous ce regard, les mutex sont des sémaphores simplifié, que ce soit structurellement ou sémantiquement. Il est cependant de bon ton de connaître cette différence qui leur est attribué dans différents domaines.



Mécanisme et structure de synchronisation

Mutex	1 Jeton	Booléen	Protège une ressource
Sémaphore	Distributeur de jetons	Compteur	Échange de signaux
Moniteur	Surveillant	Mutex + Condition + Threads	Exclut de multiples opérations



Le principe ACID

- Atomicity = Atomicité = tout est un succès ou rien
- Consistency = Consistence = Assure que la donnée passe d'un état consistant à un autre
- Isolation = Isolation = Assure que la transaction est isolée des autres transactions
- Durability = Durabilité = une fois la transaction terminée elle reste, peu importe quoi (perte de courant, etc.)

Références en programmation



- http://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/Java/Conc/ConcProgJava/
- http://msdn.microsoft.com/en-us/magazine/dd882512.aspx
- http://os-sistemasoperativos.blogspot.ca/2011/09/1.html
- http://courses.cs.vt.edu/~csonline/OS/Lessons/Deadlock/index.html
-

Technologies de Sémaphore

- MSDN
  - http://msdn.microsoft.com/en-us/library/ms228964%28v=vs.110%29.aspxL



Technologies de Transaction

- Java
  - https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html
  - https://docs.oracle.com/javaee/5/tutorial/doc/bncii.html



Transaction en Java

Pour faire une transaction en Java, il faut

- désactiver l'envoi automatique des requêtes
- accumuler les requêtes dans la connection
- faire un commit

En code:

- basededonnees.setAutoCommit(false);
- (les requêtes sont comme à l'habitude)
- basededonnees.commit();

