Concurrence rendez vous

Rémi Forax

Echange d'information

Si deux threads veulent s'échanger des données

- Les données vont transitées par le tas (dans des champs)
- Il faudra une ou plusieurs sections critiques pour éviter les états incohérents

mais ce n'est pas suffisant, il faut aussi que la thread qui veut recevoir des données soit mis en attente

Exemple

Qu'affiche ce code ?

```
public class Holder {
  private int value;

  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(() -> {
       holder.value = 12;
    }).start();
    System.out.println(holder.value);
  }
}
```

Exemple (suite)

Trouver les 2 bugs !

```
public class Holder {
  private int value;
  private boolean done;

  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(() -> {
       holder.value = 12;
       holder.done = true;
    }).start();
    while(!holder.done) {
      // ne rien faire !
    }
    System.out.println(holder.value);
  }
}
```

Exemple (suite/2)

Bug 1: attente active, le CPU tourne en boucle

```
public class Holder {
  private int value;
  private boolean done;

  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(() -> {
       holder.value = 12;
       holder.done = true;
    }).start();
    while(!holder.done) {
      // ne rien faire !
    }
    System.out.println(holder.value);
  }
}
```

On réchauffe la planète

Exemple (suite/2)

Bug 1: attente active, le CPU tourne en boucle

```
public class Holder {
  private int value;
  private boolean done;

  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(() -> {
       holder.value = 12;
       holder.done = true;
    }).start();
    while(!holder.done) {
      // ne rien faire !
    }
    System.out.println(holder.value);
  }
}
```

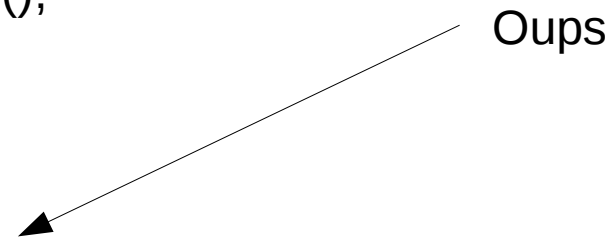
On réchauffe la planète

Exemple (suite/3)

Bug 2: Le JIT peut introduire un registre pour la valeur holder.done

```
public class Holder {
  private int value;
  private boolean done;

  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(() -> {
       holder.value = 12;
       holder.done = true;
    }).start();
    boolean r0 = holder.done;
    while(!r0) {
      // ne rien faire !
    }
    System.out.println(holder.value);
  }
}
```

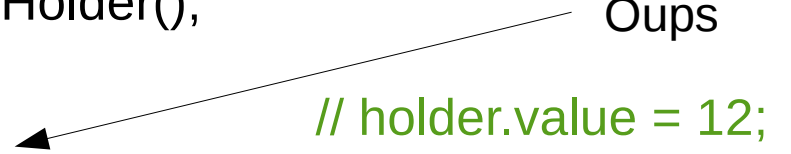


Exemple (suite/4)

Bug 2 bis: Le JIT ou le CPU peut ré-organiser les deux assignations

```
public class Holder {
  private int value;
  private boolean done;

  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(() -> {
       holder.done = true;
       holder.value = 12;
    }).start();
    while(!holder.done) {
      // ne rien faire !
    }
    System.out.println(holder.value);
  }
}
```

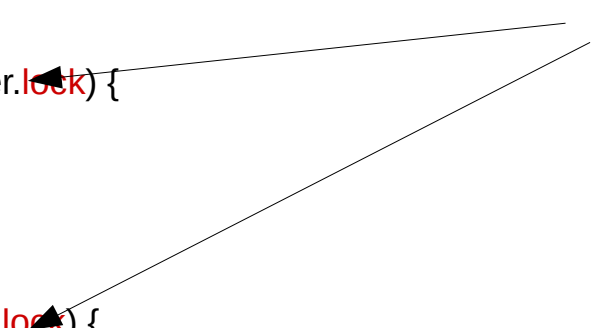


Exemple (suite/5)

Un section critique résoud le bug 2

```
public class Holder {
  private int value;
  private boolean done;
  private final Object lock = new Object();

  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(() -> {
      synchronized(holder.lock) {
        holder.value = 12;
        holder.done = true;
      }
    }).start();
    for(;;) {
      synchronized(holder.lock) {
        if (holder.done) {
          break;
        }
      }
    }
    synchronized(holder.lock) {
      System.out.println(holder.value);
    }
  }
}
```



Attention il faut bien les 2 blocs synchronized

Et cela ne résoud pas le problème
de l'attente active !

Rendez vous

On veut une façon d'arrêter une thread tant qu'une condition n'est pas vérifié

- La thread en attente doit être dé-schédulée

Lorque la condition est vrai, on veut réveillé la thread

- La thread en attente doit être re-scédulée

Ce mécanisme doit marcher avec les sections critiques car sinon on a les problèmes de visibilité, reorganisation, etc classique

Exemple de wait()/notify()
(mais pas objet)

```
public class Holder {
  private int value;
  private boolean done;
  private final Object lock = new Object();

  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(() -> {
       synchronized(holder.lock) {
         holder.value = 12;
         holder.done = true;
         holder.lock.notify();
       }
    }).start();
    synchronized(holder.lock) {
      while(!holder.done) {
        holder.lock.wait();
      }
      System.out.println(holder.value);
    }
  }
}
```

Reveil la thread si en attente sinon ne fait rien

Demande de mettre la thread en attente

Exemple de wait()/notify()
(version objet !)

```
public class Holder {
  private int value;
  private boolean done;
  private final Object lock = new Object();

  private void init() {
    synchronized(lock) {
      value = 12;
      done = true;
      lock.notify();
    }
  }

  private void display() {
    synchronized(lock) {
      while(!done) {
        lock.wait();
      }
      System.out.println(holder.value);
    }
  }
```

Reveil la thread si en attente sinon ne fait rien

Demande de mettre la thread en attente

- ```
  public static void main(String[] args) {
    Holder holder = new Holder();
    new Thread(holder::init).start();
    holder.display();
  }
}
```
```

wait()

object.wait()

- demande à arréter la thread courante
  - Celle-ci est de-schedulé
- re-donne le moniteur associé à l'objet

  une fois la thread “reveillé”
- re-prend le moniteur associé à l'objet
- demande à reprendre la thread
  - La thread est re-schédulé

Conditions de reveil

Comment une thread peut être reveillé d'un foo.wait() ?

- Si une autre thread appel foo.notify() ou foo.notifyAll()
- Si une autre thread interrompt la thread courante avec interrupt()
  - dans ce cas foo.wait() lève InterruptedException
- Sans raison apparente (spurious wake-up)

notify() / notifyAll()

foo.notify()

- Demande le réveil d'**une** thread en attente sur foo avec un foo.wait()
- Si aucune thread est en attente, le notify() est perdu et il ne se passe rien

foo.notifyAll()

- Demande le réveil de **toutes** les thread en attente sur foo avec un foo.wait()

Wait/notify et synchonized

foo.wait() ou foo.notify() ne peut être appelé que à l'intérieur d'un bloc synchonized sur foo

- Pour wait(), c'est parce que wait() rend le moniteur associé à l'objet après avoir endormi la thread courante puis reprend le moniteur lorsque la thread est réveillé et sort du wait()

- Pour notify(), c'est pour permettre à ce que la condition et le notify() soit atomique du point de vue des autres threads

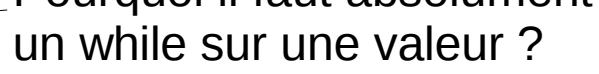
Pourquoi un while sur une valeur ?

```
public class Holder {
  private int value;
  private boolean done;
  private final Object lock = new Object();

  private void init() {
    synchronized(lock) {
       value = 12;
       done = true;
       lock.notify();
    }
  }

  private void display() {
    synchronized(lock) {
      while(!done) {
        lock.wait();
      }
      System.out.println(holder.value);
    }
  }

  public static void main(String[] args) {
    ...
  }
}
```



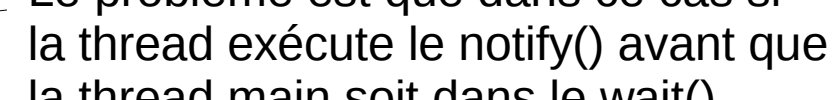
Pourquoi pas directement un wait() ?

```
public class Holder {
  private int value;
  private boolean done;
  private final Object lock = new Object();

  private void init() {
    synchronized(lock) {
       value = 12;
       done = true;
       lock.notify();
    }
  }

  private void display() {
    synchronized(lock) {
      while(!done) {
        lock.wait();
      }
      System.out.println(holder.value);
    }
  }

  public static void main(String[] args) {
    ...
  }
}
```



Le problème est que dans ce cas si la thread exécute le notify() avant que la thread main soit dans le wait(), le notify() est perdu

Et le wait() va attendre indéfiniment !

Notification perdu et condition

Pour éviter les notifications perdu et donc d'attendre indéfiniment dans le wait()

- Il faut qu'un champ change de valeur

Comme cela, on peut détecter si il faut faire le wait() ou pas

- Bien sûr, ce champs doit être modifié uniquement dans des blocs synchronized.

On appel le champ **la condition** du wait()

Pourquoi pas un if ?

```
public class Holder {
  private int value;
  private boolean done;
  private final Object lock = new Object();

  private void init() {
    synchronized(lock) {
       value = 12;
       done = true;
       lock.notify();
    }
  }

  private void display() {
    synchronized(lock) {
      if(!done) {
        lock.wait();
      }
      System.out.println(holder.value);
    }
  }

  public static void main(String[] args) {
    ...
  }
}
```

Car on n'est pas protégé contre les **spurious wakeups**

While condition + wait

Un wait() n'est jamais tout seul,

Il est toujours dans un while sur une condition

Exemple:

```
while(stack.isEmpty()) {
  lock.wait();
}
```

Et avec des ReentrantLock ?

On ne peut pas utiliser foo.wait()/foo.notify() car il suppose synchronized(foo)

A partir d'un ReentrantLock

Condition condition = reentrantLock.**newCondition**();

- **wait**: condition.**await**()
- **notify**: condition.**signal**()
- **notifyAll**: condition.**signalAll**()

Await/signal

condition.await() marche exactement comme object.wait() donc

- Lors de l'attente, le ReentrantLock à partir duquel la Condition a été créée est relacher
- Puis si
  - Une autre thread appel signal(), signalAll() ou interrupt(), ou alors à cause d'un spurious wake-up
    - la thread est réveillé et tente re-acquerir le ReentrantLock

condition.signal()/signalAll() marche comme object.notify/notifyAll donc

- Il ne peuvent être exécuter que si la thread courante possède le ReentrantLock à partir duquel la Condition a été créé

Avec ReentrantLock.newCondition()

```
public class Holder {
  private int value;
  private boolean done;
  private final ReentrantLock lock = new ReentrantLock();
  private final Condition condition = lock.newCondition();

  private void init() {
    lock.lock();
    try {
       value = 12;
       done = true;
       condition.signal();
    } finally {
       lock.unlock();
    }
  }

  private void display() {
    lock.lock();
    try {
      while(!done) {
        condition.await();
      }
      System.out.println(holder.value);
    }
  } finally {
    lock.unlock();
  }
  ...
}
```

On créé la Condition sur le ReentrantLock

On utilise signal() et await() que entre lock() et unlock()

Condition

Contrairement à synchronized + wait/notify, l'API des Condition permet de créer plusieurs conditions pour un même ReentrantLock

Pratique pour indiquer des conditons qui ont lieu par rapport à un même champs

- Par ex: pile pleine, pile vide

Exemple

J'ai 5 threads que je veux démarrer et mettre en attente toute sur une barrière

- Par exemple, pour simuler un jeu en tour par tour

Si une thread atteint la barrière

- Elle est bloquée tant que les autres threads n'ont pas atteint la barrière
- La dernière thread qui atteint la barrière, libère toutes les autres threads en attente

Comment procéder ?

Une thread doit attendre au moins une autre donc

- synchronized + wait/notify ou alors
- ReentrantLock + Condition + await/signal

Que faut-il d'autre ?

Comment procéder ? (2/2)

Une thread doit attendre au moins une autre donc

- synchronized + wait/notify ou alors
- ReentrantLock + Condition + await/signal

Que faut-il d'autre ?

Une variable qui sert de condition !!!

Exemple / Code

```
public class Barrier {
  private int party;

  public Barrier(int party) {
    this.party = party;
  }

  public void waitAt() {
    // réveille les autres threads si c'est la dernière
    // arrète la thread si ce n'est pas la dernière
  }
}
```

Exemple / Code (2/3)

```
public class Barrier {
  private int party;
  private final Object lock = new Object();

  public Barrier(int party) {
    this.party = party;
  }

  public void waitAt() throws InterruptedException {
    synchronized(lock) {
      party--;
      if (party == 0) {
        lock.notifyAll();
        return;
      }
      while (party != 0) {
        lock.wait();
      }
    }
  }
}
```

Et si ...

On veut que si l'on interrompt une thread participant à la barrière, alors les autres doivent aussi lever une InterruptedException

Exemple / Code (3/3)

```
public class Barrier {
  private int party;
  private boolean interrupted;
  private final Object lock = new Object();
  ...
  public void waitAt() throws InterruptedException {
    synchronized(lock) {
      party--;
      if (party == 0) {
        lock.notifyAll();
        return;
      }
      while (!interrupted && party != 0) {
        try {
          lock.wait();
        } catch(InterruptedException e) {
           interrupted = true;
           lock.notifyAll();
           throw e;
        }
      }
      if (interrupted) {
        throw new InterruptedException();
      }
    }
  }
}
```

On ajoute un flag pour savoir lorsque l'on est reveillé si c'est parce que toutes les threads ont atteint la barrière ou si c'est parce qu'une des threads a été interrompu