Compte Rendu Tp3

BOURENNANE Amine

Exercice 1

1. Une classe est dîte non *thread-safe* quand on ne peut pas l'utiliser dans un programme qui utilise des threads sans y avoir de bugs reliés.
2. La classe n'est pas *thread-safe* car on retrouve les noms complets `Jane Doe` et `John Odd` alors qu'à aucun moment dans le code on les trouve.
3.

```
public class HonorBoard {
  private String firstName;
  private String lastName;
  private final Object lock = new Object();

  public void set(String firstName, String lastName) {
    synchronized (lock) {
      this.firstName = firstName;
      this.lastName = lastName;
    }
  }

  @Override
  public String toString() {
    synchronized (lock) {
      return firstName + ' ' + lastName;
    }
  }
}
```

4. On peut pas car cela crée le même problème qu'avant.

Exercice 2

1. Ré-entrant signifie que plusieurs threads peuvent essayer passer dans cette zone du code.
2.

```
public class HonorBoard {
  private String firstName;
  private String lastName;
  private final ReentrantLock lock = new ReentrantLock();

  public void set(String firstName, String lastName) {
    lock.lock();
    try {
```

```
      this.firstName = firstName;
      this.lastName = lastName;
    } finally {
      lock.unlock();
    }
  }

  @Override
  public String toString() {
    lock.lock();
    try {
      return firstName + ' ' + lastName;
    } finally {
      lock.unlock();
    }
  }
}
```

Exercice 3

1. Car sinon cela pourrait créer des problèmes d'état incohérent dans la mémoire.
2. Un opération bloquante est une opération qui stop l'éxécution de tous les threads sauf celui actif et priviligie son éxécution. La méthode d'instance `interrupt()` sert à stoper l'éxécution de la thread instanciée sur laquelle a été appelé la méthode. Avec un try/catch lors de l'appel bloquant dans le *Runnable* de la thread.

```
public static void main(String[] args) throws InterruptedException {
    var thread = new Thread(() -> {
        while(true){
            ...
            try {
                Thread.sleep(1_000);
            } catch (InterruptedException e) {
                System.out.println("end");
                return ; // ou throw new AssertionError(e);
            }
            ...
        }
    });
    thread.start();
    Thread.sleep(1_000);
    thread.interrupt();
}
```

3. Dans un premier temps on peut le faire dans la partie du code du *Runnable*.

```
private static int slow() {
    var result = 1;
```

```
    for (var i = 0; i < 1_000_000; i++) {
        result += (result * 7) % 513;
    }
    return result;
}

public static void main(String[] args) throws InterruptedException {
    var thread = new Thread(() -> {
        var forNothing = 0;
        while (true) {
            forNothing += slow();
            System.out.println(forNothing);
            if (Thread.interrupted()) {
                System.out.println("end");
                return ;
            }
        }
    });
    thread.start();
    Thread.sleep(1_000);
    thread.interrupt();
}
```

4. On peut throw une *AssertionError* pour interrompre les calculs.

```
private static int slow() {
    var result = 1;
    for (var i = 0; i < 1_000_000; i++) {
        if (Thread.interrupted()) {
            System.out.println("end");
            throw new AssertionError("Interrupted");
        }
        result += (result * 7) % 513;
    }
    return result;
}
```

5.

```
private static int slow() {
    var result = 1;
    for (var i = 0; i < 1_000_000; i++) {
        if (Thread.interrupted()) {
            System.out.println("end");
            throw new AssertionError("Interrupted");
        }
        result += (result * 7) % 513;
    }
    return result;
```

```
}

public static void main(String[] args) throws InterruptedException {
    var thread = new Thread(() -> {
        var forNothing = 0;
        while(true) {
            forNothing += slow();
            try {
                Thread.sleep(1_000);
            } catch (InterruptedException e) {
                System.out.println("end");
                return;
            }
            forNothing += slow();
        }
    });
    thread.start();
    Thread.sleep(1_000);
    thread.interrupt();
}
```

6.

```
private static int slow() throws InterruptedException {
    var result = 1;
    for (var i = 0; i < 1_000_000; i++) {
        if (Thread.interrupted()) {
            throw new InterruptedException("Interrupt");
        }
        result += (result * 7) % 513;
    }
    return result;
}
public static void main(String[] args) throws InterruptedException {
    var thread = new Thread(() -> {
        var forNothing = 0;
        while(true) {
            try {
                forNothing += slow();
                Thread.sleep(1_000);
                forNothing += slow();
            } catch (InterruptedException e) {
                System.out.println("end");
                return;
            }
        }
    });
    thread.start();
    Thread.sleep(1_000);
    thread.interrupt();
}
```

7. La méthode `Thread.interrupted()` vérifie si la thread est interrompue et sinon set à *false* le flag d'interruption tandis que `isInterrupted()` vérifie juste le flag sur thread instancié. On dit qu'elle est mal nommée car elle ne fais pas que vérifié si elle est interrompue.
8.

```
public class Main8 {
    public static void main(String[] args) {
        var list = new ArrayList<Thread>();
        for (var i = 0; i < 4; i++) {
            list.add(new Thread(() -> {
                Count count = new Count();
                for (;;) {
                    try {
                        System.out.println(count);
                        count.add();
                        Thread.sleep(1_000);
                    } catch (InterruptedException e) {
                        return ;
                    }
                }
            }));
            list.get(i).start();
        }
        System.out.println("enter a thread id:");
        try(var scanner = new Scanner(System.in)) {
            while(scanner.hasNextInt()) {
                var threadId = scanner.nextInt();
                if (threadId < 0 || threadId >= 4)
                    continue;
                var thread = list.get(threadId);
                if (!thread.isInterrupted()) {
                    thread.interrupt();
                }
            }
        }
    }
}
public class Count {
    private int count;
    private final ReentrantLock lock = new ReentrantLock();

    public Count() {
        count = 0;
    }

    public void add() throws InterruptedException {
        lock.lock();
        try {
            count++;
            if (Thread.interrupted()) {
                throw new InterruptedException("Interrupt");
```

```
                }
            } finally {
                lock.unlock();
            }
        }

        @Override
        public String toString() {
            lock.lock();
            try {
                return count + "";
            } finally {
                lock.unlock();
            }
        }
    }
```

9.

```
list.forEach(t -> t.setDaemon(true));
```

Conclusion

J'ai consolidé mes connaissances sur les classes thread-safe et j'ai appris à interrompre et à gérer ces interruptions de thread.