**Concurrence**
M1 Informatique 2022-2023
TP noté – jeudi 27 octobre 2022

Nom :
Prénom :
Numéro :

---

- Cette partie de l'examen dure au maximum 45 minutes. Les réponses doivent être données directement sur le sujet.
- Le sujet comporte 6 pages et il y a 6 questions.
- Les seuls documents autorisés sont ceux se trouvant sur la page du cours.

---

**Pour chaque question, cochez oui si vous pensez que l'affirmation est correcte ou cochez non si vous pensez qu'elle est fausse. Si vous ne savez pas, il est conseillé de ne rien cocher (les mauvaises réponses sont comptées en négatif).**

1. On considère le code suivant :

```java
public class Quizz1 {
   public static void main(String[] args) throws InterruptedException {
      var thread1 = Thread.ofPlatform().start(() -> {
         System.out.println("Thread 1");
      });

      var thread2 = Thread.ofPlatform().start(() -> {
         System.out.println("Thread 2");
      });

      thread1.join();

      thread2.join();

      System.out.println("Thread main");
   }
}
```

Lors de l'exécution de `Quizz1`, ...

1. ◯ oui ◯ non ... le `thread2` se termine forcément après le `thread1`.
2. ◯ oui ◯ non ... le `thread main` se termine forcément après le `thread2`.
3. ◯ oui ◯ non ... le `thread1` peut commencer à s'exécuter avant que le `thread main` commence à s'exécuter.
4. ◯ oui ◯ non ... le `thread2` peut commencer à s'exécuter avant que le `thread1` commence à s'exécuter.
5. ◯ oui ◯ non ... on peut voir l'affichage `"Thread main"` avant `"Thread 2"`.
6. ◯ oui ◯ non ... le `thread2` est de-schedulé au moins une fois.

2. On considère le code suivant :

```java
public class Quizz2 {
   private ArrayList<Integer> list = new ArrayList<>();
   private String flag = "before";

   public void go() {
      Thread.ofPlatform().start(() -> {
         list.add(1);
         list.add(2);
         flag = "after";
         var a = list.get(0);
      });

      while (true) {
         if (flag.equals("after")) {
            break;
         }
      }
      System.out.println(flag);
      System.out.println(list);
   }

   public static void main(String[] args) {
      new Quizz2().go();
   }
}
```

Lors de l'exécution de `Quizz2`, ...

1. ◯ oui ◯ non ... l'affichage de `flag` peut être `"null"`.
2. ◯ oui ◯ non ... l'affichage de `flag` peut être `"before"`.
3. ◯ oui ◯ non ... l'affichage de `flag` peut être `"after"`.
4. ◯ oui ◯ non ... l'affichage de `list` peut être `"null"`.
5. ◯ oui ◯ non ... l'affichage de `list` peut être `"[1]"`.
6. ◯ oui ◯ non ... l'affichage de `list` peut être `"[1, 2]"`.
7. ◯ oui ◯ non ... l'exécution de l'instruction `flag = "after"` peut avoir lieu avant `list.add(2)`.
8. ◯ oui ◯ non ... l'exécution de l'instruction `list.add(2)` peut avoir lieu avant `list.add(1)`.
9. ◯ oui ◯ non ... l'exécution de `list.get(0)` peut provoquer une exception.
10. ◯ oui ◯ non ... l'exécution de `System.out.println(list)` peut avoir lieu avant la fin de l'exécution de `System.out.println(flag)`.
11. ◯ oui ◯ non ... l'exécution de `System.out.println(flag)` peut avoir lieu avant la fin de l'exécution de la boucle `while`.
12. ◯ oui ◯ non ... on fait de l'attente active.
13. ◯ oui ◯ non ... le programme peut ne pas s'arrêter.
14. ◯ oui ◯ non `Quizz2` a au moins un problème de publication.

3. On considère le code suivant :

```java
public class Quizz3 {
   private int alice;
   private int bob;
   private final Object lock = new Object();

   public void go() {
      Thread.ofPlatform().start(() -> {
         synchronized(lock){
            alice = 1;
            bob = 2;
         }
      });
      while (true) {
         synchronized(lock){
            if(alice == 1){
               break;
            }
         }
      }

      synchronized(lock){
         System.out.println(alice + " " + bob);
      }
   }

   public static void main(String[] args) {
      var quizz = new Quizz3();
      quizz.go();
   }
}
```

Lors de l'exécution de `Quizz3`...

1. ◯ oui ◯ non ... l'affichage peut être `"1 0"`.
2. ◯ oui ◯ non ... l'affichage peut être `"0 2"`.
3. ◯ oui ◯ non ... l'affichage peut être `"1 2"`.
4. ◯ oui ◯ non ... on fait de l'attente active.
5. ◯ oui ◯ non ... le programme peut ne pas s'arrêter.
6. ◯ oui ◯ non `Quizz3` a au moins un problème de publication.

4. On considère le code suivant :

```java
public class Quizz6 {
   private int alice;
   private int bob;
   private final Object lock = new Object();

   public String bob(int value) {
      synchronized (lock) {
         bob = value;
         return alice + " " + bob;
      }
   }

   public String aliceAndBob(int value1, int value2) {
      synchronized (lock) {
         alice = value1;
         bob = value2;
         return alice + " " + bob;
      }
   }

   public String two() {
      synchronized (lock) {
         return alice + " " + bob;
      }
   }

   public static void main(String[] args) {
      var quizz = new Quizz6();
      Thread.ofPlatform().start(() -> System.out.println("1) " + quizz.bob(1)));
      Thread.ofPlatform().start(() -> System.out.println("2) " + quizz.alice + " " + quizz.bob));
      Thread.ofPlatform().start(() -> System.out.println("3) " + quizz.aliceAndBob(3, 4)));
      Thread.ofPlatform().start(() -> System.out.println("4) " + quizz.two()));
   }
}
```

Lors de l'exécution de `Quizz6`, une des lignes peut afficher :

1. ◯ oui ◯ non ... `1) 0 0.`
2. ◯ oui ◯ non ... `1) 0 1.`
3. ◯ oui ◯ non ... `1) 0 4.`
4. ◯ oui ◯ non ... `1) 3 1.`
5. ◯ oui ◯ non ... `1) 3 4.`
6. ◯ oui ◯ non ... `2) 0 0.`
7. ◯ oui ◯ non ... `2) 0 1.`
8. ◯ oui ◯ non ... `2) 0 4.`
9. ◯ oui ◯ non ... `2) 3 1.`
10. ◯ oui ◯ non ... `2) 3 4.`
11. ◯ oui ◯ non ... `3) 0 0.`
12. ◯ oui ◯ non ... `3) 0 1.`
13. ◯ oui ◯ non ... `3) 0 4.`
14. ◯ oui ◯ non ... `3) 3 1.`
15. ◯ oui ◯ non ... `3) 3 4.`
16. ◯ oui ◯ non ... `4) 0 0.`
17. ◯ oui ◯ non ... `4) 0 1.`
18. ◯ oui ◯ non ... `4) 0 4.`
19. ◯ oui ◯ non ... `4) 3 1.`
20. ◯ oui ◯ non ... `4) 3 4.`

5. On considère le code suivant :

```java
public class Quizz4 {
   private int alice;
   private int bob;
   private boolean two;
   private final Object lock = new Object();

   public void go() {
      synchronized (lock) {
         alice = 1;
         lock.notify();
         bob = 2;
         two = true;
         lock.notify();
      }
   }

   public String checkAlice()
            throws InterruptedException {
      synchronized (lock) {
         while (alice != 1) {
            lock.wait();
         }
         return alice + " " + bob;
      }
   }

   public String checkBob()
            throws InterruptedException {
      synchronized (lock) {
         while (!two) {
            lock.wait();
         }
      }
      return alice + " " + bob;
   }
```

```java
   public static void main(String[] args) {
      var quizz = new Quizz4();

      var threadA = Thread.ofPlatform().start(() -> {
         try {
            System.out.println("Alice " + quizz.checkAlice());
         } catch (InterruptedException e) {
            throw new AssertionError(e);
         }
      });

      var threadB = Thread.ofPlatform().start(() -> {
         try {
            System.out.println("Bob " + quizz.checkBob());
         } catch (InterruptedException e) {
            throw new AssertionError(e);
         }
      });

      quizz.go();
   }
}
```

Lors de l'exécution de `Quizz4`...

1. ◯ oui ◯ non ... l'un des affichages peut être `"Alice 1 0"`.
2. ◯ oui ◯ non ... l'un des affichages peut être `"Alice 1 2"`.
3. ◯ oui ◯ non ... l'un des affichages peut être `"Bob 0 0"`.
4. ◯ oui ◯ non ... l'un des affichages peut être `"Bob 1 2"`.
5. ◯ oui ◯ non ... on fait de l'attente active.
6. ◯ oui ◯ non ... l'objet `lock` suit les bonnes pratiques du cours pour servir de verrou.
7. ◯ oui ◯ non ... le programme peut ne pas s'arrêter car la méthode `checkBob()` peut être bloquée dans son `wait()`.
8. ◯ oui ◯ non ... il peut y avoir de notification perdue (reçue par aucun thread) lors de l'exécution.
9. ◯ oui ◯ non ... le `threadB` peut être réveillé deux fois à cause de la méthode `go()`.
10. ◯ oui ◯ non ... le `threadB` ne peut être réveillé d'un appel à wait() qu'après que le `threadA` ait fini d'exécuter la méthode `checkAlice()`.
11. ◯ oui ◯ non ... le `threadA` ne peut sortir de sa boucle `while` qu'après que le thread `main` ait fini d'exécuter la méthode `go()`.
12. ◯ oui ◯ non ... le `threadA` ne peut être réveillé (shédulable) d'un appel à `wait()` qu'après que le thread `main` ait fini d'exécuter la méthode `go()`.
13. ◯ oui ◯ non ... le `threadB` ne peut finir d'exécuter `checkBob()` qu'après que le `threadA` ait fini d'exécuter la méthode `checkAlice()`.
14. ◯ oui ◯ non ... le `threadA` ne peut pas prendre le `lock` tant que le thread `main` est dans le `synchronized` de la méthode `go()`.
15. ◯ oui ◯ non ... le `threadB` ne peut pas prendre le `lock` tant que le `threadA` n'a pas fini d'exécuter tout le bloc `synchronized` de la méthode `checkAlice()`.

6. On considère le code suivant :

```java
public class Quizz5 {
   private int yes;
   private int no;
   private final int players;
   private final Object lock = new Object();

   public record Result(int yes, int no) {};

   public Quizz5(int players) {
      if (players < 0) {
         throw new IllegalArgumentException();
      }
      synchronized (lock) {
         this.players = players;
      }
   }

   public void yes() {
      synchronized (lock) {
         yes++;
         lock.notifyAll();
      }
   }

   public void no() {
      synchronized (lock) {
         no++;
         lock.notifyAll();
      }
   }

   public Result result() throws InterruptedException {
      synchronized (lock) {
         while (yes + no < players) {
            lock.wait();
         }
         return new Result(yes, no);
      }
   }

   public static void main(String[] args) {
      var quizz = new Quizz5(5);
      var t1 = Thread.ofPlatform().daemon().start(() -> {
         for (;;) {
            quizz.yes();
         }
      });
      var t2 = Thread.ofPlatform().daemon().start(() -> {
         for (;;) {
            quizz.no();
         }
      });
      var t3 = Thread.ofPlatform().start(() -> {
         try {
            System.out.println(quizz.result());
         } catch (InterruptedException e) {
            throw new AssertionError(e);
         }
      });
   }
}
```

1. ◯ oui ◯ non `Quizz5` est thread-safe.
2. ◯ oui ◯ non `Result` est thread-safe.
3. ◯ oui ◯ non `Quizz5` a un problème de publication.
4. ◯ oui ◯ non `Result` a un problème de publication.
5. ◯ oui ◯ non Le bloc `synchronized` dans le constructeur de `Quizz5` est inutile.

Lors de l'exécution de `Quizz5` ...

1. ◯ oui ◯ non ... le programme s'arrête parce que les threads `t1` et `t2` sont `daemon`.
2. ◯ oui ◯ non ... le programme s'arrête parce que le thread `t3` est n'est pas `daemon`.
3. ◯ oui ◯ non ... le programme ne s'arrête plus si on remplace un des `notifyAll` par un `notify()`.
4. ◯ oui ◯ non ... le programme ne s'arrête plus si on remplace les deux `notifyAll` par un `notify()`.
5. ◯ oui ◯ non ... la somme des valeurs de `yes` et `no` affichées (dans le `Result`) peut valoir moins de 5.
6. ◯ oui ◯ non ... la somme des valeurs de `yes` et `no` affichées (dans le `Result`) peut valoir 5.
7. ◯ oui ◯ non ... la somme des valeurs de `yes` et `no` affichées (dans le `Result`) peut valoir plus de 5.