プログラミングA 第9回・演習

演習の提出は Moodle で月曜日までに行ってください。各演習ごとに提出ファイルを zip 等で一つのファイルにまとめて該当する Moodle の課題に提出しなさい。この資料や関係するコードをインターネットなどに公開することは著作権上、禁止されています。

1 演習 1

30 番目から 39 番目までの 10 個のフィボナッチ数を、マルチスレッドによる並行実行により求める場合と、シングルスレッドで直列実行により求める場合とで、計算にかかる時間を比較したい。添付の FibonacciThread における「// HERE」の箇所に、FibonacciThread クラスのインスタンスを、求める 30〜39 番目のフィボナッチ数のそれぞれに対して一つずつ生成し、それらの 10 個のスレッドを並行実行して最後に待ち合わせるコードを記述し、コードを提出せよ。

完成した FibonacciThread の実行により以下のような出力を標準出力に得ること。ただしスレッドの出力順序や計算時間は環境により異なる。

```
Thread[Thread-1,5,main]2178309
Thread[Thread-0,5,main]1346269
Thread[Thread-2,5,main]3524578
Thread[Thread-4,5,main]9227465
Thread[Thread-3,5,main]5702887
Thread[Thread-5,5,main]14930352
Thread[Thread-6,5,main]24157817
Thread[Thread-7,5,main]39088169
Thread[Thread-8,5,main]63245986
Thread[Thread-9,5,main]102334155
Time spent for multi-threaded execution: 626
Thread[main,5,main]1346269
Thread[main,5,main]2178309
Thread[main,5,main]3524578
Thread[main,5,main]5702887
Thread[main,5,main]9227465
Thread[main,5,main]14930352
Thread[main,5,main]24157817
Thread[main,5,main]39088169
Thread[main,5,main]63245986
Thread[main,5,main]102334155
Time spent for single-threaded execution: 979
```

参考資料：FibonacciThread.java

```
public class FibonacciThread extends Thread {
```

```
  private int value = 0;

  public FibonacciThread(int v) {
    value = v;
  }

  public void run() {
    printFibonacci(value);
  }

  public static long fibonacci(int n) {
    return (n == 0 || n == 1) ? 1 : fibonacci(n - 1) + fibonacci(n - 2);
  }

  public static void printFibonacci(int n) {
    System.out.println(Thread.currentThread() + "" + fibonacci(n));
  }

  public static void main(String[] args) {
    long previousTime = 0;

    previousTime = System.currentTimeMillis();
    // HERE
    System.out.println("Time spent for multi-threaded execution: "
        + (System.currentTimeMillis() - previousTime));

    previousTime = System.currentTimeMillis();
    for(int i = 30; i < 40; i++) {
      printFibonacci(i);
    }
    System.out.println("Time spent for single-threaded execution: "
        + (System.currentTimeMillis() - previousTime));
  }
}
```

2 演習 2

クラス ThreadEx2 を実行すると、一方のスレッド（ThreadXX2 または ThreadYY2）からの標準出力への出力が連続し、続いて、他方のスレッドからの出力が連続することが繰り返される。一方が出力したら、続いて必ず他方が出力するように（交互に出力するように）必要なクラスを修正し、全てを提出せよ。

ヒント: synchronized, wait(), notifyAll() を用いる。Thread.currentThread() を活用する。

現状の出力例:

```
XX
XX
```

```
...
XX
YY
YY
...
YY
XX
...
```

期待する出力例（開始は XX と YY のいずれでもよい）:

```
XX
YY
XX
YY
XX
...
```

3 演習 3

QueueClient を実行すると、QueueProducer と QueueConsumer の両スレッドがそれぞれ並行実行される。QueueProducer はキューに 30 個整数を追加（enqueue）し、QueueConsumer はキューから整数をあるだけ取得（dequeue）しようとする。しかしその実行においてたいてい、キューがいっぱいにも関わらず整数を追加しようとしたり、キューが空にも関わらず整数を取得しようとして、例外が投げられて異常終了してしまう。この問題を解決するため、元の Queue をスレッドセーフとなるように修正し、enqueue においてキューがいっぱいであればそれを実行したスレッドをいったん待たせ、同様に、dequeue においてキューが空であればそれを実行したスレッドをいったん待たせるようにせよ。その修正に対応する形で QueueProducer および QueueConsumer も修正し、Queue・QueueProducer・QueueConsumer の全てのコードを提出せよ。

4 演習 4

3 人の作業員（Worker）がいる。3 人は働きつつ（work）、しばしば休憩所（Room）で休憩（rest）を取りたい。ただし休憩所の所要人数は 1 名に限られている。この状況を Worker, Room として作成したが、WorkerClient の main() を実行すると以下のように表示されてしまう。つまり、誰かが休憩している間は、他の 2 名は休憩所に入ることができず、かといって働くこともせず、待たされることになってしまった。

```
Start resting :Thread-0
End resting :Thread-0
    Work : Thread-0
Start resting :Thread-1
End resting :Thread-1
```

```
    Work : Thread-1
Start resting :Thread-2
End resting :Thread-2
    Work : Thread-2
```

そこで、誰かが休憩所で休憩している場合は、他の 2 名は待たされずに直ちに働くように Room を修正し、修正した結果の全体を提出せよ。

修正後に、再度 WorkerClient の main() を実行すると以下のように表示されること（あくまで一例であり、スレッドの登場順は実行環境やタイミングにより異なる）。

```
Start resting :Thread-0
    Work : Thread-1
    Work : Thread-2
    Work : Thread-2
End resting :Thread-0
    Work : Thread-0
Start resting :Thread-2
    Work : Thread-1
    Work : Thread-0
    Work : Thread-1
    Work : Thread-1
    Work : Thread-0
End resting :Thread-2
```

5 演習 5

添付の LockTest を実行して、二つのスレッド ThreadA および ThreadB が並行動作することで以下に示すような出力を標準出力に得て終了したい（ただし出力は一例であり、環境や状況により出力順は異なる）。しかし LockTest を実行すると大抵の場合、プログラムの進行が止まってしまう。

(5-1) その理由を説明せよ。
(5-2) 期待通りに動作するように ThreadB クラスを修正し、修正後の ThreadB を提出せよ。

期待する出力の例:

```
Thread[Thread-0,5,main] doX(0)
Thread[Thread-0,5,main] doY(0)
Thread[Thread-0,5,main] doX(1)
Thread[Thread-0,5,main] doY(1)
Thread[Thread-0,5,main] doX(2)
Thread[Thread-0,5,main] doY(2)
Thread[Thread-0,5,main] doX(3)
```

```
Thread[Thread-0,5,main] doY(3)
Thread[Thread-0,5,main] doX(4)
Thread[Thread-0,5,main] doY(4)
Thread[Thread-1,5,main] doX(0)
Thread[Thread-1,5,main] doY(0)
Thread[Thread-1,5,main] doX(1)
Thread[Thread-1,5,main] doY(1)
Thread[Thread-1,5,main] doX(2)
Thread[Thread-1,5,main] doY(2)
Thread[Thread-1,5,main] doX(3)
Thread[Thread-1,5,main] doY(3)
Thread[Thread-1,5,main] doX(4)
Thread[Thread-1,5,main] doY(4)
Thread[Thread-0,5,main] doX(5)
Thread[Thread-0,5,main] doY(5)
Thread[Thread-0,5,main] doX(6)
Thread[Thread-0,5,main] doY(6)
Thread[Thread-0,5,main] doX(7)
Thread[Thread-0,5,main] doY(7)
Thread[Thread-0,5,main] doX(8)
Thread[Thread-0,5,main] doY(8)
Thread[Thread-0,5,main] doX(9)
Thread[Thread-0,5,main] doY(9)
Thread[Thread-1,5,main] doX(5)
Thread[Thread-1,5,main] doY(5)
Thread[Thread-1,5,main] doX(6)
Thread[Thread-1,5,main] doY(6)
Thread[Thread-1,5,main] doX(7)
Thread[Thread-1,5,main] doY(7)
Thread[Thread-1,5,main] doX(8)
Thread[Thread-1,5,main] doY(8)
Thread[Thread-1,5,main] doX(9)
Thread[Thread-1,5,main] doY(9)
```