Chapter 7

上級プログラミング 1 最終課題

以下の指示に従って、クラス、オブジェクトを作成・実行せよ。なお、文中では個別に指示しないが、変更を行ったら適宜コンパイルすること。本課題はレポート提出を求める。締め切りは 6 月 5 日（水）23:59 までとする。Scomb にて提出すること。レポートを PDF 形式で提出し、加えてソースファイル Calculator.java を提出せよ。

7.1 スレッドを使った数値加算プログラム

本課題の最後に CalcNode.java と CalcMain.java のソースプログラムを提示している。これらを利用して、数字の羅列をファイルから読み取って和を求めるプログラムを作るが、スレッドを利用して分散処理を行う練習も兼ねる。以下の仕様を満たす Calculator クラスを実装せよ。

- フィールドとして次の 3 つの変数を用意せよ [1]。これらを適切に初期化せよ。
  - `List<Integer> dataList`
  - `List<CalcNode> nodeList`
  - `long sum`
- 以下の処理を行うコンストラクタを作れ。ただし、コンストラクタは String 型の変数 fileName を引数にとる。
  - fileName で指定されるテキストファイルから 1 行ずつ読み込む。
  - 読み込んだ各行を整数値に変換し、フィールドのリスト `dataList` に順次追加する。

[1] Integer クラスは int 型のラッパークラスである。

- 次の処理を行うメソッド createNodes を作れ。ただし、引数は int 型の変数 n のみをとり、戻り値なし (void) とする。
  - `n` 回、以下を繰り返す。`i` 番目の繰り返しのとき、
    - CalcNode クラスのオブジェクト `node` を生成する。このとき、コンストラクタの引数には Calculator オブジェクトを渡す必要があるが、このクラスのオブジェクト自身を使いたいので、this 変数を引数に入れる。
    - `node` の assign メソッドを呼び出し、引数にはフィールド `dataList`、`i`、`n` を渡す。
    - オブジェクト `node` を、リスト `nodeList` に追加する。
    - CalcNode クラスはスレッドクラスを継承しているので、オブジェクト `node` を別スレッドとして処理を開始する。
- 次の処理を行うメソッド addSum を作れ。ただし、引数は long 型の変数 sum のみをとり、戻り値なし (void) とする。また、一時点で単一のスレッドからのみ実行できるようにせよ。
  - 引数 `sum` の値をフィールド `sum` に加える。
- 次の処理を行うメソッド getSum を作れ。ただし、引数はなし、戻り値は long 型であるとせよ。また、メソッド内で適切に例外処理を行うこと。
  - フィールド `nodeList` はリストであり、その要素それぞれが別スレッドとして動作しているため、これらがすべて処理を終えるまで処理を停止する (join メソッドを使え)。
  - これらのスレッドがすべて終了したら、フィールド `sum` の値を返す。

以上のコンストラクタとメソッドを持つ Calculator クラスをソースファイル Calculator.java に実装せよ [2]。

これらのソースコードをコンパイルし、実行してみよ。この結果と CalcMain.java の main メソッドにある `calc.createNodes(3)` を `calc.createNodes(1)` に変えたときに得られる結果を比較し、実施している処理についての解説を交えてなぜそのような結果になるかについて説明せよ。

また、他にもさまざまな整数 n について `calc.createNode(`n`)` を試し、考察するとよい。

[2] 他の名前のファイルにソースコードを含めないこと。

【CalcNode.java】

```
import java.util.List;

// 並行に動作する計算ノード
public class CalcNode extends Thread {
  List<Integer> dataList;
  int myId;
  int numNodes;
  Calculator calc;

  CalcNode(Calculator calc) {
    this.calc = calc;
  }

  void assign(List<Integer> list, int i, int n) {
    this.dataList = list;
    this.myId = i;
    this.numNodes = n;
  }

  public void run() {
    long sum = 0;
    int numData = this.dataList.size();
    long start = System.nanoTime();
    // このノードの担当範囲の加算を実行
    for (int i = this.myId * numData / this.numNodes;
              i < (this.myId + 1) * numData / this.numNodes; i++) {
      sum += this.dataList.get(i);
      try {
        Thread.sleep(1); // 計算コストの代わり
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    calc.addSum(sum); // 計算結果を全体の sum に加える
    long end = System.nanoTime();
    System.out.println("ThreadTime:" + (end - start) / 1000000f + "ms");
  }
}
```

【CalcMain.java】

```
// メインクラス
public class CalcMain {

  public static void main(String[] args) {
    Calculator calc = new Calculator("data.txt");
    long start = System.nanoTime();
    calc.createNodes(3);
    System.out.println("SUM:" + calc.getSum());
    long end = System.nanoTime();
    System.out.println("Time:" + (end - start) / 1000000f + "ms");
  }
}
```