

情報科学演習\_第3期課題  
マルチスレッドプログラミング

6321120

横溝尚也

提出日：12月 8日（木）

## 1 課題 1：座席プログラム

座席予約システムのプログラムを作成せよ。

以下のプログラムを改変しても良いし、一から作成しても良い。

## 2 プログラムの説明

```
1 public class Reservation{
2     int seat[] [];

3     public Reservation(int n, int m){
4         seat = new int[n][m];
5         for(int i=0;i<n;i++){
6             for(int j=0;j<m;j++){
7                 seat[i][j]=-1;
8             }
9         }
10    }

11    boolean reserve(int id, int num){ //変更箇所
12        int x_coordinate = (int)(Math.random()*15);
13        int y_coordinate = (int)(Math.random()*(7-num));

14        for(int i = 0; i < num; i++){
15            if(seat[y_coordinate + i][x_coordinate] == -1){
16            }else{
17                return false;
18            }
19        }

20        for(int i = 0; i < num; i++){
21            seat[y_coordinate + i][x_coordinate] =1;
22        }
23        return true;
24    }

25    void printSeat(){ //変更箇所
26        int sum = 0;
27        for(int i=0;i<seat.length;i++){
```

```

28         for(int j=0;j<seat[i].length;j++){
29             System.out.print(seat[i][j]+"\\t");
30             if(seat[i][j]==1){sum = sum + 1;}
31         }
32 System.out.println();
33     }
34     System.out.println("sum =" + sum);
35 }

36 public void res(){ //追加箇所
37     int id = Passengers.id;
38     Reservation rs = Passengers.rs;
39     synchronized(this){
40         for(int i=0;i<10;i++){
41             int num = (int)(Math.random()*6+1);
42             if(rs.reserve(id, num)){
43                 System.out.println("ID:"+id+" reserved "+num+" seats. "\\
44 );} }
45     }
46 }

47 public static void main(String args[]){
48     int thread_num = 5;
49     Reservation rs = new Reservation(6,15);
50     Passengers ps[] = new Passengers[thread_num];
51     for(int i=0;i<thread_num;i++){
52         ps[i] = new Passengers(i,rs);
53     }
54     for(int i=0;i<thread_num;i++){
55         try{
56             ps[i].join();
57         }catch(InterruptedException e){
58         }
59     }
60     rs.printSeat();
61 }
62}

63class Passengers extends Thread{
static int id;

```

```

64     static Reservation rs;
65     public Passengers(int n, Reservation rs){
66         this.id = n;
67         this.rs = rs;
68         this.start();
69     }
70     public void run(){ //res メソッドに記述を移動した
71         rs.res();
72     }
73}

```

1～10行目：2次元配列 seat のすべての要素に-1（空席）を初期入力する

11～24行目：メソッド reserve の記述。どの列の予約を試みるかランダムに選ぶために1から15の数をランダムに出力する x\_coordinate を出力した。y\_coordinate は6行の席に顧客の予約席数（例えば3人の予約）をするときに、その3席の上端の席を表している。

予約試行時にその席のどこが一つでも満席 (1) ならば試行失敗の false, すべての席が空席 (-1) ならば予約成功の true を返す。

25～35行目：座席の空席状況を出力する部分は授業資料と同じ。さらに予約状況と座席の空席状況が一致することを示すために、満席にした席数をカウントする sum を宣言している。

36～46行目：run メソッドで実行すべきことを新たな res メソッドに記述している。

47～62行目：メイン関数内では顧客数と席数（ $6 \times 15$ ）、配列 ps に顧客の予約情報を格納する。

63～73行目：メイン関数から呼び出される Passenger メソッドには Reservation メソッドを呼び出して id を登録している。

### 3 実行結果

```

tusedls00.ed.tus.ac.jp - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
tusedls06$ java Reservation
ID:1 reserved 5 seats.
ID:1 reserved 5 seats.
ID:1 reserved 6 seats.
ID:1 reserved 5 seats.
ID:1 reserved 1 seats.
ID:1 reserved 3 seats.
ID:1 reserved 5 seats.
ID:1 reserved 2 seats.
ID:1 reserved 3 seats.
ID:4 reserved 1 seats.
ID:4 reserved 5 seats.
ID:4 reserved 1 seats.
ID:4 reserved 6 seats.
ID:4 reserved 2 seats.
ID:3 reserved 1 seats.
ID:4 reserved 6 seats.
ID:4 reserved 3 seats.
-1 1 -1 1 -1 -1 -1 1 1 1 -1 1 1 -1 -1
1 1 -1 1 1 1 -1 1 1 1 1 1 1 -1 -1
1 1 -1 1 1 1 -1 1 1 1 1 1 1 -1 -1
1 1 1 1 1 1 -1 1 -1 1 1 1 1 1 1
1 1 1 1 1 -1 1 1 -1 1 -1 1 1 -1 1
1 1 1 -1 1 -1 -1 -1 1 1 -1 1 1 -1 -1
sum =60
tusedls06$

```

図 1 実行結果 1

```

tusedls00.ed.tus.ac.jp - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
tusedls06$ java Reservation
ID:1 reserved 3 seats.
ID:1 reserved 6 seats.
ID:1 reserved 5 seats.
ID:1 reserved 2 seats.
ID:1 reserved 1 seats.
ID:1 reserved 3 seats.
ID:1 reserved 6 seats.
ID:1 reserved 1 seats.
ID:1 reserved 1 seats.
ID:4 reserved 1 seats.
ID:4 reserved 2 seats.
ID:4 reserved 6 seats.
ID:4 reserved 6 seats.
ID:4 reserved 2 seats.
ID:4 reserved 1 seats.
ID:4 reserved 6 seats.
ID:4 reserved 1 seats.
ID:4 reserved 5 seats.
ID:2 reserved 2 seats.
1 1 -1 -1 1 -1 -1 1 1 1 -1 -1 1 1 -1
1 1 -1 1 1 1 -1 -1 1 1 -1 -1 1 1 1
1 -1 1 1 1 1 -1 1 1 1 -1 -1 1 1 1
1 -1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 -1 1 -1 -1 -1 1 1 -1 1 1 1 1
1 -1 -1 -1 1 -1 -1 1 1 1 -1 -1 -1 1 1
sum =60
tusedls06$

```

図 2 実行結果 1

```

tusedls00.ed.tus.ac.jp - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
tusedls00$ java Reservation
ID:1 reserved 1 seats.
ID:1 reserved 2 seats.
ID:1 reserved 1 seats.
ID:1 reserved 2 seats.
ID:1 reserved 3 seats.
ID:1 reserved 3 seats.
ID:1 reserved 2 seats.
ID:4 reserved 5 seats.
ID:4 reserved 1 seats.
ID:4 reserved 2 seats.
ID:4 reserved 1 seats.
ID:4 reserved 2 seats.
ID:4 reserved 4 seats.
ID:3 reserved 1 seats.
ID:3 reserved 4 seats.
ID:3 reserved 2 seats.
ID:3 reserved 4 seats.
ID:3 reserved 3 seats.
ID:3 reserved 1 seats.
ID:3 reserved 5 seats.
ID:3 reserved 1 seats.
ID:4 reserved 1 seats.
ID:4 reserved 1 seats.
1 -1 -1 -1 -1 1 1 -1 -1 1 -1 1 -1 -1 -1
1 -1 1 -1 1 -1 1 -1 1 -1 1 1 -1 -1 -1
1 -1 -1 1 1 1 1 -1 1 -1 1 1 -1 1 1
1 1 1 -1 1 1 1 1 1 1 1 1 -1 1 1
-1 1 1 -1 -1 1 1 -1 1 1 1 1 -1 1 1
-1 1 -1 -1 -1 1 -1 -1 1 1 1 1 -1 1 -1
sum =52
tusedls00$

```

図 3 実行結果 1

## 4 考察

### 4.1 実行結果からわかること

ID と予約席数の出力が初めに行われているが、予約試行が同時に行われてしまうことで席予約がかぶってしまうことが考えられた。しかし、synchronized メソッドを使用することで、予約を行う（空席-1 を満席 1 に変更）する操作にロックをかけて同時予約を防いでいる。それによって、予約席数のカウント sum と 90 席の予約状況 (1 の数) が一致していることから synchronized メソッドがうまく機能していることがわかる。

### 4.2 試行結果の席の埋まり具合について

今回は顧客数 5、一人の顧客につき 10 回の予約試行を行うという状況設定で行った。その際に実行結果を 3 つ提示したが、席の埋まり具合 (sum の結果) は 60,60,52 と 6 割近い席数が埋まる結果となった。これは感覚的にも席の予約状況がこのような結果になることも想像できる。

### 4.3 このプログラムの利用方法

今回は予約システムを作成したため、これを改善していけばレストランや飛行機の予約システムが作成できそうである。

それだけでなく、座席数と顧客数、最大で一人が予約できるのかわかっている状況（例えば東京ドームでライブを行い、これまでどれくらいの人がライブを申し込んでいるかのデータ、おひとり様2席までとわかっている場合）があったとする。この時ライブ申し込み開始時に1回の試行でどれくらいの顧客が予約成功するのかといった確率的側面でもランダム関数を使用していることによっておおよその予測として結果を活用することもできる。また、試行失敗した顧客は再びサイトに入りなおしてほかの席を予約しようとするのが普通だ。その際、新たに訪れる一定の顧客に対して試行失敗した顧客がサイト内にたまることによって、サイトダウン？がいつ起こるのかといった予測にも利用できそうと考えた。

## 5 課題2：ソートの効率化

ソートは並列で行い、以下にサンプルとして記載する逐次バブルソートとの速度比較を考察せよ。  
配列の生成等は以下の Sort クラスを利用し、実際のソート処理を行う BubbleSort クラスの部分を他の並列ソートに変更すること。

## 6 プログラムの説明

```
1public class Sort{
2    private int array[];
3    public Sort(int n){
4        array = new int[n];
5        for(int i=0;i<n;i++){
6            array[i] = (int)(Math.random()*Integer.MAX_VALUE);
7        }
8        //ここからソート呼び出し
9        long start = System.currentTimeMillis();
10       QuickSort qs = new QuickSort(array);
11       array = qs.getArray();
12       //printArray(array); //配列表示
13       long end = System.currentTimeMillis();
14       System.out.println("sort?: "+sortCheck(array)+
15                           ", Processing time: "+(end-start)+"ms");
16   }

    /** ソートチェック */
17   public static boolean sortCheck(int array[]){
18       for(int i=0;i<array.length-1;i++){
19           if(array[i]>array[i+1])return false;
20       }
21       return true;
22   }

    /** 確認用の配列表示メソッド */
23   public static void printArray(int array[]){
24       for(int i=0;i<array.length;i++){
25           System.out.print(array[i]+" ");
26       }
27       System.out.println();
```



```

28     }

29     public static void main(String args[]){
30         new Sort(10000000);
31     }
32}

33class QuickSort{
34     private int array[];

35     QuickSort(int[] n){
36         array = n;
37         SubQuickSort q = new SubQuickSort(array , 0, array.length-1);
38     }

    /** ソート コンストラクタから自動で実行される */

    /** ソート結果を得るメソッド */
39public int[] getArray(){
40     return array;
41}
42}

43class SubQuickSort extends Thread{
44     public int array2[];
45     public int left, right;
46     static int NumOfThread;
47     public SubQuickSort(int[] na, int left, int right){
48 array2 = na;
49 this.left = left;
50 this.right = right;
51 NumOfThread++;
52 this.start();
53 try{
54 this.join();
55 }catch(InterruptedException e){}
56}

57public void run(){
58 sort(array2, 0, array2.length-1);
59}

60private void sort(int[] d, int left, int right){

```

```

61         if(left >= right){return;}
62
63         int p = d[(left+right)/2];
64         int l = left, r = right, a;
65
66         while(l<=r) {
67             while(d[l] < p) { l++; }
68             while(d[r] > p) { r--; }
69
70             if (l<=r) {
71                 a = d[l]; d[l] = d[r]; d[r] = a;
72                 l++; r--;
73             }
74         }
75 if(this.NumOfThread < 4){
76     // if(false){
77     SubQuickSort q1 = new SubQuickSort(d, left, r);
78         //sort(d, left, r); //ピボットより左側をクイックソート
79     SubQuickSort q2 = new SubQuickSort(d, l, right);
80         //sort(d, l, right); // ピボットより右側をクイックソート
81     try{
82         q1.join();
83         q2.join();
84     }catch(InterruptedException e){}
85     }else{
86         sort(d, left, r);
87         sort(d, l, right);
88     }
89 }
90 }
91 }

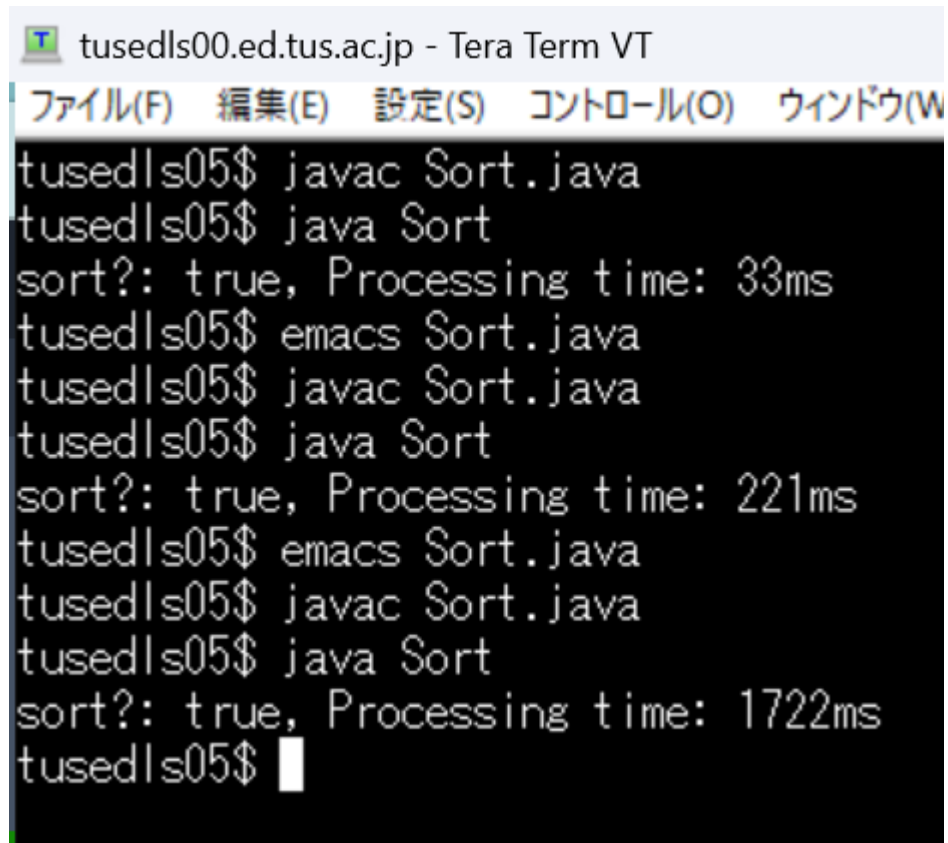
```

1～28行目：ソートしたいものを配列に格納し、ソートチェックでは大きい順に配列の要素がすべて並んでいるかチェックして bool 型で返している。確認用の配列表示メソッドはソートしたい要素数が大きくなればなるほど配列表示が長くなるので基本的に出力することはしていない。

29～32行目：メイン関数ではソートしたい要素数を指定して実行している。

33行目～：今回自分はクイックソートを並列処理にしたものと授業資料にあるバブルソートの速度比較を行った。そのため、33行目以降はクイックソートに並列処理したものを記述している。クイックソートについては ocaml の授業で以前作成したプログラムのアルゴリズムを参考にし、今回はアルゴリズムとプログラムの説明は省略する。

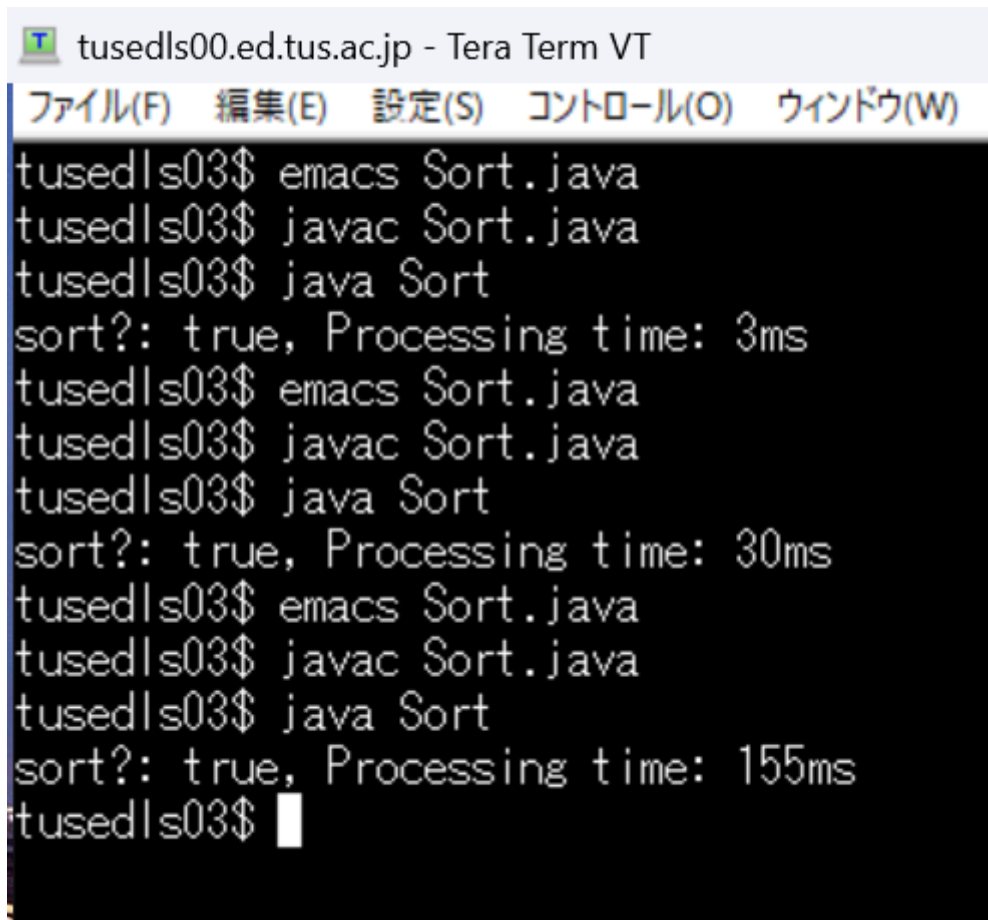
## 7 実行結果



The screenshot shows a terminal window titled "tusedls00.ed.tus.ac.jp - Tera Term VT". The menu bar includes "ファイル(F)", "編集(E)", "設定(S)", "コントロール(O)", and "ウィンドウ(W)". The terminal content shows three sets of commands and their outputs:

```
tusedls05$ javac Sort.java
tusedls05$ java Sort
sort?: true, Processing time: 33ms
tusedls05$ emacs Sort.java
tusedls05$ javac Sort.java
tusedls05$ java Sort
sort?: true, Processing time: 221ms
tusedls05$ emacs Sort.java
tusedls05$ javac Sort.java
tusedls05$ java Sort
sort?: true, Processing time: 1722ms
tusedls05$
```

図 4 並列処理をしたソート数 10 万、100 万、1000 万の実行時間



```
tusedls00.ed.tus.ac.jp - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W)
tusedls03$ emacs Sort.java
tusedls03$ javac Sort.java
tusedls03$ java Sort
sort?: true, Processing time: 3ms
tusedls03$ emacs Sort.java
tusedls03$ javac Sort.java
tusedls03$ java Sort
sort?: true, Processing time: 30ms
tusedls03$ emacs Sort.java
tusedls03$ javac Sort.java
tusedls03$ java Sort
sort?: true, Processing time: 155ms
tusedls03$
```

図5 並列処理していないソート数10万、100万、1000万の実行時間

## 8 考察

### 8.1 バブルソートとクイックソートソート時間の比較

実行結果には記さなかったが、授業資料に記載されているバブルソートのソート時間はソート数10万、100万、1000万それぞれ57,224,2098msであった。それに比べてクイックソートの並列処理を行っていないソートはそれぞれ3,30,155msであった。このことからバブルソートよりもクイックソートのほうがソートの実行時間、平均計算量が少ないことがわかる。調べてたら、マージソートが $n^2$ 回、クイックソートが $n \log n$ 回の計算量であった。これはクイックソートのほうが平均計算量がソート数が増えるにつれて処理速度が上がることを示している。

## 8.2 並列処理をすることによる速度処理の比較

実行結果より図4が並列処理したクイックソート、図5が並列処理していないクイックソートの処理時間である。並列処理したほうが実行時間が短くなることを授業で理解したが、並列処理をしたほうが実行時間が長くなるという理想に反する結果となってしまった。

この原因や改善点を考えたが、わからなかったためこのまま提出します。並列の処理の書き方は授業資料を確認したり、TAの方に確認してもらったりしたため、間違っていないと思う。並列にして処理時間を短縮したい部分の対象が違く手処理時間が短くならない可能性も考えたがわからなかった。