C:\Users\Nick\Documents\CECS327\Homework\Assignment6\src\ArrayThread.java

```
1
2 import java.util.ArrayList;
3 import java.util.Random;
4 import java.util.concurrent.locks.Lock;
5
6 public class ArrayThread extends Thread {
7
8   // Used to index each thread for the main method
9   private final int id;
10   // Used to coordinate access to the shared memory array
11   private final Lock lock;
12   // The number of operations each thread will complete
13   private final int NUM_OPERATIONS = 50;
14   // Used to generate a random number between 0 and 1100 for choosing
15   // a random number
16   private final int MAX_OPERATION_VALUE = 1101;
17   // Shared memory arrays provided by main method. Array is used for search
18   // and replace operations. Pool provides strings to replace array values
19   private String[] array, pool;
20   // Used to average the amount each thread has to wait before searching
21   // or replacing a string
22   ArrayList<Double> searchWaitTimes = new ArrayList<>();
23   ArrayList<Double> replaceWaitTimes = new ArrayList<>();
24
25   // ArrayThread constructor taking an id, a lock on the entire array, and the
26   // shared memory arrays array and pool
27   public ArrayThread(int id, Lock lock, String[] array,
28   String[] pool) {
29     this.id = id + 1;
30     this.lock = lock;
31     this.array = array;
32     this.pool = pool;
33   }
34
35   // Return the shared string array
36   public String[] getArray() {
37     return array;
38   }
39
40   // Takes a randomly generated integer value and performs a task
41   // according to that value
42   public void performTask(int operation) {
43     if (operation >= 0 && operation <= 999) {
44       searchForString();
45     } else {
46       replaceString();
47     }
48   }
49
50   // Searches for the last occurrence of a string, randomly taken from the
51   // pool, in the ARRAY array. The thread locks upon accessing the ARRAY,
52   // ensuring that other threads do not access it at the same time.
53   public void searchForString() {
54
55     // Generate a random string from the pool
56     String poolString = pool[new Random().nextInt(pool.length)];
```

```
57      // Get the initial time for when the thread starts waiting for the lock
58      double startTime = System.nanoTime();
59      // Lock on the shared lock
60      lock.lock();
61      // Get the time for when the thread stops waiting for the lock
62      double endTime = System.nanoTime();
63
64      // Add the wait time in seconds to searchWaitTimes ArrayList
65      searchWaitTimes.add((endTime - startTime));
66
67      try {
68        // Iterate over the array starting at the last element
69        for (int i = array.length - 1; i > 0; i--) {
70
71          // First occurrence of the string in reverse-order iteration
72          // is the last occurrence of the string.
73          if (array[i].equals(poolString)) {
74            break;
75          }
76        }
77      } finally {
78        // Unlock the shared lock
79        lock.unlock();
80      }
81    }
82
83    // Replaces a string within the ARRAY array with a randomly chosen
84    // string from the POOL array. The thread locks upon accessing the
85    // ARRAY array.
86    public void replaceString() {
87      // Generate a random integer value to determine which ARRAY value
88      // to replace
89      int index = new Random().nextInt(array.length);
90      // Get the initial time for when the thread starts waiting for the lock
91      double startTime = System.nanoTime();
92      // Lock on the shared
93      lock.lock();
94
95      // Get the time for when the thread stops waiting for the lock
96      double endTime = System.nanoTime();
97
98      // Add the wait time in seconds to the replaceWaitTimes ArrayList
99      replaceWaitTimes.add((endTime - startTime));
100      try {
101
102        // Generate a random string from the pool to replace the
103        // ARRAY string with
104        String replaceString = pool[new Random().nextInt(pool.length)];
105
106        // Replace the ARRAY string with the chosen POOL string
107        array[index] = replaceString;
108
109      } finally {
110        // Unlock the shared lock
111        lock.unlock();
112      }
113    }
114
115    // Run upon starting the thread. Performs 50 random operations.
```

```
116  @Override
117  public void run() {
118    // Perform a random operation 50 times
119    for (int i = 0; i < NUM_OPERATIONS; i++) {
120      performTask(new Random().nextInt(MAX_OPERATION_VALUE));
121    }
122    printWaitTimes();
123  }
124
125  // Used to print the average and standard deviation of the search
126  // and replace wait times for the thread
127  public void printWaitTimes() {
128
129    // Average the search and replace wait time ArrayLists
130    double averageSearchTime = average(searchWaitTimes);
131    double averageReplaceTime = average(replaceWaitTimes);
132
133    // Take the standard deviation of the search and wait time
134    // ArrayLists
135    double searchStandardDev
136    = standardDeviation(searchWaitTimes, averageSearchTime);
137    double replaceStandardDev
138    = standardDeviation(replaceWaitTimes, averageReplaceTime);
139
140    // Print out the thread's id and the average and standard deviation
141    // wait times
142    System.out.println("Thread " + (id) + ": Average search wait time: "
143    + (averageSearchTime) + " nanoseconds \n Average replace wait time: "
144    + (averageReplaceTime) + " nanoseconds \n Search Time Standard Deviation: "
145    + (searchStandardDev) + " nanoseconds \n Replace Time Standard Deviation: "
146    + (replaceStandardDev) + " nanoseconds \n");
147  }
148
149  // Used to find the average of the wait times for search and replace
150  private double average(ArrayList<Double> waitTime) {
151    double averageTime = 0;
152
153    // Sum the wait times in the given ArrayList
154    for (double time : waitTime) {
155      averageTime += time;
156    }
157    // Return the sum of the values divided by the ArrayList size
158    return (averageTime / waitTime.size());
159  }
160
161  // Used to find the standard deviation of the wait times for search and
162  // replace
163  private double standardDeviation(ArrayList<Double> waitTime, double avg) {
164
165    double standardDev = 0;
166
167    // Find the summation for each value minus the avverage in the given
168    // waitTime ArrayList
169    for (double time : waitTime) {
170      standardDev += Math.pow(time - avg, 2);
171    }
172
173    // Square the variance to give the standard deviation for the
174    // values of the ArrayList
```

```
175      standardDev = Math.sqrt(standardDev / waitTime.size());
176
177      // Return the standard deviation
178      return standardDev;
179    }
180 }
181
```