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

```
1 /**
2  * Evan McNaughtan and Nicholas Grant
3  * March 21, 2016
4  * CECS 327 (MW 10:00)
5  * Assignment #6
6  * evan4james@yahoo.com
7  * ngrant40@gmail.com
8  *
9  * This program is an exercise in utilizing locks to ensure only one process
10 * enters a critical section at any given time. The critical section in this
11 * program involves accessing a shared memory array. Each thread can either
12 * search for a string within the array or replace a string. Therefore, in order
13 * to ensure that the shared memory array is properly synchronized between all
14 * threads, a lock mechanism is used to ensure mutual exclusion. This
15 * implementation uses coarse-grained synchronization, that is the entire array
16 * is locked when a thread wishes to access it. Wait time data for both
17 * searching and replacing is measured for each thread operation. In addition,
18 * average wait time and standard deviation for both searching and replacing is
19 * calculated and displayed. A criticism of this implementation is that the
20 * coarse-grained synchronization may result in higher wait times on average as
21 * a bottleneck is created. To fix this, fine-grained synchronization or an
22 * optimistic locking mechanism could be used instead.
23 */
24
25 import java.util.concurrent.locks.Lock;
26 import java.util.concurrent.locks.ReentrantLock;
27 import java.util.Random;
28
29 public class ThreadMain {
30
31   public static void main(String[] args) {
32
33     // Number of threads to be run for the main method
34     final int NUM_OF_THREADS = 20;
35
36     final int NUM_OF_POOL_STRINGS = 110;
37
38     final int NUM_OF_ARRAY_STRINGS = 100;
39     // The lock to be shared by each thread for limiting access to the
40     // ARRAY array
41     Lock lock = new ReentrantLock();
42     // Keeps track of the threads created by main method
43     ArrayThread[] threadArray = new ArrayThread[NUM_OF_THREADS];
44     // The randomly generated pool of strings used to populate the ARRAY
45     // and generate strings to search for and replace with
46     String[] POOL = new String[NUM_OF_POOL_STRINGS];
47     // The array of strings used by the threads to search for and replace
48     // strings
49     String[] ARRAY = new String[NUM_OF_ARRAY_STRINGS];
50
51     // Populate the POOL array with randomly generated strings
52     populatePool(POOL);
53
54     // Populate the ARRAY array with strings randomly chosen from the POOL
55     populateArray(ARRAY, POOL);
56
```

```
57    // Populate the thread array with instances of our custom thread class.
58    // The threads take in the lock, ARRAY, and POOL, effectively making them
59    // shared variables
60    populateThreadArray(ARRAY, POOL, threadArray, lock);
61
62    // Start each thread in the thread array
63    startThreads(threadArray);
64  }
65
66  // Used to generate a random string of uppercase letters with length
67  // between 5 and 20
68  public static String generateRandomString() {
69    final int LENGTH_UPPER_BOUND = 16;
70    final int LENGTH_LOWER_BOUND = 5;
71    final int LETTER_UPPER_BOUND = 26;
72
73    // Generates a random integer value between 5 and 20
74    //for the length of the random string
75    int lengthOfString
76          = new Random().nextInt(LENGTH_UPPER_BOUND) + LENGTH_LOWER_BOUND;
77
78    // Initialize an empty string to append to
79    String randomString = new String();
80
81    // Adds lengthOfString randomly generated uppercase letters
82    // to our randomString
83    for (int i = 0; i < lengthOfString; i++) {
84      randomString
85            += (char) (new Random().nextInt(LETTER_UPPER_BOUND) + 'A');
86    }
87
88    // Return the randomString
89    return randomString;
90  }
91
92  // Iterates over the length of the POOL array and
93  // initializes each index with a random string of uppercase letters
94  public static void populatePool(String[] POOL) {
95    for (int i = 0; i < POOL.length; i++) {
96      POOL[i] = generateRandomString();
97    }
98  }
99
100 // Iterates over the length of the ARRAY array and takes a random
101 // string from the POOL array
102 public static void populateArray(String[] ARRAY, String[] POOL) {
103   for (int i = 0; i < ARRAY.length; i++) {
104     ARRAY[i] = POOL[new Random().nextInt(POOL.length)];
105   }
106 }
107
108 // Iterates over the length of the threadArray array and initializes
109 // a new thread in each index
110 public static void populateThreadArray(String[] ARRAY, String[] POOL,
111       ArrayThread[] threadArray, Lock lock) {
112   for (int i = 0; i < threadArray.length; i++) {
113     threadArray[i] = new ArrayThread(i, lock, ARRAY, POOL);
114   }
115 }
```

```
116
117   // Iterates over the threadArray array and starts each thread
118   public static void startThreads(ArrayThread[] threadArray) {
119     for (ArrayThread thread : threadArray) {
120       thread.start();
121     }
122   }
123 }
124
```