Jonathan Reznik

Assignment 4

CS20B

Questions

1. Reading 4.9 Concurrency, Synchronization and Interference
   1. Demo\_06.java - See final page

| (without synchronization) | Expected Result # Units | Unexpected Result # time |
| --- | --- | --- |
| First 100 trials | 97 | 3 |
| Second 100 trials | 99 | 1 |
| Third 100 trials | 99 | 1 |
| Fourth 100 trials | 99 | 1 |

* 1. Demo07.java - See final page

| (with synchronization) | Expected Result # Units | Unexpected Result # time |
| --- | --- | --- |
| First 100 trials | 100 | 0 |
| Second 100 trials | 100 | 0 |
| Third 100 trials | 100 | 0 |

1. Reading for Collections
   1. Efficiency
      1. Sorted list is able to use a binary search algorithm that reduces number of times the comparison is made to logN with both methods.
      2. *ArrayCollection*—the indexed location of the last element present in the collection is already known (numElements -1) and the new item is simply added to the end, which is O(1). *LinkedCollection*—new LLNode attaches to head pointer to the top/front of linked list, so simply O(1). *SortedArrayCollection*—this is O(N) because it inserts elements in middle of array requiring the remaining elements to all be shifted, which is O(N); furthermore, this implementation is unbounded so it can be worse than O(N) when enlarge() method gets used inside of add().
      3. Returning the value of numElements is a complexity O(1) operation.
   2. equals() method is useful for checking contents (when not comparing); its signature is **public boolean equals(Object o)**. And, it inherits from the Object class.
   3. Comparable interface; there is one abstract method only in the interface, which is compareTo(), its signature is **int compareTo(Object o)**. In the example changing to a comparable implementation, by adding ***implements Comparable<K>*** to the class.
   4. Section 5.4, results for the 8 tests with reasoning
      1. c1==c1, True because while it compares references not the values/contents it will be true because these are actually the same reference cells.
      2. c1==c2, False because the references are not the same
      3. c1==c3, False because the references are not the same
      4. c1==c4, False because the references are not the same
      5. c1.equals(c1), True the equals method will evaluate to equal, since 5==5.
      6. c1.equals(c2), True again because 5==5.
      7. c1.equals(c3), False because the circle radius is different for the two circles. (5!=15)
      8. c1.equals(c4) False, can’t compare null to a actual value for radius.
   5. Interfaces inheritance
      1. Methods to implement are RSVP(), and doSecurityCheck()
      2. Methods to be impemented are RSVP(), checkID(), and findPartner()

**SCROLL TO NEXT PAGE FOR THE MODIFIED CODE OF DEMO07 AND DEMO06**

Demo\_06.java

| 1. package ch04.concurrency; 2. import ch04.threads.\*; 3. import ch04.queues.\*; 4. //modified for repeated trials and to report average results 5. public class Demo06 { 6. public static void main(String[] args) throws InterruptedException { 7. int expectedResults = 0; 8. int unexpectedResults = 0; 9. for(int a = 1; a<=100; a++){ 10. try{ 11. int LIMIT = 100; 12. SyncCounter c = new SyncCounter(); 13. QueueInterface<Integer> q; 14. q = new ArrayBoundedQueue<Integer>(LIMIT); 15. for (int i = 1; i <= LIMIT; i++) 16. q.enqueue(i); 17. Runnable r1 = new IncreaseFromArray(c, q); 18. Runnable r2 = new IncreaseFromArray(c, q); 19. Thread t1 = new Thread(r1); 20. Thread t2 = new Thread(r2); 21. t1.start(); t2.start(); 22. t1.join(); t2.join(); 23. if(c.getCount()==5050) 24. expectedResults++; 25. else 26. unexpectedResults++; 27. } catch (InterruptedException e) {; }} 28. System.out.println("Expected: " + (expectedResults) + "\nUnexpected: " + (unexpectedResults)); 29. }} |
| --- |

Demo\_07.java

| package ch04.concurrency;  import ch04.threads.\*;  import ch04.queues.\*;  public class Demo07 {  public static void main(String[] args) throws InterruptedException {  int expectedResults = 0;  int unexpectedResults = 0;  for(int a = 1; a<=100; a++){  try{  int LIMIT = 100;  SyncCounter c = new SyncCounter();  QueueInterface<Integer> q;  q = new SyncArrayBoundedQueue<Integer>(LIMIT);  for (int i = 1; i <= LIMIT; i++)  q.enqueue(i);  Runnable r1 = new IncreaseFromArray(c, q);  Runnable r2 = new IncreaseFromArray(c, q);  Thread t1 = new Thread(r1);  Thread t2 = new Thread(r2);  t1.start(); t2.start();  t1.join(); t2.join();  if(c.getCount()==5050)  expectedResults++;  else  unexpectedResults++;  } catch (InterruptedException e) { ; }}  System.out.println("Expected: " + (expectedResults) + "\nUnexpected: " + (unexpectedResults));  }} |
| --- |