CS 131 Discussion 6

Winter 2015

Announcements

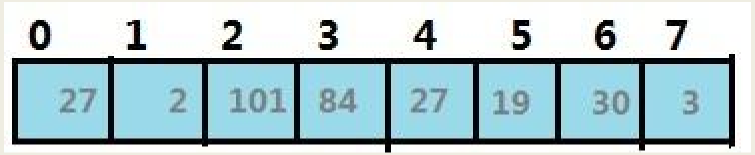
- **Homework 4**

  Due Tues, Feb 17 at 23:55

Homework 4

Background

- Suppose you have an array of N elements, each element a 1-byte number from 0 to maxval (eg. 127), inclusive



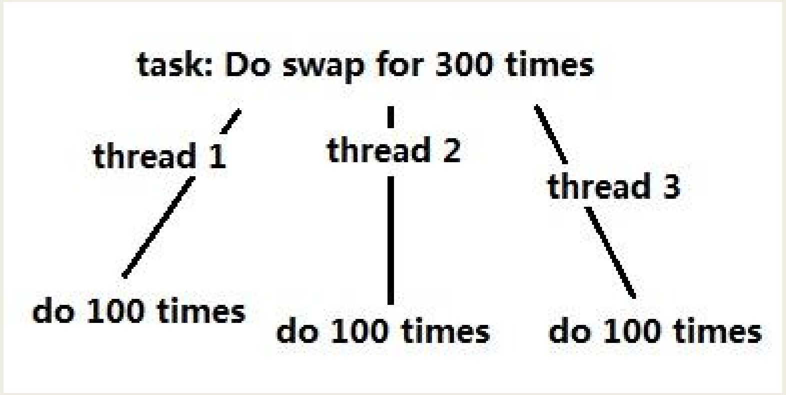
- There is only one operation you can perform on the array

```
boolean swap(int i, int j)
        if array[i] <= 0 or array[j] >= maxval, return false
        array[i] = array[i] - 1
        array[j] = array[j] + 1
        return true
```

Background

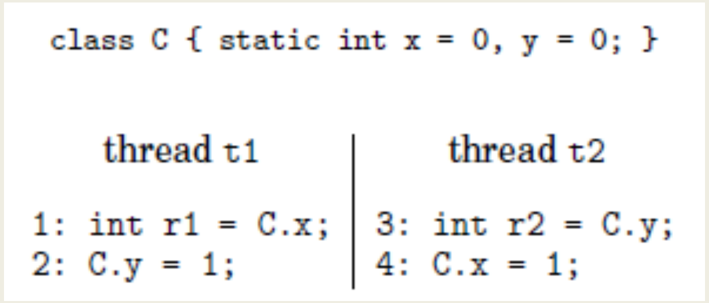
- You want to perform this swap operation many (say 300) times, on random elements of the array.

- So let's split up the work to 3 different threads!
  - Each thread runs "swap" for 300/3 = 100 times



The Problem with Concurrency

Example:



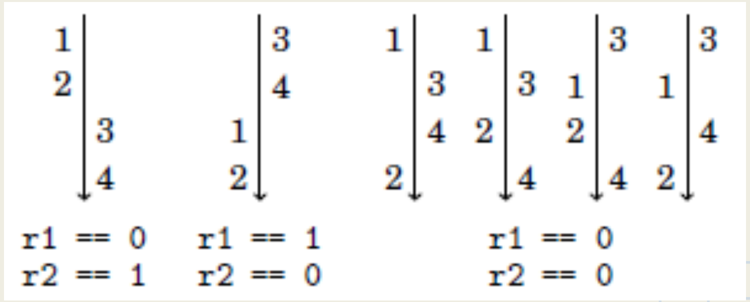
```
class C { static int x = 0, y = 0; }

     thread t1          |      thread t2

1: int r1 = C.x;        | 3: int r2 = C.y;
2: C.y = 1;             | 4: C.x = 1;
```

Sequential Consistency (SC):

- One thread executes at a time
- Every write immediately becomes visible to all threads
- Threads execute their own statements in order



- Even more trouble when SC is broken!
  - What happens when we execute in this order: 2->3->4->1?

Solutions in Java

- adding the "**synchronized**" keyword in the function definition

```
public class SynchronizedCounter {
    private int c = 0;
    public synchronized void increment() { c++; }
    public synchronized void decrement() { c--;  }
    public synchronized int value() { return c; }
}
```

- Let `count` be an instance of `SynchronizedCounter`. If thread1 is using `count.increment()`, thread2 can use neither `count.increment()`, `count.decrement()` nor `count.value()`

Solutions in Java

- Using the **volatile** keyword

note that “static” means the variable is associated with the class, not with the instance.

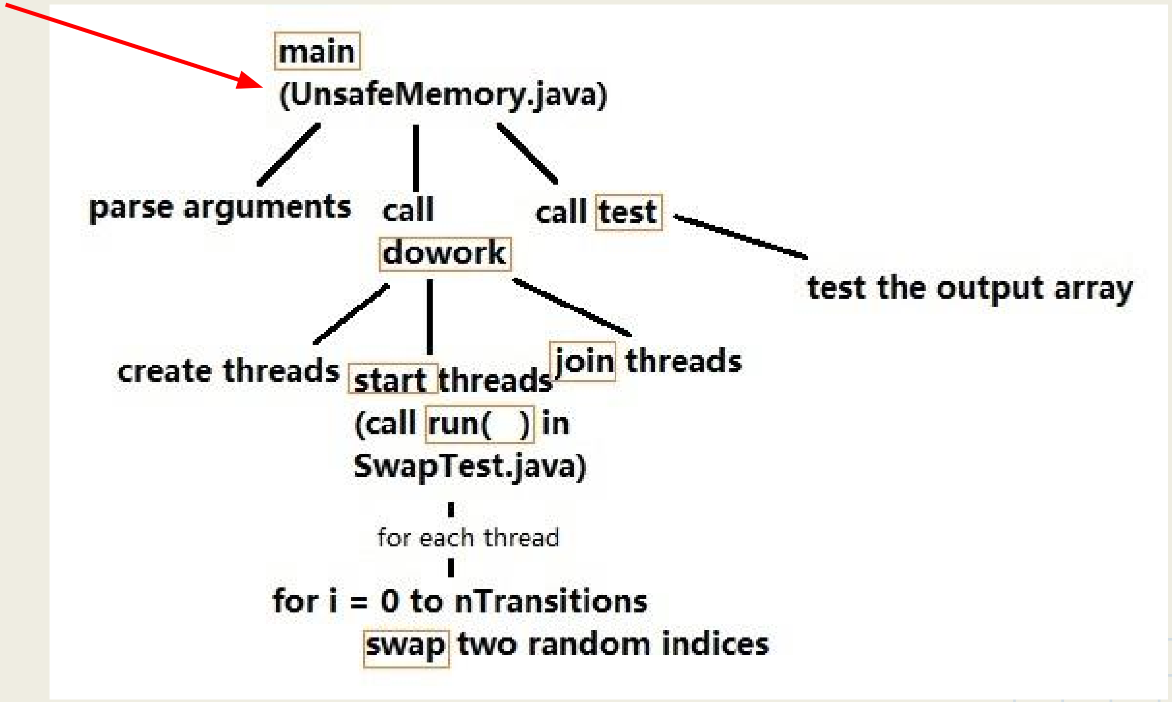
```
public class VolatileExample {
    private static volatile int MY_INT = 0;
    static class ChangeMaker extends Thread {
        public void run() {
                // make some change to MY_INT
        }
    static class ChangeListener extends Thread {
        public void run() {
                // looks for changes to MY_INT (by ChangeMaker)
                // If we don’t make MY_INT volatile, we cannot
                // guerentee change is detected
        }
}
```

Solutions in Java

- Use **`AtomicIntegerArray`** or other classes in the java.util.concurrent.atomic package
- All functions in **`AtomicIntegerArray`** are atomic: they are done as a single "step".
- These functions include **`get(i)`** and **`set(i)`**`, which you will use in the homework`

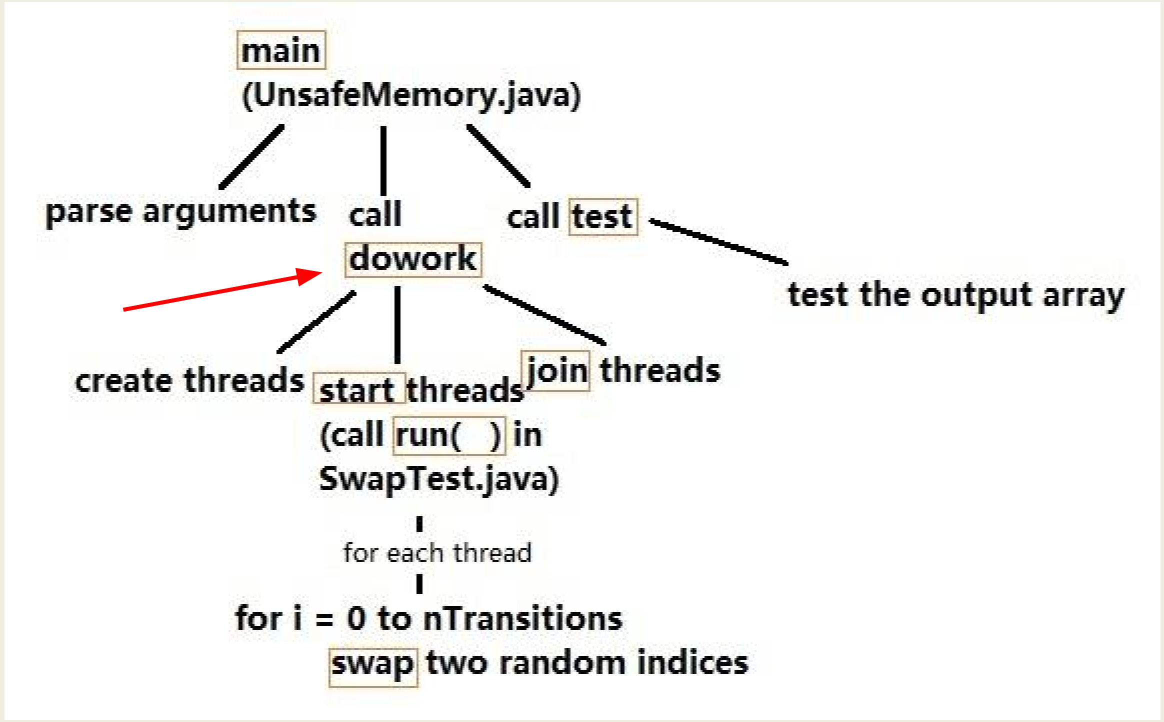
Example Code

Overview


main
(UnsafeMemory.java)
parse arguments
call dowork
call test
test the output array
create threads
start threads
(call run( ) in
SwapTest.java)
join threads
for each thread
for i = 0 to nTransitions
swap two random indices

UnsafeMemory.java

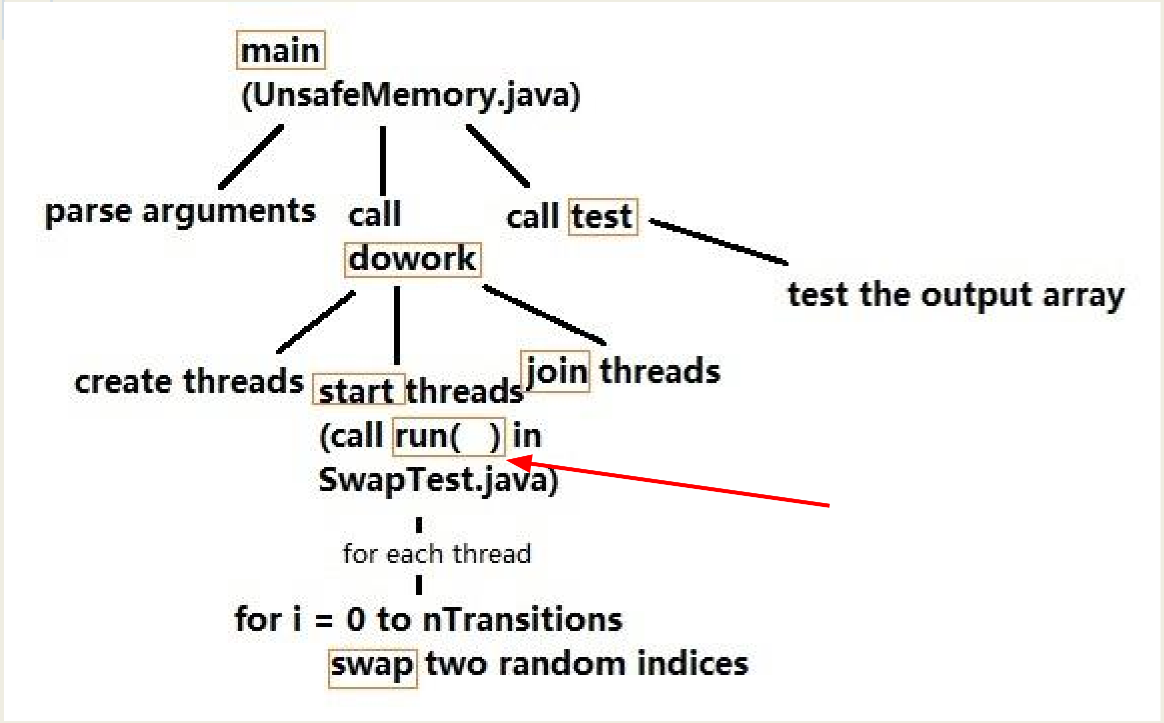
```
class UnsafeMemory {
    public static void main(String args[]) {
        int nThreads = 1rst argument
        int nTransitions = 2nd argument
        byte maxval = 3rd argument
        byte[] value = 4th, 5th, …, args.length argument
        byte[] stateArg = value.clone();
        State s = Null or SynchronisedState
        dowork(nThreads, nTransitions, s);
        test(value, s.current(), maxval);
    }
```


main
(UnsafeMemory.java)
parse arguments
call
dowork
call test
test the output array
create threads
start threads
(call run( ) in
SwapTest.java)
join threads
for each thread
for i = 0 to nTransitions
swap two random indices

UnsafeMemory.java

```
private static void dowork(int nThreads, int nTransitions, State s) {
    Thread[] t = new Thread[nThreads];
    for each thread t[i]
        t[i] = new Thread (new SwapTest (threadTransitions, s));
    long start = System.nanoTime();
    for each thread t[i]
        t[i].start ();
    for each thread t[i]
        t[i].join ();
    long end = System.nanoTime();
    double elapsed_ns = end - start;
}
```

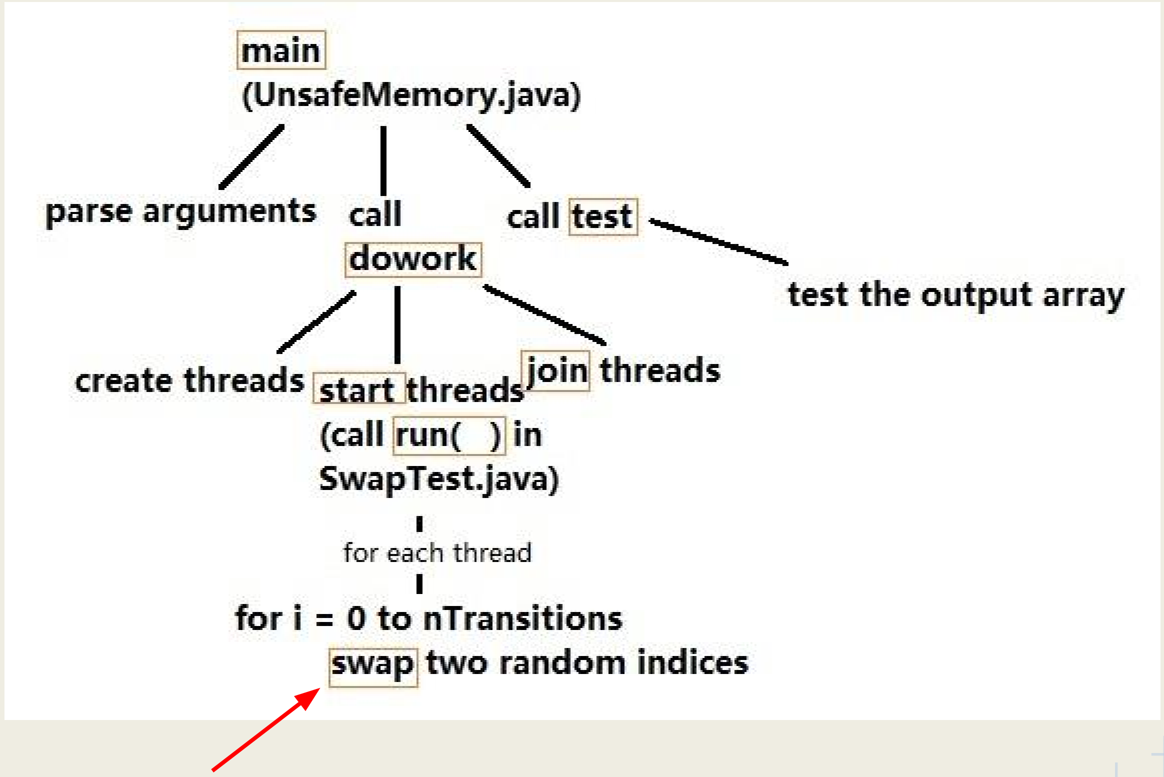
Amount of work done by each thread


main
(UnsafeMemory.java)
parse arguments
call
dowork
call test
test the output array
create threads
start threads
(call run( ) in
SwapTest.java)
join threads
for each thread
for i = 0 to nTransitions
swap two random indices

SwapTest.java

```
class SwapTest implements Runnable {
    private int nTransitions;
    private State state;
...
    public void run() {
         i = 0
         while (i < nTransitions) {
             Get two random values a,b
             if (state.swap(a, b))
                 i++;
         }
    }
}
```

Runnable in an interface. To implement it, our class must have a method called run()

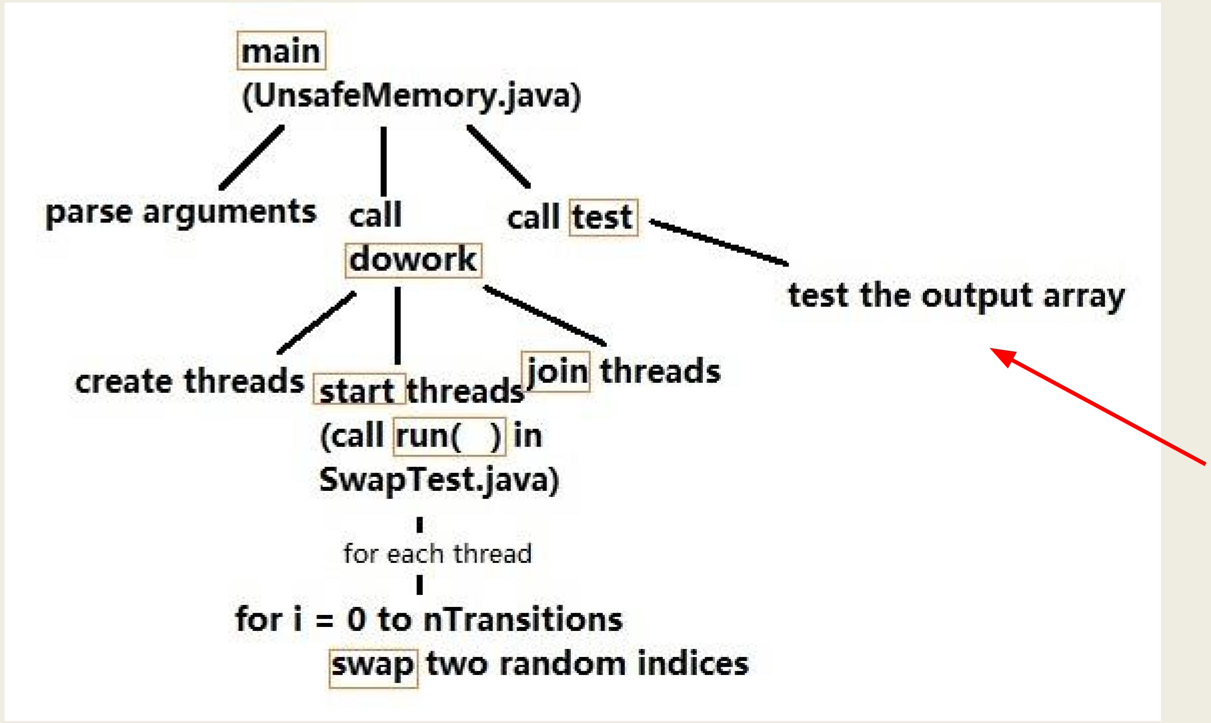

main
(UnsafeMemory.java)
parse arguments
call
dowork
call test
test the output array
create threads
start threads
(call run( ) in
SwapTest.java)
join threads
for each thread
for i = 0 to nTransitions
swap two random indices

State/NullState/SynchronizedState.java

```
interface State {
    int size();
    byte[] current(); // returns the array
    boolean **swap**(int i, int j); // does the swap operation on indices i and j
}

// For an empty array
class NullState implements State {
    private byte[] value;
    NullState(byte[] v, byte maxval) { value = v; }
    public int size() { return value.length; }
    public byte[] current() { return value; }
    public boolean swap(int i, int j) { return true; }
}
```

```
class SynchronizedState implements State {
    private byte[] value;
    private byte maxval;
     ...
    public synchronized boolean swap(int i, int j) {
     if (value[i] <= 0 || value[j] >= maxval) {
         return false;
     }
     value[i]--;
     value[j]++;
     return true;
    }
```


main
(UnsafeMemory.java)
parse arguments
call
dowork
call test
test the output array
create threads
start threads
(call run( ) in
SwapTest.java)
join threads
for each thread
for i = 0 to nTransitions
swap two random indices

UnsafeMemory.java

```
private static void test(byte[] input, byte[] output, byte maxval) {
   Check input length = output length
   Check every element in output list does not exceed maxval or go below 0
   Check sum of elements in output is the same as sum of elements in input
}
```