Concurrency

1. **Atomicity Violation** Give an example of code that uses a ConcurrentHashMap and has a concurrency bug due to an atomicity violation, describe a specific example of an incorrect behavior that can result from the atomicity violation, and show how to fix the code using the methods provided by ConcurrentHashMap. Don’t worry about exact method names; any reasonable guess will be fine.

   **Answer:** Non-atomic code:

   ```java
   /** Make a reservation for what by who. Reservation can only be made if no one else has reserved it. Return whether the reservation was successful. */
   boolean reserve(String what, String who) {
       if (reservations.containsKey(map))
           return false;
       map.put(what,who);
       return true;
   }
   
   This code is intended to update the map only if the key doesn’t exist in the map. But between the call to containsKey and the call to put, another thread may add the key to the map. Thus, if two calls to reserve, trying to reserve the same `what` from different threads overlap, they might both think they successfully reserved it. Fix by changing to:

   ```java
   boolean reserve(String what, String who) {
       return map.putIfAbsent(key, value) == null;
   }
   ```

2. **AtomicInteger** Write a method that atomically doubles the value of an AtomicInteger, and returns the doubled value.

   **Answer:**

   ```java
   public static int doubleAndGet(AtomicInteger a) {
       while (true) {
           int oldValue = a.get();
           int newValue = 2*oldValue;
           if (a.compareAndSet(oldValue,newValue))
               return newValue;
       }
   ```
3. **Fair locks** Several concurrent abstractions, such as locks, provide fair and unfair variants. For fair variants, requests are granted in a first-come, first-served order. With unfair variants, a request may be granted to one thread even if another thread is blocked waiting for a request to be granted. Assume the costs to perform the two variants are identical (e.g., the cost to acquire a fair or unfair lock are identical). It what ways might using unfair variants give better performance?

**Answer:** With fair variants, you always need to perform a context switch if another thread is waiting for the lock. When you perform a context switch, you need to wait for the OS to schedule the other thread, and there are costs associated with changing the working set of the caches. By performing fewer context switches, you can get improved throughput, even though individual requests may be delayed longer.

4. Multithreaded test case: Write a multithreaded test case for a CountDownLatch involving 3 threads. You want to show that if you initialize a CountDownLatch with a count of 2, and two threads await() on it, they both both blocked. If the third thread then calls countDown once, the threads are still blocked. If the third threads then calls countDown() a second time, both waiting threads are unblocked.

```java
static class CountDownLatchTestCase extends MultithreadedTestCase {
    CountDownLatch latch;
    @Override
    public void initialize() {
        latch = new CountDownLatch(2);
    }

    public void thread0() throws InterruptedException {
        latch.await();
        assertEquals(2, getTick());
    }

    public void thread1() throws InterruptedException {
        latch.await();
        assertEquals(2, getTick());
    }

    public void thread3() throws InterruptedException {
        waitForTick(1);
        latch.countDown();
        assertEquals(1, getTick()); // optional
        waitForTick(2);
        latch.countDown();
        assertEquals(2, getTick()); // optional
    }
}
```

**Security**

5. Concisely describe the basic mistake that leads to web applications being vulnerable to Cross site scripting (XSS).

**Answer:** Using untrusted, unchecked input (such as from an HTTP request parameter) verbatim in an HTML response.

6. What is the first line of defense against Cross site request forgery? Perhaps not a perfect solution, but something easily to add to a web site that will at least make it significantly more difficult to execute a CSRF attack, if not block it all together.

**Answer:** You need to do two things:

- Ensure that all sensitive requests, any requests that may change state, are done with POST requests rather than GET requests.
- Verify the referer header for all POST requests.
Decorators, I/O

7. SlowOutputStream Decorator: You need to write a class that is a SlowOutputStreamDecorator. A SlowOutputStream is something you might, for example, want to use in a unit test or performance test where you wanted to simulate how something would perform if it was writing to a network connection that could only handle 100,000 bytes/second. A SlowOutputStream decorates an OutputStream, and when constructed two additional arguments that give the delay per byte written, specified by millisecondsDelay and nanosecondDelay. You can sleep for the that much time using Thread.sleep(long millis, int nanos). Note: in practice, the sleep method isn't particularly accurate, particularly at sleeping for sub-millisecond intervals. But we'll ignore that for this question.

public class SlowOutputStream extends OutputStream {
    final OutputStream o;
    final int nanosPerByte;
    final long millisPerByte;

    public SlowOutputStream(OutputStream o, long millisPerByte, int nanosPerByte) {
        this.o = o;
        this.nanosPerByte = nanosPerByte;
        this.millisPerByte = millisPerByte;
    }

    @Override
    public void write(int b) throws IOException {
        Thread.sleep(millisPerByte, nanosPerByte);
        // ignoring InterruptedException
        o.write(b);
    }

    @Override
    public void close() throws IOException {
        o.close();
    }

    @Override
    public void flush() throws IOException {
        o.flush();
    }
}
MapReduce and distributed computing

8. Why do all of the Mappers need to complete before we execute any calls to the

   \texttt{Reducer.reduce((InputKey key, Iterable<InputValue> values, Context context))}

   method?

   \textbf{Answer:} For any InputKey to a reduce call, we must ensure that we have \textit{all} of the InputValues for
   that key before we invoke the reduce method. Since any mapper might produce a value for a particular
   InputKey, all Map tasks must be completed before we make any calls to a reduce method. If there are
   some workers still working on redundant computations that will be discarded, it is \textit{OK} to proceed.

   Technically, you could imagine speculating that some partitions are complete, and invalidating those
   results and redoing them if you produced an InputKey, InputValue pair after you had already reduced
   that InputKey. But doing so is is unlikely to be useful.

9. In a real Hadoop or MapReduce implementation, the workers send back progress information to the
   Master (e.g., I’m now 65% done.... I’m now 66% done...). Suggest how this information might be used.

   \textbf{Answer:} When all tasks have been scheduled, but not completed, the master speculative sends out
   redundant requests to perform tasks. With progress information, we can avoid sending out requests
   for tasks that some other node is is almost finished with. We might even be able to track the speed of
   each node, and predict for each node the time remaining until it completes its current task. With that
   information, we could send out for redundant execution the task whose estimated completion time is
   furthest away.

   There are several levels of additional refinement you could use for this.

10. Do Map tasks running on workers send all of their output keys and values to the Master node? Why
    or why not?

    \textbf{Answer:} No. The map tasks generate far too much data to funnel it all through the master node. The
        network bandwidth to the master would be a huge bottleneck. Instead, map tasks write their output
        to files, and only send information to the master about their progress and completion of the task.

11. MapReduce is inspired by functional programming, and the Map and Reduce are supposed to be
    “functional” or “not have side effects”, such as changing static fields or writing output other than via
    Context. Why? What can go wrong if mappers and reducers have side effects?

    \textbf{Answer:} Because Map and Reduce are functional, we know that if we ignore the output of a Map or
        Reduce task, it doesn’t effect the computation. Thus, we don’t have to worry about any side effects
        from aborted, failed, or redundant tasks. Also (although less directly) means it shouldn’t mean how
        values get spread/grouped as they are sent to various tasks and workers.