You will need to download 3 files:

* SynchronizedThreads.java
* Deadlock.java
* ZigZagThreads.java

**Preventing Interference Through Synchronization**

The SynchronizedThreads class demonstrates a simple program in which multiple threads (Incrementors) concurrently access a single shared object (a Counter). Review this code and make sure you understand it.

Run the main program see if the threads interfere with each other. Play around with the parameters. For example, if numThreads is 1, there should be no interference. Similarly, if numAdds is 1 or a small number, you might not experience any interference.

**Task 1** Prevent interference (aka race conditions) by modifying the code. You may wish to review the section of the Java tutorial on synchronization, especially the part on [synchronization idioms](http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html).

There are at least *two ways* to prevent inference: one way modifies the Counter class; the other way leaves Counter unchanged and modifies the Incrementor class. Implement both ways. For one that modifies the Counter class, leave the original file name. For the one that modifies the Incrementor rename the class and file as SynchronizedThreads2.

**Answer these questions:**

1. Explain how you modified the Counter class.

I made both methods synchronized by adding the synchronized keyword to the method signature.

2. Explain how you modified the Incrementor class.

I used the synchronized key word to synchronize on the counter object, locking

**Deadlock**

Check out the Deadlock example. This is copied directly from the corresponding section on [deadlock in the Java tutorial](http://docs.oracle.com/javase/tutorial/essential/concurrency/deadlock.html).

**Task 2**. Answer the questions below to explain the deadlock that happens in Deadlock in terms of locks and threads waiting on locks. The two threads in this case are alphonse and gaston. Your explanations can be short (2-3 sentences) but must be precise!

1. What objects are being locked?

The objects that are being locked are the Friend Objects.

2. Who has what lock?

They have their own locks.

3. How does deadlock occur?

For one to bow back the bower needs to obtain their lock, but it was obtained on their call to bow.

4. How would you modify the code to prevent deadlock from occurring? Try to implement these changes.

You obtain the lock on the bower and the bowee at once.

5. Examine the code from Task1 (SynchronizedThreads.java). Can deadlock occur in that code? Explain why or why not.

No, because there aren’t multiple resources being synchronized.

**Zig Zag Lock Manager**

Check out the ZigZagThreads class. The main method of this program spawns a bunch of Ziggers and then a bunch of Zaggers. Ziggers print this pattern

//////////

and Zaggers print this pattern

\\\\\\\\\\

If you run the code, you will probably see a bunch of zigs followed by a bunch of zags. This is not what we want. Instead, we want this pattern:

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

//////////

\\\\\\\\\\

Notice that before printing, both Ziggers and Zaggers must acquire a lock. Your task is to implement a lock manager that forces them to alternate, producing the desired pattern.

**Task 3**. Implement the lock manager in ZigZagThreads to achieve the desired pattern. Hint: the lock manager needs to keep track of two things. The first thing is whether someone has the lock (i.e., someone is busy printing). The second thing is whose turn it is (a Zigger or Zagger).