CS193k, Stanford
Spring, 99-00

Handout #8
Nick Parlante

HW2a — Magic

This is part (a) of the Thread homework. This part of the homework just hits the basic issues of threads and synchronized blocks. The code is a bit contrived, but it gets the point across. All of HW2 will be due Thu May 12th.

Magic Thread

Consider the following code which is available for you in the file MagicThread.java in the class directory. This code has several threading problems which it will be your pleasure to fix. Here's the code...

```
// MagicThread.java
/*
 The MagicStrings class stores some magic strings
 which the client can access one by one. The magic
 strings are a big secret, so the client can only
 get them one at a time.

 Each instance of MagicThread retrieves all the strings
 in order and concats them together along with the name of a
 staffer to make a little spell like this:
 "fibbity fabbity foo : Jason".

 The magic words have to all be there and be in the right
 order, or the spell doesn't work.

 Each magic thread concats its spell onto the "answer"
 static when it is done.
*/


/*
 Store magic strings -- give them
 out to the client one at a time.
*/
class MagicStrings {
   private String[] strings;
   private int index;

   public MagicStrings() {
      strings = new String[] {"fibbity", "fabbity", "foo"};
      index = 0;
   }

   /*
    Return the next magic string, or null
    if there are no more.
   */
   public String nextString() {
      if (index<strings.length) {
         index++;
         return(strings[index-1]);
      }
      else {
         return(null);
      }
   }

   /*
    Reset to the start so that the
    next call to nextString() will return
    the first magic string.
   */
   public void reset() {
      index = 0;
   }
}


/*
 Takes a pointer to a magic object.
 Gets the strings out of it and concats
 them together.
 When done, puts the whole thing into the
 "answer" static.
*/
class MagicThread extends Thread {
   private MagicStrings magic;
   private String name;
   private static StringBuffer answer;


   public MagicThread(MagicStrings magic, String name) {
      this.magic = magic;
      this.name = name;
   }


   public void run() {
      String result = new String();
      String next;

      // see the magic strings in order
      magic.reset();
      while ((next = magic.nextString()) != null) {
         // concat them together
         result = result + next + " ";
      }
      answer.append(result + ": " + name + "\n");
   }


   /*
    Create one magic object and three
    threads to create a spell for each staffer.
    The output should look like...
    fibbity fabbity foo : Saurabh
    fibbity fabbity foo : Jason
    fibbity fabbity foo : Nick

    although the order of the rows may be mixed.
   */
   public static void main(String[] args) {
      MagicStrings magic = new MagicStrings();

      Thread j = new MagicThread(magic, "Jason");
      j.start();
      Thread s = new MagicThread(magic, "Saurabh");
      s.start();
      Thread n = new MagicThread(magic, "Nick");
      n.start();

      answer = new StringBuffer();

      // Wait for the three to finish
      try {
         j.join();
         s.join();
         n.join();
      }
      catch (InterruptedException ignored) {}

      System.out.println(answer);
   }
}
```

The challenge is to fix the various problems with the code. Your changes will be to the MagicThread class. Your solution should continue to use a single magic object.

Part i — Null Pointer

There's a threading problem in the code that leads to a NullPointerException in rare cases. This bug can be fixed by moving one line. Please move the line and append a "// part i" comment to it. Afterwards, the code should still have mutex problems, but at least it won't crash.

Part ii — Mutex Violation

There's a mutex violation that causes the spells to be wrong sometimes. Sometimes the problem will exhibit itself and sometimes it won't. Add a single Thread.yield(); call so that the mutex error almost always exhibits itself, and append a "// part ii" comment to the line.

Part iii — Mutex Repair

Now add in the minimum amount of synchronization in the vicinity of MagicThread.run() so the code is correct, even with the Part ii yield() still operating.

Part iv — Performance

Finally, fix up the code in MagicThread.run() to minimize the amount of work done while holding the lock. Also, avoid using String where StringBuffer is better. Avoid unnecessary locking. You may assume there are at most 100 strings in magic. Surprisingly aggressive re-writing is possible in this part.