Problem Set 12 Solution

Multithreading

---

1. Use threads to implement a stop watch that displays, *once every five seconds*, the minutes and seconds that have passed since it was started. The display should be in the form `mm:ss` for minutes and seconds. When the clock reaches 15 minutes, it should wrap back and start at 0 minutes and 0 seconds. The user should be able to stop the watch at any time. Write the complete code for the application. (Not the most accurate stop watch, but the model is useful for animations in which slight inaccuracies in time would not be detrimental.)

   SOLUTION

   StopWatch.java

---

2. Suppose you use a search engine to search for a word or phrase that results in a match with a large number (hundreds) of web pages. However, the browser will only display a list of n (some variable parameter) page links in one screenful. Implement a multi-threaded program with one thread for the browser display, and another for the search engine, and have the search engine deal out hits in batches of the max size the browser can display in one screenful. You may assume that a method called fetch has already been written in the search engine to fetch the next batch of hits, returned as a list of URL strings.

   SOLUTION

```
public class Searcher implements Runnable {
      private Request req;   // search request
      private Buffer buf;    // to store hits for display
      public Searcher(Request req, Buffer buf) {
         this.req = req;
         this.buf = buf;
         new Thread(this).start();
      }
      public void run() {
         while (true) {
            while ((List hits = fetchNext(req)) != null)  { // more
      results from fetch
                buf.put(hits);
         }
      }
   }

   public class Displayer implements Runnable {
      private Buffer buf;
      public Displayer(Buffer buf) {
         this.buf = buf;
         new Thread(this).start();
      }
      public void run() {
         while (true) {
            display(buf.get());
         }
      }
   }

   public class Buffer {
      List hits;
```

```
    boolean available = false;
    public synchronized void put(List hits) {
       while (available) {
          try {
              wait();
          }
          catch (Exception e) { }
       }
       available = true;
       this.hits = hits;
       notifyAll();
    }

    public synchronized List get() {
       while (!available) {
          try {
              wait();
          }
       catch (Exception e) { }
    }
    available = false;
    notifyAll();
    return hits;
 }
```

---

3. In class, you saw the `Iterator` design pattern applied to iterate over a linked list. That implementation did not support the `remove` operation. Now suppose you were to update the iterator by supporting `remove`, which deletes the current node from the target linked list.

   Explain why this new iterator is not thread-safe. (Thread-safe means multiple threads can be run through all methods of the class via a shared instance of the class, without unsafe interference.) How would you change the implementation to make it safe? Describe your solution, no code needed.

   SOLUTION

   The iterator wouldn't be thread safe. Since access to remove isn't synchronized, two threads could try to remove the same node at once, and a particular interleaving of operations could result in one thread trying to access a node that's not there any more, and running into a null pointer exception. So the method would have to be synchronized. The other methods (`next`, `hasNext`) should not be synchronized since they don't update the list structure.