Section 8: Concurrency

0. You are designing a new social-networking site to take over the world. To handle all the volume you expect, you want to support multiple threads with a fine-grained locking strategy in which each user's profile is protected with a different lock. At the core of your system is this simple class definition:

```
class UserProfile {
   static int id_counter;
   int id; // unique for each account
   int[] friends = new int[9999]; // horrible style
   int numFriends;
   Image[] embarrassingPhotos = new Image[9999];
   UserProfile() { // constructor for new profiles
     id = id_counter++;
     numFriends = 0;
   }
   synchronized void makeFriends(UserProfile newFriend) {
      synchronized(newFriend) {
        if(numFriends == friends.length
           || newFriend.numFriends == newFriend.friends.length)
          throw new TooManyFriendsException();
        friends[numFriends++] = newFriend.id;
        newFriend.friends[newFriend.numFriends++] = id;
      }
   }
   synchronized void removeFriend(UserProfile frenemy) {
      ...
   }
}
```

(a) The constructor has a concurrency error. What is it and how would you fix it? A short English anwser is enough – no code or details required.

(b) The `makeFriends` method has a concurrency error. What is it and how would you fix it? A short English anwser is enough – no code or details required.

1) **Concurrency:** The `BubbleTea` class manages a bubble tea order assembled by multiple workers. Multiple threads could be accessing the same `BubbleTea` object. Assume the `Stack` objects ARE THREAD-SAFE, have enough space, and operations on them will not throw an exception.

```
public class BubbleTea {
      private Stack<String> drink = new Stack<String>();
      private Stack<String> toppings = new Stack<String>();
      private final int maxDrinkAmount = 8;



      // Checks if drink has capacity
      public boolean hasCapacity() {

            return drink.size() < maxDrinkAmount;

      }

      // Adds liquid to drink
      public void addLiquid(String liquid) {

            if (hasCapacity()) {

                  if (liquid.equals("Milk")) {

                        while (hasCapacity()) {

                              drink.push("Milk");
                        }

                  } else {

                        drink.push(liquid);

                  }

            }

      }

      // Adds newTop to list of toppings to add to drink
      public void addTopping(String newTop) {

            if (newTop.equals("Boba") || newTop.equals("Tapioca")) {

                  toppings.push("Bubbles");

            } else {

                  toppings.push(newTop);

            }

      }
}
```

1) (Continued)

a) Does the `BubbleTea` class above have (circle all that apply):

a race condition, potential for deadlock, a data race, none of these

If there are any problems, give an example of when those problems could occur. Be specific!

b) Suppose we made the `addTopping` method **`synchronized`**, and changed nothing else in the code. Does this modified `BubbleTea` class above have (circle all that apply):

a race condition, potential for deadlock, a data race, none of these

If there are any FIXED problems, describe why they are FIXED. If there are any NEW problems, give an example of when those problems could occur. Be specific!

2) **Concurrency:** The `PhoneMonitor` class tries to help manage how much you use your cell phone each day. Multiple threads can access the same `PhoneMonitor` object. Remember that synchronized gives you reentrancy.

```
 1
 2 public class PhoneMonitor {
 3    private int numMinutes = 0;
 4    private int numAccesses = 0;
 5    private int maxMinutes = 200;
 6    private int maxAccesses = 10;
 7    private boolean phoneOn = true;
 8    private Object accessesLock = new Object();
 9    private Object minutesLock = new Object();
10
11    public void accessPhone(int minutes) {
12
13       if (phoneOn) {
14
15          synchronized (accessesLock) {
16
17             synchronized (minutesLock) {
18
19                numAccesses++;
20                numMinutes += minutes;
21                checkLimits();
22             }
23          }
24       }
25    }
26
27    private void checkLimits() {
28
29       synchronized (minutesLock) {
30
31          synchronized (accessesLock) {
32
33             if ( (numAccesses >= maxAccesses) ||
34                  (numMinutes >= maxMinutes) ) {
35                     phoneOn = false;
36             }
37          }
38       }
39    }
40 }
```

a) Does the `PhoneMonitor` class as shown above have (circle **all** that apply):

a race condition, potential for deadlock, a data race, none of these

Justify your answer. Refer to line numbers in your explanation. Be specific!

2) (**Continued**)

b) Suppose we made the `checkLimits` method **`public`**, and changed nothing else in the code. Does this modified `PhoneMonitor` class have (circle **all** that apply):

a race condition, potential for deadlock, a data race, none of these

If there are any FIXED problems, describe why they are FIXED. If there are any NEW problems, give an example of when those problems could occur. Refer to line numbers in your explanation. Be specific!