50.003: Quiz 4 (15 Minutes)

**Name:**

**ID:**

Q1: Consider the *ThreadSafeCounter* example below:

public class ThreadSafeCounter {

private int count = 0; // The value of the counter.

public synchronized void increment() { count = count + 1; }

public synchronized int getValue() { return count; }

}

The increment() method is synchronized so that the caller of the method can complete the three steps of the operation "Get value of count," "Add 1," "Store new value in count" without being interrupted by another thread. But method getValue() consists of a single, simple step. Why is getValue() synchronized?

Answer: The reason is to make sure the “getValue” method returns the updated value. There are two possible reasons why the value may not be updated without “synchronized”. First, while reading, another thread might be in the process of writing. Second, the updated value might be in some other cache which the processor running the “getValue” has no access to.

Q2. Read the following code:

public class BookWrapper {

   private final Book book;

   BookWrapper(Book book) {

     this.book = book; //do not worry about the escape here.

   }

   public void issue(int days) {

     book.issue(days);

   }

   public Calendar getDueDate() {

     return book.getDueDate();

   }

   public void renew() {

     synchronized(book) {

       if (book.getDueDate().before(Calendar.getInstance())) {

         throw new IllegalStateException("Book overdue");

       } else {

         book.issue(14); // Issue book for 14 days

       }

     }

   }

}

Assume that class Book uses a private lock for thread-safety. Fix the above class so that it becomes thread-safe. You can mark your change on the code above or write the modified code below.

public final class BookWrapper {

  private final Book book;

  private final Object lock = new Object();

  BookWrapper(Book book) {

    this.book = book;

  }

  public void issue(int days) {

    synchronized(lock) {

      book.issue(days);

    }

  }

  public Calendar getDueDate() {

    synchronized(lock) {

      return book.getDueDate();

    }

  }

  public void renew() {

    synchronized(lock) {

      if (book.getDueDate().before(Calendar.getInstance())) {

        throw new IllegalStateException("Book overdue");

      } else {

        book.issue(14); // Issue book for 14 days

      }

    }

  }

}

Q3. You have two threads (T0 and T1) and two locks (X and Y). Which of the following situations can lead to deadlock? If deadlock can occur, circle the method call in each thread where the thread would stop in the event of deadlock. If deadlock is impossible, circle “no deadlock.”

a)

|  |  |
| --- | --- |
| T0:  X.acquire();  Y.acquire(); Y.release(); X.release(); | T1: X.acquire();  Y.acquire(); X.release(); Y.release(); |

**no deadlock**

b)

|  |  |
| --- | --- |
| T0: (same as T0 above) **X.acquire();** Y.acquire(); Y.release(); X.release(); | T1: **Y.acquire();** X.acquire(); X.release(); Y.release(); |

no deadlock

c)

|  |  |
| --- | --- |
| T0: (same as T0 above) X.acquire();  Y.acquire(); Y.release(); X.release(); | T1: Y.acquire();  Y.release(); X.acquire(); X.release(); |

**no deadlock**