Solutions to Quiz 2 (November 22, 2013)

Problem 1 (Short Answer) (21 points). Choose all correct answers for the following questions.

(a) The Java code `class MyWindow extends JFrame {...}` demonstrates the concept(s) of:

A. inheritance  
B. composition  
C. delegation  
D. forwarding  
E. locking  
F. subclassing

**Solution.** A, F. Uses *inheritance* to implement *subclassing*.  

(b) The Java code `synchronized (this) { this.accountBalance += depositAmount; }` demonstrates the concept(s) of:

A. shared memory  
B. message passing  
C. locking  
D. thread safety  
E. listeners

**Solution.** A, C, D. Uses *locking* to ensure *thread safety* of *shared memory*.  

(c) The Java code `assert(checkRep())` demonstrates the concept(s) of:

A. synchronization  
B. abstract data type  
C. rep invariant  
D. behavioral equality  
E. failing fast

**Solution.** B, C, E. *Fails fast* by ensuring that the *rep invariant* of an *ADT* is preserved.  

(d) The Java code `private List<Person> committee` demonstrates the concept(s) of:

A. representation independence  
B. information hiding  
C. static typing
D. view tree
E. abstract data type

Solution. A, B, C, E. private fields use information hiding to ensure rep independence of an ADT. Also uses information hiding to ensure rep independence from the concrete variant of the List ADT. Static typing requires us to declare the type of committee, and allows us to restrict the type of its elements.

Problem 2 (Map/Filter/Reduce) (28 points).
You are given a Python interpreter that has a broken len() function – it always throws exceptions, no matter what you pass it. But you’ve been able to verify that map(), reduce() and filter() functions are working. Recall that the method signatures for these functions are as follows:

- map(function, list)
- reduce(function, list, initialValue)
- filter(predicate, list)

(a) Use map, filter, and reduce to implement a working version of len(), the function that counts the number of elements in a list. Your solution must define the functions m, f, and r and the value init so that the len() implementation below is correct.

```python
def len(list):
    def f(__________________):
        return ___________________  
    def m(__________________):
        return ___________________
    def r(__________________):
        return ___________________
    init = ___________________
    return reduce(r, map(m, filter(f, list)), init)
```

Solution.
```python
def f(x):
    return True

def m(x):
    return 1

def r(x, y):
    return x+y
init = 0
```
(b) Also broken is \texttt{count(list, e)}, a function that counts the number of times that \texttt{e} occurs in the list.

Write a working version of \texttt{count}.

\begin{verbatim}
def count(list, e):
    def f(__________________):
        return ________________
    def m(__________________):
        return ________________
    def r(__________________):
        return ________________
    init = _________________
    return reduce(r, map(m, filter(f, list)), init)
\end{verbatim}

Solution.

\begin{verbatim}
def f(x):
    return x == e

def m(x):
    return 1

def r(x,y):
    return x+y

init = 0
\end{verbatim}

Problem 3 (Deadlock) (16 points).

In recitation, we looked at the Wizard class. A portion of its code is shown below. Recall that \texttt{Set.add()} returns true if and only if the element wasn’t found in the set and was successfully added. Similarly, \texttt{Set.remove()} returns true if and only if the element was found and then was successfully removed.

\begin{verbatim}
public class Wizard {
    private final String name;
    private final Set<Wizard> friends;
    // Rep invariant:
    // name, friends != null
    // friend links are bidirectional: for all f in friends, f.friends contains this
    ...
    public boolean isFriendsWith(Wizard that) {
        return this.friends.contains(that);
    }
\end{verbatim}
public synchronized void friend(Wizard that) {
    if (this.friends.add(that)) {
        that.friend(this);
    }
}

public synchronized void defriend(Wizard that) {
    if (this.friends.remove(that)) {
        that.defriend(this);
    }
}

(a) Which of the following may cause deadlock? (Assume the rep invariant is true before each of these examples starts.) Circle all correct answers.

A. Thread A calls snape.friend(harry), while Thread B calls harry.friend(snape)
B. Thread A calls snape.friend(harry), while Thread B calls snape.friend(harry)
C. Thread A calls snape.defriend(harry) and then harry.friend(snape), while Thread B calls snape.friend(harry).
D. Thread A calls snape.friend(harry), while Thread B calls harry.defriend(snape)

Solution. A and C. Option B acquires the locks in the same order. Option D can’t cause deadlock assuming the rep invariant holds: if both operations have acquired locks on this, only one of them will try to lock the other wizard.

(b) Which of the following would fix this deadlock problem? Circle all correct answers.

A. Lock on Wizard.class rather than the Wizard instance.
B. Make the Set a synchronizedSet, while keeping the synchronized keywords in the friend and unfriend methods.
C. Assign a unique integer id to every Wizard, and when friending or unfriending two wizards, always acquire the lock of the wizard with the smaller id first.
D. Combine the friend and defriend methods into a single method:
   public synchronized void friendOrDefriend(Wizard that, boolean friend)
   If friend==true, this method would do the friend operation, otherwise it would do the defriend operation.

Solution. A and C. In both B and D, we still try to acquire the locks on snape and harry in different orders.

Problem 4 (Interleaving) (20 points).
Consider the following class:

public class Magic {
    private static int x = 1;

    public static int magicMethod() {
    }
Thread t1 = new Thread(new Runnable() {
    public void run() {
        x *= 2;
        x *= 3;
    }
});

Thread t2 = new Thread(new Runnable() {
    public void run() {
        x *= 5;
        x *= 11;
    }
});
t1.start();
t2.start();
t1.join(); // join() means wait until the thread finishes its run() method
        t2.join();

        return x;
    }
}

Assume that all threads are running on the same processor, interleaving execution with each other. Which of the following are possible return values of magicMethod? Circle all correct answers.

• 330
• 30
• 6
• 2
• 22

Solution. 330, 30, 6, 22. Just 2 is not possible because magicMethod waits for both threads to complete, and either *3 or *11 has to win at the end. ■

Problem 5 (Thread Safety) (15 points).
Suppose you are reviewing this code for thread safety:
public class C {
    public static final String[] x = new String[] { "abc" };  
    private final int y = 0;
    ...
    public synchronized double f() {
        double z = 0;
        ...
    }
    ...
}
Which of the following statements would be true and appropriate for an argument either in favor of or against the thread safety of this code? Circle all correct answers.

(a) For x:

A. x is thread-confined.
B. x is immutable.
C. x is protected by a lock.
D. x is global.
E. x can be involved in a race condition.

Solution. D, E. While x is an immutable reference, and String values are immutable, the array is mutable, so we cannot argue for thread safety by immutability.

(b) For y:

A. y is thread-confined.
B. y is immutable.
C. y is protected by a lock.
D. y is global.
E. y can be involved in a race condition.

Solution. B.

(e) For z:

A. z is thread-confined.
B. z is immutable.
C. z is protected by a lock.
D. z is global.
E. z can be involved in a race condition.

Solution. A. Since z is a non-final primitive, even threads spun in f cannot reference the value of z. Since f is synchronized, it is true that all access to z occurs under a lock on this, but that is not an appropriate argument for its thread safety.