Massachusetts Institute of Technology
6.005: Elements of Software Construction
Fall 2011
Quiz 2
November 21, 2011

Name: SOLUTIONS

Athena* User Name:

Instructions

This quiz is 50 minutes long. It contains 7 pages (including this page) for a total of 100 points. The quiz is closed-book, closed-notes.

Please check your copy to make sure that it is complete before you start. Turn in all pages, together, when you finish. Write your name on the top of every page. Please write neatly. No credit will be given if we cannot read what you write. Good luck!

Question Name	Page	Maximum Points	Points Given
Design Patterns	2	20	
Interpreter/Visitor	3	16	
Map/Filter/Reduce	4	12	
Concurrency	5	16	
Deadlock	6	16	
Thread Safety	7	20	

*Athena is MIT's UNIX-based computing environment. OCW does not provide access to it.

Name: _____

Design Patterns [20 pts]

For each of the following statements, name the design pattern that it **best** describes, from the list below.

Interpreter
Visitor
Event Listener
Map/Filter/Reduce
Client/Server
Model/View/Controller
Composite

You may use a design pattern more than once in your answers. If you're torn between two best answers, you can give both, but in that case you should justify both answers.

(a) This design pattern produces tree-like data structures.

Composite

Interpreter or Visitor given partial credit, because these are often used to write producers of a tree-like datatype

(b) This design pattern is used for operating over sequences of elements.

Map/Filter/Reduce

(c) This design pattern uses higher-order functions.

Map/Filter/Reduce

Visitor also given full credit, because a Visitor is a functional object

(d) This design pattern is used to separate concerns in user interfaces.

Model/View/Controller

Event Listener also given full credit, because events decouple models from views and input from output

(e) This design pattern is used for message passing over a network.

Client/Server

Name: _____

Interpreter/Visitor [16 pts]

You want to write a program to perform operations on all your Foos.

A Foo can perform lots of different tricks, like *bazzle* and *glibble*.

There are (and will always be) exactly 4 types of Foo, each of which does something different when they *bazzle* or *glibble*.

But every so often your Foos learn a new trick, and you must update your program to include the new operation.

For example, last week your Foos learned how to *joople*.

a) Would it be better to use the interpreter pattern or the visitor pattern for implementing the datatype representing a Foo?

Visitor, because the variants of the Foo datatype are fixed, but new operations appear from time to time.

b) Assuming you designed your program according to your choice in part (a), now you want to add the *joople* operation. Explain what classes and methods you will change, or what classes and methods you would add, in order to support the *joople* operation.

Add a JoopleVisitor class implementing the Foo's Visitor interface. JoopleVisitor needs to define a visit() or on() method for each of the four variants of Foo. (Optional: also add a static method that makes it easy to call *joople* on a Foo, by encapsulating the construction of the JoopleVisitor.)

Name: _____

Map/Filter/Reduce [12 pts]

Suppose you want to rewrite the following Python code using map, filter, and reduce:

```
def ssp(list):      # sum of squares of positive numbers in list
    result = 0
    for x in list:
        if x > 0:
            result += x*x
    return result
```

Fill in the blanks in the map/filter/reduce version below.

```
def ssp(list):      # sum of squares of positive numbers in list
    return reduce(r, map(m, filter(f, list)), 0)

def f(   x                                    ):

    return   x > 0                                    

def m(   x                                    ):

    return   x * x                                    

def r(   x, y                                    ):

    return   x + y                                    
```

Name: _____

Concurrency [16 pts]

Read the following code:

```
public static void main() {
    Thread t1 = new Thread(new Runnable() {
        public void run() {
            System.out.print("O");
            System.out.print ("Y");
        }
    });
    Thread t2 = new Thread(new Blue());
    System.out.print("R");
    t1.start();
    System.out.print("G");
    t2.start();
    System.out.print("I");
    t1.join();
    System.out.print("V");
    t2.join();
    System.out.print("K");
}

public static class Blue implements Runnable {
    public void run() {
        System.out.print("B");
    }
}
```

Assume that print() is threadsafe and atomic. Which of the following sequences can be printed by this code? Circle possible or impossible.

ROYGBIVK	possible	impossible
ROYBGIVK	possible	impossible
RGOYIBVK	possible	impossible
OYBRGIVK	possible	impossible

Name: _____

Deadlock [16 pts]

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:	T1:
X.acquire();	X.acquire();
Y.acquire();	Y.acquire();
Y.release();	X.release();
X.release();	Y.release();

(no deadlock)

b)

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

no deadlock

c)

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

(no deadlock)

Name: _____

Thread Safety [20 pts]

Consider the following code, and answer the questions on the next page.

```
public class Widget extends Thread {
    public static List<String> strings = new ArrayList<String>();
    public int count;
    public List<Integer> numbers;

    public Widget() {
        count = 0;
        numbers = new ArrayList<Integer>();
    }

    public void run() {
        for (int i = 0; i < 1000; ++i) {
            synchronized (this) {
                count++;
                synchronized (numbers) {
                    numbers.add(i);
                }
                synchronized (Widget.strings) {
                    Widget.strings.add("x");
                }
            }
        }
    }

    public static void main(String[] args) {
        List<Widget> widgets = new ArrayList<Widget>();
        for (int i = 0; i < 1000; ++i) {
            Widget w = new Widget();
            widgets.add(w);
            w.start();
        }
        for (Widget w : widgets) {
            synchronized (w) {
                w.count++;
                synchronized (w.numbers) {
                    w.numbers.add(1000);
                }
            }
            synchronized (Widget.strings) {
                Widget.strings.clear();
            }
        }
        for (Widget w : widgets) {
            w.join();
        }
    }
}
```

You are reviewing a concurrency argument about this code. Circle whether you agree or disagree with each of the following statements in the concurrency argument, and **add a brief (1 sentence) justification of your answer**.

(a) Accesses to the `widgets` list are safe because the list is confined to the main thread.

Name: _____

AGREE DISAGREE

The only reference to the widgets list is the local variable widgets in main(), which is never shared with any other thread.

(b) Accesses to the `numbers` list are safe because they acquire the list's lock.

AGREE DISAGREE

All accesses to the numbers list happen inside a synchronized(numbers) block.

(c) Assuming that the program terminates without throwing an exception, `count` for every widget is 1001 at the end of main.

AGREE DISAGREE

All accesses to count are guarded by the Widget object's lock, and the Widget's run() increments it 1000 times while the main() loop increments it once, producing 1001.

(d) Assuming that the program terminates without throwing an exception, `strings` has size 0 at the end of main.

AGREE DISAGREE

The strings.clear() in main() races against the strings.add() calls in run(); even though both are threadsafe, it may be that the last operation executed against strings is an add().

END OF QUIZ

6.005 Elements of Software Construction

Fall 2011