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

Name:_____

Athena* User Name:_____

Instructions

This quiz is 50 minutes long. It contains 8 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	7-8	20	

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.

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

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

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

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

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?

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.

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(_____):

    return _____


def m(_____):

    return _____


def r(_____):

    return _____
```

Concurrency [16 pts]

Read the following code:

```java
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

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

Thread Safety [20 pts]

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

```java
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.

AGREE   DISAGREE

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

AGREE   DISAGREE

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

AGREE   DISAGREE

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

AGREE   DISAGREE

END OF QUIZ

6.005 Elements of Software Construction

Fall 2011