

Massachusetts Institute of Technology  
6.170 Laboratory in Software Engineering  
Fall 2003  
Quiz 1 Solutions  
Tuesday, 30 September 2003

Name: \_\_\_\_\_

Athena User Name: \_\_\_\_\_

TA (circle one):

- |                        |               |                  |
|------------------------|---------------|------------------|
| 1. Matthew Notowidigdo | 2. Lee Lin    | 3. David Saff    |
| 4. Brian Dunagan       | 5. Thomer Gil | 6. Sammer Ajmani |
| 7. Roger Moh           |               |                  |

Instructions

This quiz is 50 minutes long. It contains 29 questions in 12 pages (including this page). 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 and recitation section number on the top of every page. Please write neatly. No credits will be given if we cannot read what you write.

For all questions requiring explanations, you are restricted to using at most twenty (20) words in your answer. We will only grade the first twenty words of your answer, if it is longer than twenty words.

**True/False [2 points each]**

Circle the correct answer:

For questions 1 to 6, please refer to Figure 1 below.

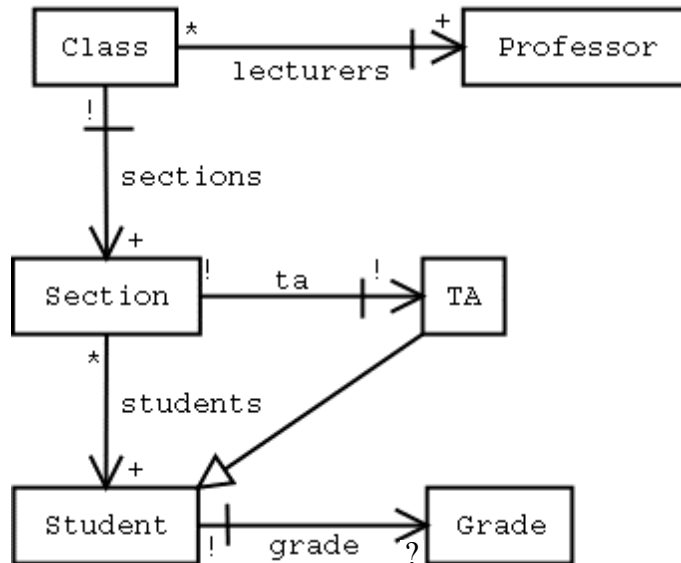


Figure 1: Object Model

1.  T /  F  
A Professor can teach more than one Class.
2.  T /  F  
A student can both attend and TA a Class.
3.  T /  F  
The TA for a given Section can change.
4.  T /  F  
The Grade for a given Student can change.
5.  T /  F  
A Section can contain zero students.
6.  T /  F  
A class can have zero TAs.

Section: \_\_\_\_\_ Name: \_\_\_\_\_

7.  T /  F  
For two Java objects A and B, if  $A == B$ , then `A.equals(B)` must be true under the Object contract.
8.  T /  F  
Given two different specifications A and B (for a method `zeeb()`), it must be that either A is stronger than B or that B is stronger than A.
9.  T /  F  
All else being equal, a specification with “modifies: none” is a stronger than one with “modifies: this”.
10.  T /  F  
Removing a requirement from a specification makes the precondition weaker and the specification weaker.
11.  T /  F  
A class’s representation invariants can be fully defined in its object model.
12.  T /  F  
A representation invariant must hold at the end of all methods of a class, including private methods.
13.  T /  F  
An observer may never mutate the concrete state of an object.
14.  T /  F  
An abstraction function must be 1-to-1.

Section: \_\_\_\_\_ Name: \_\_\_\_\_

15. [6 points] State clearly what the program prints out.

```
public class Test {
    public static void main(String[] args) {
        String a = "6.170 rocks";
        System.out.println(a);
        String b = a;
        b.toUpperCase();
        System.out.println(b);
        if ( a == b.toUpperCase() ) {
            System.out.println("a==b.toUpperCase()");
        }
        if (a.equals(b.toLowerCase())) {
            System.out.println("a equals b.toLowerCase()");
        }
    }
}
```

**6.170 rocks**

**6.170 rocks**

**a equals b.toLowerCase()**

16. [4 points] Insert the necessary downcasts into the code below to ensure that it compiles.

```
interface List {
    public void add(Object element);
    public Object get(int index);
}

public class Test {
    public static void main(String[] args) {

        List myList = new ArrayList();
        myList.add("a");
        myList.add("b");

        String first = (String) myList.get(0);
        String second = (String) myList.get(1);

        System.out.println(first.concat(second));

    }
}
```

Section: \_\_\_\_\_ Name: \_\_\_\_\_

17. [6 points] State clearly what the following codes print out when B.main() is run.

```
class A {
    public A() {
        System.out.println("A");
    }
};
class B extends A {
    public B() {
        System.out.println("B");
    }
    public static void main(String args[]) {
        A a = (A) new B();
        B b = new B();
    }
};
```

**A**  
**B**  
**A**  
**B**

18. [4 points] If the initial size check in the code below were eliminated, the method would still throw an exception when it attempted to pop an element from an empty stack. What is wrong with eliminating the check? Explain in at most twenty words.

```
public Object pop()
    if (size == 0)
        throw new EmptyStackException();
    Object result = elts[--size];
    elts[size] = null;
    return result;
}
```

**It messes up the size of the stack.**

For questions 19 to 20, refer to Figure 2 below.

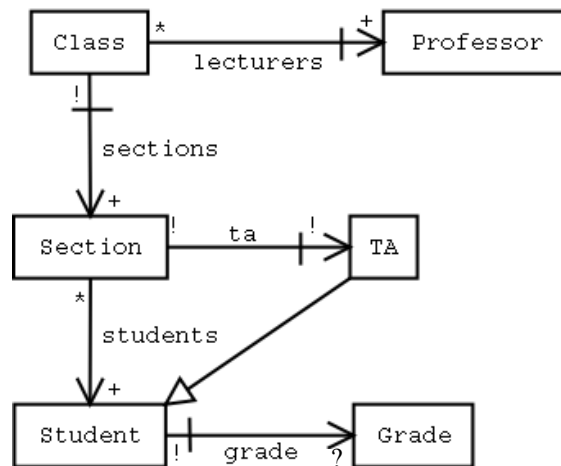
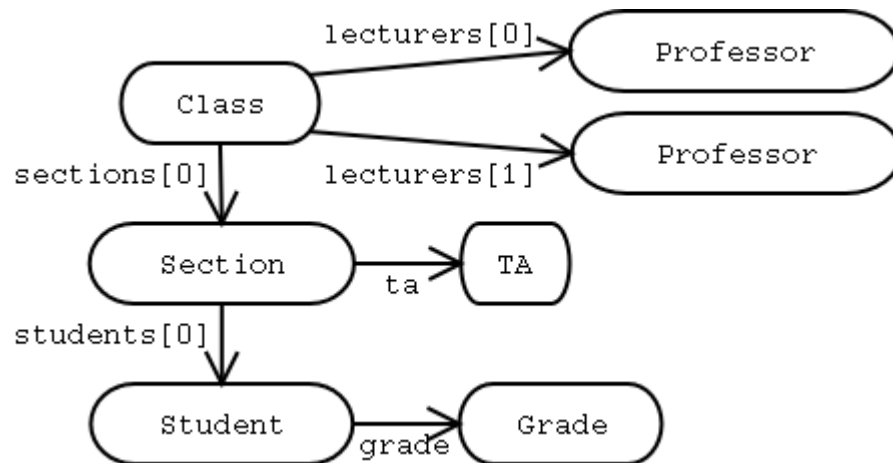


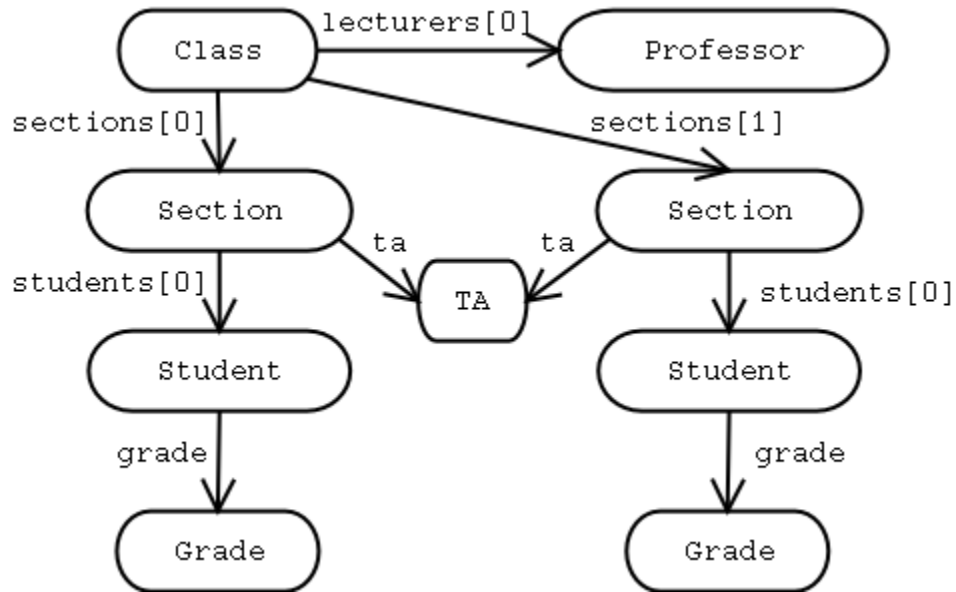
Figure 2: Object Model (Same a Figure 1)

19. [4 points] Is the following Object Diagram consistent with the Object Model in Figure 2? If it is consistent, say “consistent”. Otherwise explain why it is inconsistent in at most twenty words.



**Consistent.**

20. [4 points] Is the following Object Diagram consistent with the Object Model in Figure 2? If it is consistent, say “consistent”. Otherwise explain why it is inconsistent in at most twenty words.



**Inconsistent. A TA can only teach one section.**

21. [4 points] If we prove that a class representation invariant holds (1) after all constructors, (2) on entry to all public methods, and (3) every method in the class preserves the representation invariant, then the client only sees well-formed objects at all times that satisfy the representation invariant. Is this correct? Why or why not? Explain in at most twenty words.

**Incorrect. RI can still be violated due to rep exposure.**

Section: \_\_\_\_\_ Name: \_\_\_\_\_

22. [6 points] In the code below, insert all **necessary** (i.e., not redundant) `checkRep()` method invocations.

```
public class PositiveNumber {

    private int positiveNumber;

    public PositiveNumber(int aNum) {
        positiveNumber = aNum;
        checkRep();
    }

    public PositiveNumber add (PositiveNumber a) {
        PositiveNumber result =
            new PositiveNumber(positiveNumber + a.positiveNumber);
        return result;
    }

    private void checkRep() {

        if ( positiveNumber <= 0 ) {

            throw new RuntimeException("rep invariant violated");

        }

    }

}
```

23. **[4 points]** Generally speaking, are representation invariants better or worse than result-checking invariants at discovering the source of bugs? A result-checking invariant is something that checks if the final result is consistent with the inputs, e.g., `assert(result == a * b)`. Explain your answer in at most twenty words.

**Yes, RI detects errors at the source, which is hard to track by other means.**

24. **[5 points]** We have the following class definition:

```
class IntSet {
    //AF: Represents a set of integers
    //RI: elts is not null, elts contains no duplicates

    List elts;

    IntSet() {
        elts = new ArrayList();
    }

    void add(Integer i) {
        if (!elts.contains(i))
            elts.add(i);
    }

    void contains(Integer i) {
        return elts.contains(i);
    }
}
```

Map the following concrete representations to their abstract value if one exists. The following notation is used: `[a,b,c]` represents the `elts` array list. `{a,b,c}` represents the abstract mathematical set containing the distinct elements `a`, `b`, and `c`. Write “no value” if none exists.

Concrete Representation	Abstract Value
<code>[1,2,3]</code>	<code>{1,2,3}</code>
<code>[1,2,2]</code>	No value
<code>[3,2,1]</code>	<code>{1,2,3}</code>
<code>[]</code>	<code>{}</code>
Null	No value

Section: \_\_\_\_\_ Name: \_\_\_\_\_

25. [4 points] We have the following specification fields for a Date class.

```
/**
 * An ADT representing a point in time.
 *
 * @specfield month
 * @specfield day
 * @specfield year
 * @specfield hour
 * @specfield minute
 * @specfield second
 * @specfield millisecond
 */
```

Does the implementer need to create fields for each specified fields listed above? Why or why not? Explain in at most twenty words.

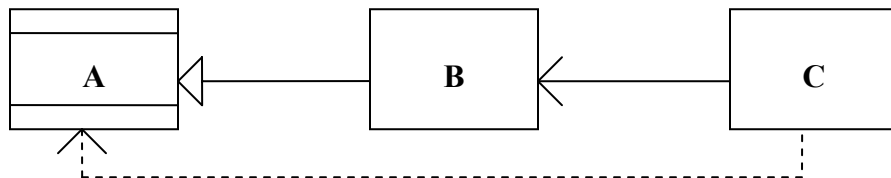
**No. The implementer need not create an implementation field for every spec field.**

**E.g., The implementer can simply create a field that contains the number of seconds since 1 Jan 1970 and derive the specfields using this single concrete field.**

26. [6 points] For the following code:

```
class C {
    public static void main(String args) {
        A a = new B();
    }
}
```

Fill in all the appropriate edges in the MDD diagram below.



Section: \_\_\_\_\_ Name: \_\_\_\_\_

27. **[5 points]** Consider the code below. Explain any problems with the code in at most twenty words. Ignore performance-related issues and assume that the code compiles with an appropriate collection and `zeeb` class.

```
try {
    Iterator t = collection.iterator();
    while(true) {
        Zeeb zeeb = (Zeeb) t.next();
        zeeb.g();
        ...
    }
}
catch (NoSuchElementException e) {
}
```

**We have no way of knowing if `NoSuchElementException` is thrown or handled. Also, we don't know if the exception was thrown by the iterator or by method `g()`.**

28. **[4 points]** Why are total procedures better than partial procedures? Explain in at most forty words.

**A partial procedure has undefined behavior for inputs outside its domain, which can cause bugs. A total procedure has defined behavior (i.e., returning values or throwing exceptions) for its entire domain and so avoids such bugs.**

Section: \_\_\_\_\_ Name: \_\_\_\_\_

29. [6 points] Explain in at most twenty words why the specification below is bad. Explain in at most twenty additional words how exceptions can improve it.

```
public static int div(float a, float b)
// effects: if b is nonzero, returns a divided by b,
//           else returns zero
```

**The caller must check the return value of this method to determine whether it failed, and the caller cannot distinguish between failure and a legitimate zero return value (i.e., when  $b > a$ ). If this method instead threw an exception when  $b$  is zero, then the caller would not have to check the method's return value.**

- END OF QUIZ -