

Your name: \_\_\_\_\_ **SOLUTION** \_\_\_\_\_

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
6.170 LABORATORY IN SOFTWARE ENGINEERING  
SPRING 2000

Quiz 2 part A  
April 4, 2000

There are 5 sections (labeled A through E) and 10 pages. Please check your copy of the quiz before you start to make sure it is complete.

Write your name at the top of **every** page of this exam before you start.

**Name:** \_\_\_\_\_ **SOLUTION** \_\_\_\_\_

**TA Name:** \_\_\_\_\_

| <b>Section</b> | <b>Max</b> | <b>Score</b> | <b>Grader</b> |
|----------------|------------|--------------|---------------|
| A              | 10         |              |               |
| B              | 7          |              |               |
| C              | 7          |              |               |
| D              | 8          |              |               |
| E              | 8          |              |               |
| <b>TOTAL</b>   | <b>40</b>  |              |               |

### A. True/False (10 points, 10 questions)

Write *TRUE* or *FALSE* in the blank to the left of each question.

- T** 1. If type A is a true subtype of type B, then all code that operates correctly on objects of type B will operate correctly on objects of type A.
- F** 2. In a Java program, if class A is declared to extend class B, then A is a true subtype of B.
- T** 3. The primary purpose of an abstraction function is to indicate how the representation data structure of an ADT is to be interpreted.
- F** 4. The only significant problem with exposing the representation of an ADT is that clients may write code that depends on the representation, making it difficult to change the representation later.
- T** 5. There may be multiple concrete states in an ADT that correspond to a single abstract state of that ADT.
- F** 6. In a module dependency diagram, a module A depends on a module B if a change in B's behavior can change A's behavior.
- F** 7. An object model is a graphical representation that indicates the objects in a program and the references between them.
- T** 8. When following a bottom-up implementation approach, each module is implemented after the modules that it depends on.
- T** 9. In his 6.170 guest lecture, Peter Schmidt stated that in his opinion, undocumented code has almost no commercial value.
- F** 10. In a correct implementation of an ADT, the representation invariant will hold for all objects of that ADT at all points during the program's execution.

**B. Short Answer (7 points, 3 questions)**

11. Why is it dangerous for the rep invariant of a subclass to be weaker than the rep invariant of the corresponding superclass?

An inherited superclass method might fail if the subclass changes data fields of the superclass in a way consistent with the subclass rep invariant but inconsistent with the superclass rep invariant.

12. When building a software system, sometimes it is appropriate to build a prototype. What are two example questions you might ask about a design you are working on that would be best answered by building a prototype?

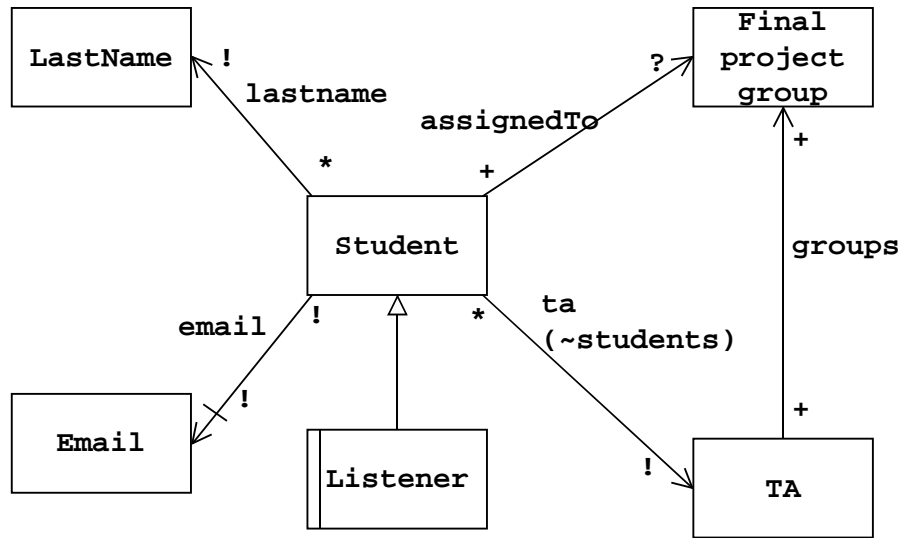
There were many possible answers to this problem. See the lecture on prototyping. A clear statement of each question was required to receive credit.

In the following table, the first column is the specification of a method from a interface A, and the second is the specification of a corresponding method from interface B. In the third column, write YES if interface B could potentially be a true subtype of interface A, and write NO if interface B could not be a true subtype of interface A.

Each row of the table is independent.

| Method from A  | Method from B  | Could B be a true subtype of A? |
|--|--|---------------------------------|
| <pre>void insert1(int x) // modifies: this // effects: adds x to the contents of this</pre>          | <pre>void insert1(int x) // modifies: this // effects: if x is odd adds it to this else does nothing</pre>             | <b>NO</b>                       |
| <pre>void insert2(int x) // modifies: this // effects: adds x to the contents of this</pre>          | <pre>void insert2(int x) // modifies: this // effects: adds x to the contents of this and prints x to the screen</pre> | <b>YES</b>                      |
| <pre>void addZero() //requires: this is not empty // modifies: this // effects: adds 0 to this</pre> | <pre>void addZero() // modifies: this // effects: adds 0 to this</pre>   | <b>YES</b>                      |

**C. Object Models (7 points, 7 questions)**



There are no textual constraints, only the graphical constraints shown in the model.

For each of the following statements, answer A, B, or C:

- A. The above OM indicates that this statement **holds for all** possible snapshots of the system.
- B. The above OM indicates that this statement **does not hold in any** snapshot of the system.
- C. The above OM indicates that this statement **may hold in some** snapshots and not in others.

\_\_\_ **A** \_\_\_ 14. No student is assigned to more than one final project group.

\_\_\_ **C** \_\_\_ 15. Every student who is not a listener is assigned to a final project group.

\_\_\_ **A** \_\_\_ 16. No two students have the same email address.

\_\_\_ **A** \_\_\_ 17. Every listener is a student.

\_\_\_ **A** \_\_\_ 18. No student is also a TA.

Decide whether the following statements are TRUE or FALSE.

\_\_\_ **F** \_\_\_ 19. The OM indicates that a student's email address may change over time.

\_\_\_ **F** \_\_\_ 20. The OM indicates that it is possible for a student who was not a listener at the start to change to become a listener.

**D. Reasoning about ADTs (8 points, 1 question)**

Consider the `IntSet` ADT implementation shown in the appendix at the end of this quiz. There is a bug in the implementation. A test case that exposes this bug is

```
IntSet i = new IntSet(5);
i.insert(2);
i.insert(3);
i.max();      // => 3
i.remove();   // => 3
i.member(2);  // => true
i.member(3);  // => false
i.max();      // => 3      bug!!
```

21. Write a representation invariant for this implementation. The rep invariant you write should have three properties:

- It should be a complete rep invariant; that is, it should constrain all aspects of the representation that need to be constrained for the operations to function correctly.
- It should indicate that the bug is in `max()` and all other functions are correct.
- If you were to change `max()` so that it is correct with respect to your rep invariant, the resulting implementation of `max()` would not be substantially slower than the one shown in the appendix.

```
mData ≠ null
^ mData.length ≥ mMaxSize
^ 0 ≤ mNextFreeIndex ≤ mMaxSize
^ (0 ≤ mIndexOfMax < mNextFreeIndex)
  ⇒ ∀ i: 0 ≤ i < mNextFreeIndex, mData[mIndexOfMax] ≥ mData[i]
^ (mNextFreeIndex > 0)
  ⇒ nodups (mData[0] ... mData[mNextFreeIndex-1])
```

**E. Reasoning (8 points, 2 questions)**

Consider the ADTs and proof shown in the appendix at the end of this test.

There are two major flaws in the reasoning. Neither flaw is a “typo” in the use of a logical symbol, or an ambiguity in the use of language; they are bigger than that.

22. There is a flaw in the proof which makes it invalid. What is the flaw?

**No base case for the induction.**

23. There is a flaw in the assumptions made by the proof, so fixing the flaw in the previous problem would not be sufficient to make the reasoning correct. What is the flaw in the assumptions?

**The proof assumes that the color of each horse is immutable, but the horse ADT is mutable.**

**END OF TEST. THERE ARE NO FURTHER QUESTIONS.**

**Code for section D (page 1 of 2)      You may detach this sheet from the quiz**

```
class IntSet {
    // An IntSet is a finite set of integers with an associated
    // maximum size maxSize. A sample IntSet is {1, 2}, maxSize 10.

    private int    mMaxSize;           // max # of elements in the set
    private int[] mData;              // stores values in the set
    private int    mNextFreeIndex;    // how many ints are in the set
    private int    mIndexOfMax;       // index of biggest element

    public void IntSet(int maxSize)
        // effects: make a new, empty IntSet with maxSize = maxSize
    {
        mData = new int[maxSize];
        mMaxSize = maxSize;
        mIndexOfMax = -1;
        mNextFreeIndex = 0;
    }

    public void size()
        // effects: return |this|
    {
        return mNextFreeIndex;
    }

    public void insert(int elem)
        // requires: |this| < this.maxsize
        // modifies: this
        // effects:  this_post = this ∪ {elem}
    {
        if (!member(elem)) {
            mData[mNextFreeIndex] = elem;
            mNextFreeIndex += 1;
            mIndexOfMax = -1;
        }
    }
}
```

**Code for section D (page 2 of 2)****You may detach this sheet from the quiz**

```
public int max()
    // requires: |this| > 0
    // effects: returns some  $a \in \text{this}$  such that  $\forall b \in \text{this}, a \geq b$ 
{
    if (mIndexOfMax < 0) {
        mIndexOfMax = 0;
        for (int i=1; i<mNextFreeIndex; i++) {
            if (mData[i] > mData[mIndexOfMax]) {
                mIndexOfMax = i;
            }
        }
    }
    return mData[mIndexOfMax];
}

public int remove()
    // requires: |this| > 0
    // modifies: this
    // effects: returns an arbitrary element  $a$  of this
    //           this_post = this - {a}
{
    int last = mData[mNextFreeIndex - 1];
    mNextFreeIndex -= 1;
    return last;
}

public boolean member(int a)
    // effects: return true iff  $a \in \text{this}$ 
{
    for (int i=0; i<mNextFreeIndex; i++) {
        if (mData[i] == a) {
            return true;
        }
    }
    return false;
}
}
```

**ADTs and proof for Section E****You may detach this sheet from the quiz**

```

class Horse {
    // A Horse is a mutable colored horse. A typical Horse is [black].
    public Horse(Color c);      // effects: makes a new Horse colored c
    public Color color();      // effects: returns the color of this.
    public void setColor(Color c); // effects: this_post.color = c.
    public boolean equals(Horse h); // standard definition of equals()
}

class HorseSet {
    // A HorseSet is a finite mutable set of colored horses in which
    // all members are of different colors. A typical HorseSet is
    // {[black], [brown]}
    public HorseSet();
    // effects: creates a fresh HorseSet = {}.
    public void insert(Horse h);
    // effects: if  $\exists h1 \in \text{this}$  where  $h1.\text{color} = h.\text{color}$ 
    // then  $\text{this\_post} = \text{this}$ 
    // else  $\text{this\_post} = \text{this} \cup h$ 
    public boolean member(Horse h);
    // effects: return true if  $\exists h1 \in \text{this}$  where  $h1.\text{equals}(h)$ 
    public int size();
    // effects: return  $|\text{this}|$ 
}

```

Conjecture:  $|S| > 1 \Rightarrow (\forall h1, h2 \in S [h1 \neq h2 \Rightarrow h1.\text{color} \neq h2.\text{color}])$

Prove the conjecture by induction on insert.

1. Induction hypothesis: Conjecture holds for S
2. Goal: Conjecture holds for S1 where S1 is the result of inserting a horse h into S.
3. Meaning of insert: if  $\exists h1 \in S$  where  $h1.\text{color} = h.\text{color}$  then  $S1 = S$  else  $S1 = S \cup \{h\}$

There are two cases, so analyze each case separately.

4. (first case) If  $(\exists h1 \in S \text{ where } h1.\text{color} = h.\text{color})$  then  $S1 = S$ , so Conjecture holds for S1 by Induction hypothesis

5. (second case) If not  $(\exists h1 \in S \text{ where } h1.\text{color} = h.\text{color})$  then  $S1 = S \cup \{h\}$ . By the induction hypothesis, we know that all horses in S have different colors. Furthermore, we know by the case we are in that none of those horses have the same color as h. Therefore we know that all horses in S1 have different colors, by the meaning of union.