

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.170 Laboratory in Software Engineering

Handout 10

Oct 14, 1999

Quiz 1

Name: _____

Name of TA: _____

This quiz is 50 minutes long. It contains 13 pages; please check your copy to make sure it contains all the pages before you start.

Problem	Max	Yours
1	20	
2	15	
3	25	
4	10	
5	20	
Total	100	

PROBLEM 1: True/False (20 points)

- _____ a. A Java class can contain several methods with the same name.
- _____ b. if x and y have the same state, $x.equals(y)$ should return true.
- _____ c. Only mutable data abstractions can have methods that perform benevolent side effects.
- _____ d. It isn't possible to define path-complete tests for some programs.
- _____ e. If procedure A is only called by B and B ensures that A 's requires clause is met, then the requires clause can be removed from A 's specification.
- _____ f. If `repOk` runs in linear time, the implementation is efficient.
- _____ g. The `rep` of an object need not satisfy the `rep` invariant while a method of the object is running.
- _____ h. The specification for an observer will never have a requires clause.
- _____ i. A class implementing a mutable type can provide a correct implementation of `clone` by inheriting the implementation from `Object`.
- _____ j. If you reimplement an abstraction without changing its specification, you will not need to change your black box tests.

PROBLEM 2: Data Abstraction (15 points)

The following PQ (priority queue) abstraction stores objects with their associated priority, and allows the objects to be retrieved in priority order.

```
public class PQ {
    // overview: A priority queue is a mutable collection of object/priority pairs {<o1, p1>, ..., <on, pn>}.
    // Priorities are integers between 0 and 9, inclusive. The queue orders its elements based
    // on their priority so that higher priority items can be retrieved before lower priority items.

    // constructor:
    public PQ ( )
        // effects: makes this be an empty priority queue.

    // methods
    public void append(Object o, int p) throws BadPriorityException, NullPointerException
        // modifies: this
        // effects: if p < 0 or p > 9 throws BadPriorityException else if
        // o = null throws NullPointerException else adds <o, p> to this.

    public int highestPriority ( ) throws NoElsException
        // effects: if this is empty throws NoElsException else returns
        // the highest priority associated with an element of this.

    public Object remove ( ) throws NoElsException
        // modifies: this
        // effects: if this is empty throws NoElsException else removes and
        // returns an element associated with the highest priority in this.
}
```

For example, consider the PQ object $pq = \{ \langle x, 7 \rangle, \langle z, 5 \rangle, \langle y, 7 \rangle \}$; here x and y have priority 7 and z has priority 5. Object $a = pq.remove()$ will cause either $\langle x, 7 \rangle$ or $\langle y, 7 \rangle$ to be removed from pq .

A copy of this specification is included at the end of the quiz. You can rip it off if you like.

2a) Give the category of each operation of PQ:

creators:

producers:

mutators:

observers:

2b) Describe a complete set of black box tests for remove.

2c) Should NoSuchElementException be a checked or an unchecked exception? Explain briefly.

PROBLEM 3: Data Abstraction Implementation (25 points)

Now consider the following implementation of PQ.

```
public class PQ {

    private Vector[] els;
    private int highest;

    public PQ( ) { els = new Vector[10]; highest = -1;
        for (int i = 0; i < 10; i++) els[i] = new Vector( ); }

    public void append (Object o, int p) throws BadPriorityException, NullPointerException {
        if (p < 0 || p > 9) throw new BadPriorityException( );
        if (o == null) throw new NullPointerException( );
        if (p > highest) highest = p;
        els[p].addElement(o);
    }

    public int highestPriority ( ) throws NoElsException{
        if (highest < 0) throw new NoElsException( ); else return highest;
    }

    public Object remove ( ) throws NoElsException {
        if (highest < 0) throw new NoElsException( );
        int s = els[highest].size( ) - 1;
        Object o = els[highest].elementAt(s);
        els[highest].removeElementAt(s); // removes last element of els[highest]
        if (s == 0)
            for (int i = highest-1; i ≥ 0; i--)
                if (els[i].size( ) > 0) { highest = i; return o; }
        highest = -1;
        return o;
    }
}
```

A copy of this implementation is appended to the end of the quiz. You can rip it off if you like.

3a) Give the abstraction function for this implementation. (English is ok.)

3b) Give the rep invariant for this implementation. (English is ok.)

3c) Consider the glass box testing of `remove`. Are any additional test cases needed over what you proposed in 2b? Explain briefly and define the test cases if any.

3d) A student argues that having highest in the rep is not really worthwhile. Do you agree with this? Explain briefly what the performance impact of removing highest from the rep would be.

3e) Consider the following additional method:

```
Vector fetchHigh( ) throws NoElsException {  
    // effects: if this is empty throws NoElsException else returns  
    // a vector containing all the elements with the highest priority in this  
    if (highest == -1) throw new NoElsException( );  
    return els[highest];  
}
```

A student claims that there is a serious problem with this method. Is this true? Explain briefly.

PROBLEM 4: Iterators (10 points)

4a) A student claims that PQ is not adequate and proposes to fix this problem by adding the following iterator:

```
Enumeration elementsAt (int p) throws BadPriorityException
// effects: If p < 0 or p > 9 throws BadPriorityException else returns a generator
// that will produce all the objects of this with priority p, each exactly once.
```

Is the claim about adequacy correct, and if so, does the addition of the `elementsAt` iterator fix the problem? Explain briefly. Also, if the problem isn't fixed, explain how to fix it.

4b) A student claims that there is a problem with the specification of `elementsAt`. Is this true and if so how should the specification be fixed? Explain briefly.

4c) Another student proposes an alternative to `elementsAt`: two methods, `startGen` and `next`:

```
void startGen (int p) throws BadPriorityException
// effects: if p < 0 or p > 9 throws BadPriorityException else prepares to
// generate all elements of this with priority p, each exactly once.
```

```
Object next ( ) throws NoSuchElementException
// requires: startGen has been called.
// effects: if there are no more elements to generate throws
// NoSuchElementException else returns the next element and
// records the return so that that element won't be returned again.
```

Is this a good solution compared to the iterator? Discuss briefly.

PROBLEM 5: Short Questions (20 points)

5a) Suppose you have written a procedure that you then have to test. During testing you find only one bug, which you fix right away. Can you now claim that your implementation is correct? Explain briefly.

5b) Consider the following abstractions:

```
int min (int[ ] a)
// effects: if a is null or a is empty returns 0 else returns minimum value of a
```

```
int min (int[ ] a) throws EmptyException, NullPointerException
// effects: if a is null throws NullPointerException; if a is empty
// throws EmptyException else returns the minimum element of a
```

```
int min (int[ ] a)
// requires: a is not null and a is not empty
// effects: returns the minimum element of a
```

Which of these abstractions is better. Explain briefly.

5c) Consider the following procedure:

```
void reverse (int[ ] a, int[ ] b) {  
    // effects: makes b contain the elements of a in reverse order  
    if (a == b || a == null || b == null || a.length != b.length) throw new BadArgsException( );  
    int bi = b.length - 1;  
    for (int i = 0; i < a.length; i++)  
        b[bi-i] = a[i];  
}
```

Complete the specification for reverse. Also, explain why the test `a == b` is present in the code.

5d) Briefly explain two problems with the following specification.

```
int squares (int[ ] a, int i)  
// requires:  $0 \leq i < a.length$   
// effects: if i is not a legal index in a throws IndexOutOfBoundsException  
// else returns the sum of squares of elements 0,...,i of a.
```

Specification of PQ data abstraction:

```
public class PQ {
    // overview: A priority queue is a mutable collection of object/priority pairs {<o1, p1>, ..., <on, pn>}.
    // Priorities are integers between 0 and 9, inclusive. The queue orders its elements based
    // on their priority so that higher priority items can be retrieved before lower priority items.

    // constructor:
    public PQ ( )
        // effects: makes this be an empty priority queue.

    // methods
    public void append(Object o, int p) throws BadPriorityException, NullPointerException
        // modifies: this
        // effects: if p < 0 or p > 9 throws BadPriorityException else if
        // o = null throws NullPointerException else adds <o, p> to this.

    public int highestPriority ( ) throws NoElsException
        // effects: if this is empty throws NoElsException else returns
        // the highest priority associated with an element of this.

    public Object remove ( ) throws NoElsException
        // modifies: this
        // effects: if this is empty throws NoElsException else removes and
        // returns an element associated with the highest priority in this.
}
```

Implementation of PQ data abstraction:

```
public class PQ {

    private Vector[] els;
    private int highest;

    public PQ ( ) { els = new Vector[10]; highest = -1;
        for (int i = 0; i < 10; i++) els[i] = new Vector( ); }

    public void append (Object o, int p) throws BadPriorityException, NullPointerException {
        if (p < 0 || p > 9) throw new BadPriorityException( );
        if (o == null) throw new NullPointerException( );
        if (p > highest) highest = p;
        els[p].addElement(o);
    }

    public int highestPriority ( ) throws NoElsException{
        if (highest < 0) throw new NoElsException( ); else return highest;
    }

    public Object remove ( ) throws NoElsException {
        if (highest < 0) throw new NoElsException( );
        int s = els[highest].size( ) - 1;
        Object o = els[highest].elementAt(s);
        els[highest].removeElementAt(s); // removes last element of els[highest]
        if (s == 0)
            for (int i = highest-1; i ≥ 0; i--)
                if (els[i].size( ) > 0) { highest = i; return o; }
        highest = -1;
        return o;
    }
}
```