SE 450 Fall 2004
Midterm Exam

October 8, 2003

Name: ________________

Directions: Write your name in the blank at the top of this page.

You have **two hours and thirty minutes** (2:30) to take this exam.

It is **closed book and closed notes**.

You may consult one letter-size page (front and back) of notes that you have prepared.

Write your answers **on the exam**.

There are 5 questions, some with mutiple parts.

If I can not read your answer clearly, then it will be marked as incorrect.

Be sure to understand the question before you study the associated code samples.

You can find some information from the j2sdk and junit on the back page.

When drawing UML diagrams, follow these guidelines:

- Be as specific as possible (prefer aggregation and dependency to association, prefer composition to aggregation).
- Show mutiplicities on aggregations and compositions.
- Do *not* give names or roles to associations.

Question	**Value**	**Score**
1	18	
2	18	
3	10	
4	12	
5	42	
Total	100	

Question 1
Consider the class `NonNegativeInteger` with the following interface:

```
class NonNegativeInteger {
  public NonNegativeInteger();
  public boolean equals (Object that);
  public void set(int v) throws IllegalArgumentException;
  public int get();
}
```

The invariants for the class are:

- if `set` has not been called, `get` should return 0
- if `set` has been called, `get` should return the value of the last `set`
- `get` should never return a negative value

In this question you must write tests for `NonNegativeInteger`.

- Write tests to check that this class obeys all of its invariants.
- Write a test to check the `equals` operation. (It should return true exactly when `this.get()` is the same as `that.get()`.)

You may use methods from the `Assert` class summarized on the last page of the exam.

```
public class NonNegativeIntegerTEST extends TestCase {
  public void test1() {
```

Question 2
Consider the following class:

```
class MutableInteger {
  private int _v;
  public void set(int v) { _v = v; }
  public int get() { return _v; }
}
```

Suppose that you are required to modify `MutableInteger` to perform some function each time a particular `MutableInteger` is `set`. However, you do not know the exact function to be performed.

(For example, it might be neccesary to print some changes to the screen, or to keep count of the number of times `set` has been invoked.)

(a) What pattern can help you re-write `MutableInteger` to suit these requirements?

(b) Draw a UML class diagram sketching your solution. Draw your classes within a package.

Show two client classes in a separate package. Client class `C1` will cause a print to occur whenever its `MutableInteger` is set; Client class `C2` will keep a count of the number of times its `MutableInteger` is set.

In each class you *must include* the names of declared methods.

For each method you *must also* include parameter names and types and a return type.

(c) Draw a UML sequence diagram showing the interactions of a program that:

- creates one instance each of `MutableInteger`, `C1` and `C2`, informing `C1` and `C2` to keep track of `set` methods on the `MutableInteger`.
- calls `set` on the `MutableInteger`.

Question 3
For the purposes of this problem, assume that you are given classes `MyStack` and `MyQueue` to implement interface `MyContainer` in the expected way.

```
interface MyContainer {
  public Object get();       // return item in container
  public void add(Object x); // add item into container
  public void remove();      // remove item from container
  public boolean isEmpty();  // check if container is empty
}
```

You are given the following classes to establish a tree. The method `children()` returns an iterator that is empty for leaves, and returns first the left, then right child for internal nodes.

```
import java.util.ArrayList;
import java.util.Iterator;
interface Tree {
  public void print();
  public Iterator children();
}
class Node implements Tree {
  private String _v;
  private Tree _l, _r;
  public Node(String v, Tree l, Tree r) { _v = v; _l =l; _r =r; }
  public void print() { System.out.println(_v); }
  public Iterator children() {
    ArrayList A = new ArrayList();
    A.add(_l); A.add(_r);
    return A.iterator();
  }
}
class Leaf implements Tree {
  private Integer _v;
  public Leaf(Integer v) { _v = v; }
  public void print() { System.out.println(_v); }
  public Iterator children() {
    return new ArrayList().iterator();
  }
}
```

You must write the output of the main program on the following page.

Consider the following implementation of `Iterator` and class `Main`.

```
import java.util.Iterator;
class TreeIterator implements Iterator {
  private MyContainer _c;
  public TreeIterator(Tree t, MyContainer c) {
    _c = c; _c.add(t);
  }
  public boolean hasNext() { return ! _c.isEmpty(); }
  public void remove() { throw new UnsupportedOperationException(); }
  public Object next() {
    Tree top = (Tree) _c.get();
    _c.remove();
    Iterator i = top.children();
    while (i.hasNext())
      _c.add(i.next());
    return top;
  }
}
```

```
import java.util.Iterator;
class Main {
  public static void main(String[] argv) {
    Tree one = new Leaf(new Integer(1));
    Tree two = new Leaf(new Integer(2));
    Tree three = new Leaf(new Integer(3));
    Tree onetwo = new Node("*", one, two);
    Tree onetwothree = new Node("+", onetwo, three);
    Tree t = new Node("-", onetwothree, onetwo);

    System.out.println("Iterate with Stack");
    TreeIterator i1 = new TreeIterator(t, new MyStack());
    while (i1.hasNext())
      ((Tree) (i1.next())).print();

    System.out.println("Iterate with Queue");
    TreeIterator i2 = new TreeIterator(t, new MyQueue());
    while (i2.hasNext())
      ((Tree) (i2.next())).print();
  }
}
```

What is the output generated by `Main`?

Question 4
Consider the following interfaces for functions on integers and integer arrays.

```
interface IntFun { public int   exec(int   x); }
interface ArrFun { public int[] exec(int[] x); }
```

Here are two classes, one implements an absolute value function, the other a cube function:

```
class Abs implements IntFun {
  public int exec(int x) { return (x < 0) ? -x : x; }
}
class Cube implements IntFun {
  public int exec(int x) { return x*x*x; }
}
```

Complete the following two questions. There is an example with output on the next page.

(a) Complete the following definition of `Comp` which composes two functions. For example,

```
new Comp(new Abs(), new Cube()).exec(-3)
```

should return 27 (computing the absolute value of -3 and the cubing it).

```
class Comp implements IntFun {
  IntFun _f, _g;
  public Comp(IntFun f, IntFun g) { _f = f; _g = g; }
  public int exec(int x) {
```

(b) Complete the following definition of `Map` which performs a integer function `f` on every element of an array, returning a new array.

```
class Map implements ArrFun {
  IntFun _f;
  public Map(IntFun f) { _f = f; }
  public int[] exec(int[] x) {
```

As an example, the following code prints:

```
[ 0 1000 8000 27000 64000 125000 216000 343000 512000 729000 ]
```

```
class Main {
  public static void print(int[] x) {
    System.out.print("[ ");
    for (int i=0; i<x.length; i++)
      System.out.print(x[i]+ " ");
    System.out.println("]");
  }
  public static void main(String[] argv) {
    int[] a = new int[10];
    for (int i=0; i<a.length; i++)
      a[i] = -i*10;

    ArrFun mabs = new Map(new Comp(new Abs(), new Cube()));
    print(mabs.exec(a));
  }
}
```

Question 5
This problem is about the construction of a simple music library. You are given the following class to start off things:

```
class Music {
  static void play(int duration) {/*...*/}
    // play note at the current pitch for the given duration
    // in milliseconds (the initial pitch is A = 440 Hz)
  static void rest(int duration) {/*...*/}
    // rest for given duration
  static void scalePitch(double factor) {/*...*/}
    // multiply the pitch frequency by the given factor
    // (a factor less than one will lower the pitch)
  static void reset() {/*...*/}
    // reset the pitch to note A = 440 Hz
}
```

For example:

```
Music.reset();             // initialize pitch to middle A (440 Hz)
Music.play(500);           // play a middle A for half a second
Music.rest(1000);          // rest for one second
Music.scalePitch(2.0);     // set pitch an octave higher (880 Hz)
Music.play(500);           // play a high A for half a second
Music.rest(250);           // rest for a quarter of a second
Music.scalePitch(1.0/2.0); // reset pitch to middle A (440 Hz)
Music.play(500);           // play a low A for half a second
```

In this problem, we will write code to build and manipulate complex musical objects that are built out of notes and rests. The basic interface is `Event`, and one implementing class is `Note`:

```
interface Event {
  public void play();
}
class Note implements Event {
  int _d;
  double _f;
  public Note(int duration, double factor) {
    _d = duration;
    _f = factor;
  }
  public void play() {
    Music.scalePitch(_f);
    Music.play(_d);
    Music.scalePitch(1.0/_f);
  }
}
```

Now we look into mechanisms for building rests and more complex musical objects. Answer the following questions.

(a) Following `Note`, write a class `Rest` that, when played, will rest for the given duration. Make the duration a parameter of the constructor.

(b) Finish the following code for the `EventGroup` class:

```
class EventGroup implements Event {
  List _events = new LinkedList();
  public void add(Event e) {


  }
  public void play() {
```

(c) Name the pattern most closely associated with `EventGroup`.

(d) Write a class `Transpose` that, when played, will play an event at a scaled pitch. Make the event to be transposed and the scaling factor both parameters of the constructor.

(e) Name the pattern most closely associated with `Transpose`.

(f) Using these classes, write Java code that declares a variable `e1` and construct an object such that "`Music.reset(); e1.play();`" does the following:

- play for 0.25 second a note at 880.0 Hz.
- rest for 0.25 second
- play for 0.5 second a note at 440.0 Hz.
- rest for 0.5 second
- play for 1.0 second a note at 220.0 Hz.

(g) Using these classes and `e1`, write Java code that declares a variable `e2` and construct an object such that "`Music.reset(); e2.play();`" does the following:

- play the events of `e1`
- rest for 1.0 second
- play the events of `e1` transposed by a factor of 2.0

(h) Draw a UML class diagram showing the relationships between these classes.

Show classes and interface only. (Do *not* show method names.)

Some classes/interfaces from the j2sdk and junit:

```
public interface Iterator {
  public boolean hasNext();
  public Object next();
  public void remove();
}
public interface Collection {
  public int size();
  public boolean isEmpty();
  public boolean contains(Object o);
  public Iterator iterator();
  public boolean add(Object o);
  public boolean remove(Object o);
  public void clear();
  ...
}
public interface List extends Collection {
  public Object get(int index);
  public Object set(int index, Object element);
  public void add(int index, Object element);
  public Object remove(int index);
  ...
}
public interface Observer {
  public void update(Observable o, Object arg);
}
public class Observable {
  public synchronized void addObserver(Observer o);
  public void notifyObservers(Object arg);
  protected synchronized void setChanged();
  protected synchronized void clearChanged();
  public synchronized boolean hasChanged();
  ...
}
public class Assert {
  static public void assertTrue(boolean condition);
  static public void assertFalse(boolean condition);
  static public void fail();
  static public void assertEquals(Object expected, Object actual);
  static public void assertEquals(int expected, int actual);
  static public void assertNull(Object object);
  static public void assertNotNull(Object object);
  static public void assertSame(Object expected, Object actual);
  static public void assertNotSame(Object expected, Object actual);
  ...
}
```