CS 61B (Clancy) **Exam 1**
September 26, 2005

Read and fill in this page now.
Do NOT turn the page until you are told to do so.

Your name: _____

Your login name: _____

Your lab t.a.: _____

Your lab time: _____

Problem 0	____	Total:	_____	/30
Problem 1	____			
Problem 2	____	Problem 4	____ ____ ____	
Problem 3	____			

You have approximately two hours to complete this test. You may consult any books, notes, or other paper-based inanimate objects available to you. To avoid confusion, read the problems carefully. If you find it hard to understand a problem, ask us to explain it. If you have a question during the test, please come to the front or the side of the room to ask it.

Some students are taking this exam late. Please do not talk to them, mail them information, or post anything about the exam to news groups or discussion forums until after Tuesday.

This exam comprises 15% of the points on which your final grade will be based. Partial credit may be given for wrong answers. Your exam should contain five problems (numbered 0 through 4) on 8 pages; a supplementary handout will also be distributed at the exam. Please write your answers in the spaces provided in the test; in particular, we will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there.

Relax—this exam is not worth having heart failure about.

Your login name: cs61b-_____

Problem 0 (1 point)

Put your login name on each page. Also make sure you have provided the information requested on the first page.

Problem 1 (2 points)

Using the conventions in the "Boxes and Arrows" document, draw the box-and-arrow diagram that results from executing the following code segment.

```
Point p1 = new Point (1, 4);
Point p2 = new Point (2, 3);
p2.y = p1.x;
p1 = p2;
```

Problem 2 (2 points)

Students who used assertEquals to compare two Measurement objects in their JUnit test were surprised that it didn't work as they expected. For example, they defined a test method that constructed a measurement named m1 representing 4'2" and another named m2 representing 2'1", then made the following call;

```
assertEquals (m1, m2.multiple(2));
```

This assertion failed, even with a correctly coded multiple method. Explain why. (An outline of the Measurement class appears on a separate supplementary handout.)

Problem 3 (8 points)

Provide the following iteration methods for the Measurement class:

- public void initIterator (Measurement end)
  initializes an iterator for the sequence of measurements that starts at this measurement and ends at end, with consecutive elements of the sequence differing by 1 inch.
- public boolean hasNext ( )
  returns true if not all the measurements between start and end have been enumerated, and false otherwise.
- public Measurement next ( )
  returns the next measurement in the sequence between start and end.

Also provide declarations for whatever instance variables your iteration will need to record its state. The iteration should not affect any other Measurement objects being used.

You should assume that this measurement's value is no greater than end.

Suppose this measurement represents 1 foot, 3 inches. After a call to initIterator with an argument representing 1 foot, 5 inches, successive calls to next would return the Measurement objects representing

1. 1 foot, 3 inches,
2. 1 foot, 4 inches, and
3. 1 foot, 5 inches.

If initIterator's argument is the same as this measurement, initIterator initializes an iteration sequence of one measurement.

An outline of the Measurement class appears on a separate supplementary handout. Make no assumptions about the instance variables in this class. Put your own code on the next page.

Space for your answer to problem 3

```
public class Measurement {

    // Instance variables from the existing Measurement class would go here.
    // None, however, are named, so you're not allowed to use variables like myFeet.

    // Instance variable(s) for the iteration:









    // Constructors and methods plus, minus, multiple, toString, and equals go here.

    // Initialize an iteration of measurements starting at this measurement and ending
    // at end, with consecutive elements of the iteration differing by 1 inch.
    public void initIterator (Measurement end) {









    }

    // Return true if there are more measurements to be returned, false otherwise.
    public boolean hasNext ( ) {









    }

    // Return the next measurement in the iteration sequence.
    // Precondition: hasNext ( ).
    public Measurement next ( ) {









    }
}
```

Problem 4 (17 points)

Consider the addition of an append method to the IntSequence class. The append method takes one argument, another IntSequence object, and returns a new IntSequence that contains first the elements of this sequence, then the elements of the argument sequence. A call to append should leave both the component sequences unchanged.

For example, the code segment

```
IntSequence seq1 = new IntSequence (3);
seq1.add (1);
seq1.add (7);
IntSequence seq2 = new IntSequence (5);
seq2.add (4);
seq2.add (9);
IntSequence seq3 = seq1.append (s2);
System.out.println (seq3);
```

should print "1 7 4 9".

Part a

One might convert the above example to a JUnit test method named testTypical. A thorough tester would provide at least three other test methods. Describe—in English, not in Java, giving examples of sequences you would test if necessary—three more test methods that would provide additional evidence about the correctness of your append method.

1.

2.

3.

Part b

Give the Java code for one of the test methods you just described.

Part c

Provide the Java code for the append method. An outline of the IntSequence class appears on a separate supplementary handout. Make no assumptions about any methods that don't appear in the outline.

```
public IntSequence append (IntSequence seq) {
```