# MIDTERM EXAMINATION
# CSC 212 ♦ SPRING 2012

*You may use one 8.5"x11" sheet of notes on this exam. You may not consult other sources of information. You will have the entire lab period (110 minutes) to complete your work. All work should be written in the exam booklet. Partial credit will be granted where appropriate if intermediate steps are shown.*

## 1. **Vocabulary** (15 points)

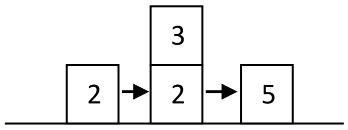
Give short (1-2 sentence) definitions for each of the terms below, in a way that explains how the items in each pair of terms differ from each other.

a). declaration vs. initialization
b). nested class vs. subclass
c). assignment vs. allocation
d). argument vs. parameter
e). class definition vs. class instance

## 2. **Stacks** (16 points)

Draw the states of the stack during processing of each of the postfix arithmetic expressions below. If an error occurs, draw the states up to the point of the error and explain what happens. The answer to the first one is shown for you as an example.

a). 2 3 +
b). 3 * 4
c). 9 7 5 – *
d). 9 7 – 5 *
e). 8



## 3. **Recursion** (14 points)

The program below uses recursion to find the minimal element of an array. Draw a diagram of the memory state for the program below at the point where execution reaches the CHECKPOINT line for the third time. (You do not need to show any stack frames for method invocations that have been completed, but if you do draw them please make sure they are clearly crossed out or erased.) Your program should show the call stack and any storage on the heap that is referenced by call stack variables.

```
/** Use recursion to find the minimum */
public class RecursiveMin {
    public static double findMin(double[] A, int n) {
        double result;
        if (n == 0) {
            result = Double.POSITIVE_INFINITY;
        } else {
            double minRemainder = findMin(A,n-1);
            if (A[n-1] < minRemainder) {
                result = A[n-1];
            } else {
                result = minRemainder;
            }
        }
        return result;  // CHECKPOINT
    }

    public static void main(String[] args) {
        double[] A = {6.0,1.0,9.0,3.0,4.0};
        System.out.println("Minimum element:  "+findMin(A,A.length));
    }
}
```

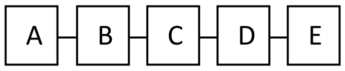
## 4. **Links** (16 points)

The program below creates several data structures in memory. Draw a diagram of the memory state for the program below at the point where execution reaches CHECKPOINT A, and a second diagram for the point where execution reaches CHECKPOINT B. Your program should show the call stack and any storage on the heap that is referenced by call stack variables.

```
public class Spaghetti {
    public Spaghetti A;
    public Spaghetti B;

    public static void main(String[] args) {
        Spaghetti f = new Spaghetti();
        Spaghetti g = new Spaghetti();
        Spaghetti h = new Spaghetti();

        f.A = f;
        f.B = g;
        g.A = h;
        g.B = f;
        h.A = g;
        h.B = g;
        // CHECKPOINT A

        g.A.B = f.B.B;
        f.A.A.A.A = g.B.B.B.B;
        h = g.B.A;
        // CHECKPOINT B
    }
}
```

## 5. **Lists** (12 points)

Suppose that at a given time, a LinkedList<String> named list is in the state shown. Predict the effect of each of the sequences of commands below and draw the resulting state of the list. (Assume that each subquestion begins with the state shown; they are not sequential.) If an exception will occur, give its name.



a). list.set(2,list.remove(2));

b). ListIterator<String> iter = list.listIterator();
iter.next(); iter.next(); iter.add("F"); iter.next(); iter.add("G");

c). ListIterator<String> iter = list.listIterator(list.size());
iter.previous(); iter. previous(); iter.remove(); iter. next(); iter.remove();

## 6. **Sorting** (12 points)

Each diagram below shows some snapshots of the state of memory for one of the sorting algorithms we have studied. (The states are shown in sequence with the earliest at the top, but not all states are shown.) Identify the algorithm used in each case.

a). Unsorted

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 7 | 2 | 4 | 8 | 9 | 1 | 6 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 2 | 4 | 8 | 9 | 1 | 6 |

| | | | | | |
|---|---|---|---|---|---|
| 2 | 4 | 8 | 9 | 1 | 6 |

etc.

Sorted

| |
|---|
| 5 |

| | |
|---|---|
| 3 | 5 |

| | | |
|---|---|---|
| 3 | 5 | 7 |

b). Array:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 7 | 2 | 4 | 8 | 9 | 1 | 6 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | 2 | 4 | 1 | 5 | 7 | 8 | 9 | 6 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 4 | 5 | 7 | 8 | 9 | 6 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 7 | 8 | 9 | 6 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

c). Unsorted

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 3 | 7 | 2 | 4 | 8 | 9 | 6 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 3 | 7 | 4 | 8 | 9 | 6 |

| | | | | | |
|---|---|---|---|---|---|
| 5 | 7 | 4 | 8 | 9 | 6 |

etc.

Sorted

| |
|---|
| 1 |

| | |
|---|---|
| 1 | 2 |

| | | |
|---|---|---|
| 1 | 2 | 3 |

## 7. **Program Analysis** (15 points)

Consider the three methods shown below. Analyze the running time of each in terms of **`n`**, where n is the number of elements in s, under the assumption that class **`List`** is (i) actually a **`LinkedList`**, or (ii) actually an **`ArrayList`**. Justify each analysis with an argument. If the method will generate an exception, state what it would be.

```
public <T> void pickRandomly1(List<T> s, List<T> t) {
    ListIterator<T> pos = s.listIterator();
    while (pos.hasNext()) {
        T item = pos.next();
        if (Math.random() > 0.5) {  // 50% chance
            t.add(item);
        }
    }
}

public <T> void pickRandomly2(List<T> s, List<T> t) {
    ListIterator<T> pos = s.listIterator();
    while (pos.hasNext()) {
        pos.next();
        if (Math.random() > 0.5) {  // 50% chance
            t.add(s.remove(pos.previousIndex()));
        }
    }
}

public <T> void pickRandomly3(List<T> s, List<T> t) {
    for (int i = 0; i < s.size(); i++) {
        int j = (int)Math.floor(Math.random()*s.size());  // random index
        t.add(s.get(j));
    }
}
```