MIDTERM EXAMINATION KEY
CSC 212 ♦ FALL 2015

You may use one single-sided 8.5"x11" sheet of notes on this exam. You may not consult other sources of information. You will have the entire 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.

Vocabulary (16 points)
Define all of the bolded terms below, making sure to explain the differences within each pair.

a.) **field** vs. **method**
a field is a class member that stores data, while a method is a class member that is an action

b.) **public** vs. **private**
public members are accessible from external code (other classes), while private members are not

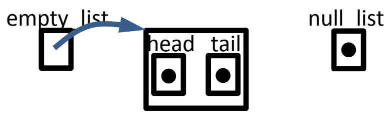
c.) **extends** (a parent class) vs. **implements** (an interface)
a class that extends another inherits all of its existing methods, while a class that implements an interface inherits nothing (it must define the set of methods described by the interface from scratch)

d.) **javadoc comment** vs. **block comment**
*a Javadoc comment appears before a class or class member, begins with /** and is used by the Javadoc tool. A block comment appears elsewhere and begins with /*.*

e.) **s1 == s2** vs. **s1.equals(s2)** (assuming s1 and s2 are strings)
the former tests whether s1 and s2 refer to the same object in memory, while the latter tests whether they contain the same sequence of characters

f.) **int** vs. **Integer**
an int is a primitive type storing an integer, while an Integer is a reference type comprising one field that stores an int.

g.) **empty LinkedList** vs. **null LinkedList** (draw a picture showing the difference)
an empty LinkedList refers to a structure that has a head and tail that are null, while a null LinkedList refers to nothing



h.) **array** vs. **ArrayList**
an array is a built-in type in Java, while an ArrayList is a class defined in the Collections framework. Both store sequences of elements that have the same type, although for ArrayList it must be a reference type. Arrays do not dynamically resize, but ArrayList does.

Data Structures (16 points)
Consider the Java class defined below, which implements a data structure we have studied. Answer the questions that follow.

```
import java.util.*;
public class DataStructure<E> {
    private Stack<E> A;
    private Stack<E> B;

    public DataStructure () {
       A = new Stack<E>();
       B = new Stack<E>();
    }

    private void to(int i) {
       for (int j = 0; j < i; j++) {
           B.push(A.pop());
       }
    }

    private void from(int i) {
       for (int j = 0; j < i; j++) {
           A.push(B.pop());
       }
    }

    public E get(int i) {
       E answer;
       to(i);
       answer = A.pop();
       A.push(answer);
       from(i);
       return answer;
    }

    public void set(E x, int i) {
       to(i);
       if (!A.isEmpty()) {
           A.pop();
       }
       A.push(x);
       from(i);
    }
}
```

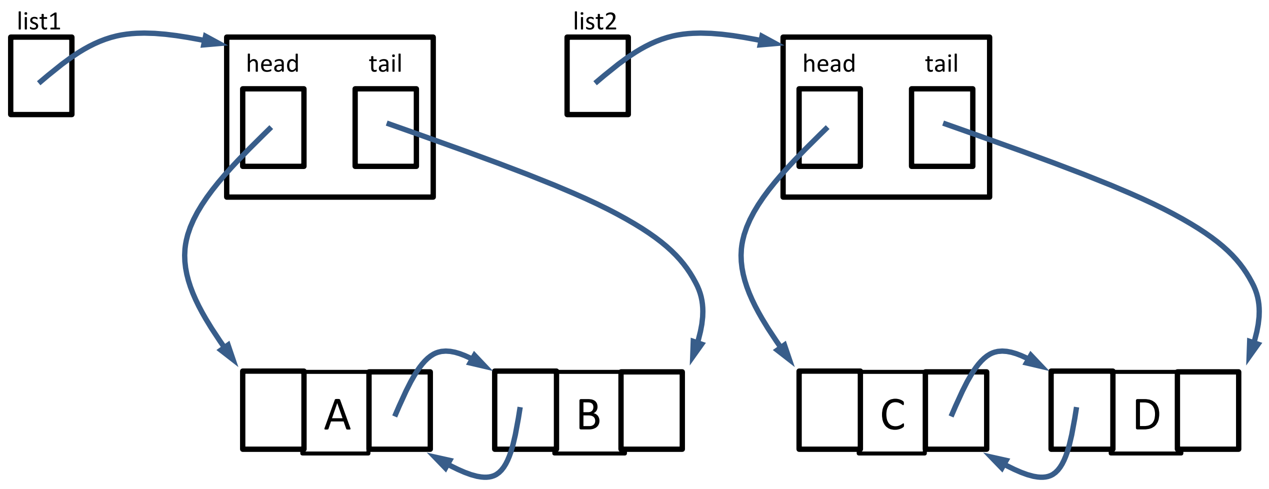
a.) Which methods of this class may be called by a different class? *Get and set*

b.) Which abstract data structure is this class implementing? *An array (get and set are key array methods)*

c.) How does this implementation compare to the usual implementation in efficiency? *It is less efficient, because it requires a loop to look up any element*

d.) Looking just at the public fields of this class, is there any way for a programmer to tell that it is implemented using stacks? Explain. *There is no way. We could replace the stack implementation with something else and nobody could tell.*

Linked Lists (16 points)
Consider the diagram below, showing two linked lists. Answer the questions that follow, assuming that each one begins with the configuration shown below. An example is done for you.



a.) [Example] Write code to update the links as necessary to remove **B** from **list1**.
list1.head.next = null; list1.tail = list1.head;

b.) Write code to make **list2** a shallow copy of **list1**. Draw a diagram showing the link structure after this operation.

list2.head = list1.head; list2.tail = list1.tail;

c.) Write code to swap the order of the elements in **list1**. Draw a diagram showing the link structure after this operation.

list1.head.previous = list1.tail; list1.tail.next = list1.head;
list1.head.next = list1.tail.previous = null;
list1.head = list1.tail; list1.tail = list1.head.next;

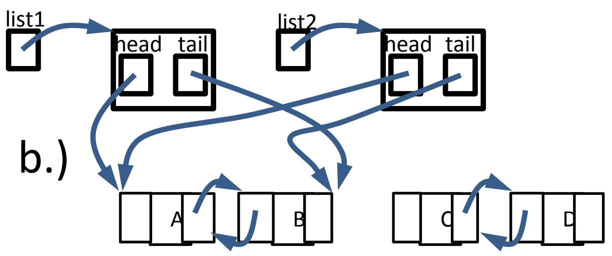
d.) Write code to update links as necessary to append **list2** at the end of **list1**, leaving **list2** empty. Draw a diagram showing the link structure after this operation.

list1.tail.next = list2.head; list2.head.previous = list1.tail; list1.tail = list2.tail;
list2.head = list2.tail = null;

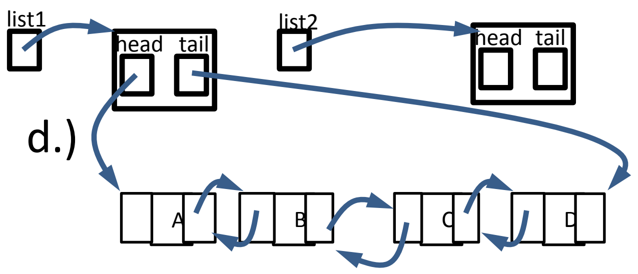
e.) Write code to update links as necessary to insert **list2** within **list1** between A and B, leaving **list2** as a shallow copy of this part of **list1**. Draw a diagram showing the link structure after this operation.

list1.head.next = list2.head; list1.tail.previous = list2.tail;
list2.head.previous = list1.head; list2.tail.next = list1.tail;

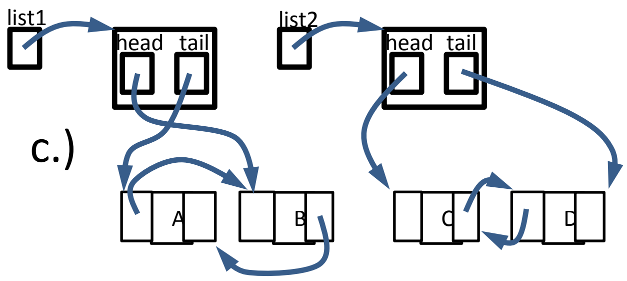
b.)



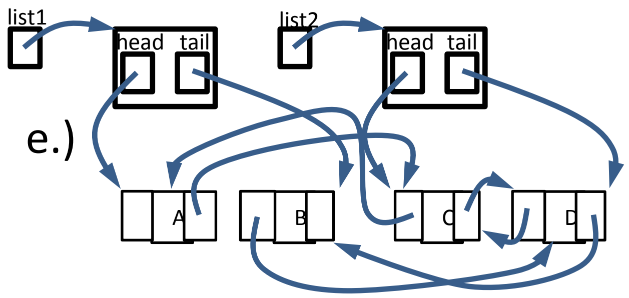
d.)



c.)



e.)



Stacks (16 points)
Consider the input stream ABCDEFGHIJKLM, where each letter is a token. (A is the first and M is the last.) Suppose that you want to write code that will convert this to a new stream in a different order, using a single stack. Using three operations, **push**, **pop**, and **pass**, write pseudocode that would transform the stream above into the new sequence DGFEHCJLKIBMA.

(Note: In this context, **push** takes an input token and adds it to the stack, **pass** takes an input token and sends it to output, and **pop** takes a token off the stack and sends it to output.)

Example: *If the input was ABC and the desired output was BCA, the answer is* ***push****;* ***pass****;* ***pass****;* ***pop****.*

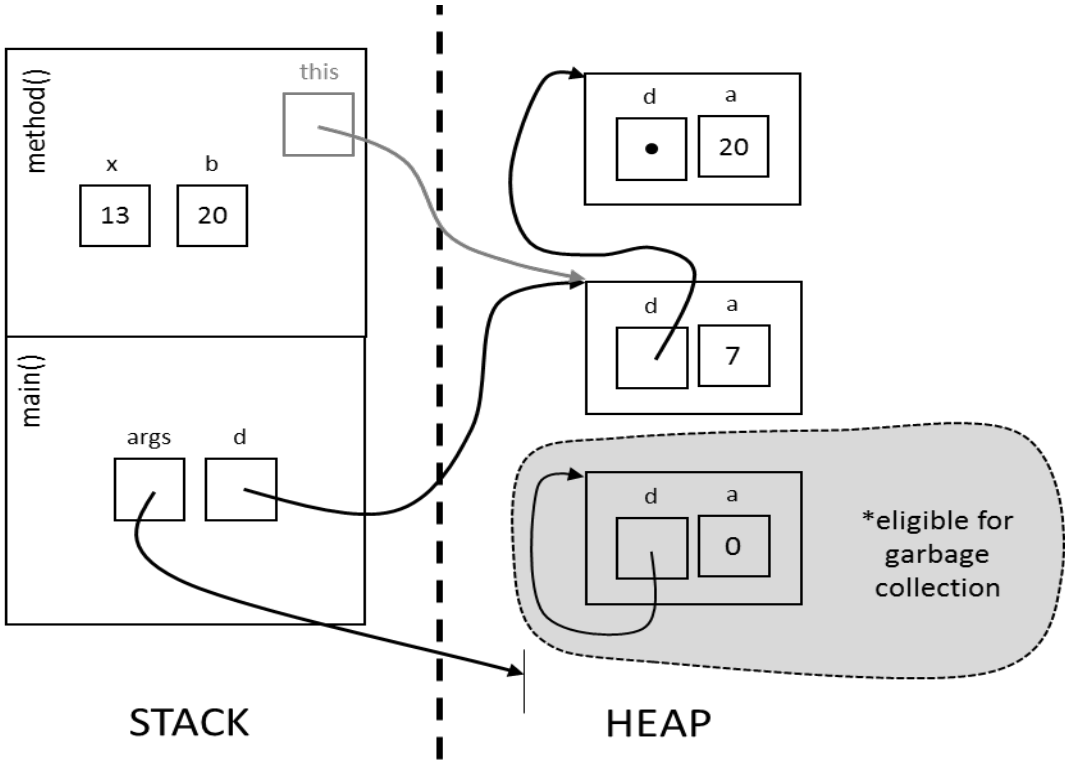
> *Push; push; push; pass; push; push; pass; pop; pop; pass; pop; push; pass; push; pass; pop; pop; pop; pass; pop;*
>
> *(Note that any pass may also be replaced with consecutive push & pop.)*

Class Design (8 points)
Briefly describe the Model-View-Controller paradigm, describing the duties of each role and how they interact with each other. Give an example from a homework or lab that fits into this paradigm, and identify for this example which class takes on each role.

Model-view-controller is used for interactive GUI programs. A model keeps track of the data related to a particular concept. The view is responsible for depicting the concept graphically, and maintaining any information related to its display characteristics. Finally, the controller coordinates the behavior of the model and the view in response to a user's actions. In the map homework, MapGrid is the model, MapViewer in the view, and Map Application is the controller.

Java Language (16 points)
Consider the program shown below. Draw a diagram showing the state of memory at the point where the statement labeled **NOW** has just been executed. Your diagram should follow the style of the ones drawn in class. It should show all local variables, with the regions corresponding to the call stack and the heap clearly denoted. The value currently stored in each variable should be shown. StAny memory that was previously allocated but is now eligible for garbage collection should be shown in a separate section of the heap, labeled **GC**. You need not show any static variables, and you may assume that the **args** parameter to **main** is an empty array.

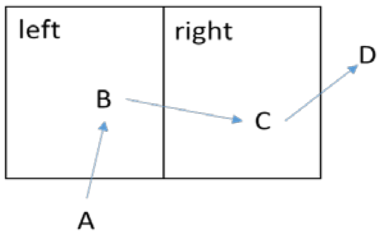
```
public class Diagram {
    private int a;
    public Diagram d;
    public Diagram() {
      a = 0;
      d = null;
    }
    public int method(int x) {
      int b = a+x;
      d = new Diagram();
      d.a = b;                  // NOW
      return b;
    }
    public static void main(String[] args) {
      Diagram d = new Diagram();
      d.a = 7;
      d.d = new Diagram();
      d.d.d = d.d;
      d.method(13);
    }
}
```



Graphical User Interfaces (16 points)
Suppose that a GUI interface window consists of two side-by-side **JButton** elements (called **left** and **right**) as shown in the illustration. Several listeners have been registered, as shown below:

```
right.addMouseListener(new MouseListener1());
left.addMouseMotionListener(new MouseMotionListener2());
left.addActionListener(new ActionListener3());
```

Suppose further that the mouse begins at the point labeled A. The user moves it to point B and clicks once, then moves it to point C, clicks again, and finally moves it to point D. List the sequence of event handler methods that will potentially be called, in the order they will be called. If a handler might be called more than once in a row, indicate the number of times if it is known.



mouseMoved (multiple times) , actionPerformed, mouseMoved (multiple times) , mouseEntered, mousePressed, mouseReleased, mouseClicked, mouseExited

Statistics:

	Vocab	DS	LL	Stack	Design	Java	GUI	Total
Mean	12.4	9.6	11.1	14.2	6.1	6.0	5.9	65.3
Median	13	9.5	12	16	6.5	6	6	67.5
Min	7	4	2	0	0	0	0	36
Max	16	16	16	16	8	16	12	85