MIDTERM EXAMINATION -- KEY
CSC 112 ♦ FALL 2009

YOU MAY USE ONE 8.5"x11" SHEET OF NOTES ON THIS EXAM.

YOU MAY NOT USE THE TEXTBOOK, A COMPUTER, OR ANY OTHER INFORMATION SOURCE BESIDES YOUR SINGLE PAGE OF NOTES.

YOU MUST COMPLETE AND TURN IN THE EXAM WITHIN THREE HOURS.

All work should be written in the exam booklet. Partial credit will be granted where appropriate if intermediate steps are shown. When you are done, please sign the statement below. Good luck!

On my honor, I certify that I have neither given nor received assistance on this exam, and I have completed it in accordance with the instructions above and the Smith Honor Code.

Signature: ____

1. **Program Comprehension** (16 points)

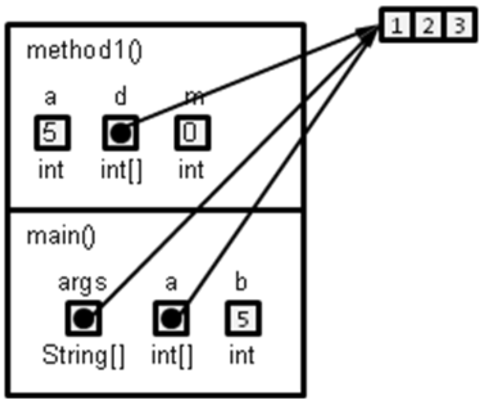
The diagram at right shows the state of data structures in the call stack and on the heap when the program has executed the point labeled A for the first time. Draw a similar diagram for the program at the point when it executes B.
Be sure to include all type and variable name annotations.

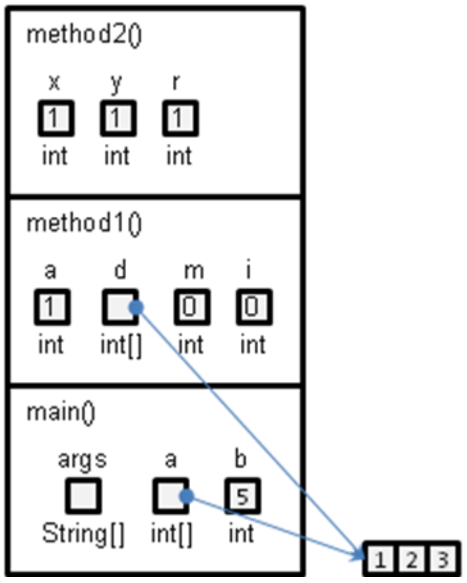
```
public class Diagram {
    public static int method2(int x, int y) {
        int r = 0;
        if (x != y) {
            r = 0;
        } else {
            r = 1;    // Point B
        }
        return r;
    }

    public static int method1(int[] d, int a) {
        int m = 0;    // Point A
        for (int i = 0; i < d.length; i++) {
            m = m+method2(d[i],a);
        }
        return m;
    }

    public static void main(String[] args) {
        int[] a = {1,2,3};
        int b = 5;

        System.out.println(method1(a,b));
        System.out.println(method1(a,a[0]));
    }
}
```





2. **Sorting** (16 points)

Consider the array of numbers below, which are to be sorted in increasing order from left to right. Simulate the array version of the algorithms specified, and show the state of the data after each swap performed.

4, 2, 6, 1, 5, 3

a.) Insertion sort, array implementation, growing the sorted region from left to right.

246153, 241653, 214653, 124653, 124563, 124536, 124356, 123456

b.) Merge sort, list implementation (show lists formed after each merge).

24, 16, 35, 1246, 123456

3. **Iterators** (10 points)

Write a simple loop using iterators to find the index of the minimum element of the `ArrayList` shown below. Methods of `ListIterator` you may wish to use: `next()`, `previous()`, `hasNext()`, `hasPrevious()`, `nextIndex()`, `previousIndex()`.

```
ArrayList<Float> list = new ArrayList<Float>();  // this is the list
// assume that elements are added to the list here
float minval = Float.MAX_FLOAT;  // this is the minimum value yet
int minind = -1;  // this is the index where it was found
for (ListIterator<Float> mark = list.listIterator();
    mark.hasNext(); ) {
    float x = mark.next();
    if (x <= minval) {
        minval = x;
        minind = mark.previousIndex();
    }
}
// at end, minind should contain the index of the smallest element
```

4. **GUI Programming** (8 points).

Which of the following are allowed/feasible when creating a GUI program?

a.) Register the same `ActionListener` object with more than one `JButton`. *Yes*

b.) Register several `ActionListener` objects with just one `JButton`. *Yes*

c.) Use the same event handler method for a button and a mouse click. *No*

d.) Register the same listener object for a button and a mouse click. *Yes*

e.) Use four instances of the same listener class to produce one of four different behaviors when each of four buttons are clicked. *Yes*

f.) Use one instance of the same listener class to produce one of four different behaviors when each of four buttons are clicked. *Yes*

g.) Use four instances of the same listener class to produce four different behaviors when one single button is clicked. *Yes*

h.) Register a non-subclass of `ActionListener` as a listener for a `Jbutton.` *No*

5. **Programming Style** (16 points)

"Effective program documentation requires a combination of programming techniques and disciplines." Discuss this statement, including at least three examples of programming techniques that can make your work easier to understand.

Many factors work together to make programs easier to read. Selecting simple algorithms and reducing redundant or unnecessary code, while not usually considered documentation, is one factor. The most obvious step you can take is to include Javadoc comments before each class, field, and method, with appropriate keywords included. Inline comments to clarify obscure commands or to identify sections of code can be helpful. Selection of descriptive names for variables and methods can be enormously helpful, making the code effectively self-documenting. The final act of documentation often involves creation of a user manual or other instruction set, separate from the program itself.

6. **Stacks and Queues** (12 points)

Suppose that you have an `Stack<Integer> s` with operations `push()` and `pop()`, and an `Queue<Integer> q` with operations `in()` and `out()`. Suppose further that at some moment in time the contents of the queue (listed from head to tail) are 1, 2, 3, 4, 5, and 6, while the stack is empty. Write a sequence of operations using no additional storage that would end in a configuration where the queue would contain 5, 6, 3, 4, 1, and 2 (again listed from head to tail). For example, s.push(q.out()) would move the 1 from the head of the queue to the top of the stack, and q.in(s.pop()) would move it back from the stack to the tail of the queue. Hint: slow and methodical is the ticket. Be careful!

s.push(q.out());s.push(q.out()); q.in(s.pop());q.in(s.pop());
s.push(q.out());s.push(q.out());q.in(s.pop());q.in(s.pop());
s.push(q.out());s.push(q.out());q.in(s.pop());q.in(s.pop());
s.push(q.out());s.push(q.out());s.push(q.out());
s.push(q.out());s.push(q.out());s.push(q.out());
q.in(s.pop());q.in(s.pop());q.in(s.pop());
q.in(s.pop());q.in(s.pop());q.in(s.pop());

7. **Data Structure Selection** (10 points)

Select an appropriate class from the Java core for the following tasks. Keep in mind efficiencies in both memory and execution time, where applicable.

a.) Represent integral (x,y) coordinates of a pixel *Point*

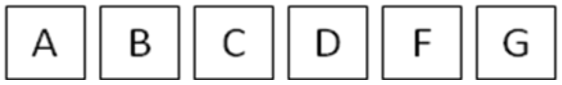
b.) Parent class for a user-defined object that will create a component visible in an applet window. *JComponent* c.) Data structure to hold password-checking tasks. The password checker works at a constant speed, but requests to check passwords may come in at any time, and must be handled in the order they are received. The number of concurrent requests is expected to be fairly small. *ArrayDeque*

d.) Data structure for a simulation of DNA mutation. The data structure will hold base pairs of DNA, subsequences of which may be deleted from one region and inserted in others. *LinkedList*

e.) Data structure to parse XML files. (XML has a nested tag structure similar to XHTML.) The files in question are expected to have a very deep nesting structure, meaning that the number of tags open may be very large. *LinkedList*

8. **Linked Lists** (12 points)

Suppose that a list has the contents shown below. After executing the code that follows, what would the list contents look like? Draw the state of the list after the lines marked R, S, and T in the code. Include the location of the mark in your diagram.



```
Character c;
ListIterator<Character> mark = list.listIterator(1);
c = mark.previous();
mark.set('E');  // Point R:  ^EBCDEF
mark.next();
mark.next();
c = mark.next();
mark.next();
mark.add('I');  // Point S:  EBCDI^EF
mark.next();
mark.remove();
mark.add(c);
list.removeLast();  // Point T:  EBCDIC (mark invalid)
```