MIDTERM EXAMINATION
CSC 212 ♦ Spring 2017

You may use one double-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 the concepts below, clearly explaining how each element of the pair is related or has something in common, as well as what is distinct or different.

a.) Declaration vs. assignment
Declaration introduces the name of a variable and specifies its type. Assignment gives a value to the variable. They may be combined in a single line, e.g., `int i = 0;`

b.) Allocation vs. initialization
Allocation means reserving a block of memory for an object or array. Typically this is in the context of creating a new object on the heap using **new**. *Initialization means setting up a new variable or object with its starting value. For an object this task is supposed to be carried out by the constructor, which automatically gets called when* **new** *is invoked.*

c.) **while** loop vs. **for** loop
Both types of loops can accomplish the same tasks. The `for` *loop bundles together initialization, continuation, and updates for the loop, whereas the* `while` *loop only specifies the continuation.*

d.) accessor vs. manipulator
Both typically relate to the fields of a class. An accessor returns the value of a field. A manipulator provides a mechanism to change the value.

e.) nested class vs. subclass
Both are classes somehow related to another class. A nested class is defined within an outer class, and is used to represent a concept that best makes sense within the context of the outer class. A subclass inherits fields and methods from its parent class, and is used to represent a refinement or specialization of the parent class concept.

f.) listener vs. event handler
Both are concepts related to GUI development. A listener is a class that implements one or more listener interfaces, and can be registered with a GUI element to be notified when events occur. An event handler is a method within the listener class that gets called when a registered event occurs.

g.) Array vs. list.
Both store sequences of values, all of the same type. An array stores the values contiguously in memory, while the list stores each element in an arbitrary location along with a link to the next element.

h.) Abstract data type vs. data structure implementation.
An abstract data type is the description of a data structure that is independent of any implementation; typically this includes the possible operations. An implementation is the realization of the abstract data type in software.

Sorting (16 points)

Consider the following unsorted list of numbers: 10, 4, 13, 7, 8, 2, 11, 7, 15, 9.
For each of the items below indicate which of the three main algorithms we studied might encounter the configuration shown in the process of sorting the list. If a particular configuration would not be produced by any of the three algorithms, write "none". (The three algorithms are insertion sort, selection sort, and merge sort.)

a.)	2, 4, 7, 7, 8, 10, 11, 13, 15, 9	*Insertion sort*
b.)	2, 7, 8, 11; 4, 7, 9, 10, 13, 15	*Merge sort*
c.)	4, 7, 8, 10, 13, 2, 7, 9, 11, 15	*None (this is from a backwards heap sort)*
d.)	2, 4, 7, 10, 13, 8, 11, 7, 15, 9	*Selection sort*

Stacks (12 points)

Consider a program that uses a stack as its internal data structure. In addition to **push()** and **pop()**, the system needs a capacity for **undoPop()** and **redoPop()**. That is, calling **undoPop()** after a **pop()** will put the just-popped item back on the stack, but calling **redoPop()** will take it back off. When a new item is pushed on the stack, all pop "history" is lost, and calling undoPop() will have no effect. Explain in words and/or pseudo-code how you could write a implement the undo/redo functionality using a single stack in addition to the main stack. Do not write Java code for this question.

Whenever an item is popped off of the main stack, it is immediately pushed onto the auxiliary stack. Calling redoPop() will pop whatever is on the auxiliary stack (checking first to see that it is not empty) and push it onto the main stack. Calling redoPop() will pop() an item from the main stack and push it onto the auxiliary stack (after checking a flag that indicates that a pop() has taken place previously). Calling push() on the main stack will empty the auxiliary stack.

Java Language (16 points)

Consider the code below. Simulate the execution of the program, and predict the output.

```
public class Refs {
    public int[] p;
    public int q;
    public Refs() {
        p = new int[1];
        p[0] = q = 0;
    }
    public static void m1(int[] p, int q) {
        Refs r = new Refs();
        r.p = p;
        q = 2;
        r.m2(r);
        System.out.println(r.p[0]);
        System.out.println(r.q);
        System.out.println(p[0]);
        System.out.println(q);
    }
    public void m2(Refs r) {
        System.out.println(r.p[0]);
        System.out.println(r.q);
        System.out.println(p[0]);
        System.out.println(q);
        p[0] = 9;
        r.q = 6;
        m3(r.p,q);
        System.out.println(r.p[0]);
        System.out.println(r.q);
        System.out.println(p[0]);
        System.out.println(q);
    }
    public void m3(int[] p, int q) {
        Refs r = new Refs();
        r.p[0] = p[0];
        q = r.q;
        r.q = 5;
        p[0] = 4;
    }
    public static void main(String args[]) {
        Refs r = new Refs();
        int[] p = {7};
        int q = 8;
        r.p[0] = p[0];
        r.q = q;
        m1(r.p,r.q);
        System.out.println(r.p[0]);
        System.out.println(r.q);
        System.out.println(p[0]);
        System.out.println(q);
    }
}
```

```
7
0
7
0
4
6
4
6
4
6
4
2
4
8
7
8
```

Bowling (32 points)
Bowling is a sport where competitors attempt to knock down a set of pins by rolling heavy balls at them. Matches are divided into "strings", where each string consists of ten "frames". For each frame, the bowler rolls two (or three, depending on the version) balls in succession gets one point for each toppled pin. They get a score (from 1 to 10) for each frame, and those scores a summed to get a total for the string. The class below is used to tally scores in bowling. (For the bowlers among you, we are ignoring "spares" and "strikes" for the time being.)

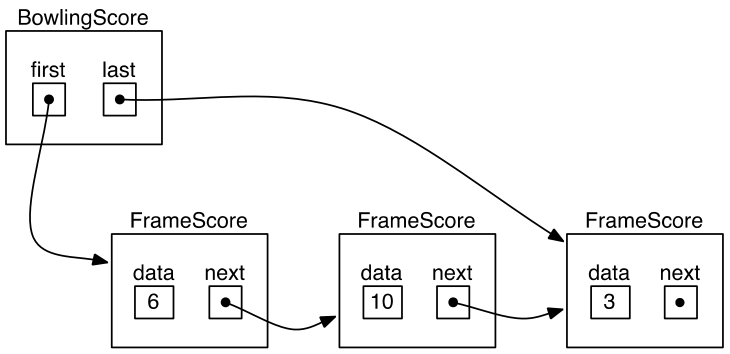
```
public class BowlingScore {
    FrameScore first;
    FrameScore last;

    public BowlingScore() {
        first = null;
        last = null;
    }

    public void addFrameScore(int score) {
        FrameScore fs = new FrameScore(score);

        if (first == null) {
            first = fs;
            last = fs;
        } else {
            last.next = fs;
            last = fs;
        }
    }

    public int getFrameScore(int idx) {
        FrameScore fs = first;
        for (int i=0; i<idx; i++) {
            fs = fs.next;
        }
        return fs.data;
    }

    public int total() {
        FrameScore fs = first;
        int tot = 0;
        while (fs.next != null) {
            tot += fs.data;
            fs = fs.next;
        }
        return tot;
    }

    private class FrameScore {
        int data;
        FrameScore next;
        public FrameScore(int data) {
            this.data = data;
        }
    }
}
```

a.) A bowler scores 6, 10, and 3 in her first three frames, and the following instructions are executed. What gets printed to the screen?

```
BowlingScore bs = new BowlingScore();
bs.addFrameScore(6);
bs.addFrameScore(10);
bs.addFrameScore(3);
System.out.println(bs.getFrameScore(0));
System.out.println(bs.getFrameScore(1));
System.out.println(bs.total());
```

6
10
19

b.) Draw a diagram of the BowlingScore object along with associated FrameScore objects in the manner we've done in class.



c.) The next instruction, shown below, causes a **`NullPointerException`** to be thrown. Explain why this happens, and identify the precise line of code where the exception occurs.

```
System.out.println(bs.getFrameScore(3));
```

The exception is thrown by the line "fs = fs.next;". During the third iteration, the fs variable gets set to null (since the previous FrameScore has a null pointer for next). When the next field of the null is accessed, the exception is thrown.

d.) Explain how you could modify the code above to gracefully handle the call above (that is, prevent the exception from being thrown).

There are several solutions:
- Checking to see if the current value of fs == last, and breaking out of the loop
- Surrounding the offending line in a try-catch block
- Storing the number of frames in a data field, or creating a method to count them, and comparing with argument

e.) The **getFrameScore()** function has *O(n)*, or linear, runtime complexity, where *n* is the number of frames bowled. Explain why. How might you modify class **`BowlingScore`** and/or **`getFrameScore()`** to run in *O(1)*, or constant time?

Each call to total must iterate along the linked list of FrameScore objects, and there are n of them. This could be avoided by keeping a running total in a data field, and adding to it when addFrameScore() is called, and returning it with total().

f.) Imagine that we rewrote the **`BowlingScore`** class using an array of type **`int`** for internal storage of scores. Think about (but do not write out) how you would implement the constructor, plus the methods **`addFrameScore()`**, **`getFrameScore()`**, and **`total()`**. What would be the performance differences between the two implementations for each method? (In other words, would the running time change in terms of its big-O notation, and if so, how?)

addFrameScore(int score) - faster, runs in O(1)

getFrameScore(int idx) - faster, runs in O(1)
total() - same, runs in O(n)

g.) A bowler gets three balls per frame in candlepin bowling. When a bowler knocks down all ten pins with their first two balls, it is called a "spare", and the score from the first ball of the next frame gets added into the frame with the spare (it is double-counted). How would you modify the code to accommodate this detail? Be specific.

Among other possibilities, one might modify addFrameScore() to accept three arguments, one representing each roll. In the case of a spare, the third argument is null, and a data field called "spare" is set to True. When addFrameScore() is next called, it first checks to see if the spare flag is set, and if so, it adds the score from its first ball into the data field of the tail.

h.) Write a method that takes an array of **BowlingScore** objects and concatenates them together. Assume scores **bs1** holding {2, 4, 6}, **bs2** holding {10, 9, 8, 7}, and **bs3** holding {1, 0}:

```
public BowlingScore concatScores(BowlingScore[] bsArr) {
    // FILL IN HERE
}

BowlingScore[] bsArr = new BowlingScore[3];
bsArr[0] = bs1;
bsArr[1] = bs2;
bsArr[2] = bs3;
BowlingScore bsAll = concatScores(bsArr);
// above should produce object holding { 2, 4, 6, 10, 9, 8, 7, 1, 0 }
```

Destructive:

```
public static BowlingScore concatScores(BowlingScore[] bsArr) {
    BowlingScore bsNew = new BowlingScore();
    bsNew.first = bsArr[0].first;
    bsNew.last = bsArr[0].last;
    for (int i=1; i<bsArr.length; i++) {
        bsNew.last.next = bsArr[i].first;
        bsNew.last = bsArr[i].last;
    }
    return bsNew;
}
```

Non-destructive:

```
public static BowlingScore concatScores2(BowlingScore[] bsArr) {
    BowlingScore bsNew = new BowlingScore();
    for (int i=0; i<bsArr.length; i++) {
        FrameScore fs = bsArr[i].first;
        while (fs != null) {
```

```
                  bsNew.addFrameScore(fs.data);
                  fs = fs.next;
              }
          }
          return bsNew;
      }
```

Graphical User Interfaces (8 points)

Describe in detail how user actions can trigger programmatic responses. What parts of the program are involved? What preparation does the program need to make before the interface will be responsive?

User actions are detected first by the operating system, and communicated to the Java Virtual Machine (JVM) which is the infrastructure responsible for running a Java program. Individual events may be registered with program elements like window components and timers. These objects keep a list of "listener" objects that must be notified whenever an appropriate event occurs. The listeners are activated one at a time in turn by calling an event handler method that corresponds to the event which has occurred. Although they originate with user actions causing an OS response, from the programmer's point of view, the objects that call event handlers may be thought of as generating the events for the rest of the program to handle.

The programmer who wishes to write an interactive program must write one or more listener classes (usually nested within the GUI manager class) that implement a listener interface and contain the appropriate event handler methods. The listeners must be registered with the objects that generate the events of interest. The event handlers in the listener classes must contain the code that responds to the event. Typically these responses will interact with the other parts of the program, such as the data models or views.