Practice Midterm 1

You have 2 hours to complete this exam. This exam is meant solely for practice and topics that are not in this exam may be covered while topics in it may not be covered. This exam is out of 100 points. Every line may have at most one statement (including closing brackets).

Problem	Points
1	12
2	5
3	6
4	7
5	8
6	12
Total	50

1 (Vitamin) C what's going on? (4 pts)

Step through the running of the following program. At certain point in the methods there are comments with letters. In the corresponding blanks for each letter, write the values of o1.x[0], o1.x[1], o2.x[0], and o2.x[1]. Assume that we start off with the constructor being called.

```
public class OJ{
    int[] x;
    OJ z;
    OJ(int x, int y){
        this.x = new int[2];
        this.x[0] = x;
        this.x[1] = y;
    }
}

public class Juice {
    public OJ o1;
    public static OJ o2;

    Juice() {
        o1 = new OJ(1, 2);
        o1.z = new OJ(5, 6);
        o2 = new OJ(3, 4);
        o2.z = new OJ(7, 8);
        pulpify();
        vitaminSeed();
        appleImposter();
    }

    public void pulpify() {
        o1.x[1] = o2.x[1]; //a
    }

    public void vitaminSeed() {
        o1.x[0] = o1.z.x[0]; //b
        o2.x[0] = o2.z.x[1]; //c
        o1.z = o2;
    }

    public void appleImposter() {
        o1.x[1] = o2.x[0];
        o2.x[0] = o1.x[1];
        o2.x[1] = o1.z.x[0]; //d
    }
}
```

a o1.x[0]_____, o1.x[1]_____, o2.x[0]_____, o2.x[1]_____

b o1.x[0]_____, o1.x[1]_____, o2.x[0]_____, o2.x[1]_____

c o1.x[0]_____, o1.x[1]_____, o2.x[0]_____, o2.x[1]_____

d o1.x[0]_____, o1.x[1]_____, o2.x[0]_____, o2.x[1]_____

2 Errrrr.....er: (5 pts)

Below we have a buggy class. Your job is to identify all errors. In the lines below write the line number for each line that has an error. For each error, write down "C" if it a compilation error and "R" if it is a runtime error. If a line relies on something that has errored out, you can assume that the prior error was fixed. Assume we start off by instantiating the class CorR. There are at most 10 errors total.

```
1  public class CorR {
2      public static final int[] arr = new int[10];
3      int i;
4      XD xd;
5      public class XD {
6          public int val;
7          XD(int x) {
8              val = x;
9          }
10     }
11     CorR() {
12         i = 5;
13         xd = new XD(i);
14         diggity();
15         dawg();
16         coolCat();
17     }
18
19     public static void diggity() {
20         arr[0] = 10;
21         xd.val = 0;
22         arr[2] = 15;
23     }
24
25     public void dawg() {
26         for (int i = 0; i < arr.length; i++) {
27             arr[i] = i * 2 + 1 - 10 + .5;
28         }
29         diggity();
30         new int[] temp = new Integer[10];
31         arr = temp;
32     }
33
34     public static void coolCat() {
35         i++;
36         dawg();
37         arr[2] = arr[1] + 5;
38         xd = new XD(10);
39         int[] temp = ((int[]) new double[10]);
40         temp = arr;}
41     }
42 }
```

(a) _____ (c) _____ (e) _____ (g) _____ (i) _____

(b) _____ (d) _____ (f) _____ (h) _____ (j) _____

3 Casts and Inheriting Broken Bones (12 pts)

Below is a set of classes. We will have a series of method calls. In the lines following each method call (one line per line that is displayed after the call), write what is printed, if anything at all. If there is a compilation or runtime error please say which one it is. You may assume that previous lines affect the following.

```
public class Garmr {
    int size;
    boolean hasfangs;
    String name;
    public Garmr() {
        size = 0;
        hasfangs = false;
        name = "Grimie";
        System.out.println("Woof");
    }
    public Garmr(int size, boolean fangs, String name) {
        this.size = size;
        this.hasfangs = fangs;
        this.name = name;
        System.out.println("Hoowl");
    }
    public void open() {
        System.out.println("there");
    }
    public void bite() {
        if (!hasfangs) {
            System.out.println("all bark");
        } else {
            System.out.println("snarl");
        }
    }
}

public class Fenrir extends Garmr {
    int anger;
    public Fenrir() {
        System.out.println("hungry");
        this.size = 100;
    }
    public Fenrir(int size, boolean fangs, String name, int fur) {
        super(fur, fangs, name);
        System.out.println("snarl");
        this.anger = anger;
    }
    public void howl() {
        System.out.println("howl");
    }
    public void howl(int loudness) {
        System.out.println(loudness + " OW");
    }
    public void bite() {
        System.out.println("yum!");
    }
}
```

(a) `Garmr smallBjorn = new Garmr();`

(b) `smallBjorn.bite();`

(c) `Garmr furt = new Fenrir();`

(d) `furt.howl();`

(e) `furt.bite();`

(f) `Fenrir golt = new Garmr();`

(g) `Fenrir nutty = (Fenrir) smallBjorn;`

(h) `Fenrir yuli = new Fenrir(5, true, "jimbo", 10);`

(i) `golt.bite();`

(j) `yuli.bite();`

(k) `((Fenrir) furt).howl(10);`

(l) `yuli.howl();`

4 It's always my de-Fault (5 pts)

Use the below interfaces to create a class that has the minimal amount of methods implements Viking and compiles.

```
public interface Norse {
    final static int burliness = 100;
    void breathe();
    void grunt();
}

public interface Viking extends Norse{
    final static int burliness = 300;
    boolean fly(boolean wings);
    default void grunt(String t){
        System.out.println(t + "ARRRRRRGH");
    }

 public class Bjorn implements Viking {

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------
 }
```

5 Osmosis (10 pts)

We want to add a method to IntList so that if 2 numbers in a row are the same, we add them together and make one large node. For example:
$1 \rightarrow 1 \rightarrow 2 \rightarrow 3$ becomes $2 \rightarrow 2 \rightarrow 3$ which becomes $4 \rightarrow 3$

```
public class IntList {
    public int first;
    public IntList rest;
    public IntList(int f, IntList r) {
        ...
    }

     public void addAdjacent() {

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------

        ------------------------------------------
    }
}
```

6 Teenage Mutant Ninja Hurdles (8 pts)

For each of the following subproblems follow the instructions

(a) General Colonel wants to make a method that both overloads and overwrites the method of a parent class. Is this possible? Explain your answer.

(b) When would you use a comparable over a comparator? Which one is preferable?

(c) True or False, an instance of a class has a broader scope than just the class. That is, from an instance, you can always call at least as the variables and methods as you can from just the class.

(d) Does overloading a method take into account the return value? Basically would changing public int hello(int hi)... to public boolean hello(int hi)... be valid? Explain why.

7 Riddle Me This (0 pts)

What is it that no person wants to have but no person wants to lose?

8 A Test Within a Test (12 pts)

Corn on the Cobb is a IntList Romer who wants to edit the IntList class so that it can help him roam safely. His idea is to write a method *dreaming* which is a method that adds a number to the end of an IntList. However, there is a catch. The IntList must have a size less than or equal to 3. If the size ever exceeds 3, the first element of the IntList must be removed and the second element should become the new start of the IntList. For example, say we have the following calls

```
1 a = new IntList();
2 a.dreaming(1); a.dreaming(2); a.dreaming(3);a.dreaming(4);
```

a.first would be 2, a.rest.first would be 3, a.rest.rest.first would be 4 equivalent to the IntList $2 \rightarrow 3 \rightarrow 4$. It is this IntList because 1 is removed when 4 is added since the IntList size would exceed 3.

(a) Complete the IntList class such that it fulfills the above requirements. Below is the IntList class for reference. A reminder that the *dreaming* method is void. (8 pts) abscissa

```
public class IntList {
    public int first;
    public IntList rest;
    public IntList(int f, IntList r) {
        ...
    }
    public int size() {
        ...
    }

    public void dreaming(int n) {
        if (this.size() == 0) {

            ------------------------------------;
        } else {
            if (----------------------------------) {

                ------------------------------------------------------

                ------------------------------------------------------
            }
            IntList p = ----------------------------------

            while (----------------------------------------) {

                ------------------------------------------
            }
            ------------------------------------------------
        }
    }
```

(b) Corn on the Cobb now wants to test this code to make sure that he does not lose his way in his dreams (that would really suck). Write a basic JUnit test to make sure that your code works as expected.
Note: The IntList.list(1, 2, 3,4, 5) would make an int list 1→ 2 → 3 → 4 → 5

```
public void testDreaming(){

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------

    ------------------------------------------
}
```

9 Compearisons and other fruits (20 pts)

Note, the below problem is fairly hard, it is recommended to make sure the prior problems are done properly before attempting this one.
You have a special BNode class. Each BNode class looks as follows

```
public class BNode {
    int sugar;
    BNode next;
}
```

You are now attempting to construct a BList. A BList is similar to an SList- specifically, it looks as follows:

```
public class BList {
    BNode first; //First node
    int sweetness; //The sum of the sugar of all BNodes in BList
    int size; //The number of BNodes in the BList
}
```

Your job is to complete the following methods in the BList class.

- *BList()*: The Constructor for a BList fill it as you see fit.
- *BList addtoBList(BNode elem, BList lst)*: Returns a BList with elem added to lst. If elem's sugar value is greater than lst's sweetness value then returned BList should have elem at the front– otherwise it should have elem at the end of the BList.
- *BList addBLists(BList a, BList b)*: Returns a BList with a and b added together. Use the following scheme for adding BLists:
  1. The BList whose sweetness value is larger should come before the BList with the smaller sweetness value.
     For example, say we had list a which has a sweetness value of 3 and list b which had a sweetness value of 4. The resulting BList would be $b \rightarrow a$
  2. If the sweetness values for the two BLists are the same then the one with the larger size comes first.
     - For example, say we had list a which has a sweetness value of 3 and size of 3 and list b which also had a sweetness value of 3 but had a size of 2. The resulting BList would be $a \rightarrow b$
  3. If the sizes are the same too then put a as the first list followed by b.
     - For example, say we had list a which has a sweetness value of 3 and size of 3 and list b which also had a sweetness value of 3 and a size of 3. The resulting BList would be $a \rightarrow b$
- *BList addToEnd(BList a, BList b)*: Takes in two BLists, a and b, and adds b to the end of a.
  Say list a is equal to $q \rightarrow f \rightarrow w \rightarrow e$ and list b is equal to $h \rightarrow i$. calling addToEnd(a,b) would result in the list $q \rightarrow f \rightarrow w \rightarrow e \rightarrow h \rightarrow i$

In addition to this, you will have the option to complete a *BListComparator* class (cross it out if you choose not to use it.)

Remember to update instance values when you add elements to BLists.

```
public class BList {
    BNode first; //First node
    int sweetness; //The sum of the sugar of all BNodes in BList
    int size; //The number of BNodes in the BList
     public BList(BNode a) {

        this.first = ____________________________________;

        this.sweetness = ________________________________;

        this.size = _____________________________________;
     }

     public BList addtoBList(BNode a, BList b) {

        ________________________________________________;

        ________________________________________________;
     }

     public BList addBLists(BList a, BList b) {

        ________________________________________________

        ________________________________________________

        ________________________________________________

        ________________________________________________

        ________________________________________________

        ________________________________________________
     }

     public BList addToEnd(BList a, BList b) {

        BNode head = ____________________________________;
        BNode curr = head;
        while (curr != null) {
            curr = curr.next;
        }
        curr.next = _____________________________________;

         ________________________________________________

         ________________________________________________

         ________________________________________________

     }
}
```

```
public class BListComparator implements Comparator<____________________> {

    public int compare(_____________________________________) {

        ________________________________________________;

        ________________________________________________;

        ________________________________________________;

        ________________________________________________;
    }
}
```

10 Arrrrghrays (25 pts)

Purplebeard and his lackey Turquoisenail are sailing the 10 seas. In order to sail well, they want to be able to create a map. They managed to create their **square** map, but Turquoisenail tripped and put it through a paper shredder. They managed to store the scrap images into a 1d array, but they need to piece it back into an NxN map. You are lucky because on each piece you have the longitude and latitude written down. Write a short program to help put the pieces back together. The pieces should be as follows:

0,20	10,20	20,20
0,10	10,10	20,10
0,0	10,0	20,0

In the upper left corner of the table, 0 is the longitude and 20 is the latitude.

For this problem you have access to only arrays

a For the first part of this problem, make a Piece class that store longitude and latitude.

```
public class Piece{

    ------------------------------------;

    ------------------------------------;

    ------------------------------------
    public Piece(int x, int y){

        ------------------------------------;

        ------------------------------------;

        ------------------------------------
    }
}
```

b The next part of this problem is take the Pieces in the given 1D Piece array, where Pieces are in no particular order, and put it into a 2D array where each row filled with Pieces that have the same latitude.

```
public Piece[][] groupByLat(Piece[] p){

    int width = (int) ____________________________________;

    Piece[][] latGroup = new Piece[____________][____________];

    for(int i = 0; i < ________________________________; i++){

        for(int j = 0; j < ________________________________; j++){

            if(latGroup[j][_________]==_________________________){

                ____________________________________________________;
                break;
            }
            else if(____________________________________________________){
                int counter;
                for(counter=0; _________________ < p.length-1; counter++){

                    if(____________________________________________){
                        break;
                    }
                }
                ________________________________= ____________________________;
                break;
            }
        }
    }
return latGroup;
}
```

c Our goal is to now to complete the process of taking in a 1D unsorted Piece array and transform it such that it becomes a sorted 2D array as shown on the first page of this problem (longitudes increase from left to right and latitudes increase from down to up). To complete this problem you have the following methods.

- *groupByLat(Piece[] p)*: From part b, takes in a 1D Piece array and converts it into a 2D Piece array where Pieces share a row if they have the same latitude.
- *sortbyLat(Piece[][] p)*: Takes in a 2d array of Pieces and returns it sorted in the correct order such that the row that contains the smallest latitudes is at the 0th index.
- *sortHalfLong(Piece[] p)*: Takes in a 1D array of Pieces and **half sorts** them all by longitude. In this problem, the term half sort means that the array is fully sorted except the first half of the sorted array is switched with the second half of the sorted array. For example:
  say we have an array $[9, 2, 4, 0]$. This array sorted would be $[0, 2, 4, 9]$. This array half sorted would be $[4, 9, 0, 2]$ since the first half of the sorted array, $[0, 2]$, would be swapped with the second half, $[4, 9]$.

```
public Piece[][] solvePuzzle(Piece[] scattered){

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------

    ------------------------------------------------
}
```