

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	4.5
2	7
3	10
4	7
5	8
6	12
Total	50

1 (Vitamin) C what's going on? (4.5 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];
    }

    public void vitaminSeed() {
        o1.x[0] = o1.z.x[0];

        o2.x[0] = o2.z.x[1];
        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];
    }
}
```

Solution:

a o1.x[0]_____1_____, o1.x[1]_____4_____, o2.x[0]_____3_____, o2.x[1]_____4_____

b o1.x[0]_____7_____, o1.x[1]_____4_____, o2.x[0]_____3_____, o2.x[1]_____4_____

c o1.x[0]_____7_____, o1.x[1]_____4_____, o2.x[0]_____8_____, o2.x[1]_____4_____

d o1.x[0]_____1_____, o1.x[1]_____8_____, o2.x[0]_____8_____, o2.x[1]_____8_____

Explanation:

2 Errrrr.....er: (7 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 is 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 }
```

Solution:

- | | | | | |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| (a) <u>line 21, C</u> | (c) <u>line 30, C</u> | (e) <u>line 35, C</u> | (g) <u>line 38, C</u> | (i) <u>line 40, C</u> |
| (b) <u>line 27, C</u> | (d) <u>line 31, C</u> | (f) <u>line 36, C</u> | (h) <u>line 39, C</u> | (j) _____ |

Explanation:

3 Casts and Inheriting Broken Bones (10 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

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" );
    }
}
```

Solution:

```
public class Bjorn implements Viking {
    public void breathe(){
        System.out.println("breathing noise");
    }
    public boolean fly(boolean wings){
        if(wings) {
            System.out.println("I am flying!!!");
            return true;
        }
        return false;
    }
    public void grunt(){
        System.out.println("arrggh");
    }
}
```

Explanation:

5 Osmosis

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) {
        /**
         * ...
         */
    }
}
```

Solution:

```
        p.rest = new IntList(x, null);
    }

    public void addAdjacent() {
        IntList p = this;
        if (p == null) {
            return;
        }
        IntList s = p;
        while (s.rest != null) {
            if (s.first == s.rest.first) {
                s.first = s.first * 2;
                s.rest = s.rest.rest;
                s = p;
            } else {
                s = s.rest;
            }
        }
    }
}
```

Explanation:

6 Teenage Mutant Ninja Hurdles

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.

Solution:

No

Explanation:

Making one method that both overloads and overwrites a method is impossible. For once overloading only refers to methods in the same class, when you have different arguments for a method with the same name. Overriding on the other hand is having a class that extends another class and has a method with the same name and same arguments.

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

Solution:

Neither is really better than the other, it just depends how you are using them. A comparable is used for having a “natural” order, basically you want the objects to always be compared in some fashion. A comparator is more of a “one-time” use case and just used to compare for a specific time.

Explanation:

See Above

- (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.

Solution:

True

Explanation:

An instance of a class has access to static and dynamic methods and variables.

- (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.

Solution:

You cannot change the return value because there would be no way to differentiate which method java should choose. How would it know which return value you are expecting?

Explanation:

An instance of a class has access to static and dynamic methods and variables.

7 Riddle Me This

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

Solution:

A Lawsuit

Explanation:

8 A Test Within a Test

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)

```
public class IntList {
    public int first;
    public IntList rest;

    public IntList(int f, IntList r) {
        /**
         * ...
         */
    }

    public int size() {
        /**
         * ...
         */
        int count = 0;
        IntList p = this;
        while (p != null) {
```

Solution:

```
    }  
}  
  
public void dreaming(int n) {  
    if (this.size() == 0) {  
        this.first = n;  
    } else {  
        if (this.size() == 3) {  
            this.first = this.rest.first;  
            this.rest = this.rest.rest;  
        }  
        IntList p = this;  
        while (p.rest != null) {  
            p = p.rest;  
        }  
    }  
}
```

Explanation:

- (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 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

Solution:

```
public void testDreaming(){
    IntList i = new IntList(0, null);
    i.dreaming(1);
    i.dreaming(2);
    IntList q = new IntList(0, null);
    q.addLast(1);
    q.addLast(2);
    for (int len = 0; len < q.size(); len++) {
        assertEquals(q.first, q.first);
    }
}
```

Explanation:

9 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

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

Solution:

```
public class Piece{
    public int longitude;
    public int latitude;
    public Piece(int x, int y){
        longitude = x;
        latitude = y;
    }
}
```

Explanation:

- 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.

Solution:

```
public Piece [][] groupByLat(Piece [] p){
    int width = (int) Math.sqrt(p.length);
    Piece [][] latSort = new Piece[width][width];
    for(int i = 0; i < p.length; i++){
        for(int j = 0; j < latSort.length; j++){
            if(latSort[j][0] == null){
                latSort[j][0] = p[i];
                break;
            }
            else if(latSort[j][0].latitude == p[i].latitude){
                int counter;
                for(counter=0; counter < width; counter++){
                    if(latSort[j][counter] == null){
                        break;
                    }
                }
                latSort[j][counter] = p[i];
                break;
            }
        }
    }
    return latSort;
}
```

Explanation:

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].

Solution:

```
public Piece [][] solvePuzzle(Piece [] scattered){
    Piece [][] grouped = groupByLat(scattered);
    int upper = (int) Math.ceil((double) grouped.length/2);
    int lower = (int) Math.floor((double) grouped.length/2);
    for(Piece [] row : grouped ){
        Piece [] halvesort = sortHalf(row);
        Piece [] tmp = new Piece[row.length];
        System.arraycopy(halvesort , 0,tmp,lower , upper );
        System.arraycopy(halvesort , upper ,tmp,0 , lower );
    }
    Piece [][] sorted = sortByLat(grouped);
    return sorted;
}
```

Explanation: