Practice Midterm 1

This test has 9 questions worth a total of 100 points, and is to be completed in 110 minutes. The exam is closed book, except that you are allowed to use one double sided written cheat sheet (front and back). No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. Write the statement out below in the blank provided and sign. You may do this before the exam begins.

#	Points	#	Points
1	4.5	6	20
2	7	7	8
3	10	8	0
4	7	9	10
5	8	10	25
		Total	100

1. (Vitamin) C what's going on?: (4.5 pts) Step through the running of the following program and at each blank 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]; _____
  }
}
```

2. Errrrr…..er: (7 pts) Find all the compilation errors and mark them with a "C". Find all runtime errors and mark them with a "R". Note, if a line relies on something that has errored out, you can assume that the prior error was fixed. Assume we start off by calling the constructor. For every error give a brief explanation why it errors out.

```
public class CorR {
  public static final int[] arr = new int[10];
  int i;
  XD xd;

  private class XD {
     private int val;

     XD(int x) {
        val = x;
     }
  }

  CorR() {
     i = 5;
     xd = new XD(i);
     diggity();
     dawg();
     coolCat();}

  private static void diggity() {
     arr[0] = 10;
     xd.val = 0;
     arr[2] = 15;}

  private void dawg() {
     for (int i = 0; i < arr.length; i++) {
        arr[i] = i * 2 + 1 - 10 + .5;
     }
     diggity();
     new int[] temp = new Integer[10];
     arr = temp;}

  static void coolCat() {
     i++;
     dawg();
     arr[2] = arr[1] + 5;
     xd = new XD(10);
     int[] temp = ((int[]) new double[10]);
     temp = arr;}}
```

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, write what is printed, if anything at all. If there is a compilation or runtime error please say which one it is and provide an explanation. You may assume that previous lines affect the following.

```
public class Container {
  int size; boolean haslid; String name;}
  public Container(){
     size = 0;
     haslid = false;
     name = "bad container";
     System.out.println("no constructor");}

  public Container(int size, boolean liddy, String name){
     this.size = size;
     this.haslid = liddy;
     this.name = name;
     System.out.println("here it is");}

  public void open() {
     System.out.println("there");}

  public void close() {
     if (!haslid) {
        System.out.println("Can't close what isn't there");}
    else{
          System.out.println("Closed");
}}}
public class NutellaJar extends Container {
            int sweetness;
            public NutellaJar() {
               System.out.println("I'm so hungry");
               this.sweetness = 100;}

            public NutellaJar(int size, boolean lid, String name, int sweetness) {
               super(size, lid, name);
               System.out.println("oink");
               this.sweetness = sweetness;}

            public void taste() {
               System.out.println("Just one more scoop");}

            public void taste(int scoops) {
               System.out.println("I just ate" + scoops + ". yum");}

            public void close() {
               System.out.println("This is too hard!");}}
```

Container plainjar = new Container(); _____ _____ _____

plainjar.close(); _____ _____ _____

Container tasty = new NutellaJar(); _____ _____ _____

tasty.taste(); _____ _____ _____

tasty.close(); _____ _____ _____

NutellaJar scrumptious = new Container(); _____ _____ _____

NutellaJar nutty = (NutellaJar) plainjar; _____ _____ _____

NutellaJar n = new NutellaJar(5, true, “sweet thang”, 10); _____ _____

scrumptious.close() _____ _____ _____

nutty.close(); _____ _____ _____

((NutellaJar) tasty).taste(10); _____ _____ _____

nutty.taste(); _____ _____ _____

4) It's always my de-Fault: (7 pts) Use the below interfaces to create a class that 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;
  void attack();
  boolean fly();
  void grunt();
  default void grunt(String t){
     System.out.println(t + "ARRRRRGH");
  }

}
```

5. Osmosis: (8 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 ⇒ 1 ⇒ 2 ==>3 becomes 2 ⇒ 2 ⇒ 3 which becomes 4 ⇒ 3.
For this problem, you will not have access to any add, size, or remove method.

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

  public IntList(int f, IntList r) {
     this.first = f;
     this.rest = r;
  }

  public void addAdjacent() {

    ____________________________________________________________

     if (_____________________) {

     ____________________________________________________________
     }
     IntList s = p;
     while (_____________________) {

      ____________________________________________________________

      ____________________________________________________________

      ____________________________________________________________

      ____________________________________________________________

      ____________________________________________________________

      ____________________________________________________________

      ____________________________________________________________

      ____________________________________________________________

      ____________________________________________________________
       }
     }
  }
}
```

6. Interfering Interfaces: (20 points) We want to write two classes, Sandwich and Pizza. Both of these classes will implement the interface ToppableFoods. A quality of all ToppableFoods is that you can add all the toppings you want that anything that extends the Ingredient class.

Two specific classes that extend the Ingredient class are Topping and Saucy.

The first item added to a sandwich must be a slice of bread and you can add anything up until you add a second slice of bread (sorry no triple deckers). For Pizzas' the first item added must be "Dough", any amount of ingredients may be added.

Regarding Sauces, Sandwiches can add any amount of sauces whenever you want. On the other hand, pizzas can have a maximum of 2 sauces added BEFORE toppings are added (you can also go sauceless like a savage).

Just like we can add toppings and sauces, we can also remove them. In both cases, only the most recently added item is allowed to be removed. Any attempt to remove any other item should result in an IllegalArgumentException. You can remove ingredients until there are no more left.

- Note for the purpose of this problem, Bread and Dough are considered Toppings.
- You may use a LinkedListDeque or ArrayList Deque
- You may find instanceOf useful
    - Implementation is as follows: <Object> instanceOf <Class name>
    - Returns a boolean value (true if it is an instance of that class, false otherwise).
- All methods that will be common to both Sandwich and Pizza should be in ToppableFoods
- You will have four pages total to write the Pizza class and the Sandwich class (2 for each). You may include any helper classes and instance variables.

Below we have implemented the Ingredient class, the Topping class, the Saucy class,an example implementation of a class that implements Topping, and the Deque interface.

```
public class Ingredient{
String name;
Ingredient(String
name){
this.name = name;}}
```

```
public class Saucy extends
Ingredient{
    Saucy(String name){
     super(name);
}}
public class Topping
extends Ingredient{
     Topping(String name){
     super(name);
}}
```

```
public class Bread
extends Topping{
  int grain;
   Bread(int grain){
   super("Bread");
   this.grain = grain;
}
```

```
interface
Deque<Item> {
void addFirst(Item x);
 void addLast(Item x);
boolean isEmpty();
int size();
void printDeque();
Item get(int index);
Item removeFirst();
Item removeLast();}
```

public interface ToppableFoods

public class Sandwich _____ {

public class Pizza _____ {

7. Troe & False: (8 pts)
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) An instance of a class has a broader scope than just the class. That is, calling a method or variable from a class may cause a compilation error, but an instance of it won't.

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.

8) Riddle me this (0 pts): What weighs six ounces, sits in a tree, and is very dangerous?

9) A Test Within a Test?: (10 pts)

a) Corn on the Cobb wants to is attempting to use an IntList to get through dreams. At any given time, the IntList's size must be a maximum of 5. If the IntList's size ever seems like it is going to exceed 5, you must remove the first node, making the second element the new first. For example adding 5 to the following list 0 ⇒ 1 ⇒ 2 ⇒ 3 ⇒ 4 would yield 1 ⇒ 2 ⇒ 3 ⇒ 4 ⇒ 5. Write dreaming which adds nodes and makes sure that the IntList fulfills the requirements that we put on it above. (8 pts)

```
public class IntList{
   public int first;
   public IntList rest;
   public int size(){...}
}
public void dreaming(int n){

        if(_____________________________________){

        }
        else{
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
              ____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
        }
}
```

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 (3 pts)

```
@Test
public testDreaming(){
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
              ____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
             _____________________________________________________________
        }
```

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 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 make it into a 10x10 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

-100, 30	-50, 30	-25, 30
-100, 20	-50, 20	-25, 20
-100, 10	-50, 10	-25, 110

In the upper left corner, -100 is the longitude and 30 is the latitude.

For this problem, you have access to IntLists, ArrayDeques, arrays, and LinkedListDeques

Note: This problem is very hard, and it is expected that you attempted earlier problems before looking at this one.

a) (2 pts) For the first part of this problem, make a Piece class that store longitude and latitude. (**2 pts)** public class Piece{

b) (15 pts) For the second part of this, make a method that takes in an array of all the pieces and returns a 2d array that is sorted by latitude. You may use both pages to write the problem.

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

c) (8 pts) The final part of this problem is to sort the latitude-sorted array by longitude as well. Assume that the passed in array from the part b works as expected.

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