UC Berkeley – Computer Science
CS61B: Data Structures
Midterm #1, Kartik

This test has 10 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.**

"I have neither given nor received any assistance in the taking of this exam."

Eh _____

_____

Signature:_____

#	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

Name: _____
SID: _____
Three-letter Login ID:____BBB_____
Login of Person to Left: ____LON_____
Login of Person to Right: ____ZOO_____
Exam Room:__BOWELS OF SODA_____

Tips:

- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- There are a lot of problems on this exam. **Work through the ones with which you are comfortable first. Do not get overly captivated by interesting design issues or complex corner cases you're not sure about.**
- Not all information provided in a problem may be useful.
- Unless otherwise stated, all given code on this exam should compile. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs in the exam, we'll announce a fix. Unless we specifically give you the option, the correct answer is not 'does not compile.'
- When the exam says "write only one statement per line", a for loop counts as one statement.

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];    o1.x[0] =  1, o1.x[1] = 4 , o2.x[0] = 3 , o2.x[1]= 4
  }

  public void vitaminSeed() {
     o1.x[0] = o1.z.x[0]; o1.x[0] =  7, o1.x[1] = 4 , o2.x[0] = 3 , o2.x[1]= 4

     o2.x[0] = o2.z.x[1]; o1.x[0] =  7, o1.x[1] = 4 , o2.x[0] = 8 , o2.x[1]= 4
     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]; o1.x[0] =  1, o1.x[1] = 8 , o2.x[0] = 8 , o2.x[1]= 8
  }
}
```

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; Compilation Error, cannot reference instance variable from static method
     arr[2] = 15;}

   private void dawg() {
     for (int i = 0; i < arr.length; i++) {
        arr[i] = i * 2 + 1 - 10 + .5;Compilation Error, cannot put a double in an int array
     }
     }
     diggity();
    new int[] temp = new Integer[10]; Compilation Error, cannot create an Integer Array when
the static  type is an int array
     arr = temp;} Compilation Error, cannot reassign the address of the final array

  static void coolCat() {
   i++; Compilation Error, cannot reference a reference variable from a static method
     dawg();
   arr[2] = arr[1] + 5;
     xd = new XD(10);Compilation Error, cannot instantiate within static methods.
```

int[] temp = ((int[]) new double[10]); Compilation Error, cannot cast a double to an int
temp = arr;}}
There was not a single runtime error on this problem

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(); no constructor _____ _____

plainjar.close(); Can't close what isn't there _____ _____

Container tasty = new NutellaJar(); no constructor I'm so hungry _____

tasty.taste(); Compilation error _____ _____

tasty.close(); This is too hard! _____ _____

NutellaJar scrumptious = new Container(); Compilation Error _____ _____

NutellaJar nutty = (NutellaJar) plainjar; Runtime Error _____ _____

NutellaJar n = new NutellaJar(5, true, "sweet thang", 10); Here it is oink _____

scrumptious.close() Compilation Error _____ _____

nutty.close(); Runtime Error _____ _____

((NutellaJar) tasty).taste(10); I just ate 10 scoops. yum _____ _____

nutty.taste(); Runtime Error _____ _____

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");
   }

}
```

```
public class Thor implements Viking {
   public void breathe(){
      System.out.println("breathing noise");
   }
   public void attack(){
      System.out.println("fight on with my " + burliness + " men");
   }
   public boolean fly(){
      System.out.println("I am flying!!!");
      return true;
   }
   public void grunt(){
      System.out.println("arrggh");
   }
}
```

*This problem was not too difficult, the important thing to realize is that when implementing Viking, you must use the methods from Norse (specifically breathe since grunt() was overridden.

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

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

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

          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; //addAdjacent() is also allowed
               break; //This line is used if addAdjacent was used
             } else {
                s = s.rest;
             }
          }
        }
}
```

The 1st error that is easy to make is checking to see if "this" is null. This can never be null in java, so we will have a pointer towards this. Since we did not allow access to size, you had to do this (non elegant) check. Now getting on to our while loop. The problem was easy to enough if you did not have to go backwards. If we initially started with the list 2 ⇒ 1 ⇒ 1 and you only went through 1 iteration of the while loop, you would end up with 2 ⇒ 2. This is not allowed under our conditions so you would set s = p, and start the process from scratch, as soon as you find 2 elements are the found to be the same . One could have done a recursive call to addAdjacent in its place, either method works.

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. Once you remove an item, you should return it.

- 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 two pages to write the Pizza class and the Sandwich class. 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 void addIngredient(Ingredient t);
  public Ingredient removeIngredient(Ingredient t);
}
public class Sandwich implements ToppableFoods{
  LinkedListDeque<Ingredient> ig;
  int amountofbread;

  public Sandwich() {
    this.ig = new LinkedListDeque<Ingredient>();
    this.amountofbread = 0;
  }

  public void addIngredient(Ingredient t) {
    if (ig.size() == 0 && t.name != "Bread") {
      throw new IllegalArgumentException("Need to have bread first?");
    }
    if (amountofbread >= 2) {
      throw new IllegalArgumentException("No more toppings allowed, you already have 2
breads");
    }
    if (t.name == "Bread") {
      amountofbread += 1;
    }
    ig.addLast(t);
  }

  public Ingredient removeIngredient(Ingredient t) {
    if (ig.size() == 0) {
      throw new IllegalArgumentException("Can't remove nothing");
    }

    if (ig.get(ig.size() - 1).name != t.name) {
      throw new IllegalArgumentException("This is not the most recently added element");
    }
    if (t.name == "Bread") {
      amountofbread -= 1;
    }
    return ig.removeLast();
  }
}
```

```
public class Pizza implements ToppableFoods {
  LinkedListDeque<Ingredient> ig;
  int numberOfSauces;
  boolean topping;

  public Pizza() {
    this.ig = new LinkedListDeque<Ingredient>();
    this.numberOfSauces = 0;
    this.topping = false;
  }

  public void addIngredient(Ingredient t) {
    if (ig.size() == 0 && t.name != "Dough") {
      throw new IllegalArgumentException("Can’t add to nothing");
    }
    if (t instanceof Saucy && numberOfSauces >= 2) {
      throw new IllegalArgumentException("Too many sauces");
    }
    if (t instanceof Saucy && (ig.get(ig.size() - 1) instanceof Topping) && (ig.get(ig.size() -
1).name != "Dough")) {
      throw new IllegalArgumentException("Can't put a sauce after a topping");
    }

    if (t instanceof Saucy) {
      numberOfSauces++;
    }
    ig.addLast(t);
  }

  public Ingredient removeIngredient(Ingredient t) {
    if (ig.size() == 0) {
      throw new IllegalArgumentException("There’s nothing to remove");
    }
    if (t.name != ig.get(ig.size() - 1).name) {
      throw new IllegalArgumentException("This is not the most recently added element");
    }
    if (t instanceof Saucy) {
      numberOfSauces -= 1;
    }
    return ig.removeLast();
  }
}
```

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.

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?

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.

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.

True, an instance of a class has access to static and dynamic methods and variables. The only time this

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.

No, this is not valid. 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?

8. Riddle me this (0 pts). What is it that no person wants to have but no person wants to lose?

A lawsuit

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. 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 void dreaming(int n){

        if(this.size == 0){
                this = new IntList(n, null);
        }
        else{
                if this.size() == 5{
                        this.first = this.first.rest;
                        this.rest = this.first.rest;
                }
                Intlist p = this;
                while(this.rest!=null){
                        p = p.rest;
                        }
                p.rest = new IntList(n, null);
                }
        }
```

b) (3 pts) 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

```
@Test
public testDreaming(){
        IntList tst = new IntList();
        tst.list(2,3,4,5);
        Intlist second = new IntList();
        second.dreaming(1);
        second.dreaming(2);
        second.dreaming(3);
        second.dreaming(4);
        second.dreaming(5);
        assertEquals(5, second.size());
```

```
        assertEquals(tst, second);}
```

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 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:

-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.

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

b) 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. **(15 pts)**

```
public Piece[][] sortByLat(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 = 0;
        while(counter + 1 < p.length && latSort[j][counter] != null){
          counter++;
        }
        latSort[j][counter] = p[i];
        break;
      }
    }
  }
  latSort = sortLatitudes(latSort);
  return latSort;
}

private Piece[][] sortLatitudes(Piece[][] unsorted){
  Piece[][] sorted = new Piece[unsorted.length][unsorted.length];
  int count = 0;
  while(count < unsorted.length){
    int maximum = Integer.MIN_VALUE;
    int maxindex =0;
    for(int i = 0; i < unsorted.length; i++){
      if(unsorted[i] != null && unsorted[i][0].latitude > maximum){
        maximum = unsorted[i][0].latitude;
        maxindex = i;
      }
    }
    sorted[count] = unsorted[maxindex];
    unsorted[maxindex] = null;
    count++;
  }
  return sorted;
}
```

This Solution is pretty complicated. We started by finding all the elements that have the same latitude and putting them into “buckets”. Then we sort the buckets themselves by latitude.

c) 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. **(8 pts)**

```
public Piece[][] sortFully(Piece[][] p) {
  for (int i = 0; i < p.length; i++) {
     Piece temp;
     for (int j = 1; j < p.length; j++) {
        for (int k = j; k > 0; k--) {
           if (p[i][k].longitude < p[i][k - 1].longitude) {
              temp = p[i][k];
              p[i][k] = p[i][k - 1];
              p[i][k - 1] = temp;
           }
        }
     }
  }
  return p;
}
```

Note: The solution in part C uses a type of insertion sort that will be looked over later on in the course.