Homework 4

You can use java's built-in lists. You may not use their hash tables, heaps, priority queues, etc.. Using these constitutes cheating and results in an F in the course. If in doubt, ask on the forum.

Permissions

1. You are allowed to add any methods or properties you want.
2. You are not allowed to remove/not implement the ones given in the class diagrams.
3. **<u>You may not modify test cases</u>**. Seriously. Come on, stop it.
4. You may not modify package names. If you did, you'd never pass the tests.
5. You may import code from one of your questions to another.
6. You may not have two source code files anywhere in your submission that have the same name.

Rules

1. Submit a zip (not rar, stuffit, etc.) file containing the java source code you wrote (i.e., if you give us your copy of the tests, we will erase them).
2. **<u>If your code breaks the grading script, you will get a 0 for the assignment</u>**. Concretely, do not submit code that doesn't compile or has an infinite loop. Either don't submit those files or make them compile (just return null or -1).

---

1. **Heap Sort** (25 points)

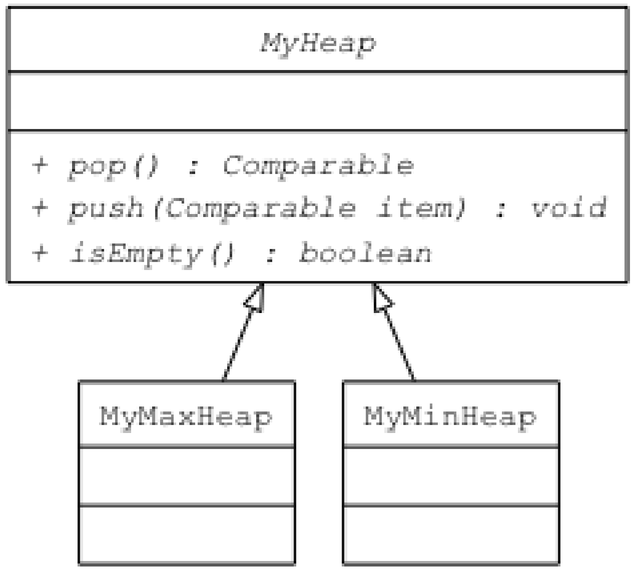
You have a bunch of stuff. You want to sort it. There are lots of ways to do that but here's a fast and easy one – use a heap. i'll leave it to you to figure out how.

You can either write your own heap or use java's built-in heap / priority queue. One of those choices will allow you to solve this problem in five minutes. The other won't have you expelled from the class for cheating. If you still don't know which is which, use the built-in one and see what happens.

You could also use java's built-in sorting methods. If you don't know whether or not this will cause you to fail the class, i will explain it when you take this class again next semester.

```
                     HeapSorter

+ sortAscending(List<Comparable> objectsToSort) :
     List<Comparable>
+ sortDescending(List<Comparable> objectsToSort) :
     List<Comparable>
```



As a reminder, the diagram says that MyMinHeap and MyMaxHeap are subclasses of the abstract parent class MyHeap. i will trust your ability to Google what that means if you don't remember from previous homework and classes.

Both MyHeap and MyHeapSorter work on objects implementing java's built-in Comparable interface. It has been pointed out that i baby you too much so rather than tell you how Comparable works, i will trust your ability to Google this.

2. **Tree Analyzer** (25 points)

Heaps are a *nearly complete binary tree* (that's a technical term). There are many other types of trees. For this problem, we'll give you a tree shoved into an array and you'll tell us interesting things about it.

```
TreeAnalyzer
--------------------------------------------------------

--------------------------------------------------------
+ isAMaxHeap(Comparable[] flatTree) : boolean
+ isAMaxSemiHeap(Comparable[] flatTree) : boolean
+ isAMinHeap(Comparable[] flatTree) : boolean
+ isAMinSemiHeap(Comparable[] flatTree) : boolean
+ isComplete(Object[] flatTree) : boolean
+ isFull(Object[] flatTree) : boolean
+ isPerfect(Object[] flatTree) : boolean
```

Each method answers a question. We covered what each technical terms means in class so i won't insult you by repeating them here.

3. My Hash Table (25 points)

Because i know someone will ask, no, you cannot use java's built-in hash maps.

```
MyHashTable<K,V>
--------------------------------------------------------------
+ buckets : List<KeyValuePair<K, V>>[]
--------------------------------------------------------------
+ MyHashTable(int initialCapacity, float loadFactor) : ctor
+ contains(K key) : boolean
+ get(K key) : V
+ put(K key, V value) : void
+ getCapacity() : int
+ getIndexOfBucket(K key) : int
+ getKeys() : List<K>
+ getKeyValuePairs() : List<KeyValuePair<K, V>>
+ getSize() : int
+ getValues() : List<V>
+ remove(K key) : void
```

As mentioned in class, i want you to do this with chaining rather than probing.

Notes:

- Hash codes can be negative.
- When you are over loaded, you need to increase capacity (number of buckets, not bucket size).
- When increasing capacity, it should increase to the next prime number.
- I gave you some code to start with.
- It's important you understand what `buckets` is. If you aren't sure, take it apart, piece-by-piece, inner to outer.
- Refer to notes for more info on Hash Maps, then ask the TAs if you have any further questions

4. Pyramid Scheme (25 points)

For reasons i don't truly understand, multi-level marketing companies are unusually popular in Minnesota. In an MLM, you convince your friends to buy things from you (cosmetics, jewelry, cleaning supplies, kitchen supplies, sex toys, underwear, food, etc.). If you convince a friend to sell things to their friends, you get a percentage of their sales (after all, you're sort of like her boss). If they convince a friend to sell things, they get a percentage of that friend's sales, and so on. The guy at the top of the pyramid can make a lot of money.

How much money? That's what you're going to figure out. Using MyHashTable from the previous question, figure out how much each person makes if they get not only their own sales but 10% of the income of the person they recruited.

```
                 MLMRevenueTracker
-----------------------------------------------------

-----------------------------------------------------
+ getNetIncome(String salesmanName,
     MyHashTable<String, Integer> sales,
     MyHashTable<String,String> sponsors) : int
+ getRevenue(String salesmanName,
     MyHashTable<String, Integer> sales,
     MyHashTable<String,String> sponsors) : int
-----------------------------------------------------

```

What you need to know:

- Revenue is how much money someone made. This is the combination of the person's sales plus downline revenue of 10% of the revenue of each person they recruited.
  - All sales and revenue are integers representing cents. When getting decimal or fractional values for revenues, round down to the nearest cent.
- Net Income is how much they have left after paying bills. Your bills are ½ the cost of the goods you sold (i.e., you sold them at 100% markup) plus 10% of your revenue to the person who recruited you. When getting decimal or fractional values for net income, round down to the nearest cent.
- For sales, key=salesman name, value=revenue from product sales.
- For sponsors, key=salesman name, value=name of person who recruited him.
  - For top-level salesman, key = salesmanName, value = “”.
- The first bullet point should make this clear but just in case it isn't super obvious, you must know the total revenue (sales+downline) of a person before you calculate the 10% they owe you.