CIT-590 Final Exam

Name:______________________________________________________

Penn Key (Not ID number): ________________

If you write a number above, you will lose 1 point

Instructions:

● You will have two hours to complete this exam. If you finish in the last 15 minutes, you must remain seated until time is called so as not to disturb your classmates.

● Do not rip off ANY PAGE, except for the last page which you may use as scratch paper. This page WILL NOT be graded.

● There is a second scratch paper that can be graded, and is marked "DO NOT RIP OFF". You may write on this page, but indicate in the exam to "see scratch page"

● Do not write on the back of pages. Any writing on the backs of pages will not be graded.

● If your cell phone goes on during the exam, you will lose 5 points. Make sure it's off.

● Do not discuss any portion of this exam at any time, including into the future indefinitely. No portion of this exam may be posted to websites like course hero. Your grade can be changed retroactively if this is found to have been done.

● All code written should be in Java syntax unless otherwise stated.
Section 0: Short answer

Answer each question in 1 sentence or less. If you write more than one sentence, you will lose a point.

0.1) What interface does an object need to implement to be used in a TreeSet?

0.2) If a parent class has no zero-argument constructor, what is the NAME of the first method the child must call?

0.3) If a parent class has an abstract method, what does this mean for the child?

0.4) What is the significance of the @Before tag in a JUnit Test?

0.5) What is the default implementation of the equals(Object o) method?

0.6) What does the word collision mean with respect to hashing?
Section 1: Short Coding

Write each of the following functions in the space provided.

1.1) Write a function that takes in a List of Integers and returns the sum:

1.2) Write a function that takes in a String and returns its reverse. (Hint: use the charAt(int index) function)
Section 2) Processing

Section 3) HashSets

Given the following (familiar looking) class, answer the following questions about HashSets. Draw what the hashtable looks like after each step.

```java
public class PezDispenser {
    private String name;
    private int capacity, count;

    public PezDispenser(String name, int capacity) {
        this.name = name;
        this.capacity = capacity;
        count = 0;
    }

    . . . //code here is irrelevant to this problem

    public int hashCode() {
        return capacity;
    }
}
```

When filling in the hashtable, simply put the NAME field in place. Example, new PezDispenser("Yoshi", 12) would be written as "Yoshi" in the table. **Resolve collisions using Clustering** (not Linked Lists). Assume the hash table is size 5. Assume it will not expand.

1) table.add(new PezDispenser("Cat", 3)  
2) table.add(new PezDispenser("Dog", 4)

3) table.add(new PezDispenser("Yoshi", 12)  
4) table.add(new PezDispenser("Superman", 12)

What is the load factor on this hash table?

Section 4) Simple abstract class example
Section 5) File I/O and HashMaps

Consider the following object below:

```java
public class VotingTotals {
    public Map<String,Integer> votes;

    public VotingTotals() {
        votes = new HashMap<String,Integer>();
    }

    public static Map<String,Integer> parseFile(String filename) {
        //TODO: You will implement this method
    }

    public void addVotes(Map<String,Integer> newVotes) {
        //TODO: You will implement this method
    }
}
```

In this class, you are keeping track of voting totals for candidates. The `votes` attribute is a HashMap matching candidate names to a number of votes (i.e. Steve - 14, Doug 5, etc.). This call will allow to combine voting files from several districts. You will implement two methods:

First, you will implement `parseFile`. This method takes in a file as formatted below:

```
Doug,14
Steve,25
Cathy,4
```

This method takes this file and generates a hashmap such that the key "Doug" gives the number 14, the key "Steve" gives the number 25, etc. **Assume the file is well formatted with nothing ever being wrong or weird. Don't waste time doing error checking.** This method is static, meaning it does NOT affect any instance of the method. This is simply taking a file and generating a HashMap. The input file is formatted as several lines of [Name, vote#], separated by a comma.

Second, you will implement `addVotes`. In this method, you will take in a Map, formatted the same as the result of `parseFile`, and this method will add those votes to the `VotingTotals` instance. Specifically, the votes are added to the `votes` attribute in the instance. There are two possible cases:

1) The candidate is already a key in `votes`. At which point, add the votes in the input map to that candidate in `votes`
2) The candidate isn't in `votes`. This means add the key to `votes` and set the starting number of votes based on the input map.

Example:
Let's say `votes` contains: [Bob-5] and [Jane-7]
If I call `addVotes` with a map like [Bob-6] [Jane-3] [Steve-2], then this should alter `votes`
The end result is `votes` now contains: [Bob-11], [Jane-10], and [Steve-2]

Given this specification, first implement `parseFile`. I have gotten you started:
On this page, implement **addVotes**. I have gotten you started:
Cheat Sheet

**HashMap<K,V> methods:**
- V get(K key): Get the value stored at key
- void put(K key, V value): puts the value V into the hashmap with the key "key".
- boolean containsKey(K key): returns true if the key is in the hashmap. Returns false if the key is not in the hashmap

**BufferedReader methods:**
- **Constructor**: takes in a FileReader object.
- String readLine(): returns the next line in the file as a String. Returns a null String if you are at the end of the file.

**FileReader methods:**
- **Constructor**: Takes in a filename as a String