Name:____ (PRINT CLEARLY)

Final Exam

CS 2102, B-term 2019

Question 1:	________	(30 points)
Question 2:	________	(30 points)
Question 3:	________	(40 points)
TOTAL:	________	(100 points)

1. (30 points, 5 points each)

Read all code carefully! All code is valid and will compile correctly.

For each of the following programs, write what will be displayed to the console when the program is run. For ease of grading, please write your answers on the lines here rather than below your code (although you are welcome to write below your code and then copy your answers here). **Answers not on these lines will not be graded!**:

1a. ____

1b. ____

1c. ____

1d. ____

1e. ____

1f. ____

1a.
```
public class Main {
     public static void main(String[] args) {
          String trek = "To" + " " + "Boldly" + " " + "Go";
          System.out.println(trek.length());
     }
}
```

1b.
```
import java.util.LinkedList;

public class Main {
     public static void main(String[] args) {

          LinkedList<Integer> nums = new LinkedList<Integer>();

          nums.add(2);
          nums.add(8);
          nums.add(-6);
          nums.add(4);

          int maxVal = 0;

          for (int i = 0; i < nums.size(); i++)
          {
               int thisNum = nums.get(i);

               if(maxVal < thisNum)
                    maxVal = thisNum;
          }

          System.out.println(maxVal);
     }
}
```

1c.
```
abstract class Animal {

     private int age;

     public Animal(int age) {
          this.age = age;
     }

     public String getAge() {
          return "age is " + this.age;
     }
}

class Mammal extends Animal {

     private String name;

     public Mammal(String name, int age) {
          super(age);
          this.name = name;
     }

     public String getAge() {
          return "Name is " + name + " and " + super.getAge();
     }
}

class Dog extends Mammal {

     private String breed;

     public Dog(String name, int age, String breed) {
          super(name, age);
          this.breed = breed;
     }

     public String getInfo() {
          return this.getAge();
     }
}

public class Main {
     public static void main(String[] args) {
          Dog myPet = new Dog("Susie", 3, "Corgi");
          System.out.println(myPet.getInfo());
     }

}
```

1d.

```
class MyException extends Exception {

     private String msg;

     public MyException(String msg) {
          this.msg = msg;
     }

     public MyException() {
          this.msg = "Default message";
     }

     public String getMsg() {
          return this.msg;
     }
}

class MyOtherException extends MyException {
     public MyOtherException() {
          super("Error condition!");
     }
}

class MyClass {

     public void performOperation() {
          try {
               doThing("A message");
          }
          catch (MyOtherException e) {
               System.out.println(e.getMsg());
          }
     }

     public void doThing(String something) throws MyOtherException {
          throw new MyOtherException();
     }
}

public class Main {
     public static void main(String[] args) {
          MyClass mc = new MyClass();
          mc.performOperation();
     }
}
```

1e.

```
import java.util.ArrayList;

class NoSuchElementException extends Exception {

}

class QueueAL<T> {

     private ArrayList<T> queue;

     public QueueAL() {
          this.queue = new ArrayList<T>();
     }

     public void add(T t) {
          this.queue.add(t);
     }

     public T remove() throws NoSuchElementException {
          if(this.queue.size() <= 0)
               throw new NoSuchElementException();
          else {
               T removeElement = this.queue.remove(0);
               return removeElement;
          }
     }
}


public class Main {
     public static void main(String[] args) {

          QueueAL<String> heroes = new QueueAL<String>();

          heroes.add("Wonder Woman");
          heroes.add("Iron Man");
          heroes.add("Captain Marvel");

          try {
               System.out.println(heroes.remove());
          }
          catch (NoSuchElementException e) {
               System.out.println("Exception caught!");
          }

     }
}
```

1f.

```java
class HelloWorld {
     private String hello;

     public HelloWorld(String hello) {
          this.hello = hello;
     }

     public boolean equals(HelloWorld other) {
          if(this.hello.equals(other.hello + "1234"))
               return true;
          else
               return false;
     }
}

public class Main {
     public static void main(String[] args) {

          HelloWorld hw1 = new HelloWorld("Hello World");
          HelloWorld hw2 = new HelloWorld("Hello World");

          if (hw1.equals(hw2))
               System.out.println("The two are equal!!!");
          else
               System.out.println("No equality!");
     }
}
```

2. Short answer (30 points, 5 points each)

2a. The Big-O of looking for a value in an AVL tree is O(log n). Briefly explain why.

2b. Examine the following code:

```
Heap h = new Heap();
h.addElt(1);
h.addElt(3);
h.addElt(6);
h.removeMinElt();
h.add(2);
h.add(4);
h.removeMinElt();
h.add(5);
h.add(1);
```

Draw the heap `h` after the above code is executed.

2c. We have a poorly-designed hash map where all of the stored values are in one bucket (meaning they're all in one LinkedList). Explain why this undermines the advantages of a hash map.

2d. Examine the BST below, and recall that a tree is just an acyclic (and, in this case, undirected) graph. Let's say we start at node 18 and want to determine if a path exists to node 19. List the order in which our method will visit the nodes of the tree, assuming a) we are using breadth-first search (BFS), b) our method visits the left subtree before it visits the right subtree, and c) our method terminates when we reach node 19.

```
      18
    /    \
  16      21
 /  \    /  \
15  17 19   23
           /
          22
```

2e. Below is a visualization of a hash map with six buckets (memory spaces). What is the load factor of this hash map? Explain how you know. (You may assume that parentheses indicate a LinkedList.)

(23, 46)
(25, 402)
(28, 39)
(16, 89)
(90, 321)
(110, 40)

2f. The following code throws an `IndexOutOfBoundsException`. Explain why.

```
import java.util.LinkedList;

public class Main {
      public static void main(String[] args) {

            LinkedList<String> names = new LinkedList<String>();

            names.add("Garfield");
            names.add("Odie");
            names.add("Jon");

            for (int i = 0; i < names.size(); i++)
            {
                  System.out.println(
                        names.get(i) + " and " + names.get(i + 1));
            }
      }
}
```

3. (40 points) Let's return to the grocery store problem from the midterm. The grocery store wants to start keeping track of each item that's removed. The store's software engineer tackles the problem by writing the following code, but it has some problems that you're going to fix:

```
 class NotFreshRecordHashMap {

       HashMap<String, GroceryItem> notFreshMap;

       public NotFreshRecordHashMap() {
             notFreshMap = new HashMap<String, GroceryItem>();
       }

       void addGroceryItem(String name, GroceryItem item)
       {



             notFreshMap.put(name, item);
       }

       GroceryItem getGroceryItem(String name) {

             if(!notFreshMap.containsKey(name)) return null;


             return notFreshMap.get(name);
       }
}

class ItemNotFoundException extends Exception {
       String err;

       public ItemNotFoundException(String err) {
             this.err = err;
       }

       String getErr() {
             return this.err;
       }
}
```

For questions a - c, you do not need to rewrite all of the code. Simply mark up the above code to indicate your changes. You may ignore import statements:

a. Modify the above code so that it uses the appropriate access modifiers.

b. Modify `getGroceryItem` so that instead of returning null if the item is not in the HashMap, it implements exception handling using `ItemNotFoundException`. You do not need to modify any other methods for this step.

c. Modify the above code to handle collisions. Use `getGroceryItem` in other methods if you need to get something from your hash map to modify. If `getGroceryItem` throws an exception, respond by instantiating something.

d. Now that we've fixed `NotFreshRecordHashMap`, the software engineer also wants to write a different version of this class called `NotFreshRecordSet`. However, the part of the program that calls these classes should not know whether the data is being stored in a hash map or a set. Explain in one sentence how we can solve this problem with an interface, and write the code for this interface. Use the version of the `NotFreshRecordHashMap` class after your modifications in parts a – c as your starting point. You may assume that each location in both the hash map and the set stores values in the same way, and you do not have to worry about import statements.