CS2852 Exam 1 Name:

No note-sheets, calculators, or other study aids on this exam. Write your initials at the top of each page except this one. Read through the whole exam before you get started.

Have fun!

-

1. [10 points total] Consider the code

   ```
   public void displayNames(Collection<String> c) {
      c.add("Bobby");
      System.out.println("First name: "+c.get(0));
      System.out.println("All names:");
      for(String name : c) {
         System.out.println("  name: "+c);
      }
   }
   ```

   a. (5 points) ***Circle above*** the ***one*** line in this program fails to compile when placed into IntelliJ.

   b. (5 points) ***Describe*** why this compile error might be a ***good*** thing. If you are at a loss, describe why this compile error occurs. (And maybe something will come to mind.)

2. [15 points total] Consider the code below, which has a very different error than the one in Problem 1.

   ```
    public void displayFirstName(List list) {
        String name = (String)(list.get(0));
        System.out.println("First name: "+name);
    }
   ```

   a. (5 points) Re-write the program to use generics:

   ```
   public void displayFirstName(

       String name =

       System.out.println("First name: "+
   }
   ```

   b. (5 points) ***Select*** whether the original program will compile or not. ***Explain*** your answer. (You may assume the necessary imports are included.)

   c. (5 points) ***Complete*** a method that calls the original displayFirstName and crashes it.

```
public void crashDisplayFirstName() {
    List list = new ArrayList();




     displayFirstName(list);
}
```

3. (5 points) ***Write*** the Big-O runtime of the method below. ***Show*** your work.
```
int count = 0;
for(int i = 0; i<n; i++) {
    if(Math.random() < 0.5) {
        for(int j = 0; j<n*2; j++) {
           count ++;
        }
    } else {
        if(Math.random() < 0.5) {
           count++;
        }
    }
}
System.out.println("count: "+count);
```
4. (10 points) ***Write*** the Big-O runtime for the following methods
    a. In Java's ArrayList:
        i. add(E)
        ii. indexOf(Object)
        iii. contains(Object)
        iv. add(0,E)
    b. In our ArrayList:
        i. add(E)
        ii. get(int)
        iii. remove(Object)
        iv. set(int, E)
        v. size()
5. (10 points) ***Write*** the Big-O runtime for the following methods
    a. In Java's LinkedList:
        i. it.next() (where Iterator<String> it = iterator())
        ii. it.hasNext() (where Iterator<String> it = iterator())
        iii. size()
        iv. set(0, E)
        v. get(n/2)
    b. In our LinkedList:
        i. remove(Object)
        ii. iterator()
        iii. clear()
        iv. indexOf(Object)

6. (25 points) ***Implement*** the `remove(Object)` method of the `ArrayList` that we implemented in class. (This `ArrayList` has only one instance variable – `E[] array`.) ***You may call indexOf***, but do not call any other methods of the ArrayList class.

remove

```
public boolean remove(Object o)
```

Removes the first occurrence of the specified element from this list, if it is present. If the list does not contain the element, it is unchanged. More formally, removes the element with the lowest index `i` such that `(o==null ? get(i)==null : o.equals(get(i)))` (if such an element exists). Returns `true` if this list contained the specified element (or equivalently, if this list changed as a result of the call).

Parameters:

`o - element to be removed from this list, if present`

Returns:

`true if this list contained the specified element`

7. (15 points) Implement the `add(E)` method of the `LinkedList` class that we implemented in class.

add

```
public boolean add(E e)
```
Appends the specified element to the end of this list.

This method is equivalent to `addLast(E)`.

Parameters:

```
      e - element to be appended to this list
```
Returns:

```
      true (as specified by Collection.add(E))
```

```
/** The node class, as we started it in class */
public class LinkedList<E> implements List<E> {
    private class Node {
        private Node next;
        private E value;
        private Node(Node next, E value) {/* … */}
    }
    private Node head = null;
// TODO: Implement add(E)







//… more of the class..
}
```

8. In this problem, you may draw memory-maps of linked lists if it helps your discussion.
   a. (5 points) ***Explain*** why, for our LinkedList, add(E) is O(n), but for Java's it is O(1).

   b. (5 points) ***Write*** the Big-O runtime of add(0,E) for our LinkedList. ***Explain*** your answer. (This is add(int, E), but only for an index of zero.)