Instructions: Use the space provided for each answer. If necessary, use the back of the page. You must complete this exam independently, with no outside resources of any kind. This exam consists of 5 questions, worth a total of 100 points.

Below, we provide the List interface, should you need it for reference:

interface List<E>:

boolean add(E e); // add to the end of the list
guid add(int index, E element); // insert at specified index
guid clear(); // remove all elements
boolean contains(Object o); // true if list contains the object
guid get(int index); // get element at specified index
guid index0f(Object o); // index of first occurrence (or -1)
boolean isEmpty(); // list is empty (size() == 0)
guid lastIndex0f(Object o); // index of last occurrence (or -1)
E remove(int index); // remove at specified index
boolean remove(Object o); // remove first occurrence of object
E set(int index, E element); // set item at index to be given element
int size(); // number of elements in list

And the Comparable<T> interface:

interface Comparable<T>:

int compareTo(T o);

The official description of compareTo is: “Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.”
1. (15 points) Suppose the following class has been defined:

```java
public class Person
{
    public String name;
    public Person valentine;

    public Person(name, valentine) {
        this.name = name;
        this.valentine = valentine;
    }

    public String toString() {
        if (valentine == null) {
            return name;
        } else {
            return name + " -> " + valentine.name;
        }
    }
}
```

What does the following code print out when run? (To save time, you may abbreviate names by their first initial. For example, you may write “D” in place of “Dakota.”)

```java
Person p1 = new Person("Dakota", null);
Person p2 = new Person("Jessie", p1);
Person p3 = new Person("Quinn", p2);
Person p4 = new Person("Casey", p3);

p3.valentine.valentine.valentine = p3;
p2.valentine = p3.valentine;
p1.valentine.valentine = p4;
p4.valentine.valentine = p2;

System.out.println("1. " + p1);
System.out.println("2. " + p2);
System.out.println("3. " + p3);
System.out.println("4. " + p4);
```

(Names are intentionally gender-neutral.)

**ANSWER:**

1. Dakota -> Quinn  
2. Jessie -> Jessie  
3. Quinn -> Jessie  
4. Casey -> Quinn
2. (15 points) **Opposites Attract!** What does the following code print out for each input? (Recall that the String representation of a list is comma-separated and surrounded by brackets, as in the specified inputs below.)

```java
public static void mystery(List<Integer> l)
{
    for (int i = 0; i < l.size(); i++) {
        int ival = l.get(i);
        for (int j = i + 1; j < l.size(); j++) {
            int jval = l.get(j);
            if (ival == -jval) {
                l.remove(j);
                l.add(i, jval);
                break;
            }
        }
    }
    System.out.println(l);
}
```

```
[1,-2,2]  
[1,2,-1,-2]  
[0,2,-2,-2,0]  
```

**ANSWERS:**

```
[1, -2, 2]  
[-1, 1, -2, 2]  
[0, 0, -2, -2, 2]  
```
3. (20 points) Use the following class definitions to answer the questions on the next page. (You may tear out this page if helpful.)

```java
public class Flower {
    public void color() {
        System.out.println("White");
    }

    public void smell() {
        color();
        System.out.println("Sweet");
    }
}

public class Violet extends Flower {
    public void color() {
        System.out.println("Blue");
    }

    public void smell() {
        super.smell();
        System.out.println("Light");
    }
}

public class Rose extends Flower {
    public void smell() {
        super.smell();
        System.out.println("Strong");
    }
}

public class YellowRose extends Rose {
    public void color() {
        System.out.println("Yellow");
        super.color();
    }

    public void smell() {
        super.smell();
        System.out.println("Citrusy");
    }
}
```
Suppose the following variables are defined:

```java
Flower f1 = new Flower();
Flower f2 = new Violet();
Flower f3 = new Rose();
Flower f4 = new YellowRose();
Object o1 = new YellowRose();
```

Indicate on each line below the output produced by each statement shown. If the statement produces more than one line of output indicate the line breaks with slashes as in a/b/c to indicate three lines of output with a followed by b followed by c. If the statement causes an error (either at compile time or runtime), write the word “error” to indicate this.

```java
f1.color();  __White___________________________________________

f1.smell(); __White/Sweet_____________________________________

f2.color(); __Blue____________________________________________

f2.smell(); __Blue/Sweet/Light________________________________

f3.color(); __White___________________________________________

f3.smell(); __White/Sweet/Strong______________________________

f4.color(); __Yellow/White____________________________________

f4.smell(); __Yellow/White/Sweet/Strong/Citrusy_______________

((Rose) o1).color(); __Yellow/White_________________________________

((Violet) o1).smell(); __error_____________________________________
```

(Problem instructions by Reges and Stepp.)
4. (30 points) Valentine’s Day isn’t just about hearts and flowers. February 14th is also the day of an annual Eugene Zombie Walk (yes, really).

Consider the Human class, which has the following public methods (and no public or protected member variables):

- Human(String name)
  Constructs a Human with the specified name.
- String getName()
  Returns the Human’s name.

Your task is to implement the Zombie class, which is a special kind of undead Human that has a name and an amount of hunger (initially zero). In addition to the methods inherited from Human, your class must include the following new public methods:

- Zombie(String name)
  Constructs a Zombie with the specified name.
- void walk()
  Increase hunger by 1 and print out: "<name> wants <hunger> brains!", where <name> is the zombie’s given name and <hunger> is its current hunger level.
- void feed(int numBrains)
  Decrease the zombie’s hunger by the specified number of brains. If hunger becomes negative, set it to zero.

You must also implement the Comparable<Zombie> interface, so that zombies are sorted first by hunger (in order of increasing hunger) and second by name (alphabetically). (You are encouraged to use the String method compareTo(String other) in your implementation.)

Example class usage:

```
Zombie z = new Zombie("Yorick");
z.walk();
z.walk();
z.walk();
z.feed(1);
z.walk();
z.feed(100);
z.walk();
```

Example output:

```
Yorick wants 1 brains!
Yorick wants 2 brains!
Yorick wants 3 brains!
Yorick wants 3 brains!
Yorick wants 1 brains!
```

(If you prefer vegetarian zombies, you may replace all occurrences of the word “brains” with “grains” in your solution.)
public class Zombie extends Human implements Comparable<Zombie> {
    private int hunger;

    public Zombie(String name) {
        super(name);
    }

    public void walk() {
        hunger++;
        System.out.println(getName() + " wants " + hunger + " brains!");
    }

    public void eatBrains(int numBrains) {
        hunger -= numBrains;
        if (hunger < 0) {
            hunger = 0;
        }
    }

    public int compareTo(Zombie other) {
        if (hunger != other.hunger) {
            return hunger - other.hunger;
        } else {
            return getName().compareTo(other.getName());
        }
    }
}
5. (20 points) Add the method `rotateRight()` to `LinkedIntList`, which moves the node at the end of the list to the front of the list. For example, if the `LinkedIntList` contains `[1, 2, 3, 4]`, then after calling `rotateRight()`, it should contain `[4, 1, 2, 3]`. If the list has fewer than two nodes, do nothing. **You may not call any other methods of LinkedIntList.** You may not create any new `ListNode`, although you are free to declare `ListNode` variables.

```java
public class ListNode {
    public int data;
    public ListNode next;
}

public class LinkedIntList {
    // Reference to the first node in the list
    private ListNode front;

    ...
}

public class ListNode {
    public int data;
    public ListNode next;
}

public class LinkedIntList {
    // Reference to the first node in the list
    private ListNode front;

    ...
}
```

Below is the declaration of the method you are to implement as a member of the `LinkedIntList` class. (If you run out of space, you are free to use the back of the page or ask for scratch paper.) Remember not to call any other methods in `LinkedIntList`.

```java
/**
 * Move the last element to the front of the list.
 * (Does nothing if the list has only 0 or 1 node.)
 */
public void rotateRight() {
    if (front == null || front.next == null) {
        return;
    }
    ListNode curr = front;
    while (curr.next.next != null) {
        curr = curr.next;
    }
    ListNode temp = curr.next;
    curr.next = null;
    temp.next = front;
    front = temp;
}
```