1   Debugging

1.1   For each scenario below, describe what you think caused each error.

(a) `Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 55`
`         at Sum.sum(Test.java:15)`
`         at Sum.main(Test.java:28)`

where line 15 of `Sum` is:

`total += nums[i];`

(b) `Exception in thread "main" java.lang.NullPointerException`
`         at IntNode.traverse(IntNode.java:25)`
`         at IntNode.main(IntNode.java:63)`

where line 25 of `IntNode` is:

`nextNode = currNode.next;`

(c) `Test.java:67: error: non-static variable name cannot be referenced`
`from a static context`
`                System.out.println(name);`
`                                  ^`

2   Arrays

Arrays are ordered sequences of fixed length. Unlike Python lists, the `length` must be known when creating an array.

```java
int[] a = new int[3];
```

It is possible to initialize and fill an array in a single expression.

```java
int[] b = new int[]{1, 2, 3};
```

Java can infer the type of the array from its context, yielding this shorthand.

```java
int[] c = {1, 2, 3};
```

Uninitialized values have a default value like `0`, `false`, or `null`.

```java
String[] c = new String[1];
c[0] == null;
```

2.1 Provide a descriptive name for each of the following methods. Assume that `values` contains at least one element.

(a)

```
public static boolean _____(int[] values) {
    int k = 0;
    while (k < values.length - 1) {
        if (values[k] > values[k + 1]) {
            return false;
        }
        k = k + 1;
    }
    return true;
}
```

(b)

```
public static void _____(int[] values) {
    int k = 0;
    while (k < values.length / 2) {
        int temp = values[k];
        values[k] = values[values.length - 1 - k];
        values[values.length - 1 - k] = temp;
        k = k + 1;
    }
}
```

2.2   Implement squareSum, which takes an **int[]** and of values and returns the
sum of all the squared values.

```java
public class Arrays {
    public static int squareSum(int[] values) {




    }
}
```

2.3 Implement `squareSum` *using recursion.* Assume the input array of `values` is non-empty.

```java
public class Arrays {
    public static int squareSumRecursive(int[] values, int index) {




    }
}
```

3   Mystery

3.1 Define `skip`, which takes in an `IntList` and destructively removes every other node starting from the second node. Do not use recursion.

```java
public class IntList {
    public int first;
    public IntList rest;
    public IntList(int first, IntList rest) {
        this.first = first;
        this.rest = rest;
    }
    public static void skip(IntList L) {




    }
}
```

3.2   What does `mystery` do? Draw the box-and-pointer diagram!

```java
public class IntList {
    public int first;
    public IntList rest;
    public IntList(int first, IntList rest) {
        this.first = first;
        this.rest = rest;
    }
    public static IntList mystery(IntList L) {
        if (L == null || L.rest == null) {
            return L;
        } else {
            IntList x = mystery(L.rest);
            L.rest.rest = L;
            L.rest = null;
            return x;
        }
    }
    public String toString() {
        String result = "";
        IntList p = this;
        while (p != null) {
            result = result + p.first + " ";
            p = p.tail;
        }
        return result;
    }
    public static void main(String[] args) {
        IntList x = new IntList(2, new IntList(3, new IntList(4, new IntList(5, null))));
        System.out.println(x);
        IntList y = mystery(x);
        System.out.println(x);
        System.out.println(y);
    }
}
```