INSTRUCTIONS:

- You have **60 minutes** to complete the exam.
- You *will* receive a deduction if you keep working after the instructor calls for papers.
- This exam is closed-book and closed-notes. You may not use any computing devices including calculators.
- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
- Do not abbreviate code, such as “ditto” marks or dot-dot-dot (“…” marks). The only abbreviations that are allowed for this exam are: `S.o.p` for `System.out.print` and `S.o.pln` for `System.out.println`.
- You do not need to write import statements in your code.
- You may not use extra scratch paper on this exam. Use the provided spaces for extra work.
- If you write work you want graded on a strange page, clearly label it.
- If you enter the room, you must turn in an exam before leaving the room.
- You must show your Student ID to a TA or instructor for your exam to be accepted.
- If you get stuck on a problem, move on and come back to it later.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td></td>
<td>6</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Σ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
Question #1: Scratch Work
Mechanical Missions.
This section tests whether you are able to trace through code of various types in the same way a computer would.

1. Recursive Tracing [15 points]

```java
public static void mystery(int n) {
    if (n <= 1) {
        System.out.print(n);
    } else {
        System.out.print(n + " ",");
        mystery(n / 2);
        System.out.print(", " + n);
    }
}
```

For each of the following, fill in the String printed when the given mystery method call is run.

<table>
<thead>
<tr>
<th>mystery Call</th>
<th>String Printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) mystery(1)</td>
<td>1</td>
</tr>
<tr>
<td>(b) mystery(3)</td>
<td>3, 1, 3</td>
</tr>
<tr>
<td>(c) mystery(4)</td>
<td>4, 2, 1, 2, 4</td>
</tr>
<tr>
<td>(d) mystery(6)</td>
<td>6, 3, 1, 3, 6</td>
</tr>
<tr>
<td>(e) mystery(12)</td>
<td>12, 6, 3, 1, 3, 6, 12</td>
</tr>
</tbody>
</table>
2. ListNode Before & After [15 points]

Your task in this question is to write code to turn the “before” picture into the “after” picture. You may not change any existing node’s data value. You may not construct any new nodes. In each part, you may declare up to two new variables of type ListNode. Recall the ListNode class from lecture:

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

    public ListNode(int data) { ... }
    public ListNode(int data, ListNode next) { ... }
}
```

<table>
<thead>
<tr>
<th>Before &amp; After</th>
<th>Your Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="" alt="Diagram A" /></td>
<td>list2 = list1.next.next; list1.next.next = null;</td>
</tr>
<tr>
<td><img src="" alt="Diagram B" /></td>
<td>list2.next = list1.next; list1.next = list2; list2 = null;</td>
</tr>
<tr>
<td><img src="" alt="Diagram C" /></td>
<td>list1.next.next = list2.next; list2.next = list1; list1 = list1.next; list2.next.next = null;</td>
</tr>
<tr>
<td><img src="" alt="Diagram D" /></td>
<td>ListNode temp = list2; list2 = list1.next.next; list1.next.next = list1; list1 = list1.next; list1.next.next = temp; list2.next = temp.next; temp.next = null;</td>
</tr>
</tbody>
</table>
Programming Pursuits.
This section tests whether you synthesized various topics well enough to write novel programs using those topics.

3. AryItLst [15 points]
Add a method collapse to the ArrayIntList class that collapses the contents of the ArrayIntList of integers by replacing each successive pair of integers with the sum of the pair. If the list stores an odd number of elements, the final element should not be collapsed.

Example Output

<table>
<thead>
<tr>
<th>Before list.collapse()</th>
<th>After list.collapse()</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7, 2, 8, 9, 4, 13, 7, 1, 9, 10]</td>
<td>[9, 17, 17, 8, 19]</td>
</tr>
<tr>
<td>[1, 2, 3, 4, 5]</td>
<td>[3, 7, 5]</td>
</tr>
</tbody>
</table>

Implementation Restrictions
- You may not call any other methods on the ArrayIntList object (e.g., add, remove)
- You may not use a foreach loop on the ArrayIntList.
- You may not use any other data structures such as arrays, lists, queues, etc.
- Your solution should run in $O(n)$ time, where $n$ is the number of elements in the list.

```java
public class ArrayIntList {
    private int size;
    private int[] elementData;

    // Write Your Solution Here
}
```
Question #3: Extra Solution Space

Solution:

```java
public void collapse() {
    for (int i = 0; i < size / 2; i++) {
        this.elementData[i] = this.elementData[2 * i] + this.elementData[2 * i + 1];
    }
    if (this.size % 2 == 0) {
        this.size /= 2;
    } else {
        this.elementData[this.size / 2] = this.elementData[this.size - 1];
        this.size = this.size / 2 + 1;
    }
}
```
4. Recursion [20 points]
Write a recursive method called `commonChars` that takes two Strings as parameters and returns a new string representing the characters that they have in common. The characters that are different between the two strings should be represented by a "." in the new String. You may assume that the parameters do not contain any "." characters.

**Example Output**

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>commonChars(&quot;&quot;, &quot;&quot;)</code></td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td><code>commonChars(&quot;a&quot;, &quot;a&quot;)</code></td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td><code>commonChars(&quot;AaA&quot;, &quot;Aaa&quot;)</code></td>
<td>&quot;Aa.&quot;</td>
</tr>
<tr>
<td><code>commonChars(&quot;abc&quot;, &quot;cba&quot;)</code></td>
<td>&quot;.b.&quot;</td>
</tr>
<tr>
<td><code>commonChars(&quot;hello world&quot;, &quot;heyya world&quot;)</code></td>
<td>&quot;he... world&quot;</td>
</tr>
</tbody>
</table>

**Implementation Restrictions**
- You must use recursion.
- You may assume that both input Strings are the same length.
- You may assume that both input Strings are not null.
- You may only use the `String` methods on the cheatsheet.
Question #4: Extra Solution Space

Solution:

```java
public static String commonChars(String s1, String s2) {
    if (s1.length() == 0) {
        return "";
    }

    else {
        if (s1.charAt(0) == s2.charAt(0)) {
            return s1.charAt(0) + commonChars(s1.substring(1), s2.substring(1));
        }

        else {
            return "." + commonChars(s1.substring(1), s2.substring(1));
        }
    }
}
```
5. Some Maps [20 points]
Write a method called `sumMaps` that takes two Maps from Strings to ints as parameters and returns a new map that associates each key from either/both of the inputs with the sum of the values in both maps. The map returned by your method should be as fast as possible.

Example Output

<table>
<thead>
<tr>
<th>Input Maps</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{}</code>, <code>{a=1, e=3}</code></td>
<td><code>{a=1, e=3}</code></td>
</tr>
<tr>
<td><code>{a=1, b=2, c=3, d=4}</code>, <code>{a=2, c=5, e=6}</code></td>
<td><code>{a=3, b=2, c=8, d=4, e=6}</code></td>
</tr>
</tbody>
</table>

Implementation Restrictions

- You may assume that the arguments are not null
- Your method should not modify the parameters
- Your method should run in $O(n)$ time (e.g., you should not have any nested loops)
Question #5: Extra Solution Space

Solution:

```java
public static Map<String, Integer> sumMaps(Map<String, Integer> m1, Map<String, Integer> m2) {
    Map<String, Integer> output = new HashMap<String, Integer>();
    for (String key : m1.keySet()) {
        output.put(key, m1.get(key));
    }
    for (String key : m2.keySet()) {
        if (!output.containsKey(key)) {
            output.put(key, 0);
        }
        output.put(key, output.get(key) + m2.get(key));
    }
    return output;
}
```
6. Stacks and Queues... [15 points]
Write a method cancel that takes a Stack of integers as a parameter and modifies it by removing all consecutive pairs that sum to zero. Whenever removing one pair that sums to zero results in another consecutive pair that sums to zero, make sure to remove that pair as well.

Example Output

Implementation Restrictions
- You may assume the input stack is not null.
- You may create at most one stack and no other data structures.
- You may not use recursion to solve this problem.
- Your solution must run in $O(n)$ time, where $n$ is the size of the stack.
Solution:

Question #6: Solution Space

```java
public static void cancel(Stack<Integer> stack) {
    Stack<Integer> tmp = new Stack<Integer>();
    while (!stack.isEmpty()) {
        int top = stack.pop();
        if (!stack.isEmpty() && top + stack.peek() == 0) {
            stack.pop();
        } else if (!tmp.isEmpty() && tmp.peek() + top == 0) {
            tmp.pop();
        } else {
            tmp.push(top);
        }
    } while (!tmp.isEmpty()) {
        stack.push(tmp.pop());
    }
}
```
boolean weekdays;
int time;
int [] brain;
// Let the wake-up begin!
for (int i = 0; i < numBrainCells; i++) {
    turnOn(brain[i]);
    System.out.println("Yawn");
}

getTime(time);
isItaWorkday(weekday);

void smile() {
    int [] usualDisArray;
    System.out.println("Honey, where are you?");
    System.out.println("Great job!");

    System.out.println("Son, all I wanted was a cup of coffee to start my day."
    );
    System.out.println("Then why'd you ask for Java?");
}