Practice Exam 1 (Exam 1 date: Friday 3/4)

Logistics:

- In-class, written exam.
- Closed-book, notes, and technology.
- All relevant Java API documentation will be provided for you.
- Exam is out of 100 points (10% of your overall grade).
- This practice exam mimics the layout and types of problems on the actual exam.
- All topics covered through Tuesday 3/1 are fair game.

Name: ____________________________

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1: Cellulose Coding Fibers

Consider the following methods:

```java
public static int[] mystery1(int k) {
    int[] arr = new int[k];
    for (int i = 0; i < k; i++) {
        arr[i] = i + 1;
    }
    // Point A
    mystery2(arr)
    // Point C
    return arr;
}

public static void mystery2(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        arr[i] = arr[i] * arr[i + 1];
    }
    // Point B
}
```

Give complete stack-and-heap diagrams outlining the state of memory at each of the given points above. The style of your diagrams should be identical to that presented in class. You should reproduce the diagrams completely at each point of the program.

Say we call `mystery1(3)` ...

// POINT A

```
<table>
<thead>
<tr>
<th>Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>3</td>
</tr>
<tr>
<td>arr</td>
<td></td>
</tr>
</tbody>
</table>
```

// POINT B

```
<table>
<thead>
<tr>
<th>Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>3</td>
</tr>
<tr>
<td>arr</td>
<td></td>
</tr>
</tbody>
</table>
```

// POINT C

```
<table>
<thead>
<tr>
<th>Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>3</td>
</tr>
<tr>
<td>arr</td>
<td></td>
</tr>
</tbody>
</table>
```

```
Problem 2: I Have a Proposition for You

Consider the following Java method:

```java
public static int calculate(int x1, int x2, Scanner console) {
    int result = 0;
    int input = console.nextInt();
    if (x1 > x2) {
        int temp = x2;
        x2 = x1;
        x1 = temp;
    }
    // Point A
    while (x1 < input && input < x2) {
        // Point B
        result += input;
        if (input - x1 > x2 - input) {
            x1 = console.nextInt();
        } else {
            x2 = console.nextInt();
        }
    }
    // Point C
    return result;
}
```

For each of the propositions below, determine if the proposition never holds (×), sometimes holds (?), or always holds (√) at the given program points.

<table>
<thead>
<tr>
<th></th>
<th>x1 &gt; x2</th>
<th>input &lt; x2</th>
<th>result &gt;= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>×</td>
<td>?</td>
<td>√</td>
</tr>
<tr>
<td>B</td>
<td>×</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>C</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

In a sentence or two, describe what the method returns as output:

Calculate returns the successive additions of an input to the user as long as that input remains sandwiched between two other inputs that change every round. (Dumb, I know >:<).
Problem 3: Developing a Complex

For each of the following methods:

1. Give a mathematical function or recurrence that models the time complexity of the method. State explicitly what operations your function tracks as well as what the input to the function represents. If your model is a recurrence relation, solve that relation for an explicit mathematical function.

2. Give a tight upper-bound for your function using Big-O notation. You can simply state the upper-bound rather than formally proving it correct.

```java
public static int f1(int[] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        sum += arr[i][0];
        for (int j = 0; j < arr.length; j++) {
            sum += arr[i][j];
        }
    }
    return sum;
}
```

\[
T(n) = \text{array accesses as a function of input array length}\]

\[
T(n) = \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} \sum_{j=1}^{n} 1 = n + n^2
\]

\[
T \in O(n^2)
\]

```java
public static String f2(String s, int i) {
    if (i < s.length()) {
        return s.charAt(i) + f2(s, i + 1) + s.charAt(i);
    } else {
        return s.charAt(i);
    }
}
```

\[
T(n) = \text{chars accessed as a function of length of input string under consideration.}\]

\[
T(0) = 0 \quad \text{[after k]} \quad \text{[setting k=n]}
\]

\[
T(n) = 2 + T(n-1) = 2k + T(n-k) = 2n
\]

\[
T \in O(n)
\]
Problem 4: The Classiest Problem

Write a complete Java class, DonationTracker, that tracks donations for a charity event. The class should support the following constructor and methods:

- DonationTracker(): constructs a new, empty DonationTracker.
- void add(String name, int amount): adds the given donation to the tracker. The donation amount must be non-negative; if it is not, the method throws an IllegalArgumentException.
- int sum(): reports the total amount of money in the donation tracker.
- void printAll(): prints the donations to the console, one per line, in the format: "name : amount".

You are free to create any auxiliary classes you need to solve the problem. You may also use the ArrayList<T> class from the Java standard library in your solution. The List<T> interface defines three useful methods for this problem: void add(int v), T get(int n), and int size().

```java
public class Donation {
    public int amt;
    public String name;
    public Donation (int amt, String name) {
        this.amt = amt;
        this.name = name;
    }

    public void printAll() {
        for (int i=0; i < donations.size(); i++) {
            System.out.println(String.format("%s : %d", donations.get(i).name, donations.get(i).value));
        }
    }

    public void add(String name, int amount) {
        if (amount < 0) throw new IllegalArgumentException();
        donations.add (new Donation(name, amount));
    }

    public int sum () {
        int ret = 0;
        for (int i=0; i < donations.size(); i++) {
            ret += donations.get(i).amt;
        }
        return ret;
    }

    public class DonationTracker {
        private List<Donation> donations;
        public DonationTracker () {
            donations = new ArrayList<>();
        }

        public void add (String name, int amount) {
            if (amount < 0) throw new IllegalArgumentException();
            donations.add (new Donation(name, amount));
        }

        public int sum () {
            int ret = 0;
            for (int i=0; i < donations.size(); i++) {
                ret += donations.get(i).amt;
            }
            return ret;
        }
    }
```
Problem 5: A Link to the Past

Consider the following implementation of a generic linked list:

```java
public class Node<T> {
    public T value;
    public Node<T> next;
    public Node(T value, Node<T> next) {
        this.value = value;
        this.next = next;
    }
}
```

```java
public class LinkedList<T> {
    private Node<T> first;
    // ...
}
```

Write a method for the `LinkedList<T>` class called `removeEvery(n)` that removes every `n`th element from the list, starting with the first. In addition, state the worst case time complexity of the method using Big-O notation in terms of the length of the list `n`; you don’t need to formally prove this bound.

```java
void removeEvery(int n) { // if (n == 0) first = null; if (first == null) return; Node<T> cur = first;
    while (cur.next != null) { for (int i = 0; i < n - 2; i++) { cur = cur.next; if (cur.next == null) return; } cur.next = cur.next.next; // deletion; cur = cur.next; // next node after deleted node
```