INSTRUCTIONS:

• You have 50 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.
Mechanical Missions.
This section tests whether you are able to trace through code of various types in the same way a computer would.

1. Mystery(Parameter, Parameter) [12 points]

For each line of output, write down what is printed out.

<table>
<thead>
<tr>
<th>Line</th>
<th>Text Printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Line 1</td>
<td>they totally can like yes</td>
</tr>
<tr>
<td>(b) Line 2</td>
<td>Shannon totally maybe like can</td>
</tr>
<tr>
<td>(c) Line 3</td>
<td>like totally they like can</td>
</tr>
<tr>
<td>(d) Line 4</td>
<td>Michael totally no like can’t not</td>
</tr>
</tbody>
</table>
2. assert(true) [15 points]
In this question, you will identify various assertions as being either always true, never true or sometimes true/sometimes false at various points in program execution. The comments in the method below indicate the points of interest.

```
public static void mystery(int x, int y) {
    int z = 0;
    // Point A

    while (x < y) {
        // Point B
        z++;
        if (z % 2 == 0) {
            x = x * 2;
            // Point C
        }
        else {
            y--; // Point D
        }
    } // Point E
    System.out.println(z);
}
```

Fill in the table below with the words ALWAYS/SOMETIMES/NEVER. You may abbreviate them.

<table>
<thead>
<tr>
<th>Point</th>
<th>x &lt; y</th>
<th>z == 0</th>
<th>z % 2 == 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>S</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>C</td>
<td>S</td>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>S</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>E</td>
<td>N</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>
For each line of output, write down what is printed out.

<table>
<thead>
<tr>
<th>Line</th>
<th>Text Printed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>Line 1 10 5 3</td>
</tr>
<tr>
<td>(b)</td>
<td>Line 2 10 5 3 5 3</td>
</tr>
<tr>
<td>(c)</td>
<td>Line 3 5 3 10</td>
</tr>
<tr>
<td>(d)</td>
<td>Line 4 10 3 10 3 10</td>
</tr>
</tbody>
</table>
Programming Pursuits.
This section tests whether you synthesized various topics well enough to write novel programs using those topics.

4. Boxed In [15 points]
Write a method called printBox that takes a String (side) as a parameter and prints a square with dimensions equal to the length of side. Your method should use side as the text for each of the four sides of the box.

Example Output

```
printBox("***")
---→
 ***
 * *
 ***

printBox("abcde")
---→
 abcde
 b d
 c c
d b
edcba
```

Implementation Restrictions
- You may not define auxiliary structures (no extra Strings or arrays).
- You may (but are not required to) define extra static methods to reduce redundancy.
- You may assume that the input String is at least one character long.
- You may only use the String methods on the cheatsheet.

Solution:
```java
public static void printBox(String side) {
  if (side.length() > 1) {
    System.out.println(side);
  }

  for (int i = 1; i < side.length() - 1; i++) {
    System.out.print(side.charAt(i));
    for (int j = 0; j < side.length() - 2; j++) {
      System.out.print(" ");
    }
    System.out.println(side.charAt(side.length() - 1 - i));
  }

  for (int i = side.length() - 1; i >= 0; i--) {
    System.out.print(side.charAt(i));
  }
  System.out.println();
}
```
5. 1 + 1 = 2 [18 points]
An arithmetic sequence is an array of numbers where the difference between every adjacent pair of integers is the same constant. Write a method called `isArithmeticSequence` that takes an int array as a parameter and returns true if the array is a valid arithmetic sequence and false otherwise. Arrays with fewer than two elements should be considered arithmetic sequences.

**Example Output**

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isArithmeticSequence({1, 2, 3, 4})</code></td>
<td>true</td>
</tr>
<tr>
<td><code>isArithmeticSequence({28, 32, 36, 40})</code></td>
<td>true</td>
</tr>
<tr>
<td><code>isArithmeticSequence({20, 10, 0, -10})</code></td>
<td>true</td>
</tr>
<tr>
<td><code>isArithmeticSequence({2, 3, 5, 7, 11})</code></td>
<td>false</td>
</tr>
</tbody>
</table>

**Solution:**

```java
public static boolean isArithmeticSequence(int[] arr) {
    if (arr.length < 2) {
        return true;
    }
    int diff = arr[1] - arr[0];
    for (int i = 0; i < arr.length - 1; i++) {
        if (arr[i + 1] - arr[i] != diff) {
            return false;
        }
    }
    return true;
}
```
6. Single File Please! [18 points]
Write a method called gradeMCTest that takes two Scanners as parameters: students and correct which represent student answers to a multiple choice test and the correct answers to the same test, respectively.

For example, the file students points to might look like:

```bash
>> ABCAC
>> ACCDC
>> CCCCC
```

Notice this input has five questions and three students to grade.

And the file correct points to might look like:

```bash
>> A
>> C
>> B
>> D
>> A
```

Notice each answer is on its own line, and the entire file represents a single set of answers to the test.

To “grade” a multiple choice test, for each student, compare the correct answer for each question with their answer, and count the number of correct responses. Your method should output the number of correct answers for each student, one student per line.

The correct output to the inputs above would be:

```bash
>> 1
>> 3
>> 1
```

**Implementation Restrictions**
- You may (but are not required to) define extra static methods to reduce redundancy.
- You may assume that the input Scanners point to valid inputs (e.g., the right length for each line).
- You may not assume a particular number of questions or a particular number of students.

**Solution:**

```java
public static void gradeMCTest(Scanner students, Scanner correct) {
    String answers = "";
    while (correct.hasNext()) {
        answers += correct.next();
    }
    while (students.hasNextLine()) {
        String student = students.nextLine();
        int result = 0;
        for (int i = 0; i < answers.length(); i++) {
            if (student.charAt(i) == answers.charAt(i)) {
                result++;
            }
        }
        System.out.println(result);
    }
}
```