Name (print): ____________________________________________

- **INSTRUCTIONS:**
  - Keep your eyes on your own paper and do your best to prevent anyone else from seeing your work.
  - Do NOT communicate with anyone other than the professor/proctor for ANY reason in ANY language in ANY manner.
  - This exam is closed notes, closed books, and no calculator.
  - Turn all mobile devices off and put them away now. You cannot have them on your desk.
  - Write neatly and clearly indicate your answers. What I cannot read, I will assume to be incorrect.
  - Stop writing when told to do so at the end of the exam. I will take 5 points off your exam if I have to tell you multiple times.
  - Academic misconduct will not be tolerated. Suspected academic misconduct will be immediately referred to the Emory Honor Council. Penalties for misconduct will be a zero on this exam, an F grade in the course, and/or other disciplinary action that may be applied by the Emory Honor Council.
  - You may assume all `import` statements are included automatically for you.
  - You may use standard abbreviations such as S.O.P. for `System.out.println`.
  - Unless otherwise specified, you do **not** need to write an entire program. Just writing the specified method or snippet of code is sufficient.

- **TIME:** This exam has 9 questions on 13 pages including the title page. Please check to make sure all pages are included. You will have 75 minutes to complete this exam.

```
I commit to uphold the ideals of honor and integrity by refusing to betray the trust bestowed upon me as a member of the Emory community. I have also read and understand the requirements and policies outlined above.

Signature: ____________________________________________
```

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>8</td>
<td>21</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>75</td>
</tr>
<tr>
<td>Score</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. Consider the class below.

```java
public class ShadowVars {
    public static int q;
    public static int z;

    public static void method1(int z) {
        z = 3;
        System.out.println("qz 1: "+ q + " " + z);

        q = z - ShadowsVars.z;
        System.out.println("qz 2: "+ q + " " + z);
    }

    public static int method2(int q) {
        int a = 4;
        System.out.println("qz 3: "+ q + " " + z);
        return a + q;
    }

    public static void main(String[] args) {
        z = 1;
        q = 3;
        System.out.println("qz 4: "+ q + " " + z);
        int z;
        z = 14;
        q = 12;
        method1(z);
        q = method2(4);
        System.out.println("qz 5: "+ q + " " + z);
    }
}
```

(a) (1 point) List all the class variables in the code above.

**Solution:** q and z

(b) (1 point) How many parameter variables occur in the code above?

**Solution:** 3 (one per method)

(c) (1 point) How many local variables are in the `main` method?

**Solution:** 1 (z)

(d) (5 points) Give the output of the program above.

**Solution:**

```
qz4: 3 1
qz1: 12 3
qz2: 2 3
```
| qz3: 4 1 |
| qz5: 8, 14 |
-1 each incorrect print statement
2. For each of the parts below, give the output of the code.

(a) (6 points)
```java
public static void muppet(String oscar, String elmo, String grover) {
    System.out.println(oscar + " is my favorite, but " + grover +
        "is cool too. I don't like " + elmo);
}

public static void main(String[] args) {
    String elmo = "grover";
    String oscar = "elmo";
    String grover = "oscar";

    muppet(grover, oscar, elmo);
    muppet("grover", grover, "oscar");
    oscar = grover;
    grover = "big bird";
    muppet(grover, oscar, elmo);
}
```

Solution: oscar is my favorite, but grover is cool too. I don’t like elmo.
grover is my favorite, but oscar is cool too. I don’t like oscar.
big bird is my favorite, but grover is cool too. I don’t like oscar.
1 pt per sentence

(b) (3 points)
```java
int[] x = new int[5];
System.out.println("x[1]: " + x[1]);

for(int i = 0; i < x.length; i+=2) {
    x[i] = i+2;
}

System.out.println("x: " + Arrays.toString(x));
int z = x[1] + x[2];
System.out.println("x[1] + x[2]: " + z);
```

Solution: (.5pt) x[1]: 0
(2 pts) x: [2, 0, 4, 0, 6]
(.5pt) x[1] + x[2]: 4
(c) (4 points) What is the output of the code below if it is run with the method call `lizSays()`?

```java
public static void carolSays(int x) {
    if (x % 2 == 0) {
        System.out.println("Hi!");
    }
    if (x % 3 == 0) {
        System.out.println("My name is Carol.");
    }
}

public static void lizSays() {
    int x = 8;
    if(x > 5) {
        System.out.println("Nice to meet you.");
    } else {
        System.out.println("I don’t like you.");
    }

    int num = 1;
    while (num < 5) {
        carolSays(num);
        num = num + 1;
    }
}
```

Solution: Nice to meet you.
Hi
My name is Carol.
Hi

(d) (4 points)
```java
String[] stringArray = {"hello", "world", "cs", "170!!"};
String s = "";

for(int i = 0; i < stringArray.length; i++) {
    if (stringArray[i].length() >= 3) {
        s = s + stringArray[i].charAt(i);
    }
}

System.out.println(s);
```

Solution: “ho!”
(e) (4 points)
public static int method1(int x) {
    x = x + 5;
    return 12;
}

public static void method2(int[] x) {
    for (int i = 0; i < x.length; i++) {
        x[i] = x[i] + i;
    }
}

public static void main(String[] args) {
    int x = 2;
    System.out.println("x: " + x);

    int y = method1(x);
    System.out.println("xy: " + x + "," + y);

    int[] z = {1, 2, 3, 4, 5};
    System.out.println("z: " + Arrays.toString(z));

    method2(z);
    System.out.println("z: " + Arrays.toString(z));
}

Solution: (.5pt) x: 2
(1pt) xy: 2, 12
(.5pt) z: [1,2,3,4,5]
(2pts) z: [1,3,5,7,9]
3. Label each of the following statements as true or false. If the statement is false, briefly (no more than 1 sentence!) explain why.

(a) (1 point) Parameters to a method are optional; that is, you can write a method which contains no parameters.

   (a) **true**

(b) (1 point) When a program calls (or invokes) a method, program control is transferred to the called method. The called method returns control to the caller when its return statement is reached or when the end of the method is reached (i.e., its closing curly brace is encountered).

   (b) **true**

(c) (1 point) The scope of a variable is the part of the program where the variable can be used or referenced.

   (c) **true**

(d) (1 point) The arguments that are passed to a method should have the same number, type, and order as the parameters in the method header (or method signature).

   (d) **true**

(e) (1 point) “Pass by value” means that when you invoke a method with a parameter, the value of the argument is passed to the method.

   (e) **true**

(f) (1 point) The method header (or signature) specifies the return value type, method name, and parameters of the method.

   (f) **true**

(g) (1 point) An index to an array must be either an integer or an integer expression.

   (g) **true**

(h) (1 point) Arrays in Java are static. That is, they cannot be dynamically resized to add or delete elements.

   (h) **true**

(i) (1 point) A `return` statement is not needed for a `void` method, but it can be used for terminating the method and returning to the method’s caller. The syntax for this statement is `return void;`

   (i) **false; just return;**
4. Searching.
   (a) (2 points) Briefly describe the algorithm used in a sequential or linear search.

   **Solution:** You examine each element in the array, in order and compare it against the value you’re searching for. If there isn’t a match, you proceed to the next element until you either find the value you’re searching for or reach the end of the array.

(b) (2 points) Briefly describe the algorithm used in a binary search.

   **Solution:** At each step, you examine the element from the middle of the array. If the value you’re searching for is greater than that element, you “throw out” the lower half of the array and continue searching in the upper half. If the search value is less than the middle element, you throw out the upper half of the array and continue searching the lower half. Thus, at each step, you eliminate 50% of the remaining elements to be searched.

(c) You are given an array with 31 elements in it.
   i. (2 points) Assuming the array is unsorted, how many elements must you inspect in order to definitively determine that a number does not occur in the array?

   **Solution:** 31

   ii. (2 points) Assuming the array is sorted, how many elements must you inspect in order to definitively determine that a number does not occur in the array?

   **Solution:** 5
   middle element of 31 element array
   middle element of remaining 15 element array
   middle element of remaining 7 element array
   middle element of remaining 3 element array
   last element remaining
5. (6 points) The array \{4, 8, 6, 2, 0\} can be sorted via different sorting algorithms including Selection Sort, Insertion Sort, and Bubble Sort. Label each sequence of sorting steps below with the name of the algorithm used to sort the initial array. The initial array has been repeated for you as Step 1 for clarity.

(a) Step 1: \{4, 8, 6, 2, 0\}
    Step 2: \{0, 8, 6, 2, 4\}
    Step 3: \{0, 2, 6, 8, 4\}
    Step 4: \{0, 2, 4, 8, 6\}
    Step 5: \{0, 2, 4, 6, 8\}

**Solution:** Selection Sort

(b) Step 1: \{4, 8, 6, 2, 0\}
    Step 2: \{4, 6, 2, 0, 8\}
    Step 3: \{4, 2, 0, 6, 8\}
    Step 4: \{2, 0, 4, 6, 8\}
    Step 5: \{0, 2, 4, 6, 8\}

**Solution:** Bubble Sort

(c) Step 1: \{4, 8, 6, 2, 0\}
    Step 2: \{4, 8, 6, 2, 0\}
    Step 3: \{4, 6, 8, 2, 0\}
    Step 3: \{2, 4, 6, 8, 0\}
    Step 4: \{0, 2, 4, 6, 8\}

**Solution:** Insertion Sort
6. (5 points) Write a function named dominant which takes 3 integers as parameters. It returns a boolean value. The function should return true if any one of the three integers is larger than the sum of the other two integers.

Solution:

```java
public static boolean dominant(int a, int b, int c) {
    if (a > b+c || b > a+c || c > a+b) {
        return true;
    } else {
        return false;
    }
}
```

+1 for function header
+1 for a g.t b+c case (returning true)
+1 for b g.t a+c case (returning true)
+1 for c g.t a+b case (returning true)
+1 for returning false if none matched
7. (5 points) Write a function which converts a number of puppies to their equivalent buckets of cuteness. The function should be named `puppiesToBuckets` and take an integer representing the number of puppies as the input parameter. The function should return the equivalent buckets of cuteness as calculated below. You should decide what the appropriate return type is for this function! If the user enters a number of puppies less than or equal to 0, the function should return 0. In the equation below, $b$ represents buckets of cuteness and $p$ represents the number of puppies:

$$b = p + \frac{p + 2.5}{p^2}$$

Solution:

```java
public static double puppiesToBuckets(int p) {
    double b = 0.0;
    if (p > 0) {
        b = p + ( (p+2.5) / (p*p) );
    }
    return b;
}
```

Common mistake: not using correct Java syntax and simply restating the mathematical formula given above in the code.
8. (5 points) Write a method named allOdd which takes an array of integers as an input parameter and returns a boolean value. The function should return true if all elements in the parameter array are odd and false if any element in the array is not odd.

Solution:

```java
public static boolean allOdd(int[] a) {
    for(int i = 0; i < a.length; i++) {
        if (a[i] % 2 != 1) {
            return false;
        }
    }
    return true;
}
```

Common mistake: returning true inside loop. If you find an even element, you can immediately return false if you encounter an even element, but you must wait until you examine all elements in the array (and determine they are all odd) before returning true.
9. (8 points) Write a function named `filterNums` which takes an array of ints, `list` and a single int, `limit` as a parameter. The function should return a new array containing only the elements in `list` that were greater than `limit`.

**Solution:**

```java
public static int[] filterNums(int[] list, int limit) {
    int count = 0;
    for (int i = 0; i < list.length; i++) {
        if (list[i] > limit) {
            count++;
        }
    }

    int[] newlist = new int[count];
    int j = 0;
    for (int i = 0; i < list.length; i++) {
        if (list[i] > limit) {
            newlist[j] = list[i];
            j++;
        }
    }

    return newlist;
}
```

+1 method header
+2 determines number of elements g.t. `limit`
+2 creates new array to be returned of correct size
+2 populates new array with correct elements/data
+1 returns new array/doesn’t modify old one

Common mistakes: returning the new array at the wrong place (eg inside an if statement or the for loop). Using the same index for both the new (sometimes smaller) array and the original array. Trying to resize the arrays or somehow create the array without an initial size.