This is a closed note, closed book exam. Any place where you are asked to write code, you must declare all variables that you use. However, I just want code fragments, you must not write extra code such as a class specification or extraneous print statements.

1. [10 points] Given the following javadoc specification, show how to call and get a return value from the function `pow()`. Let $a$ equal 5, $b$ equal 3, and store the result in $c$. Print out the value that is returned.

   ```java
   static double pow(double a, double b)
   Returns of value of the first argument raised to the power of the second argument.
   ```

   ```java
double a, b, c;  // 2 points
a = 5;           // 1 point
b = 3;           // 1 point
c = pow(a, b);   // 4 points
System.out.println(c);  // 2 points
   ```

2. [10 points] Write a method called `circleArea()` that takes as a parameter the radius of a circle and returns its area.

   ```java
   public static double circleArea(double radius) {  // 3 points (return type, name, parameter)
       double area;  // 2 points
       area = Math.PI * radius * radius;  // 3 points
       return area; // 2 points
   }
   ```
3. [10 points] Given the following numbered lines of code

```java
1   class Foo {
2   3       public static void main(String[] args) {  
4           int a = 5;
5   6           for(int b = 0; b < 100; b++) {
7           b = a*b;
8            }  
9   10          int c;
11   12          c = foo(a);
13   14           System.out.println(c);
15        }  
16   17       public static double foo(int d) {
18           double e = 1/d;
19            return e;
20            }  
21    }
```

Write the line numbers of the lines that constitute the scope of each variable:

- a: 5-15    2 points
- b: 6-8     2 points
- c: 10-15   2 points
- d: 17-20   2 points
- e: 18-20   2 points
4. [10 points] What does this program print out? Why?

class TestProgram{
    public static void main(String[] args){
        int a = 1, b = 2;
        System.out.println(a);
        System.out.println(b);
        swap(a, b);
        System.out.println(a);
        System.out.println(b);
    }
}

static void swap(int x, int y) {
    int temp;
    System.out.println(x);
    System.out.println(y);
    temp = x;
    x = y;
    y = temp;
    System.out.println(x);
    System.out.println(y);
}

It prints out:
1  1 point
2  1 point
1  1 point
2  1 point
2  1 point
1  1 point
1  1 point
2  1 point

The reason it does this is that in swap, the values of x and y are swapped, but in main the values of a and b are not swapped because x and y are merely copies of a and b. 2 points
5. Recursion
   a) [10 points] Write a non-recursive implementation of the `pow()` function from problem 1

```java
public static int pow(int a, int b) {
    int result = 1;
    for(int i = 0; i < b; i++) {
        result = result * a;
    }
    return result;
}
```

b) [10 points] Write a recursive implementation of the same function

```java
public static int pow(int a, int b) {
    if(b == 0)
        return 1;
    else
        return a*pow(a, b-1);
}
```
6. Arrays
a) [5 points] Declare and create storage for an array of 10 integers called foo

```java
int[] foo; 2 points
foo = new int[10]; 3 points
```

b) [5 points] Write a method called `bar()` that takes an array of integers and adds 1 to each element

```java
public static void bar(int[] theArray) {
    for(int i = 0; i < theArray.length; i++) {
        theArray[i]++; 2 points
    }
}
```

c) [5 points] Show how you would call `bar()` with the array you created as a parameter

```java
bar(foo); 5 points
```

d) [5 points] After the call to `bar()`, is the original array changed in any way? Why?

Yes. Because when you pass an array to a method, you are passing a reference to the array, so the formal parameter refers to the same storage as the actual parameter. 5 points

7. [10 points] Write a method called `arrayMin()` that takes an array of doubles as a parameter and returns the index of the smallest element of the array.

```java
public static int arrayMin(double[] theArray) {
    int min = 0; 1 point (initializing min)
    for(int i = 0; i < theArray.length; i++) {
        if(theArray[i] < theArray[min]) {
            min = i; 3 points (setting min)
        }
    }
    return min; 2 points (returning min)
}
```
8. [10 points] Conway’s Life program simulates cell life. It is “played” on a 2D array of elements that represent cells. It has three basic rules:

Rule 1. If a cell is dead and it has exactly three neighbors that are alive, it comes to life in the next generation.
Rule 2. If a cell is alive and it has fewer than 2 or more than 3 neighbors that are alive, it dies in the next generation.
Rule 3. Otherwise, the cell will be the same in the next generation as it is in the current generation.

Suppose that we have a life game board that is a 2D array of booleans, where the boolean value false means that a cell is dead and true means that a cell is alive. Write a method called live() that takes the board as a parameter and returns an updated board that shows what it looks like after exactly one generation.

```java
public static void live(boolean[][] board) {
    boolean[][] boardCopy = board.clone();
    for(int i = 0; i < board.length; i++) {
        for(int j = 0; j < board[i].length; j++) {
            int n = neighbors(boardCopy, i, j);
            if(board[i][j] == false && n == 3)
                board[i][j] = true;
            else if(board[i][j] == true && n != 2 && n != 3)
                board[i][j] = false;
        }
    }
}
```

```java
public static int neighbors(boolean[][] board, int i, int j) {
    int n = 0;
    if(i >0 && j > 0 && board[i-1][j-1]) n++;
    if(i > 0 && board[i-1][j]) n++;
    if(i > 0 && j < board[i].length-1 && board[i-1][j+1]) n++;
    if(i < board.length-1 && j > 0 && board[i+1][j-1]) n++;
    if(i < board.length-1 && j < board[i].length-1 && board[i+1][j+1]) n++;
    if(j > 0 && board[i][j-1]) n++;
    if(j < board.length[i]-1 && board[i][j+1]) n++;
    return n;
}
```