CS 142 Sample Midterm Exam #1

1. Array Mystery

Consider the following function:

```
public static void mystery(int[] list) {
    for (int i = 1; i < list.length - 1; i++) {
        if (i % 2 == 0) {
            list[ i ] = list[ i ] - list[ i - 1 ];
        } else {
            list[ i ] = list[ i ] + list[ i - 1 ];
        }
    }
}
```

In the left-hand column below are specific arrays of integers. Indicate in the right-hand column what values would be stored in the array after the call to function mystery in the left-hand column. Write your answer surrounded by curly braces with number separated by commas.

Original Contents of Array	**Final Contents of Array**
`int[] a1 = {1};` `mystery(a1);`	_____
`int[] a2 = {2, 5, 9};` `mystery(a2);`	_____
`int[] a3 = {2, 3, 7, 2};` `mystery(a3);`	_____
`int[] a4 = {2, 4, 5, 4, 3};` `mystery(a4);`	_____
`int[] a5 = {5, 1, 8, 4, 8, 4};` `mystery(a5);`	_____

2. ArrayList Mystery

Write the final contents of each of the `ArrayList`s after it is passed to the following method:

```
public static void mystery4(ArrayList<Integer> list) {
     int size = list.size();
     for (int i = 1; i <= size; i++) {
         list.add(i, i * 10);
     }
     for (int i = 1; i < list.size(); i++) {
         list.remove(i);
     }
 }
```

Your answer should be formatted identically (including spacing) to the inputs listed below.

Original Contents of ArrayList	**Final Contents of ArrayList**
`{34, 2, 5}`	_____
`{12, 34, 65, 1, 4}`	_____
`{54, 23, 7, 2, 4, 2}`	_____

3. Complexity

Give a tight bound of the nearest runtime complexity class for each of the following code fragments in Big-Oh notation, in terms of the variable `N`. In other words, write the code's growth rate as `N` grows. Write a simple expression that gives only a power of `N` using a caret `^` character for exponentiation, such as `O(N^2)` to represent `O(N2)` or `O(log N)` to represent `O(log2 N)`. Do not write an exact calculation of the runtime such as `O(2N3 + 4N + 14)`.

// a)

```
ArrayList<Integer> list = new ArrayList< >();
for (int i = 1; i <= N; i++) {
    list.add(0, 2 * i);
}
ArrayList<Integer> list2 = new ArrayList<>();
for (int i = 0; i < list2.size(); i++) {
    list2.add(list.get(i));
}
System.out.println("done!");
```

// b)

```
int sum = 0;
for (int i = 1; i <= 100000; i++) {
    for (int j = 1; j <= i; j++) {
        for (int k = 1; k <= N; k++) {
            sum++;
        }
    }
}
for (int x = 1; x <= N; x += 2) {
    sum++;
}
System.out.println(sum);
```

// c)

```
ArrayList<Integer> q = new ArrayList<
   >();
for (int i = 1; i <= 4 * N; i++) {
    q.add(i);
}

int counts[ 100 ];
while (!q.isEmpty()) {
    int k = q.remove(0);
    counts[ k % 100 ] = -2 * k;
}
System.out.println("done!");
```

// d)

```
int map [ N ];
for (int i = 1; i <= N * N; i++) {
    map[ i % 10] = i * i + 1;
}
ArrayList<Integer> set = new ArrayList< >();
for (int i = 0; i < map.length; i++) {
    set.add(map[ i ]);
}
System.out.println("done!");
```

4. Polymorphism Mystery

Assuming that the following classes have been defined:

```
public class Gandalf {
    public void method1() {
        System.out.println("Gandalf
   1");
    }

    public void method2() {
        System.out.println("Gandalf
   2");
        method1();
    }
}

public class Bilbo extends Gandalf {
    public void method1() {
        System.out.println("Bilbo 1");
    }
}

public class Frodo extends Bilbo {
    public void method1() {
        System.out.println("Frodo 1");
        super.method1();
    }

    public void method3() {
        System.out.println("Frodo 3");
    }
}

public class Gollum extends Gandalf {
    public void method3() {
        System.out.println("Gollum 3");
    }
}
```

`var1.method1();`	_____
`var2.method1();`	_____
`var3.method1();`	_____
`var4.method1();`	_____
`var5.method1();`	_____
`var6.method1();`	_____
`var1.method2();`	_____
`var2.method2();`	_____
`var3.method2();`	_____
`var4.method2();`	_____
`var5.method2();`	_____
`var6.method2();`	_____
`((Bilbo)var1).method3();`	_____
`((Gandalf)var1).method2();`	_____
`((Frodo)var4).method1();`	_____
`((Gandalf)var6).method2();`	_____
`((Gandalf)var4).method1();`	_____
`((Frodo)var6).method3();`	_____
`((Frodo)var3).method3();`	_____
`((Frodo)var5).method3();`	_____

And assuming the following variables have been defined:

```
Gandalf var1 = new Frodo();
Gandalf var2 = new Bilbo();
Gandalf var3 = new Gandalf();
Bilbo var4 = new Bilbo();
Bilbo var5 = new Frodo();
Gandalf var6 = new Gollum();
```

In the table above, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in `a/b/c` to indicate three lines of output with `a` followed by `b` followed by `c`. If the statement causes an error, fill in the right-hand column with either the phrase **`compiler error`** or **`runtime error`** to indicate when the error would be detected.

5. Array Programming

Write a static method called `removeZeros` that takes an array of integers as a parameter and that moves any zeros in the array to the end of the array, otherwise preserving the order of the list. For example, if a variable called "list" stores the following values:

```
[7, 2, 3, 0, 4, 6, 0, 0, 13, 0, 78, 0, 0, 19, 14]
```

then the call:

```
removeZeros(list);
```

should rearrange the values in the array so that it stores the following:

```
[7, 2, 3, 4, 6, 13, 78, 19, 14, 0, 0, 0, 0, 0, 0]
```

Notice that the six zeros have been moved to the end of the array and the other values are in the same order as in the original list. You are not allowed to use an auxiliary data structure such as a temporary array or `ArrayList` to solve this problem and you are not allowed to call any methods of the `Arrays` class or the `Collections` class.

6. ArrayList Programming

Write a static method called `removeAdjacentMatches` that removes extra copies of a value that are adjacent to it in an `ArrayList` of integers. For example, if a variable called list contains the following:

```
[1, 3, 3, 15, 2, 2, 2, 2, 1, 19, 3, 42, 42, 42, 7, 42, 1]
```

and we make the following call:

```
removeAdjacentMatches(list);
```

Then list should store the following values after the call:

```
[1, 3, 15, 2, 1, 19, 3, 42, 7, 42, 1]
```

Notice that the two occurrences of `3` that appear next to each other have been replaced with a single occurrence of `3`, the four occurrences of `2` that appear next to each other have been replaced with a single `2`, and the three occurrences of `42` that appear next to each other have been replaced with a single `42`. Notice, however, that the list still contains three occurrences of `1`, two occurrences of `3`, and two occurrences of `42`. That is because those duplicate values do not appear right next to each other in the list.

You may not construct any extra data structures to solve this problem. You must solve it by manipulating the `ArrayList` you are passed as a parameter.

7. Critter Programming

Write a class called `Eagle` that extends the `Critter` class. The instances of the `Eagle` class switch back and forth between red and blue in an increasing pattern. They should be red for their first move and then blue for one move. Then they should be red for two moves and then blue for two moves. Then they should be red for three moves and then blue for three moves. Then they should be red for four moves and then blue for four moves. And so on, each time increasing the number of moves at a particular color by one. They should be displayed using a less-than character followed by a greater-than character (<>). They should always move `North` until they fight (always by `Pounc`ing). Once they fight they should always move `East`.