Key to CSE142 Sample Final Exam, Winter 2018

1. Reference Mystery. The program produces the following output.

```
21 [41] w: 50, h: 11
20 [41] w: 50, h: 11
21 [43] w: 51, h: 12
20 [43] w: 51, h: 12
```

2.

Original List	Final List
{2, 5, 6, 9}	{2, 8, 17, 9}
{1, 3, 3, 5, 8}	{1, 4, 9, 17, 8}
{4, 4, 5, 8, 2}	{4, 9, 17, 19, 2}
{3, 6, 9, 12, 15}	{3, 12, 24, 39, 15}
{2, 4, 6, 8, 10, 12}	{2, 8, 16, 26, 38, 12}

3. Inheritance. The output produced is as follows.

```
b       d       d       b
b 1     a 1     b 1     b 1
c 2     a 2     c 2     b 2
```

4. Token-Based File Processing. One possible solution appears below.

```
public static void redact(Scanner input) {
    while (input.hasNext()) {
        String next = input.next();
        if (next.equals("CLASSIFIED")) {
            int count = input.nextInt();
            for (int i = 0; i < count; i++) {
                System.out.println("[redacted]");
                input.next();
            }
        } else {
            System.out.println(next);
        }
    }
}
```

5. Line-Based File Processing. One possible solution appears below.

```
public static void diff(Scanner input1, Scanner input2) {
    int line = 0;
    int count = 0;
    while (input1.hasNextLine()) {
        line++;
        String line1 = input1.nextLine();
        String line2 = input2.nextLine();
        if (!line1.equalsIgnoreCase(line2)) {
            count++;
            System.out.println("DIFF: difference found on line #" +
                               line);
            System.out.println("[" + line1 + "] versus [" + line2 + "]");
        }
    }
    if (count > 0) {
        System.out.println();
        System.out.println(count + " line(s) are different");
    }
}
```

6. Arrays. Two possible solutions appear below.

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

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

7. ArrayLists. Three possible solutions appear below.

```
// inserting negatives earlier in the list (using return value of remove)
public static void negativesToFront(ArrayList<Integer> list) {
    int nextSpot = 0;
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i) < 0) {
            int n = list.remove(i);
            list.add(nextSpot, n);
            nextSpot++;
        }
    }
}

// moving nonnegatives to the end, remembering a count of nonnegatives
public static void negativesToFront(ArrayList<Integer> list) {
    int numMoved = 0;
    for (int i = 0; i < list.size() - numMoved; i++) {
        if (list.get(i) >= 0) {
            list.add(list.get(i));
            list.remove(i);
            numMoved++;
            i--;
        }
    }
}

// keeping track of how many values to process and decrementing it for
// each nonnegative that has been moved to the end
public static void negativesToFront(ArrayList<Integer> list) {
    int size = list.size();
    for (int i = 0; i < size; i++) {
        if (list.get(i) >= 0) {
            list.add(list.get(i));
            list.remove(i);
            size--;
            i--;
        }
    }
}
```

8. Critters. One possible solution appears below.

```
public class WatchDog extends Critter {
    private int moveCounter;
    private Direction dir;
    private boolean panic;

    public WatchDog(Direction dir) {
        this.dir = dir;
    }

    public Direction getMove() {
        if (panic) {
            return dir;
        } else {
            moveCounter++;
            if (moveCounter % 6 == 0) {
                return Direction.CENTER;
            } else if (moveCounter % 12<= 5) {
                return Direction.WEST;
            } else { // moveCounter % 12 <= 11
                return Direction.EAST;
            }
        }
    }

    public Attack fight(String opponent) {
        if (opponent.equals("%")) {
            // watch out for panic = !panic;
            panic = true;
        }
        return Attack.ROAR;
    }

    public String toString() {
        if (panic) {
            return "!";
        } else {
            return "D";
        }
    }

    public boolean eat() {
        // watch out for destroying old value of panic here
        bool eat = panic;
        panic = false;
        return eat;
    }
}
```

9. Arrays. Two possible solutions appear below.

```
// loop bounds is half the length and uses 2 * i
public static int[] expand(int[] data) {
    int len = 0;
    for (int i = 0; i < data.length / 2; i++) {
        len = len + data[2 * i];
    }
    int[] result = new int[len];
    // keeps tracks of index which increments by one for each value
    int index = 0;
    for (int i = 0; i < data.length / 2; i++) {
        int times = data[2 * i];
        int n = data[2 * i + 1];
        for (int j = 0; j < times; j++) {
            result[index] = n;
            index++;
        }
    }
    return result;
}

// Uses i+=2 to look at every other value
public static int[] expand(int[] data) {
    int len = 0;
    for (int i = 0; i < data.length; i += 2) {
        len += data[i];
    }
    int[] result = new int[len];
    // keeps track of the beginning index of each "chunk"
    int index = 0;
    for (int i = 0; i < data.length; i += 2) {
        int times = data[i];
        int n = data[i + 1];
        for (int j = 0; j < times; j++) {
            result[j + index] = n;
        }
        index += times;
    }
    return result;
}
```

10. Programming. One possible solution appears below.

```
public static void compress(ArrayList<String> words) {
    int index = 0;
    while (index < words.size()) {
        int count = 1;
        while (index + 1 < words.size() &&
               words.get(index).equals(words.get(index + 1))) {
            words.remove(index);
            count++;
        }
        if (count > 1) {
            words.set(index, count + "*" + words.get(index));
        }
        index++;
    }
}
```