Bryce Griffin

a.) Audio Narration

b.) One difficulty I encountered while creating this program was randomizing the arrays of students. Initially, I decided to use integer arrays, but when I constructed the random array, I would sometimes have duplicates: the same student was accounted for twice. To solve this problem, I decided to use arraylists as opposed integer arrays, since the size of arraylists is dynamic. Thus, when a random value was pulled from the array of students, that student would no longer be able to be selected. Another issue I faced was accounting for the remainder students. If a teacher entered an amount of students that was not evenly divisible by the amount of groups they wanted, the remaining students would form an extra group, often being comprised of only one or two students. To get around this problem, I decided to print the groups vertically. This way, instead of the remaining students forming their own group, they were distributed amongst already formed groups as was necessary.

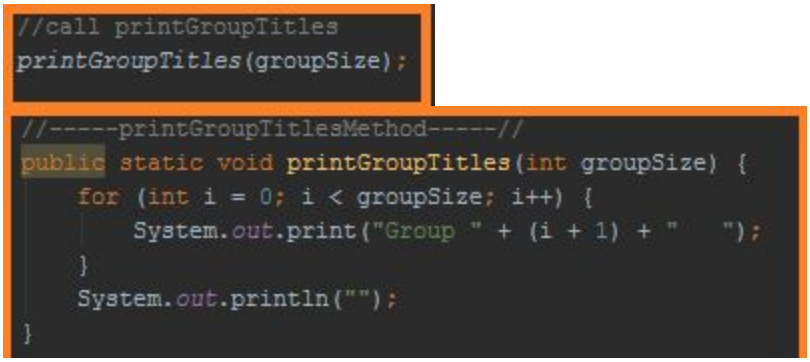
c.)

```
//construct studentArray
for (int i = 1; i <= Integer.parseInt(studentsString); i++) {
    studentArray.add(i);
}

//randomize randomArray
for (int i = 1; i <= Integer.parseInt(studentsString); i++) {
    random = rng.nextInt(studentArray.size());
    randomArray.add(studentArray.get(random));
    studentArray.remove(random);
}
```

The above section of code features the two core algorithms of my program. The for loop labeled //construct studentArray adds students to an array list for the number of students enterede by the user. The loop below that section of code, labeled //randomize randomArray, pulls random values from studentArray, which we just constructed with a for loop. This process continues until all the values from studentArray have been put into randomArray in a random sequence. This interaction between these two algorithms serves as the core of this program's functionality, but both of the algorithms/loops can stand on their own. They both perform their individual process, but work together to allow the program to function.

d.)



```
//call printGroupTitles
printGroupTitles(groupSize);
```

```
//-----printGroupTitlesMethod-----//
public static void printGroupTitles(int groupSize) {
    for (int i = 0; i < groupSize; i++) {
        System.out.print("Group " + (i + 1) + "    ");
    }
    System.out.println("");
}
```

In my program, abstraction takes the form of a Java method. Specifically, the method printGroupTitles is called to make use of an algorithm that generates as many group headings as needed. The method printGroupTitles is abstraction because it is merely a label that represents a functional algorithm that is integral to the functionality of the program. The Java method is called right before the array of randomly ordered students is printed, so that the groups may be labeled. The method is able to function because it requests a specific integer variable, groupSize, which tells it exactly how many headings to print. This process is represented by the syntax "public static void printGroupTitles(int groupSize)". This helps to manage the complexity of the program by helping to organize the printing process. The part of the program that outputs the formed groups already has to account for whether or not the student number is a two-digit or single-digit number, and when a new row should be started. Although the groups are organized vertically, the program itself prints the random values horizontally, periodically starting new rows.