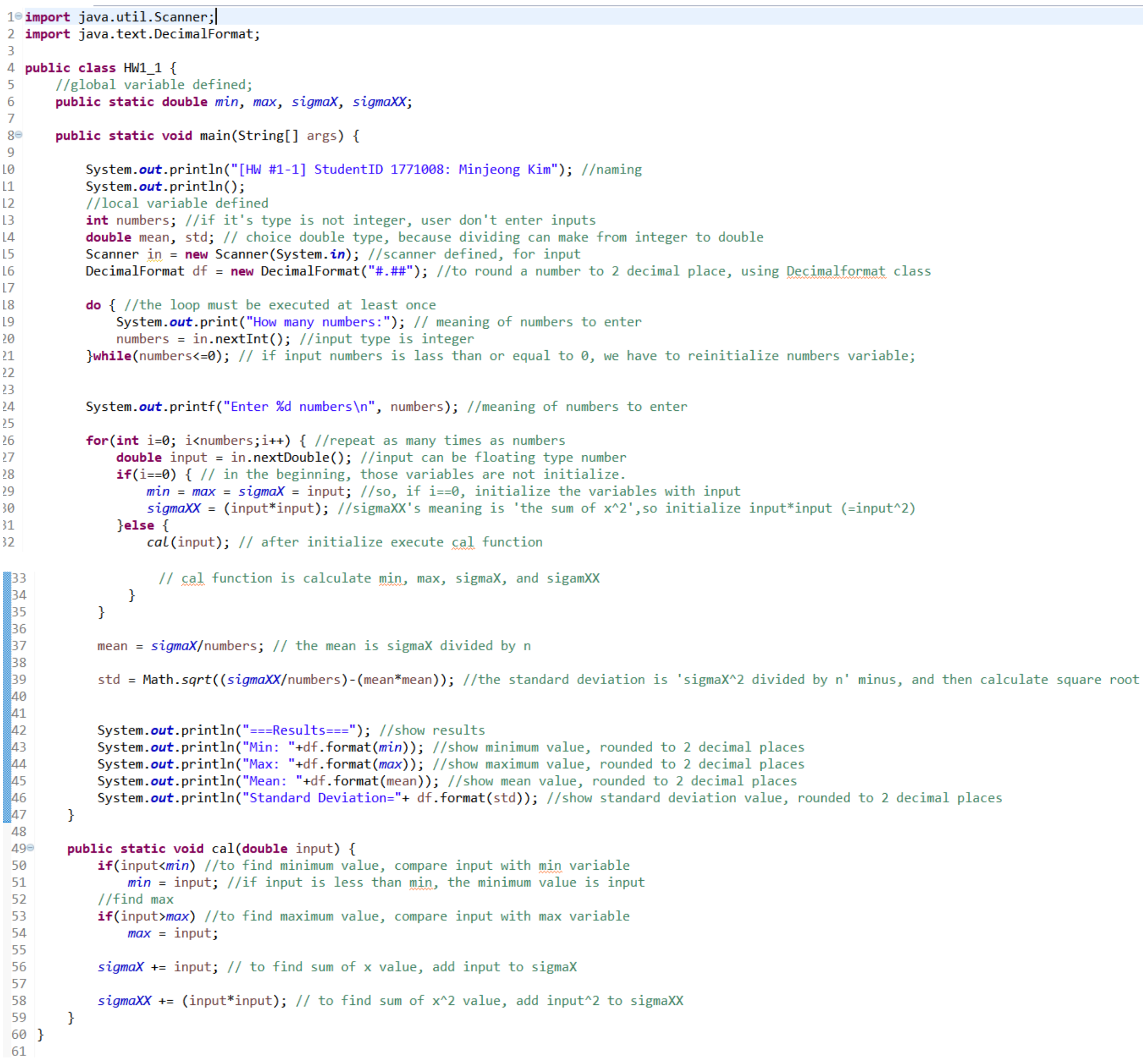
REPORT

JAVA Programming and Labs

Student ID : 1771008
Name : Minjeong Kim

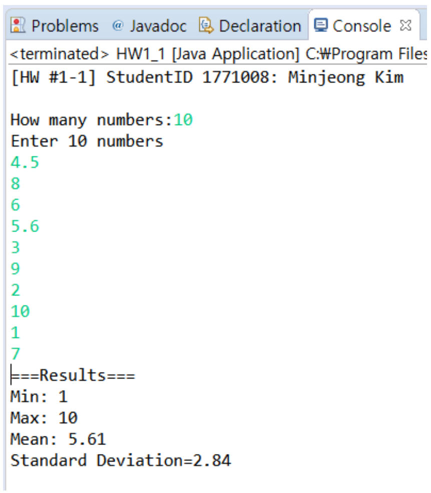
HW1_1

[Code]



```java
import java.util.Scanner;
import java.text.DecimalFormat;

public class HW1_1 {
    //global variable defined;
    public static double min, max, sigmaX, sigmaXX;

    public static void main(String[] args) {

        System.out.println("[HW #1-1] StudentID 1771008: Minjeong Kim"); //naming
        System.out.println();
        //local variable defined
        int numbers; //if it's type is not integer, user don't enter inputs
        double mean, std; // choice double type, because dividing can make from integer to double
        Scanner in = new Scanner(System.in); //scanner defined, for input
        DecimalFormat df = new DecimalFormat("#.##"); //to round a number to 2 decimal place, using Decimalformat class

        do { //the loop must be executed at least once
            System.out.print("How many numbers:"); // meaning of numbers to enter
            numbers = in.nextInt(); //input type is integer
        }while(numbers<=0); // if input numbers is lass than or equal to 0, we have to reinitialize numbers variable;


        System.out.printf("Enter %d numbers\n", numbers); //meaning of numbers to enter

        for(int i=0; i<numbers;i++) { //repeat as many times as numbers
            double input = in.nextDouble(); //input can be floating type number
            if(i==0) { // in the beginning, those variables are not initialize.
                min = max = sigmaX = input; //so, if i==0, initialize the variables with input
                sigmaXX = (input*input); //sigmaXX's meaning is 'the sum of x^2',so initialize input*input (=input^2)
            }else {
                cal(input); // after initialize execute cal function

                // cal function is calculate min, max, sigmaX, and sigamXX
            }
        }

        mean = sigmaX/numbers; // the mean is sigmaX divided by n

        std = Math.sqrt((sigmaXX/numbers)-(mean*mean)); //the standard deviation is 'sigmaX^2 divided by n' minus, and then calculate square root


        System.out.println("===Results==="); //show results
        System.out.println("Min: "+df.format(min)); //show minimum value, rounded to 2 decimal places
        System.out.println("Max: "+df.format(max)); //show maximum value, rounded to 2 decimal places
        System.out.println("Mean: "+df.format(mean)); //show mean value, rounded to 2 decimal places
        System.out.println("Standard Deviation="+ df.format(std)); //show standard deviation value, rounded to 2 decimal places
    }

    public static void cal(double input) {
        if(input<min) //to find minimum value, compare input with min variable
            min = input; //if input is less than min, the minimum value is input
        //find max
        if(input>max) //to find maximum value, compare input with max variable
            max = input;

        sigmaX += input; // to find sum of x value, add input to sigmaX

        sigmaXX += (input*input); // to find sum of x^2 value, add input^2 to sigmaXX
    }
}
```

[Output console]



```
<terminated> HW1_1 [Java Application] C:₩Program Files
[HW #1-1] StudentID 1771008: Minjeong Kim

How many numbers:10
Enter 10 numbers
4.5
8
6
5.6
3
9
2
10
1
7
===Results===
Min: 1
Max: 10
Mean: 5.61
Standard Deviation=2.84
```

[Code analysis]

I set the global variables as min, max, sigmaX, sigmaXX. The user repeats the input until numbers variable value. Each time an input is entered, those global variables are used via the cal() function. In the cal() function, each global variables value are updated. In the cal() function, If the min value is greater than the input, then the min value is updated with input value. And if the max value is less than the input, then the max value is updated with input value. The value of input is added to the sigmaX variable, because we have to calculate the sum of input value to find the mean value. With same context, the square of input value added to the sigmaXX variable, since we have to calculate the sum of the square of input value to find the standard Deviation value.

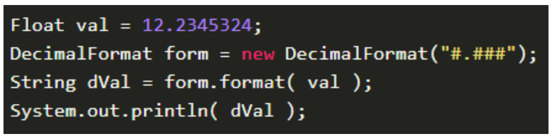
After the repeat, we calculate the mean value using the $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$ formula. In this formula, $\sum_{i=1}^{N} x_i$ is equal to the value of sigmaX. Therefore we get the mean value by dividing sigmaX by numbers. Also we can calculate standard deviation value by using the $\sigma = \sqrt{\left(\frac{1}{N}\sum_{i=1}^{N} x_i^2\right) - \mu^2}$ formula. In this formula, $\sum_{i=1}^{N} x_i^2$ is equal to the value of sigmaXX, and μ^2 is the same as the squared value of the mean value. So, I assigned the calculation result of Math.sqrt((sigmaXX/numbers)-(mean*mean)) to std variable.

Finally print each value by println() for the ouput format.

[What I learned from this homework]

First, I learned about how to round a number to n decimal place

DecimatlFormat class's purpose is to display the number of decimal places. Depending on the format , the decimal point is rounded. And the result is a string, not a numeric type. I will explain how to use it



```
Float val = 12.2345324;
DecimalFormat form = new DecimalFormat("#.###");
String dVal = form.format( val );
System.out.println( dVal );
```

Set a variable of type DecimalFormat and create an object by using new operator. Inside the bracket, we must set the specific pattern. The pattern is setting by using '0' or '#'. In the case of '0', the value of the corresponding digit must be present. And the case of '#', it rounds to the decimal place. For example, the case of the value is 78.53981. '0.###' : 78.54 , '000.##' : 078.54 , '00.#' : 78.5. After setting pattern, we call '.format' method. And in the method's bracket, enter a real number. The must know that the return type is String, when assign the result value to a variable.

Second, I make a clear understanding for static.

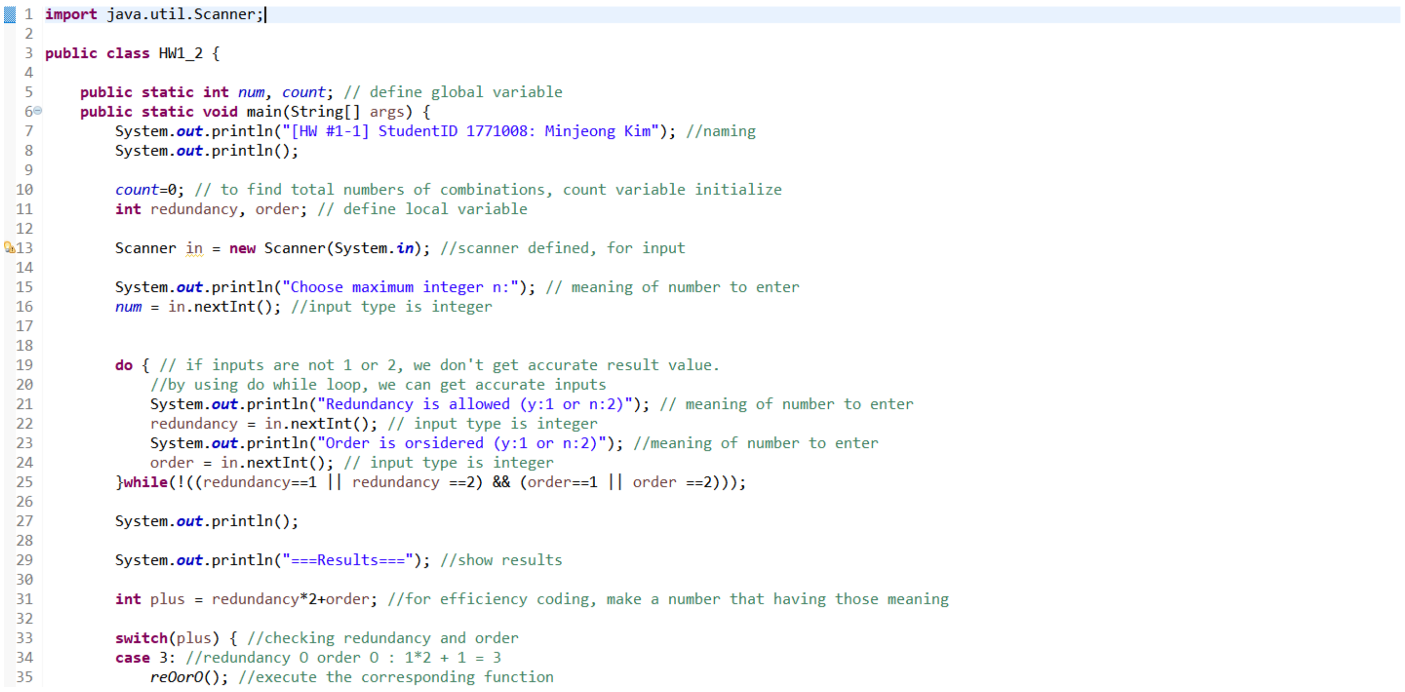
Static methods or variables are defined as soon as Java is compiled. So, it is impossible to calling non-static object, in static object. if I don't define a Class object, we don't use the method that non-static, even though same class. so, I can use the cal() method even if I didn't declare the calls object, since static.

If you set the variable to static, you allocate memory only once, so you can use it for sharing purpose. The sharing allow you to use variable as global variables.

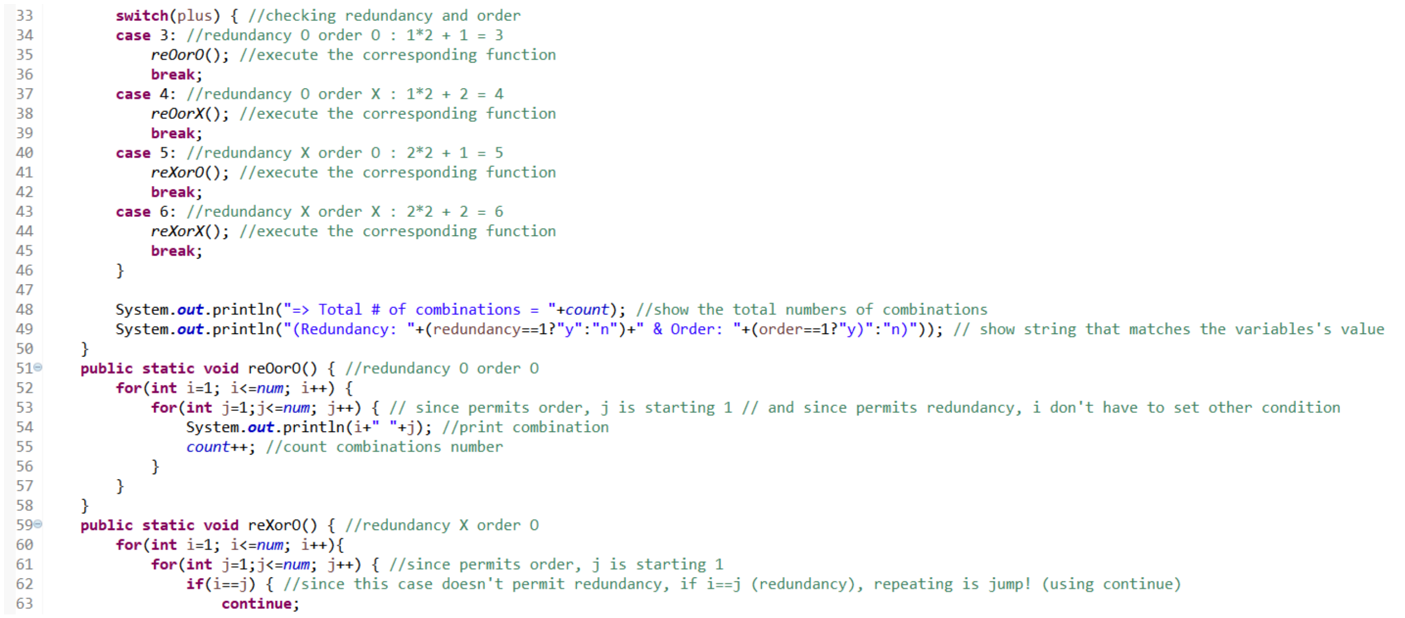
With this homework, I was able to see the concept of JAVA more clearly

HW1_2

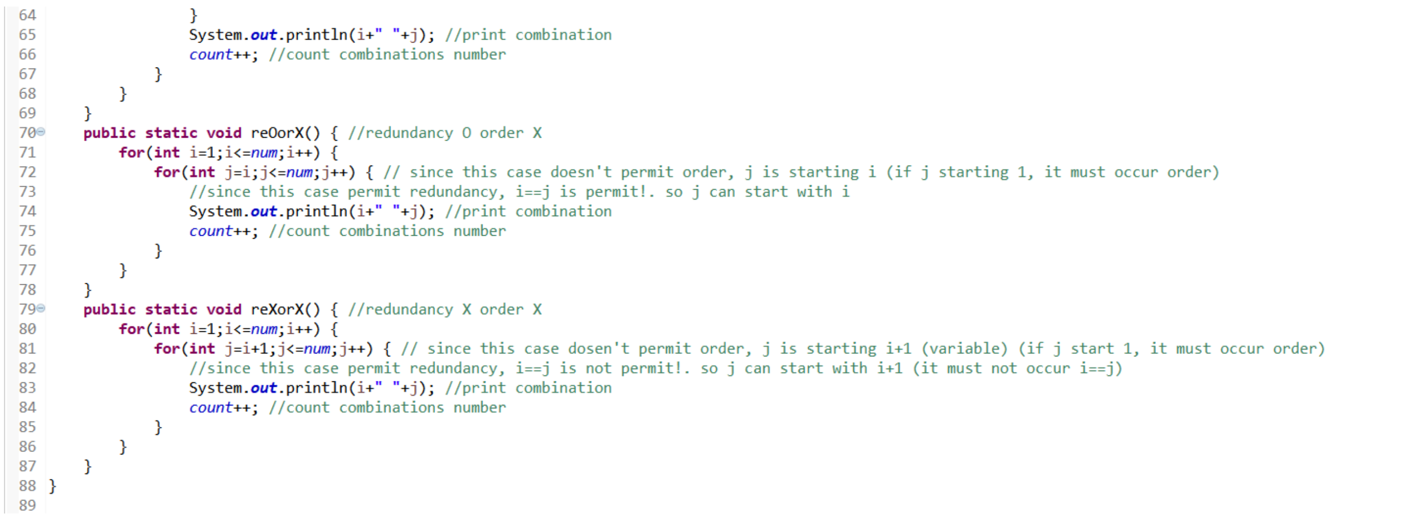
[Code]



```java
import java.util.Scanner;

public class HW1_2 {

    public static int num, count; // define global variable
    public static void main(String[] args) {
        System.out.println("[HW #1-1] StudentID 1771008: Minjeong Kim"); //naming
        System.out.println();

        count=0; // to find total numbers of combinations, count variable initialize
        int redundancy, order; // define local variable

        Scanner in = new Scanner(System.in); //scanner defined, for input

        System.out.println("Choose maximum integer n:"); // meaning of number to enter
        num = in.nextInt(); //input type is integer


        do { // if inputs are not 1 or 2, we don't get accurate result value.
            //by using do while loop, we can get accurate inputs
            System.out.println("Redundancy is allowed (y:1 or n:2)"); // meaning of number to enter
            redundancy = in.nextInt(); // input type is integer
            System.out.println("Order is orsidered (y:1 or n:2)"); //meaning of number to enter
            order = in.nextInt(); // input type is integer
        }while(!((redundancy==1 || redundancy ==2) && (order==1 || order ==2)));

        System.out.println();

        System.out.println("===Results==="); //show results

        int plus = redundancy*2+order; //for efficiency coding, make a number that having those meaning

        switch(plus) { //checking redundancy and order
        case 3: //redundancy O order O : 1*2 + 1 = 3
            reOorO(); //execute the corresponding function
```

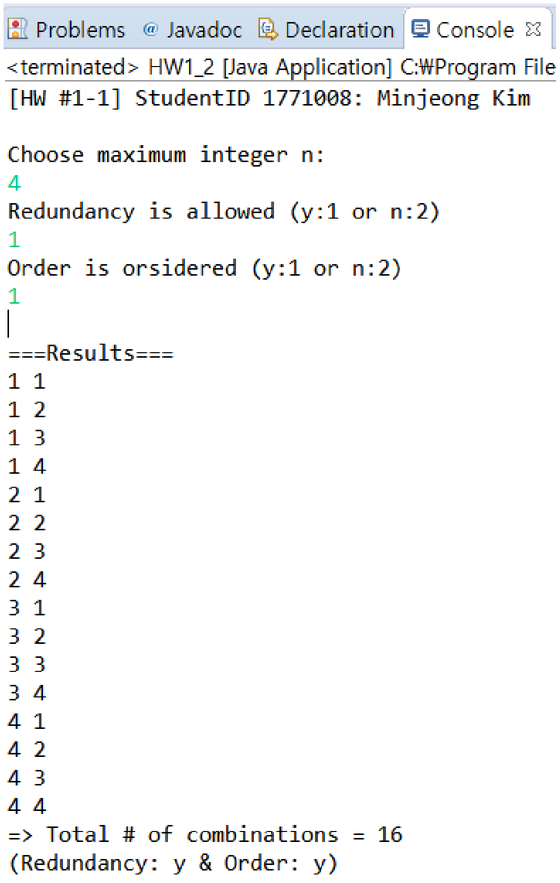


```java
        switch(plus) { //checking redundancy and order
        case 3: //redundancy O order O : 1*2 + 1 = 3
            reOorO(); //execute the corresponding function
            break;
        case 4: //redundancy O order X : 1*2 + 2 = 4
            reOorX(); //execute the corresponding function
            break;
        case 5: //redundancy X order O : 2*2 + 1 = 5
            reXorO(); //execute the corresponding function
            break;
        case 6: //redundancy X order X : 2*2 + 2 = 6
            reXorX(); //execute the corresponding function
            break;
        }

        System.out.println("=> Total # of combinations = "+count); //show the total numbers of combinations
        System.out.println("(Redundancy: "+(redundancy==1?"y":"n")+" & Order: "+(order==1?"y)":"n)")); // show string that matches the variables's value
    }
    public static void reOorO() { //redundancy O order O
        for(int i=1; i<=num; i++) {
            for(int j=1;j<=num; j++) { // since permits order, j is starting 1 // and since permits redundancy, i don't have to set other condition
                System.out.println(i+" "+j); //print combination
                count++; //count combinations number
            }
        }
    }
    public static void reXorO() { //redundancy X order O
        for(int i=1; i<=num; i++){
            for(int j=1;j<=num; j++) { //since permits order, j is starting 1
                if(i==j) { //since this case doesn't permit redundancy, if i==j (redundancy), repeating is jump! (using continue)
                    continue;
```

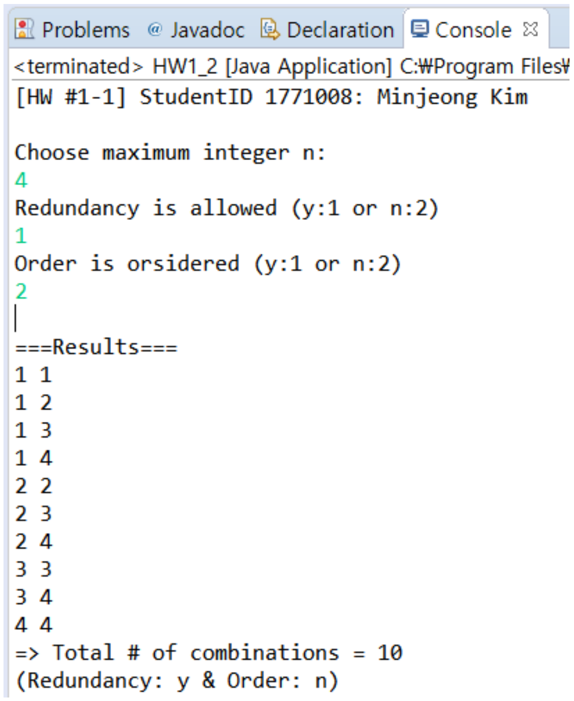


```java
                }
                System.out.println(i+" "+j); //print combination
                count++; //count combinations number
            }
        }
    }
    public static void reOorX() { //redundancy O order X
        for(int i=1;i<=num;i++) {
            for(int j=i;j<=num;j++) { // since this case doesn't permit order, j is starting i (if j starting 1, it must occur order)
                //since this case permit redundancy, i==j is permit!. so j can start with i
                System.out.println(i+" "+j); //print combination
                count++; //count combinations number
            }
        }
    }
    public static void reXorX() { //redundancy X order X
        for(int i=1;i<=num;i++) {
            for(int j=i+1;j<=num;j++) { // since this case dosen't permit order, j is starting i+1 (variable) (if j start 1, it must occur order)
                //since this case permit redundancy, i==j is not permit!. so j can start with i+1 (it must not occur i==j)
                System.out.println(i+" "+j); //print combination
                count++; //count combinations number
            }
        }
    }
}

```

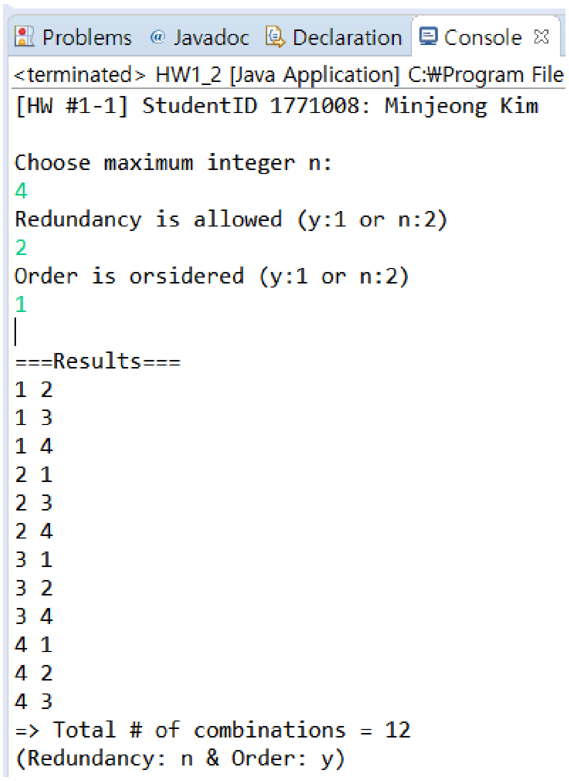
[Output console]



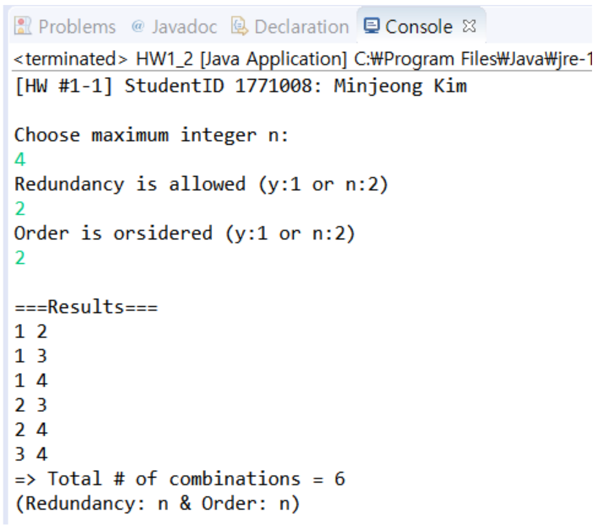
```
<terminated> HW1_2 [Java Application] C:₩Program File
[HW #1-1] StudentID 1771008: Minjeong Kim

Choose maximum integer n:
4
Redundancy is allowed (y:1 or n:2)
1
Order is orsidered (y:1 or n:2)
1

===Results===
1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
3 1
3 2
3 3
3 4
4 1
4 2
4 3
4 4
=> Total # of combinations = 16
(Redundancy: y & Order: y)
```



```
<terminated> HW1_2 [Java Application] C:₩Program Files₩
[HW #1-1] StudentID 1771008: Minjeong Kim

Choose maximum integer n:
4
Redundancy is allowed (y:1 or n:2)
1
Order is orsidered (y:1 or n:2)
2

===Results===
1 1
1 2
1 3
1 4
2 2
2 3
2 4
3 3
3 4
4 4
=> Total # of combinations = 10
(Redundancy: y & Order: n)
```



```
<terminated> HW1_2 [Java Application] C:₩Program File
[HW #1-1] StudentID 1771008: Minjeong Kim

Choose maximum integer n:
4
Redundancy is allowed (y:1 or n:2)
2
Order is orsidered (y:1 or n:2)
1

===Results===
1 2
1 3
1 4
2 1
2 3
2 4
3 1
3 2
3 4
4 1
4 2
4 3
=> Total # of combinations = 12
(Redundancy: n & Order: y)
```



```
<terminated> HW1_2 [Java Application] C:₩Program Files₩Java₩jre-1
[HW #1-1] StudentID 1771008: Minjeong Kim

Choose maximum integer n:
4
Redundancy is allowed (y:1 or n:2)
2
Order is orsidered (y:1 or n:2)
2

===Results===
1 2
1 3
1 4
2 3
2 4
3 4
=> Total # of combinations = 6
(Redundancy: n & Order: n)
```

[Code analysis]

I analyzed four cases depending on the presence or absence of redundancy and order. After receiving the input for the redundancy and order values, I calculate those value to separate into four case, and the variable to store each case is plus. The formula is plus = redundancy*2 + order, to understand easier, let's look at the table below.

	Redundancy O (1*2)	Redundancy X (2*2)
Order O (1)	Plus = 3 (1*2+1=3)	Plus = 5 (2*2+1=5)
Order X (2)	Plus = 4 (1*2+2=4)	Plus = 6 (2*2+2=6)

And use the switch statement to perform the operation according to the plus value. In each case, the form of the variable in the for statement is slightly different, so I classified into functions corresponding to four cases. Let, analyze each case.

First, case 3 that redundancy, and order. I'll see the results first, [1 1] [1 2] [1 3] [1 4] / [2 1] [2 2] [2 3] [2 4] / [3 1] [3 2] [3 3] [3 4] / [4 1] [4 2] [4 3] [4 4]. It can be seen that I and j are repeated four times. So repeat all four times in the nested root.

Second, case 4 that redundancy, and not order. The results are [1 1] [1 2] [1 3] [1 4] / [2 2] [2 3] [2 4] / [3 3] [3 4] / [4 4]. The nested for loop's the number of iterations is reduced by one. And the numbers starting at each iteration has been increased by one. So the i variable is repeat 1 to num(num), and j variable is repeat i to num.

Third, case 5 that not redundancy, and order. The results are [1 2] [1 3] [1 4] / [2 1] [2 3] [2 4] / [3 1] [3 2] [3 4] / [4 1] [4 2] [4 3]. This is the same as except for cases where I and j are equal in the case3 loop. So use continue to not print when i and j are qual.

Finally, case 6 that not redundancy, and not order. The results are [1 2] [1 3] [1 4] / [2 3] [2 4] / [3 4]. The nested for loop's number of iterations is reduced by one. And the number starting at each iteration has been increased by one, and the j value is on greater than i. So the I variable is repeat 1 to num, and j variable is repeat i+1 to num.

And to count the number of combinations, increate the value of the count variable by 1 in each iteration of the function. Then print count value, and print redundancy and order value in string by using conditional statement.

[What I learned from this homework]

In this homework, I use '?' operator, Ternary operator, to write conditional statements shortly! In simple cases like if-else conditional statement, replacing with '? operator' makes the code concise. The format is below

(boolean) ? c(case true) : d(case false) ;

If Boolean condition is true, c is executed, otherwise d is executed. The important point is that the statement (ex. System.out.println()) doesn't work, if it comes after the '? operator'. After the '? operator', we must be used String, Boolean, Numeric type.

Second, I noticed the caution when using the while loop. when using the conditional statement in while(or do-while) loop, the results have changed according to the parentheses. I realized that I needed to clarify the meaning of programming by putting parentheses in the formula.