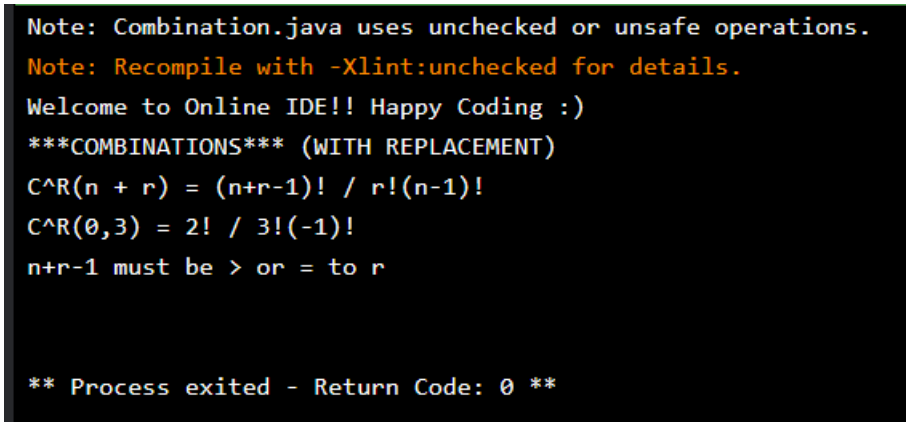
********** OUTPUT **********

```
Note: Combination.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(0,0) = -1! / 0!(-1)!
n and r can not both be equal to zero


** Process exited - Return Code: 0 **
```

```
Note: Combination.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(5,2) = 6! / 2!(4)!
15


** Process exited - Return Code: 0 **
```

```
Note: Combination.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***COMBINATIONS*** (WITH REPLACEMENT)
C^R(n + r) = (n+r-1)! / r!(n-1)!
C^R(2,5) = 6! / 5!(1)!
6


** Process exited - Return Code: 0 **
```



// *** CODE **

/*

Online Java - IDE, Code Editor, Compiler

Online Java is a quick and easy tool that helps you to build, compile, test your programs

online.

*/

// This has been created to ensure I can utilize any random functions more efficiently.

// It is a creation of the combinations (with replacement) calculator.

// It has used techniques I learnt including recursion and also memoization to speed up execution.

// I will incorporate this into Java applications I created previously..

//TEST CASES

//r=2 n=5 PASS

//r=5 n=5 PASS

//r=1 n=4 PASS

//r=0 n=3 PASS

//r=0 n=0 PASS

// now going to flip the above

//r=5 n=2 PASS

//r=5 n=5 PASS

```
//r=4  n=1    PASS
//r=3  n=0    FAIL.... FIXED

//test to make numerator less than r
// n = 4    r=3  FAIL     n+r-1 must be > or = to r   (FIXED)



import java.math.*;
import java.util.*;

public class Combination
{
   public static void main(String[] args) {
   System.out.println("Welcome to Online IDE!! Happy Coding :)");
   int originalNumber=0;
   int n=originalNumber;
   int r =3;
   Map <Integer, Long> m = new HashMap<>();
   System.out.println("***COMBINATIONS*** (WITH REPLACEMENT)");
   System.out.println("C^R(n + r) = " + "(n+r-1)! / r!(n-1)!");
   System.out.println("C^R(" + n+","+r+") = " + (n+r-1)+ "!" + " / " + r+"!"+"("+(n-1)+")!");
   System.out.println(Combinations (n,r,originalNumber, m));
}
public static long Combinations (int n, int r, int originalNumber, Map factorialResults)
{
   // n are objects
   // r is sample

   /*
   ***CALCULATION***
   (n+r-1)! / r!(n-1)!
```

```
   */
   long result=0;
   int denominator1; //denominator split two parts since there are two factorial calculations
   int denominator2; //denominator split two parts since there are two factorial calculations
   int Numerator=n+r-1; // Numerator
   int zero=0;    // this will be used to create entry in Map for 0!
   long zeroFactorial = 1;   //0! equals 1

   // if no sample or objects, there are no outcomes...
   if (originalNumber==0 && r==0)
   {
      System.out.println("n and r can not both be equal to zero");
      System.exit(0);
      return 0;
   }

   //this situation would occur if n is 0 only and r is any positive number accept 0 (if statement
above)
   //for instance  (C^R (n,r))  = (0,3)    0+3-1  = 2    2<3

   if (originalNumber==0 && originalNumber+r-1<r)
   {
      System.out.println("n+r-1 must be > or = to r");
      System.exit(0);
      return 0;
   }



   if (Numerator>=1)
// this will ensure that all factorials as low as 1! are processed
```

```
//reason for this is since   C^R(n,r) for instance C^R(1,0)

//(n+r-1)  =  0

// This does not seem like an issue but since the numerator is calculated as follows:

//result = ((n+r-1)* (Combinations (n-1, r,originalNumber, factorialResults)));

// it can be seen that Java will not be content with 0 * another number.....

// As an offset, since the denominator relies on mapped values for numerator, there will be no entry in the map for

//0!.   The only way to overcome this is to put an entry manually for 0! = 1...


  {

     //System.out.println("value of n: " + Numerator);

     // EXAMPLE

     // C^R (5,6) = (5+6-1)! / 6! (5-1)! = 3628800 / (6! * 4!) = 3628800 / 720 * 24 = 210

     result = ((n+r-1)* (Combinations (n-1, r,originalNumber, factorialResults))); // this

     //completes factorial for numerator

     factorialResults.put(Numerator,result); //result stored in the Map

     //factorialResults.put(n-1,result); //result stored in the Map

     //System.out.println("getting result back out numerator: " + (Numerator) + " " + factorialResults.get(n+r-1));


     if (n==originalNumber) // this will occur once

     {

        denominator1 = r; // r sample size has not changed

        // originalNumber required since n has reduced as part of the recursive calls

        denominator2 = originalNumber-1;

        // this is using the Java Memoization technique to ensure the factorial outcome is not calculated again, to save program cycles.

        // since the returns are done in reverse order.... n = 1 is processed first and n=6 last... Hence in practice

        // there will be entry in Map for all factorials, ready for the denominator..

        //n+r-1 r=6 n = 3 or 2 or 1 so recursive values going in set are 8! , 7!, 6! factorial

        // but the put method for the set would recursively call and populate others up to 1!
```

```
        // this is where it currently fails for cases such as  C^R  (3,0), (1,0)
        //reason is since it would have not created an entry for 0! at any point in time
        //entry being created now

        factorialResults.put(zero,zeroFactorial);    //0!  is equal to 1

        //System.out.println("den1:" + denominator1);
        //System.out.println("den2:" + denominator2);

        if (factorialResults.containsKey(denominator1) &&
factorialResults.containsKey(denominator2))
        {
           //System.out.println("here");
           //System.out.println("This is exact value of factorial denominator part1 " +
(denominator1) + " : " + factorialResults.get(denominator1));
           //System.out.println("This is exact value of factorial denominator part2 " +
(denominator2) + " : " + factorialResults.get(denominator2));

           long returnValue = result /  ((long)factorialResults.get(denominator1) *
(long)factorialResults.get(denominator2));
           return returnValue;
        }
     }
     return result;
  }

  return 1; // it will reach here only when condition not met (Numerator>=1)
}
}
```