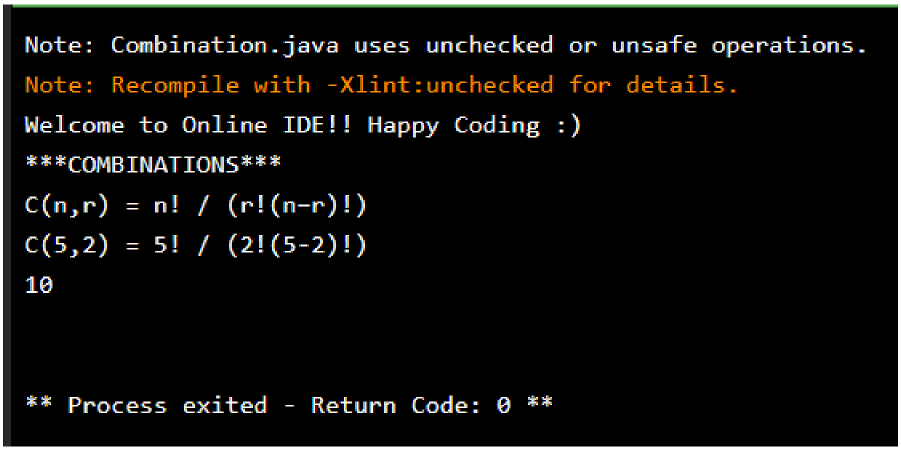
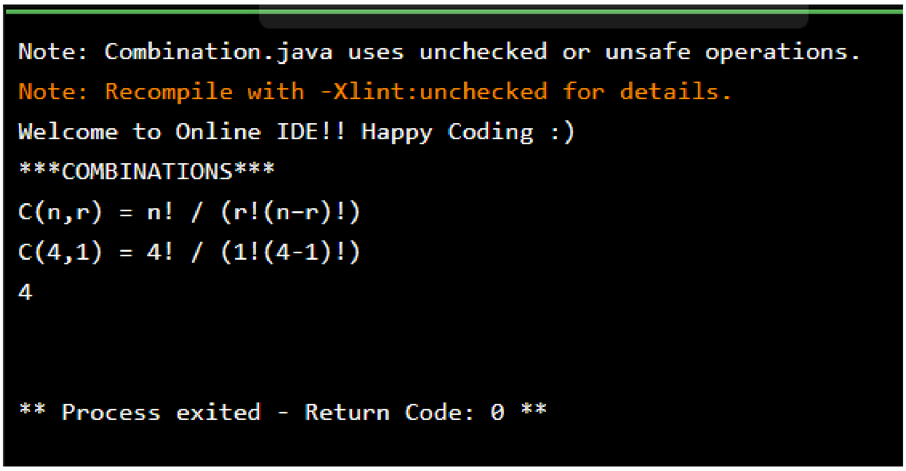
************ OUTPUT ************************



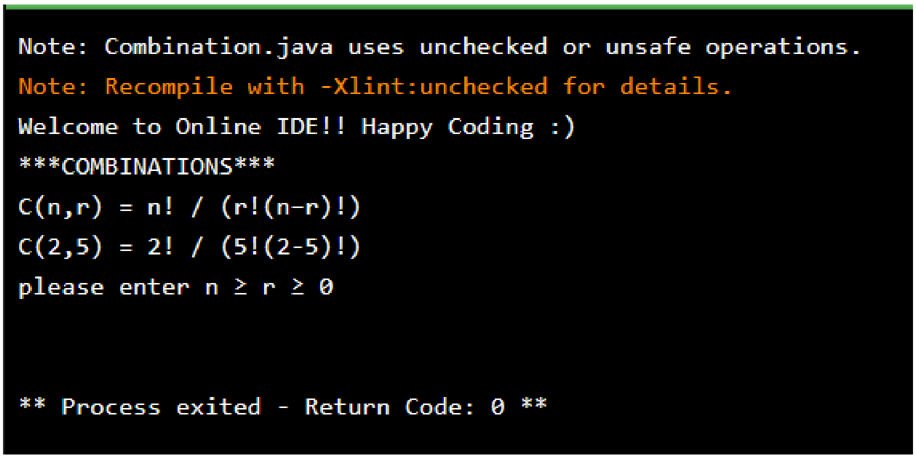
```
Note: Combination.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***COMBINATIONS***
C(n,r) = n! / (r!(n-r)!)
C(5,2) = 5! / (2!(5-2)!)
10


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



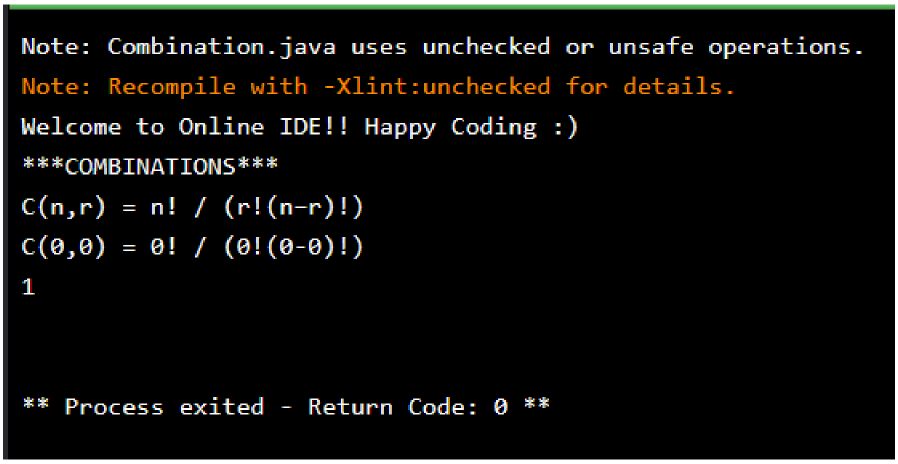
```
Note: Combination.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***COMBINATIONS***
C(n,r) = n! / (r!(n-r)!)
C(4,1) = 4! / (1!(4-1)!)
4


** 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***
C(n,r) = n! / (r!(n-r)!)
C(2,5) = 2! / (5!(2-5)!)
please enter n ≥ r ≥ 0


** 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***
C(n,r) = n! / (r!(n-r)!)
C(0,0) = 0! / (0!(0-0)!)
1


** 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 NcR combinations 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   //PASS

//test to make numerator less than r
// n = 4     r=3  //PASS

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=5;
    int n=originalNumber;
    int r =5;
    Map <Integer, Long> m = new HashMap<>();
    System.out.println("***COMBINATIONS***");
    System.out.println("C(n,r) = n! / (r!(n−r)!)");
    System.out.println("C(" + n+","+r+") = " + n+"!" + " / " + "("+r+"!"+"("+n+"-"+r+")!)");
    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***
  P(n,r) = n! / (r!(n−r)!)
  */
  long result=0;
  int denominator1;
  int denominator2;
  int zero=0;    // this will be used to create entry in Map for 0!
  long zeroFactorial = 1;   //0! equals 1
  //this is example scenario   C(n,r)  = C(2,5)
  if (r>originalNumber|| r<0)
  {
    System.out.println("please enter n ≥ r ≥ 0");
    System.exit(0);
    return 0;
  }

  // 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(0,3)
  // This will become issue since numerator is calculated as follows:
  //result = (n* (Combinations (n-1, r,originalNumber, factorialResults))); // this completes factorial for numerator
  // 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...

  if (n>=1)
  {
    // EXAMPLE
    // P (5,6) = 6* 5* 4 * 3 * 2 * 1 / 6! (6-5)! = 720 / (5! * 1!) = 120 / 5*4*3*2*1 * 1 = 720 / 120 = 6

    result = (n* (Combinations (n-1, r,originalNumber, factorialResults))); // this completes factorial for numerator

    factorialResults.put(n,result); //result stored in the Map
    //System.out.println("getting result back out numerator " + n+": " + factorialResults.get(n));

    if (n==originalNumber) // this will occur once
    {
      denominator1 = originalNumber-r;
      // originalNumber required since n has reduced as part of the recursive calls

      denominator2 = r; // r sample size has not changed

      // 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..


       // this is where it currently fails for cases such as  C^R  (5,5), (3,0)

      //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 " + (denominator1) + " : " +
factorialResults.get(denominator1));

        //System.out.println("This is exact value of factorial " + (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 when this condition is not met (n>=1)
```

```
}

}
```