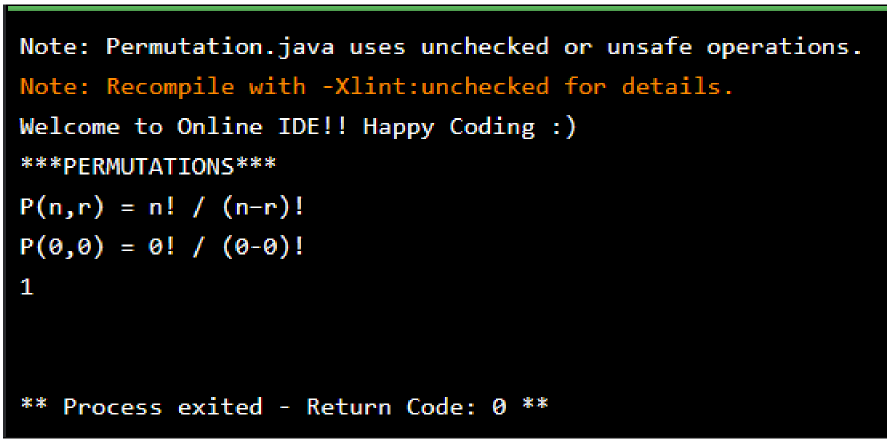
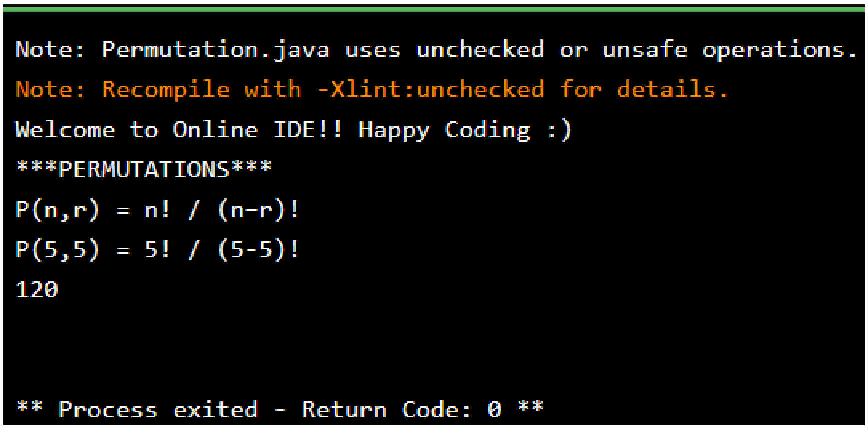
************ OUTPUT ************************



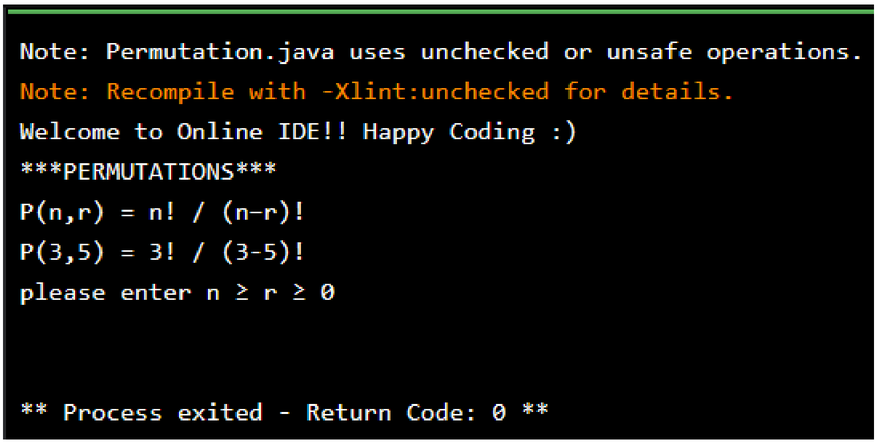
```
Note: Permutation.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***PERMUTATIONS***
P(n,r) = n! / (n-r)!
P(0,0) = 0! / (0-0)!
1


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



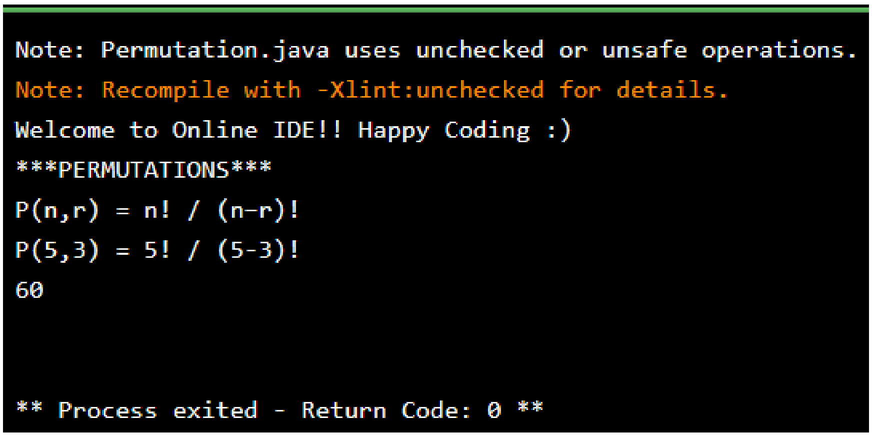
```
Note: Permutation.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***PERMUTATIONS***
P(n,r) = n! / (n-r)!
P(5,5) = 5! / (5-5)!
120


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



```
Note: Permutation.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***PERMUTATIONS***
P(n,r) = n! / (n-r)!
P(3,5) = 3! / (3-5)!
please enter n ≥ r ≥ 0


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



```
Note: Permutation.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Welcome to Online IDE!! Happy Coding :)
***PERMUTATIONS***
P(n,r) = n! / (n-r)!
P(5,3) = 5! / (5-3)!
60


** 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 nPr permutation 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


//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 Permutation

{

   public static void main(String[] args) {

   System.out.println("Welcome to Online IDE!! Happy Coding :)");

   int originalNumber=4;

   int n=originalNumber;

   int r =3;

   Map <Integer, Long> m = new HashMap<>();

   System.out.println("***PERMUTATIONS***");

   System.out.println("P(n,r) = n! / (n−r)!");

   System.out.println("P(" + n+","+r+") = " + n+"!" + " / " + "("+n+"-"+r+")!");

   System.out.println(Permutations (n,r,originalNumber, m));

}


public static long Permutations (int n, int r, int originalNumber, Map factorialResults)

{

   // n are objects

   // r is sample

   /*

   ***CALCULATION***

   P(n,r) = n! / (n−r)!
```

```
  */
  long result=0;
  int temp;
  int denominator;



  if (originalNumber<r || r<0)
  {
    System.out.println("please enter n ≥ r ≥ 0");
    System.exit(0);
    return 0;
  }

  if (n>=1)
  {
    // EXAMPLE
    // P (5,6) = 5* 4 * 3 * 2 * 1 / (6-5)! = 24 / 2! = 24 / 2 * 1 = 24/2 = 12
    result = (n* (Permutations (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
  {
    denominator = originalNumber-r; // originalNumber required since n has reduced as part of the recursive calls
    //System.out.println("This is denominator: " + denominator);
    // this is using the Java Memoization technique to ensure the factorial outcome is not calculated again, to save program execution 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..
    if (factorialResults.containsKey(denominator))

    {

      //System.out.println("here");

      //System.out.println("This is exact value of factorial denominator " + (denominator) + " : " + factorialResults.get(denominator));

      return result / (long)factorialResults.get(denominator); // this is number permutations

    }

  }

  return result; // this will be returning already calculating numerator part

  }

  return 1; // // it should reach here if this is false: (n>=1) }

}

}
```