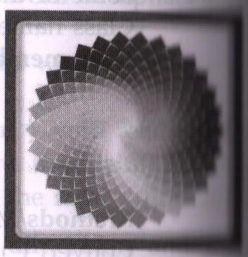
# 11 RECURSION

**In this chapter**

**Introduction, Recursive function, Base case and Recursive case, Advantages and limitations of Recursive function, Types of recursions; Direct Recursion: Tail recursion, Binary recursion, Indirect Recursion, Differences between Recursion and Iterations**

## Introduction



We have already come across functions/methods in the previous chapters, which are program modules designed for specific purposes. A function is basically used in the program to perform similar operations at different instances. It may be returnable or non-returnable depending upon our requirements.

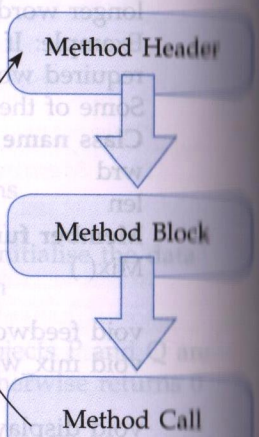
*Recursion is a programming technique to call a function from its block.* It is essentially important to know recursion because some programs can be written only by using recursive technique. Recursive technique allows the user to design the program logic by using minimum number of statements. Hence, the program code becomes clear and easy to understand.

## Recursive Function

A function that is written by using recursive technique is known as 'Recursive Function'. It calls itself (i.e. the function is called from its body or the same block).

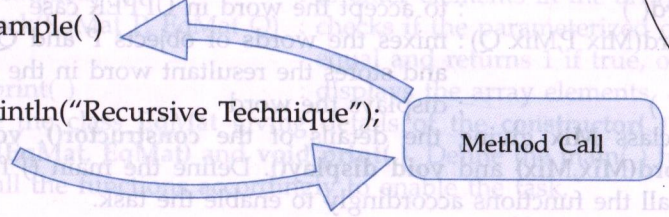
### Syntax:

```
<Access Specifier> <Return Type> <Method Name>(Parameters)
{
Method Block
.
.
<Call> <Method Name> (Parameter)
}
```




**Example 1:**

```
public void sample( )
{
System.out.println("Recursive Technique");
sample();
}
```



The recursive function shown in the example calls itself infinitely. It will keep displaying the message 'Recursive Technique' unless 'java.lang.StackOverflowError' appears. Let us take another example to understand the way to design a finite recursive function.

**Example 2:**

```
static int count = 0;
public void sample( )
{
count++;
if(count < 5)
{
System.out.println("Recursive Technique");
sample( );
}
}
```

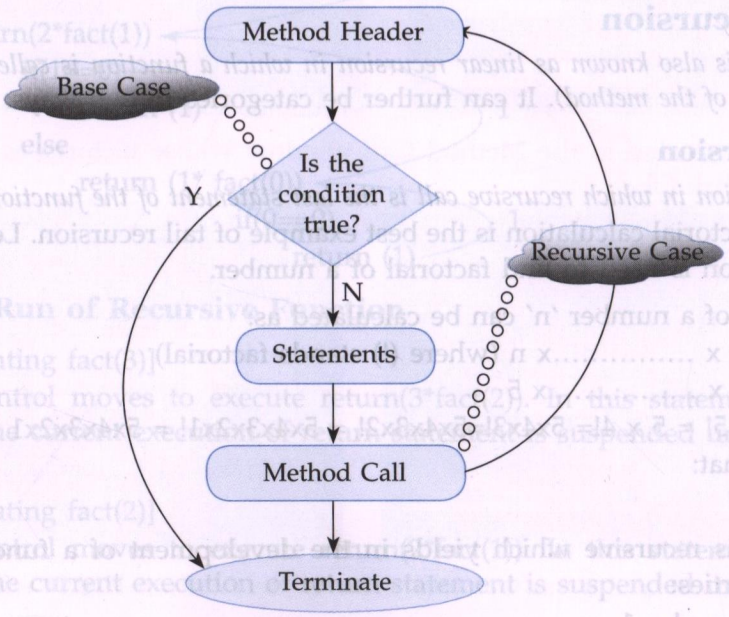
**Output:**

```
Recursive Technique
Recursive Technique
Recursive Technique
Recursive Technique
```

In the example shown above, the number of function calls is managed with the help of a counter (i.e. count). As soon as the counter value reaches to 5, the control stops calling the function any more. Hence, the message 'Recursive Technique' will be displayed only four times.

## Base Case vs Recursive Case

A recursive function makes the code small and clear. Every recursive function uses a condition to terminate. It is known as 'Base Case'. As soon as the condition for base case is true, the control terminates the function otherwise, it keeps calling the function again and again. The statement that calls the function repeatedly from its block is said to be the 'Recursive Case'. Let us see the recursive nature of function call with the help of a flow chart shown below:




*Flow Diagram of Recursive Process*

Hence, Base Case is a situation or condition that is used to terminate a recursive process whereas, Recursive Case is the function calling statement that invokes the function repeatedly unless the base case is satisfied.

### Use of Stack in Recursion

Like single dimensional array, the stack is a memory structure that stores data values such that the last value entered into the stack can be taken out first for the processing (i.e., LIFO: Last In First Out). It performs two vital operations which are as follows:

- PUSH: To enter a value into the stack.
- POP: To retrieve a value from the stack.

The system maintains a stack internally during the execution of a recursive function. The reference of each function call is pushed into the stack unless the base case condition is true. As soon as the base case is satisfied, the references of function calls are popped out from the stack and executed simultaneously to get the desired result.

### Advantages of using Recursive Functions

The recursive functions are advantageous due to the following reasons:

- Minimum number of variables are used to develop the logic.
- As the function calls itself various times, the application of iterative loop is ignored.
- The recursive algorithm is precise, readable and efficient.
- Easier to debug and obtain error free result.

### Limitations of Recursive Function

The disadvantages of recursive functions are as mentioned below:

- Recursion takes more execution time than iteration.
- It may result in stack overflow, if recursive call continues longer.
- It occupies more space in stack.

## Types of Recursion

A recursive function can be defined in many ways but it is broadly classified into the following categories:

### 1. Direct Recursion

*Direct recursion is also known as linear recursion in which a function is called from within the same block (body of the method).* It can further be categories in two ways:

### (a) Tail Recursion

*A recursive function in which recursive call is the last statement of the function is known as tail recursion.* The factorial calculation is the best example of tail recursion. Let us understand how tail recursion is used to find factorial of a number.

The Factorial of a number 'n' can be calculated as:

n! = 1 x 2 x 3 x .....x n (where (!) stands factorial)

5! = 1 x 2 x 3 x .....x 5

It means that 5! = 5 x 4!= 5x4x3!=5x4x3x2! = 5x4x3x2x1! = 5x4x3x2x1

This proves that:

n! = n x (n-1)!

This process is recursive which yields in the development of a function having the following properties:

(*i*) if n = 0 then n! = 1

(*ii*) if n > 0 then n! = n x (n-1)!

To find factorial of 3 the following steps can be taken:

1. 3! = 3 x 2!
2. 2! = 2 x 1!
3. 1! = 1 x 0!
4. 0! = 1
5. 1! = 1 x 1 = 1
6. 2! = 2 x 1 = 2
7. 3! = 3 x 2 = 6

Back tracking is a process to use the outcome of a function in the previous function call.

Back tracking

**Example 3:**

Recursive Function to find factorial of a number (n):

```
class Recursion
{
int fact(int n)
    {
    if(n==0)
        return(1);
    else
        return(n*fact(n-1));
    }
}
```

## Recursive calls of the function

Let us use the value of n = 3.

```
fact(3)
if(3==0)
      return(1);
else
      return(3*fact(2))
       if(2==0)
         return(1)
        else
           return(2*fact(1))
                if(1==0)
                   return (1)
                else
                   return (1* fact(0))
                        if(0==0)
                           return (1)
```

6

2

1

1

## Stepwise Dry Run of Recursive Function

**Step 1:** [Evaluating fact(3)]

3 ≠ 0 then control moves to execute return(3*fact(2)). In this statement function is recalled hence, the current execution of return statement is suspended unless fact(2) has been executed.

**Step 2:** [Evaluating fact(2)]

2 ≠ 0 then control moves to execute return(2*fact(1)). In this statement function is recalled hence, the current execution of return statement is suspended unless fact(1) has been executed.

**Step 3:** [Evaluating fact(1)]

1 ≠ 0 then control moves to execute return(1*fact(0)). In this statement function is recalled hence, the current execution of return statement is suspended unless fact(0) has been executed.

**Step 4:** [Evaluating fact(0)][Now backtracking Starts]

0 = 0 then it will return 1 at the function call position fact(0) of return statement in step 3.

**Step 5:** the result of execution of return statement in step 3 (i.e. return(1*1) = 1) will be passed back to the function call position fact(2) of return statement in step 2.

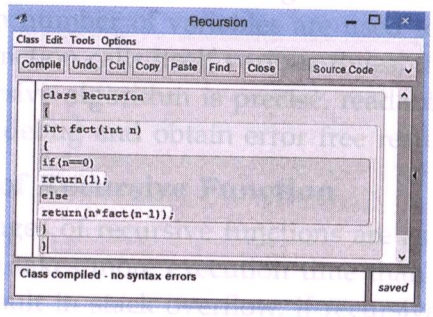
**Step 6:** the result of execution of return statement in step 2 (i.e. return(2*1) = 2) will be passed back to the function call position fact(3) of return statement in step 2.

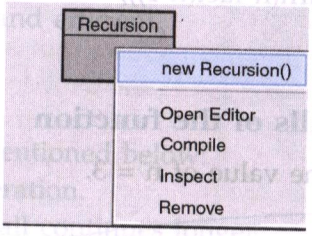
**Step 7:** the result of execution of return statement in step 1 (i.e. return(3*2) = 6) will be the final value of factorial calculation and will be returned to the main( ) function.

### Compilation and Execution of the Function:

**Step 1:** Compile the class possessing fact() function and make it error free.

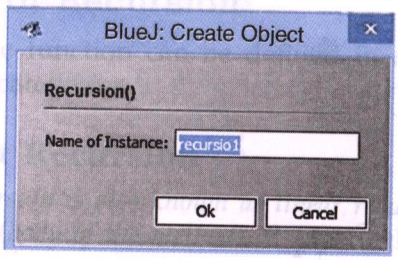
**Step 2:** Right click the folder 'Recursion' and select new Recursion( ). A Create Object window will appear.

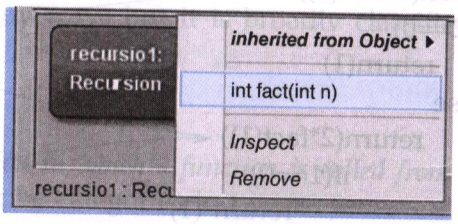


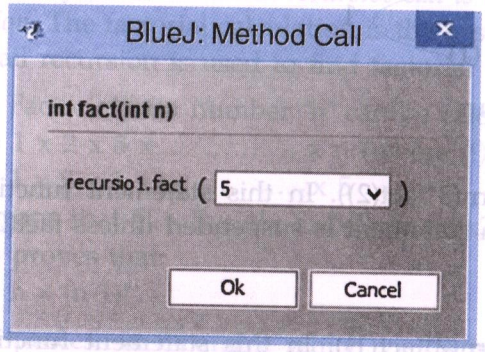

**Step 3:** Click Ok to create the object of the class Recursion.

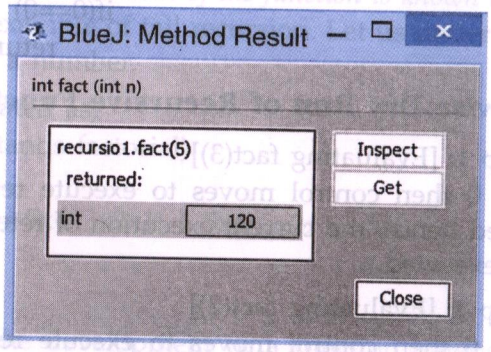
**Step 4:** Right click and select int fact(int n).




**Step 5:** Enter a value in the Method Call window whose factorial is to be calculated and click Ok.



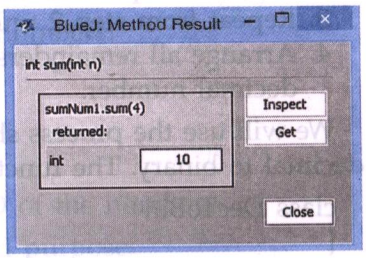




Thus, the factorial of the number will show in the Method Result window.

**Example 4:**

```
//A recursive function to find sum of first N numbers.
class SumNum
{
int sum(int n)
    {
    if(n==0)
        return(0);
    else
        return(n+sum(n-1));
    }
}
```

**Output:**



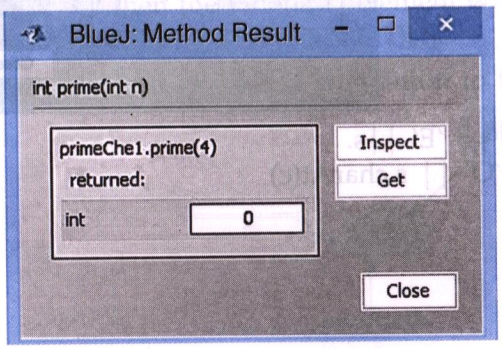

**Example 5:**

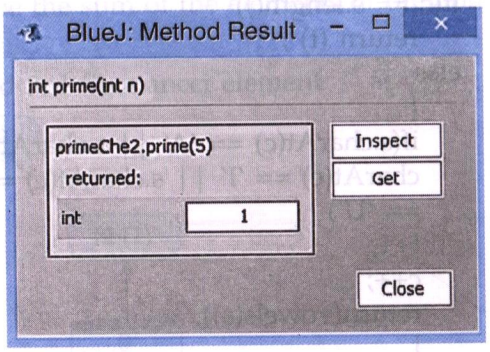
A recursive function to check whether a number N is prime or not. If it is prime then function returns 1 otherwise, 0.

A number N is said to be prime if it has only two factors, 1 and N. If N is prime then it should not get divided by any other numbers from 2 to (N-1). We are going to use the same concept in the function defined below:

```
class PrimeCheck
{
static int c = 2;
int prime(int n)
    {
    if(c == n)
        return(1);
    else
        {
    if(n % c++ == 0)
        return (0);
    else
        return(prime(n));
        }
    }
}
```

**Output:**



**Example 6:**

To convert a number N from decimal to binary.

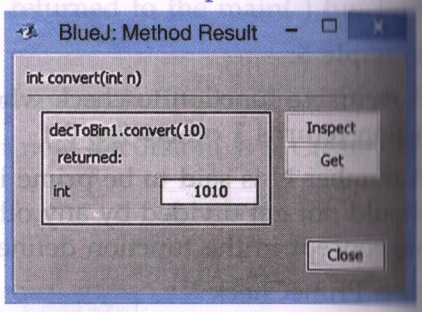
The binary equivalent of a decimal number can be obtained by using the following process:

1. Divide the decimal number by 2 and get quotient and remainder.
2. Divide the quotient of step 1 by 2 and again get quotient and remainder.
3. Repeat from step 2 unless quotient turns to 0.
4. Arrange all remainders from last to first that will be binary equivalent of the given decimal number.

We will use the process shown above, to define a function to convert number N, from decimal to binary. The function definition is:

```
class DecToBin
{
int convert(int n)
    {
    if(n == 0)
        return(0);
    else
        {
        return(convert(n/2)*10 + n % 2);
        }
    }
}
```

**Output:**




**Example 7:**

To count the number of vowels in a string.

Number of vowels in a string can be counted by using the following process:

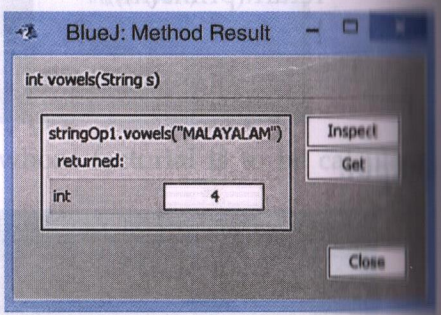
**Step 1:** Extract the next character from the string (if no character has been extracted then the first character will be treated as the next character).

**Step 2:** Compare it with each vowel. If found then increase the counter by 1.

Repeat the process from step 1 until the end of string is reached.

Finally, return the counter value

**Output:**




```
class StringOp
{
static int c = 0,t = 0;
int vowels(String s)
    {
    if(c == s.length())
        return (t) ;
    else
        {
        if(s.charAt(c) == 'A' || s.charAt(c) == 'E' ||s.
        charAt(c) == 'I' || s.charAt(c) == 'O' || s.charAt(c)
        == 'U')
        t++;
        c++;
        return(vowels(s));
        }
    }
}
```

### (b) Binary Recursion

A recursive function in which the function calls itself from two distinct points is said to be binary recursion. Both the function calls may be included in a single statement or can exist in two separate statements.

**Example 8:**

Recursive function to find GCD of two numbers N1 and N2.

The following algorithm can be used to find GCD of two numbers:

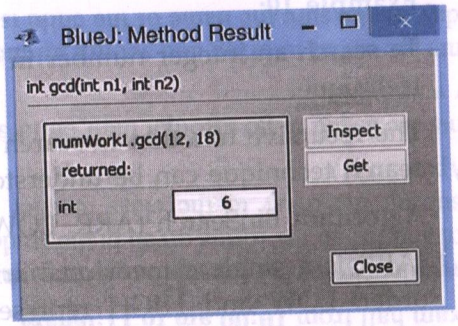
1. Check whether N1 > N2 or not.
2. If yes then find N1 = N1 – N2 otherwise,
3. Find N2 = N2 – N1
4. Repeat from step 1 unless N1 = N2.
5. As soon as N1 = N2, any one of them can be said GCD of the numbers.

Let us design a recursive function for the above mentioned purpose:

```
class NumWork
{
int gcd(int n1, int n2)
    {
    if(n1 == n2)
    return(n1);
    else
    if(n1 > n2)
        return(gcd(n1 - n2, n2));
    else
        return(gcd(n1, n2 - n1));
    }
}
```

Two points recursive calls

**Output:**




**Example 9:**

Recursive function to find Nth element of Fibonacci sequence.

The Fibonacci series can be generated as:

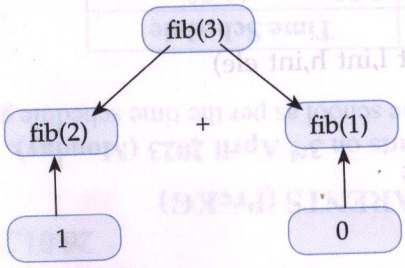
0, 1, 1, 2, 3, 5, 8, 13, .....

The series starts with first two terms 0 and 1. The next term will be the sum of first two terms (i.e., 0 + 1 = 1). Next element is the sum of the elements of previous two terms (i.e., 1 + 1 = 2) and so on.

We can find Nth Fibonacci element as per the following algorithm:

1. If N = 1 then the Fibonacci element will be 0.
2. If N = 2 then the Fibonacci element will be 1.
3. If N > 2 then the Fibonacci element will be the sum of the elements of previous two terms [i.e., (N – 2)nd + (N – 1)st].

Let us see the hierarchical evaluation to find 3rd fibonancci element

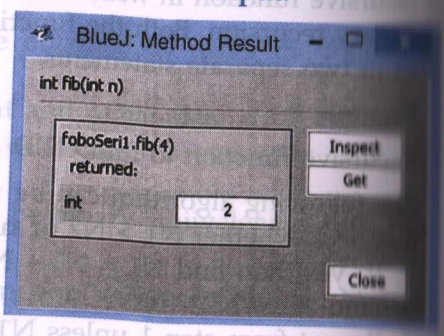



From the above hierarchical enaluation, it is evident that the 3rd fibonacci element will be 1

```
class FoboSeries
{
int fib(int n)
    {
    if(n == 1)
        return(0);
    else
        if(n == 2)
        return(1);
    else
        return(fib(n - 2)+ fib(n - 1));
    }
}
```

Two points recursive calls

**Output:**




**Example 10:**

To search an integer number in a given set of sorted numbers by using binary search technique.

The recursive function to search an element among a set of elements by using binary search technique can be understood by the algorithm shown below:

Algorithm BinSearch (ARR, LOW, HIGH, ELE)

[ARR is a single dimensional array containing a set of elements sorted in ascending order. LOW and HIGH represents the lower and higher boundaries of the array respectively. An element ELE is to be searched in the array to find whether it is available in the list of elements or not.]

| Steps | Description |
|---|---|
| 1. | [ Check for the base case possibility]<br>If (LOW > HIGH)<br>Return ("Not Found")<br>else |
| 2. | [ Find mid element and compare it with the element to be searched]<br>If (ELE < ARR [(LOW + HIGH) / 2])<br>Return(BinSearch (ARR, LOW, (LOW + HIGH) / 2 – 1, ELE))<br>else<br>If (ELE > ARR [(LOW + HIGH) / 2])<br>Return(BinSearch (ARR, (LOW + HIGH) / 2 + 1, HIGH, ELE))<br>else<br>Return(null); |

Recursive function using Binary Search Technique

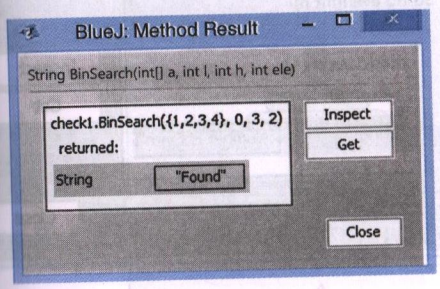
```
class Check
{
String BinSearch(int a[ ],int l,int h,int ele)
{
 if (l>h)
     return ("Not Found");
else
     if(ele == a[(l+h)/2])
         return("Found");
else
```

```
    if(ele < a[(l+h)/2])
        return(BinSearch(a,l,(l+h)/2 - 1,ele));
else
    if(ele > a[(l+h)/2])
        return(BinSearch(a,(l+h)/2 + 1,h,ele));
else
    return("");
}
}
```

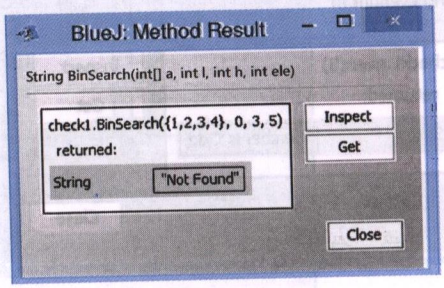
Two points recursive calls

**Output:**

**Output 1:** Result showing that the element is found.

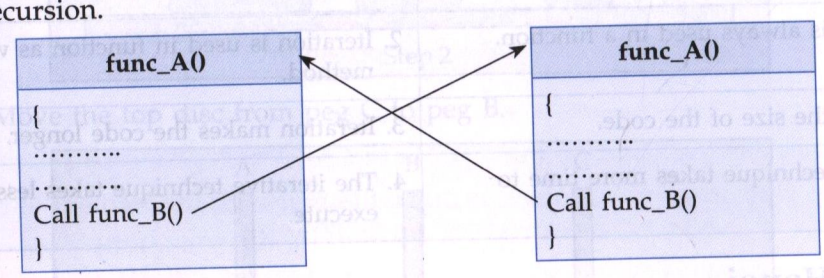



**Output 2:** Result showing that the element is not found.




## 8. Indirect Recursion

This is another type of recursion in which a function is called from another function block. It is also called Mutual Recursion. Let two functions be A and B such that function B is called from the body of function A and vice-versa then the recursive technique is said to be mutual recursion.




**Example 11:**

To check whether a number is even or not.

```
class Check
{
string even(int n)
{
    if (n == 0)
        return("Number is Even");
    else
        return(odd(n - 1));
}
string odd(int n)
{
    if (n == 0)
        return("Number is Odd");
    else
        return(even(n - 1));
```

Mutual Recursive Calls