**Day24**

**Recursion**

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.

**What is base condition in recursion?**

In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

int fact(int n)

{

if (n < = 1) // base case

return 1;

else

return n\*fact(n-1);

}

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached. How a particular problem is solved using recursion? The idea is to represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop the recursion. For example, we compute factorial n if we know factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0. Why Stack Overflow error occurs in recursion? If the base case is not reached or not defined, then the stack overflow problem may arise. Let us take an example to understand this.

int fact(int n)

{

// wrong base case (it may cause

// stack overflow).

if (n == 100)

return 1;

else

return n\*fact(n-1);

}

If fact(10) is called, it will call fact(9), fact(8), fact(7) and so on but the number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on the stack, it will cause a stack overflow error. What is the difference between direct and indirect recursion? A function fun is called direct recursive if it calls the same function fun. A function fun is called indirect recursive if it calls another function say fun\_new and fun\_new calls fun directly or indirectly. Difference between direct and indirect recursion has been illustrated in Table 1.

Direct recursion:

void directRecFun()

{

// Some code....

directRecFun();

// Some code...

}

Indirect recursion:

void indirectRecFun1()

{

// Some code...

indirectRecFun2();

// Some code...

}

void indirectRecFun2()

{

// Some code...

indirectRecFun1();

// Some code...

}

**JAVA Class 1 (Codes)**

Q1. Print numbers from 5 to 1.

public static void printNumbers(int n) {

if(n == 0) {

return;

}

System.out.println(n);

printNumbers(n-1);

}

Q2. Print numbers from 1 to 5.

public static void printNumbers(int n) {

if(n == 6) {

return;

}

System.out.println(n);

printNumbers(n+1);

}

Q3. Print the sum of first n natural numbers.

class Recursion1 {

public static void printSum(int n, int sum) {

if(n == 0) {

System.out.println(sum);

return;

}

sum += n;

printSum(n-1, sum);

}

public static void main(String args[]) {

printSum(5, 0);

}

}

Q4. Print factorial of a number n.

Apna College

class Recursion1 {

public static void printFactorial(int n, int fact) {

if(n == 0) {

System.out.println(fact);

return;

}

fact \*= n;

printFactorial(n-1, fact);

}

public static void main(String args[]) {

printFactorial(5, 1);

}

}

Q5. Print the fibonacci sequence till nth term.

class Recursion1 {

public static void printFactorial(int a, int b, int n) {

if(n == 0) {

return;

}

System.out.println(a);

printFactorial(b, a+b, n-1);

}

public static void main(String args[]) {

printFactorial(0, 1, 5);

}

}

Q6. Print x^n (with stack height = n)

class Recursion1 {

public static int printPower(int x, int n) {

if(n == 0) {

return 1;

}

if(x == 0) {

return 0;

}

Apna College

int x\_ = printPower(x, n-1);

int xn = x \* x\_;

return xn;

}

public static void main(String args[]) {

int x = 2, n = 5;

int output = printPower(x, n);

System.out.println(output);

}

}

Q7. Print x^n (with stack height = logn)

class Recursion1 {

public static int printPower(int x, int n) {

if(n == 0) {

return 1;

}

if(n % 2 == 0) {

return printPower(x, n/2) \* printPower(x, n/2);

}

else {

return x \* printPower(x, n/2) \* printPower(x, n/2);

}

}

public static void main(String args[]) {

int x = 2, n = 5;

int output = printPower(x, n);

System.out.println(output);

}

}