DSA

Module 4

Recursion

Author

Srinivas Dande

4. Recursion

- Calling the Method from itself is called as Recursion.
- When You call the Method Recursively, You must specify the Condition to Terminate the Recursive Calls. That Condition is Called as Base Consition or Base Case.

Recursion is the process of calling one method from itself until the given base case is met.

- You can solve every problem is two approaches
  - Iterative Approach
  - Recursive Approach
- Iterative Approach is best suited in some use-cases and Recursive Approach will be good in some use-cases.

Q) When We have to Recursion?
Ans:

- When You are able to divide the Problem into Sub-Problems and You are able to find the solution for Sub-Problem then Go for Recursion

<u>Structure of Recursive Methods.</u>

void show(){
//.1 Base Case
//2. SubTask Logic
//3. Recursive Call
}
	void show(){
	//1. Base Case
	//2. Recursive Call
	//3. SubTask Logic
	}

- Types of Recursion
  - Direct Recursion
  - InDirect Recursion

//Direct Recursion
// In the below example, show() is calling itself directly.
//That why Direct Recursion
 	
class Hello{
public static void main(String as[]){
show();
}
static void show(){
// Some Code
show();
}
}
	//Indirect Recursion
	// In the below example, m1() is calling m2() and m2() is calling m1().
	//That why Indirect Recursion
	
	class Hello{
	public static void main(String as[]){
	m1();
	}
	static void m1(){
	// Some Code
	m2();
	}
	static void m2(){
	// Some Code
	m1();
	}
	
	}

4.1. Understanding Method Calls

- When You have method calls then One Stack will be Created by the JVM internally.
- When You call the method then that Method call will be pushed into Stack.
- When the method execution is completed then that Method call will be poped from Stack

- See the below Example on Method Calls without Recursion.

Lab1.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab1 {

	public static void main(String[] args) {
		System.out.println("main - begin");
		m1();
		System.out.println("main - end");
	}

	static void m1() {
		System.out.println("m1 - begin");
		m2();
		System.out.println("m1 - end");
	}

	static void m2() {
		System.out.println("I am m2");
	}
}
```

4.2. Understanding Recursive Method Calls

- Every time the Recursive call is happened then that Method call will be pushed into Stack.
- When the method execution is completed then that Method call will be poped from Stack
- See the below Example on Method Calls with Recursion **without Basecase**.
- When Base case is not specified then **StackOverflowError** will be thrown at Runtime.

Lab2.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab2 {

	static void show() {
		System.out.println("Hello Guys");
		show();
	}

	public static void main(String[] args) {
		show();
	}
}
```

4.3. Understanding Recursive Method Calls with Basecase

- See the below Example on Method Calls with Recursion **with Basecase**.
- When Base case is specified then **Recusrsive Calls will be Terminated when the Baecase is met.**
- You can writethe Basecase anywhere in the Recursive Method.
- You can writethe Recursive Call anywhere in the Recursive Method.
- See Lab3,Lab4,Lab5

Lab3.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab3 {
	static void show(int n) {
		if(n==0) return;
		System.out.println("Hello Guys");
		show(n-1);
	}
	public static void main(String[] args) {
		show(5);
	}
}
```

Lab4.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab4 {
	static void show(int n) {
		if(n==0) return;
		show(n-1);
		System.out.println("Hai Guys");
	}
	public static void main(String[] args) {
		show(5);
	}
}
```

Lab5.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
public class Lab5 {
	static void show(int n) {
		System.out.println("Welcome to JLC");
		if(n==0)  return;
		show(n-1);
	}
	public static void main(String[] args) {
		show(5);
	}
}
```

4.4. Types of Recursion

- There are Two types of Recursion.
    - Tail Recursion
    - Non-Tail Recursion

Tail Recursion:

- When the Recursive Call is the Last Statement then It is Called as Tail Recursion
- Nothing to do after the Recursive Call.

Ex:

```
void show(){
//.1 Base Case
//2. SubTask Logic
//3. Recursive Call   =>  Last Statement
}
```

Lab6.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
//Print Numbers from  N to 1
public class Lab6 {
        static void printNumbers(int n) {
                if(n==0) return;
                System.out.println(n);
                printNumbers(n-1);  // Recursive Call

        }
        public static void main(String[] args) {
                int n=5;
                printNumbers(n);
        }
}
```

Non-Tail Recursion:

- When the Recursive Call is Not the Last Statement then It is Called as Non-Tail Recursion
- You have Somthing to do after the Recursive Call

Ex:

```
void show(){
//1. Base Case
//2. Recursive Call   => Not a Last Statement
//3. SubTask Logic
}
```

Lab7.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */
//Print Numbers from  1 to N
public class Lab7 {
	static void printNumbers(int n) {
		if(n==0) return;
		printNumbers(n-1);
		System.out.println(n);
	}
	public static void main(String[] args) {
		int n=5;
		printNumbers(n);
	}
}
```

Q) Which is Better – Tail or Non- Tail Recursion?
Ans:

- Choose Tail Recursion if Possible
- In the case of Tail Recursion, Code will be Optimized by the Compiler
- So Tail Recursion faster than Non-Tail Recursion

Q) Can I Use Tail Recursion Always?
Ans:

- Not Possible ,
- Because that Depends on the Problem You are solving

Ex :

- Quick Sort can use Tail Recursion
- Merge Sort needs Non-Tail Recursion only

Q) How the Compiler Optimizing the Code with Tail Recursion?
Ans:

// Your Code
 	
static void printNumbers(int n) {
 	
if(n==0) return;
System.out.println(n);
printNumbers(n-1);
}
	// Optimized Code
	
	static void printNumbers(int n) {
	
	JLC:
	if(n==0)
	return;
	System.out.println(n);
	n = n-1;
	goto JLC;
	}
	
	Now No Recursion in the Optimized Code

4.5. Problems on Bitwise

Lab8.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

//Find the Sum of Numbers from 1 to N

public class Lab8 {
        static int  sum(int n) {

                if(n==0)
                        return 0;
                else
                        return n + sum(n-1);
        }
        public static void main(String[] args) {
                int n=5;
                int sum = sum(n);
                System.out.println(sum);
        }
}

//Non-Tail Recursive
//Time Complexity : O(n)
//Aux Space Complexity :  O(1)
```

Lab9.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

/Find the Factorial of given number

public class Lab9 {

	static int  fact(int n) {

		if(n==0 || n==1)
			return 1;
		else
			return n * fact(n-1);
	}

	public static void main(String[] args) {
		int n=1;
		int f = fact(n);
		System.out.println(f);
	}

}

// Non-Tail Recursive
// Time Complexity : O(n)
//Aux Space :  O(1)
```

Lab10.java

```
package com.jlcindia;
/*
* @Author : Srinivas Dande
* @Company: Java Learning Center
* */

//Find the Sum of Individual Digits of given number
// n= 135 => 1+ 3 + 5 => 9

public class Lab10 {

        static int  digitSum(int n) {

                if(n==0)
                        return 0;
                else
                        return n%10 + digitSum(n/10);
        }
        public static void main(String[] args) {
                int n=135;
                int sum = digitSum(n);
                System.out.println(sum);
        }
}

// Non-Tail Recursive
// Time Complexity : O(n)
//Aux Space :  O(1)
```