Abstract Method in interface:

```
import java.io.*;
interface Sum{                          Declaring Interface
    public abstract int Operation (int a, int b);   → Declaring abstract method inside interface
    int Operation_three (int a, int b, int c);
}
                                        Main Class
public class GFG implements Sum{
    public int Operation (int a, int b) {
        return a*b;
    }
    public int Operation_three (int a, int b, int c){
        return a*b*c;
    }
                                        Main Function
    public static void main (String args[]){      Operates the declard
        Sum obje = new GFG();
        System.out.println (obj.Operation (10,20));
        System.out.println (obj.Operation_three(10,20,30));
    }
}
```

DAY-5

Scope

We can update inside but cannot initialize..

Declared within the same function cannot access outside

Block Scope

```
int a = 10;
int b = 20;
{
    a=78 → can be done
    int a=78; → Cannot Initialize
    int c=4;
}
System.out.print(a) => 78
```

Variables initialized outside the block can be updated inside the box.

Same rules for string

```
int c=15;
a=50;
}
```
→ int a is declared outside the block so we can update

System.out.print(c) ⇒ 15 In this values are updated
(a) ⇒ 50

We cannot initialize outside can update inside.
Initialize inside scope cannot be used outside.

Loop Scope

```
for(int i=0; i<4; i++){
    System.out.println(i);
}
```
Cannot access outside block

Shadowing: Same variable with diff value. Lower Level override upper level.

Does not work on Methods

Eg:

```
public class Shadowing{
    static int x=90;
    public static void main(String args[]){
        System.out.print(x); ⇒ 90
        int x=10;
        System.out.println(x); ⇒ 10
        fun();
    }
    Static void fun(){
        System.out.println(x); ⇒ 90
    }
}
```
Scope will begin when value is initialized

Static value is displayed here.

Variable-Length Arguments

We can change the length of the arguments.

Used to take variable no. of arguments. The Method that takes is varargs method

Syn: `static void fun (int ... a){ // body }`

array of integers

Eg:
```
public class VarArgs {
    public static void main(String [] args){
        fun(1, 2, 3);
    }
    static void fun ( int ... v){
        System.out.pointln (Arrays.toString (v));
    }
}
```

Always at end

Values Print in Array

O/p: [1, 2, 3]

Method Overloading:

Method can be assigned More times

Eg:
```
public class Overload {
    public static void main (String args[]){
        fun(7); -①
        demo(... v: 2, 4, 5); -②
        fun (name: "Priya"); -③
    }
    static void fun (int a) {
        System.out.println(a); ⇒ (7)
    }
    static void fun (String name){
        System.out.println (name); ⇒ (Priya)
    }
    static void demo(int ... v){
        System.out.println(Arrays.toString(v)); ⇒ [2, 4, 5]
    }
}
```

Display in this order

If int asked first print int value

Armstrong Number

$153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$

153 cube & Sum a=153

```
while (a>0){
    rem = a %.10
    cube =rem
    Sum += cube
     a = a/10
```

Program:

```
public static void main, (String [] args) {
    Scanner sc = new Scanner (System.~~out.pri~~in);
    int n = sc.nextInt();
    System.out.println(isArmstrong(n));
}
static boolean isArmstrong (int n){
    int value = n;
    int sum=0;
    while(n>0){
        int rem = n%.10;
        n = n/10;
        sum = sum + rem*rem*rem;
    }
    return sum = value;
}
```

O/p:
153
True