Methods

→ Sometimes our program grows in size & we want to seperate the logic from the main method to form other methods.
→ For instance, if we are calculating the average of a number pair 5 times, we can use methods to avoid repeating the logic.
DRY = Don't repeat yourself

Syntax of Method

A method is a function written inside a class. Since Java is a OOP language, we need to write method inside some class.

Syntax

```
returnType nameOfMethod() {
    // Method body
}
```

For e.g.

```
int mysum(int a, int b){

    int c = a+b;
    return c; //Return value
}
```

→ In the above method, int is the return data type of the mysum function.
→ mysum takes 2 parameters a & b
→ It returns c.

Calling a Method

A method can be called by creating an object of the class in which the method exists followed by the method call:

```
Calc obj = new Calc(); // Object Creation

obj.mysum(a, b); // Method call upon an object
```

The values from the method call (a & b) are copied to the a & b of the mysum. Thus even if we modify the values of a & b inside the method, the values in the main method won't change.

Void return type

When we don't want any our method to return anything, we use void as return type.

Static keyword

→ The static keyword is used to associate a method of a given class with the class rather than the object.
→ You can call a static method without creating an instance of the class.
→ In Java, the main() method is static, so that JVM can call the main.

Process of method Invocation in Java

```
package com.company;
class calculate {
    int sum(int a, int b){
        return a+b;
    }

public static void main(String[] args)
{
    Calculate obj = new calculate();
    int c = obj.sum(5,4);
    Sout(c);
}
```

o/p = 9

→ Inside the main method we've created an object of the calculate class.
→ obj is the name of the calculate class
→ Then we've invoked the sum method & passed 5 & 4 as arguments

Note: In case of arrays, the reference is passed. The same is the case for object passing to methods.

Methods Overloading

→ In Java it is possible for a class to contain two or more methods with the same name but with different parameters
→ Such methods are called as Overloaded methods.
→ It increases the readability of the code

E.g.

```
void foo()
void foo(int a)          // Overloaded func^n foo
int foo(int a, int b)    // foo
```

Ways to perform Method Overloading

a) By changing the return type

b) By changing the number of arguments accepted by the method

→ Now let's suppose you want to overload an "add" method. The "add" method would accept one argument for the first time & every time the number of arguments passed will be incremented by 1 till the number of arguments is equaled to 10.

→ One approach to solve this problem is to overload the "add" method 10 times, but it isn't the optimal approach

→ Hence Varargs (Variable Arguments) were introduced with the release of ~~JDK 5.0~~ JDK 5.

Syntax:

```
public static void foo(int ... arr)
{
// arr is available as int[] arr
}
```

For e.g.

```
class calculate{

  static int add(int ... arr) {
    int result=0;
    for(int a : arr){
        result = result+a;
    }
    return result;
}
```

```
p s v m (String[] args)
{
    Sout(add(1,2));
    Sout(add(2,3,4));
    Sout(add(4,5,6,7));
}

}
o/p
3
9
22
```

Recursion : A function in Java can call itself. Such calling of function by itself is called Recursion.

e.g. factorial of a no.

$$fact(n) = n * fact(n-1) \quad \forall \; n \geq 1$$

Java Methods Practice set

Q1 Write a Java method to print the following pattern normally & with recursion

```
*
**
***
****
*****
```

Q2 Method to print the mult^n table of n

```
package com.company;

~~static void multiplication (int n)~~
public class Methods {
  static void multiplication (int n)
  {
   for (int i=1; i<=10; i++) {
    System.out.format("%d x %d = %-d\n",
                      n, i, n*i);
    }
  }


static void pattern1(int n) {
  for (int i=0 ; i<n; i++) {
    for(int j=0; j<i+1; j++) {
        Sout print("*");
      } println
      Sout();
      }
}

static void pattern1_rec(int n){
  if (n>0) {
      patter1_rec(n-1);
  for(int i=0; i<n; i++) {
      ~~Sout~~(Sout print("*");
      }
    Sout println ();
  } }
```

```
p s v m (String[] args) {

    multiplication (7);
    pattern1(5);
    pattern1_rec(5);
    }
}
```

o/p:

```
7 x 1 = 7
7 x 2 = 14
  :
  :
  :
7 x 10 = 70
```

```
*
* *
* * *
* * * *
* * * * *
```

Same patten for rec.