Method in java

A method is a set of code that is meant to do some specific, well-defined task and it is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name. Think of a method as a subprogram that acts on data and often returns a value.

Each method has its own name. When that name is encountered in a program, the execution of the program branches to the body of that method. When the method is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.

Advantages of using methods

1. Generally a difficult problem is divided into sub problems and then solved. This divide and conque technique is implemented in Java through methods. A program can be divided into methods, each of which performs some specific task. So the use of Java methods modularizes and divides the work of a program.
2. When some specific code is to be used more than once and at different places in the program the use of methods avoids repetition of that code.
3. The program becomes easily understandable, modifiable and easy to debug and test. It becomes simple to write the program and understand what work is done by each part of the program

Types of Java methods

Depending on whether a method is defined by the user, or available in standard library, there are two types of methods:

1. Standard Library Methods
2. User-defined Methods

Standard Library Methods

The standard library methods are built-in methods in Java that are readily available for use. These standard libraries come along with the Java Class Library (JCL) in a Java archive (*.jar) file with JVM and JRE.

For example,

print() is a method of java.io.PrintSteam. The print("...") prints the string inside quotation marks.
sqrt() is a method of Math class. It returns square root of a number.

Here's an working example:

```
public class Numbers {
  public static void main(String[] args) {
    System.out.print("Square root of 4 is: " + Math.sqrt(4));
  }
}
```

User-defined Method:

You can also define methods inside a class as per your wish. Such methods are called user-defined methods.

Here is how you define methods in Java.

```
public static void myMethod() {
        System.out.println("My Method called");
}
```

In Java, every method must be part of some class which is different from languages like C, C++ and Python.

```
public class className {
   [modifier]   dataType methodName ([input Parameters]) [throws – exception list]
    {
      //block of code to be executed  or method body
   }
}
```

Here,

Modifiers : modifier can be access modifier(only one) or non-access modifiers(may be more than one)

(i) Access Modifier : private, public,protected, no-modifier (absence of private,public or protected)/default

(ii) Non-Access Modifier : static, synchronized, native, abstract, final,strictfp

returnType - A method can return a value.
It can return native data types (int, float, double etc.), native objects (String, Map, List etc.), or any other built-in and user defined objects.
If the method does not return a value, its return type is void.

nameOfMethod - The name of the method is an identifier.
You can give any name to a method. However, it is more conventional to name it after the tasks it performs. For example, calculateInterest, calculateArea, and so on.

Parameters (arguments) - Parameters are the values passed to a method. You can pass any number of arguments to a method.

Method body - It defines what the method actually does, how the parameters are manipulated with programming statements and what values are returned. The codes inside curly braces { } is the body of the method.

throws keyword : A method may throw exceptions

Types Of Methods:

The methods can be classified into four categories on the basis of the arguments and return value.

1. Methods with no arguments and no return value.
2. Methods with no arguments and a return value.
3. Methods with arguments and no return value.
4. Methods with arguments and a return value.

1. Methods With No Arguments And No Return Value

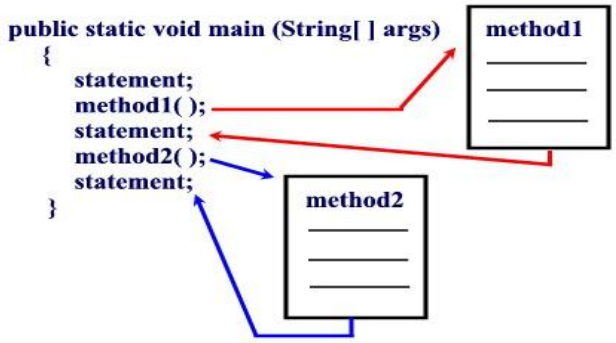
Methods that have no arguments and no return value are written as

void methodName()

```
class MethodExampleNoArgumentsAndNoReturnValue
{
                    static void   m ()
                    {
                    ……..
                    Statements;
                    …….
                    }

                     public static void main( )
                     {
                    ……..
                    m ( ) ;
                    ……..
                    }
}
```

Each method has its own name. When that name is encountered in a program, the execution of the program branches to the body of that method. When the method is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.



In the above example, the method m( ) is called by main (). **As the method m( ) has no arguments, main( ) can not send any data to m( ) and since it has no return statement, hence method can not return any value to main(). There is no communication between the calling and the called method, Since there is no return value**. These types of methods cannot be used as operands in expressions.

Example 1:

```
import java.util.Scanner;
class MethodDemo
{
 public static void main(String args[])
```

```
 {
  add();
 }

 public static void add()              //static mathod
 {
  Scanner in=new Scanner(System.in);
  int a,b;
  System.out.print("Enter The First Number:-");
  a=in.nextInt();
  System.out.print("Enter The Second Number:-");
  b=in.nextInt();
  System.out.println("Addition:-"+(a+b)); ;
 }
}
```

Example 2 :

```
import java.util.Scanner
public class Test
{
   void areacircle()          // instance method
   {
      System.out.print("enter the radius :");
      Scanner s = new Scanner(System.in);
      float r = s.nextFloat();
      float ar;
      ar = (r * r) * 22 / 7;
      System.out.println("area of the circle is : "+ar+" sq units.");
   }
}
class MethodDemo1
{
   public static void main(String args[])
   {
      Test obj = new Test();
      obj.areacircle();
   }
}
```

2. Methods With No Arguments But a Return Value

These types of functions do not receive any arguments but they can return a value. These method can be written as

int methodName()

```
class  MethodExampleNoArgumentsButAReturnValue
{
                    static  int    m ()
                    {
                    ……..
                    Statements;
                    …….
                     return 10;
                    }
                     public static void main( )
                     {
                    ……..
                    int k = m ( ) ;
                    …….
                    }
}
```

The return statement is used in a method to return a value to the calling method. It may also be used for immediate exit from the **called method** to the **calling method** without returning a value.
This statement can appear anywhere inside the body of the method.

There are two ways in which it can be used- .

return;
return (expression);

Example

```
class Box
{
   double width;
   double height;
   double depth;
   double volume()
   {
      Scanner s = new Scanner(System.in);
      System.out.print("enter width : ");
      double width = s.nextDouble();
      System.out.print("enter height : ");
      double height = s.nextDouble();
      System.out.print("enter depth: ");
      double depth = s.nextDouble();
      return width * height * depth;
   }
}
class ReturnDemo
{
   public static void main(String args[])
   {
```

```
    Box cuboid = new Box();
    double vol;
    vol = cuboid.volume();
    System.out.println("Volume is " + vol);
  }
}
```

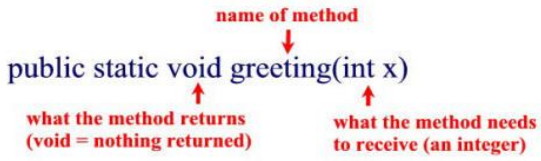
3. Methods with arguments and no return value.

These types of methods have arguments, hence the calling method can send data to the called method but the called method does not return any value. These methods can be written as:

void methodName(int x, int y)

```
class  MethodExampleWithArgumentsAndNoReturnValue
{
                    static  void    m (int x, int y)
                    {
                    ……..
                    Statements;
                    …….
                    }
                     public static void main( )
                     {
                    ……..
                     m (10,20 ) ;
                    ……..
                    }
}
```

The parameter list in the parentheses specifies the types and number of arguments that are passed to the method.



Method Arguments

The calling method sends some values to the called method for communication; these values are called arguments or parameters,

Actual arguments: The arguments which are mentioned in the method call are known as actual arguments, since these are the values which are actually sent to the called method.

Actual arguments can be written in the form of variables, constants or expressions or any method call that returns a value. For example-

```
 fun(x)
 func(a*b, c*d+k)
func( 22, 43 )
func( 1, 2, sum(a, b) )
```

Formal arguments: The name of the arguments, which are mentioned in the method definition are called formal or dummy arguments since they are used just to hold the values that are sent by the calling method.

These, formal arguments are simply like other local variables of the method which are created when the method call starts and are destroyed when the method ends. However there are two differences, First is that formal arguments are declared inside parentheses while other local variables are declared at the beginning of the method block. The second difference is that formal arguments are automatically initialized with the values of the actual arguments passed, while other local variables are assigned values through the statements written inside the method body.

The order, number and type of actual arguments in the method call should match with the order, number and type of formal arguments in the method definition.

Example:

```
import java.util.Scanner;
public class Rectangle
{
   double width;
   double height;
   double depth;
   void recarea(double x,double y)
   {
      double ar = x * y;
      System.out.print("area of the
       rectangle is :  "+ar);     }
}
```

```
class ParameterDemo
{
   public static void main(String args[])
   {
      Scanner s = new Scanner(System.in);
      System.out.print("enter length : ");
      double l = s.nextDouble();
      System.out.print("enter breadth : ");
      double b = s.nextDouble();
      Rectangle rec = new Rectangle();
      rec.recarea(l,b);      //        l and b are known as actual arguments
   }
}
```

4. Methods with arguments and a return value.

These types of functions have arguments, so the calling function can send data to the called function, ,it can also return any value to the calling function using return statement. This function can be written as

int methodName(int x, int y)

```
class  MethodExampleWithArgumentsAndAReturnValue
{
                    static  int  m (int x, int y)
                    {
                    ……..
                    Statements;
                    …….
                    return 30;
                    }
                     public static void main( )
                     {
                    ……..
                    Int k =  m (10,20 ) ;
                    ……..
                    }
}
```

Example :

```
class Add{
  public static int add_int(int x,int y){
    return x+y;
  }
```

What is Actually Passed to a Method?

In Java, parameters sent to methods are passed-by-value:

What is passed "to" a method is referred to as an "argument". The "type" of data that a method can receive is referred to as a "parameter". (You may see "arguments" referred to as "actual parameters" and "parameters" referred to as "formal parameters".)

```
//Invoke (call) the method
int number1 = 25;
int number2 = 47;
int sum = add(number1, number2);

//Method definition
public int add(int x, int y)
{
   return (x + y);
}
```

actual parameters (or arguments)

formal parameters

First, notice that if the arguments are variable names, the formal parameters need not be those same names (but must be the same data type).

Pass-by-value means that when a method is called, a **copy of the value** of each argument is passed to the method. This copy can be **changed** inside the method, but such a change will have **NO** effect on the argument.

```
 public static void main(String[] args){
   int z;
   z = add_int(2,4);
   System.out.println(z);
 }
}
```

More Examples on method concepts

1. Program to find the sum of digits of any number

```
import java.util.Scanner;
class SumOfDigitsOfAIntegerNumber
{
 static int sum (int n)
{
int i, sum=0, rem;
while(n>0)
 {
 rem=n%10;
sum+=rem;
n/=10;
}
return   sum ;
}  // sum

public static void main (String[] args)
 {
Scanner s = new Scanner(System.in);
 int num;
System.out.println ("Enter the number ") ;
num = s.nextInt();
 System.out.printf ("Sum of digits of %d is %d\n", num, sum(num) );
 }  //main

}  // class
```

2. Program to find the reverse of a number and check whether it is a palindrome or not.
(Palindrome is a number that remains same when reversed.)

```
import java.util.Scanner;
class PalindromeIntegerNumber
{
```

```
static int reverse( int  n)
 {
int rem, rev=0;.
 while(n>0)
  {
   rem=n%10;
    rev=rev*10+rem;
    n/=10;
   }
return rev;
}   // reverse

public static void main (String[] args)
 {
Scanner s = new Scanner(System.in);
int num;
System.out. println ( "Enter a number .") ;
num=  s.nextInt() ;
System.out.printf ("Reverse of  %d is  :%d\n", num, reverse (num) ) ;
 If(num==reverse(num) )
   System.out.println("Given number is Palindrome");
 else
    System.out.println( "Number is not a palindrome");
}  //main
} //class
```

3. Program to find whether the number is prime or not.

```
import java.util.Scanner;
class IsPrime
{
static boolean isPrime (int n)
{
int i;
boolean flag=true;

 for (i=2; i<=Math.sqrt (n) ;i++)
 {
   if(n%i==0)
   {
   flag=false;
   break;
  }
return flag;
```

```
} // isPrime
public static void main (String[] args)
 {
Scanner s = new Scanner(System.in);
int num;
System.out.printf ("Enter a number ") ;
num =s.nextInt() ;
 if(isPrime(num))
     System.out.println("Number is prime");
 else
     System.out.println("Number is not prime");
} //main

} // class
```

4. Program to print all prime numbers less than 500.

```
class PrimeNoUpto500
{
static boolean isPrime (int n)
{
int i;
boolean flag=true;
 for (i=2; i<=Math.sqrt (n) ;i++)
 {
   if(n%i==0)
   {
   flag=false;
   break;
  }
return flag;
} // isPrime

public static void main (String[] args)
 {
int i;
 for(i=1;i<=500;i++)
  if (isPrime (i))
    System.out.print(I + "     ");
}  //main

}  //class
```

5. Program to print twin primes less than 500. If two consecutive odd numbers are both prime (e.g. 17, 19) then they are known as twin Primes.

```
class TwinPrimeNoUpto500
{
static boolean isPrime (int n)
{
int i;
boolean flag=true;
 for (i=2; i<=Math.sqrt (n) ;i++)
 {
   if(n%i==0)
   {
   flag=false;
   break;
  }
return flag;
} // isPrime

public static void main (String[] args)
 {
int i=3,j;
 while(i<500)
 {
j =i;
i=i+2;
if(isPrime(j) && isPrime(i))
     System.out. printf ("%5d  %5d\n", j, i) ;
}  //main

} //class
```

6. Program to convert a decimal number to binary number.

```
import java.util.Scanner;
class DecimalToBinary
{
static long binary (long  num)
{
long rem, a=1, bin=0;
while(num>0)
 {
 rem=num%2;
 bin=bin+rem*a;
```

```
 num/=2;
  a*=10;
)
 return bin;
}  //binary

public static void main (String[] args)
 {
Scanner s = new Scanner(System.in);
long  num;
System.out. printf("Enter the Decimal number ");
num=s.nextInt() ;
System.out. printf ("Decimal number  = %d        Binary Number  = %d", num,binary(num));
}
}
```

Recursion in java

Recursion is a powerful technique of writing a complicated algorithm in an easy way. According this technique a problem is defined in terms of itself. To implement recursion technique in programming, a method should be capable of calling itself this facility is available in Java. The method that calls itself inside method body again and a again is know as a recursive method. In recursionn the calling method and the called method are same. For example

```
class RecursiveMethod
{

public static void main(String[] args)
{
…….
 recursiveMethod();
…….
}

static void recursiveMethod()
{
…….
recursiveMethod();        ——————→ recusive method calls
}

}
```

Here recursiveMethod ( ) is called inside the body of method recursiveMethod ( ). There should be a terminating condition to stop recursion, otherwise recursiveMethod ( ) will keep on calling itself infinitely and will never return.

```
static void recursiveMethod()
{
…….
If(…..)       // terrninating condition
   recursiveMethod();  ——————→  recusive method calls
……..
}
```

Before writing a recursive method for a problem we should consider these point :

1. We should be able to define the solution of the problem in terms of a similar type of smaller problem. At each step we get closer to the final solution of our original problem.

2. There should be a terminating condition to stop recursion.

Now we will take some problems and write recursive functions for solving them. (i) Factorial (ii) Power (iii) Fibonacci numbers

a) Program for factorial using recursive method

We know that' the factorial of a positive integer n can be found out by multiplying all integers from 1 to n.
n! = I * 2 '" 3 * * (n-l) * n
This is the iterative definition of factorial
Now we'll try to find out the recursive definition of factorial.
We know that 6! = 6*5*4*3*2*1
We can write it as 6! = 6 * 5!
Similarly we can write 5! = 5*4!

So in general we can write
n! = n * (n-l)!

We know that the factorial of 0 is 1. This can act as the terminating condition. So the recursive definition of factorial can be written as

$$n! = \begin{cases} 1 & n=0 \\ n*(n-1)! & n>0 \end{cases}$$

This method returns 1 if the argument n is 0 therwise it returns n*fact(n-l). To return n*fact(n-1), the value of fact(n-1) has to be calculated for which fact( ) has to be called again but this time with an argument of n-1. This process of calling fact( ) continues till it is called with an argument of 0, Suppose we want to find out the factorial of 5.

Initially main( ) calls fact(5)
Since 5>0, fact(5) calls fact(4)
Since 4>0, fact(4) calls fact(3)
Since 3>0, fact(3) calls fact(2)
Since 2>0, fact(2) calls fact(1)
Since 1>0, fact(1) calls fact(0)
Whe factl( ) is called with n=0 then the condition inside if statement becomes true, so now the recursion stops and control returns to fact(1) Now every called function will return the value to the previous function. These values are returned in the reverse order of function calls.

```
import java.util.Scanner;
class Factorial
{
  static long fact (int n)
  {
    if (n==0)
       return l ;
   else
      return  n*fact(n-1) ;
  }

public static void main (String[] args)
 {
Scanner s = new Scanner(System.in);
int num;
System.out. printf ("Enter a number ") ;
num=s.nextInt();
 System.out.printf ("Factorial of  %d  is % d\n", num, fact (num) ) ;
}
}
```

b) program to write a recursive function for finding out the a^n. The iterative definition for finding an is a^n = a * a * a * n times.

The recursive definition can be written as

$$a^n = \begin{cases} 1 & n=0 \\ a*a^{n-1} & n>0 \end{cases}$$

```
import  java.util.Scanner;
class Power
{
static float power (float a, int n)
{
if(n==0)
    return 1 ;
 else
   return   a*power(a,n-1) ;
}
public static void main (String[] args)
 {
Scanner s = new Scanner(System.in);
float a, p;
int n;
System.out.println ("Enter a and n ") ;
a = s.nextFloat();
n=s.nextInt();
 p=power(a,n) ;
System.out. printf("%f  raised  to power  %d  is %f \n",a,n,p);
}
}
```

Advantages And Disadvantages Of Recursion

The use of recursion makes the code more compact and elegant. It simplifies the logic and hence makes the program easier to understand. But the code written using recursion is less efficient since recursion is a slow process because of many method calls involved in it.
Most problems with recursive solutions also have an equivalent non recursive(generally iterative) solutions. A non recursive solution increases performance while a recursive solution is simpler.

Local Variables In Recursion

We know each method has some local variables that exist only inside that method. When a method is called recursively, then for each call a new set of local variables is created, their name is same but they are stored at different places and contain different values. These values are remembered by the compiler till the end of function call, so that these values are available in the unwinding phase.

c) write a recursive definition for finding fibonacci numbers.

$$fib(n) = \begin{cases} 1 & n=0 \text{ or } n=1 \\ fib(n-1) + fib(n-2) & n>1 \end{cases}$$

```
import  java.util.Scanner;
class Fibonacci
{
 static int fib(int n)        /*recursive function that returns nth term of fibonacci series*/
{
if (n==0 || n==l)
     return 1;
else
    return  fib(n-1)+fib(n-2) ;
}
```

```
public static void main (String[] args)
 {
Scanner s = new Scanner(System.in);
int nterms, i;
System.out. printf ("Enter number of terms ") ;
nterms = s.nextInt();
 for(i=0;i<nterms;i++)
     System.out.printf("%3d ",fib(i)) ;
}
}
```