Tuesday, May 12, 2020 1:13 PM

Method / function

Code Modularize :

-> breaking the entire task into tiny modules and sub-modules is known as code modularizing.

-> each module/ sub-module is executed by a specific block of code which is known as methods / functions

Advantage :

1. Code will be organized
2. identifying and fixing bugs becomes easy
3. code can be re-used (increase reusability)

methods / functions :

it is a block of instruction which can perform a specific task

syntax To create a method :

```
[modifiers]  return_type  name(  [formal_arguments , ]   )
{

        statements ;
}
```

1. method Signature :

    > method name
    > formal arguments

> method name
> formal arguments

2. method Declaration statement :

> modifiers
> return type
> method Signature

3. Method Definition :

> method declaration statement
> method body / method implementation

Where can I create a method ?
Ans :

We can create a method only inside the class block or interface block.

When does a method execute ?
Ans :

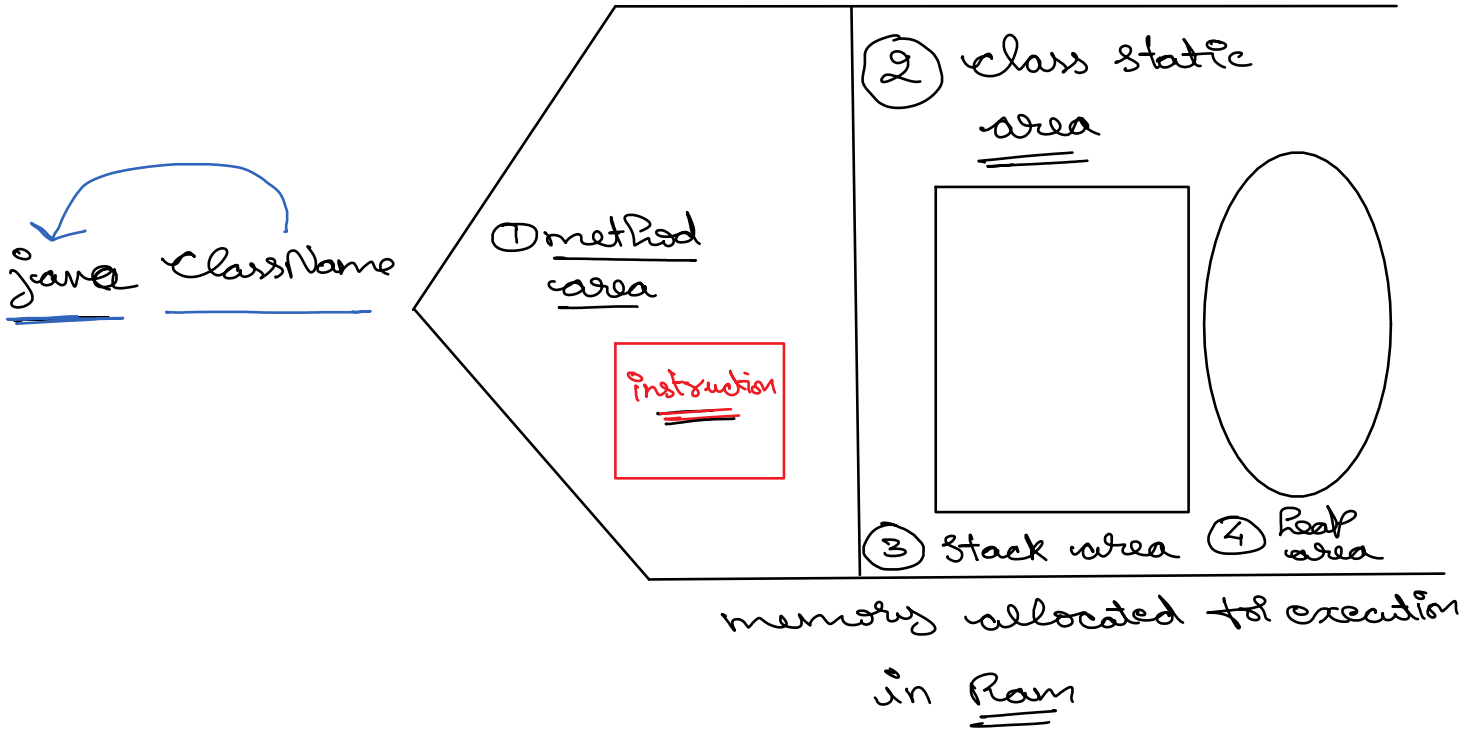
A method gets executed only when it is called

Wednesday, May 13, 2020 1:11 PM

Note :

The environment where, a java program is executed is known as JRE (Java Runtime Environment)

1. JRE contains many components like jvm, ClassLoader, JIT, ..
2. In Ram(primary memory of the system), a part of memory is allocated for the execution of the java program



1. Method area
2. stack area
3. class static area
4. heap area

→ discuss later

1. Method Area :

   The instructions will be stored in the method area.

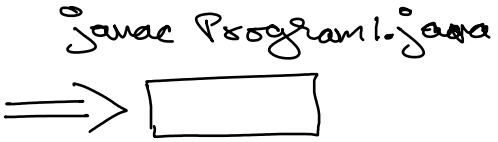
2. Stack Area :

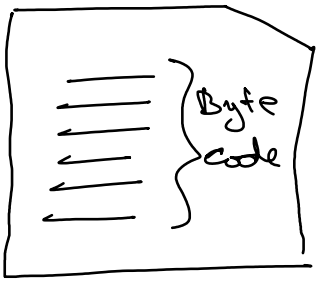
   The execution of instructions will happen in the stack area.

Example1 :

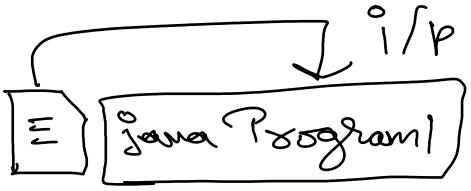
```
class Program1
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

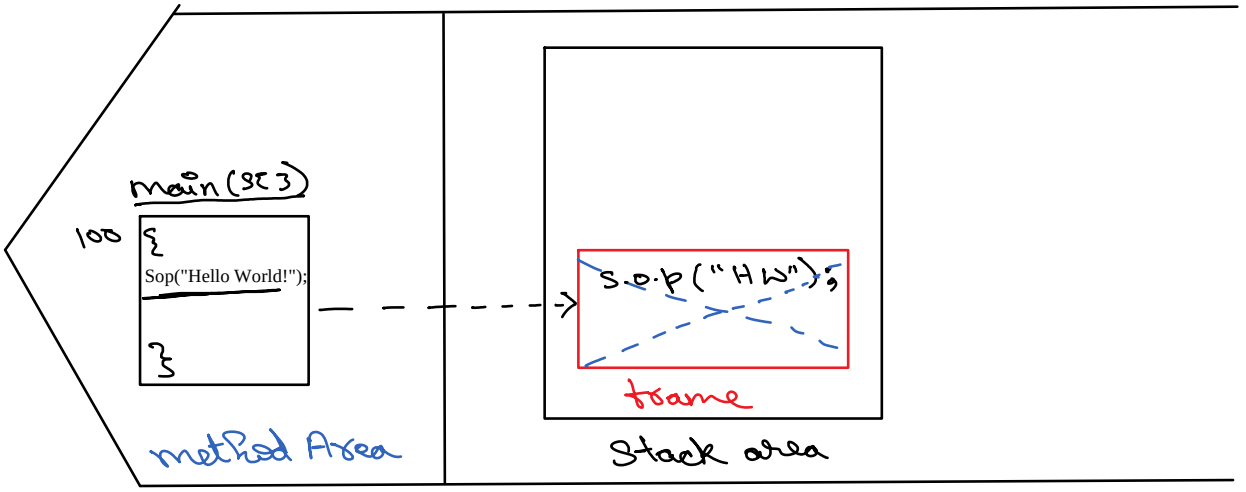
Program1.java



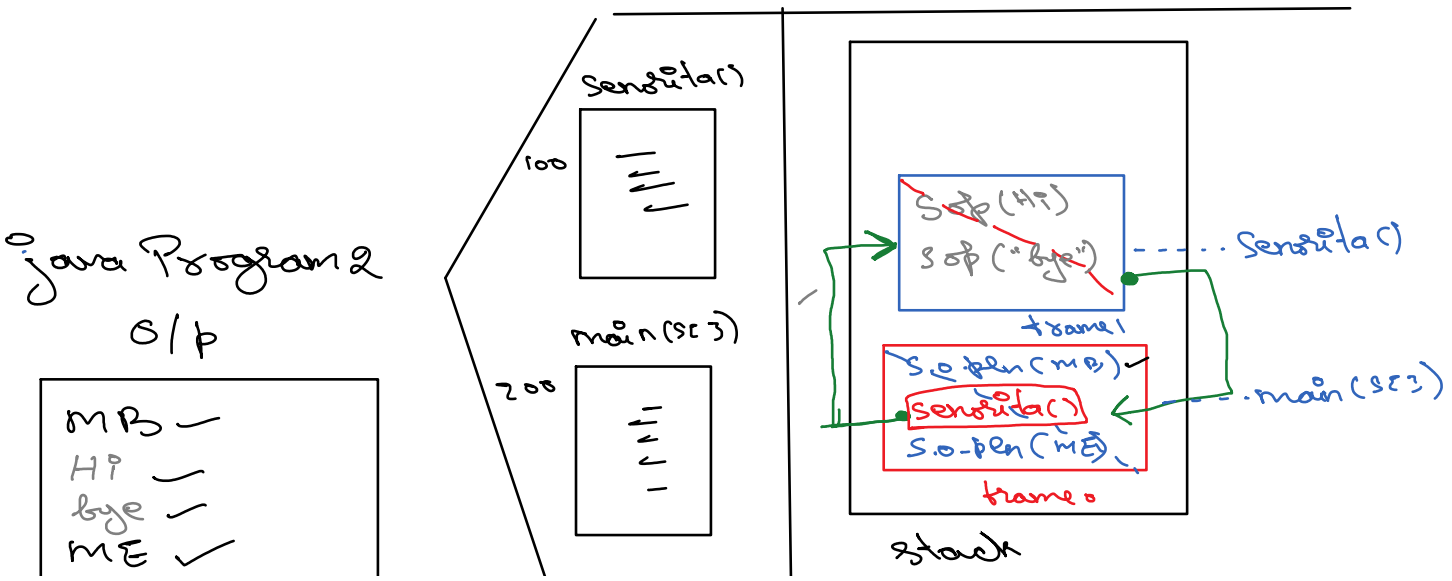


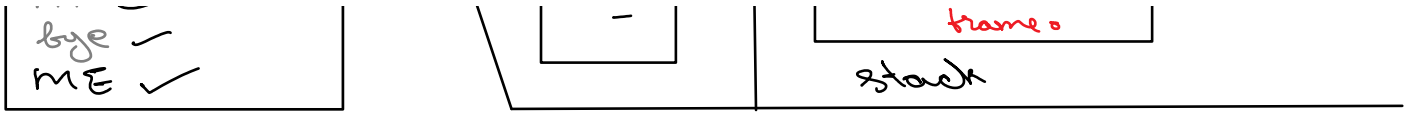
Program1.class





Example2 : **app7/ methods1/Program2.java**





What happens when a method is called ?

1. Execution of current method is paused (calling method --> main(String[]))
2. A new frame is created on top of the stack for the called method (Senorita()).
3. The control is transferred from the current frame (main(String[])) to the newly created frame(called method --> Senorita())
4. Execution of called method(Senorita()) will start, once the execution of called method(Senorita()) is completed the frame will be removed.
5. The control is transferred back to the calling method(main(String[])), and the execution of calling method will resume.

Pg3

Wednesday, May 13, 2020 1:58 PM

We can classify methods into two types :

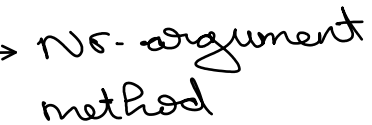
1. No-argument method
2. parameterized methods

1. No-argument Method :

The method created without formal arguments, is known as no argument method.

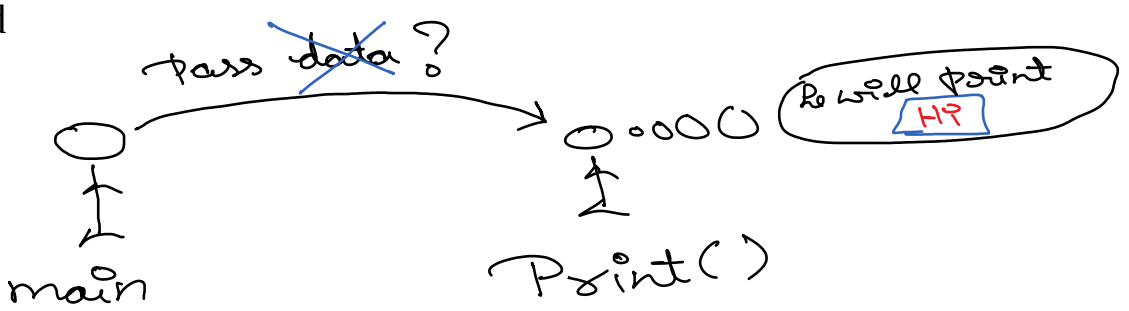
Example :

```
public static void print( )
{
        System.out.println( "hi"  ) ;
}
```



Note :

1. for no-argument method we cannot pass any input while calling the method



Syntax to call no-argument method :

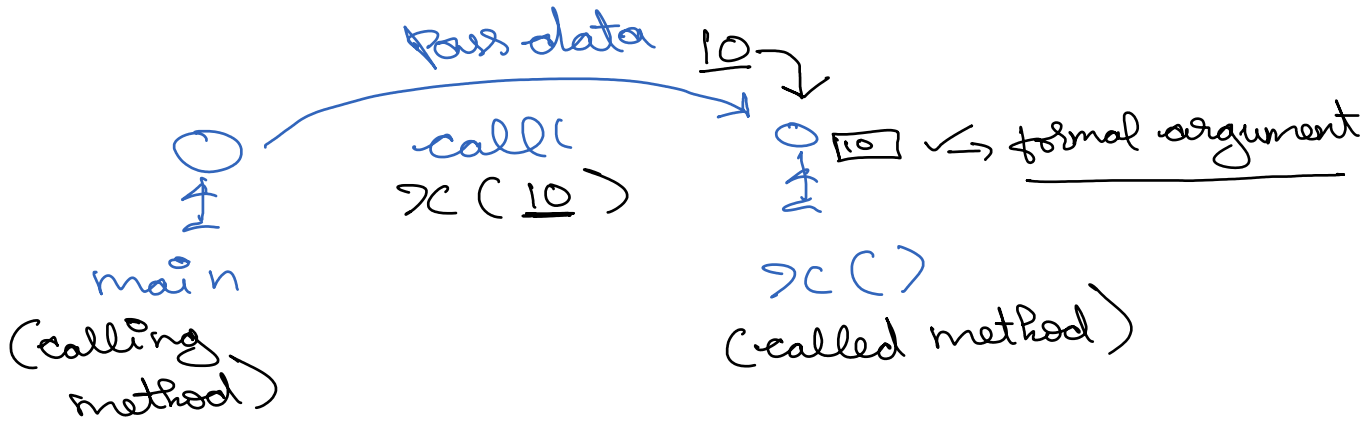
```
method_name( ) ;
```

ex: print( ) ; // ----------> method call statement

2. Parameterized method :

   The methods which have formal arguments declared in the method signature it is known as parameterized method.

   Note:
   1. the main purpose of parameterized method is pass data to the called method.



[modifiers] return_type name (datatype variable , datatype variable ,…)
{

}

Formal arguments :

the variables declared in the method declaration statement is known as formal arguments.

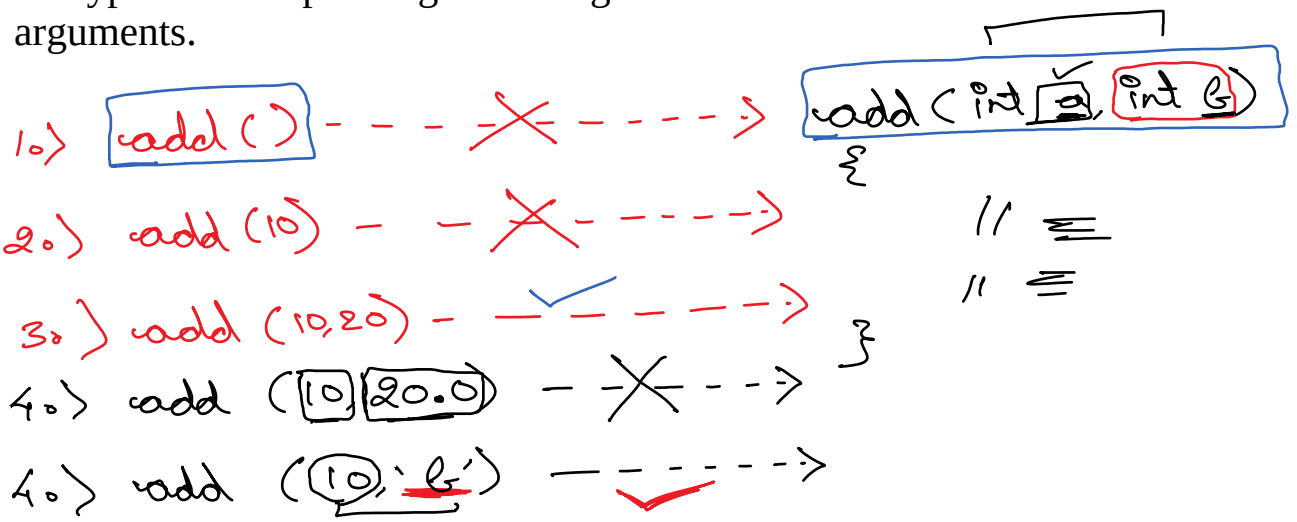
Note :
1. formal arguments are local variables to the method.

How to call a parameterized method ?

1. method call statement should follow the signature of the method.
2. in the method call statement we need to pass the **actual values**, these actual values are known as **actual arguments**.

rules :

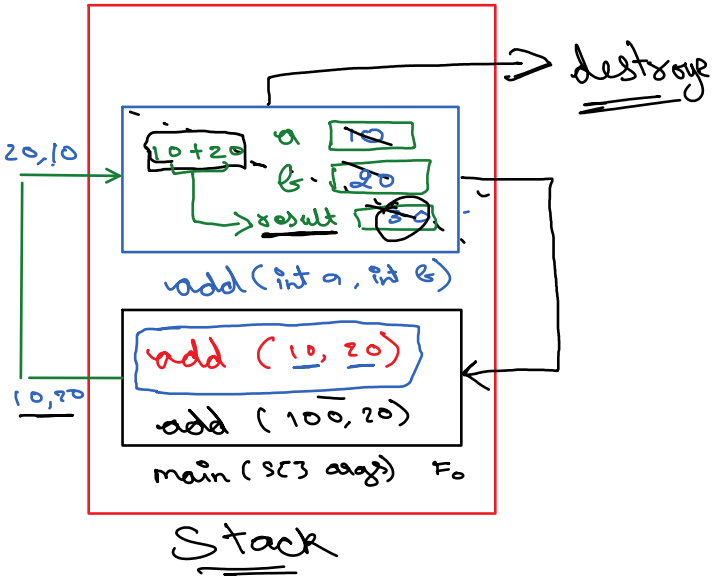
1. the number of actual arguments should be same as formal arguments.
2. the type of corresponding actual argument should be same as formal arguments.




```
class P3
{
    public static void add(int a , int b )
    {
        int result = a + b ;
        System.out.println( a + " + " + b + " = " + result ) ;
    }

    public static void main( String[ ]  args )
    {
        add( 10 , 20 ) ; // CTS
    }
}
```

O/P

10 + 20 = 30



Assignment :

1. Design a method to divide two numbers.
2. Design a method to accept m and n (m < n) and prints all the even numbers between m and n
3. Design a method to accept m and n (m < n) and prints the number odd numbers present between the given range m and n.
4. Design a method to print nth table up to m multiples.

n = 4 and m = 5
4 * 1 = 4
4 * 2 = 8

.
.
4 * 5 = 20

Can a method return data back to the Calling method ?

Yes a method can return data back to the calling method.

We can achieve this design by following the two steps :

1. We need to analyze the type of data to be returned by the method, that should be specified in the method declaration. It is known as return type of a method.

   Note :

   1. if the method does not return anything, then the return type is known as **void**.

void :	it is a keyword, used as return type. it means the method returns nothing the caller.

   2. if the method has to return data to the caller then it is mandatory to specify the type of data (datatype).
   3. The datatype can be either primitive type or non-primitive type.

2. If the return type is anything other than void, it is mandatory to use a **return** statement.

return	it is a keyword. it is a control transfer statement, which stops the execution of the current method, removes the frame and transfers the control back to the calling method. Note : 1. if the return type is void, using return statement is optional. 2. if the return type is anything other than void than using return statement with a data/expression is mandatory.

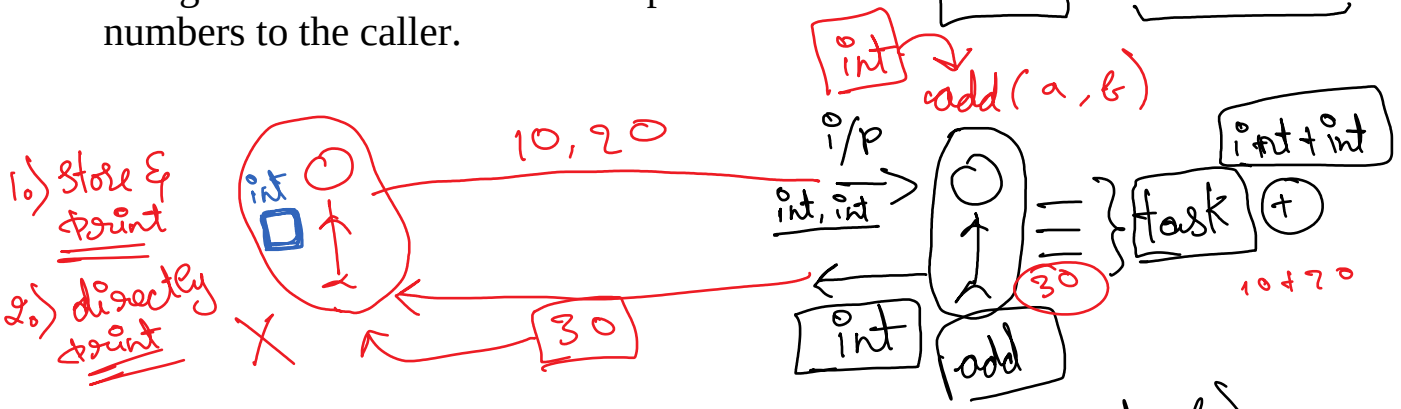
example to understand return statement :

1. Design a method which can accept 2 numbers and prints the largest of two numbers.

```
class P5
{
    public static void largestOfTwo(  int num1 , int num2   )
    {
        if (num1 > num2 )
        {
            System.out.println( num1 ) ;
            return ;
        }

        System.out.println( num2 ) ;

    }

    public static void main(String[] args)
    {
        largestOfTwo( 20 , 100 )  ;
    }
}
```

example to return a value back to the caller :

1. Design a method which can accept 2 numbers and returns the sum of those numbers to the caller.




```
public static int add(int a, int b)
{
    return a+b;
}
```

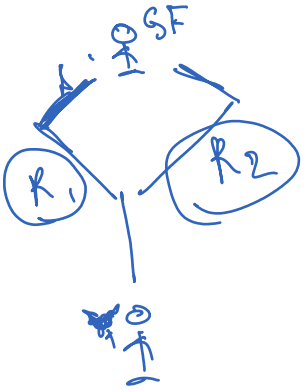
1.) Solve exp
2.) then return

Note :

1. a method can return only once.
2. a method can return only one value.
3. return statement should be the last statement of any block, if not we get CTE.
4. we can use more than one return statement, properly designed.

example : Design a method which returns the largest of two numbers.

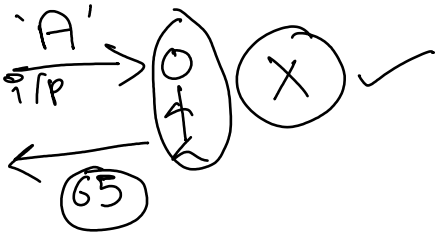
```
public static int largestOfTwo( int a , int b )
{
    if ( a > b )
        return a ;
    return b ;
}
```



==========

Assignment :

1. design a method to accept n and returns summation of numbers from 1 to n.
2. design a method which can return the number of even numbers present between m & n.
3. design a method which can accept m & n as input and returns the count of odd numbers present between them.
4. design a method which returns largest of 5 numbers
5. design a method which returns smallest of 5 numbers
6. design a method which returns an ASCCI value of a character.



Method Overloading :

A class having more than one method with the same name but different signature is known as method overloading.

Rule :

1. method name should be same.
2. formal arguments must differ either by length or type.
3. return type of the methods can be anything.

example :

refer **app7/methods3/ P2.java**

Note :

1. Compiler decides which method implementation to be executed.
2. At the Compile time
3. By considering the values (actual arguments) passed in the method call statement.

This is known as **CompileTimeBinding.**

Note : Compiler follows the following priority to perform CompileTimeBinding .

1. compiler binds with the same signature.
   (If same signature is not available)
2. it tries to perform widening in the given order (increasing order of the primitive data types)