Note : Boolean cannot be converted into any other primitive datatype. (or) anyother primitive cannot be converted into Boolean type.

* byte < short(char) < int < long < float < double *

Program :

① Class Widening

```
{
Public static void main(String[] args)
{
byte a=10;
int b;
b=a;
System.out.println(a);
System.out.println(b);
}
}
```

Convert byte to int (b=a;) byte < int

byte a [10]

int b [ ]

b=a, b=10

a=10, b=10 (converted byte to int)

//o/p 10 10 // Successfully converted byte to int (a) (b)

② small int i=20; i [20]

higher double y=i; y [20.0]

System.out.println(i);

System.out.println(y);

③

smaller byte i=20; i [20]

higher short m=i; m [20]

System.out.println(i);

System.out.println(m);

③ char ch='A';

int n=ch;

System.out.println(ch);

System.out.println(n);

2Byte char [A]

4byte int [ ] ← 65 ASCII Value

Note : Boolean cannot be converted into any other primitive datatype. (or) anyother primitive cannot be converted into Boolean type.

* byte < short(char) < int < long < float < double *

Program :

① Class Widening

```
{
Public static void main(String[] args)
{
byte a=10;
int b;
b=a;
System.out.println(a);
System.out.println(b);
}
}
```

Convert byte to int (b=a;) byte < int

byte a [10]

int b [ ]

b=a, b=10

a=10, b=10 (converted byte to int)

//o/p 10 10 // Successfully converted byte to int (a) (b)

② small int i=20; i [20]

higher double y=i; y [20.0]

System.out.println(i);

System.out.println(y);

③

smaller byte i=20; i [20]

higher short m=i; m [20]

System.out.println(i);

System.out.println(m);

③ char ch='A';

int n=ch;

System.out.println(ch);

System.out.println(n);

2Byte char [A]

4byte int [ ] ← 65 ASCII Value

ASCII Value : American Standard Code for Information Interchange

Values from A-Z is 65-90 0-9 → 48-57
a-z is 97-122

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90

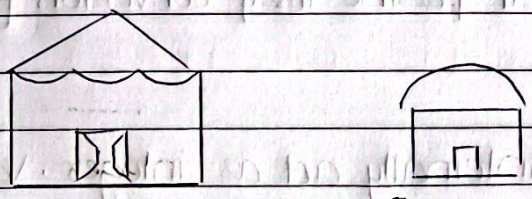
0 1 2 3 4 5 6 7 8 9
48 49 50 51 52 53 54 55 56 57

a b c d e f g h i j k l m n o p q r s t u v w x y z
97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122

Narrowing : It is a process of converting larger primitive type into smaller primitive type.

* There is a possibility of data loss.
* Implicit type casting is not allowed because of data loss and much risk.
* Narrowing must be done explicitly by using Type Cast Operator

Ex:



Shifting, language
loss will be there

double a = 10.0;
int b = a; → Not possible implicitly,
Need to do explicit type casting.

Type cast Operator : It is a Unary (one operand) Operator used to convert higher type of data to lower type of data.

Syntax : (type) value/Expression/variable;

type → datatype (primitive)

Ex : double b = 10.0
int a = b;

Typecast
double 10.0 int □ → double b = 10.0;
int a = (int)b;
↓ ↳ Variable.
Typecast

6/9/23

* byte and short → will be converted to int value.

byte b = 10; (internally act as int, 4 byte)
char ch = b;
System.out.println(b);
System.out.println(ch);

Short s = 30; (internally act as int (4 byte))
Char ch = s;
System.out.println(s);
System.out.println(ch);

→ // o/p error saying that possible lossy conversion from byte to char //

Here byte and short are internally act as integer value.

int is 4 bytes and char is 2 bytes, compiler can not do this implicitly.

To Overcome this we need to use type cast Operator.

10 + 2.5 = 12.5 (Here 10 is int type 4 byte
2.5 is double of 8 byte)
int double
so, it is converting int to double (lower to higher)

7/9/23

Methods

Method is a set of instruction written inside a block to perform a task. Method can also be called as functions.

Syntax : modifiers returntype name (formal arguments)
{

return value/Expression.
}

Here, modifiers, formal arguments and return value/Expression are optional.

1. Method Signature :

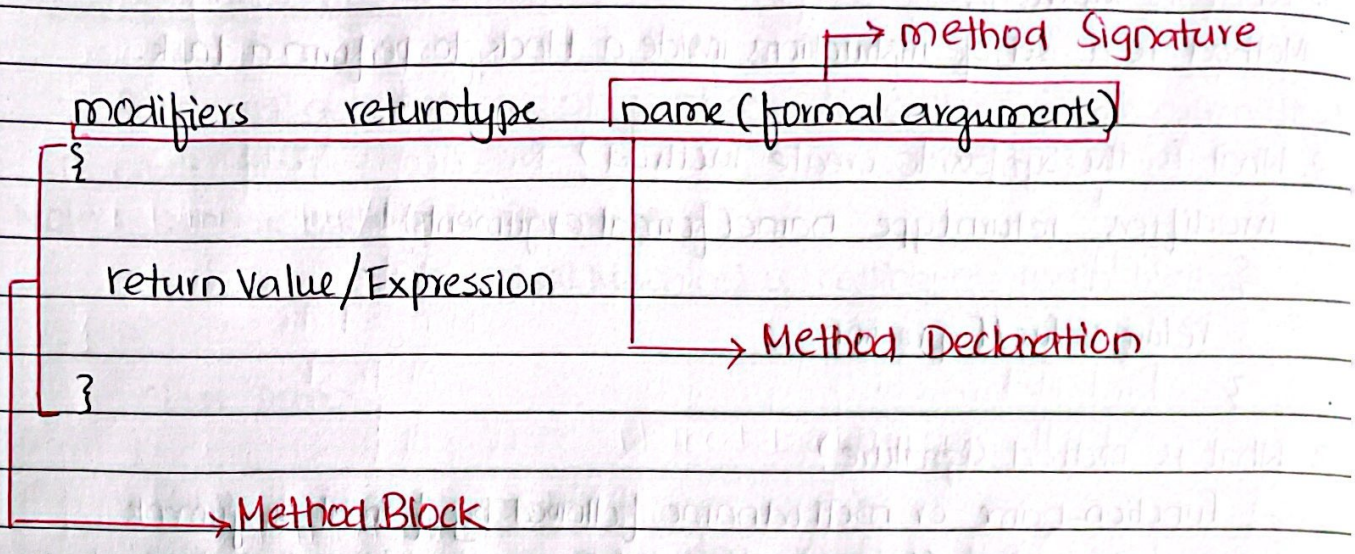
function name or method name followed by formal argument

name (formal argument)

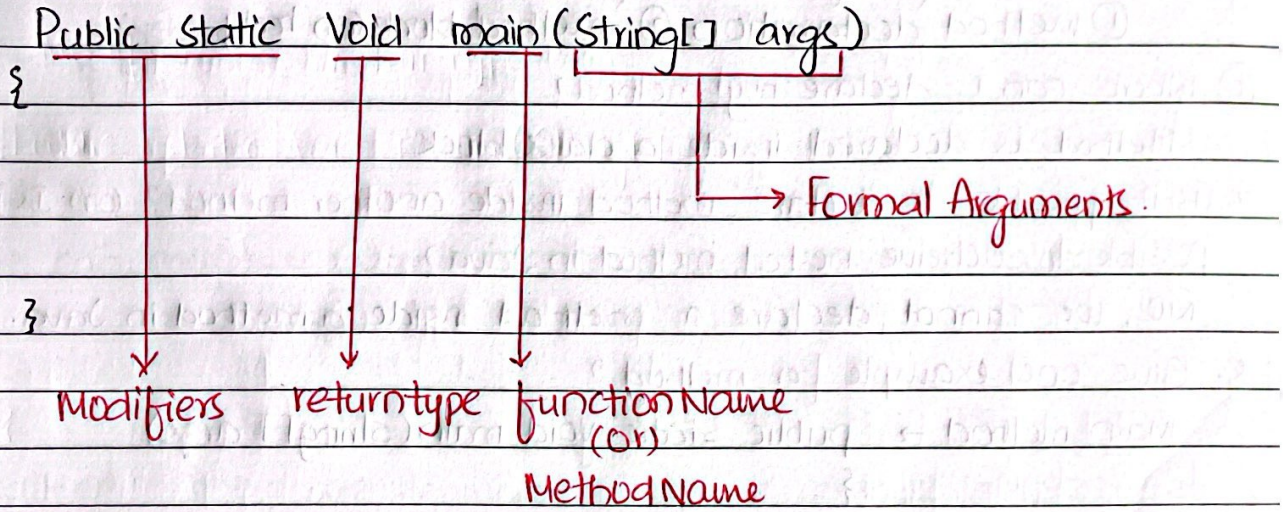
2. Method Declaration
   - (a) Modifier
   - (b) return type
   - (c) method Signature

modifiers return type name (formal arguments)

3. Method defination
   - (a) method declaration
   - (b) method block or body.



In Main method :



Where can I declare method?

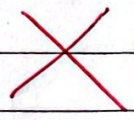
Inside the class Block

```
class Demo
{
   Method

}
```

Method inside a method is possible?

```
class Demo
{ public static void main (String[] args)
  {
   Public static void main-test( )
   {
   }
  } We cannot declare a method
    inside a method.
```



nested method is not allowed

1. What is Method?

Method is a set of instructions inside a block to perform a task.

2. What is the Syntax to create method?

```
modifiers returntype name(formal arguments)
{
    return value/Expression
}
```

3. What is method signature?

Function name or methodname followed by formal arguments

name(formal arguments)

4. What is method declaration?

① Modifier ② returntype ③ method signature

modifiers returntype name(formal arguments)

5. What is method definition?

① method declaration ② method block (or) body

⑥ Where can I declare my method?

Method is declared inside a class block

7. Is it possible to declare method inside another method? (or) Is it possible to acheive nested method in Java?

NO, we cannot declare a method inside a method in Java.

8. Give one example for method?

main method → public static void main(String[] args)

```
{
}
```

print statement → System.out.println( );

```
int num = sc.nextInt();
String str = sc.nextLine();
char ch = sc.next().charAt(0);
```

Characterstics of Methods:

- A method get execute only when it is called, we can call methods n' no.of times.

Note: Who is calling main method?

→ JVM (Java Virtual Machine) is calling the main method

```
Class Demo
{
Public static void main(String[] args)
{
System.out.println("Hi");
}
Public static void test()
{
System.out.println("Bye");
}
}
```

JVM will call

test() → We must call it explicitly

* We can call the method 'n' no.of times which is called code-reusability.

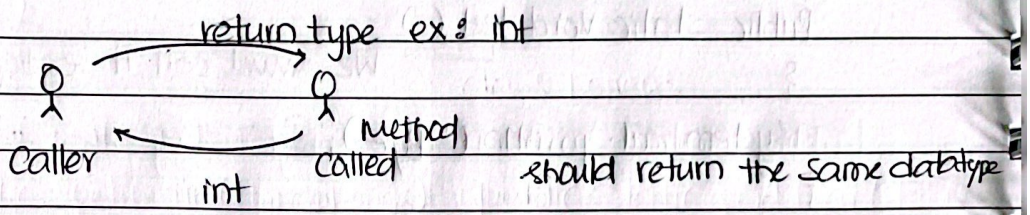
Modifier: They are keywords, they are responsible to change the behaviour of the class membex (variable, method)

```
Modifiers returntype functionName (formal arguments)
{
return value/Expression
}
```

Modifiers

- Access Modifiers
  - Public
  - Private
  - Protected
  - default
- NO-Access Modifiers
  - Static
  - NON-static
  - Abstract
  - ⋮

Return type : Return type is the datatype, it tells what type of data is given back to the caller once after the method is executed.

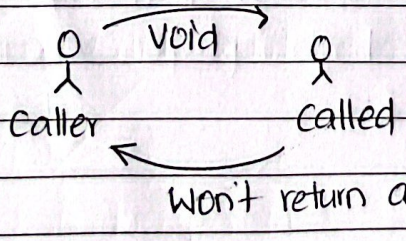


When we are creating a method we can have some possible returntypes.

① Primitive type ② Non-primitive type ③ Void.

- ① int, byte, short long, double, float char, boolean
- ② String
- ③ Won't return any value.

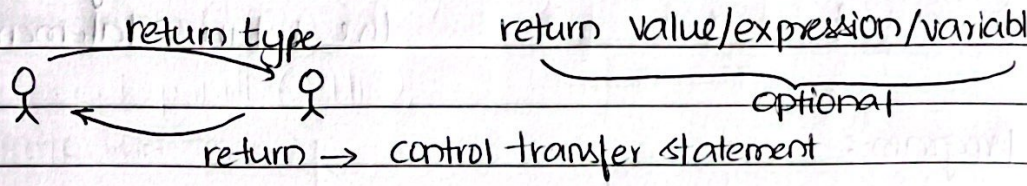
Void : Void is also a returntype which tells us we are returning nothing.



Function name → Identifiers → giving name

Formal arguments → These are the variables or data passed during declaration of method.

return : It is a keyword always in lowercase.
return control transfer statement, it stops the execution of current method and transfer control back to the caller.



Rules of returntype : ① return statement is mandatory if return-type of the method is any value other than the void.

```
modifier returntype name(formal-args)
{
    return / Expression
}
```
P-P / NP-NP / void-nothing

② If the return-type is void, return statement is optional.

void → we can have the return statement but it should not return any value.

③ Return statement can return only one value.

return variab/expression/value

return 10 ✓
return 10,20 ✗ return a,b ✗

④ After return statement we should not write any set of codes.

```
class Demo
{
psvm(I)
{
return value/expression
S.O.println("HI");
}
}
```

✗ → It will through an error if there is any statement after the return statement.

Program:

```
class Demo5
{
Public static void main(String[] args)
{
System.out.println("HI");
}
Public static void disp()
{
System.out.println("Hello");
}
}
```

O/p → HI

Here we can see only 'HI' is printed on the console, not 'Hello' because we didn't call the display method in main method

```
class Demo5
{
public static void main(String[] args)
{
System.out.println("HI");
disp();
}
Public static void disp()
{
System.out.println("HELLO");
}
}
```

O/p → HI
HELLO

* Always execution starts from main method only

① class M1
{
Public static void main (String[] args)
{
System.out.println ("HI");
disp();
test();
}
Public static void disp()
{
System.out.println ("HELLO");
}
Public static void test()
{
System.out.Println ("Bye");
}
}
}

O/p → HI
HELLO
Bye

②
Class Demo
Public Static void add()
{
int a = 10;
int b = 20;
System.out.println (a+b);
}
Public static void main(String[] args)
add();
}
} O/p 30

```
Class program
{
Public Static Void main(String[] args){
System.out.println("main strat");
demo();
demo();
System.out.println("main end");
}
Public static void demo()
{
System.out.println("from demo");
}
}
```

JVM will strats from main method, execution starts from here

* When method call statement execute

1. Execution of current method is paused
2. Control is transfered from caller to called method.
3. Execution of called method starts from the beginning
4. Once the execution is completed, control returns to the caller.
5. Execution of caller will resume.

```
Class program {
Public Static Void test()
System.out.println("testing....");}
public static void main(String[] args){
System.out.println("main start");
test();
disp();
test();
disp();
System.out.println("main end");}
Public static void disp();
System.out.println("displaying....");
```

O/p - main start
testing.....
displaying....
testing.....
displaying....
main end

Methods

Based on Arguments
① parameterized method
② NO - argument method

Based on Modifiers
① Static Method
② NON - static method
③ Final method
④ Abstract Method
⑤ Concrete Method.

NO- Argument Method : A method declared without formal arguments

Syntax : modifier returntype functionName()
{

}

Ex : public static void test()

Parameterized Method : A method declaring with parameters (or) formal arguments.

Modifiers returntype functionname(formal args)
{

}

Ex: Class Demo {
Public static void add(int a, int b)
}

Actual Argument : The data which we are passing to the method during method gets called.

```
Public static void add(int a, int b);
add (100, 500);
```
→ Actual Arguments

Rules : 1. The no. of formal argument and actual argument should be same
2. The type of corresponding actual and formal arguments should be same.

Program for NO - Argument Method:

```
class Method
{
 Public static void add()
 {System.out.Println (a+b); int a=10;
                            int b=20;
 }
 Public static void main (String[] args)
 {
  [illegible] add();          o/p 30
  [illegible]
 }
}
```

Program for Parameterized method

```
class Method {
Public static void add(int a, int b) {
System.out.Println(a+b); }
public static void main {
add (100, 500);
}
}          o/p 600
```

Actual Argument : The data which we are passing to the method during method gets called.

```
Public static void add(int a, int b);
add (100, 500);
```
→ Actual Arguments

Rules : 1. The no. of formal argument and actual argument should be same
2. The type of corresponding actual and formal arguments should be same.

Program for NO - Argument Method:

```
class Method
{
 Public static void add()
 {System.out.Println (a+b); int a=10;
                            int b=20;
 }
 Public static void main (String[] args)
 {
  [illegible] add();          o/p 30
  [illegible]
 }
}
```

Program for Parameterized method

```
class Method {
Public static void add(int a, int b) {
System.out.Println(a+b); }
public static void main {
add (100, 500);
}
}          o/p 600
```

11/9/23.

Programs based on the types

① Program without parameterised and without return type.

```
class Product
{
Public static void product()
{
 int a= 2;
 int b = 3;
 System.out.println(a*b);
}
Public static void main(String[] args){
 Product();
}
}          // o/p → 6 //
```

② Program to find sum of n numbers by considering no parameter and no returntype.

```
import. java.util. Scanner;
class Sum {
Public static void sum() {
Scanner sc = new Scanner(System.in);
System.out.println("Enter N value");
int n=s.nextInt();
int sum = 0;
for (int i=0; i<=n; i++){
Sum = sum+i;
}
System.out.println(Sum); }
Public static void main(String[] args) {
 Sum();
}
}
```