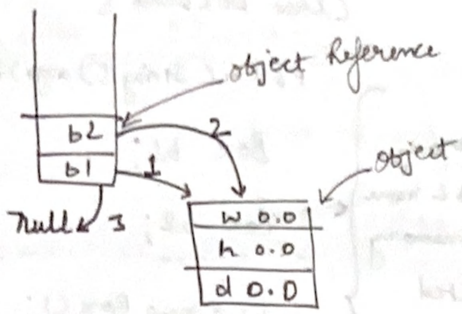



```
Class BoxDemo{
    PSVM (){
        Box b1 = new Box();
        Box b2 = b1;
        b1 = null;
        double vol;
        Vol = b2.w * b2.h * b2.d;
        SOP(Vol);
        Vol = b1.w * b1.h * b1.d;
        SOP(vol);
    }
}
```

Null pointer exception. ←//

04/03/2024

```
Box b1 = new Box();
Box b2 = b1;
```

Object reference which is pointing to an Object

```
b1.w = 2.0; b1.h = 3.0; b1.d = 4.0;
b1 = null;
SOP(b2.w);
```

Output => 2.0

```
Class Box
{
    double w, h, d;
    PSVM(String[]aw){
        Box b1 = new Box();
        :
        :
    }
}
```

```
Class Box
{
    double w, h, d;
    Void volume(){
        SOP(w * h * d);
    }
}
```

```
Class BoxDemo2{
    PSVM(_){
        Box B1 = new Box();
        Box b2 = new Box();
        b1.w=2; b1.h=3; b1.d=4;
        b1.w=20; b2.h=30, b2.d=
        b1.volume();
        b2.volume();
    }
}
```

Output: 24
6000

Method

↳ modifiers returntype method_name (arguments)

Void volume() ← default access level.

```
Class Box{
    double w, h, d;
    double volume(){
        return w * h * d;
    }
}
```

```
class Box{
    double w, h, d;
    void setDim(double w1, double h1, double d1){
        w=w1;  h=h1;  d=d1;
    }
    Void volume(){
        SOP(w*h*d);
    }
}
```

formal arguments

Output ⇒ 24
6000

```
class BoxDemo{
    PSVM(_){
        Box b1 = new Box();
        Box b2 = new Box();
        b1.setDim(2, 3, 4);
        b2.setDim(10, 20, 30);
        b1.volume();
        b2.volume();
    }
}
```

```
Void setDim(double w, double h, double d){
    w=w;
    h=h;
    d=d;
}
```

When local var. name & instance variable name is same, local variable will hide the instance variable

In this case w=w both w will be local variable hence no value will be assigned to instance variable

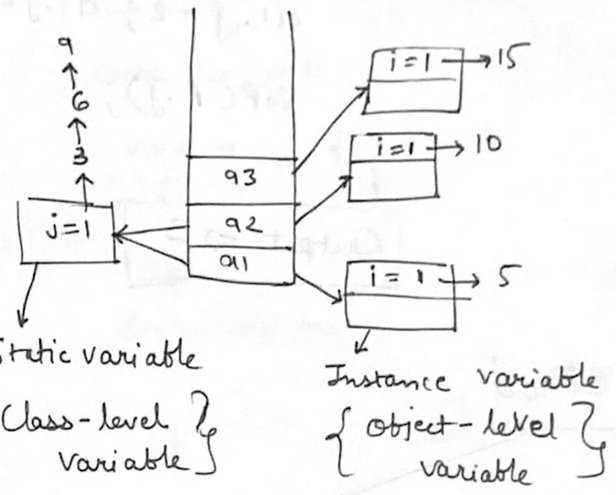
To Refer to instance member we have to use this keyword.

```
Void setDim(double w, double h, double d){
    this.w = w;
    this.h = h;
    this.d = d;
}
```

5/3/2024

Static Variable

```
Class A
{
    int i=1;
    Static int j=1;
}
```




```
Class B{
    PSVM(_){
        A a1 = new A();
        A a2 = new A();
        A a3 = new A();
        a1.i=5;  a1.j=3;
        a2.i=10;  a2.j=6;
        a3.i=15;  a3.j=9
        SOP(a1.i); SOP(a1.j);
    }
}
```

Output ⇒

5	9
10	9
15	9

Static variable → There will be only one copy of for a class and all objects will share the same copy. Default value will be same as of instance variable

→ A static variable can be referred by any of the object or with the class name.

Eg → a1.j , a2.j , a3.j Or A.j

→ A static variable gets memory at the time of loading the class, there is no need to create any object

for referring a static variable.

```
class A {
    int i;
    static int j;
}
```

```
Class B {
    PSVM(-) {
        A.j = 1;
        A a1 = new A();
        A a2 = new A();
        a1.j = 2; a2.j = 3
        SoP(A.j);
    }
}
```

Output => 3

Method

- Instance Method → {which is associated with each instance}
- Static Method → {associated with each class}

Static Method →

It have three restrictions :->

① A static method cannot use non-static data members of same class directly.

② A static method cannot use non-static methods of the same class directly.

③ this & super like keywords cannot be used in static methods.

```
Class A {
    int i = 0;
    static int j = 1;
    Void m1()
    {
    }
    Static void m2()
    {
    }
    Void m3()
    {
        SOP(i);  ✓
        SOP(j);  ✓
        m1();    ✓
        m2();    ✓
    }
    Static Void m4(){
        SOP(i);  ×
        SOP(j);  ✓
        m1();    ×
        m2();    ✓
    }
}
```

```
class A {
    int i = 1;
    static int j = 2;
    PSVM(_) {
        SOP(i);  ×
        A a1 = new A();
        SOP(a1.i);  ✓
    }
}
```

java A

internally {A.main()}

6/3/24

Constructor

→ Just like methods without return type and with the same name as class.

```
class Box {
    double w, h, d;
}
```

If we don't have a Constructor like above our Compiler will write a Constructor for us which "Do-nothing Constructor" and known as "Default Constructor".

our `Box b1 = new Box();` will call the default constructor if we don't have any in 'Box'.

Compiler will create `Box(){}` in our class.

① Constructor Constructs the object ✗

② Constructor allocate default values ✗

```
Class Box {
    double w, d, h;
    Box() {
        w = 5;
        h = 5;
        d = 5;
    }
    Void Volume(){
        SoP(w*h*d);
    }
}
```

```
Class BoxDemo {
    PSVM(_) {
        Box b1 = new Box();
        Box b2 = new Box();
        b1.Volume();
        b2.Volume();
    }
}
```

Output → 125
125

```
class Box {
    double w, h, d;
    Box() { w=h=d=5; }
    Box(double s)
    { w=h=d=s; }
    Box(double w1, double h1, double d1){
        w=w1;
        h=h1;
        d=d1;
    }
    void volume(){ SoP(w*h*d); }
}
```

```
Class BoxDemo {
    PSVM(_){
        Box b1 = new Box();
        Box b2 = new Box(10);
        Box b3 = new Box(2,3,4);
        b1.Volume();
        b2.Volume();
        b3.Volume();
    }
```

Output ⇒ 125
1000
24

If we did not have `Box(){w=h=d=5;}` then we will get error, if we make any one Constructor then default Constructor is not created.

```
class Box {
    double w, h, d;

    Box ( double s ) {
        w = h = d = s;
    }
    Box ( double w, double h, double d ){
        w = w;  h = h;  d = d;
    }
    void volume (){
        sop (w * h * d);
    }
}
```

```
Box b1 = new Box(2, 3, 4);

b1.volume();
```

Output ⇒ 0

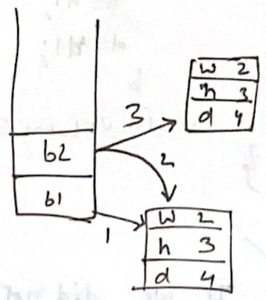
7/3/24

```
Box ( double w, double h, double d){
this.w = w;  this.h = h;  this.d = d;
}

Box( Box b){
    w = b.w;  h = b.h;  d = b.d;
}
```

```
Box b1 = new Box (2,3,4);

Box b2 = new Box (b1)
```



When constructor ended b2 stopped reffering to b1's object ie. 2nd ~~Box~~ object Connection will break.

Argument Passing Mechanism

→ Primitive type pass as Value

→ Object reference as Value

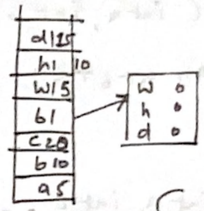
```
Class Box {

    double w, h, d;

    Void setDim (double w1, double h1
                 double d1) {
        w = w1;  h = h1;  d = d1;
    }

    Void Volume (){
        sop ( w * h * d);
    }
}
```

```
double a, b, c;
a=5; b=10; c=20;
Box b1 = new Box();
b1.setDim ( a, b, c);
b1. volume();
```



w1, h1, d1 will have copy of values of a, b, c rather than reference, there is no concept of passing reffrences in java.

```
Class Complex {

    int real, img;

    Complex ( int r , int i ){
        real = r;  img = i;
    }
    Complex add ( complex c1, complex c2){
    {
```

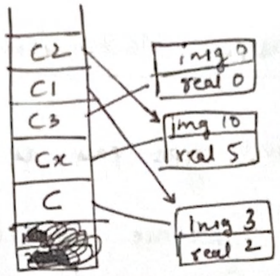
```
Complex c = new Complex (2,3)

Complex cx = new Complex(5,10);

Complex c3 = new Complex (0,0);

c3.add(c, cx);
```

{ C1, C2 will refer to the whole object rather than its reference c & cx }



{Variable no. of arguments} → Varargs

```
class Addition{
    int/real add (int a, int b){
        return a+b;
    }
int add (int a, int b, int c){
    return a+b+c;
}
int add (int a, int b ---  ){
    return a+b+c+d;
}
    .
    .
    .
}
```

```
(int ...a)
class Addition{
int add (int ...a)
{
  int i=0;
  for(int a1 : a){
      i=a+i;
  }
  return i;
}
}
PSVM (_){
  Addition a1 = new Addition();
  SOP(a1.add(1,2,3,4,5));
  SOP(a1.add(1,2,3,4,5,6,7,8));
```

{ a will become array of elements, size will be dependent upon no. of elements passed }

8/3/24

Void m1 (String arg, int ...v); ⟶ m1("Hello", 2, 3);

Void m2 (int ...v, String message); ⟶ Error

{ If we have Varargs in method then it must be the last element of method arguments }

⇒ { You can have more than 1 methods in a class which takes different type of variable length arguments }

eg → Void m1 (int ...v){_}

Void m2 (double ...d){_} ⇒ Void m1 (double ...d){_}

Void m3 (String ...s){_} → Void m1 (String ...s){_}

{ Multiple varargs are not allowed }

Void m1 (int ...i, boolean ...b){_} ✗

{ Void m1 (int i){_}
  Void m1 (int i, int ...j){_} } error

Case of ambiguity.

Runtime

```
Void m1 (int ...v){_}

Void m1 (boolean ...b){_}
```

```
m1();
  ↓
ambiguity
  ↓
runtime error
```