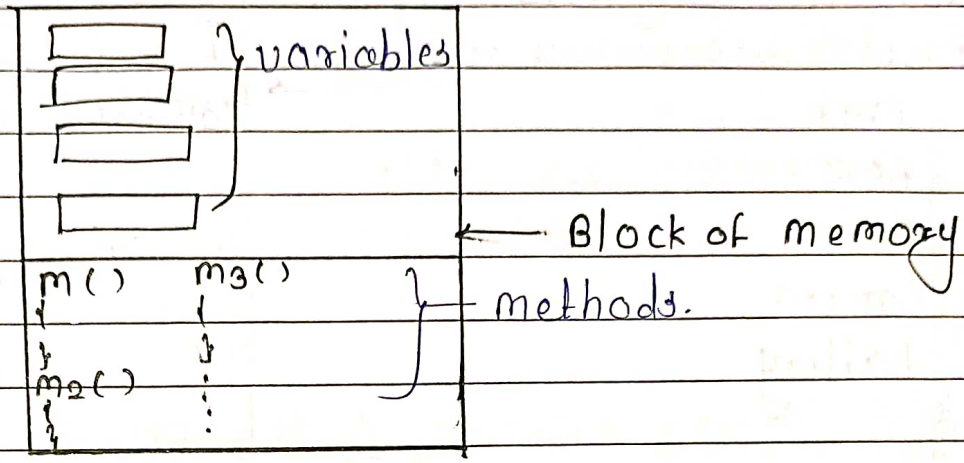
Section - 2

object - It is single block of memory which contains multiple variables and multiple methods together.



- Each an every object created in java will have a different and unique address.
- JVM will assign the address to the object.

e.g. 0221

23	24	25	26

Imp Why do we need to create an object in java ?

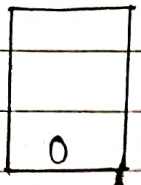
- object in java is used to represent the real world objects into the software world.

Real world object: It is an entity which has both properties and behaviours.

Eg.) Marker

Properties: brand, color, price, shape.

Task or behaviour: writing, drawing, throwing, catching.



brand
model
color
price
RAM
ROM
Camera
battery

Properties

calling
texting
clicking photo
save contact
games

task or behaviour

→

brand	price	Camera
model	RAM	battery
color	ROM	

variables.

calling() { } clicking photo() { }

texting() { } save contact() { }

methods

Ex.)

title
Author
price.

B1	title - SQL author - sohail sir price - 200.0
B2	title - Java author - James goshling prize - 250.0.
B3	title - webtech author - nihal sir price - 150.0.

Real world objects.

Obj1

title
SQL
author
~~sohail sir~~
price.
200.0

Obj2

title
Java
author
James Goshling
price
250.0

Obj3

title
webtech
author
~~nihal sir~~
Price
150.0

Software world

How to fetch the members present within the object:

- In order to fetch a member present within the object we make use of the object reference.

Syntax: objectreference.membername;

Eg:
1> System.out.println(o2.author);//James Gosling
2> System.out.println(o3.price);//150.0.

How to update the data of a variable present inside the object:

Eg. 1> o3.price = o3.price + 120;
= 150.0 + 120
= 270.0 Reinitialize

2> o1.author = "Dinga"; Reintialize.

H.W 1. 5 Employees. object.
1>display all the details of 2nd emp & 4th emp.
2>update the salary by 15% of 1st Emp & 3rd emp 23%.

Steps to create object in java:

Step 1: In order to create object in java, Blueprint of an object is necessary. Hence, Either used the existance blueprint or create the new blueprint.

Blueprint: Blueprint provides specification for an object like variables and methods to be declared or stored inside the object.

How to create a blueprint in java?

→ In java, we can create a blueprint ~~in java~~ for an object by providing specifications within the class. Hence, class Block or class Components acts as a blueprint for each an every object create in java.

* **class**

1) It is a keyword used to create a component in java.

functionalities of class: 1) Its used execute a java program.
2) Its acts as blueprint for an object.

Syntax:
```
class className.        ← non-primitive datatype
{

};
```

Eg:) class Book
{

}.

Step 2: After creating a blueprint or making use of the existing blueprint we can create the object in java by using two important factors or parameters mentioned below.

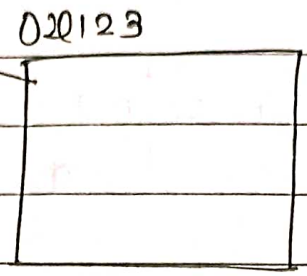
1> New common braces.
2> className ();

Syntax to create the object:

~~A~~ new className();
↙ operator/keyword. ↘ constructor.

- Whenever, An object created statement is encountered the control will be given to the "new keyword or operator first.
- The new keyword will create the object in memory space and unique address will assign by JVM.
- After creating the object new keyword transfers the control to className (); //constructor
- constructor will load or allocate the memory for the variables and the methods within the object.
- After loading process is completed the control is again transfer back to new keyword.
- new keyword will return the object address to the object creation statement.

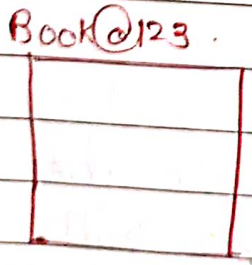
new className(); // object creation statement.



Eg:)

```
class Book
{
  public static void main (String[] args)
  {
    sopln(new Book()); // Book@123
  }
}
```

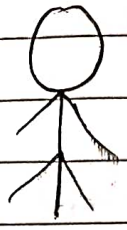
O/P:



H.W

ENAME
EID
SAL.

E1

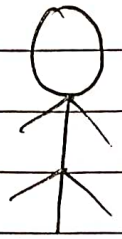


ENAME - Bheem
EID - 1
SAL - 20000.0

Obj1

ENAME
Bheem
EID
1
SAL
20000.0

E2

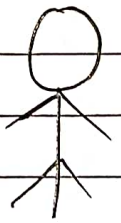


ENAME - Raju
EID - 2
SAL - 24000.0.

Obj2

ENAME
Raju
EID
2
SAL
04000.0

E3

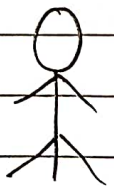


ENAME - chutki
EID - 3
SAL - 20000.0.

Obj3

ENAME
Chutki
EID
3
SAL
20000.0

E4

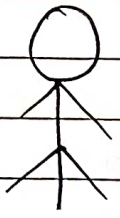


ENAME - golu
EID - 4
SAL - 23000.0.

Obj4

ENAME
golu
EID
4
SAL
23000.0

E5



ENAME - Dholu.
EID - 5
SAL - 12000.0.

Obj5

ENAME
Dholu
EID
5
SAL
12000.0.

→ 1) System.out.println(o12.ENAME); // Raju
System.out.println(o12.EID); // 2
System.out.println(o12.SAL); // 24000.0.

→ 2) System.out.println(o14.ENAME); // golu
System.out.println(o14.EID); // 4
System.out.println(o14.SAL); // 23000.0.

classmate
Date 12/6/24
Page

Eg.

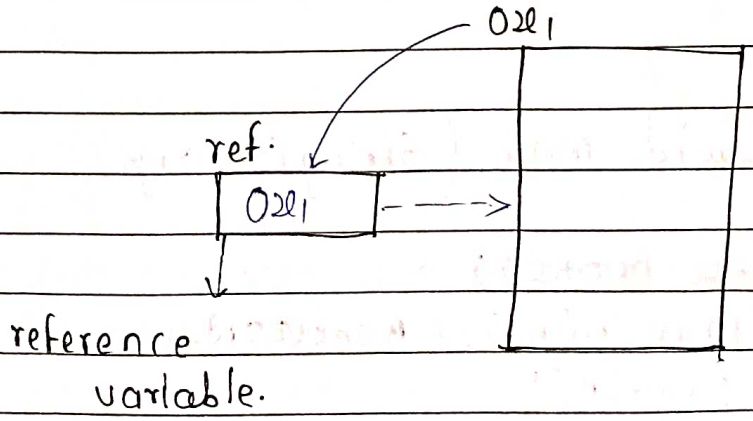
```
class Book {
    public static void main (String [] args)
    {
        System.out.println (new Book());
        System.out.println (new Book());
        System.out.println (new Book());
    }
}
```

O/P: Book@2fg2e0f4.
Book@28a418fc
Book@5305068a.

Note: For each an every object creation statement encountered in java program a new object will created with a different and unique address.

* Reference variable.

- A variable which is used to store the address or the reference of an object is termed as reference variable.



How to create a reference variable in java:

- In java we can create a reference variable with the help of a non-primitive datatype (classnam

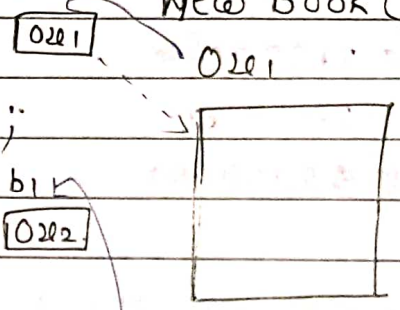
Syntax to create a reference variable:

```
ClassName ref-variablename = new ClassName();
```

Eg:-

```
class Book
{
}.
```

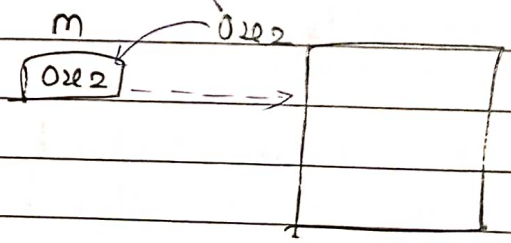
```
Book b;
Book b1;
```



X → CTE

```
class mobile
{
}.
```

```
mobile m;
new mobile();
```



Eg:-

```
class Book
{
    public static void main (String[] args)
    {
        Book b = new Book();
        System.out.println(b); // Book@2f92ce.
        System.out.println("=======");
        Book b1 = new Book();
        System.out.println(b1); // Book@530508a.
    }
}
```

what can we store inside a reference variable?

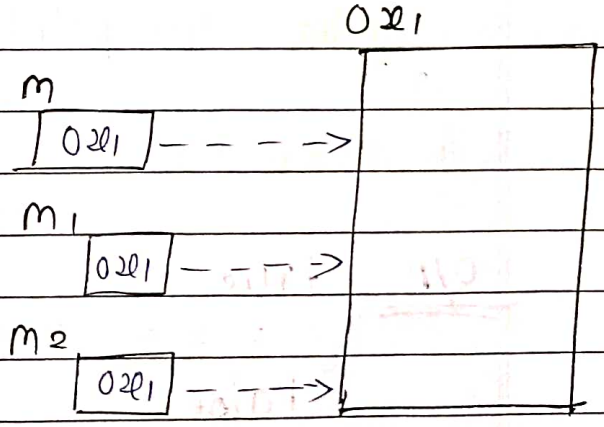
→ within a reference variable we can store only two data's the object address and null value.

Note:

1) For a single blueprint, we can create multiple objects
2) Each an every object will have different and unique address.
3) Multiple variables can point to a single object.
reference

Eg.. 3) Point

```
class Marker
{
public static void main (String[] args)
{
  Marker m = new Marker();
  Marker m1 = m;
  Marker m2 = m;
}
}.
```



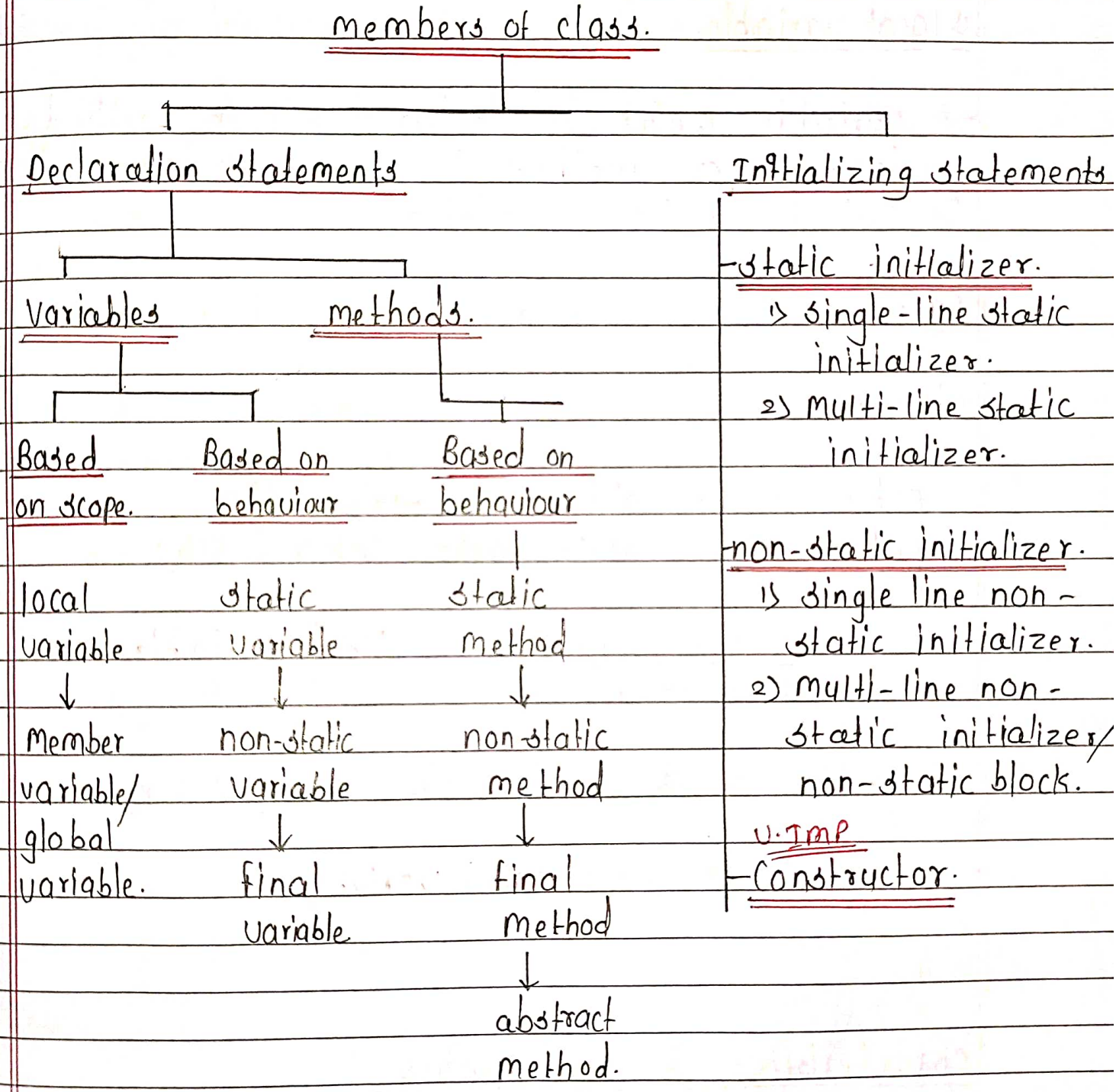
4) we can compare the two reference variable using double equal to (==) operator.

① class Marker
{
public static void main (String [] args)
{
Marker m = new Marker();
Marker m1 = m;
Marker m2 = m;
System.out.println(m1 == m2); // true.
}
}.

② class Book
{
public static void main (String[] args)
{
Book b = new Book();
Book b1 = b;
System.out.println(b == b1);
system.out.println("=======");
Book b2 = new Book();
System.out.println(b1 == b2);
}
}.

O/P: true
=======
false.

* Members of class.



* Classification of variables.

Based on scope.

1) local variable.

- A variable which is declared within method declaration or method body or implementation is called as local variable.

Eg.)

```
class A
{
  public static void add (int a, int b)
  {
    int res = a+b;          // local variable.
    Sop ln (res);
  }
  public static void main (String[] args)
  {
    int c = 10;  → local variable.
  }
}.
```

Characteristics of local variable:

1) local variables can be accessed only within the method body or block within which it is declared.

2) local variables cannot be declared as static or non-static.

3) local variables are ~~non-~~ not initialize with the default value implicitly. i.e., we programmers have to initialize the local variable explicitly.

Eg:1

```
class A
{
  public static void add(int a, int b)
  {
    int res = a+b;
    System.out.println(res);
  }
  public static void main(String[] args)
  {
Static int C; ⟶ CTE
    System.out.println(C); // CTE - local variables are not initialize with default value.

    System.out.println(res); // CTE - local variable Cannot be accessed outside the method body.
  }
}.
```

Eg.

```
{
  int a = 20;
}
```

} local variable.

2) Member variable / global variable.

- The variable which is declared within the class Blog and outside the method be defination is termed as a member variable or global variable.

Eg.)

```
class A
{
   int a = 20; <-- member variable.
   {
     int b = 30; <-- local variable.
   }
}
```

Chracteristics of global variable / member variable:

1) The member variables can be access within any of the method declared within a same class.
2) member variables can be declared as static or non-static.
3) If the programmer fails to initialize a member variable, then implicitly a default value will be initialize.

Eg:)

```
class A
{
  static int X=78;
  static int y;
  public static void add (int a, int b)
  {
    int res = a+b+x+y;
    System.out.println(res);
  }
  public static void main(String[] args)
  {
    system.out.println(x);
    System.out.println(y);
    add (a:3, b:8);
  }
}
```

O/P: 78
0
89.

15/06/24

Non-static members

1) non-static variable
2) non-static method
3) Constructor
4) non-static block

⇒ non-static variable.

- Variable which is declared within the class block and without prefixing a static keyword is called as non-static variable.

Syntax:

```
class className.
{
  Datatype v.name1, v.name_2, ....;
}
```

Eg:)
```
class Demo
{
  int a = 20; ⟶ non-static variable.
  {
    b = 30; ⟶ not a non-static variable/local variable.
  }
}.
```

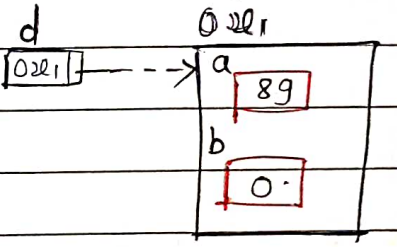
Characteristics of non-static variable.

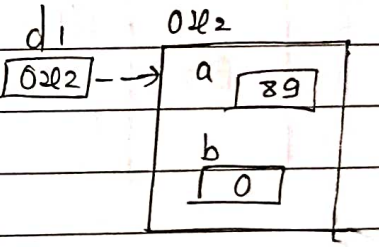
1) The memory for the non-static variable will be allocated inside the object.
2) To access a non-static variable object creation is necessary.
3) non-static variable can be accessed only with the help of object reference.
4) The non-static variables will be allocated inside each and every object created for a class.

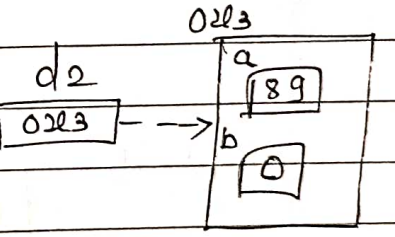
=> non-static variables will be initialized with the default values.

Eg:)

```
class Demo
{
  int a = 89;
  int b;
  public static void main (String[] args)
  {sopln(a)); ⟶ CTE.
  System.out.println("====object 1 ====");
  Demo d = new Demo();
  System.out.println(d.a);
  System.out.println(d.b);
  Demo d1 = new Demo();
  System.out.println(d1.a);
  System.out.println(d1.b);
  Demo d2 = new Demo();
  System.out.println(d2.a);
  System.out.println(d2.b);
  }
}
```







O/P:

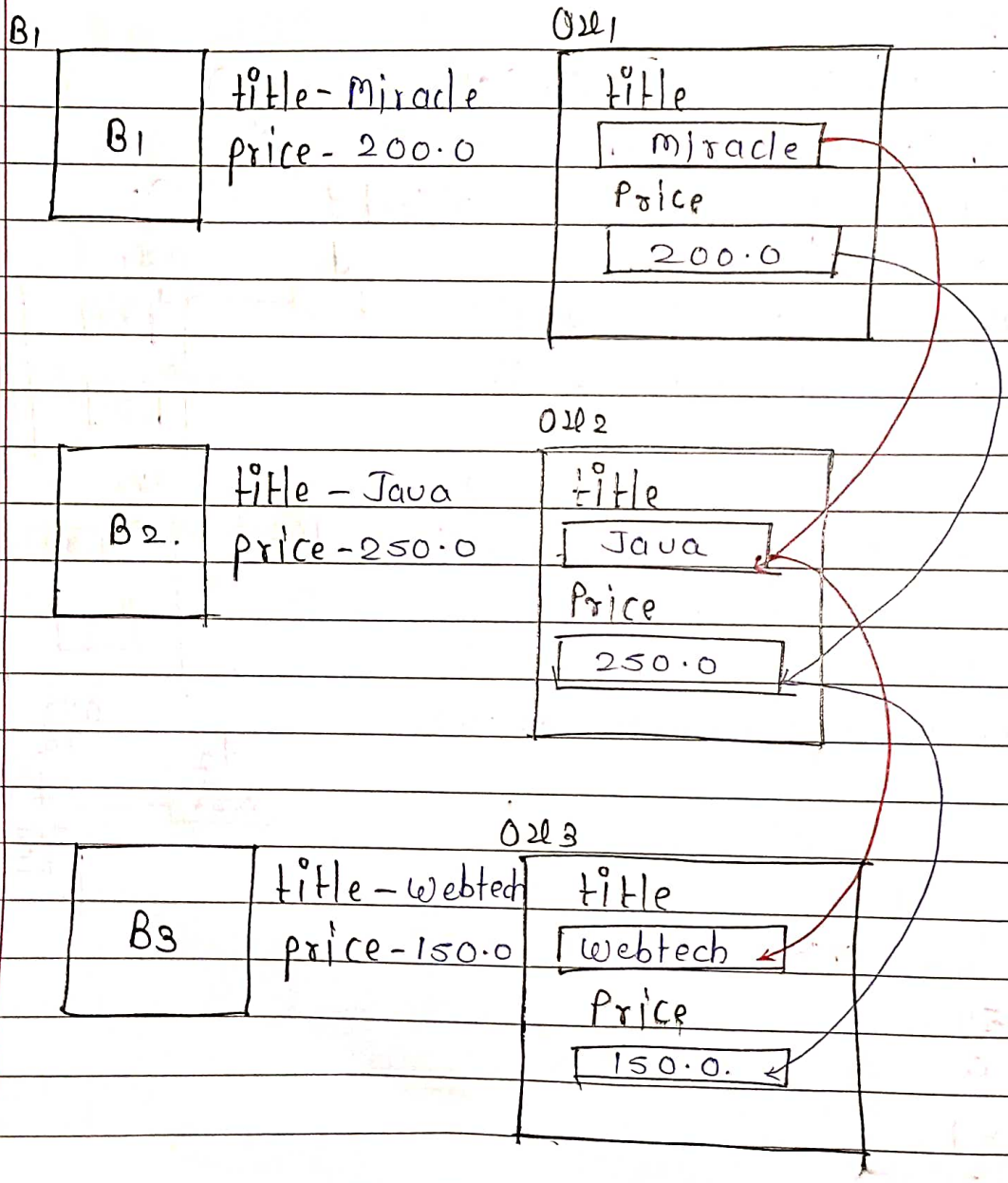
```
==== object 1 ====
89
0
89
0
89
0.
```

default value of string is null

double 0.0

U.IMP Why do we need to declare a variable as non-static / why do we need a non-static variable.

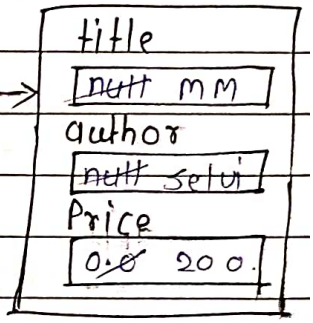
- Example.

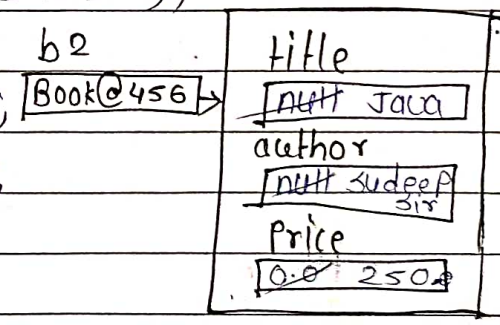


- If the data in a variable is changing from one obje to another object we have to declare those variables as non-static variables.

```
class Book
{
    String title;
    String author;
    String price;
    public static void main (String[] args)
    {
        Book B1 = new Book();
        Sopln ("=== before initialization ===");
        Sopln ("Title : " + b1.title);
        Sopln ("Author : " + b1.author);
        Sopln ("Price : " + b1.price);
        b1.title = "Miracle Morning";
        b1.author = "Selvi";
        b1.price = 200;
        Sopln ("=== After initialization ===");
        Sopln ("Title : " + b1.title);
        Sopln ("Author : " + b1.author);
        Sopln ("price : " + b1.price);

        Book b2 = new Book();
        Sopln ("=== before initialization ===");
        Sopln ("Title : " + b2.title);
        Sopln ("Author : " + b2.author);
        Sopln ("Price : " + b2.price);
        b2.title = "Java";
        b2.author = "Sudeep sir";
        b2.price = 250;
        Sopln ("=== After initialization ===");
        Sopln ("Title : " + b2.title);
        Sopln ("Author : " + b2.author);
        Sopln ("price : " + b2.price);
```

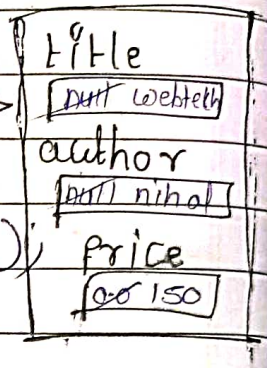


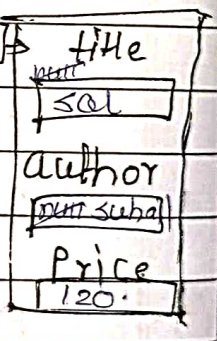


```
Book b3 = new Book();
sopln("=== before Initialization ===");
sopln("Title:" + b3.title);
sopln("Author:" + b3.author);
sopln("Price:" + b3.price);
b3.title = "webtech";
b3.author = "nihal sir";
b3.price = 150;
sopln("=== After initialization ===");
sopln("Title:" + b3.title);
sopln("Author:" + b3.author);
sopln("Price:" + b3.price);

Book b4 = new Book();
sopln("=== before Initialization ===");
sopln("Title:" + b4.title);
sopln("Author:" + b4.author);
sopln("Price:" + b4.price);
b4.title = "SQL";
b4.author = "Sohail sir";
b4.price = 120;
sopln("=== After initialization ===");
sopln("Title:" + b4.title);
sopln("Author:" + b4.author);
sopln("Price:" + b4.price);

}
}
```



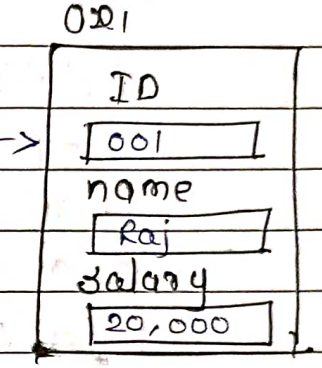


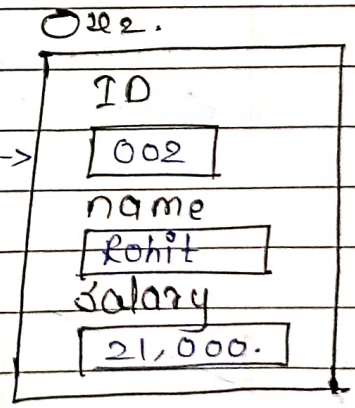
H.W Employee Details.

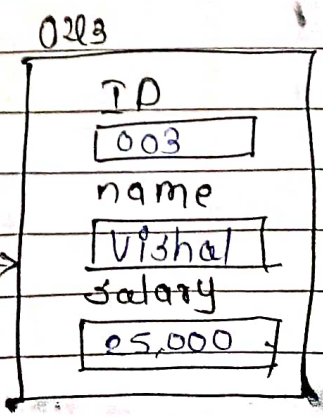
```
class Employee.
{
  int ID;
  String name;
  double salary;
  public static void main (String[] args)
  {
    Employee e1 = new Employee();
    Sopln("Details of 1st Employee");
    e1.ID = 001;
    e1.name = "Raj";
    e1.salary = 20,000;
    Sopln ("ID:" + e1.ID);
    Sopln ("Name:" + e1.name);
    Sopln ("Salary:" + e1.salary);

    Employee e2 = new Employee();
    Sopln("Details of 2nd Employee");
    e2.ID = 002;
    e2.name = "Rohit";
    e2.salary = 21,000;
    Sopln ("ID:" + e2.ID);
    Sopln ("Name:" + e2.name);
    Sopln ("Salary:" + e2.salary);

    Employee e3 = new Employee();
    Sopln ("Details of 3rd Employee");
    e3.ID = 003;
    e3.name = "Vishal";
    e3.salary = 25,000;
```







```
System.out.println("ID:" + e3.ID);
System.out.println("Name:" + e3.name);
System.out.println("Salary:" + e3.salary);

    }
}
```

O/P:

Details of 1st Employee.
ID: 001
Name: Raj
Salary: 20,000.

Details of 2nd Employee.
ID: 002
Name: Rohit
Salary: 21,000.

Details of 3rd Employee.
ID: 003
Name: Vishal
Salary: 25,000.

2) non-static method.

A method which is declared without a static keyword is called as non-static method.

Syntax:

```
Access_modifier return_type m_name(formal_args)
{
    ____
    ____

    return stmt;
}
```

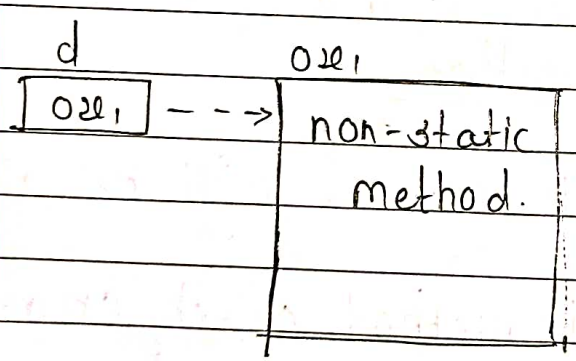
Eg:)
```
public void demo()
{
    sopln("Non static method");
}
```

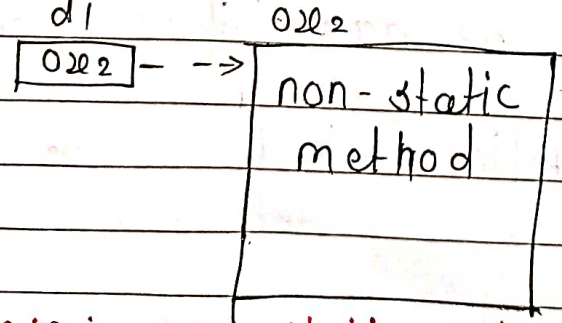
Chracteristics of non-static method.

1) The memory for non-static method will be allocated inside the object.
2) To access the non-static method object creation is necessary.
3) Non-static method can be access only with the help of object reference.
4) The memory for non-static method will be allocated inside each and every object created for a class.

Eg:

```
class Demo2
{
  Public void demo()
  {
    sopln("non static method");
  }
  public static void main(String[] args)
  {
    demo(); //CTE - memory not allocated for non-static
    Demo2 d = new Demo2(); method without creating objec
    d.demo();
    Demo2 d1 = new Demo2();
    d1.demo();
  }
}
```





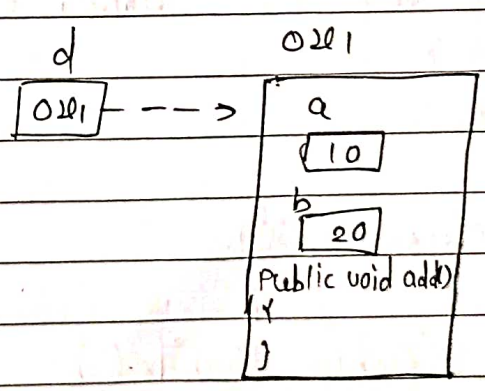
O/P: non static method
non-static method.

Why do we need to declare a method as non-static

- The main purpose of non-static method is to access the property of the object. (It could be for reading or writing or validating or manipulating - (changing)).

Eg:)

```
class Demo
{
 int a = 10;
 int b = 20;
 public void add()
 {
   sopln (a+b);
 }
 public static void main (String[] args)
 {
   Demo d = new Demo();
   sopln (d.a);
   sopln (d.b);
   d.add();
 }
}
```



O/P :

10
20
30

↳ It is not method.

3) Constructor - it is initializer.

- Constructor is a special non-static member which is used to load and initialize the non-static members to the object.

Syntax:

```
Access_modifier  className (formal arguments)
{
   ____
   ____
}
```

Eg.)

①
```
class Marker {}
{
  public Marker()
  {
    sopln ("marker constructor");
  }
}.
```

②
```
class Mobile
{
  public Mobile()
  {
    sopln ("Mobile constructor");
  }
}
```

Chracteristics of a Constructor

1) Constructor is a non-static initializer.
2) The name of the constructor should be always the className.
3) There is no return-Type for a constructor.
4) Constructors can be invoked or executed only at the time of ~~executi~~ object creation.

Empty return stmt we can write in constructor.

Note: The memory for the constructor will be allocated inside the object.

Why constructor is non-static initializer.

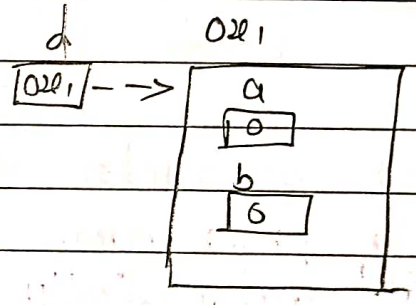
→ When programmer fails to initialize variable then it will automatically initialize variable.

* If we are not taking className same as constructor Name then it will assume normal method (return type)

Note: At the time of object creation, After creating the memory block for the object, the new operator transfers the control to the constructor.
Now the constructor is invoked and it performs 2 main functionalities: 1) It will load all the non-static properties and non-static method to the object and it will initialize non-static variables with default values.
2) After loading process is completed the statement return within the constructor body gets executed.

Eg:-

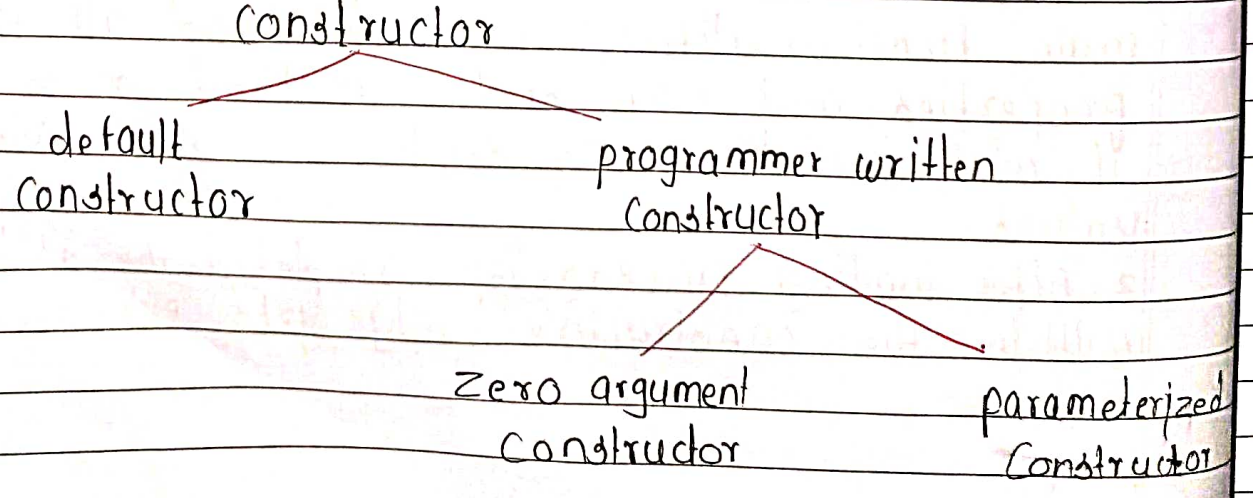
```
class Demo2
{
 int a;
 int b;
 public Demo2()
 {
  sopln("Demo2 constructor");
 }
 public static void main(String[] args)
 {
  sopln("main starts");
  Demo2 = new Demo2;
  sopln("main ends");
```



O/P:

main starts
Demo2 Constructor
main ends.

Types of constructor



When the programmer fails to add the constructor then compiler add impiicitly. default constructor

Classmate Date Page

⇒ default constructor

Note: Each an every class must and should have constructo to create an instance or object.

Imp If the programmer fails to write a constructor explicitly then the compiler will add the constructor implicitly at the time of compilation process. and that Constructor which is added by the compiler at Compilation time is known as default constructor

Eg:)

Before Compilation.

```
⇒ class Book
{
public static void main (String[] args)
{
   Book b = new Book();
}
}.
```

After compilation.

```
class Book
{
  Book()
  {
   // default constructor
  }
  public static void main (String[] args)
  {
    Book b = new Book();
  }
}
```

2) Before compilation.

```
class Marker
{
  public Marker (int a)
  {

  }
  public static void main (String[] args)
  {
    Book b = new Book ();
  }
}.
```

↓

After compilation.

```
class Marker
{
  public Marker (int a)
  {

  }
  public static void main (String [] args)
  {
    Book b = new Book ();
  }
}.
```

2) Programmer written constructor.

i] Zero-argument constructor

- A constructor which is declared without any formal arguments is termed as zero argument constructor.

Syntax:

```
Access_modifier  className()
{
    ____________
    ____________
}.
```

Eg.)

```
class Book()
{
  public Book()
  {
    sopln("zero argument method");
  }
  public static void main(String[] args)
  {
    new Book();
    new Book();
  }
}.
```

O/P: zero argument constructor
zero argument constructor.

How to differentiate a local variable a member non static variable having same name.

→

within a static context

- within static context we can differentiate a local variable and a member variable ~~to~~ non-static variable with the help of object reference.

Eg:-

```
class Demo
{
  int a = 25;
  public static void main (String[] args)
  {
    int a = 70;
    sopln(a); //70
    Demo d = new Demo();
    sopln(d.a); //25.
  }
}
```

static context