(23) Static Keyword in Java

↳ 10th Nov live class
↳ Hyder abbas Sir.

① Static Keyword :-

the static keyword in Java is used for memory Management mainly. We can apply static keyword with Variables, Methods, blocks and nested classes.

The static can be
① Variable (also known as a class Variable)
② Method (also known as a class Method)
③ Block
④ Nested class

→ Class { } In a class we have
Static Variables
Static Block
Static Method
non-static Variables (instances)
Java block
non-static Method
}

→ Static Variables are accessed inside static block, static Method, Java block and non-static Method
accessed - used

→ Non-static Variables are accessed (used) inside Java block and non-static Method.

→ we cannot use instance Variable inside static block and static Method.

→ for non-static Variables (instance Variables) Memory is allocated inside object.

→ object is created first and Memory will be allocated for instance Variables.

→ non-static Variables (or instance Variables) are object dependent.

→ there is like non for instance Variable if object is not present.

(1.1) Why Instance Variable are not accessed in static block and method?

→ All static Elements in class are not object dependent. they are object independent. but Instance Variables are object dependent.

→ static Elements will get Executed without Creating object.

```
static {
    int a;
}
```

no life for it because no object created.

→ JVM checks for Main Method in a Program. and Java class will be loaded if it is present

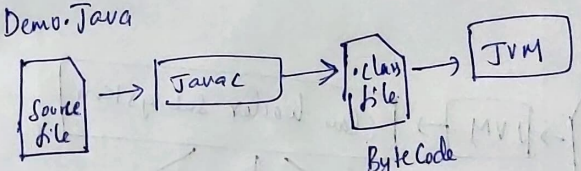
→ JVM checks for main if it is present it proces the Execution.

→ Main Method will not Execute first.

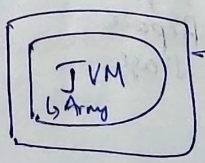
(1.2) order of Execution

1. Static Variables
2. Static Block
3. Static Method
4. non-static (or) Instance Variables
5. non-static block
6. Constructor
7. Method

→ if static Variable and block not present in the program then Main Method Executes first.



→ to Execute Java Program an Environment will be provided.



→ JRE (Java Runtime Environment)
↳ Battle field

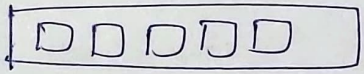
1.3 What JVM will do?

JVM:-

① JVM we class loader subsystem

↳ class loader loads the class and do certain activities

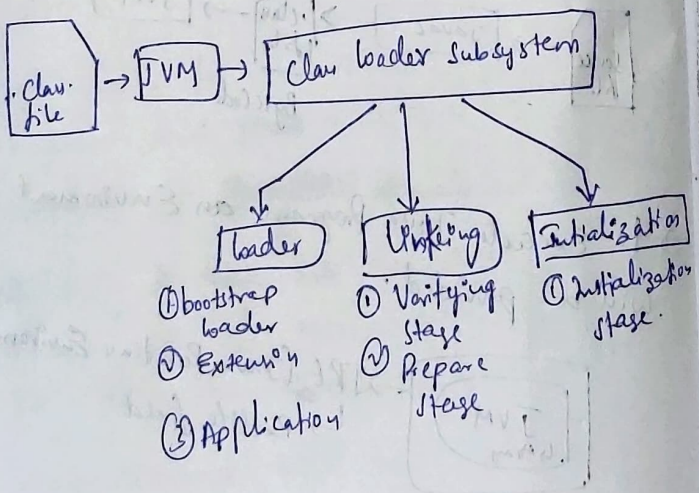
②



→ JVM have data.

③ Later Execution will happen.

↳ JVM will use of interpreter, Just in time compiler and many native references.



→ class loader subsystem has 3 important parts

ⓐ loading - bootstrap, Extension, Application

ⓑ linking

ⓒ Initialization

ⓐ Loading:

(i) Bootstrap loader

(ii) Extension loader

(iii) Application loader

→ load Entire .class file and keep it in Method area. (Part of data area)

→ by delegation work they do the Execution

→ bootstrap loader loads inbuilt classes. Eg scanner, string class.

→ Extension and Application loader loads class which we have written.

ⓑ Linking Stage :- (i) Verification stage

→ byte Code Verifier Verifies the class file.

(ii) Preparation stage:

→ it check in the class what ever is loaded for VIP (or) static variables.

→ if ~~static variables are there in heap area~~ ~~for memory static variables are allocated~~.

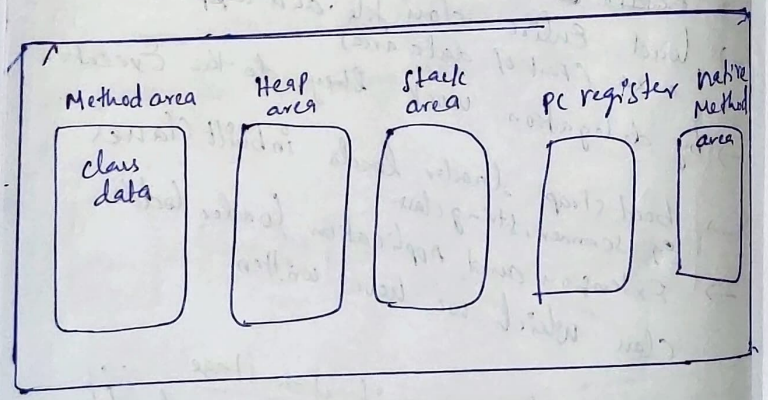
③ Initialization Stage:-

→ In initialization stage static block will be Executed.

⇒ Static Variables and static block get Executing during class loading itself.

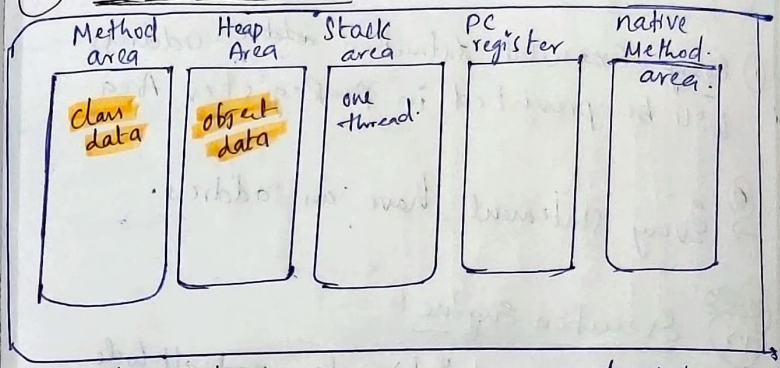
→ Class loader's System is part heap area.

JVM Data areas:-



→ if static Variables are there after verifying the Memory for static Variable allocated on heap area

1.4 JVM Data areas:-



① Method area:-

→ All the class data whenever it is loaded present in Method area.

class data → byte codes

→ Any static block (or Variables are Executed at initialization stage.

② Stack area:-

→ In order to Execute static block (or Variable that block will be brought into stack area and one frame will be created and that frame will be Executed.

→ Stack area used for Execution

③ Heap area:- the Memory for object data is given in heap area.

④ PC Register :-

① Every Executed instruction address will be presented in PC Register Area.

② → Every statement have an address.

1.5 Execution Engine :-

→ JVM will use interpreter, and all code Converted into 0's and 1's using interpreter and it will be Executed

⇒ JIT Complier used only at Specific time.

→ if a Method is called Multiple times then it is used.

Eg:- add() { } → add Method Converts into 0's and 1's

=

}

add() { → if Method is called another time JIT used the already Converted Code by interpreter.

=

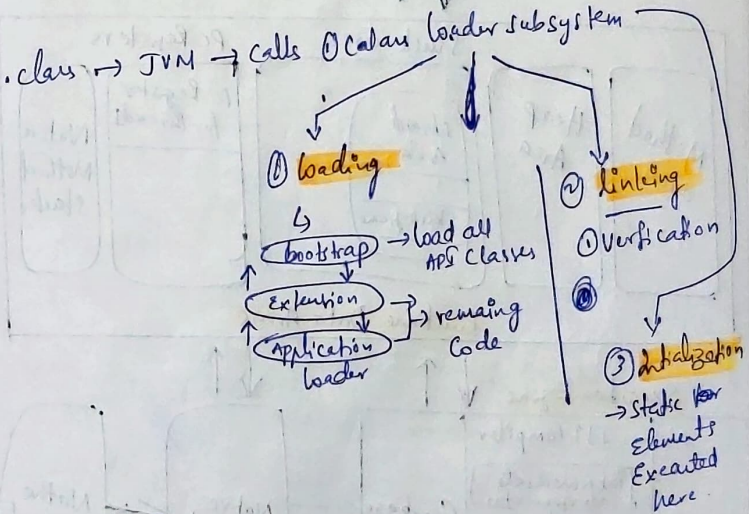
}

↳ Same 0's and 1's are used and get Executed.

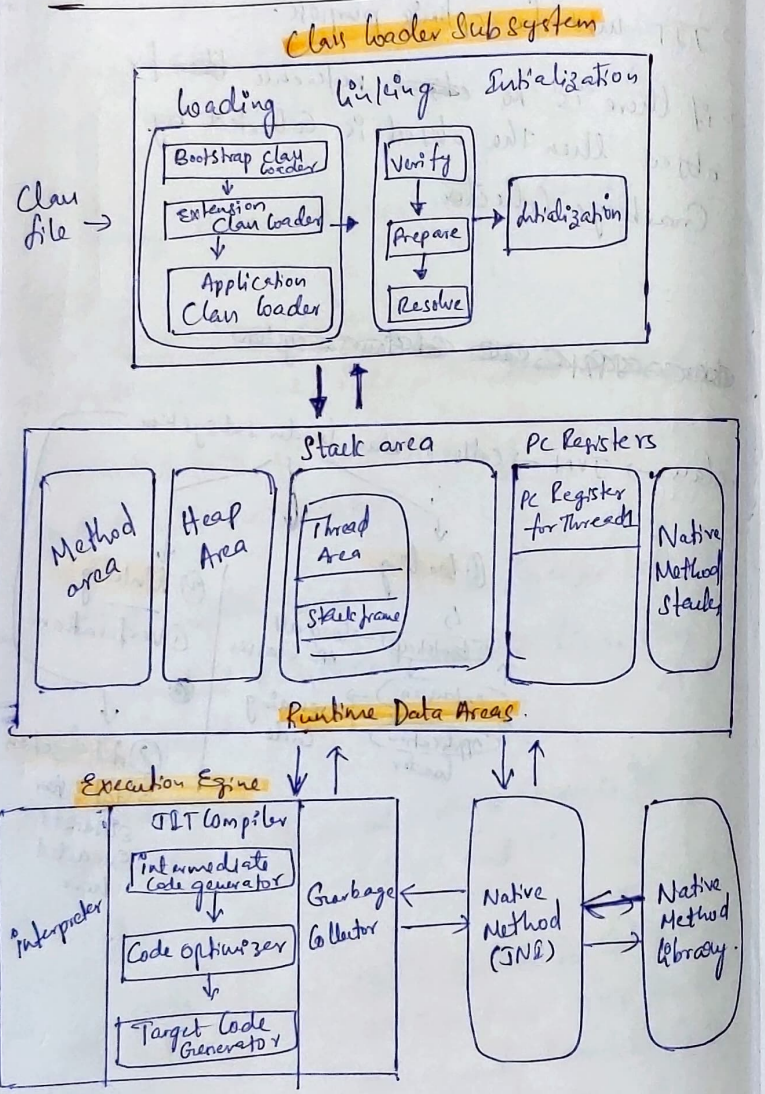
→ JIT avoids duplicates.

→ JIT used for optimise purpose.

→ if there is no reference for object then the object is Collected by Garbage Collector.



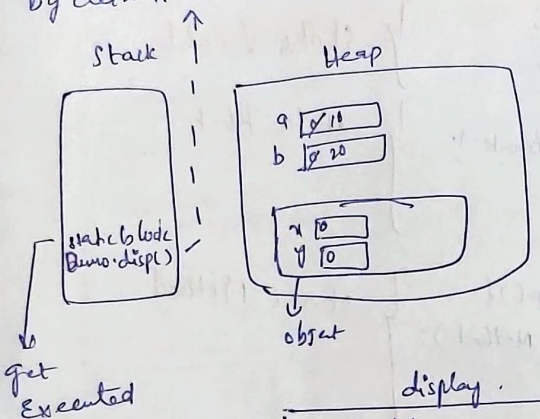
JVM Architecture Diagram :-



Code :-

```
Class Demo {
  static int a;                    } Static Variable
  static int b;

  static {                         } static block
  S.o.p("static block");
  a=10;
  b=20;
  }
  static void disp() {             } static Method
    S.o.p("static Method");
    S.o.p(a);
    S.o.p(b);
  }
  int x;                           } Instance Variable
  int y;

  {                                } non-static java block
    S.o.p("non static java block");
    x=30;
    y=40;
  }
  Demo() {                         } Constructor
     S.o.p("Constructor");
  }
  Void disp1() {                   } Non-static Method
    S.o.p("Non static Method");
    S.o.p(x);
    S.o.p(y);
  }
}
Public class static1 {
      Public static Void main(String[] args) {
        Demo.disp()  → accessing static Elements
        Demo d = new Demo(); → accessing non-static Elements
        d.disp1();
}
```

→ Static ~~Static~~ Methods are directly called by class name



display.

"static block"
"static Method"
10 — a
20 — b
Non static Java Block
Constructor
Non static Method
30 → x
40 — y

During object Creation 3 things happen :-

① Memory for instance Variable
② Java ~~B~~ block get Executed
③ Constructor will get Executed

→

→ Java block

```
{
S.o.p("non static java block");
x = 30;
y = 40;
}
Demo() {          → Constructor.
S.o.p("Constructor");
}
```

⇓

Behind Secens Java block included in Constructor.

```
Demo() {
{   → ~~Static~~ Java block  —①
}
   → body of Constructor.  —②
}
```

→ Java block Executes first ~~and~~ then body of Constructor Executes.

⇒ Memory for static Variable is thrion heap area.

Conclusion :-

① Static Variables - are Executed first. Memory will be on heap area. it Executed during Class loading.

② Static block → to Initialize static Variables
↳ it Executed during class loading.

③ Static Method →
① Main Method Executed automatically
② other static Method Executed
↳ after Main Method.
② We have to call Methods in order to Execute.

→ up to Main Method all thing Executed automatically.

④ during object Creation
① Instance Variable Memory will be Created inside object
② [illegible]
③ Java block get Executed.
③ Constructor get Executed.

Code :-

```
class Demo {
// static Variables
   static int a;
   static int b;
// static block
   static {
      S.o.p("static block");
      a=10;
      b=20;
   }
// b. static Method
   static void disp() {
      S.o.p("static Method");
      S.o.p(a);
      S.o.p(b);
   }
// Instance Variable
   int x;
   int y;
   {
      S.o.p("non static java block");
      x=30;
      y=40;
   }

Public class static1 {
      Public static void main(string[] args) {
         Demo.disp();
         Demo d = new Demo();
         d.disp1();
      }
```

Code-②

```
Public class static2{

    Static int a;
    Static int b;

System.out.println("static block");
a=10;
b=20;
}
Static Void disp(){
    S.o.p("static Method");
    S.o.p (a);
    S.o.p (b);
}
Public static Void Main (String[] args){
        S.o.p("Main Method");
        disp(); → Calling
}
```

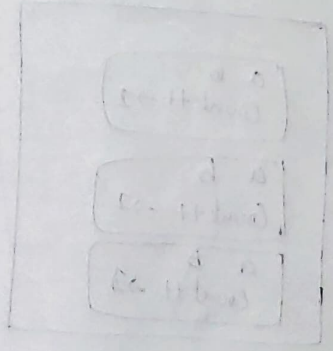
→ Static Variable and static block are Executed during class loading.

→ Main Method Executed by JVM.
  ↳ Executed by default.

→ We Can Call Method inside Method but we Cannot Create Method inside Method.

→ Memory instance Variable is given during object creation.

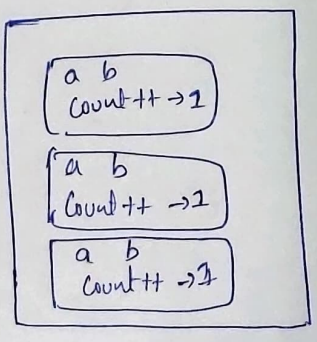
→ instance Variable have life when object Created.



Write a program to Count number of objects Created ?

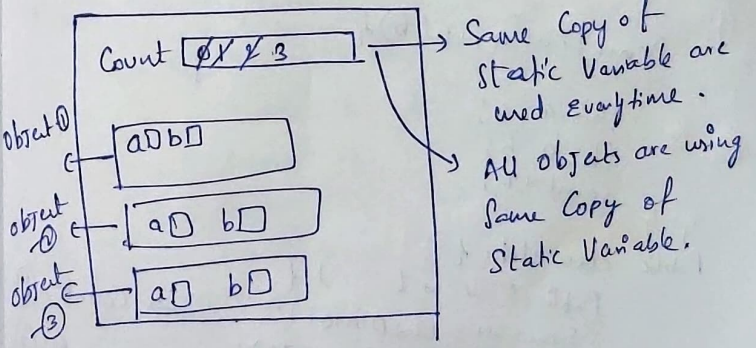
Code ③

```
Class Demo
{
  int a;
  int b;
  int count;
  Demo()
  {
    count++;
  }
}
Public class Static {
    Public static Void main (String [] args){
        Demo d1 = new Demo();
        Demo d2 = new Demo();
        Demo d3 = new Demo();
    }
}
```



Code ④ Every object uses same copy of Static Variable

```
Class Demo {
  int a;
  int b;
  static int count;
  Demo() {
    count++;
  }
}
Public class Static {
    Public static Void main (String [] args) {
        Demo d1 = new Demo();
        Demo d2 = new Demo();
        Demo d3 = new Demo();
    }
}
```

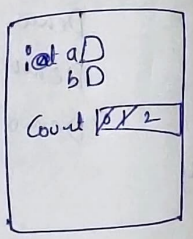


→ all objects are using same copy of static variables.

→ That's the reason static variables are called as class variables.

Code ③ :-

```
class Demo {
    int a;
    int b;
    static int count;

    {
        ~~count++;~~
    }

    Demo() {
        count++;
    }

    Demo(int a) {
        this.a = a;
    }

    Demo(int a, int b) {
        this.a = a;
        this.b = b;
        count++;
    }
}

Public class static {
    Public s v m() {
        Demo d1 = new Demo();      → count=1
        Demo d2 = new Demo(10);
        Demo d3 = new Demo(10,20); → count=2
        S.o.P(Demo.count);  //output: 2
    }
}
```



Code ④ :- Which content is common in all constructor write in non static block.

```
class Demo {
    int a;
    int b;
    static int count;

    // Non-static block
    {
        count++;
    }

    Demo() {                    → Constructor-1
    }

    Demo(int a) {               → Constructor-2
        this.a = a;
    }

    Demo(int a, int b) {        → Constructor-3
        this.a = a;
        this.b = b;
    }
}

Public class static {
    Public static Void main(String[] args) {
        Demo d1 = new Demo();
        Demo d2 = new Demo();
        Demo d3 = new Demo();
        S.o.P(Demo.count);   O/P : 3
    }
}
```

→ Constructor will include non static block and it executes it first then body of the Constructor.

Conclusion :- Static Variables

① to Create this Variable we use static keyword
② Memory for static Variable allocated during class loading.
③ Memory allocated in heap area.
④ Memory allocated only once in heap area for static Variables.
⑤ one Copy of static Variables used by all the object.
⑥ they are accessed by using class name.
⑦ they are also called class Variable.
⑧ they are object dependent.
⑨ they can be used inside static and non-static Elements.

Purpose of static Block :-

① to initialize static Variables.
→ Code inside static block Executed during class loading.

Code ⑤:

```
Class Demo5{
    static int a;
    static{
      a=10;
    }
    static void disp1(){
      s.o.p("value of a:"+a);
    }
}

Public class static{
    static void disp2(){
        s.o.p("Disp 2");
    }
    Public static void main(string[] args){
// disp1() Method is called using class Name
    Demo.disp1(); ——①
// disp1() Method is called using object
    Demo d=new Demo();
    d.disp1();  } ——①
// disp2() Method called within class.
    disp2(); ——③
}}
```

① Static Method is invoked using class name (or) object reference

② Method outside the class will be called using class name.

③ if Method is in same class then it is invoke by using Method name

④ if the value of variable are Constant use static block.

(1.6)	Static Method	Non static Method
1.	① Can be invoked using class name.	① it is invoked by creating object.
	② Can be invoked using object reference	② Can be invoked using Object reference
	③ it is object independent	③ object dependent.