15/11/22 (11) Classes, Objects, JVM Data Areas

(# Today's topic of discussion

(1) Introduction to oops (Classes and objects)

(2) Types of Variables

Division-1

(a) Primitive Variable

(b) Referen Variable

Division-2

(a) instance Variable

(b) local Variable

(c) static Variable

(3) JVM Area of Execution

(a) Method Area

(b) Heap Area

(c) stack Area

(d) PC-Register

(e) Native Method area

(1) OOPs (Object-oreinted programming System) :-

→ it actuall theory Concept, which is implemented by many programming language like C++, java, Python.

→ Any real time problem Can be solved if we follow's OOP's principle.

Note :-

(i) Softlware means Collection of many programs.

(ii) Programs means set of instructions.

(iii) To write instruction we need to have a language

→ (iv) OOPs :- object-oreinted programming is a methodology (or) Paradigm to design a program using classes and objects.

→ (v) object :- means a real-world Entity Such as Pen, chair, table etc

→ physical Existence of any Element we say as object.

→ In OOP's, while solving the problem we need to first Mark the objects.

Object Examples (1) Book My show

objects; Person, Ticket, Ticket issuer, Cinemahall, Chair, 3D-glasses, Screen.

→ All there are Virtually available in Mobile phone (software (or) App (on Web)

→ In OOP's, while solving the problem

① We need to first mark the objects.

② Every object we mark should have 2 Parts

ⓐ HAS - Part (store information as Variables)

ⓑ Does - Part (represents them as Methods)

→ HAS - Part / fields / attributes

→ Does - Part / behaviours

① What is Has-Part and Does-Part of an object represents?

ⓐ Has-Part :- What it Can hold.

ⓑ Does-Part :- indicates What it Can do

Eg:- ① student

ⓐ Has part → name, age, gender, address (Variables / identifiers)

ⓑ does part → What student do.

↳ play, study, sleep, drink. (methods)

↳ to represent has part we have Variable / identifiers

↳ for this we have methods.

(iii) Class :- A class is a user defined blueprint (or) prototype from which Objects are Created.

~~②~~ ~~To Represent~~

③ To represent an object, first we need to have a blueprint of an object.

~~① What is~~

(ii) Blueprint in java and how to represent it ?

→ In Java to represent a blue print we have a reserve Word called "Class".

→ Reserved word always starts with Small Case.

Eg:- ① // Blue Print of Student object

```
Class Student {
        int id;
        String name;
        int age;
        Char gender;
        String address;

        Void play () {

        }
        Void study () {

        }
        Void drink () { }
        Void Sleep () { }
}
```

① Has-Part (Variables / identifiers)

② does part (Methods)

(ii) Conventions followed by java developers While writing a Class :-

(a) Class name Should be in "Pascal Convention".

Eg:- (1) BufferedReader, FileReader

(2) OutputStream → 2 words.

(3) String

⇒ Pascal Convention :- is a naming Convention in which the first letter of Each word in a Compound Word is Capitalized.

(b) Variables are represented in "CamelCase".

Eg:- regNo, firstName, length

$\underset{\text{small}}{\text{reg}}\underset{\text{upper case}}{\text{N}}\text{o}$, first$\underset{\text{upper case}}{\text{N}}$ame

→ Variables are there to hold ~~[illegible]~~ Has-part.

→ Methods will be there to represent Does-part.

CamelCase Convention :-

→ Starts with a lowerCase letter and then Capitalized the first letter of Every Subsequent word.

→ Java follows Camel-Case Syntax for naming the Class, interface, method and Variable.

(c) methods are represented in "CamelCase"

Eg:- toUpper(), toLower(), toString, nextInt()..

(d) We use "new" Keyword / Reserve word to Create an object for a blueprint (Class)

→ To Create an object in java we use "new" Keyword.

Syntax:-

```
ClassName Variable = new ClassName();
```

↳ holding info of class.

↳ holds information regarding class name

↳ data type of Variable should be classname.

Eg int x = 10; → (Primitive data)

x → Variable

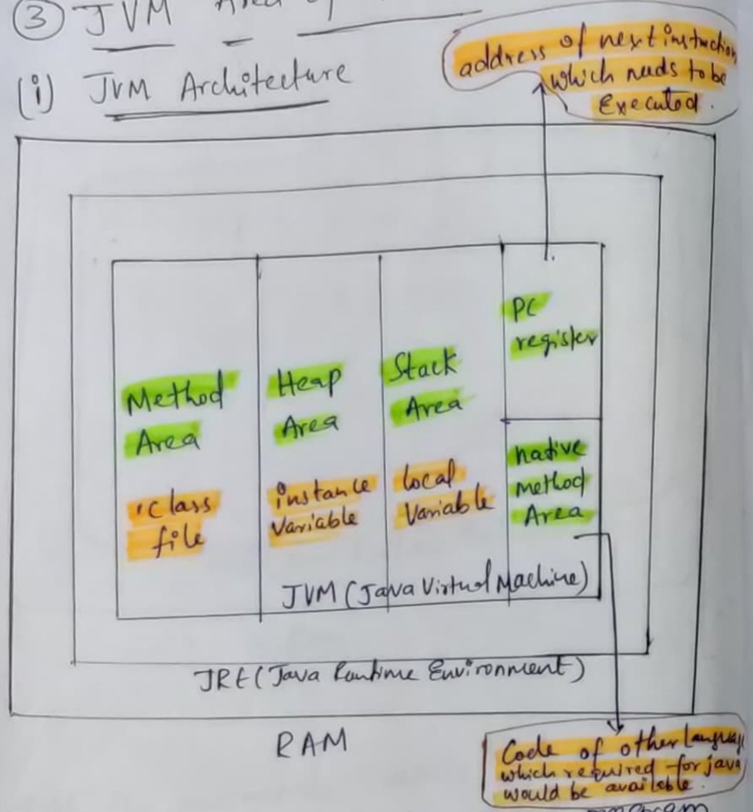
→ className(); holding information of class

→ new Keyword :-

↳ it is a Signal to jvm to Create Some Space for the object in the heap area.

③ JVM Area of Execution :-

(i) JVM Architecture



address of next instruction which needs to be Executed.
Method Area
'class file
Heap Area
instance Variable
Stack Area
local Variable
PC register
native method Area
JVM (Java Virtual Machine)
JRE (Java Runtime Environment)
RAM
Code of other language which required for java would be available.

→ At the run to Execute a Java program Space is given.

→ os allocates this space to Execute java Program (JRE :- Java Runtime Enviroment)

java filename ↑

⑥ → In Method area :- .class file presented in it.

→ Instance Variable in Heap area.

→ local Variable in Stack area.

→ address of next instruction which needs to be Executed presented in PC Register

→ Code of other languages which is required for java would be available ~~here~~ in native method Area

⇒ JVM Area for Execution

ⓐ Method Area (.class data/ static data)

ⓑ Heap Area (instance Variables / objectdata)

ⓒ Stack Area (local Variables)

ⓓ PC - Register

ⓔ Native method Area.

Code :-

```
Class student {
    int studentId;  }  // Has part (Variables)
    String name ;   }

  // Does part (methods)
    Void playCricket(){
        S.o.P("Student is playing Circket");
    }
    Void sleep(){
        S.oP(" student is sleeping");

    }
}  (Class)
// using student class class for testing code
Class Student class {
// Driving Code -> automatically Called by JVM
Public static Void main(String [] args){
//step① Creating an object of student class
      Student std = new Student();

}
```

Creation of Object

(i) Syntax:

ClassName Variable = new ClassName();

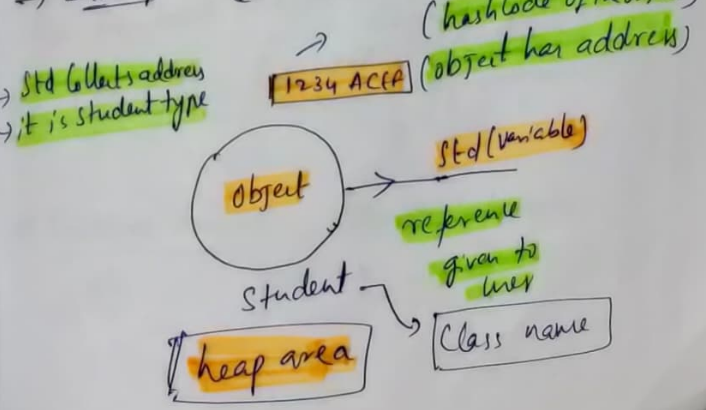
(ii) new keyword :- it is a Signal to jvm to Create Some Space for the object in the heap area.

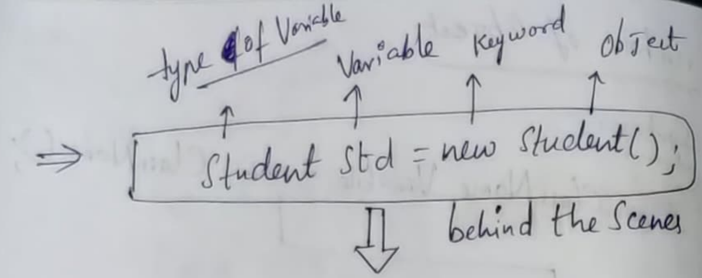
→ JVM asks for class name. (ClassName),

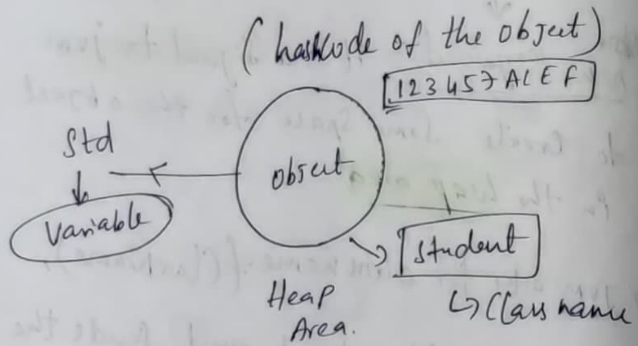
→ JVM Create the object and sends the " ~~hash Code~~ "hash Code" to the user.

→ user Should Collect the hash Code through " reference Variable ".

→ Example: $\boxed{\text{Student std = new Student();}}$







→ type of Variable is ~~class~~ same as Classname.

→ hash code is stored in Variable(std) and given to user.

(e) Every object should always be in constant interaction.

(f) ~~easy~~ Useless object doesn't Exists.

(g) Types of Variables :-

division-1 :- Based on the type of Value represented by a Variable all Variables are divided into 2 types. they are

1. Primitive Variables
2. Reference Variables.

1. Primitive Variables :- Primitive Variables can be used to represent primitive Values.

Eg:- int x=10

2. Reference Variables :- Reference Variables can be used to refer objects.

Eg:- $\boxed{\text{students s = new student();}}$

→ ~~Instance Variables~~ ~~std Student std1=~~

Division ② Based on the behaviour and position of declaration all Variables are divided into the following 3 types they are.

① **instance Variable** :-

Student Std 1 = new Student(); id = 10
name = Sachin

Student Std 2 = new Student();
↳ id = 10
↳ name = dhoni

→ if a Variables declared inside a class and outside the method are Called Instance Variable

(or)

→ if the Value of the Variable Changes from Object to object then such Variables are Called as "instance Variable".

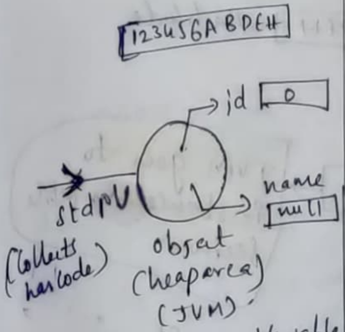
Eg:- ① Student Std 1 = new Student(); id = 10
name = Sachin

② Student Std 2 = new Student();
↳ id = 7
↳ name = dhoni

~~Student Std1 = new Student();~~

Student std 1 = new Student()
// filling the value to id, name

std 1.name = "Sachin";
std.id = 10



→ JVM goes to instance Variable Section

```
Class student {
    int id;
    String name;
}
```

→ int → 4 bytes.
→ JVM doing action in heap area. it gives default for Variables.

id 0
name null

Student Std2 = new Student()
// filling the value to id, name

Std2.name = "dhoni";
Std2 id = 7;

→ default Value of int is 0 when JVM do action in heap area.

→ null is default Value of String.

① Student std1 = new Student();

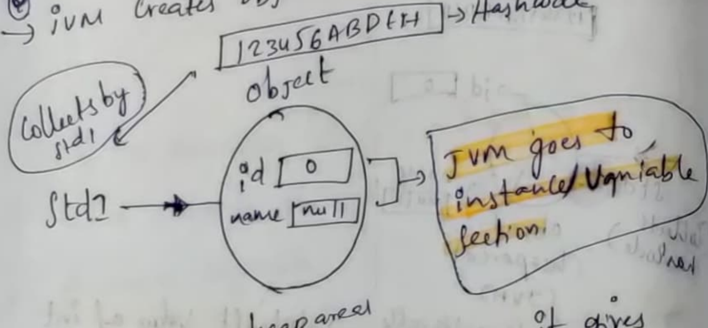
// filling the value to id, name

std1.name = "Sachin";

std2.id = "10";

⇓

② → JVM Creates object in heap Area



→ When JVM do action in heap area it gives default Value for Variable

→ int → 4 bytes

→ default value of int = 0

→ default Value of String = null.
↳ (object).

default Value of object reference

→ std1.name = "Sachin";
↳ Memory get activated.

filling the Values to id and name

① std1. name = "Sachin";
↳ Memory get activated
↳ accessing name
↳ null will be Overridden and Sachin given to name

name = ~~null~~
Sachin

② Id: std1.id = "10";
↳ 0 is over ridden and 10 is given as Id.

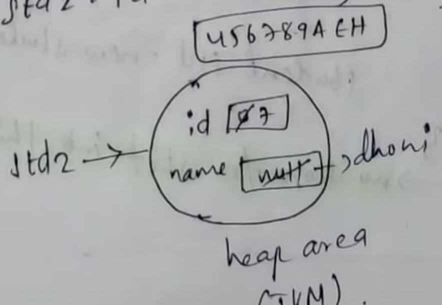
→ JVM doe this Job behind Secenes.

→ Instance Variables get Memory in heap area.

② Student. std2 = new Student();

std2.name = "dhoni"

std2.id = "7";



→ ~~when~~
→ when will the memory for instance Variable be given?

(A) when the object is Created. by JVM will Create a memory and by default jVM will also assign the default Value based on the data type of Variable

Eg int → 0
float → 0.0f
String → null.
boolean → false;
Char → Space;

→ when std Comes out main loop no scope for std Variable

```
~~Public static main~~
Class Test
{
    Public static Void main(String[] args)
    {
        student std = new student();
    }
} ———→ no scope for std into this line.
```

→ After no scope of std lose its Value

→ std1 and std2 not points to object

→ object is not accessed by them.

→ it Cannot access Variables in object also.

→ ① the scope of instance Variable will be available as long as reference pointed to it. (object)

Note ↕

→ ② if object reference becomes null, then we Can't access "instance Variables".

Eg

```
Public Class Test {
    boolean b;
    Public static Void main(String[] args){
        Test t = New Test();
        S.o.P(t.b);
    }
}
```

Eg ②

```
Public Class Test {
      int i=10; // instance Variable
   Public ~~void~~ static Void main (String [] args) {
        S.o.p (i); // → CE : instance Variable Can't
                            accessed directly in
                            static Context
                         └→ object not Created
        Test t=new Test(); → object Created
        S.op(t.i); // 10       → i=10 is store in
                                  heap are
                    └→ accessing i
                       using object accessing (i)
        t.methodOne(); → using object Calling
   }                     Method.
   Public Void methodOne() {
   // Inside instance method instance Variable
      Can be directly accessed.
   → S.o.p(i); // → 10 becoz it is a
                    instance Variable
   }
}
```

⇒ Key Points about instance Variables 16/11/22

① if the value of a Variable is varied from object to object such type of Variables are called instance Variable.

② For Every Object a seperate Copy of instance Variables will be Created.

③ Instance Variables will be Created at the time of object Creation and destroyed at the time of object destruction hence the scope of instance Variables is Exactly same as scope of objects.

④ Instance Variables will be stored on the heap as the part of object.

⑤ Instance Variables should be declared with in class directly but outside of any Method (or block (or Constructor.

⑥ Instance Variables Can be accessed directly from Instance area. But Cannot be accessed directly from Static area.

⑦ But by using object reference we Can access instance Variables from the static area.

② local Variables :- A local Variable in java is a Variable that's declared within the body of a method. Then you Can use the Variable only within that method.

④ memory would be on Stack Area.

Eg:
```
Class Test
{
    Public ~~void~~ static Void main(String[] args)
    {
        int a=10;   } local variables
        int b=20;
        int c= a+b;
        System.out.println(c);
    }
}
```

④ Variables which are Created inside the Stack area, are Called local Variables.

⑤ During the Execution of the Method the memory for local Variables will be given. and after the Execution of Method the memory of the Variables will be taken out from Stack.

→ ~~default value of~~

```
int d;
s.o.p(d);
```

(4) Local Variable default Value will not be given by JVM, programmer Should. give the default Value.

(5) if the programmer doesn't give default Value and if he uses the Variable inside the method the Program would result in "CE".

Eg (1)-

```
Public Class Test{
    Public static Void main(String[]args){
        int i=0;
        for(int j=0; i<3; j++){
            i=i+j;
        }
        s.o.p(j) -> CE-
                (j) Variable not
                declared.
        s.o.p(i) -> Valid.
    }
```

→ Scope of i is inside for loop.

Eg (2)

```
Class Test{
    Public static Void main(String[] args)
    {
        try{
            int i= integer.parseint("ten");   ] Scope of "i"
        }
        Catch (Null PointerException e){
            s.o.p(i);  // CE: (i) not declared.
        }
    }
```

Eg (3)-

```
Class Test{
    P S V m(String[] args){
        int x;
        s.op("hello");  // hello
    }
```

→ Code would be Compiled becoz it is not used any where.

[illegible]

[illegible]

Keypoints of local Variable :-

① Sometimes to meet temporary requiremen of the programmer we can declare Variables inside a method. (or) block (or) Constructor Such type of Variables are Called local Variables (or temporary Variables (or Stack Variables.

② local Variables will be stored inside Stack.

③ The local variables will be created as a part of the the block Execution in which it is declared and destroyed once that block Execution Completes. hence the Scope of the local Variable is Exactly Same as Scope of the block in which we declared.