

20th Oct 2022

Today's topic of discussion

Introduction to oops(classes and objects)

Types of variables

Division - 1

- a. primitive variable
- b. reference variable

Division -2

- a. instance variable
- b. local variable
- c. static variable

JVM Area for execution

- a. MethodArea(.class data/static data)
- b. Heap Area(instance variables/ object data)
- c. Stack Area(local variables)
- d. PC-Register
- e. Native method area

OOPs

It is actually theory concept, which is implemented by many programming language like c++, java, python, ...

Any real time problem can be solved if we follow oop's principle.

In OOP's, while solving the problem

1. We need to first mark the Objects.
2. Every Object we mark should have 2 parts
 - a. HAS-Part/fields/attributes (store the information as variables)
 - b. Does-Part/behaviours (represent them as methods)

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

4. we use "new" keyword/reserve word to create an Object for a blueprint(class). New keyword

5. Every Object should always be in constant interaction

6. Useless Object doesn't exist.

What is Object?

Physical existence of any element we say as Object.

eg: book, Car, Computer, Dog, Student,

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

HAS-Part => indicates what it can hold

Does-Part => indicates what it can do

eg: Student

|=> sid,

name, age, gender, email, address(variables/identifiers)

|=> play, study, drink, sleep(methods)

What is blueprint in java and how to represent it?

In java to represent a blue print we have a reserve word called "class".

Conventions followed by java developers while writing a class is

- a. className should be in "PascalConvention".

eg:

BufferedReader, FileReader, InputStream, OutputStream, String, ...

- b. variables are represented in "camelCase".

eg: regNo, firstName, lastName, length, javaFullStack

- c. methods are represented in "camelCase".

eg: toUpper(), toLower(), toString(), nextInt(),

eg#1.

```
//Blue print of Student Object
class Student{//Student -> PascalConvention

    //HAS-Part ----> camelCaseConvention
    int sid;
    String name;
    int age;
    char gender;
    String address;

    //Does-Part ----> camelCaseConvention
    void play(){}
    void study(){}
    void drink(){}
    void sleep(){}
}
```

To create an object in java we use "new" keyword

Syntax:

```
ClassName variable=new ClassName();
```

new -> it is a signal to jvm to create some space for the Object in the heap area.
Tell the className, we inform the classname, JVM create the object and sends the "hashCode" to the user.
User should collect the hashCode through "reference variable".

realtime example : BookMyShow

Objects : Person, Ticket, Cinemahall, Chair, 3D-glasses, Screen,

Note : Software means collection of many programs

Programs means set of instructions.

To write instructions we need to have a language.

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

Primitive variables:

Primitive variables can be used to represent primitive values.

Example: int x=10;

Reference variables:

Reference variables can be used to refer objects.

Example: Student s=new Student();

instance variable

If the variable is declared inside the class, but outside the methods such variables are called as "instance variables".

Instance variable

or
if the value of the variables changes from object to object then such variables are called as "instance variables"

eg#1.

```
Student std1= new Student();//id = 10, name =sachin  
Student std2= new Student();//id = 7, name=dhoni
```

When will the memory for instance variable be given?

Ans. Only when the object is created JVM will create a memory and by default jvm will also assign the default value based on the datatype of the variable.

eg: int -> 0, float-> 0.0f, boolean -> false, char -> , String -> null,....

Note: scope of instance variable would be available only when we have reference pointing to the object,
if the object reference becomes null, then we can't access "instance variables".

Key points about instance variables

Instance variables:

- => If the value of a variable is varied from object to object such type of variables are called instance variables.
- => For every object a separate copy of instance variables will be created.
- => 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.
- => Instance variables will be stored on the heap as the part of object.
- => Instance variables should be declared within the class directly but outside of any method or block or constructor.
- => Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area.
- => But by using object reference we can access instance variables from static area.

eg#1.

```
public class Test {  
    boolean b;  
    public static void main(String[] args) {  
        Test t=new Test();  
        System.out.println(t.b); //false //default value printed  
    }  
}
```

eg#2

```
public class Test {  
    int i=10; //instance variable  
    public static void main(String[] args) {  
        System.out.println(i); //CE: instance variable can't be accessed  
        directly in static context  
  
        Test t=new Test(); //Object created i = 10 is stored in heap area  
        System.out.println(t.i); //10  
        t.methodOne();  
    }  
    public void methodOne(){  
        //inside instance method instance variable can be directly  
        accessed.  
    }  
}
```

```

        System.out.println(i);//10 becoz it is an instance variable
    }
}

```

local variables

1. Variables which are created inside the method are called local variables and memory for those variables will be given in the stack area.
2. During the execution of the method the memory for local variables will be given, and after the execution of the method the memory for variables will be taken out from the stack area.
- ✓ 3. Local variables default value will not be given by the JVM, programmer should give the default value.
4. If the programmer doesn't give default value and if he uses the variable inside the method then program would result in "CE".

Keypoints of Local variables

Local variables:

=> Some times to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.

=> Local variables will be stored inside stack.

=> The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly same as scope of the block in which we declared.

=> It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.

eg#1.

```

public class Test {
    public static void main(String[] args) {
        int i=0;
        for(int j=0;j<3;j++)
        {
            i=i+j;
        }
        System.out.println(i);//valid
        System.out.println(j);//CE: 'j' variable not declared
    }
}

```

eg#2.

```

class Test {
    public static void main(String[] args) {
        try{
            int i=Integer.parseInt("ten");
        }
        catch(NullPointerException e){
            System.out.println(i);//CE: 'i' not declared
        }
    }
}

```

try block is the scope of i.

```

eg#3.
class Test{
    public static void main(String[] args){
        int x;
        System.out.println("hello");//hello
    }
}

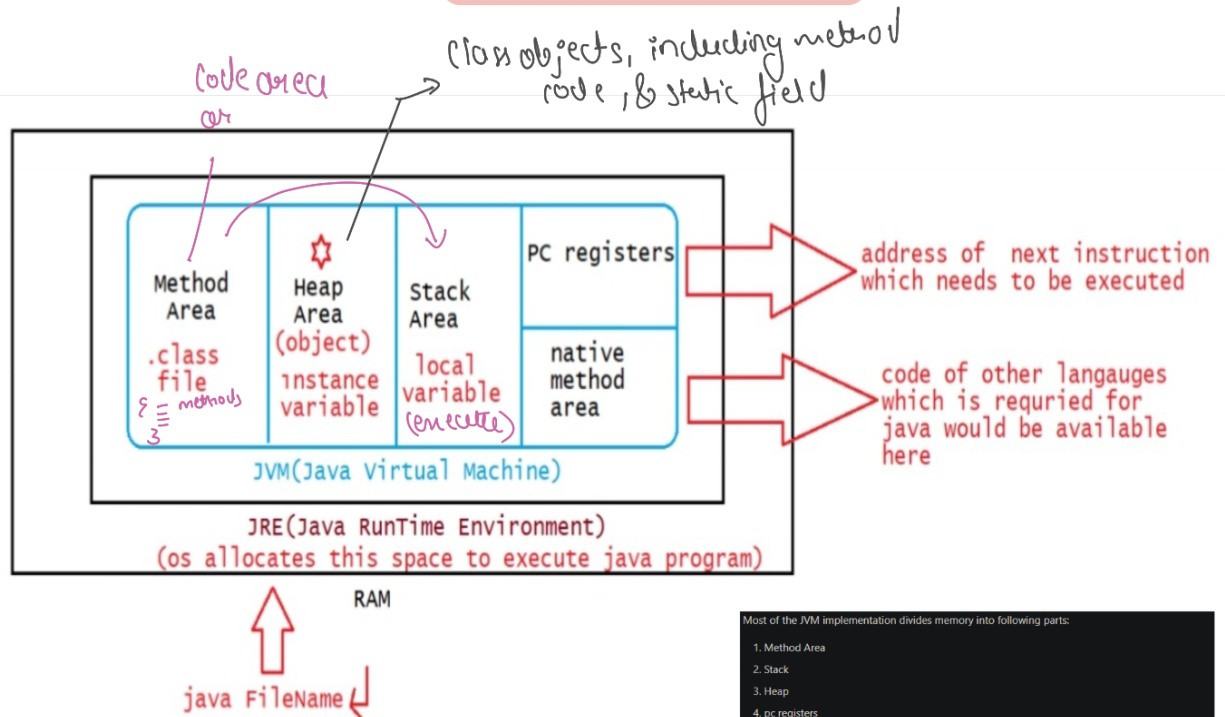
```

Note: code would be compiled becoz variable x is not used anywhere.

```

eg#4
class Test{
    public static void main(String[] args){
        int x;
        System.out.println(x);//CE: 'x' not initialized
    }
}

```



Most of the JVM implementation divides memory into following parts:

1. Method Area
2. Stack
3. Heap
4. pc registers
5. Native method stacks.

Lets talk about Method Area, Stack and Heap only.

For e.g Take a class

```

class Lava {
    int i = 5;
    static int j = 10;
    void flow() { //some implementation
    }
}

```

When an instance of this object is created from a class X

```

Lava l = new Lava();

```

First, Class type of Lava, i.e. Lava.class is stored in your Method area, with details like methods, fields and other referencing type. Also static variables like j in our example is stored in Method area itself.

Second the instance of Object Lava is stored in Heap Area as well as its instance variable i.e. i.

Third, Its reference, i.e. l in our example is stored in Stack area, which point to instance that is created in Heap.

```

student std = new Student();

```

behind the scenes

(hashcode of the object)
123457ACEF

