16/11/22 (12) Methods in Java and Fundamentals of Stack and Heap area.

(1) Methods :- Methods in Java is Collection of instructions that perform a specific task..

→ Method have

(1) name
(2) input (Parameters)
(3) Body
(4) return type.

```
 (4)        (1)      (2)
returntype name (Parameter)
{
    Activity / Body.
              (3)
}
```

→ Method Syntax

→ instance Variables, properties, field are refering Same

→ Method, Activity, behaviour, does part are refering Same.

Examp (1) Method approach - (1)

method not accepting Parameters but doing work.

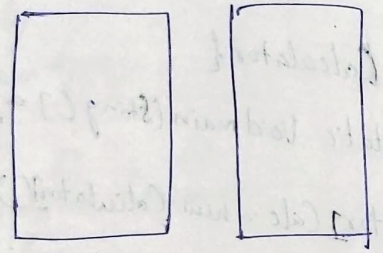
```
Class Calculator1
{
    int a,b,c;   // Instance Variables
   (4)   (1) (2)
    Void add()        → method
    {                 → no input and no output method
        a=10;
        b=20;   (3)
        c=a+b;
        System.out.Println(c);
    }
}

Public Class Calculator{
    Public static Void main(String[] args){
        Calculator1 Calc = new Calculator1();   → Object Creation
        Calc.add();   → add Method is Called
    }
}
```

→ Calc Managing object type of data
↳ Calculator.

→ Class Calculator1 got life after Creating object

→ name of the object is Calci

→ JVM executes Java program
↳ it arranges data area and Executes on it.



→ Any activity / task in program should brought in Stack area. for Execution.

→ whatever there in get Executed.

→ JVM got byte code of Entire written Program

↳ JVM got byte code of Entire written program.

Execution

(1) JVM got byte code of Entire written program.
JVM calls Main method.

(2) JVM gives control of Execution.

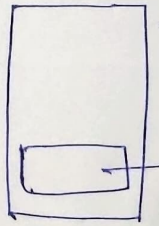
(3) In order Execute task inside main method on Stack area one record will be Created
[activation record of Main Method (or) Stack frame of main Method]

(4) body of Main Method brought into

Stack area.

```
class Calculator1
{
  int a,b,c;
  void add()
  {
    a=10;
    b=20;
    c=a+b;
    S.o.p(c)
  }
}
```

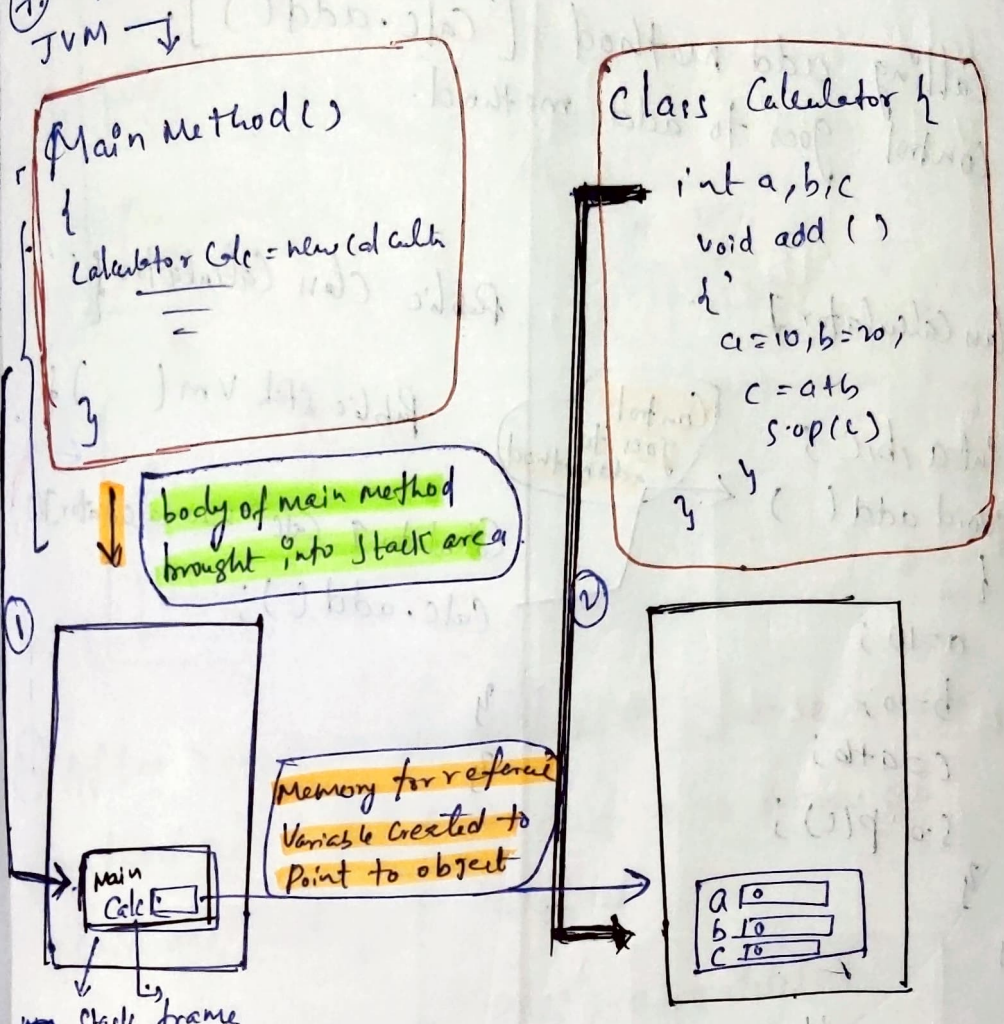
```
Public class Calculator{
  Public static Void main(){
    Calculator1 Calc = new Calculator1();
    Calc.add();
  }
}
```



(5) whatever there in Stack area that will get start Executing. object Creation is happened.

(6) on heap area Memory will allocated for Instance Variable (a,b,c).

(7) for objects Memory allocated on heap Area.



(4) default values are given to Instance Variable on heap area.

reference

```
Calculator1 Calc1 = new Calculator1();
```

(8) on Stack area Memory for reference Created

(9) Calc1 holds address of object.

(10) Calc° is reference Variable refering to object. on stack area Memory for reference Var created. it holds address of object.

```
Calculator I Calc = new Calculator I();
```
Calc → reference Variable

(11) Calling add method [Calc.add()].
Control goes to add method.

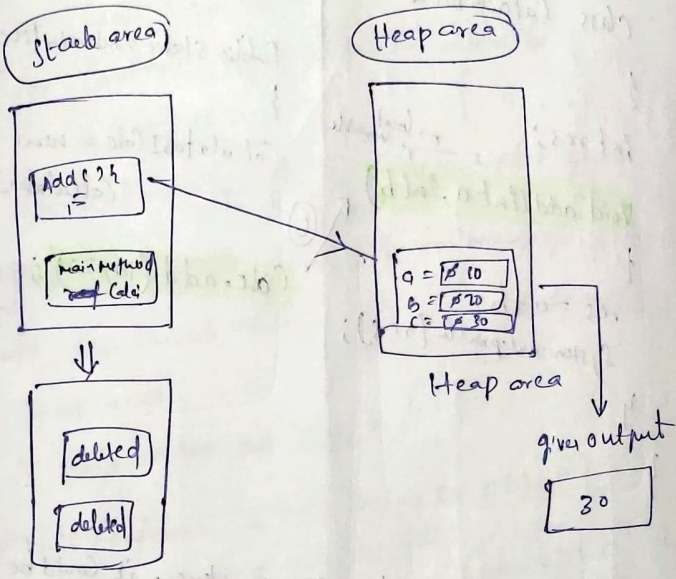
```
Class Calculator I
{
  int a, b, c;
  Void add( )
  {
    a=10;
    b=20;
    c=a+b;
    S.o.p(c);
  }
}
```

Control goes to add method

```
Public Class Calculator {
  Public stat Vm( ) {
    Calculator I Calc = new Calculator I();
    Calc.add( );
  }
}
```

(12) Whatever there in Add Method brought to Stack area.

(13) one Stack frame of add method will be Created.



(14) After Execution of the frame of add method get deleted and also frame of main method get deleted.

→ reference Variable (Calc°) also deleted.

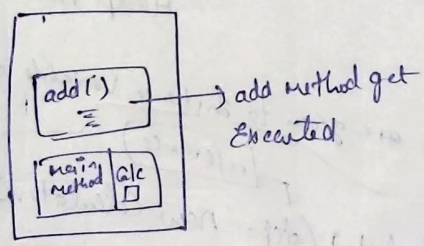
⇒ Garbage Collector Scans in heap area.

⇒ is there any object no one is refering?

↳ keep Scaning heap area for object no one is refering.

(15) garbage Collector will Collect this object Memory for this will be deallocated.

(9) Add method



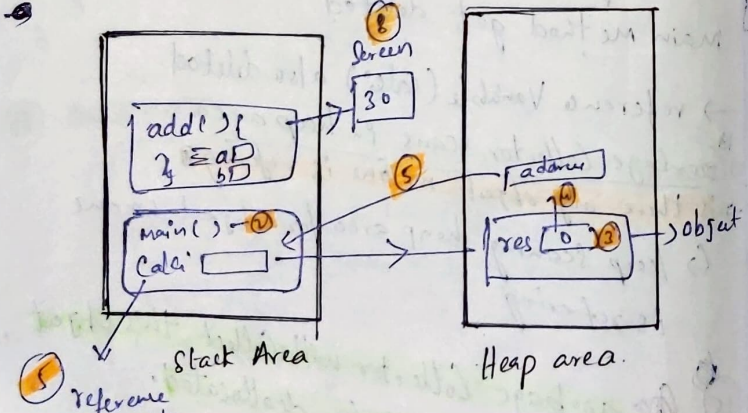
② Method Approach - ②

Method doing task by accepting parameters and which not returning anything

```
class Calculator1
{
int res;                          ← local variable
Void add(int a, int b)
{
 res = a+b
 System.out.println(res);
}
}
```

```
Public class launch Cal{
Public static Void main(String a)
{
Calculator1 Calc = new
                   Calculator1();
Calc.add(10,20);
```

→ add method accepting parameters.. it could be any type.



① Jvm gives Control of Execution.

② Stack frame of main method created on Stack area

③ on heap area object is created. and Memory for instance Variable will be allocated

res 0

④ default value for instance Variable given

⑤ Reference Variable :- responsible for managing object.

→ for Ref Variable memory will be created on Stack area.

→ it hold address of object.

→ Calc (Reference Variable refers to object)

Calculator1 Calc = new Calculator1

↳ Reference Variable

⑥ Calling add method :- [calc.add(10,20)].

↳ Call to method.

↳ Control of Execution goes to add method.

↳ Method Expecting some data which is called Parameters (a,b).

giver → Calc.add(10,20); ↳ arguments

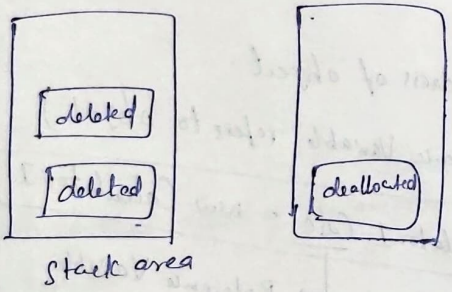
taker → Void add (int a, int b) ↳ parameters

(7) to Execute add mail method Stack frame of add method created on Stack.

↳ Memory for local variable allocated in stack frame.

(8) output is printed on Screen after Execution

(9) After Execution of add method Memory will be deallocated.



(10) Control comes to Calc.add(10,20).

↳ Memory of main method also deallocated.

↳ no ref. to object in heap area.

(11) garbage collector Collects the object and deallocate memorey of it.

(3) Method approach (3)

```
Class Calculator1
{
    int a,b,res;
    int add()
    {
        a = 10;
        b = 20;
        res = a+b;
        return res;
    }
}
```

(Return type)

```
Public class launchCalc1{
    Public Static Void main(String[] args){
        Calculator1 Calc = new Calculator1();
        // Calc.add();
        taking returned value
        int c = Calc.add();
        S.o.P(c);
    }
}
```

→ it is not accepting parameters. After Execution it is returning a value.

→ if method accepting parameters we must give parameters if it is giving or returning a value it is upto us to take or not.

→ Variable c is used to take value from add method.

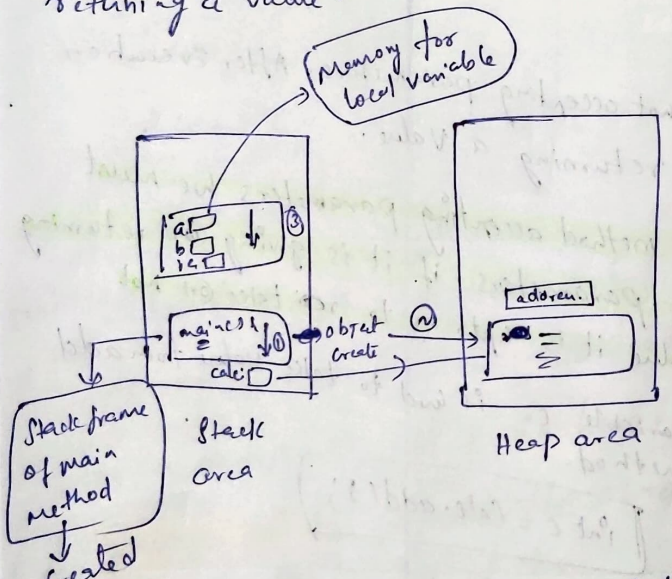
```
int c = Calc.add();
```

④ Method Approach – ④

```
Class Calculator {
    int add(int a, int b){
        int res = a+b;
        return res;
    }
```

```
Public class LaunchCalc1 {
    Public Static Void main( ){
        Calculator1 Calc = new Calculator1();
        // Calc.add();
        int res = Calc.add(10,20);
        S.o.p(res);
    }
```

→ here Method accepting Parameters. and it is returning a value.



① whatever there in Stack get Executed.
② Object is created on heap area
③ Control goes to add method. these add method is brought to stack area and get Executed.

④ on stack area Memory for local variable are Created.
⑤ After Execution Memory of Add method and Main method get deallocated.
⑥ garbage Collector Collects object and de allocate Memory of it.

Possible ways of writing Method :

①
```
Void add ( )
{
    a = 10;
    b = 20;
    c = a+b;
    S.o.p(c);
}
```
Calc.add( );
↳ add method is called.

② accepting parameter and not returning value

```
int res
Void add(int a, int b)
{
    res = a+b;
    S.o.p(res);
}
```
Calc.add(10,20);

③
```
int add ( ){
    a = 10;
    b = 20;
    res = a+b;
    return res;
}
```
```
// Calc.add();
int c = calc.add();
S.o.p(c)
```

④
```
int add(int a, int b){
    int res = a+b;
    return res;
}
```
```
int res = Calc.add(10,20)
S.o.p(res);
```

⑦ Possible Ways of writing Method:-

① Method not accepts parameters and not return any value

```
Void add ( ){   <-  Calc.add( );
  a=10;             ↳ add Method is
  b=20;               Called.
  c=a+b;
  s.o.p(c);
}
```

② Method accepts parameters and not return any value.

```
v.int res;
  Void main( int a, int b){   <-  Calc.add( );
    res = a+b;
    s.op (res);
}
```

③ Method not accepts Parameters and but returns value.

```
int add( ){   <-   // Calc.add( );
  a=10;            int c= Calc.add( );
  b=20;            s.o.p(c)
  res= a+b;
  return res;
}
```

④ Method accepting parameter

```
int add ( int a , int b) {   <-  int res = Calc.add(10,20
    int res= a+b;                s.o.p (res);
    return res;
}
```

Code Snippets (21/10/22) 1:45:32

①

```
int x= 0,1,2,3
Switch (x)
{
  default: s.o.p("default");    ↓ fall through.
  Case 0: s.o.p ("0");
          break;
  Case 1: s.o.p("1");           ↓ fall through.
  Case 2; s.o.p("2");
}
```

x=0 o/p = 0   x=3 (o/p = default = 0)

x=1 o/p = 1, 2 -> fall through

x=2 o/p = 2

②

```
Boolean b1 = true;     -> wrapper Class
boolean b2 = false;    -> Primitive
  "     b3 = true;
if ((b1 & b2) | (b2 & b3) & b3)  -> F    (bitwise operator return type -> boolean)
    s.o.p("alpha");
if ((b1 = false) | (b1 & b3) | (b1 | b2))  -> F
    s.o.p("beta");
```

-> no output