17/11/22 (13) Method overriding and Array Introduction

↳ Hyder Abbas Sir.

(1) Method overloading :- if a class has Multiple Methods having Same name but different in parameters, it's known as Method overloading

Eg :- Dev1

```
Class Calculator1
{
(1)=> int add (int a, int b)}
  {
    retun a+b;
  }
(2)=> int add( int a, int b, int c)
  {
    return a+b+c;
  }
(3) => float add (int a, float b)
  {
    return a+b; }
(4) => float add( float a, float b)
  {
    return a+b;
  }
(5) => float add (int a, float b, float c)
  {
    return a+b+c;
  }
(6)=> double add (int a, int b, double c)
  {
    return a+b+c;
  }
(7) => double add (double a, double b, double c)
  {
    return a+b+c;
  }
(8) => double add (int a, double b, int c) {
    return ~~a+b+c~~; a+x+c;
  }
```

=> Dev(2)

```
Public Class LauchMO {
    Public Static Void main(String[] args){
        Calculator1 Calc = new Calculator1();
        int a=10, b=30, c=20;
        float m=10.5f, n=20.5f, o=30.5f;
        double x=15.5, y=25.5, z=35.f;
S.o.p ( Calc.add(a,b)); → result print on Console.
S.o.p (Calc.add (m,n)); → adding two float num.
S.o.p ( Calc.add (a,b,c)); → Adding 3 int no.
S.o.P ( Calc.add (x,y,z));
S.o.P (Calc.add (a,b,x));
    }
}
```

① developers has to remember all methods name in a class in order to call if methods have different names.

② ~~in~~ to avoid this problem In Java class we Can write multiple methods with ~~different~~ Same names. but different in parameters.

③ Method overloading refers to process of writing more than one method with Same name and different parameters within same class.

④ developer Effort has reduced.

⑤ 1: many called polymorphism.

⑥ add - one method performing multiple activities
↳ false poly~~pho~~morphism. (illusion.

⑦ In reality one method performing one task.

⑧ 1 class → many methods
↓
Samename
↓
Same no. of parameters
↓
not Same data type

⑨ all add methods are active when called add method.

Calc.add(~~int~~ a ~~int~~ b)

↳ All add method which are accepting ~~method~~ two inputs

⑩ Compile time polymorphism :
Compiler resolving this issue based on
① Parameter (no. of Parameters)
② Data type of Parameter
③ Order of data type of Parameter.

→ Calc.add(a,b,x); → ⑥ becomes active.
int double

↳ Add method which is accepting 3 parameters. (2 integers, 1 double)

Eg ① double add (~~double~~ int a, ~~double~~ int b, double c){
return a+b+c; → This becomes active.
}

② double add (int a , double b , int c)
{
return a+b+c;
}

Eg ① Calc.add (~~int~~ a, b, x) → int → double.

→ ① Calls add method; All add method get active.
② ~~[illegible]~~ Methods with ~~two~~ 3 parameters get active
③ Methods ~~with~~ accepting 2 integer and 1 double get ~~value~~ active.

→ In built method in java ~~[illegible]~~ using Concept of Method overloading.

⇒ Inbuilt Methods are using Method ~~[illegible]~~ overloading

Eg. 2
```
System.out.println("hello");
System.out.println(a);
```

→ Method overloading also Called as Early binding / Compile time polymorphism.

⇒ Compiler resolve the Conflict:-
ⓐ no. of Parameters
ⓑ Data type of parameters
ⓒ order of data type of parameters

~~Eg~~ ② Coding Snippets

```
① int add(int a, int b)
  {
     return a+b;
  }
  Void add(int a, int b)  ← Cal.add(10,20)
  {                         → gives Compiletime Error
     int res = a+b;
     s.o.p(res);
  }
```

→ return type has no role play, it's only method name parameter.

② Method overloading with numeric ~~②~~ Promotions:-

```
~~float~~
float add(float a, int b)        implicit type conversion
{                                 int → float
  return a+b;
}                                s.o.p(Calc.add(10,20))
float add(float a, float b, int c)           ↓
{                                            int
  return a+b+c;                              ↓
}                                            30
```

→ System.out.println(~~[illegible]~~)
↳ is one of the statement of the program
↳ it is not output.

(3)

~~float add (int a, float b)~~ {

~~return a+c~~;

}

~~float add~~ (~~float~~

(3)

```
float add(float a, int b)
{
    return a+b;
}
float add (int c, float a)
{
    return a+c;
}
```

Calc.add(10,20);

↓

Calc.add(10, 20) → both accepts two parameters
↳ Method having Capacity to accept two integer values.
↳ 2 Methods have Capacity.
↳ Compiler get Comfused (or) ambigious
↳ not clear
↳ Compiler give Error

(4) General Method

```
Void disp() {
    S.o.p("ineuron");
}
Void disp(String name) {
    S.o.p(name);
}
Void disp(int age) {
    S.op(age);
}
```

(1)

d.disp();
d.disp(28);
d.disp("ashish");

→ Can we overload main method :-

We can overload main method however JVM will Call such a Main method which accepts String[] args as parameters.

```
Public static Void main (String[] args) {

(1) => PSVM(String[] args) {
        S.o.P("its actual main method");
      }
(2) => PSVM( int[] args) {
        S.o.P("it accepts int args");
      }
(3) => PSVM (double[] args) {
        S.o.P("double Values");
      }
```

→ JVM will ~~start~~ Search point

18/11/22

① **Array** :- Array is an object which contains Elements of a Simplar data type. Additionally the Element of a array are stored in Contigous memory location.

① **why array?**

① Convinient / Traditional way to store data is to Create Variable. not good approach to store

• Large volume of data.

② Large volume of data Stored in Single Variable (Array)

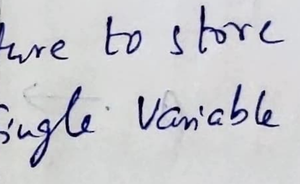
② **what is array?**

(i) array is indexed based data structure to store Large Volume of data using Single Variable name.

(ii) Array Can Store homogenius type data.

(iii) Array in java treated as object.
↳ memory stored in heap area.

```
              ↱ Type of data
int[] a = new int[5];  →
```

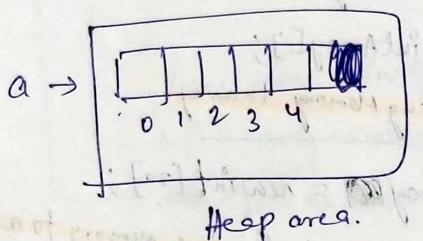


```
 |    |   ↳ a is array of integer
 |    ↳ [ ] ⇒ array ⇒ Collection of data.
 ↳ ~~integer~~ type of data.
```

(iv) Homogenius Data :-

array is Collection of Similar data. (or) Homogenous Data.

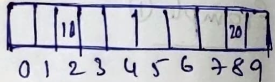
```
          ┌→ array of integer
int [ ] a = new int[5];
```



Eg① 10 - Students marks
↓int

→ a is array of integer. a is reference Variable. it is refering array of integer.

int [ ] a = new int[10];



a[2] = 10;
a[8] = 20;

// to access data in array.

S.o.P (a[2]); → Print data in 2
↳ $\boxed{10}$

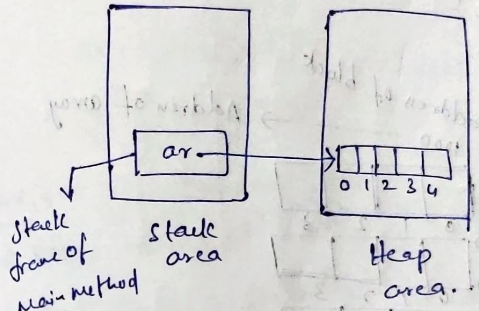
ID - Array

① Case-1 :- Create an array to store Marks of 5 Students

→. Students = 5.

int[ ] ar = new int[5];

→ Memory map.



Case ② :- 2D - array.

Create an array to store Marks of 3 classes Each With 4 Students.

Classes	Students
	4
0	4
1	4
2	4

data is regular

Regular / Jagged array.

→ In 2D array's we have

→ 2D - regular Array.

→ we have Create 2-D regular array for above data.

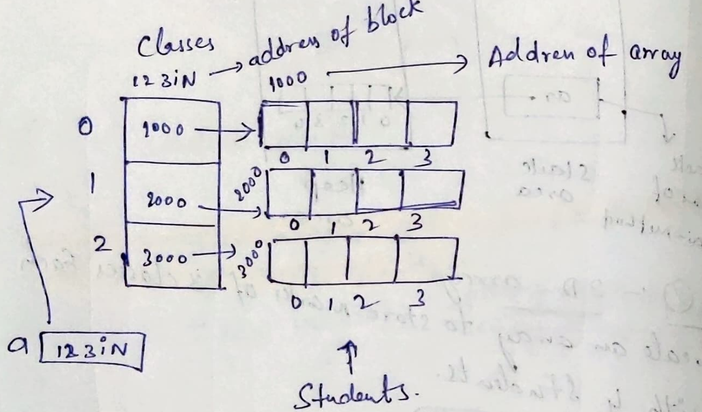
⇒ 2D Regular Array.

2 dimentional array of integer.

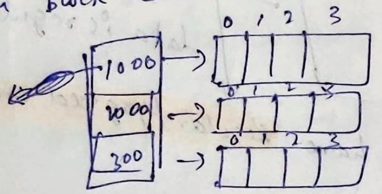
int [ ] [ ] ar = new int[3][4];

Classes Students.

⇒ Memory Map :-



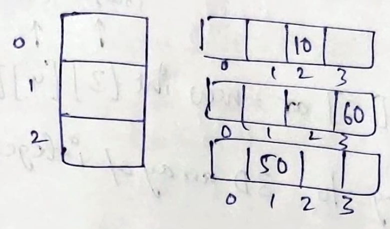
→ Each student block address stored in classes.



→ Address of class block stored in ar

ar 123iN

Classes



ar [0][2] = 10;

ar [2][1] = 50;

ar [1][3] = 60;

Case③ :- 3D Regular Array

⇒ store Marks of 3 Colleges 4 classes having.

Each Class 3 students.

→ 3 dimensional.

Colleges	Class	Students
0	0 → 3 1 → 3 2 → 3 3 → 3	
1	0 → 3 1 → 3 2 → 3 3 → 3	
2	0 → 3 1 → 3 2 → 3 3 → 3	

→ All Colleges have same no:of Classes and Students

→ it is Called 3D → regular array.

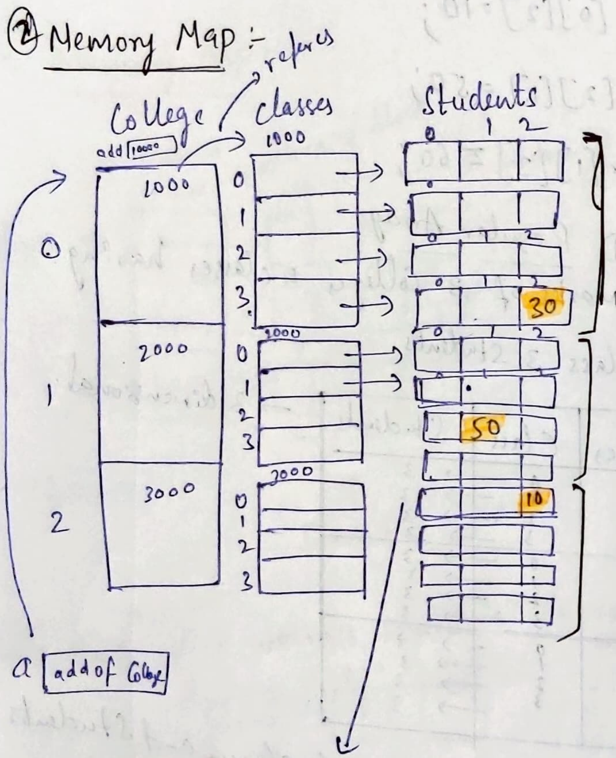
7 3D → Regular array :-

$$\begin{array}{ccc} \text{Colleges} & \text{Class} & \text{Students} \\ \uparrow & \uparrow & \uparrow \end{array}$$

```
int [ ][ ][ ] ar = new int [3][4][3];
```

→ ar is reffering to 3D array of integer.

2 Memory Map :-



```
ar [2][0][2] = 10;
ar [0][3][2] = 30;
ar [1][2][1] = 50;
```

Case 4 :- 2D Jagged array :-

→ 3 classes with different number of students
→ irregular data so it is called 2D-Jagged array.

Class	Students
0	5
1	3
2	4

→ 2D-Jagged array

we should not leave Empty.

```
int [ ][ ] ar = new int[3][ ];
```

not sure leave it

array of 0th class ↔ ar[0] = new int[5];
array " 1st class ↔ ar[1] = new int [3];
array " 2nd class → ar[2] = new int [4];

Case 5 :- 3D Jagged array

College	classes	students
0	0 1 } 2	4 2
1 } 3	0 1 2 3 } 4	3 1 5 2
2	0 1 2 } 3	3 4.

3D array of integer.

```
int[][][] ar = new int[3][ ][ ];
```

Colleges

Classes

```
ar[0] = new int[2][ ];
ar[1] = new int[4][ ];
ar[2] = new int[3][ ];
```

→ Zeroth college and Zeroth class students

Students

```
ar[0][0] = new int[4];
ar[0][1] = new int[2];

ar[1][0] = new int[3];
ar[1][1] = new int[1];
ar[1][2] = new int[5];
ar[1][3] = new int[2];

ar[2][0] = new int[3]
ar[2][1] = new int[4];
ar[2][2] = new int[2];
```

Case ⑤ College and Class are same but different no. of Students in Class.

```
int [ ][ ][ ] ar = new int [3][4][ ];
```

Coll Class Stu

```
ar[0][0] = new int[4];
ar[0][1] = new int[2];
ar[0][1] = new int[3];
ar[0][3] = new int[1];
```

⇒ for accessing Elements in 3D array.

ar[0][1][0] → Indexes

→ 1D, 2D, 3D ⇒ Regular
2D, 3D ⇒ Jagged.
4D ⇒

→ Multi dimensional array.