Method Overloading

What is the need of method overloading ?

Suppose there is need to add numbers and adding those numbers might have multiple cases like adding

1. 2 int numbers
2. 3 int numbers
3. 2 int numbers and one float number
4. 2 float and 1 int
5. 3 float numbers
6. 2 float numbers
7. 2 double numbers
8. 2 double and one float

And so on……

Creating methods with different names and for all these cases is a tedious task , and creating object for each individual method increases the program storage

To avoid this problem we use method over loading

Method overloading refers to writing more than one method with same name and with different parameters within same class

Eg: Method\_Overloading

One : many

One method performs multiple activities

Compile time polymorphism / Early binding :

When compiler starts execution from the main method

It goes line line by line execution ,

The compiler checks for object declarations in according to syntax rules or not .

After that it goes to the line where we called the method with the object

The compiler will go to the method

After that there are ,multiple methods with same name in the class , then compiler will check for

1. No of parameters
2. Data type of parameters
3. Order of data type of parameters

It locates the method that suits for the arguments we have passed in the main method and gets executed.

All this is done by compiler so it is called compile time , and polymorphism means poly -> means many , and morphism means forms here , so totally compile time polymorphism

Eg: println method is an example of method overloading .

We can write multiple println methods in a program and each of the them perform different task.

Return type has no role to play ,it is only method name and parameters.

Eg:Method\_Overloading\_Eg2

Eg: Method\_Overloading\_With\_Implicit\_Type\_Casting

Here in the arguments we have passed two int numbers the compiler checks for the method that accepts , 2 int parameters.

Since there is no method that accepts to int parameters , the compiler will check data types of the parameters . if the compiler finds a method that is suitable for implicit typecasting , the arguments get applied to that method and that method is executed.

Eg:Method\_Overloading\_Implicit\_Type\_Casting \_Eg2

// we pass the arguments in main method, and the compiler finds there is no method that has exact data type parameters, and compiler finds multiple methods for implicit typecasting , it shows compile time error .

Even though implicit type casting is possible for multiple methods the compiler doesn’t know to which method the argument should be passed . so it throws compile time error.

Eg: Method\_Overloading\_Eg3

//when there are two methods with same signature , and you make a call , which method is called will be a ? to compiler .

Compiler will never consider return type in binding the method call.

Array

Why array ?

suppose there is need to store 100 integer numbers , we can create 100 variables , but it is not a good approach , would be difficult to remember names of that variables.

To solve this we need array

What is array ?

Array is a index based data structure to store large volume of similar type of data using single name

In java arrays can store only homogenous type of data

Arrays in java is treated as objects , since they are treated as objects , they are stored in heap area .

Array syntax :

1. new ( since we need object for creating an array)
2. new int [5] ( if I want int type data to store declare as int , and I need to store 5 numbers so 5 is given in square brackets )
3. a = new int[5] ( Now a name should be given , let us consider a )
4. int[ ] a = new int[5] ( and a is an array of type int )

note : you have to specify the indexes in square brackets

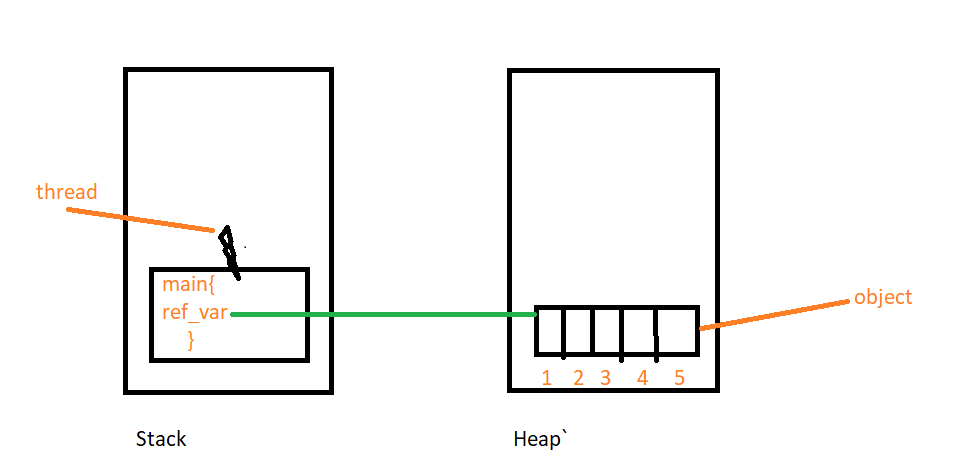
storing in array

1. array index starts from 0 , so if array size is 5
2. index of that array with size 5 starts from 0,1,2,3,4
3. if I want to store 50 in index 2 in array a[]

a[2] = 50

retrieve

to retrieve the element in index 2 , a[2]



Here are we are storing only single dimensional data so it can be said as single dimensional array

2d array

Eg: class students

1. 5
2. 5
3. 5

We are storing data for 2 dimensions, it is 2-dimensional array

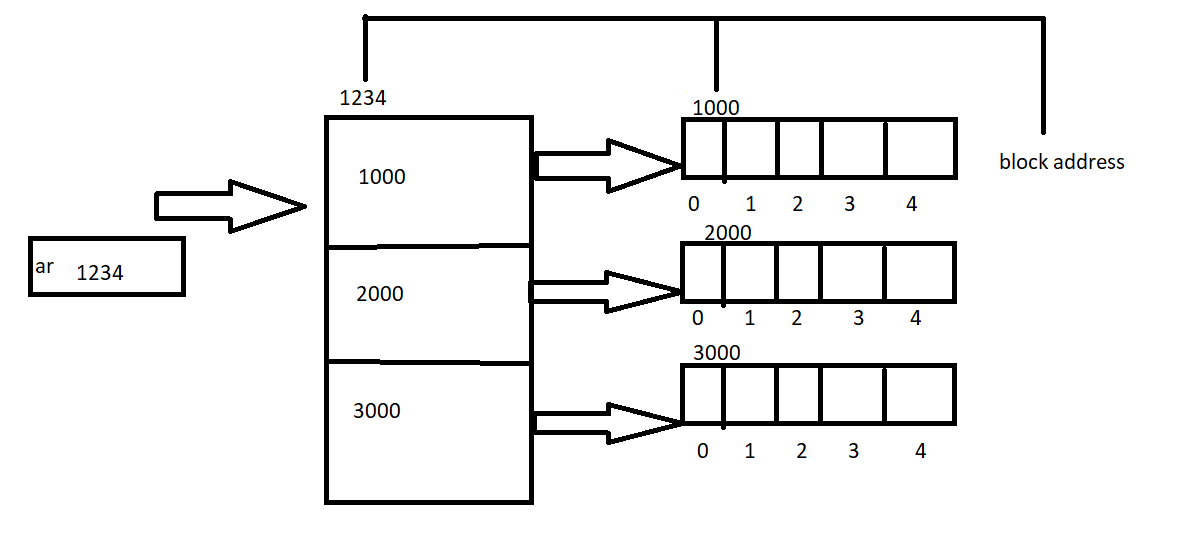
In arrays we have regular array , jagged array

Regular array 2d array

class students

Syntax : int [][] ar = new int [3] [5 ]

Memory map for 2 dimensional array



If I want to store the data 50 in class 0 and student 3 simply code will be

ar[0][2] = 50

here ar is a used defined name

3d regular array declaration :

Eg: store marks of 3 collages each collage 4 classes and each class 3 students

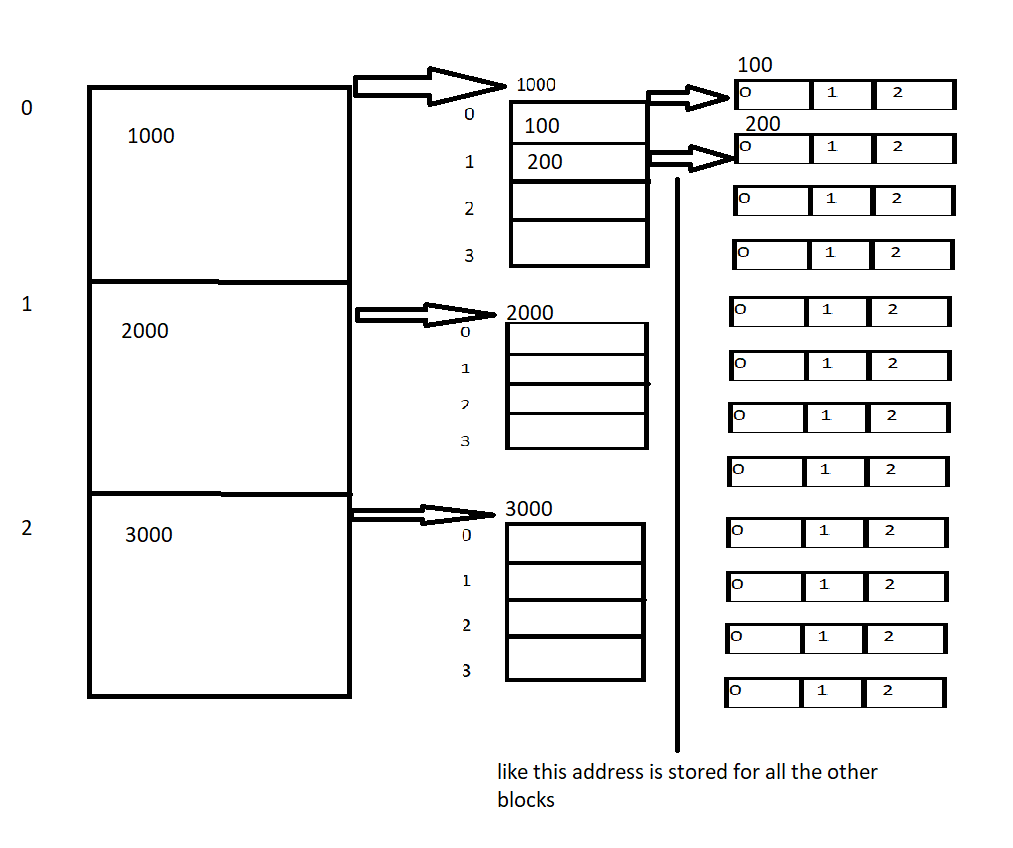
|  |  |  |
| --- | --- | --- |
| Collage | Class | No.of students |
| 0 | 0 | 3 |
|  | 1 | 3 |
|  | 2 | 3 |
|  | 3 | 3 |
| 1 | 0 | 3 |
|  | 1 | 3 |
|  | 2 | 3 |
|  | 3 | 3 |
| 2 | 0 | 3 |
|  | 1 | 3 |
|  | 2 | 3 |
|  | 3 | 3 |

Since there are three dimensions we can call it as three dimension arrays.

3-dimensional regular array declaration

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

memory map for 3d regular array



Eg: 3 collage, 2 class , 2 student marks can be stored as

ar[2][1][1] = 98

2d jagged array

|  |  |
| --- | --- |
| Class | Students |
| 0 | 5 |
| 1 | 3 |
| 2 | 6 |

Since there are two dimensions we can say it as 2 dimensional and data is irregular in the students we can say as 2 dimensional jagged array .

Syntax:

Int[2][] arr = new int [2][ ]

Since there is irregular data in the in students leave second one as empty .

Only this much syntax is not enough to create jagged array

arr[0] = new int[5]

arr[1] = new int[3]

arr[2] = new int[6]

3d jagged array

|  |  |  |
| --- | --- | --- |
| Collage | class | Students |
| 0 | 0 | 4 |
|  | 1 | 2 |
| 1 | 0 | 3 |
|  | 1 | 1 |
|  | 2 | 5 |
|  | 3 | 2 |
| 2 | 0 | 3 |
|  | 1 | 4 |
|  | 2 | 3 |

Syntax:

Assigning no of classes

Int[][][] arr = new int [3][][]

arr[0] = new arr[2][]

arr[1] = new arr[4][]

arr[2] = new arr[3][]

now assigning no of students

arr[0][0] = new int[4]

arr[0][1] = new int[2]

arr[1][0] = new int[3]

arr[1][1] = new int[1]

arr[1][2] = new int[5]

arr[1][3] = new int[2]

arr[2][0] = new int[3]

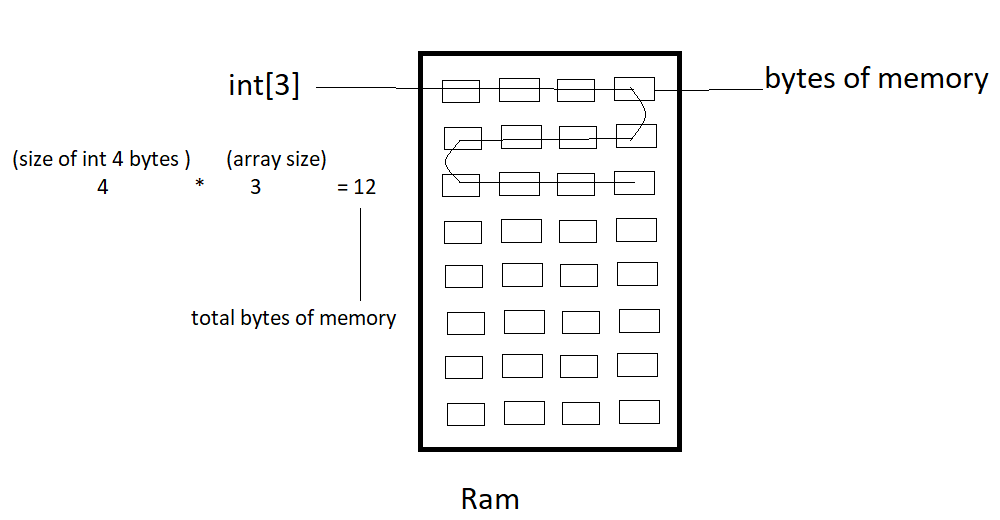
arr[2][1] = new int[4]

arr[2][2] = new int[3]

like this we can create 4d ,5d… regular and jagged arrays , so from that we can say arrays in java is multi-dimensional.

How array are secured in java ?

Buffer Overrun problem :



Here we allocated int type array of size 3 .

To store them we need

(int size is 4 bytes) 4 \*3(array size) = 12 bytes

And 12 bytes of storage is allocated in the ram during the execution

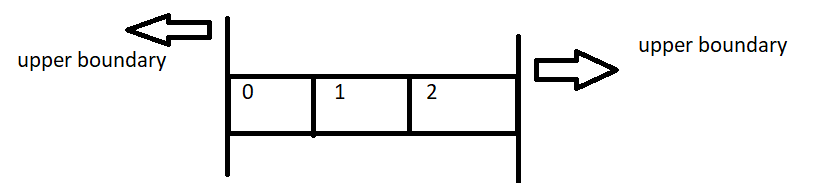
Since we assign the array size 3 , if we try to increase the array size , and add 4th element it is possible . but the actual size we have declared is 3 .

So memory on ram is increased . by chance if the byte memory has already some data it is overridden , it leads to serious problem.

This is called buffer over run problem .

**There is no buffer over run problem in java** , because there are boundaries for array in java .

Eg: int[] arr = new int[3]



We have declared the array size to 3 , we cannot increase its size ,since it has boundaries , if we try to increase it shoes runtime error (or) array index out of bonds exception.