3/9/23

Array:

⇒ A collection of similar datatype

Syntax:

(i) datatype[] variable_name = new datatype[size];

(ii) datatype[] variable_name;
variable_name = new datatype[size];

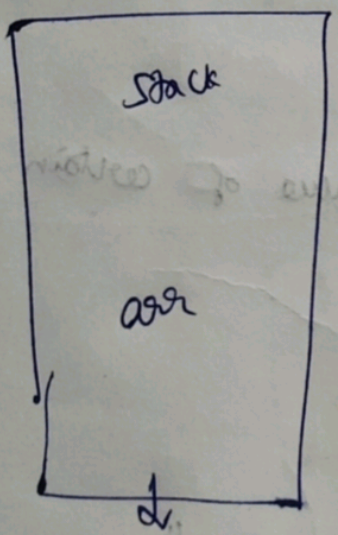
(iii) datatype[] variable_name = {value1, value2, ... valuen};

Example:

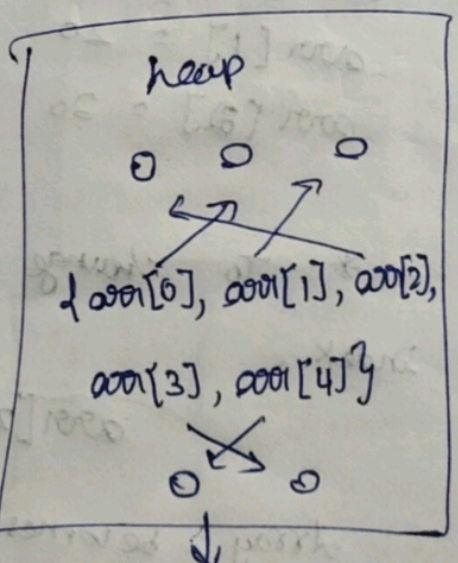
int[] arr = new int[5]

Initialize the variable and it will get initialized in stack during compile time.

* New → used to create a new object
* It will created in the heap memory during runtime



Reference variable Initialized



If reference variables having nothing to point, it will return
* "Null" for string array
* "0" for int array

* In Java, memory allocation totally depends on JVM whether it is continuous or not.

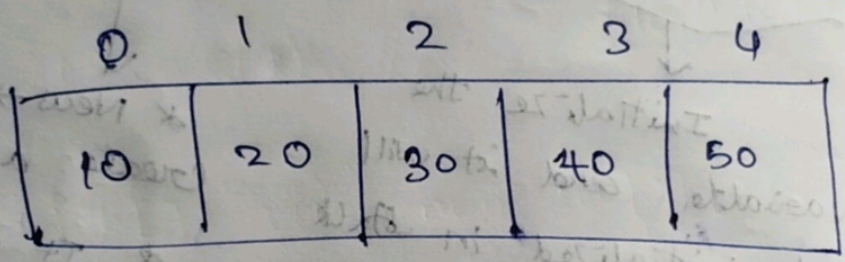
Reason:

⇒ Objects are stored in heap memory.

⇒ In Java language specification, it is mentioned, that helps object are not continuous.

⇒ Dynamic memory allocation. That's why array object may not be continous.

Index:



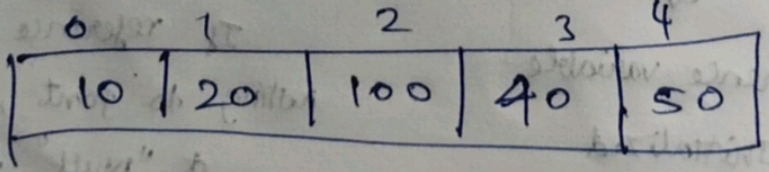
* Index starts from 0

arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50

* To change the value of certain index,

arr[2] = 100

- Array becomes,



toString()

1) import java.util.Arrays

2) Arrays.toString(arr)

⇓

Gives the output in proper format

=> Array is mutable (ie) changable but strings are immutable.

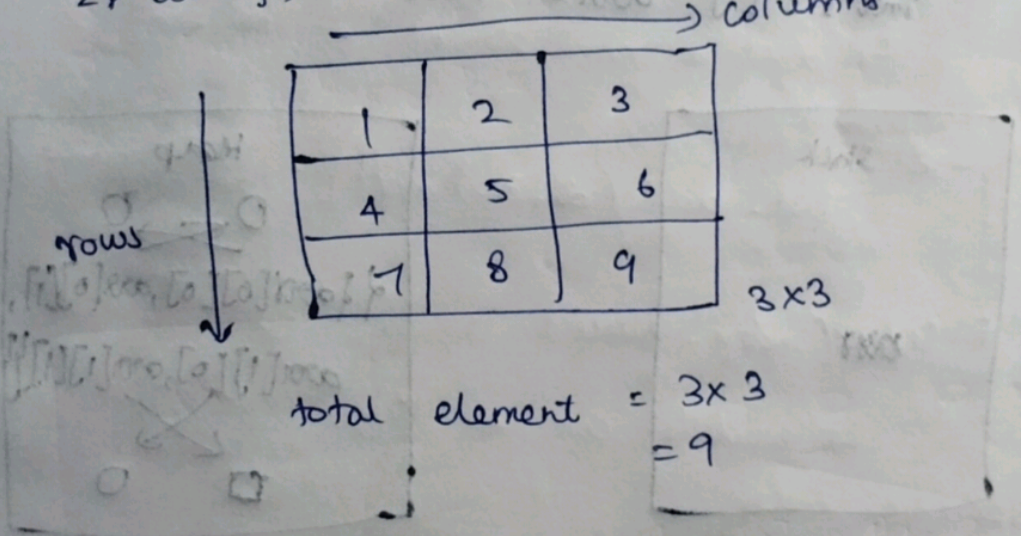
2D Array:

=> Looks or visualized like matrixs

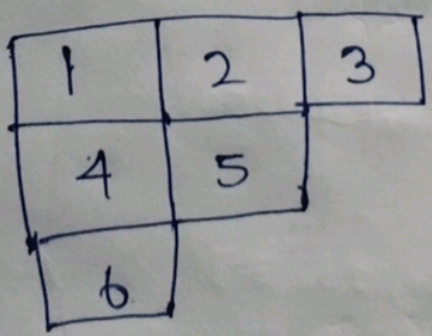
1D array like

int[] arr = {1,2,3,4,5};

2D array,



total element = 3x3
= 9



→ dynamic array

Row = 3

Col = dynamic

=> It is also possible in java

Syntax:

```
datatype[][]  variable_name = new datatype[
                row_size] [col_size];
```

(or)

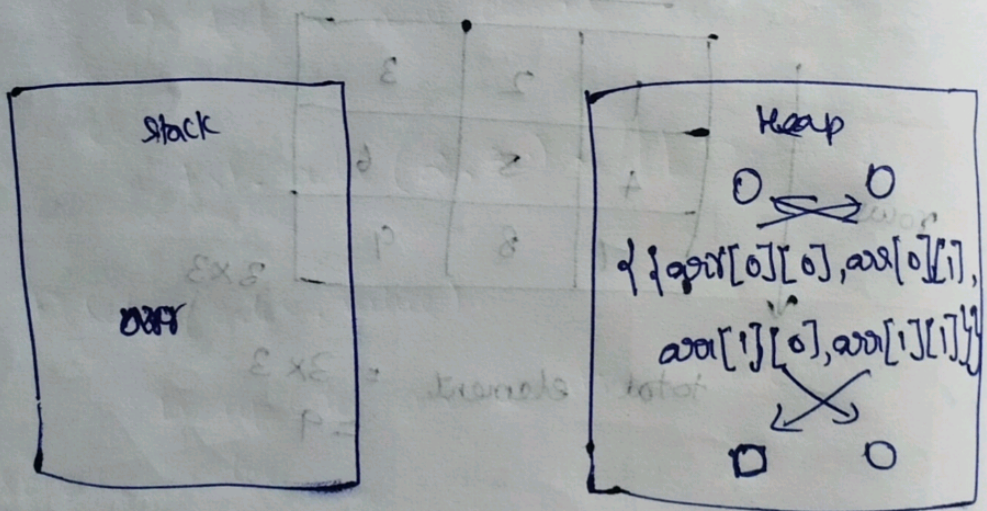
```
datatype[][] var_name ;
var_name = new datatype[row_size] [col_size];
```

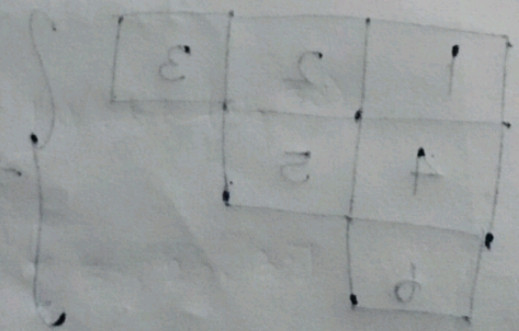
(or)

```
datatype[][] var_name = { {1, 2, 3},
                          {4, 5, 6},
                          :
                          {10, 11, 12} };
```

Example:

```
int [][] arr = new int [2] [2];
```





Array :

⇒ Fixed length or size

⇒ Created using both primitive as well as non-primitive datatypes.

To overcome the array, there will be

ArrayList

⇒ As many element as want even through an initial size is specified.

⇒ cannot be created using primitive datatype.

⇒ It can only be created using objects or wrapper classes

⇒ Import Package → import java.util.ArrayList;

Syntax:

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

- ArrayList → Class
- <Integer> → Datatype (wrapper class)
- list → reference variable
- new ArrayList<Integer>() → ArrayList Constructor

If you need string ArrayList,

```
ArrayList<String> list2 = new ArrayList<String>();
```

for boolean,

```
ArrayList<Boolean> list3 = new ArrayList<Boolean>();
```

Some methods in ArrayList:

* add()

↳ Add a new element to ArrayList

Syntax:

variable_name.add(10);

Example:

list.add(10);
list.add(20); // [10, 20]

* get(index)

↳ used to retrieve an existing value

ex: list.get(0); // Output: 20

* add(index, value)

↳ Add a new element in between

ex: list.add(2, 30); // [10, 20, 30]

* set(index, value)

↳ update an existing value for a specific index

ex: list.set(0, 50);

// output:

[50, 20, 30]

* remove(index)

↳ Delete an element in specific index

ex:

```
list.remove(0); // [50,20]
```

ArrayList:

```
ArrayList<Integer> arr = new ArrayList<Integer>(3);
```

↓ Array-size mentioned

arr = | 1 | 2 | 3 |

⇒ If decided to add more element to ArrayList. No enough space.

What will Java Do?

* It will Create a new list with new size (ie) its depend enough to accommodate new elements.

* It will copy the old list → new list.

* Old list will be deleted

arr = | 1 | 2 | 3 | 4 | 5 | |