Arrays in java

unit-1

- feat. of arr
- Implemented as obj.
- Arr r ref. var.
- Creating & initialzing arr.
- length
- 2D arr, jagged/irregular arr.
- Out of bounds checking
- Copying arr - shallow, deep copy.
- inline/anonymous arr.

Properties/features of arr

- Arr in java are
  1) Dynamic - new operator is used.
  2) implemented as obj - functions like clone() & length.
  3) ref var - passed as ref in fun.

Creating arr

- In C, C++, arr can be static or dy. but in java, all arr r dynamic. Arr can't be static. So new operator is used to create arr in java.

//steps to create arr.

- int ar[]; //declare a ref var/ arr var.
  ** do not specify size

  ar = new int [5]; //allocate m/m using new. Specify size here.

Note * in java bracke [] can be b4 arr name

eg • int []ar;
  ar = new int [5];

There steps can be combined.

- int ar[] = new int [5];

  ↙ . this shud be blank.   ↙ No. of elements.

ar ↑ of 5 el. is created

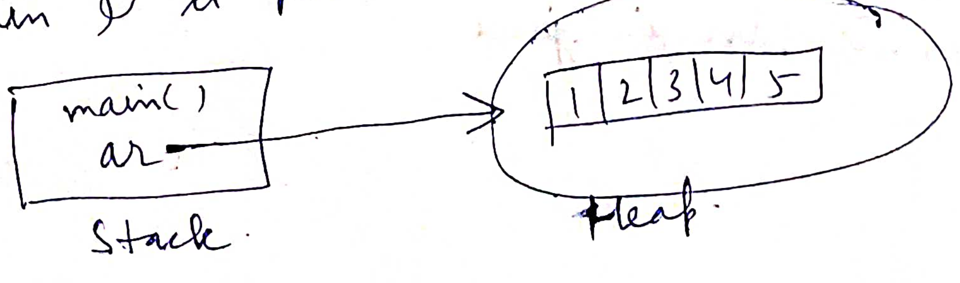
Size can be specified. using variable

int ar[] = new int [n];

↙ No. of el.

How r arr represented in java.

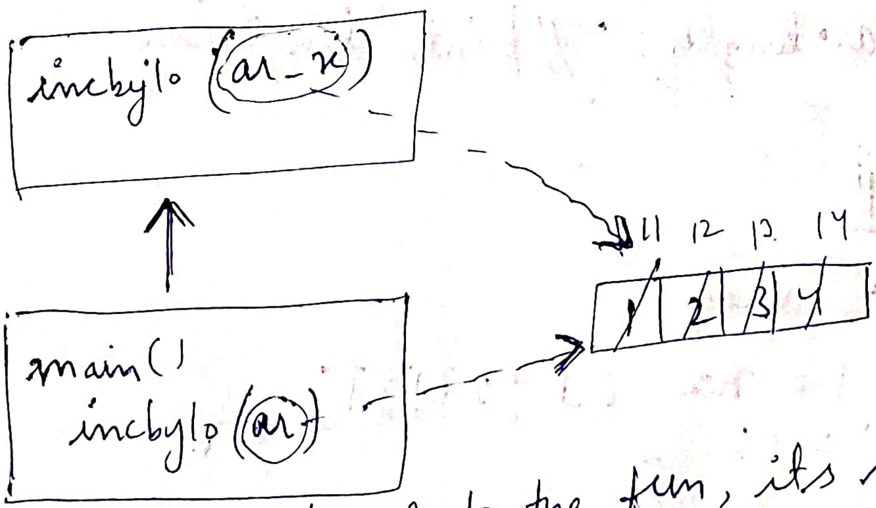
- Arr r implemented using obj.
- M/m is allocated in heap area.
- The ar var is a reference var inside main & it points to the ar in heap.



Arr are references, so fun is called by ref **

~~C D { P S V M ( ) {~~ //Class demo {
public static ...... {

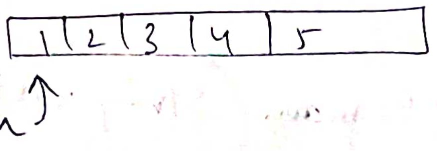
```
class demo {
    public static void main(String ar[]){
        int ar = {1, 2, 3, 4};
        incby10(ar); //ar passed to fun;
                     //called by ref ***
    }
    public static void incby10(int ~~arx~~ ar_x[]){
        //this fun increments each ~~el by 10~~
          element by 10.
        for(int i=0; i< ar_x.length; i++){
            ar_x[i] += 10;
        }
    }
}
```



• When arr is passed to the fun, its not passed by value. It is passed by reference., so both point to same array.

How to initialize arr

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



- when ar is initialized new opr is not used. ar of required size is created.

---

length of ar

- ar r represented as obj
- Each ar obj has a length var wh. contains size of arr.
- length is not a fun so length() sud not be used.

eg int ar[]={1,2,3,4,5};
    sopln (ar.length); // prints length/size.

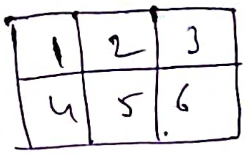
2D arr

① Creating 2D array.

int ar[][] = new int [2][3];
    ar[0][0] = 0;
    ar[0][1] = 1;   } initialize
    ar[0][2] = 2
    ⋮   ⋮   ⋮

② Creating & initializing 2D arr.

```
int ar[][] = { {1,2,3},      → row1
               {4,5,6} };    → row2
```

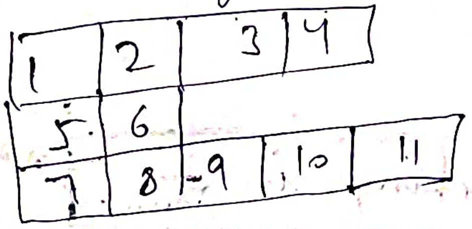


ar

//No need to use new.
//ar size is determined by No. of el.

Jagged / Irregular arrays ★★

• 2D arr can be created where No. of el in each row is diff.



// irregular / jagged arr

• Creating irregular arr.

```
1) int ar[][] = { {1,2,3,4},       → row 0
                  {5,6},           → row1
                  {7,8,9,10,11} }  → row2
```

```
2) int ar[][] = new int [3] [];    → leave blank.
                          → No. of rows...
   ar[0] = new ar[4];
   ar[1] = new ar[2];
   ar[2] = new ar[5];
   ↑            → # of el in each row.
   row #
```

How to print a jagged ar

```
int ar[][]= { {1,2,3,4},
              {5,6},
              {7,8,9,10,11} };

for(int i=0; i<ar.length; i++){
            # of row in 2D arr
   for(int j=0;j<ar[i].length; j++){
            length of ith row
        Sopln(ar[i][j]+" ");
   }
   Sopln();
}
```

Out of bounds checking in ~~ar~~ java ***

- In java out of bounds (oob) checking is performed by JRE at runtime.
- If you try to access ar el out of ar size then runtime error will occur & pgm will stop.

eg int ar[] = {1,2,3,4,5} ar of 5 el is created. If ar el larger than index 4 accessed then runtime error will occur.

```
Sopln(ar[0]); // OK. Within bounds.
Sopln(ar[1]); //
Sopln(ar[5]); // out of bounds error.
Sopln(ar[100]); //    Pgm will stop.
                      Runtime error occurs..
```

• In C, C++, out of bounds checking is not performed.
You can ~~access~~ index any el of ar.

```
int ar[] = {1, 2, 3, 4, 5};
Cout << ar[0] << ar[1] << ar[10] << ar[100];
```

• within bounds
• 1, 2 printed

• out of bounds.
• garbage value.

Why OOB checking done in java

1) bcoz of security.
- accessing ar out of bounds can allow a hacker to access other parts of system m/m.
- So to prevent this OOB access not allowed.

2) To prevent wrong O/P:
- it is illogical ~~to~~ access el out of bounds.
- it may result in wrong O/P.

Copying 1 ar to another

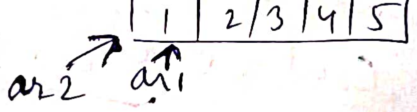
Shadow vs deep copy

~~ar st~~

- Ar variables r ref var. They should not be copied using assignment operator (=)

```
int ar1[] = { 1,2,3,4,5 };
int ar2[] = ar1;
```

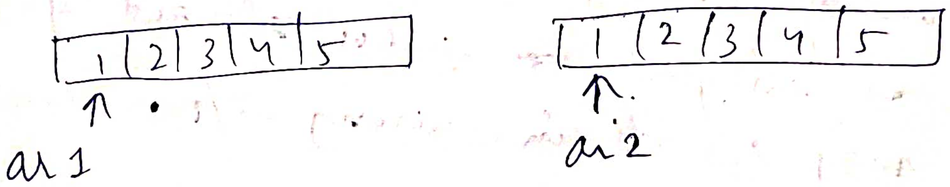
//shadow copy ~~is~~ occurs ~~created~~ using =
i.e both ar1 & ar2 point to same arr.



//shadow copy

- To copy ar use clone() fun.

eg
```
int ar1[] = {1,2,3,4,5};
int ar2[] = ar1.clone();
```



- "deep copy" is done using clone().
- In "deep copy" a new copy of ar is created & ar1 & ar2 point to diffent arrays!

Inline / anonymous arrays

- Inline arr r temp arr wh. r not stored in any var.
- They r created for 1 time use in any fun.
- syntax:-

$$\underbrace{\text{new}}_{\text{opr.}}\ \underbrace{\text{int}}_{\text{type of ar}}\ [\,]\ \{\underbrace{1,2}_{\text{el of ar.}}\};$$

//No ar name is specified.

eg create an ar & pass to fun.

```
int ar[] = {1,2,3};
add(ar);
```

eg2 Alternative :-

$$\text{add}(\underbrace{\text{new int}[\,]\{1,2,3\}}_{\text{anonymous array.}});$$