Decimal to Any Base

```
1 usage   new *
public static void decimalToAnyBase(int n, int b) {

    int ans = 0;
    int places = 1; // 10 ^ 0

    while (n > 0) {

        int rem = n % b;
        n = n / b;
        ans = ans + rem * places;
        places *= 10;

    }
    System.out.println(ans);

}
```

update

Places

n = 108

b = 4

$0 \times 10^0 + 3 \times 10^1 + 2 \times 10^2 + 1 \times 10^3$

$0 + 30 + 200 + 1000$

1230

base → decimal ✓

base → base ✓

$n = 798$

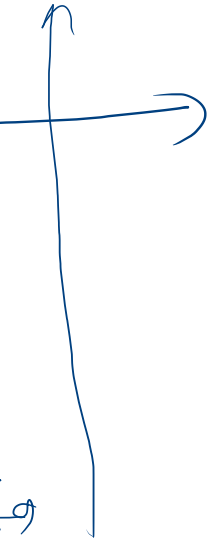
$b_1 = 10$

$b_2 = 8$

$[798]_{10} \longrightarrow [1436]_8$

$[1436]_8 \rightarrow [\quad]_2$

2	1436	
2	718	$0 \rightarrow 8^0$
2	359	$0 \rightarrow 8^1$
2	179	$1 \rightarrow 8^2$
2	89	$1 \rightarrow 8^3$
2	44	$1 \rightarrow 8^4$
2	22	$0 \rightarrow 8^5$
2	11	$0 \rightarrow 8^6$
2	5	$1 \rightarrow 8^7$
2	2	$1 \rightarrow 8^8$
2	1	$0 \rightarrow 8^9$
	0	1



decimal

1×8^2

$= 1 \times 64 \Rightarrow 64$

Any base → Any base

$b_1 \longrightarrow b_2$

Step 1 $b_1 \longrightarrow [\quad]_{10}$

Step 2 $[\quad]_{10} \longrightarrow b_2$

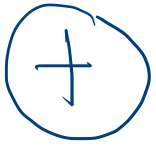
Why → + / * − → operators
works for decimal

Any box addition

$n_1 = 10$

$n_2 = 5$

$b = 10$

 → operator

default → $b = 10$

ans = n_1  n_2 = $10 + 5 = 15$ ✓

↓

$b = 10$

$n_1 = 72$

$n_2 = 16$

$b = 8$

$72 \oplus 16 = 88$ X

↓

défaut

$b = 8$

$n_1 = 72$

$n_2 = 6$

$72 + 6 = 78$ X

it is in decimal

default

base = 10

70

71

72

73 +1

74 +1

75 +1

+1

76

+1

77

+1

100

$n_1 = [72]_8 \longrightarrow [\quad]_{10} = [\quad]_{10}$

$n_2 = [16]_8 \longrightarrow [\quad]_{10}$

$b = 8$

$am = [\quad]_8$

n_1, n_2, b

Step 1 → $n_1 \longrightarrow [n_1]_{10}$

$n_2 \longrightarrow [n_2]_{10}$

Step 2 → $[n_1]_{10} + [n_2]_{10} = [sm]_{10}$

Step 3 → $[sm]_{10} \longrightarrow [sm]_b$

```
1 usage  new *
public static int anyBaseAddition(int n1, int n2, int b) {

    int decimalEqN1 = anyBaseToDecimal(n1, b);
    int decimalEqN2 = anyBaseToDecimal(n2, b);

    int decimalEqAns = decimalEqN1 + decimalEqN2;

    int ans = decimalToAnyBase(decimalEqAns, b);

    return ans;
}
```

S1

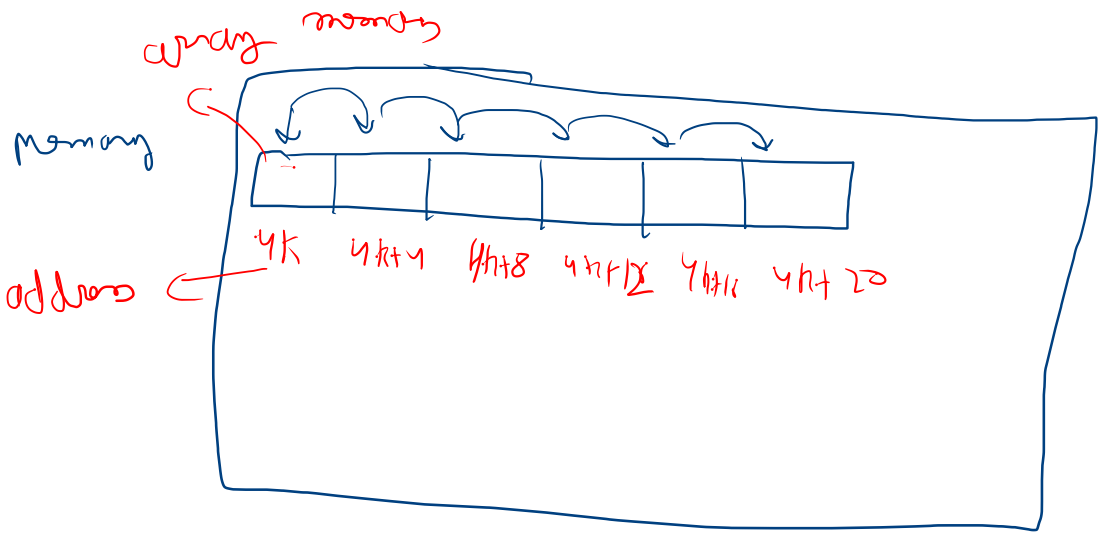
S2

S3

Arrays

→ Present Continuous in memory → Collection of similar datatype

→ Non-primitive



Syntax =>

int array => int [ ] arr = new int [size]

int → Primitive

arr → variable

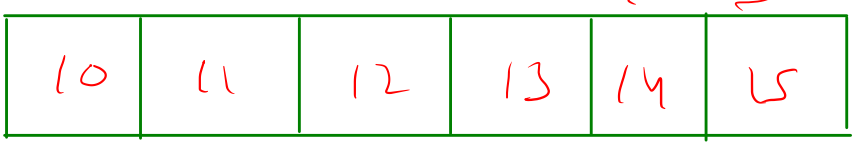
String array => String [ ] strArr = new String [size]

String → Non Primitive

strArr → variable

Size = 6 index = 0 1 2 3 4 5

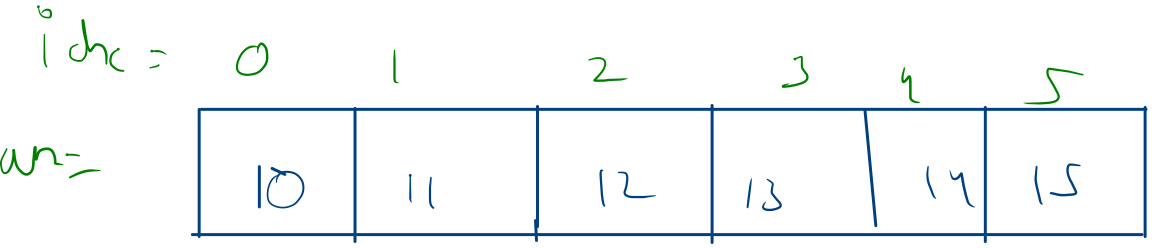
Represent →



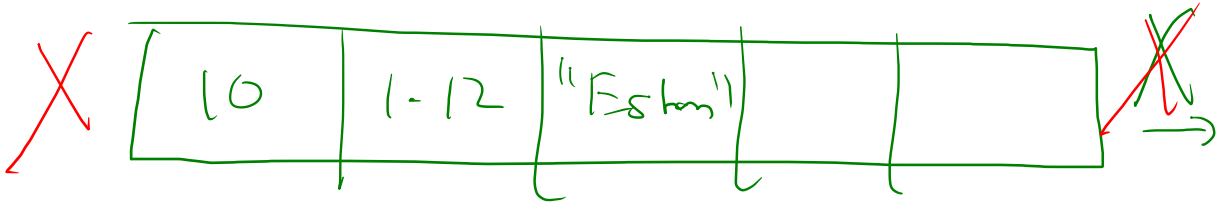
1st 2nd 3rd 4th 5th 6th

index = Position - 1

array → collection of similar data type & It is continuous in memory



integer → datatype is similar



→ not collection of similar datatype

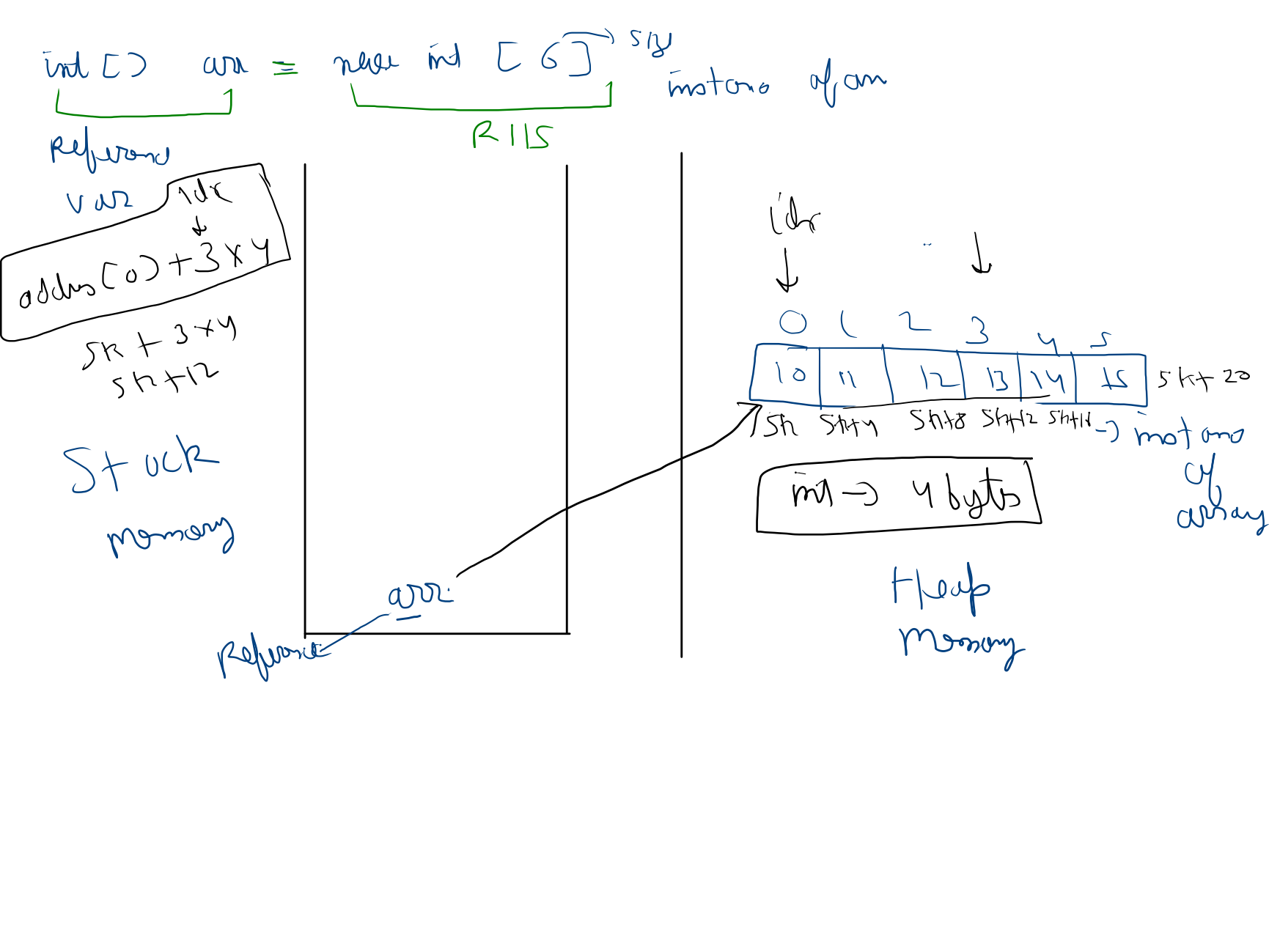
array → Continuous in memory.

```
int[] arr = new int[6];
```

(size)

LHS — RHS

1. Keyword ⇒ new → Represents non primitive → you are deciding to allocate memory
2. LHS → int[] arr → reference variable arr
3. RHS ⇒ new int[6] → instance of array / object


int[] arr = new int[6]
size
instance of arr
R15
Reference
var
idx
address(0)+3x4
5k+3x4
5k+12
Stack
memory
arr
Reference
idx
0
1
2
3
4
5
10
11
12
13
14
15
5k+20
5k
5k+4
5k+8
5k+12
5k+16
instance of array
int → 4 bytes
Heap
Memory

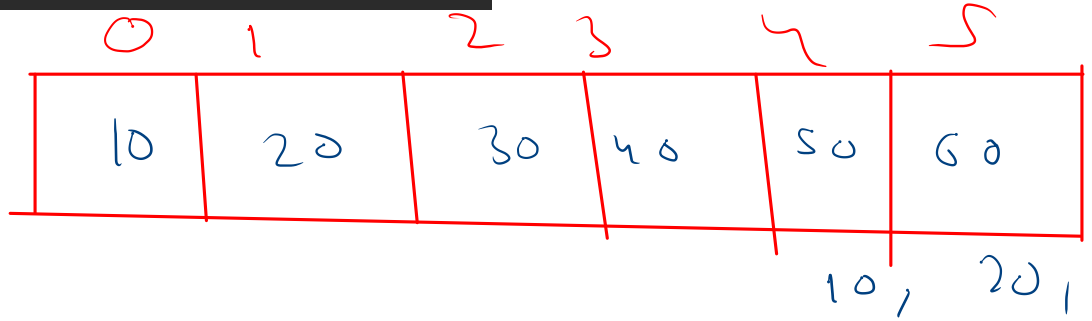
```java
public static void intArr() {

    // creation of arr
    int[] arr = new int[6];

    // 1st position ⟶ 0th idx
    arr[0] = 10;
    // 2nd position ⟶ 1st idx
    arr[1] = 20;
    arr[2] = 30;
    arr[3] = 40;
    arr[4] = 50;
    arr[5] = 60;
//      arr[6] = 70; ---> Error is arrayIndexOutOfBounds

    int val = arr[0];
    System.out.println(arr[0] + ", " + arr[1] + ", " + arr[2]);

}
```



val = arr[0]
val = 10

10, 20, 30

Java don't have garbage value

Garbage Collector → Java FIFO

↓

Clean your unused memory

Default an all = 0

int = 0 → default