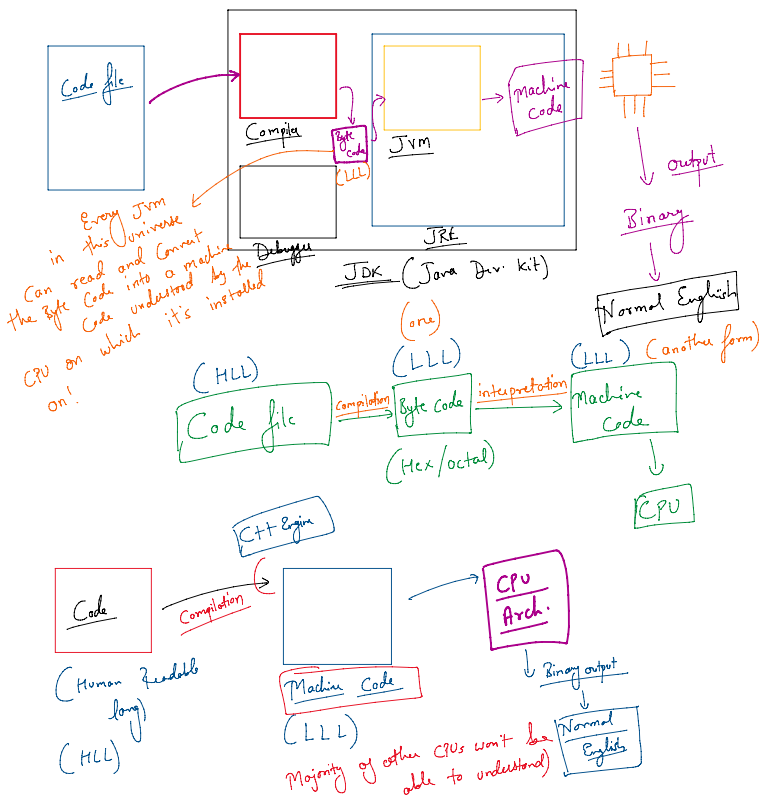
Core Java (Basics)

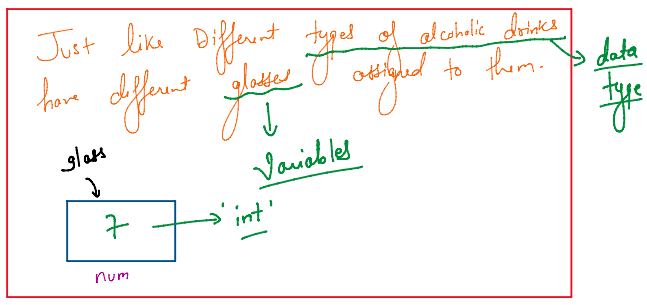
04 December 2023 18:28

★ How Java Code Runs ? ★



★ Variables & Data Types ★

Container



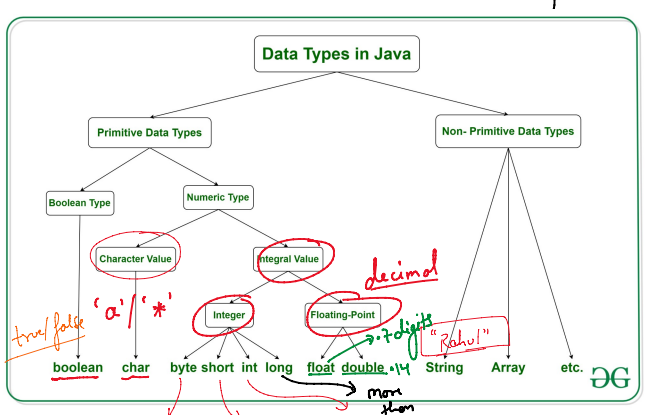
★ Creating a Variable ★

Declaring a Variable : type <Var_name>

```
int num;
```

★ Putting Data into a Variable ★ (Initializing a Variable)

```
num = 7;
```



byte id

1

2

:

10

```
float num = 7.2f;
double num = 7.257;
long num = 25---L;
String name = "Rahul";
```

Capital

```
char ch = '/';
char ch = ' '; (Space is also a character)
```

LHS RHS

```
int num = 7;
String name = "Rahul";
```

★ Output Method ★

```
System.out.println("Number :" + num);
```

String name

```
System.out.print ("Number :" + num);
```

Number : 7 Name: Rahul

```
System.out.print ("Name :" + name);
```

★ Operators ★

(quotient) (Remainder)

① Arithmetic : +, -, /, %, *, ++, --

```
double a = 5;
double b = 2;
double quotient = 5/2;
double rem = 5 % 2
int num = -2;
```

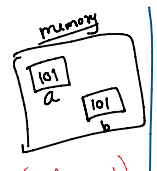
(2.5)

(1)

2 — quotient
2 | 5
- 4
1 — Rem

++ (Increment)

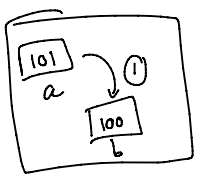
Pre-increment




```
a = 100;
b = ++a;
```

(Left most)

Post-increment




```
a = 100;
b = a++;
```

output : a → 101, b → 101

output : a → 101, b → 100

② Conditional : <, >, >=, <=, !=, == (Exactly Equal to?)

③ Assignment : =, +=, -=, *=, /=, %=

a (+)= b

a = a (+) b

④ Logical : AND, NOT, OR

&&, !, ||

a ✓ T, b ✓ T → T

```
double marks = 75;
if ( marks > 55 && marks < 80)
  {
    Sout(" Grade B");
  }
```

⑤ Bitwise : &, |, <<, >>

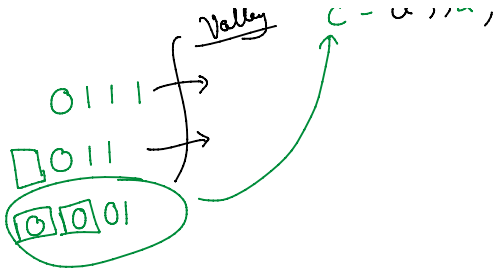
```
int a = 7;
int b = 3;
int c = a & b;
```

0 1 1 1
& 0 0 1 1
0 0 1 1 → Decimal → 3

①

```
c = a >> 2;
```

0 1 1 1 → Valley

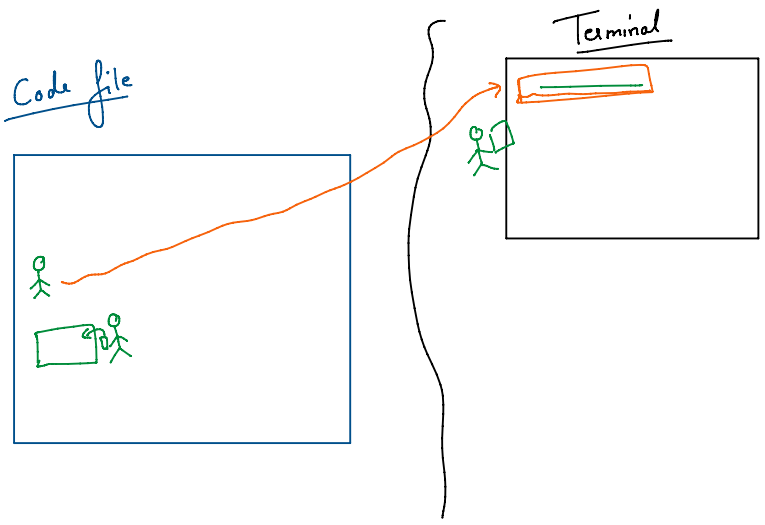


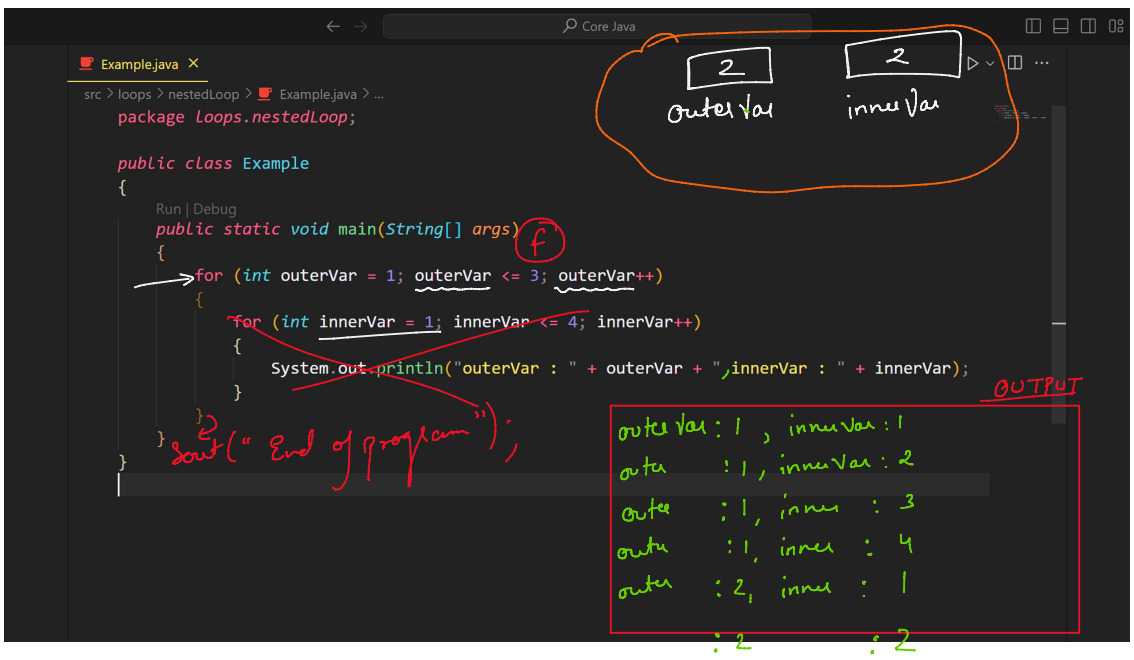
File Name - Class Name

Class Name - Pascal Convention

Variable/function/object Names - CamelCase

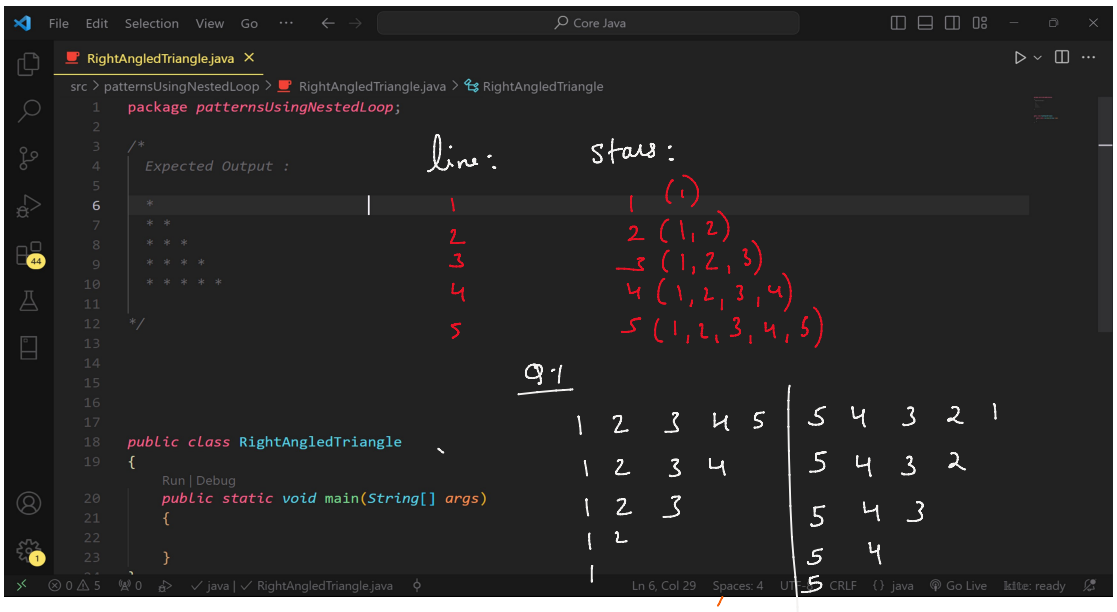
Every folder Containing Java files is called 'A Package'

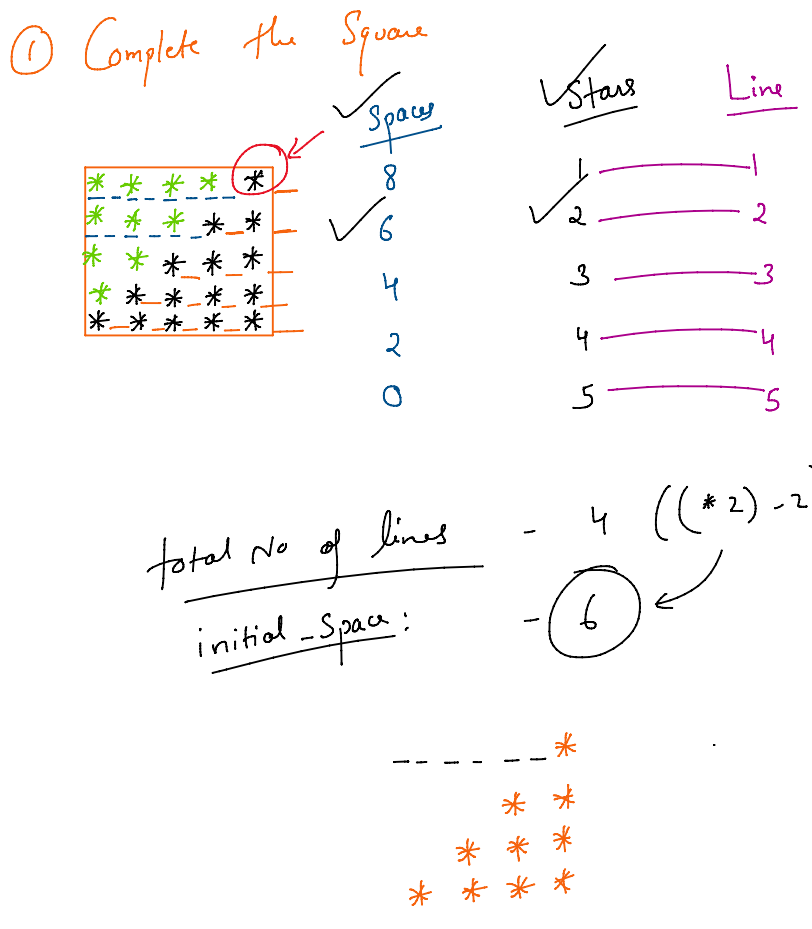




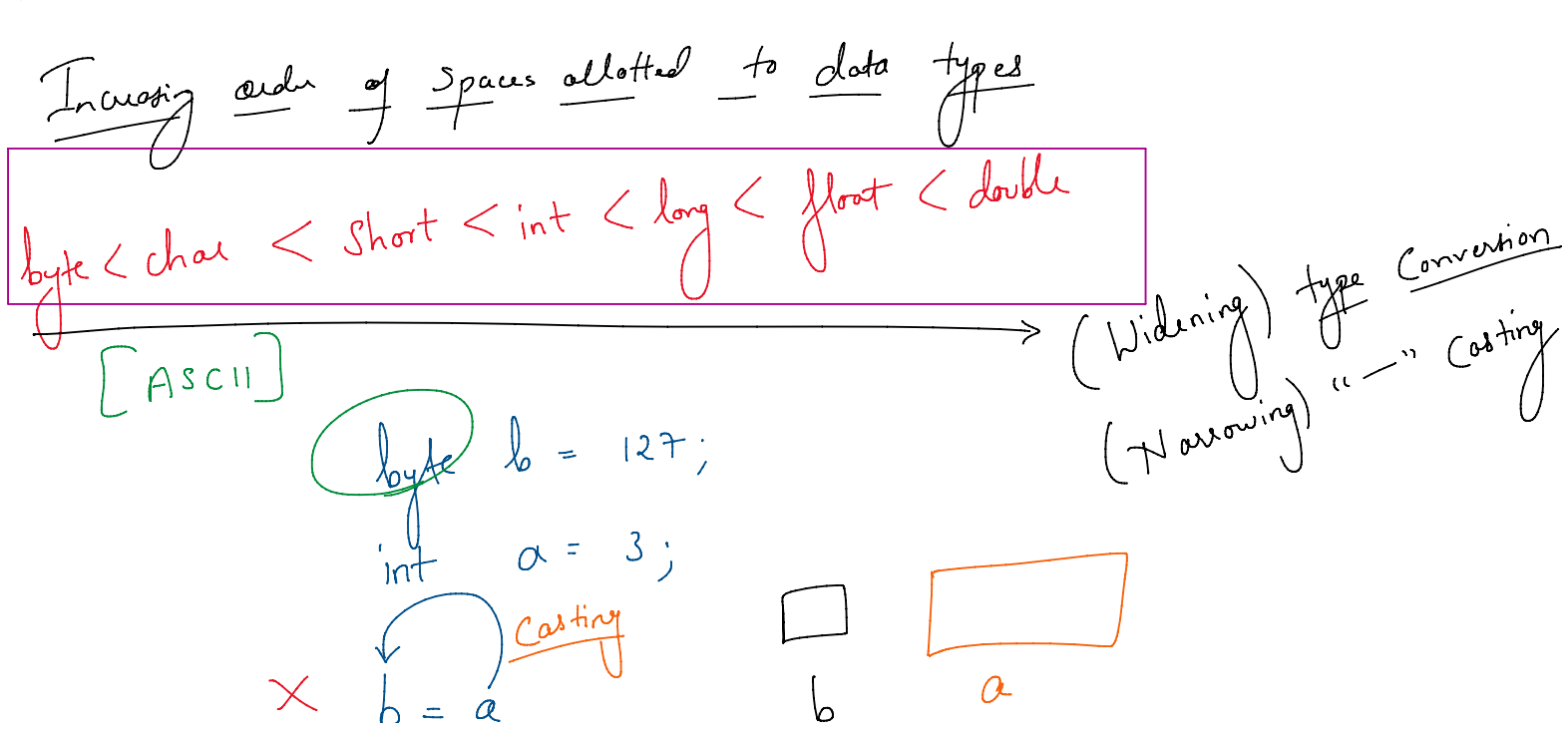
: 2 : 2
2 2
2 3
3 1
3 2
3 3
3 4

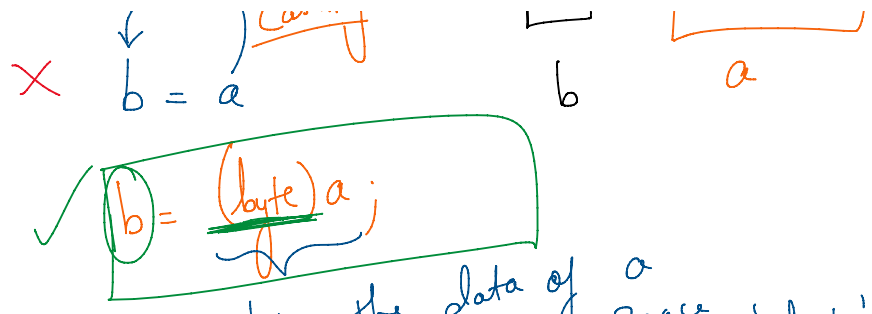
End of program


File Edit Selection View Go
Core Java
RightAngledTriangle.java
src > patternsUsingNestedLoop > RightAngledTriangle.java > RightAngledTriangle
line: stars:
1 (1)
2 (1,2)
3 (1,2,3)
4 (1,2,3,4)
5 (1,2,3,4,5)
Q.1
1 2 3 4 5 | 5 4 3 2 1
1 2 3 4 | 5 4 3 2
1 2 3 | 5 4 3
1 2 | 5 4
1 | 5
public class RightAngledTriangle
{
Run | Debug
public static void main(String[] args)
{
}
Ln 6, Col 29 Spaces: 4 UTF CRLF java Go Live


① Complete the Square
Space
8
6
4
2
0
Stars
Line
1 — 1
2 — 2
3 — 3
4 — 4
5 — 5
total No of lines - 4 ((*2)-2)
initial_Space: - 6


RightAngledTriangle.java
InvertedRightAngledTriangle.java
src > patternsUsingNestedLoop > InvertedRightAngledTriangle.java > InvertedRightAngledTriangle > main(String[])
Run | Debug
public static void main(String[] args)
{
int totalLines = 5;
int initialSpace = (totalLines * 2) - 2;
for (int line = 1; line <= totalLines; line++)
{
for (int spaces = initialSpace; spaces > 0; spaces--)
{
System.out.print(" ");
}
for (int star = 1; star <= line; star++)
{
System.out.print("* ");
}
initialSpace = initialSpace - 2;
System.out.print("\n");
}
}
i Space
0
line
Space
star
6
0
4
End of program
Bye Vishaka


Increasing order of Spaces allotted to data types
byte < char < Short < int < long < float < double
[ASCII]
(Widening) type Conversion
(Narrowing) "—" Casting
byte b = 127;
int a = 3;
Casting
X b = a
b
a

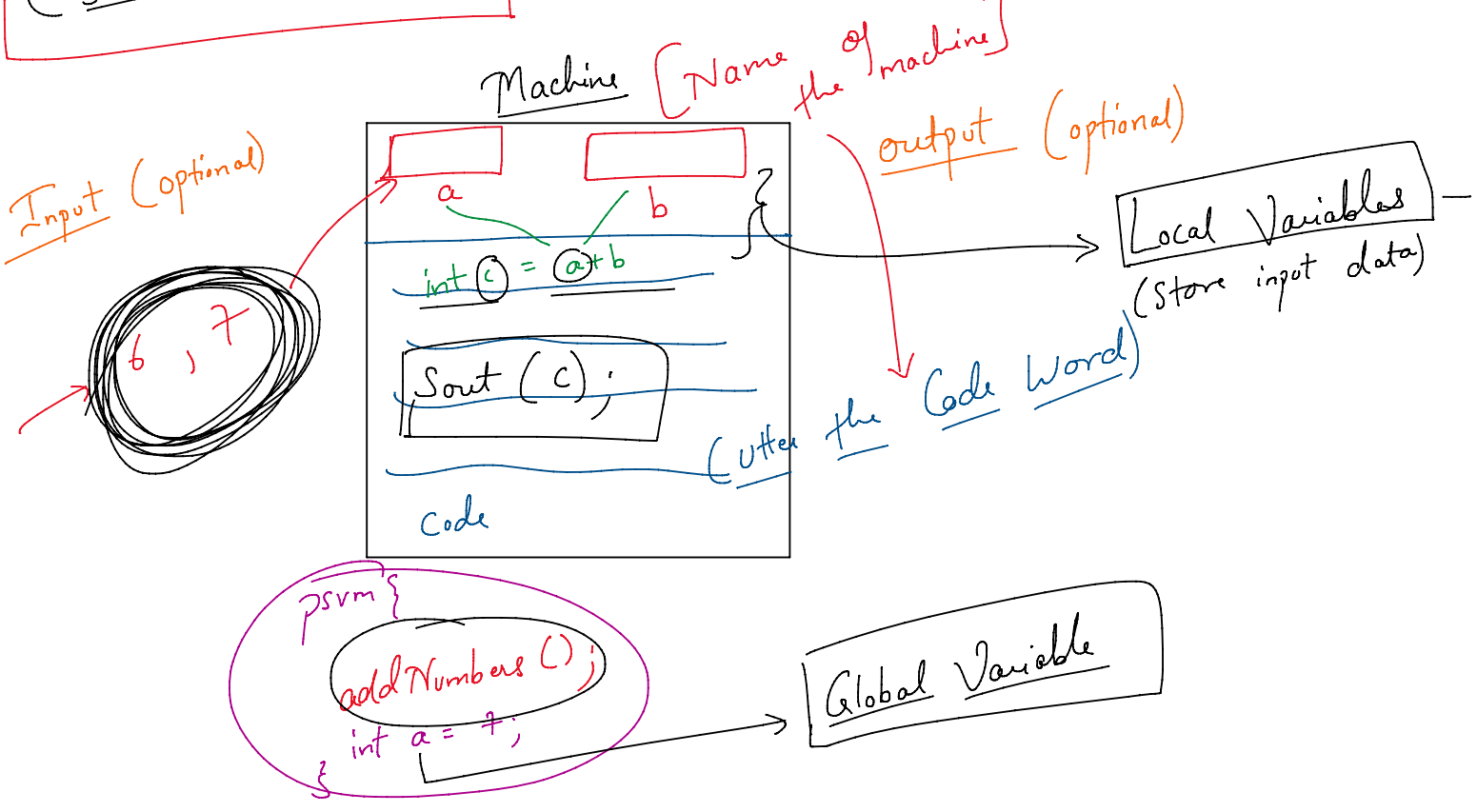



```
int a = 7;
float f;
f = a;
```

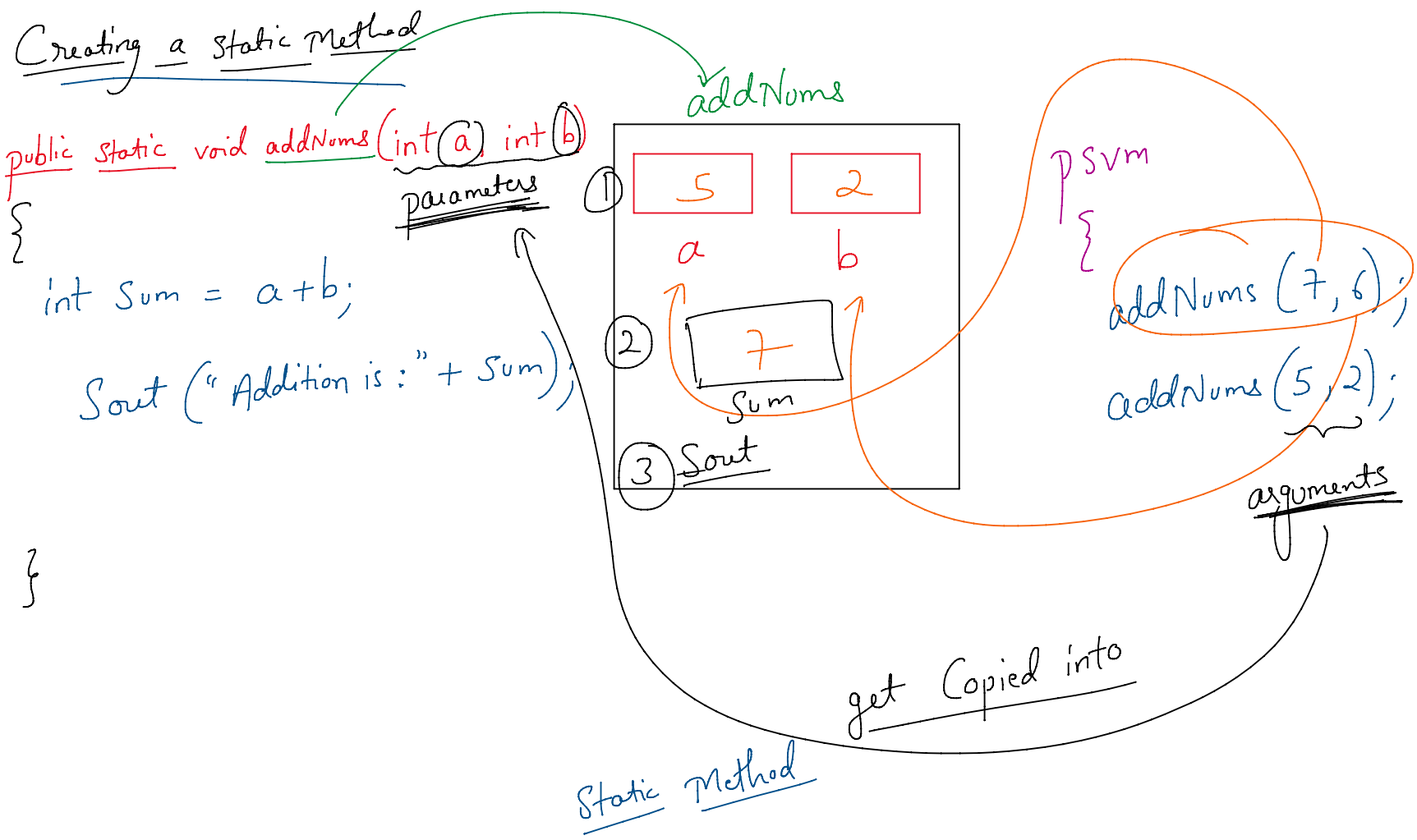
Type Conversion

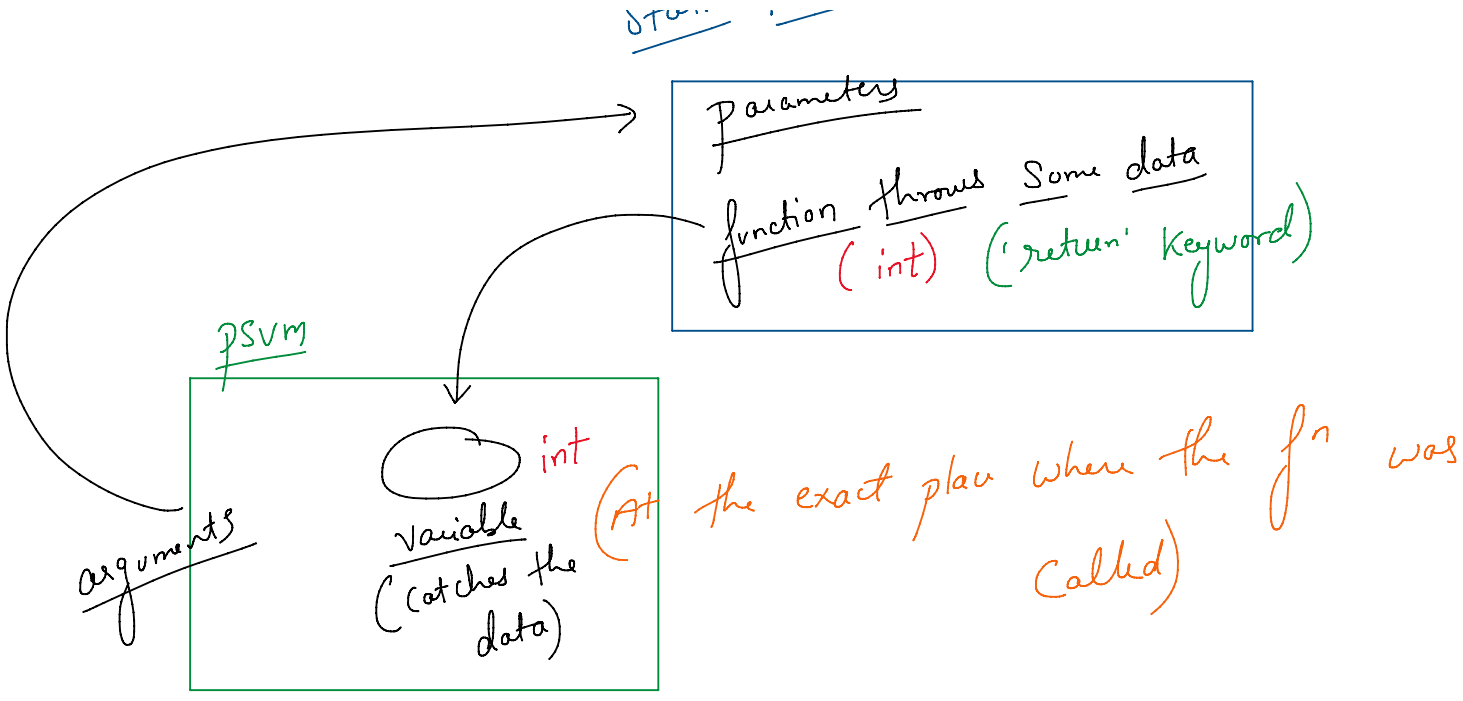
☆ Functions ☆

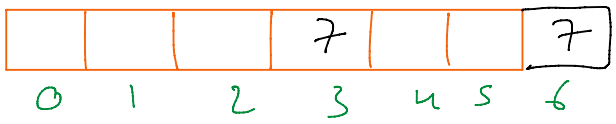
(Static Methods)



Creating a Static Method



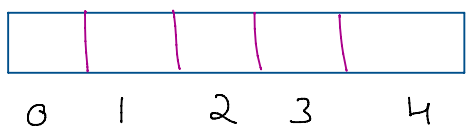


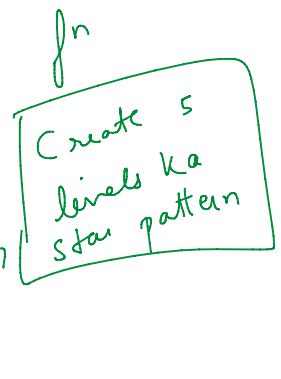


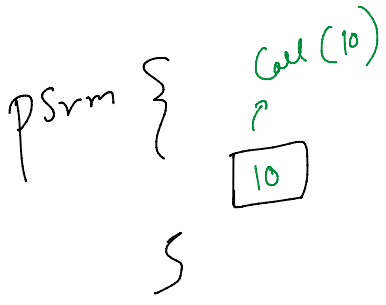
array.push (7)

.push (3 , 7) ;

★ Array ★ (Same data-type)

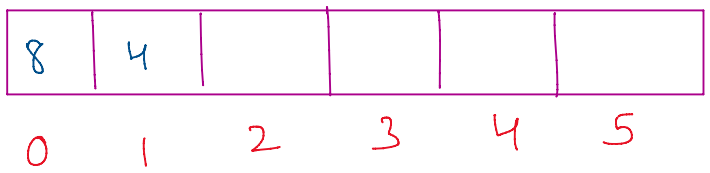


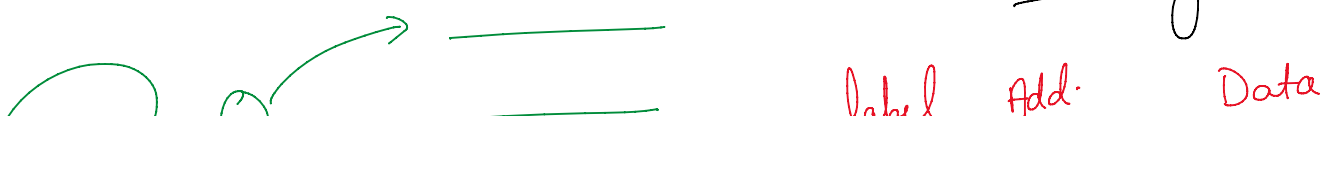




★ Arrays ★

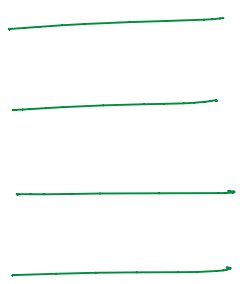
(1) Elements are stored in Consecutive memory location
(2) Each element Can be Considered as a Variable.





int a = 5;

int b = 7;



label	Add.	Data
a	3001H	0000 0000 (1 byte)
	3002	" — "
	3003	" — "
	3004	0000 0101
	3005	0 0 0 7

1 bit = Smallest unit

8 bits = 2 Nibbles = 1 byte

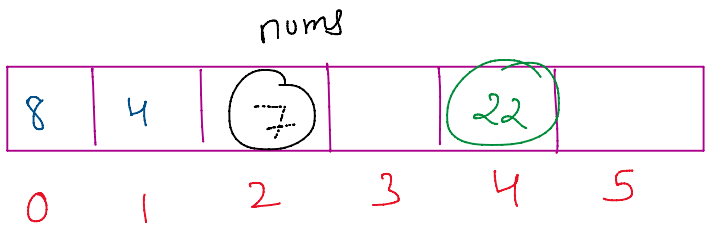
1024 bytes = 1 Kb

1024 Kb = 1 mb

1024 mb = 1 Gb

1024 Gb = 1 Tb

1024 Tb = 1 Pb



name of array + indices } refer to elements

nums[2] = 7;

nums[4] = 22;

[Arrays have fixed size]

★ Creating an Empty Array ★

No. of elements that array shall hold.

int [ ] nums = new int [5];

int [ ] nums

int nums [ ] =

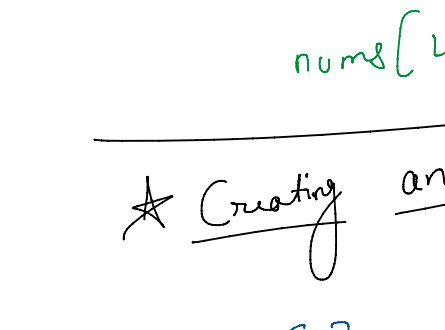
int nums [ ] =

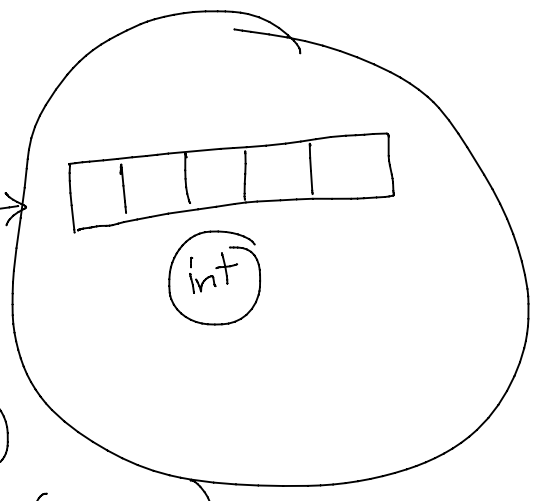
nums [0] = ——;

nums [1] = ——; 1 → 5 (n)

nums [2] = ——; i → 0 to 4 (0, n-1)

name of 1st element






```
Sc                                                    5
for( int index = 0; index < nums.length; index ++)
{   Sout (" Element Ki Value enter karo:");
    nums [index] = Sc.nextInt();
}
```

background → (nums[4] = 22)

: 22

for (

)

5

n)

```
{
    Sout (nums[index]);
}
```

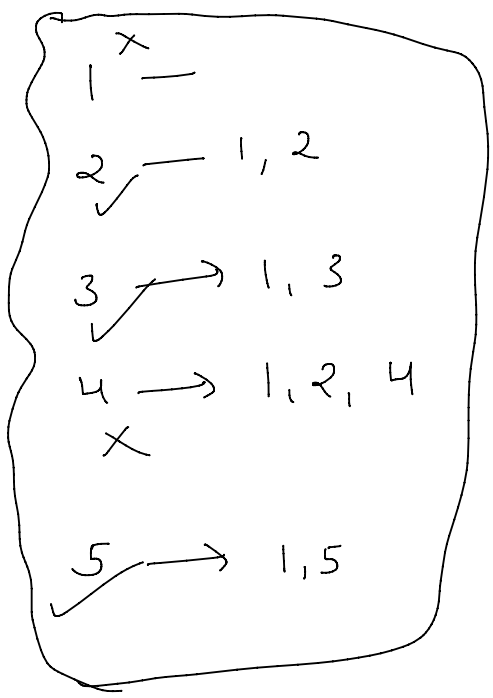
Declaration + Initialization of an Array :

```
int nums[] = {5, 7, 2, 3, 8, 1, 3};
```
(length 7)

PRIME NUMBER PROGRAM USING ARRAYS :

[Divisible only by 1 or themselves]

5	21	7	10	2



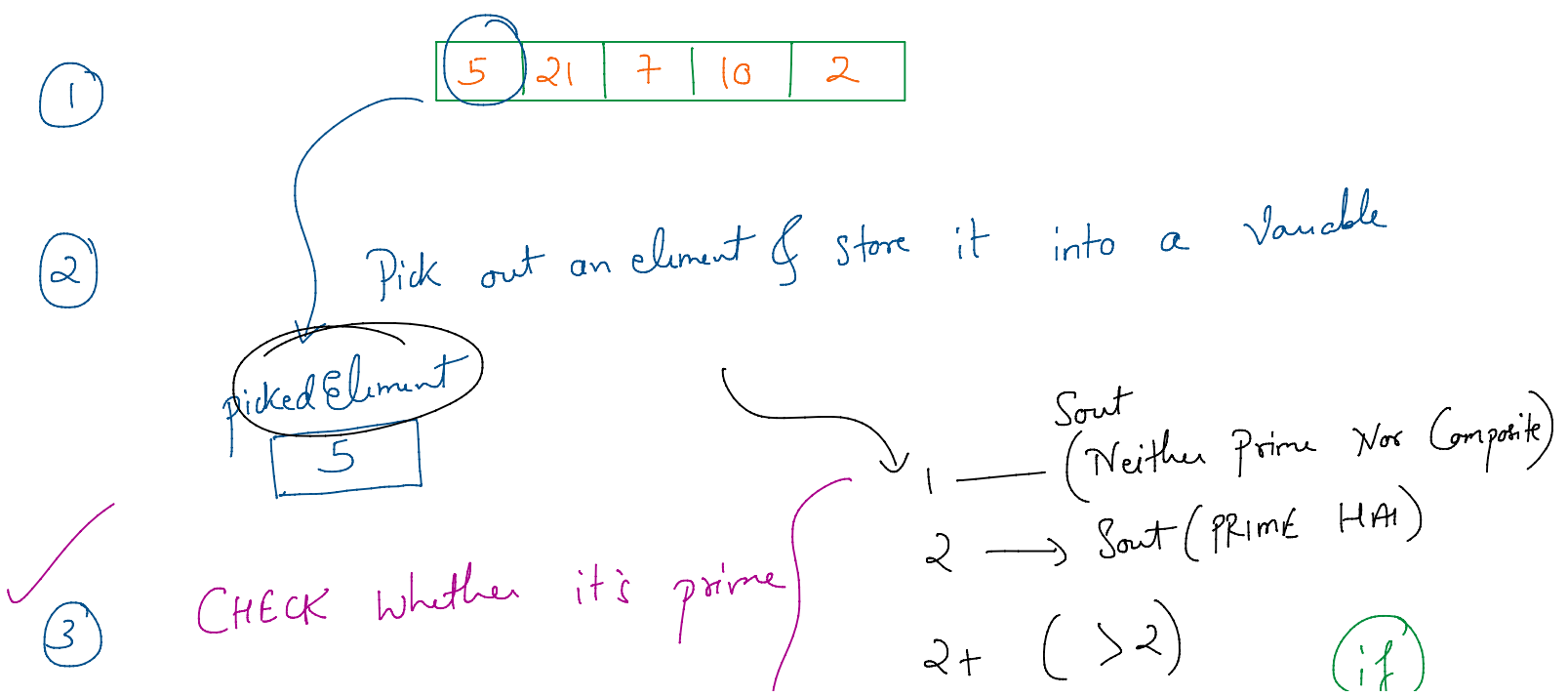
OR

[Not Divisible by any Number rather than 1 or itself]

i.e ||

[If a Number (n) gets divided by any number in the range 2 — (n-1), then it's NOT A PRIME NUMBER]

2 - 10



3 CHECK whether it's P

4 Sout(Fact);

2 — (n-1)

2+ (>2)

{

Checker

Rem Division

4

5

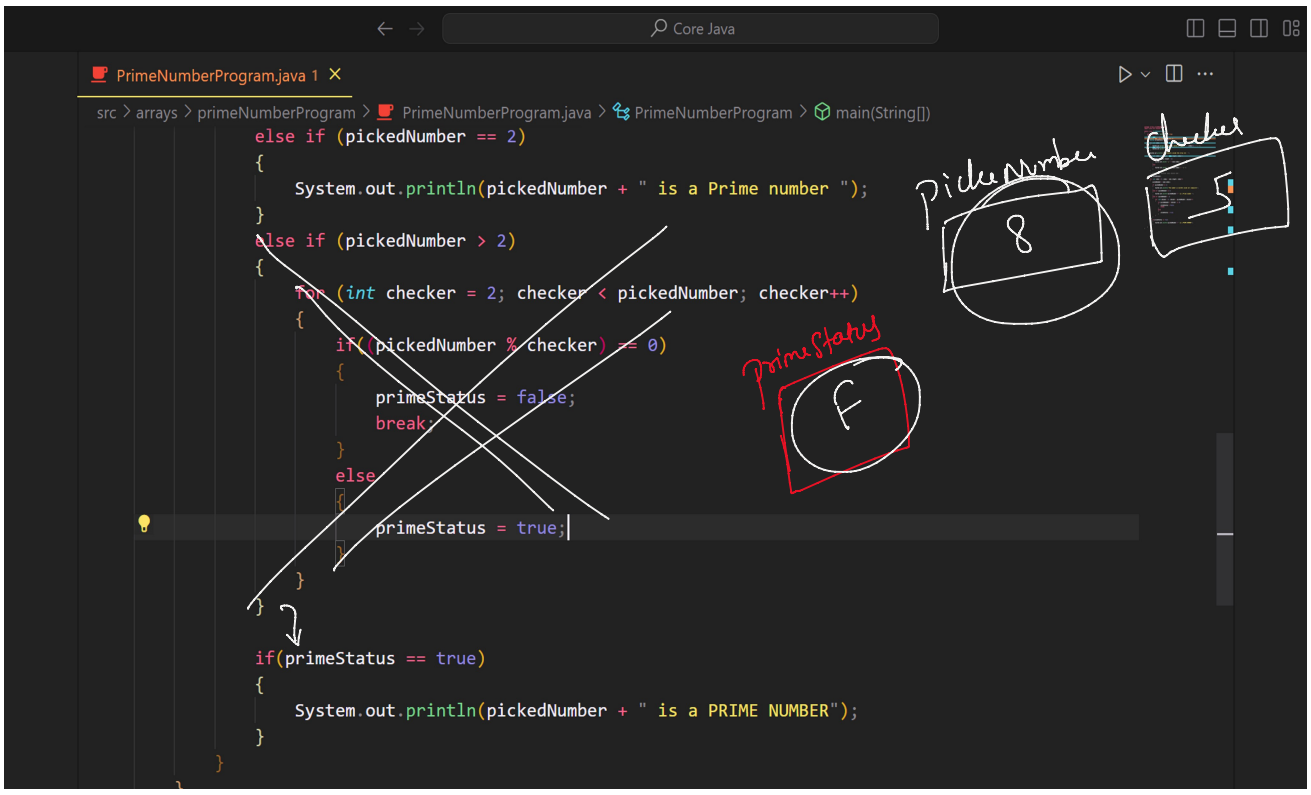
Checker ++

if

Rem 0:

NON PRIME

Stop execution of the loop



1 for (1 — 100

pickedNumber

Checking

The prime nos in the range of 1 to 100 are

7, , ____

Else

2 ____ PRIME

3 ____ NOT A

2 (min 5 elements) [ ]

2	5	7	8	6

(Sum of all the Elements of Array)

28

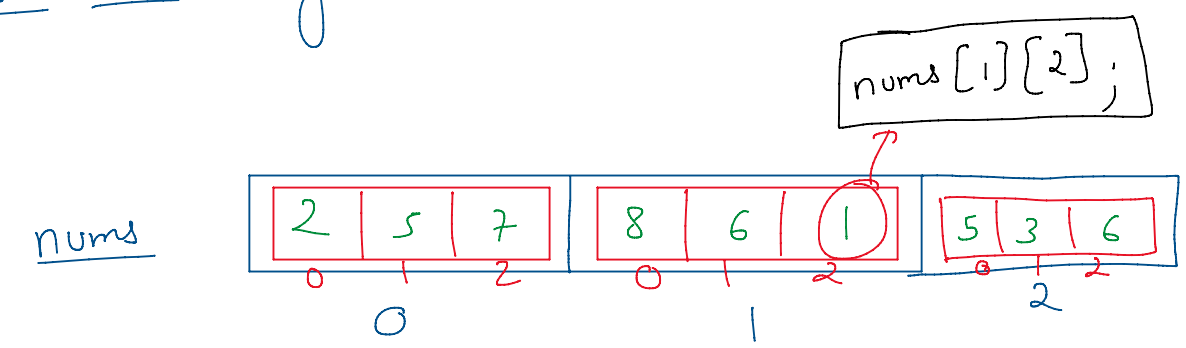
Sum

★ 2D Array

Array inside another Array.

∪ Array inside another Array.

3	5	6
7	8	1
10	2	9

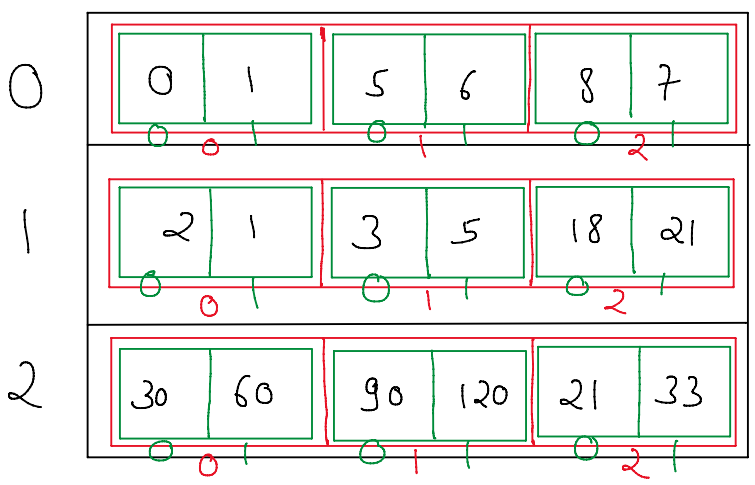



```
int nums[][] = new int[3][3];
```

[0][0]
[0][1]
[0][2]
[1][0]
[1][1]
[1][2]
[2][0]
[2][1]
[2][2]

★ 3D Array ★

Array inside an Array inside an Array

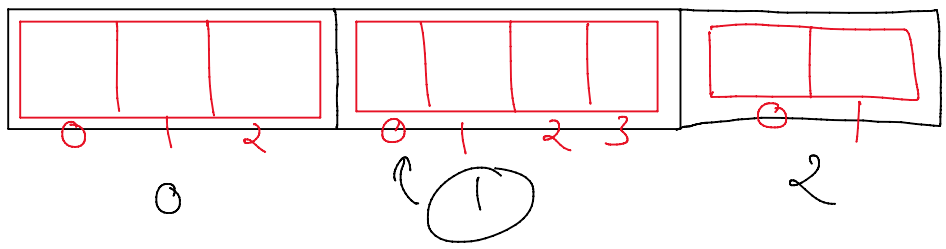



```
for
{
    for
    {
        for
        {
        }
    }
}
```

0 0 0
0 0 1
0 1 0
0 1 1
0 2 0
⋮

```
int nums[][][] = new int[3][3][2];
```

★ Jagged Array ★




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

☆ Arraylist ☆

(Variable sized Array)

Declaration : `Arraylist <Wrapper class> name = new Arraylist<>();`

Methods (Predefined):

1. add
2. get
3. remove
4. Size (length of Arraylist);
5. Set

☆ OOPS ☆ (Solve: Real Life Problems - Virtually)

- Plan (class)
- object

Every object
- Certain Attributes
- Functions (Methods)

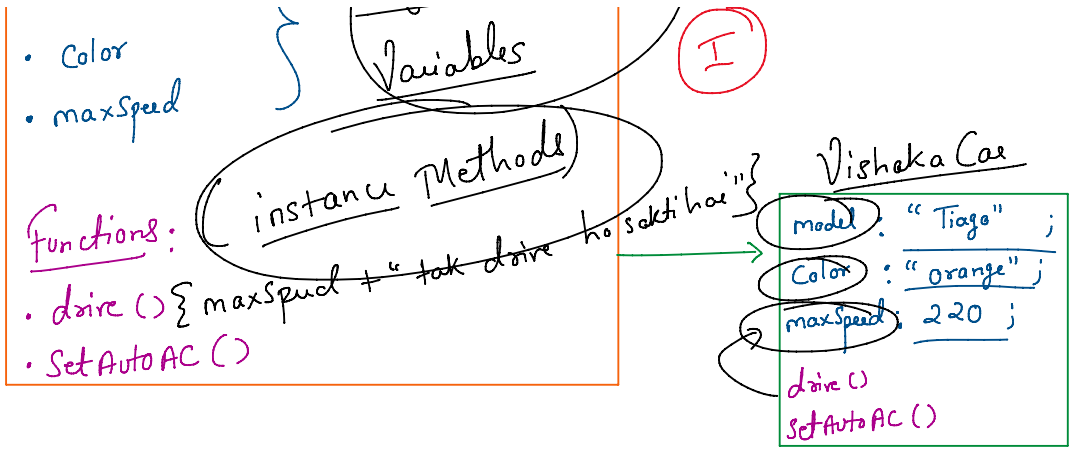
Plan (class) Cars

Attributes
- model
- Color

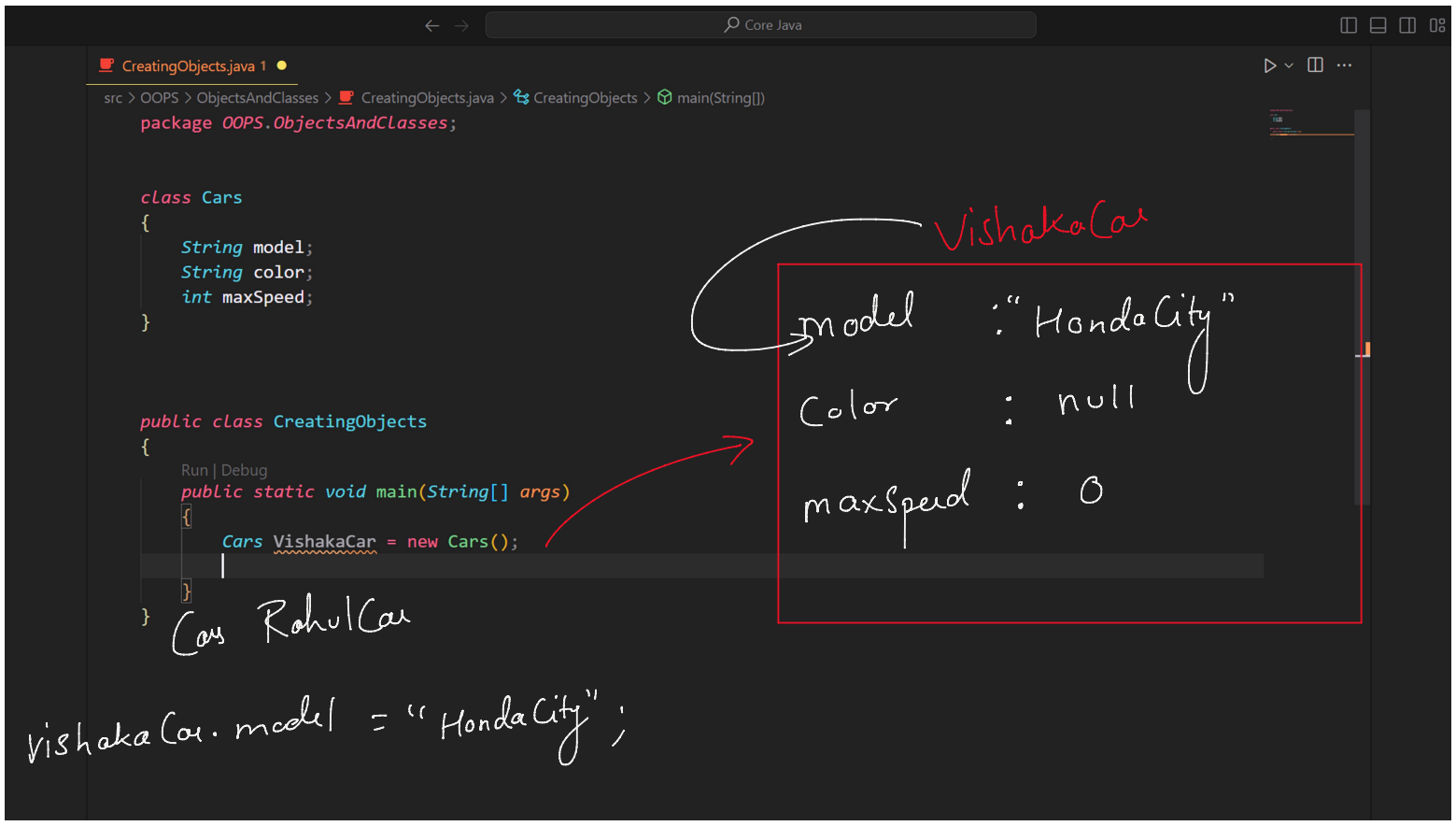
Object instance Variables

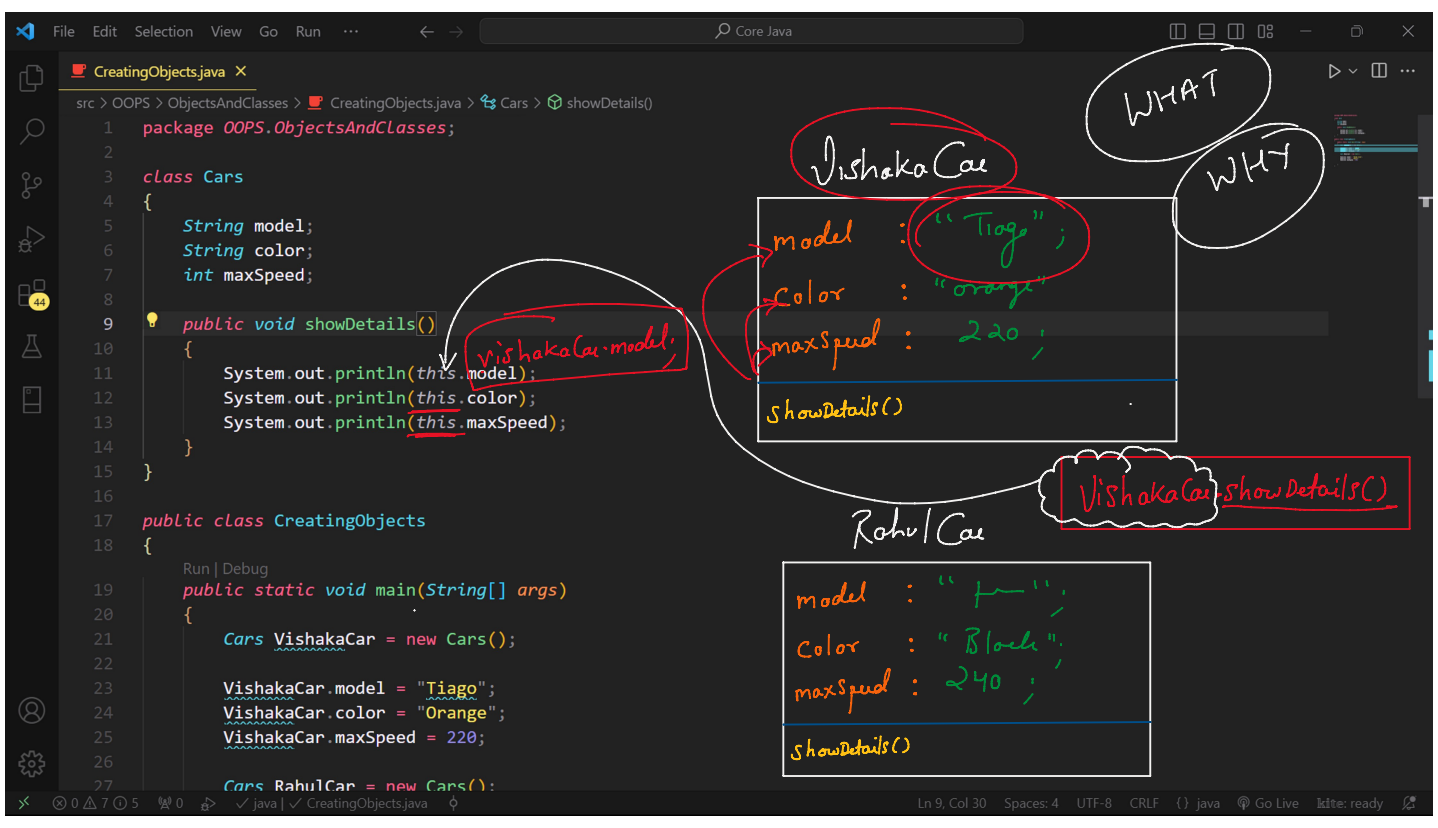
RahulCar
- model : "Honda City";
- Color : "black";
- maxSpeed : 240;
- drive()
- setAutoAC

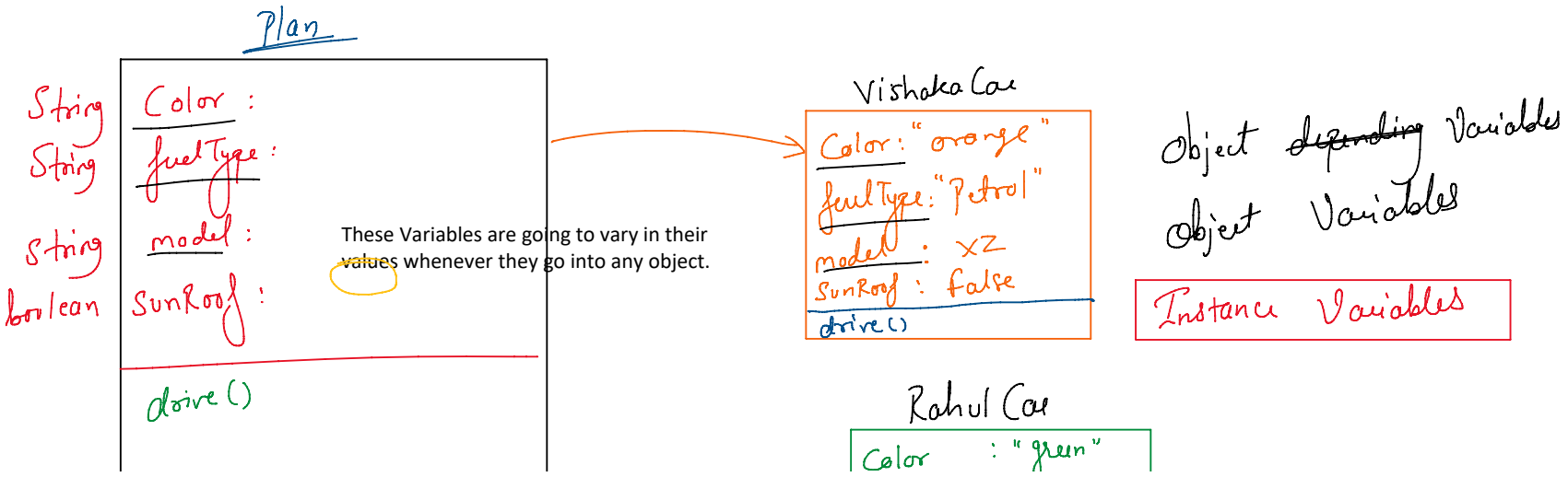
An object is also called "instance".

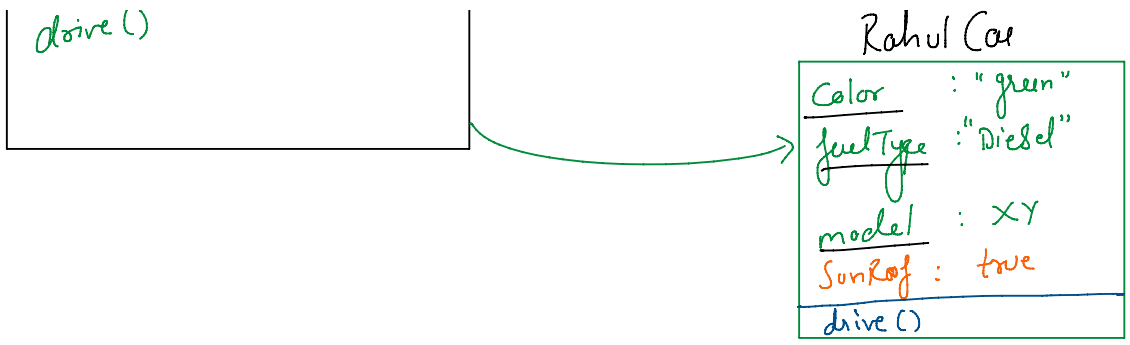

Color
maxSpeed
Variables
I
instance Methods
Functions:
drive () { maxSpeed + " tak drive ho sakti hai" }
SetAutoAC ()
Vishaka Car
model : "Tiago" ;
Color : "orange" ;
maxSpeed : 220 ;
drive ()
SetAutoAC ()
" instance ".

Static


Core Java
CreatingObjects.java 1
src > OOPS > ObjectsAndClasses > CreatingObjects.java > CreatingObjects > main(String[])
package OOPS.ObjectsAndClasses;
class Cars
{
String model;
String color;
int maxSpeed;
}
public class CreatingObjects
{
Run | Debug
public static void main(String[] args)
{
Cars VishakaCar = new Cars();
}
}
Cars RahulCar
VishakaCar
model : "HondaCity"
Color : null
maxSpeed : 0
VishakaCar.model = "HondaCity";

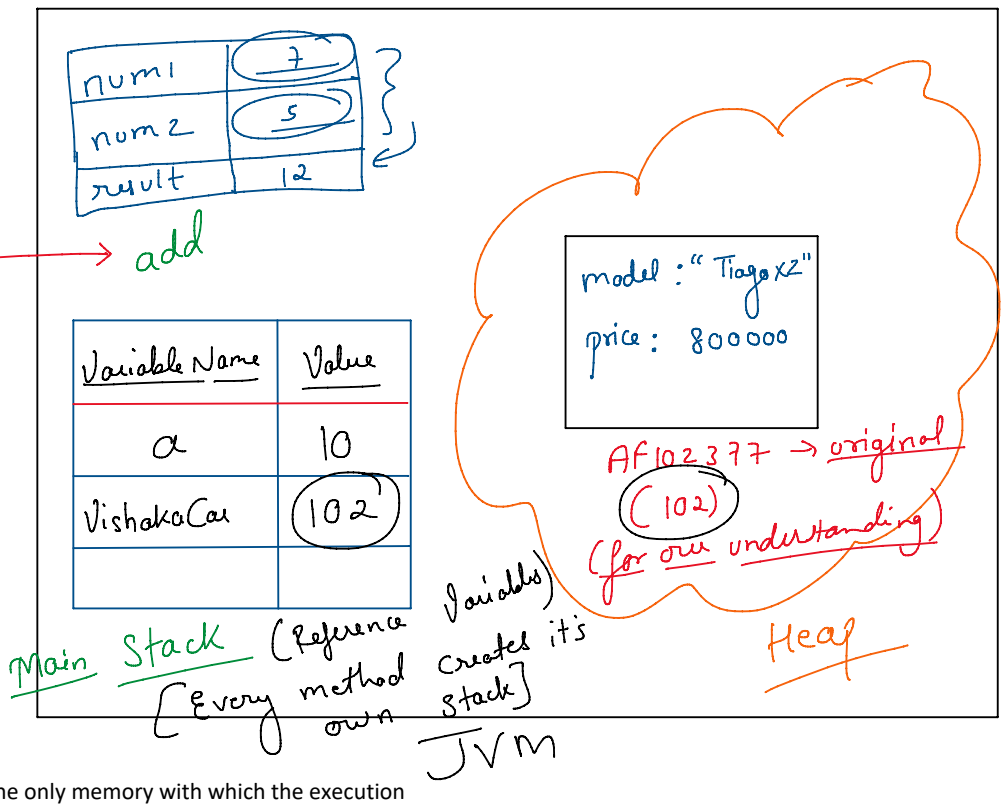

File Edit Selection View Go Run
Core Java
CreatingObjects.java
src > OOPS > ObjectsAndClasses > CreatingObjects.java > Cars > showDetails()
package OOPS.ObjectsAndClasses;
class Cars
{
String model;
String color;
int maxSpeed;
public void showDetails()
{
System.out.println(this.model);
System.out.println(this.color);
System.out.println(this.maxSpeed);
}
}
public class CreatingObjects
{
Run | Debug
public static void main(String[] args)
{
Cars VishakaCar = new Cars();
VishakaCar.model = "Tiago";
VishakaCar.color = "Orange";
VishakaCar.maxSpeed = 220;
Cars RahulCar = new Cars();
WHAT
WHY
VishakaCar
model : "Tiago" ;
Color : "orange"
maxSpeed : 220 ;
showDetails()
VishakaCar.model
VishakaCar.showDetails()
RahulCar
model : "Ford" ;
Color : "Black" ;
maxSpeed : 240 ;
showDetails()


Plan
String Color :
String fuelType :
String model :
boolean SunRoof :
drive ()
These Variables are going to vary in their values whenever they go into any object.
VishakaCar
Color : "orange"
fuelType : "Petrol"
model : XZ
SunRoof : false
drive ()
Object depending Variables
object Variables
Instance Variables
RahulCar
Color : "green"



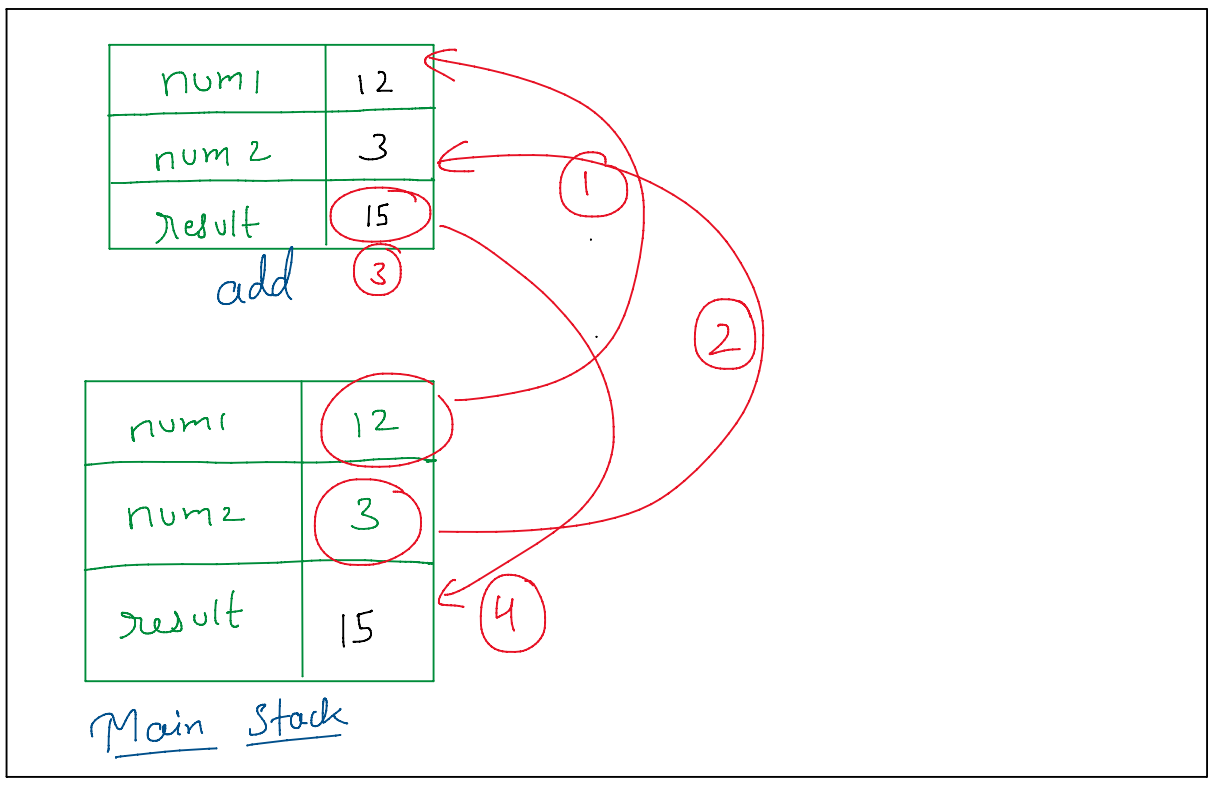
☆ Heap & Stack ☆

```java
Example.java 1 ×
src > OOPS > stackAndheap > Example.java > Example > main(String[])
1   package OOPS.stackAndheap;
2
3   class Cars
4   {
5       String model;
6       float price;
7   }
8   public void add(int num1, int num2)
9   public class Example
10  {
11      Run | Debug
12      public static void main(String[] args)
13      {
14          int a = 10;
15
16          Cars VishakaCar = new Cars();
17          VishakaCar.model = "Tiago XZ";
18          VishakaCar.price = 8_00_000f;
19          System.out.println(VishakaCar.price);
20      }
21  }
```



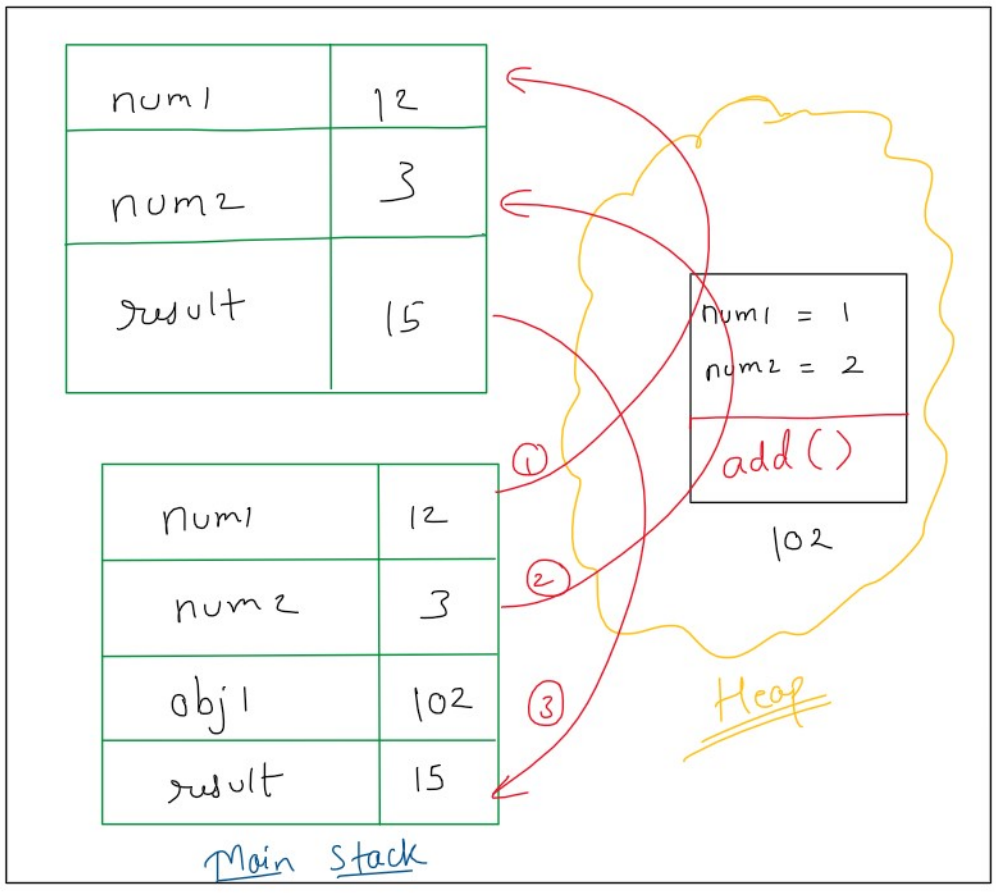
Stack memory is the only memory with which the execution engine can interact. That means our execution engine cannot interact with heap. So stack is a way which creates a link between heap and itself.

```java
TwentyDecember.java ×
src > OOPS > stackAndheap > TwentyDecember.java > TwentyDecember > main(String[])
package OOPS.stackAndheap;

public class TwentyDecember
{
    public static int add(int num1, int num2)
    {
        int result = num1 + num2;
        return result;
    }

    Run | Debug
    public static void main(String[] args)
    {
        int num1 = 12;
        int num2 = 3;
        int result = add(num1, num2);
        System.out.println(result);
    }
}
```

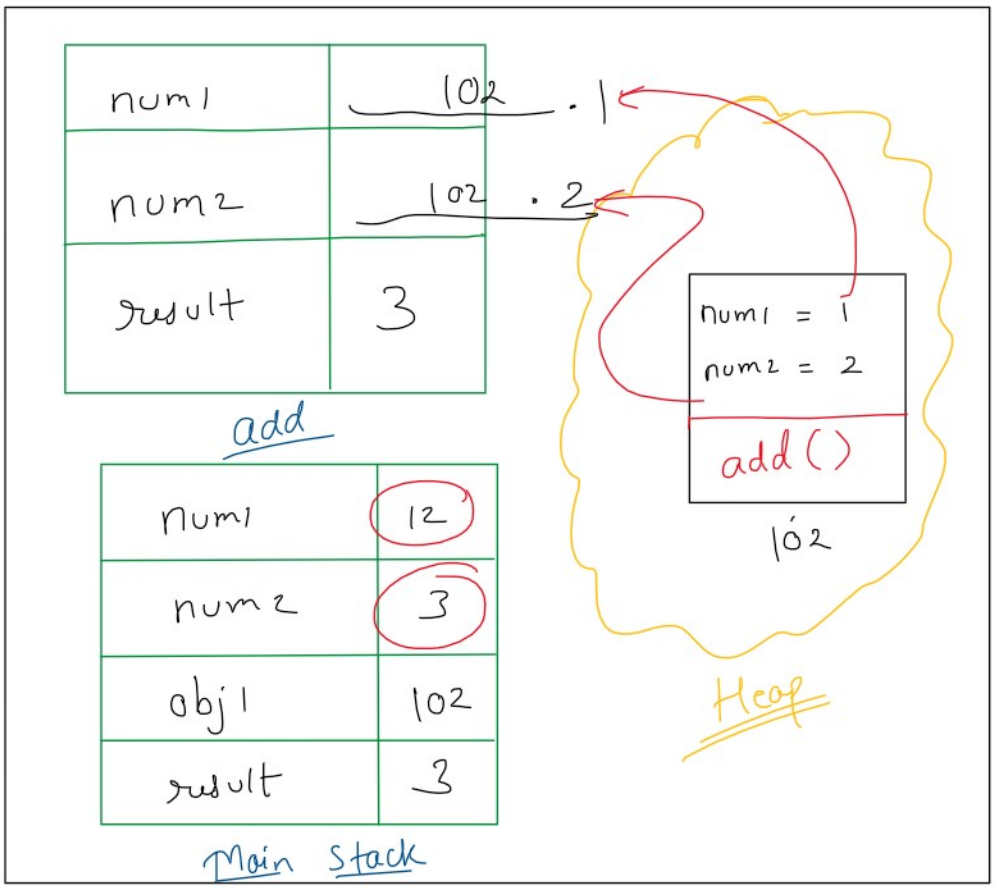


Each stack can have variables of unique names only

```java
package OOPS.stackAndheap;

class ArithmeticCalc
{
    int num1;
    int num2;

    public ArithmeticCalc(int NUM1, int NUM2)
    {
        this.num1 = NUM1;
        this.num2 = NUM2;
    }

    public int add(int num1, int num2)
    {
        int result = num1 + num2;
        return result;
    }
}

public class TwentyDecember2
{
    Run | Debug
    public static void main(String[] args)
    {
        int num1 = 12;
        int num2 = 3;

        ArithmeticCalc obj1 = new ArithmeticCalc(1, 2);

        int result = obj1.add(num1, num2);
        System.out.println(result);
    }
}
```




```java
package OOPS.stackAndheap;

class ArithmeticCalc
{
    int num1;
    int num2;

    public ArithmeticCalc(int NUM1, int NUM2)
    {
        this.num1 = NUM1;
        this.num2 = NUM2;
    }

    public int add(int num1, int num2)
    {
        num1 = this.num1;
        num2 = this.num2;
        int result = num1 + num2;
        return result;
    }
}

public class TwentyDecember2
{
    Run | Debug
    public static void main(String[] args)
    {
        int num1 = 12;
        int num2 = 3;

        ArithmeticCalc obj1 = new ArithmeticCalc(1, 2);

        int result = obj1.add(num1, num2);
        System.out.println(result);
    }
}
```



★ Strings ★

1. Every String object has indices in it
   [Every letter has an index]
2. Strings are stored inside Heap memory in the String Constant Pool
3. "Strings in java are immutable"
   [Can't be modified / changed]

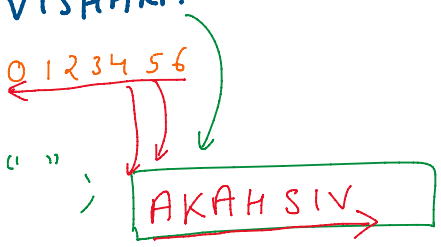
Str = "RAHUL"
0 1 2 3 4

Sout(str.charAt(3)); 'U'

revStr
[str.length() - 1]

Sout(str.charAt(3)); ∪

name = "VISHAKA"

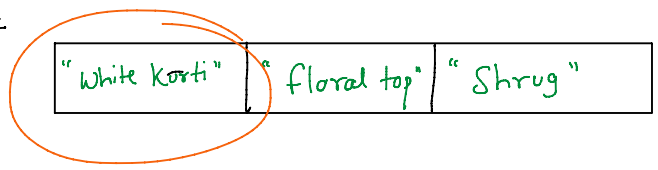


rev_str = " ";

```
for(int index = str.length()-1; index >= 0; index--)
{
    revStr = revStr + str.charAt(index)
}
```

AK

Q.1



"Skirt"	"Jeans"	"Leggings"

White Kurti : Skirt

____ : ____

____ : ____

floral top : ____

____

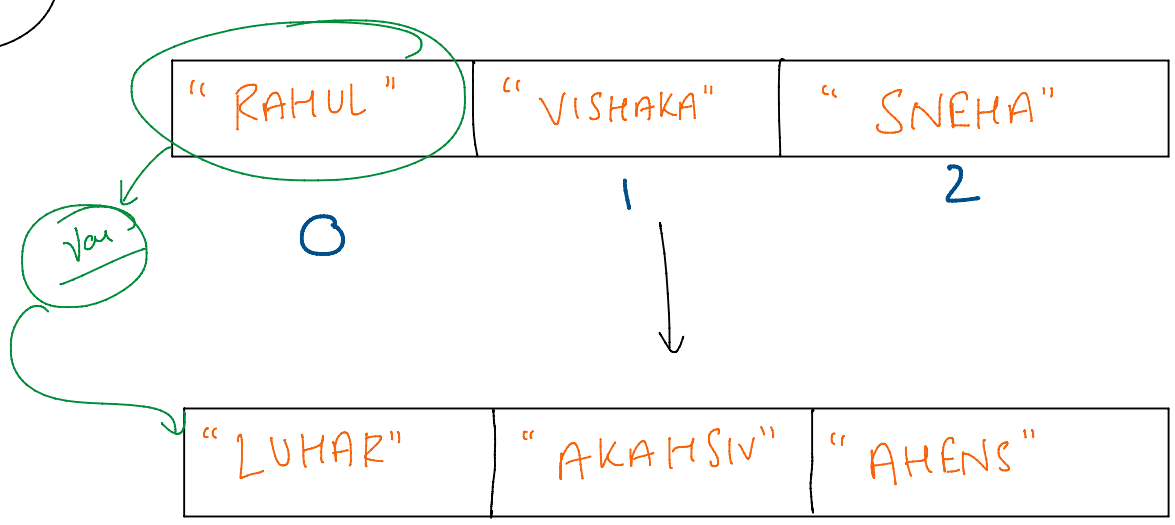
White Kurti : Skirt | White Kurti : Jeans ....

Sout all the Combined Elements which are inside arraylist using foreach Loop.

CombString = ____ + " : " + ____

outfits.add( );

Q-2

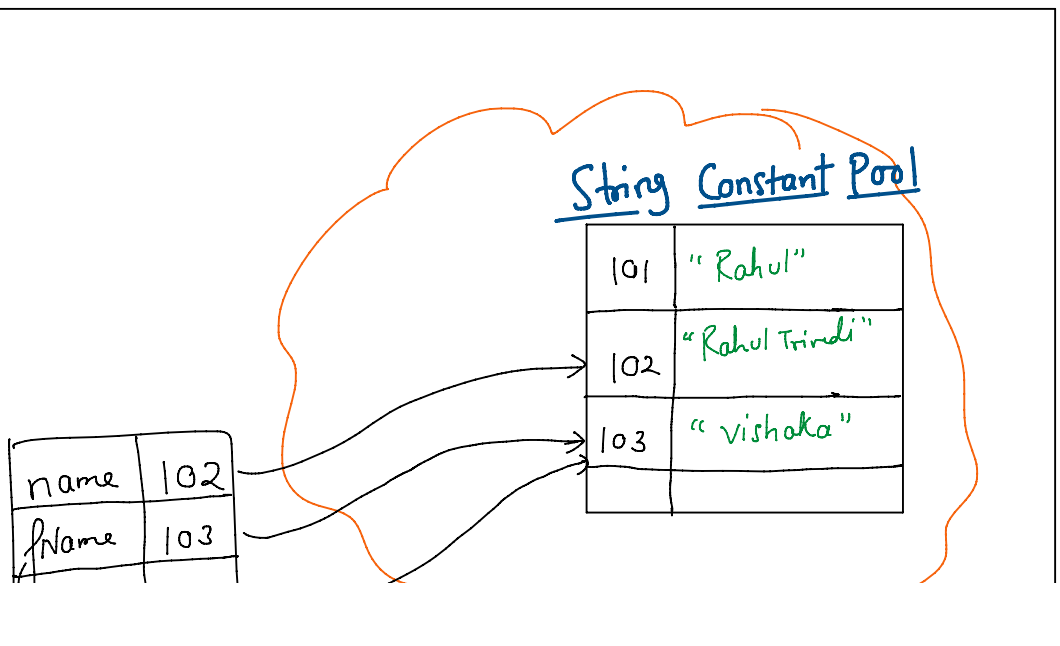


name = "Rahul"

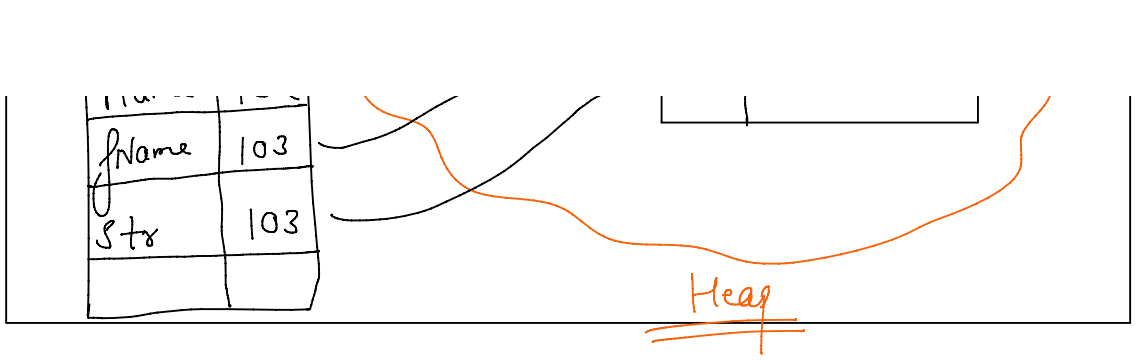
name = "Rahul" + "Trivedi";

fName = "vishaka";

str = "vishaka";



str = "vishaka";



Whenever We make any updations in the existing string object, Another object gets created with the updated value but the original object stays unaffected. Hence, it is said that strings in Java are immutable because the original object is not experiencing any mutation.

Str = "RAHUL"

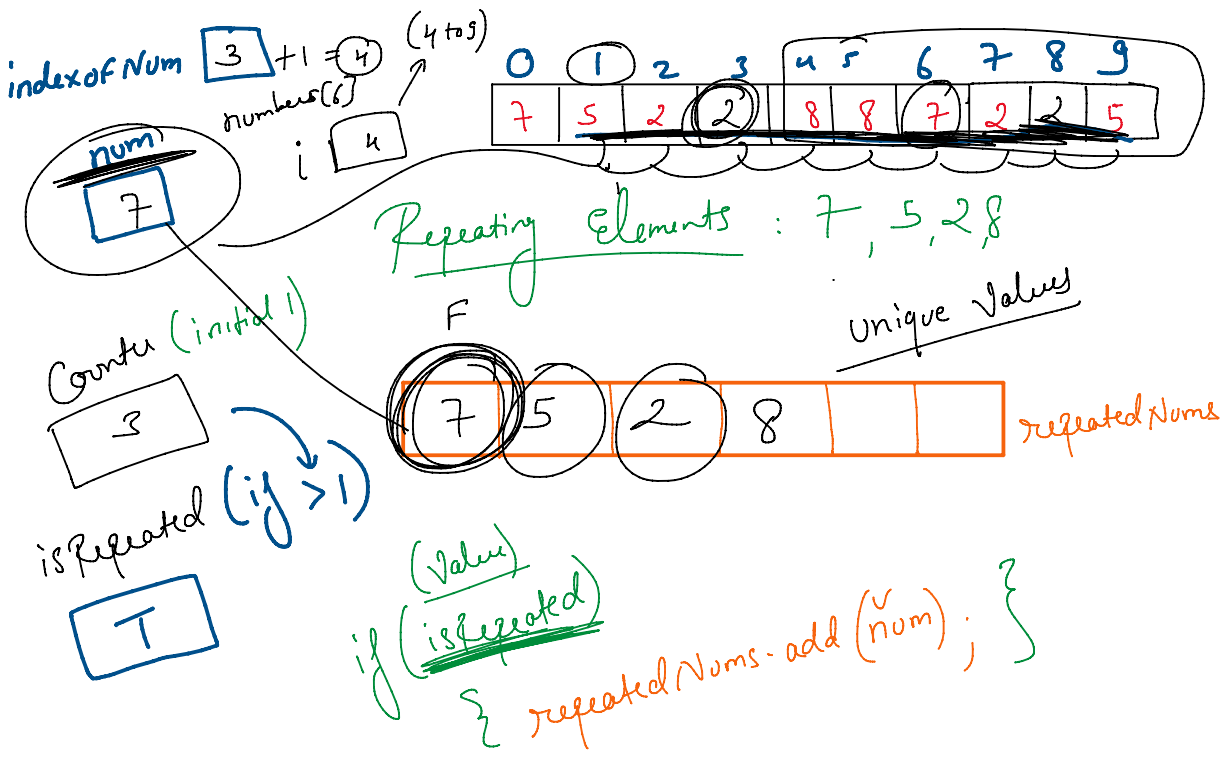
name = "RAHUL"

Sout(Str.equals(name)); (String class) (by-default method)

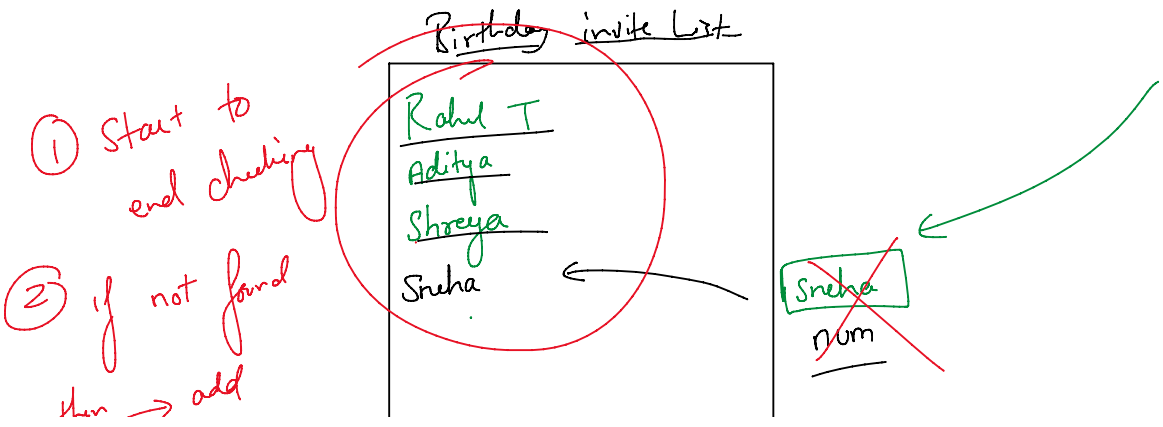
true

Sout(S

(7,8) →




```
if(num == numbers[i])
{
  Counter ++
}
```



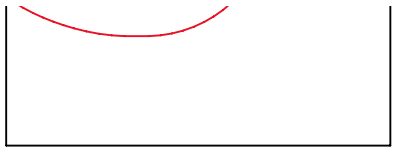
str == name)

↓

false

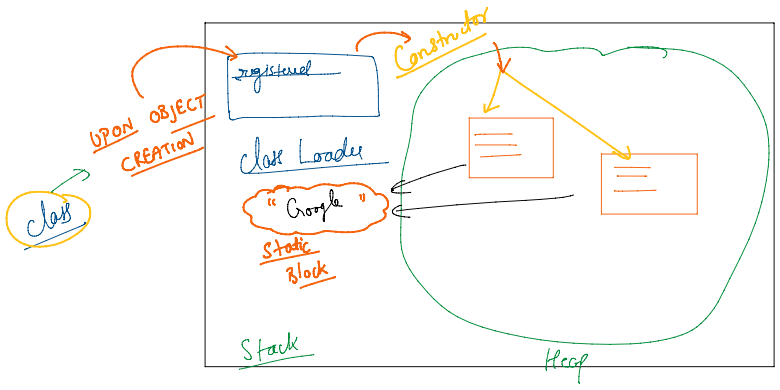
then → add

else: discard



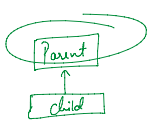
Arraylist

num

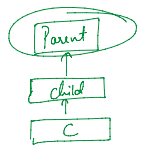


Types of Inheritance in Java :

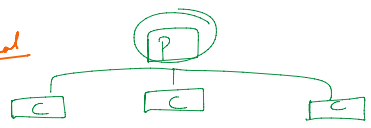
① Single



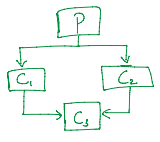
② Multilevel :



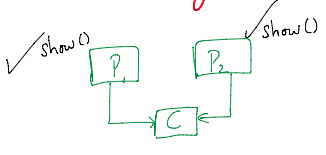
③ Hierarchical



④ Hybrid



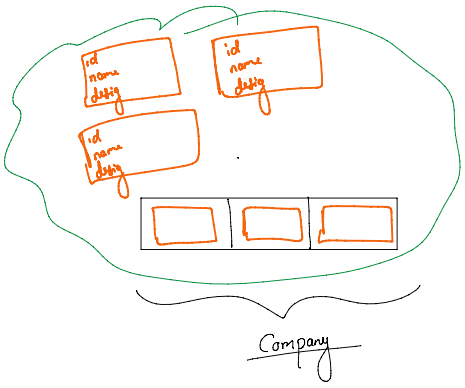
⑤ Multiple inheritance (Not Supported using classes)



interface fm
{
=== // features
}

[A newer version of Something Cannot be a successor of two things (though it may implement features/concepts of other things)]

class SmartPhone extends Phone implements fm, camera

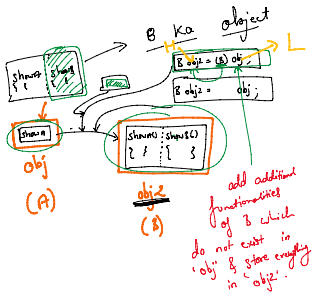


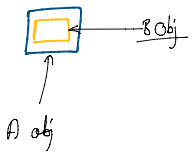
H ← L

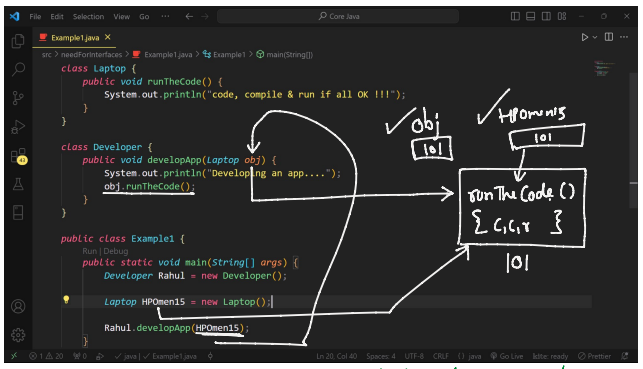
Downcasting

L ← H

Upcasting

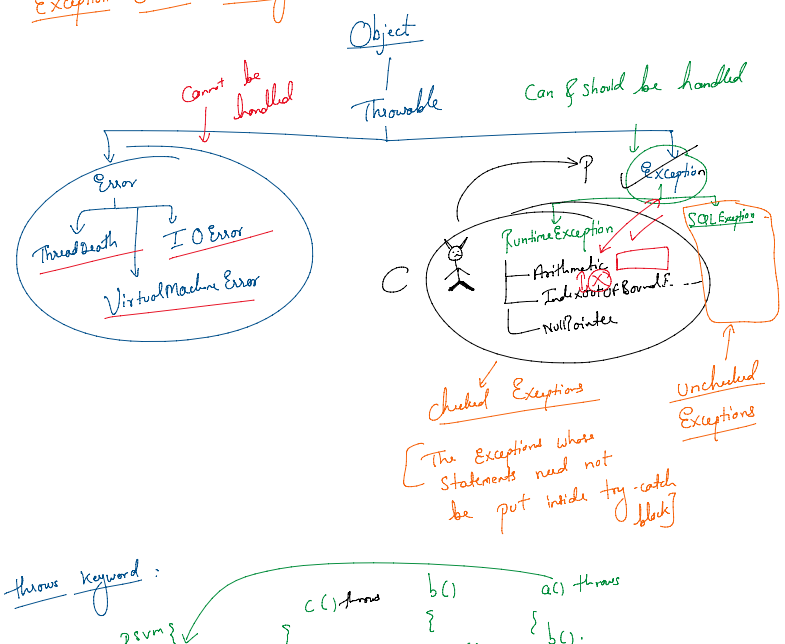




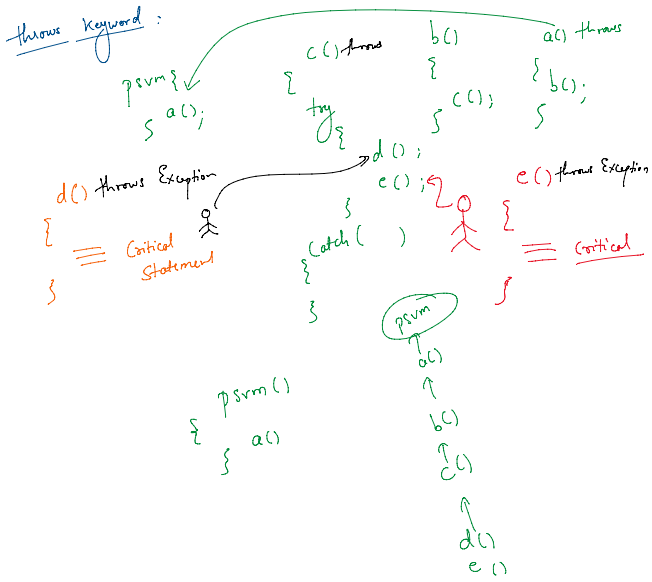


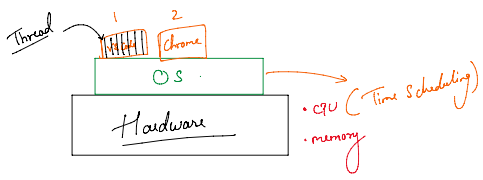
All the classes in java are child classes of the 'Object' class

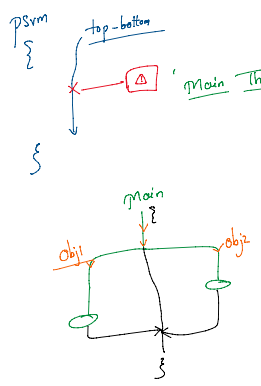
Exception classes Hierarchy



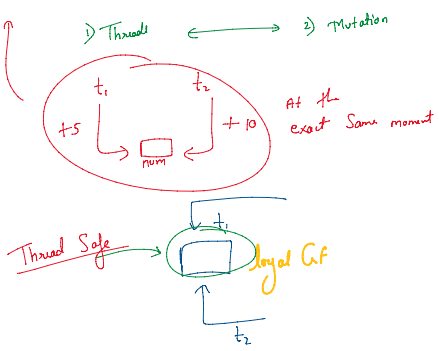
throws Keyword :

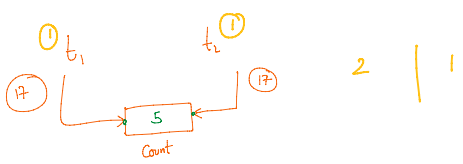


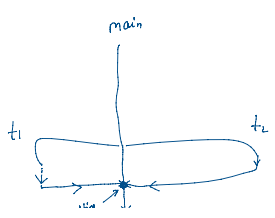


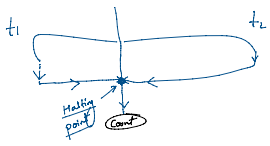


Race Condition :

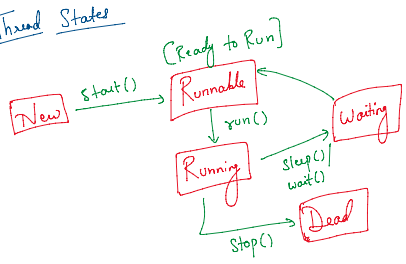


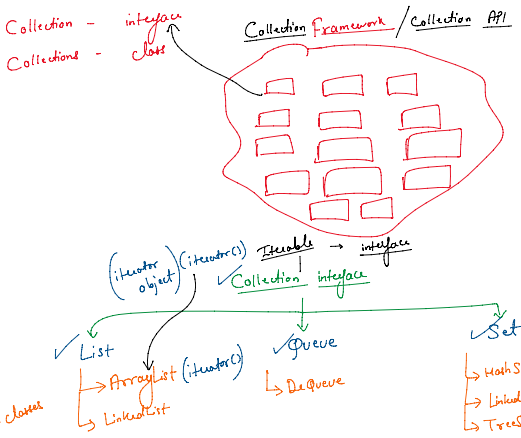






Thread States





Map

↳ TreeMap