Java

Methods & Functions

— Prateek Narang

18 - Jan - 2022

Functions / Methods

+ Memory Management

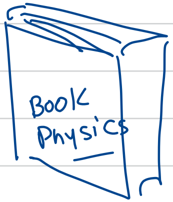
Part -1

- → ✓ Functions
- → ✓ Parameters
- → ✓ Return types
- → ✓ Scope (Fn)
- → ✓ Call Stack

Part-II

- → Stack vs Heap
- → Primitives vs objects
- → Garbage Collection

Functions / method

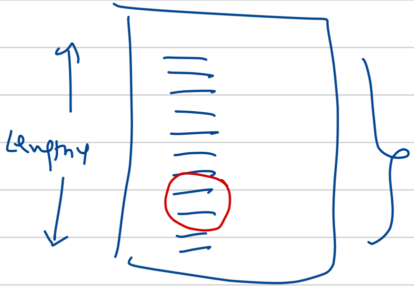


Chapters
↳ Specific Theme
[Gravity]
↳ Light

→ Organised
→ Re-used ✓
→ Readability

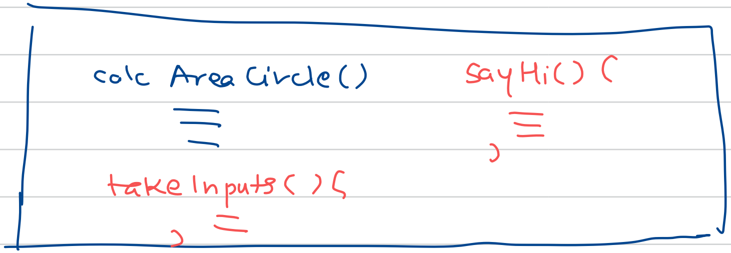
⇒ Million Lines of Code

⇒ Large Code



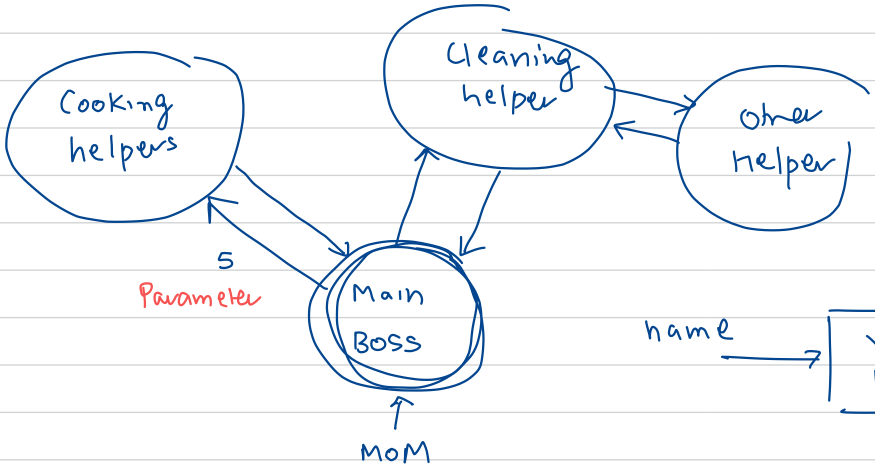
→ Low Readable
→ Maintain
→ Extend
→ Re-usable ✗

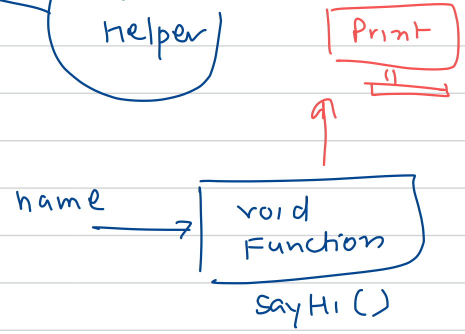
Code

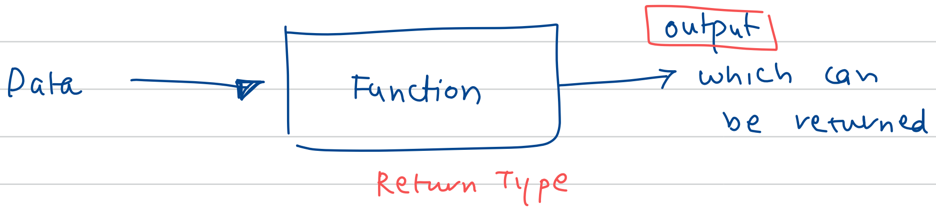


```
calc AreaCircle()
    ≡

takeInputs( ){
    ≡
}

sayHi() {
    ≡
}
```

Function / Method : is a block of code executes/runs only when it is called. You can Also pass data knowns as parameters to a function. Functions or methods are used to perform certain actions.







Terminology

Function Declaration

visible

class Function

doesn't return anything

Name

Parameter

```
public static void sayHi(String name){
    System.out.println("Hi " + name);
}

public static void main(String[] args) {
    sayHi("Malay"); //Function Call
    sayHi("Rishab");
    Scanner sc = new Scanner(System.in);
    String name = sc.nextLine();
    sayHi(name);
}
```

Function Definition

Argument

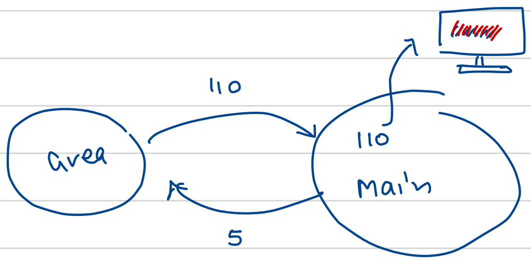
Think

Youtube

```
shareVideo() {          publishComment(){          Subscribe(){

}                       }                          }

uploadVideo() {         search() {

}                       }
```

```
                        True    False             Source         Account
public  static  boolean  uploadVideo ( VideoFile v , AccountId id ) {

                    Blobstore
                    processVideo (        );

}                   ~ True / False
```

Demo :

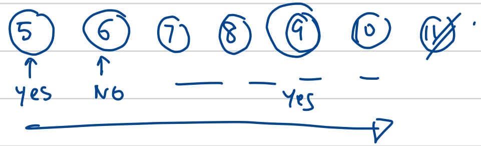


Modular print all Primes, Range A to B

- ↳ main( )
  - ↳ print Primes (A, B)
    - ↳ check Prime (No)

Call Stack

A = 5   B = 10

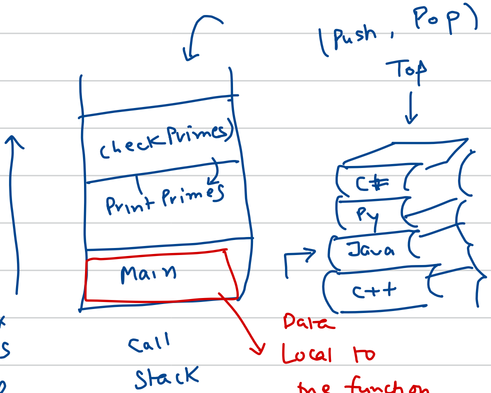


Total Fn A = 10 , B = 20

$\boxed{13}$

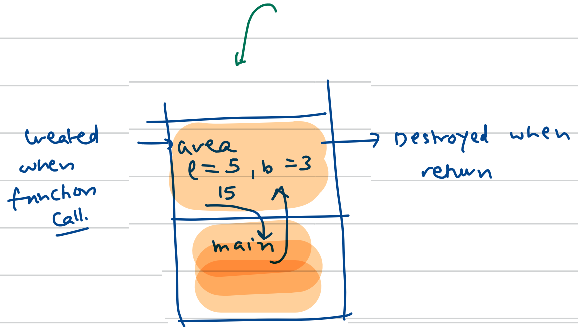
max fns inside stack

$\boxed{3}$



Data Local to the function "Stack Frame"

```
int area (int l, int b){
        return l×b;
}

void main() {
    area(5,3)
    = return;
}
```



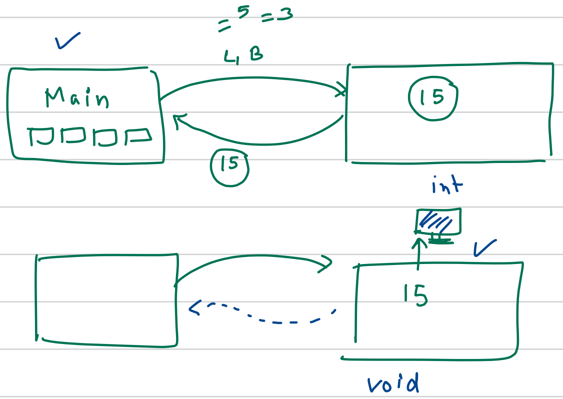
Stack is finally empty

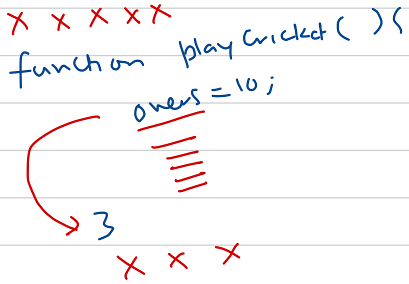
LIFO

(Last In First out)

Mom → ~~Cooking~~ → ~~Other~~

Last In First out





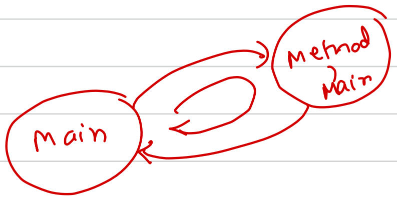
In Java, variables are only accessible inside the region they are created. This is called scope.

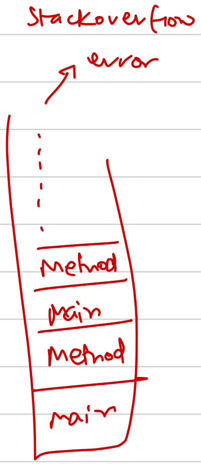
for(int i=0; i<=10; i++){

→ Block Scope

3

Variables declared directly inside a method are available anywhere in the method following the line of code in which they were declared


main
method
main


Stackoverflow
error
method
main
method
main

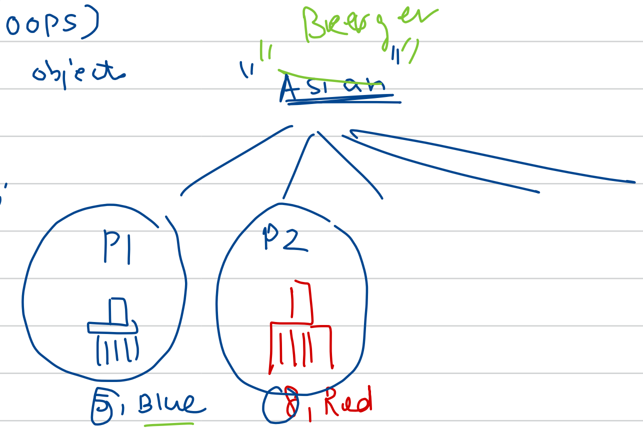
Static (oops)

template

Shared ~~as~~ across all object

```
class PaintBrush {
    static String brand = "Asian";
    Const int size;
    String color;
    |
    paint( ) ;
    changeColor( );
}
```

→ value can be set only once.



different for every paintBrush

object Name
↓
p1. color = "green"

Paintbrush. brand = "Beeper"

Class Name

code → file

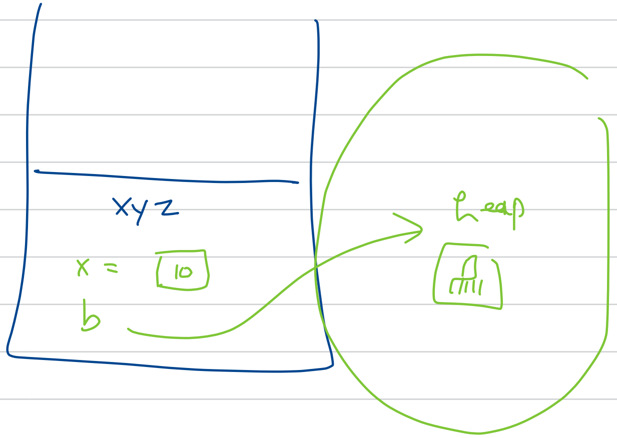
↓

[harddisk

↓

[RAM

```
Function xyz ( ) {
    {
        int x = 10;
        PaintBrush b = new PaintBrush(        object
                  5, "Blue");
    }
}

xyz();
```



To run an application in an optimal way, JVM divides memory into stack and heap memory.

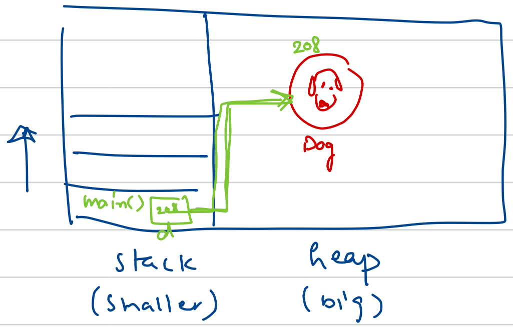
Stack vs Heap memory

↳ JVM

data

Primitives
- int
- long
- short

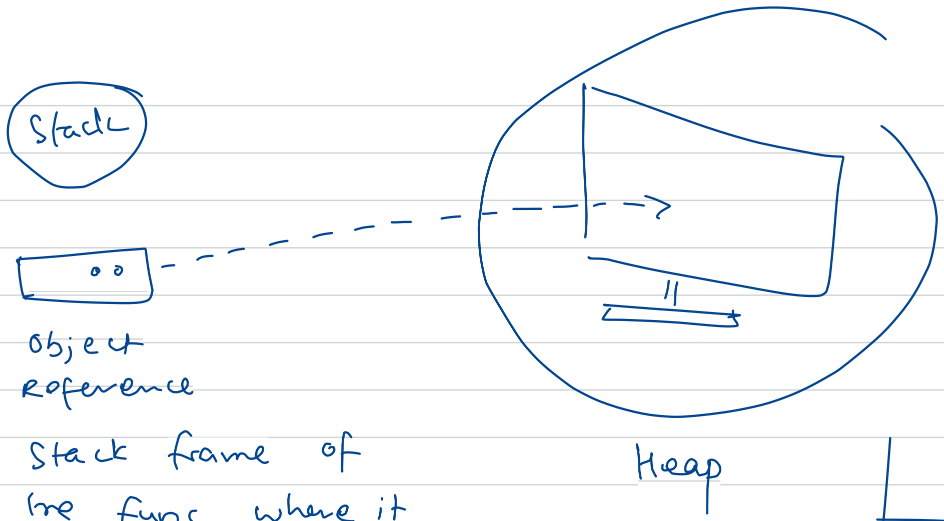
objects
- String
- Array
- PaintBrush
- Dog
- ⋮



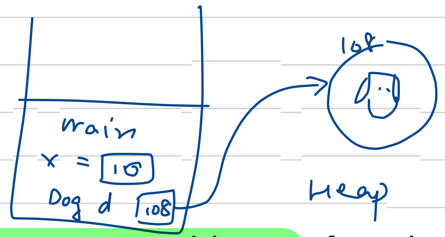
```
class Dog {
    — ht
    — wt
    — color
}

main() {
    Dog d = new Dog(); → object on heap
        int x = 10;
}
```

↑ Object Reference

Objects go to heap,
reference go on Stack.



(Stack frame of the func where it is created)

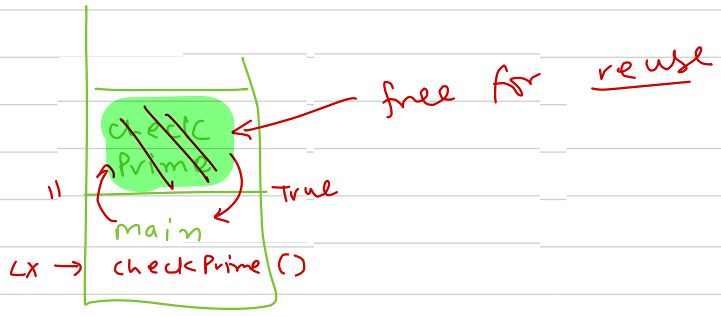


Stack Memory

It contains primitive values that are specific to a method and references to objects referred from the method that are in a heap.

Access to this memory is in Last-In-First-Out (LIFO) order. Whenever we call a new method, a new block is created on top of the stack.

When the method finishes execution, its corresponding stack frame is flushed, the flow goes back to the calling method, and space becomes available for the next method.



Key Features of Stack Memory

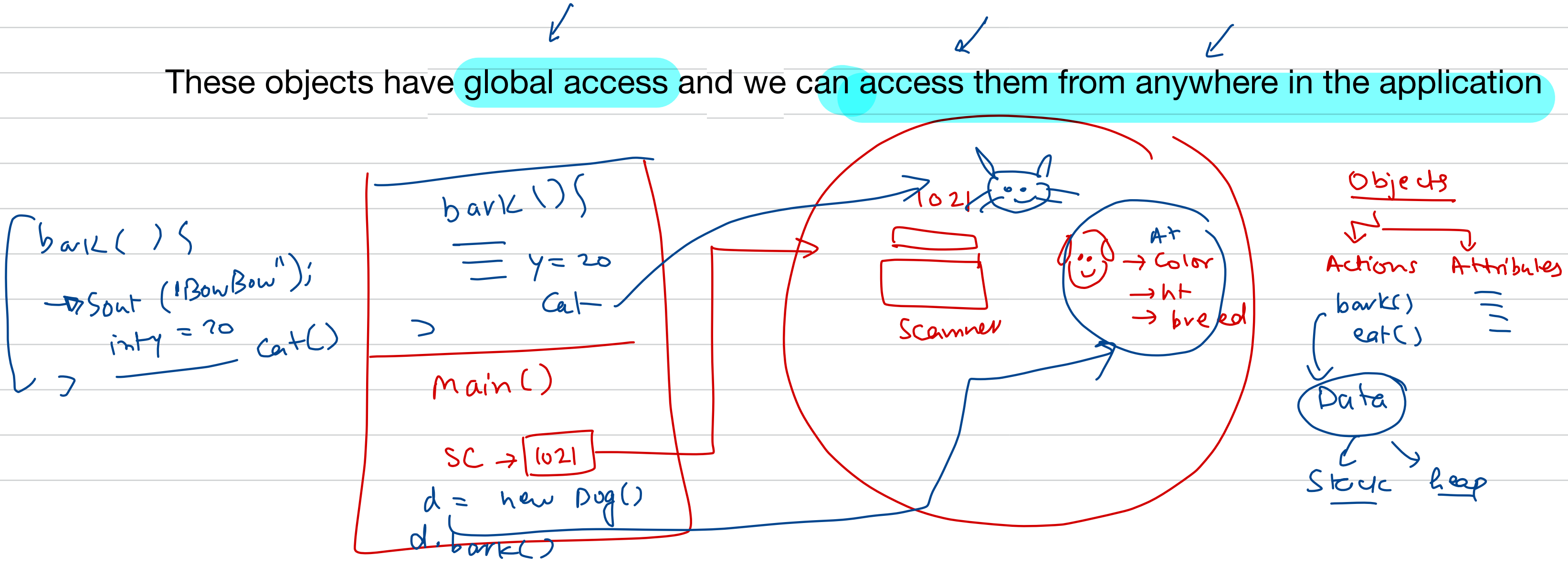
- It grows and shrinks as new methods are called and returned, respectively.
- Variables inside the stack exist only as long as the method that created them is running.
- It's automatically allocated and deallocated when the method finishes execution.
- If this memory is full, Java throws java.lang.StackOverFlowError.
- Access to this memory is fast when compared to heap memory

Heap Memory

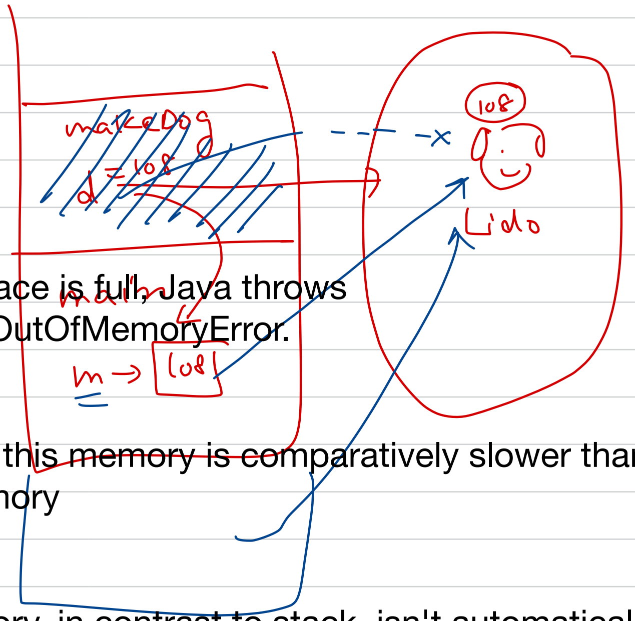
Heap space is used for the dynamic memory allocation of Java objects at runtime.

New objects are always created in heap space, and the references to these objects are stored in stack memory.

These objects have global access and we can access them from anywhere in the application



```
makeDog() {
    Dog d = new Dog("Lido")
    return d;
}

main() {

    Dog m = makeDog()

}
```



If heap space is full, Java throws java.lang.OutOfMemoryError.

Access to this memory is comparatively slower than stack memory

This memory, in contrast to stack, isn't automatically deallocated. It needs Garbage Collector to free up unused objects so as to keep the efficiency of the memory usage.

If heap space is full, Java throws **java.lang.OutOfMemoryError.**

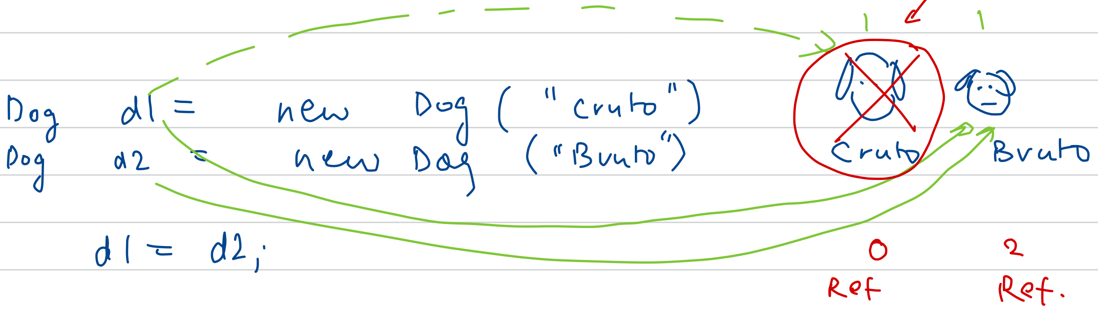
Access to this memory is comparatively **slower than stack memory**

This memory, in contrast to stack, **isn't automatically deallocated**. It needs Garbage Collector to free up unused objects so as to keep the efficiency of the memory usage.

[Life & Death on Heap]

Free up the unused objects

heap

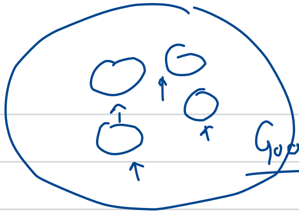


Garbage Collection deals with finding and deleting the garbage(unused objects) from memory.

However, in reality, Garbage Collection **tracks each and every object** available in the JVM heap space and removes unused ones.

In simple words, GC works in two simple steps known as Mark and Sweep:

Mark – it is where the garbage collector identifies which pieces of memory are in use and which are not
Sweep – this step removes objects identified during the “mark” phase



Good

↳ Unused memory is automatically managed / freed up

↳ memory leak (avoid)

Bad

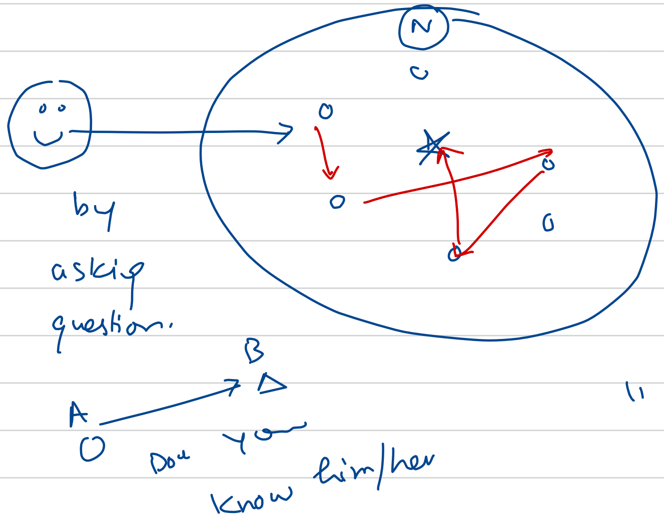
↳ CPU more power

↳ No control, when & what time ?
over scheduling.

↳ Not that efficient as "manual" memory management.

↑ delete obj;

Celebrity Puzzle

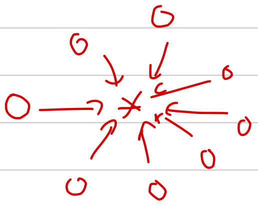


★ Star celebrity
~~kno~~ knows no one

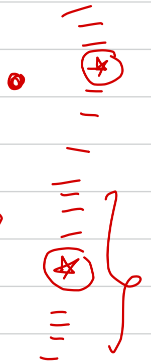
○ Know celebrity
but they may/may not know each other.

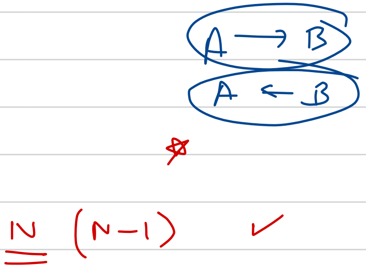
"Min Questions to find the celebrity"
& How?

Amazon Int



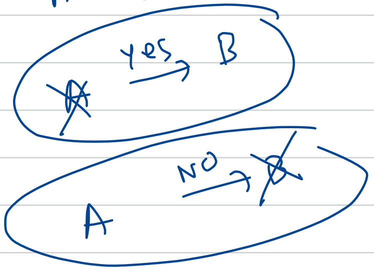
N-1 Questions $\boxed{N-1}$

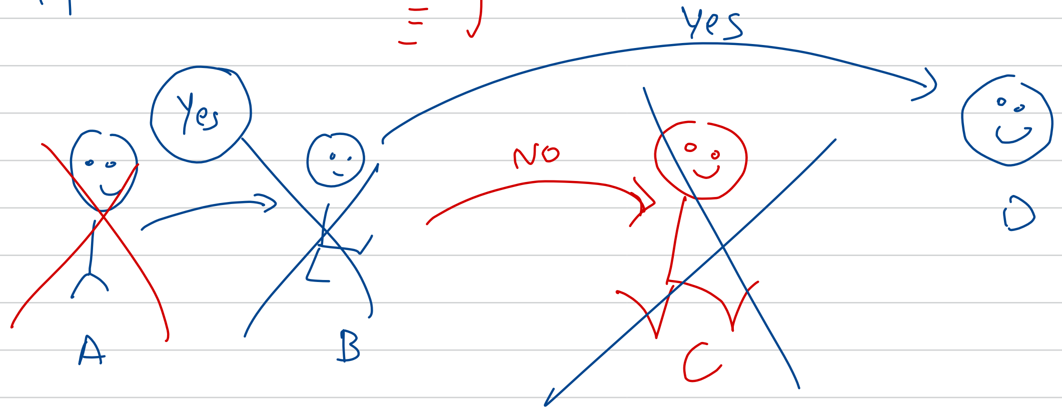




1 Question → 1 person

Ask q




Yes
Yes
NO
A
B
C