Book.

→ Head First Java

Breadth

Depth of libraries

=> (online doc, Reference)

JRE

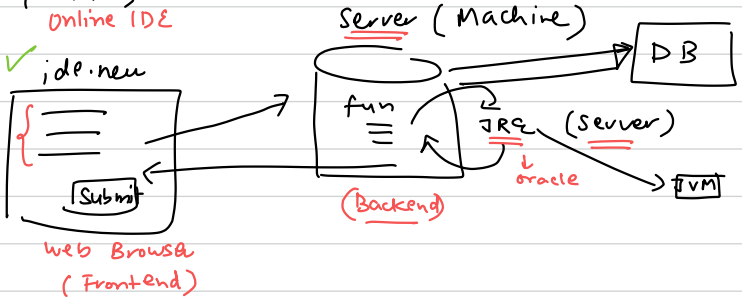
online ide

→ Intellij (Community Edition)

→ Online IDE (Cider)

{ → vscode
  → Eclipse

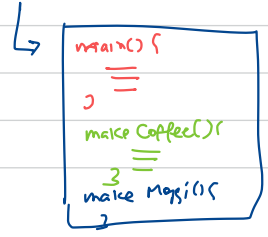
Online IDE

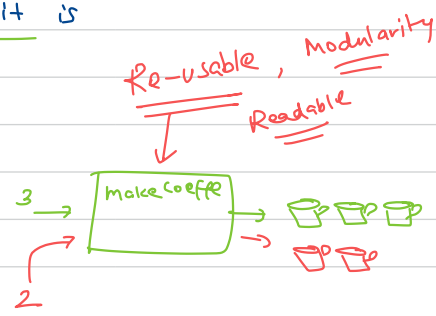


Agenda

- ✓ → Functions / methods
- ✓ → Return Types, Parameters
- ✓ → Call Stack
- ✓ → Stack vs Heap Memory
- ✓ → Object References
- ✱ → Garbage Collections
- ◎ → Problems + Puzzles (Next Class)

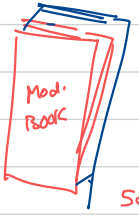
Methods → "Block" of code that run when it is called





500 Pages

A



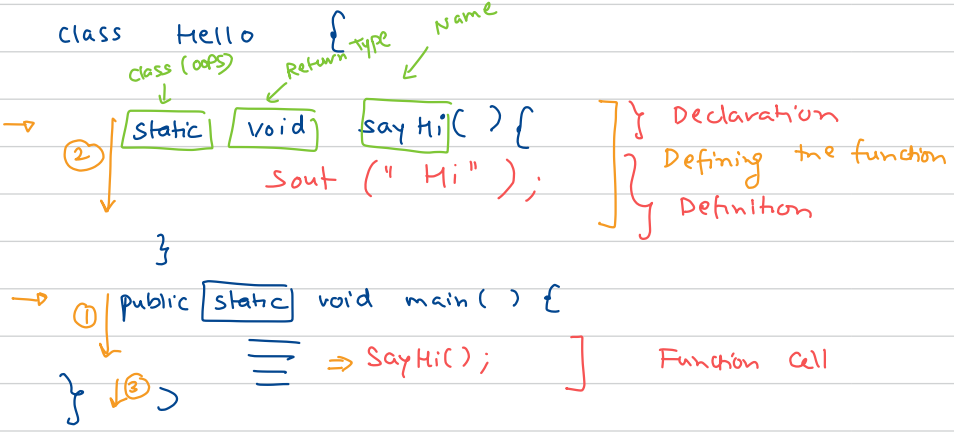
→ Chapters

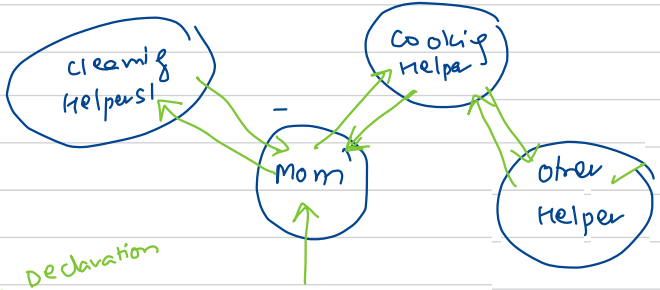
500 Pages

B

Why ?

→ organised
→ easy to understand
→ More readable
→ "Logical units"





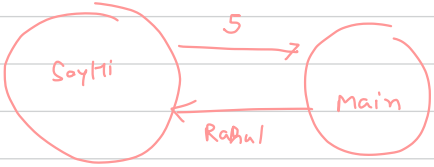
Function Declaration

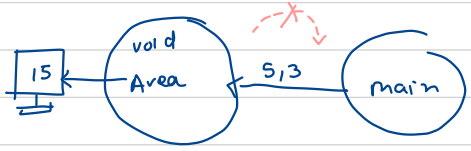
Main

Arguments / Parameters

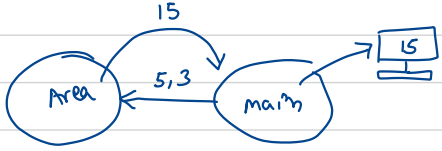
Definition { }

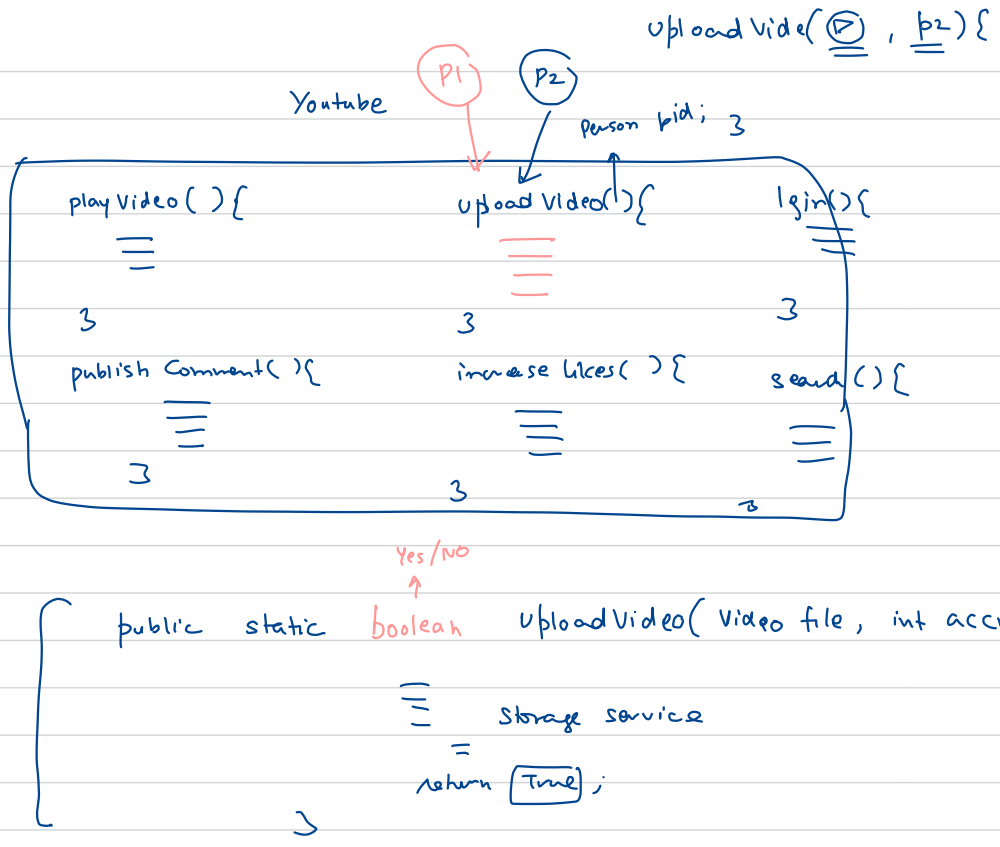
```
static void sayHi(String person){
    System.out.println("Hello " + person);
}

public static void main(String[] args) {
    System.out.println("In Main");
    sayHi("Rahul"); //Function Call - 1
    sayHi("Ruchika"); // Function Call - 2

}
```






```
Static void area (int L, int B){
        print ( L * B),
}
3
```




uploadVide(, p2) {
Youtube
P1
P2
Person bid; 3
playVideo( ){
uploadVideo( ){
login( ){
3
3
3
publish Comment( ){
increase likes( ){
send( ){
3
3
Yes/NO
public static boolean uploadVideo(Video file, int accnt_id){
Storage service
=
return Time;
}

Q. Write a modular code to print PRIMES in range A to B

→ check Prime(N) → Yes / NO

→ print All Primes(A, B)

→ main()

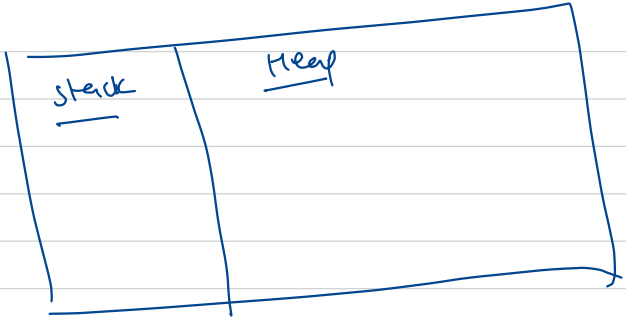
5 Mins

10.35

Stack vs Heap Memory

In order to run an application, the JVM divides the memory(RAM)
Into two parts - Stack Memory (smaller) and Heap Memory (Bigger)

Operations:
Declare a new variable, or you create a string, create a object —> either happen on the stack or on the heap.



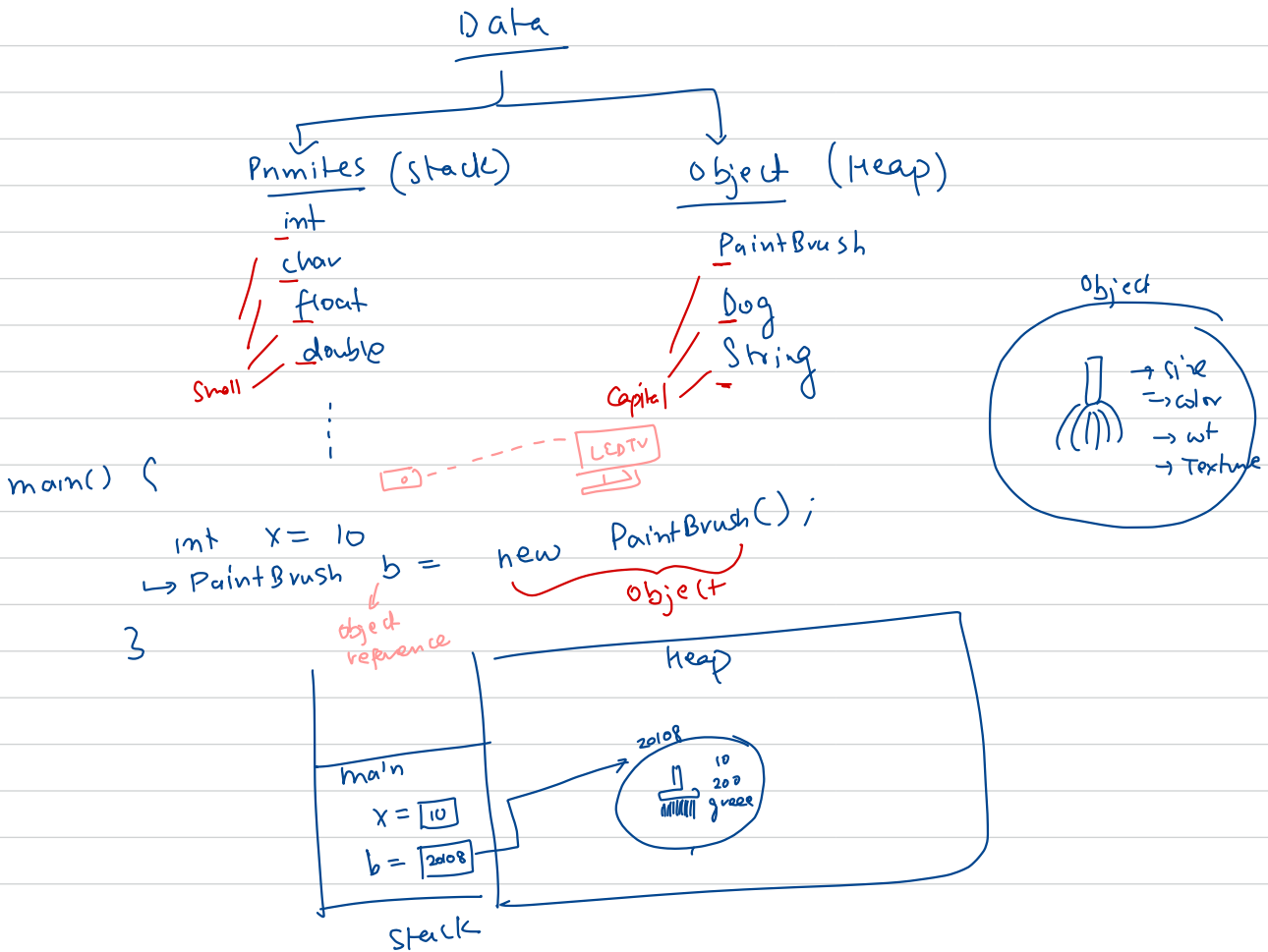
Stack

Predecided, can't change on run time

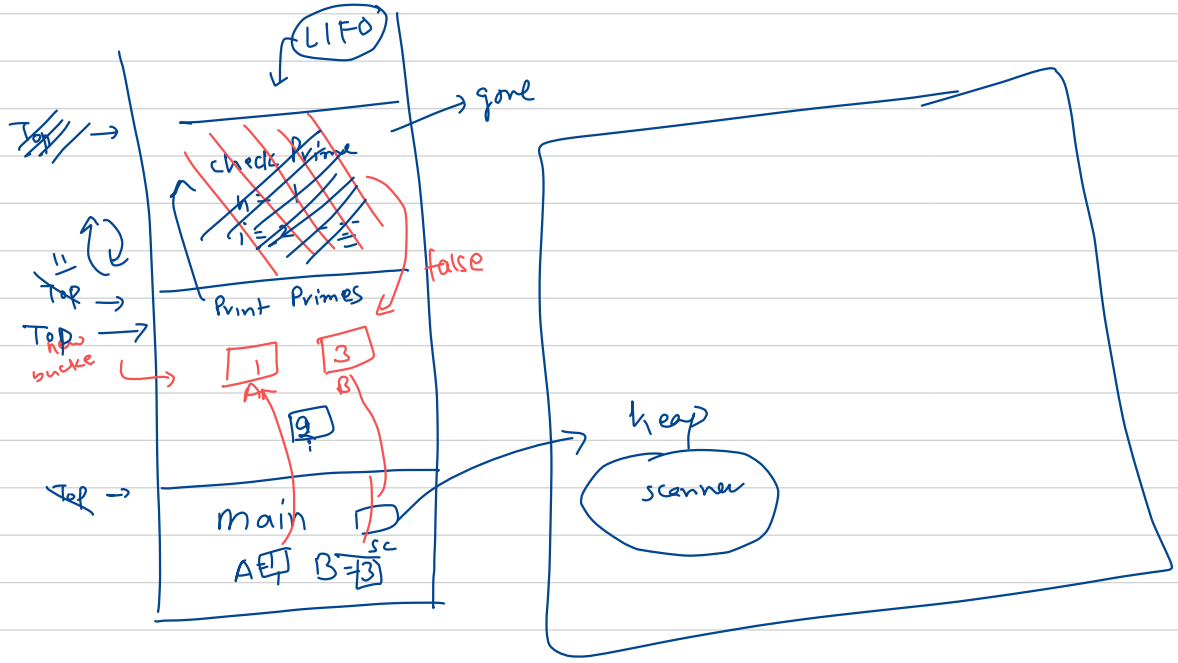
Used static memory allocation, primitives that you create, or the object references but actual objects go on the heap.

LIFO Order - whenever a new function is called, it is created at the top of stack. (Last In First Out) ordering.

When a method is finished executing, it gets removed from the top of the stack and space becomes available for re-use.


Data
Primitives (stack)
int
char
float
double
Small
Object (Heap)
PaintBrush
Dog
String
Capital
LCDTV
0
1
Object
size
color
wt
Texture
main() {
int x = 10
PaintBrush b = new PaintBrush();
object reference
object
}
Heap
main
x = 10
b = 2008
20108
10
200
green
stack

Call Stack


LIFO
gone
check Prime
false
Print Primes
main
A
B
sc
scanner
Heap

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
.
Access to this memory is fast when compared to heap memory

String h = "hello" → Pool
"hello" object (Heap)

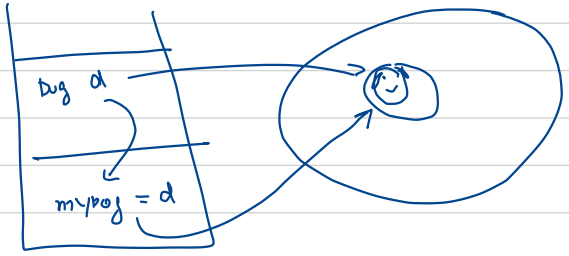
Heap Memory

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

Dog d = new Dog

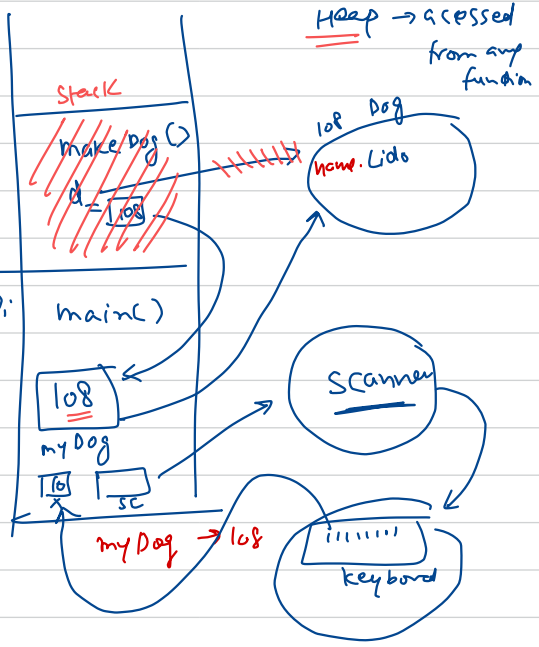
New objects are always created in heap space, and the references to these objects are stored in stack memory.

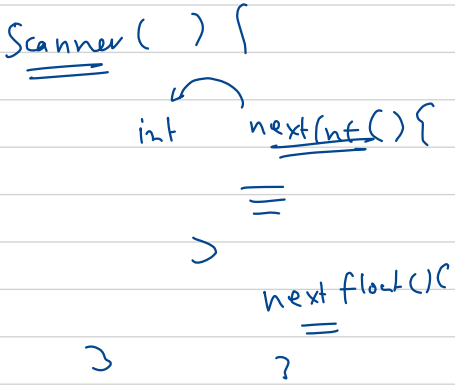
These objects have global access and we can access them from anywhere in the application

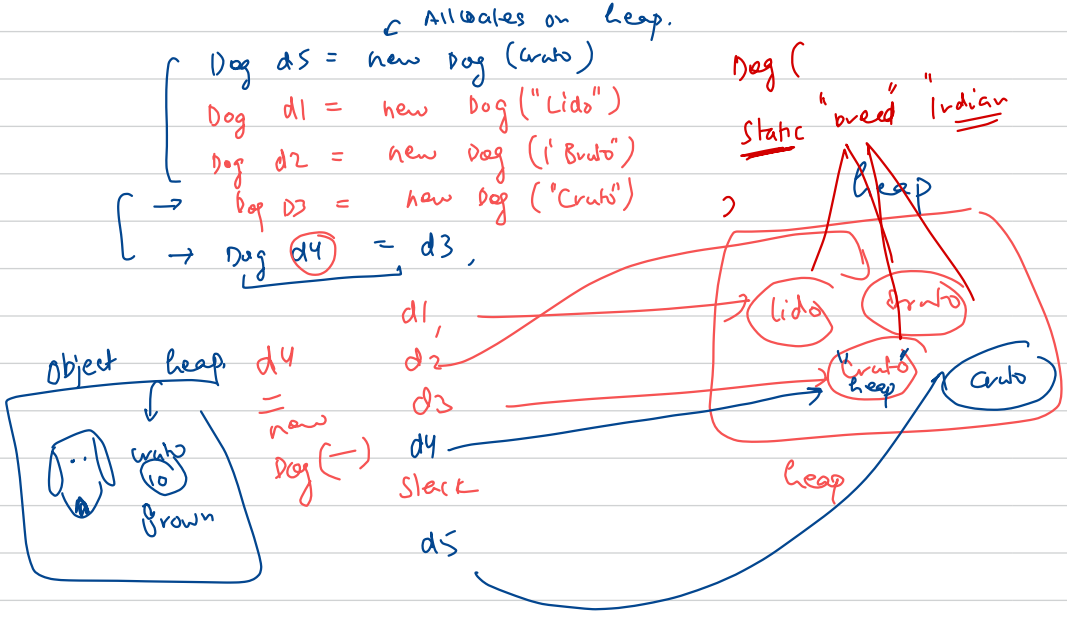


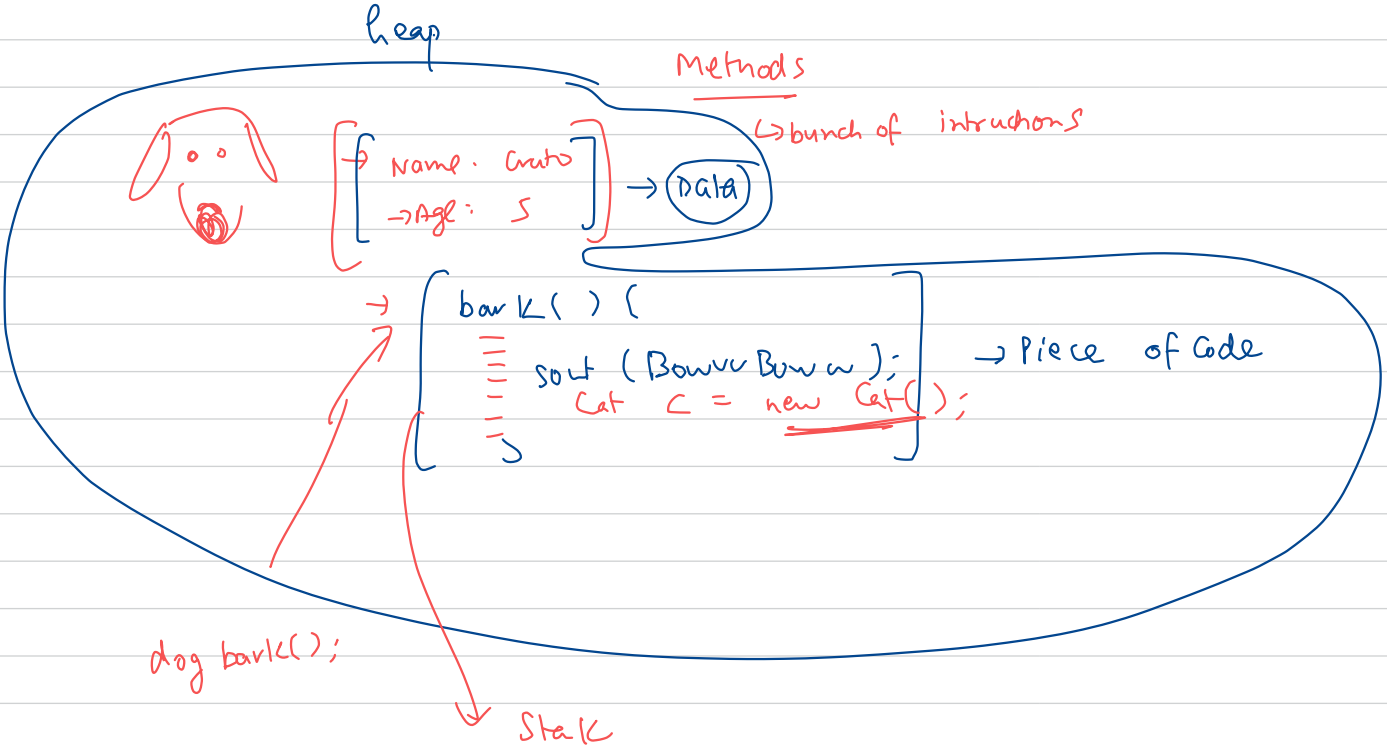
```
Dog makeDog() {
    Dog d = new Dog("Lido");
    return d;
}

main() {
    Scanner sc = new Scanner();
    int x = sc.nextInt();
    Dog myDog = makeDog();
    System.out.println(myDog.name);
}
```

Lido




Scanner ( ) {
int
nextInt() {
}
nextFloat() {
}


Allocates on heap.
Dog d5 = new Dog (Cruto)
Dog d1 = new Dog ("Lido")
Dog d2 = new Dog ("Bruto")
Dog d3 = new Dog ("Cruto")
Dog d4 = d3;
Dog (
Static
"breed"
"Indian
heap
Lido
Cruto
Cruto
heap
Cruto
d1,
d2
d3
d4
Stack
d5
heap
Object heap
d4
=
new
Dog(--)
Cruto
10
Brown


Heap
Methods
↳ bunch of instructions
Name: Cruto
Age: 5
Data
bark( ) {
sout (Bowwe Bowww);
Cat C = new Cat( );
}
→ Piece of code
dog bark( );
Stack

Heap Memory Features

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 the Heap

" garbage collectible Heap"

GC

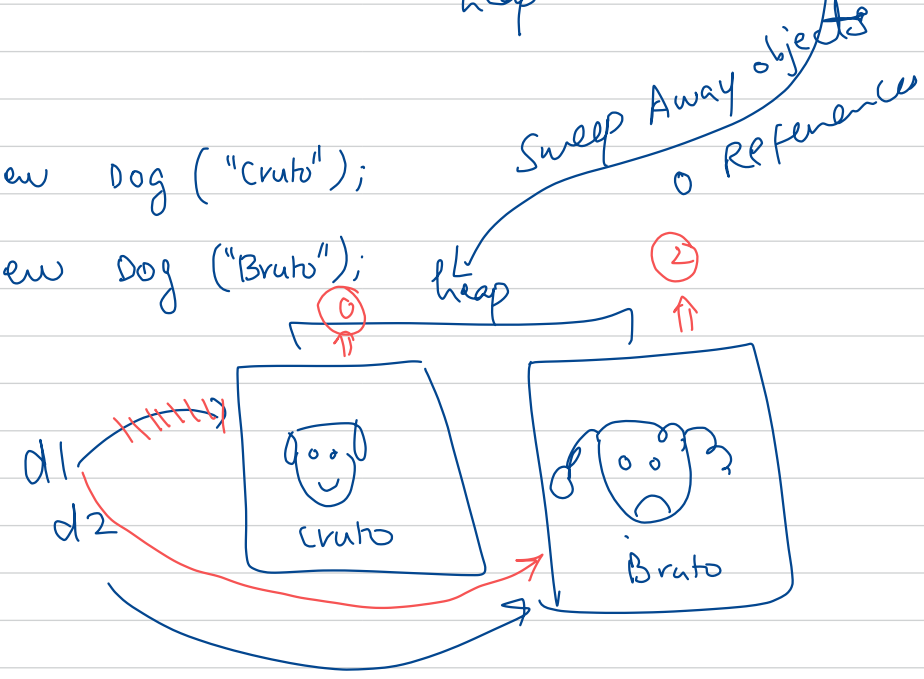
Sweep Away objects

0 References

① `Dog d1 = new Dog("Cruto");`

① `Dog d2 = new Dog("Bruto");`

`d1 = d2;`



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