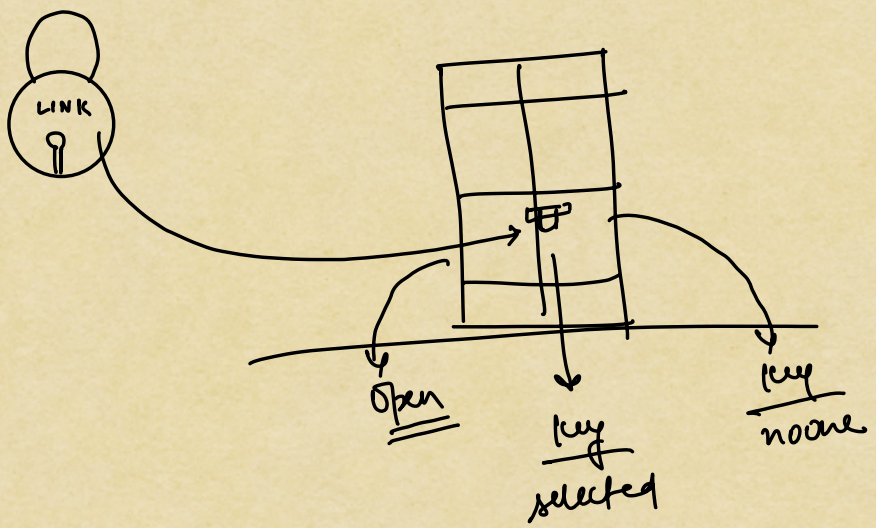
today
- → Access modifiers
- → Constructor → types
- → Destructor

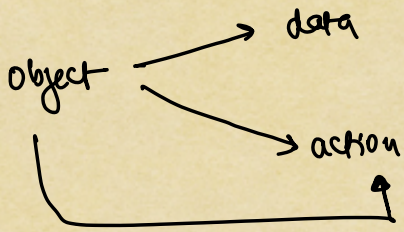
next
- → Inheritance
- → Polymorphism → types

- **Access modifiers :-**

access modifier => something that can modify access to anything



=> Classes and Object :-



Class

↓

blueprint

or.

cake tin/

cake mould

using this

cake tin/mould.

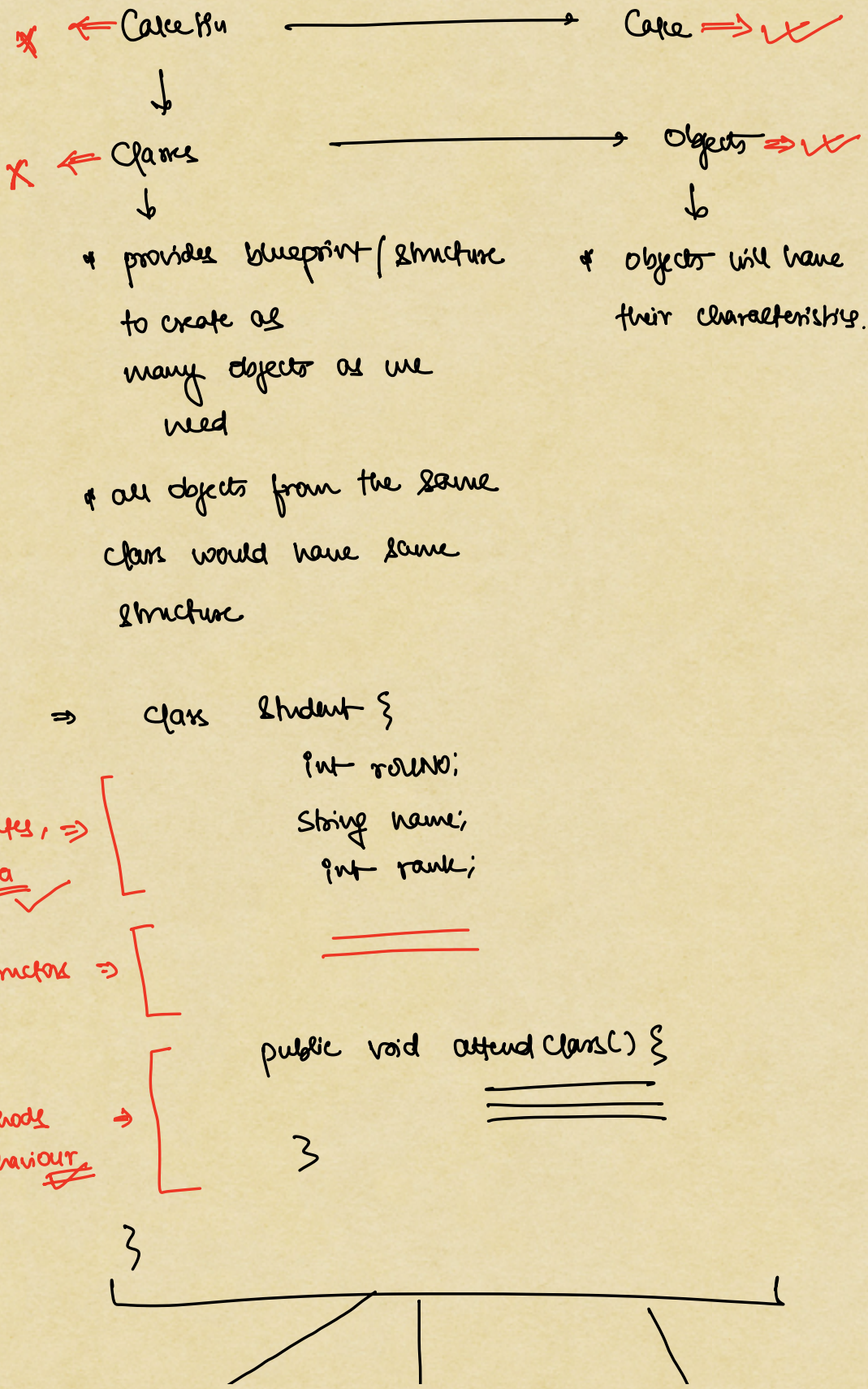
I can make as many cakes as I want

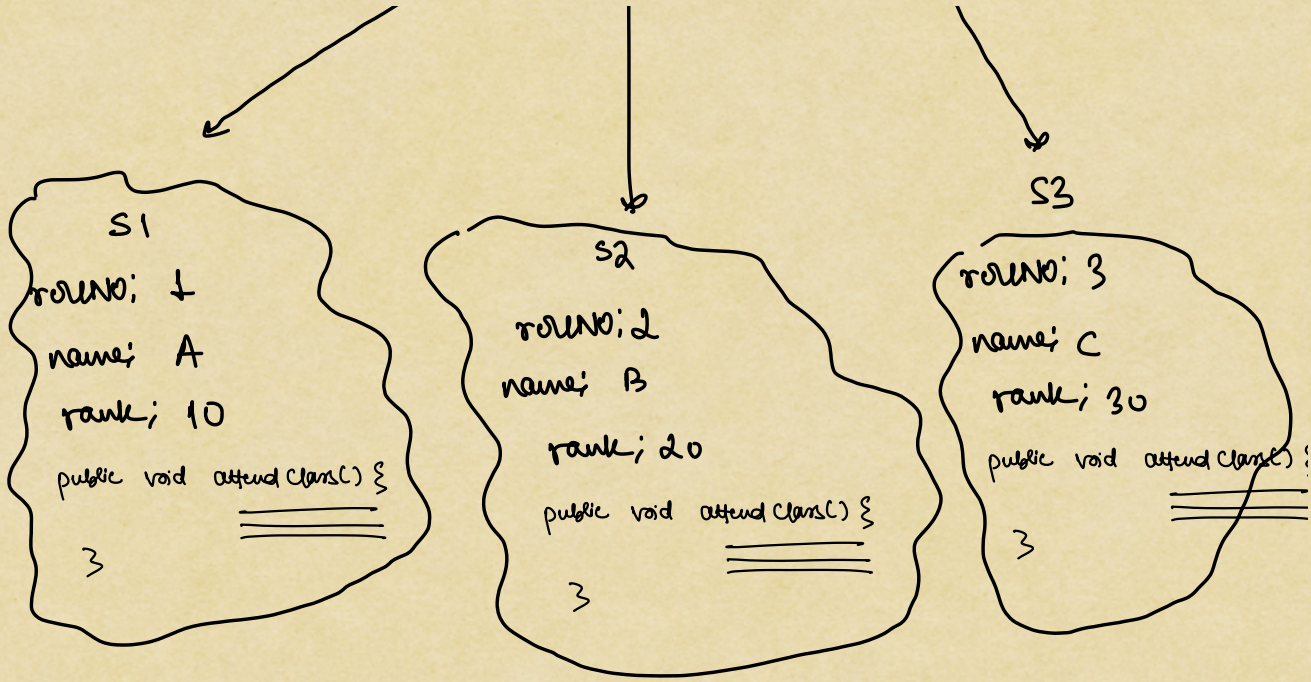
↓

↓

* every cake built from this cake tin, would have some structure [size, weight, circumference, diameter]

* structure might be same but every cake can have its own characteristics → flavour → top design
→ toppings


Class
Objects
* provides blueprint/structure to create as many objects as we need
* objects will have their characteristics.
* all objects from the same class would have same structure
⇒ class Student {
int rollNo;
String name;
int rank;
attributes, ⇒ or, data
Constructors ⇒
public void attendClass() {
}
methods ⇒ or, behaviour
}



3 things to access:-

i) data points → attributes

ii) constructors

iii) methods

Access modifiers

(anywhere in code)

	Class	package	child (same pkg)	child (diff pkg)	World
public	✓	✓	✓	✓	✓
private	✓	✗	✗	✗	✗
protected	✓	✓	✓	✓	✗
default (package-private)	✓	✓	✓	✗	✗

⇒ Constructors

Class Student ⇒ Object

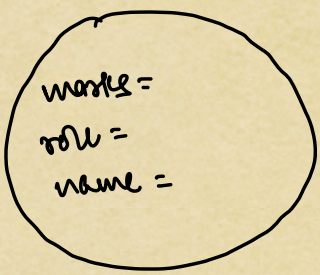
⇓

Syntax ⇒ `Student sObj = new Student();`

- `Student` → ClassName
- `sObj` → Object Name
- `=` → `new` (keyword)
- `Student()` → Call to constructor

```
class Student {
    int marks;
    int roll;
    String name;
}
```

⇒



Object [memory]

⇒ constructor helps to initialise the data to create objects.

- ⇒ Default constructor
- ⇒ Manual / Args / Parameterized constructors
- ⇒ Copy constructor

⇒ Default constructor

* when we dont add any constructor, to the class then while compilation java automatically add a constructor ⇒ "default constructor".
  - ↓
  - Initialises all the variables with their default values

```
public Student() {
}
```

⇒ default constructor

- name of constructor & class is same
- Constructor is not a func/method
- written by developer we call it
- No Args constructor

⇒ Manual / Args / Parameterised constructor:-

```
public Student(int sMarks, int sRoll, String sName){
    marks = sMarks;
    roll = sRoll;
    name = sName;
}
```

⇒ Copy constructor:-

iPhone ⇒ 11, 55000, A11, 12MP, Battery, Screen.

iPhone ⇒ 12, 65000, A12, _____

iPhone ⇒ 13, 7500, A12, _____

iPhone ⇒ 14, 80000, A13, _____

iPhone ⇒ 10 attributes

↓

new

→ 3 datapoints

→ 7 same

default ⇒ 10 lines

param ⇒ 10 values

iPhone11

↓

(iPhone11)

↓

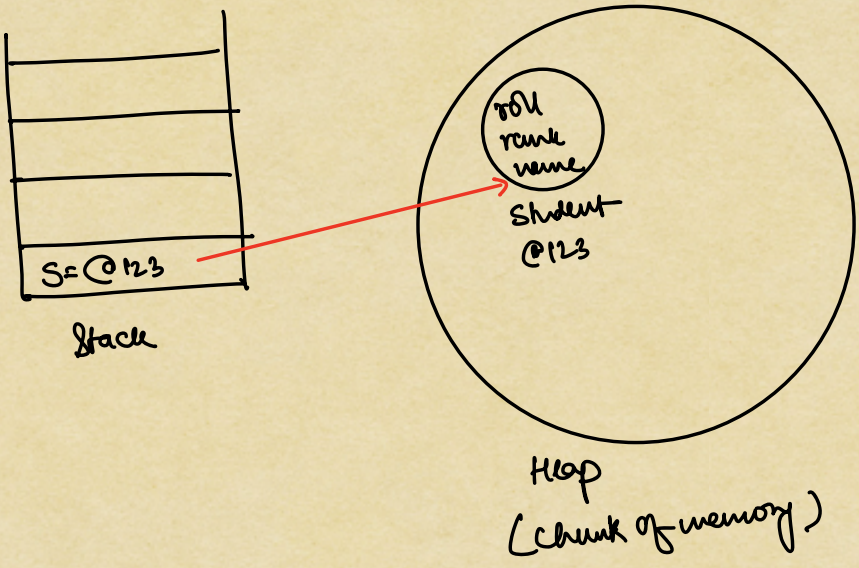
iPhone12

(iPhone12)

↓

iPhone13

=> Deep and shallow copy



Student s = new Student();

reference variable

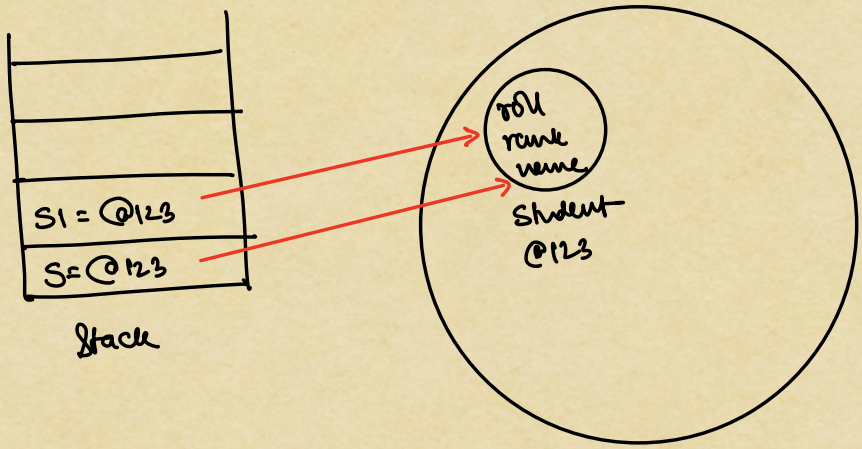
i) ref. variable => Stack

ii) Object => heap

iii) address of object stored in ref. variable

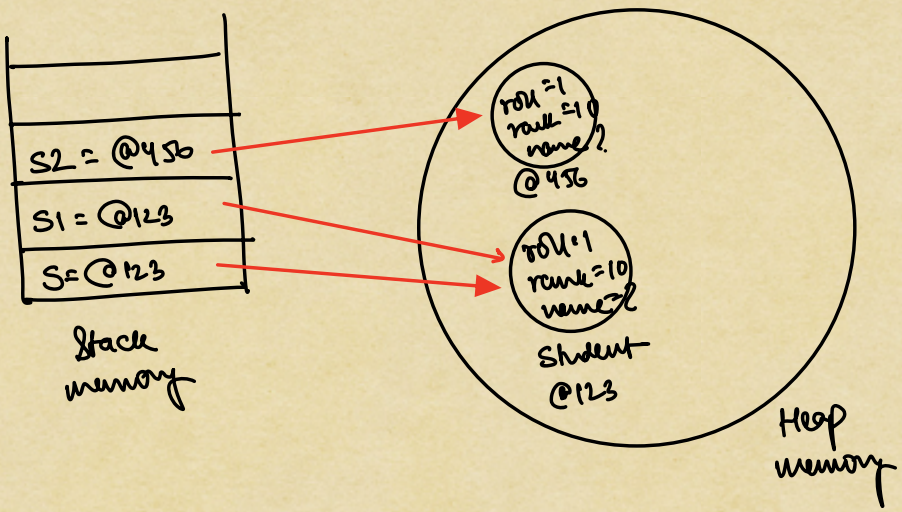
Student s = new Student(1, 10, "Dinraj");

Student s1 = s; ← shallow copy




```
Student s = new Student(1, 10, "Dhruva");
Student s2 = new Student(s);
```

← deep copy



Shallow copy → 1 more ref. variable

---

copy is happening only for reference variable

Deep copy ⟶ 1 ref + 1 Object (new)

copy is happening only for values,
we are getting a new ref. variable as
well as new object.

- Pass by value || Pass ~~by reference~~

↓

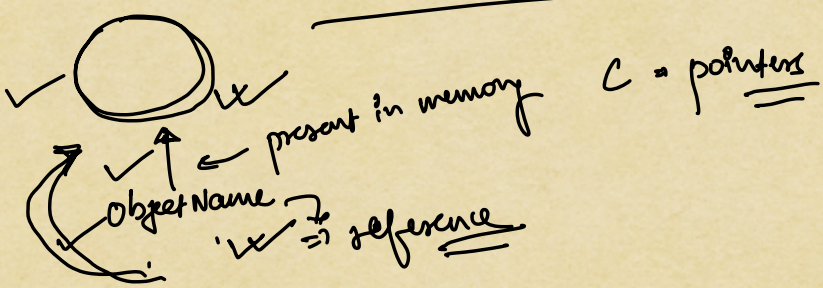
only pass by value

Java is strictly pass by value, but the value it passes the address stored in the ref. variable, inside the method a shallow copy happens

C

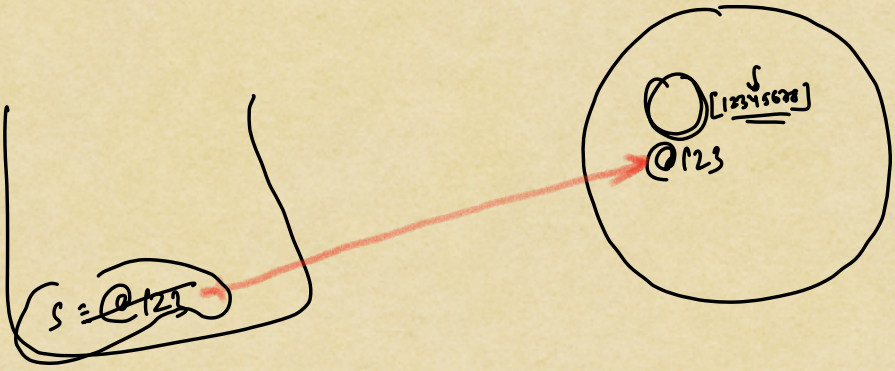
references ⇒ global

ObjectName = {object}



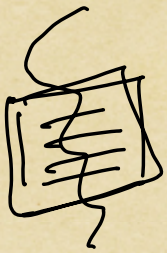
C ⇒ pointers

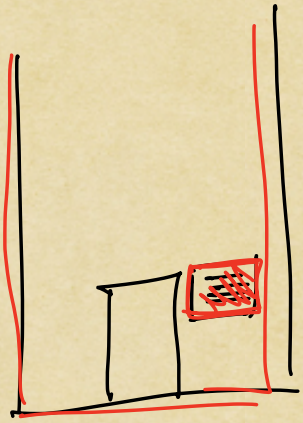
Student s = new Student

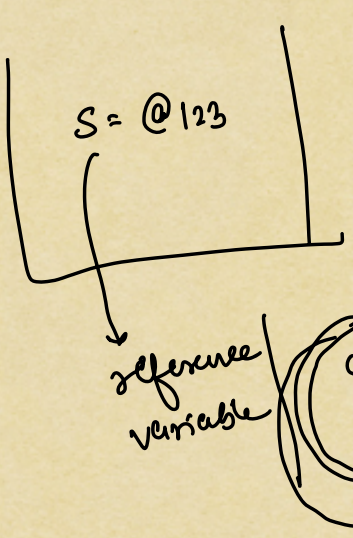


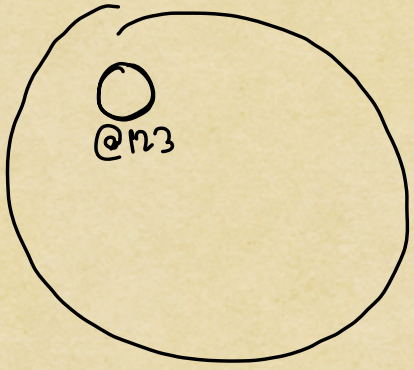
s = null

s = @456









pass by value

address

=> Destructor

Java → ✗

C++ → ✓

Garbage Collector : automatically destroy Object