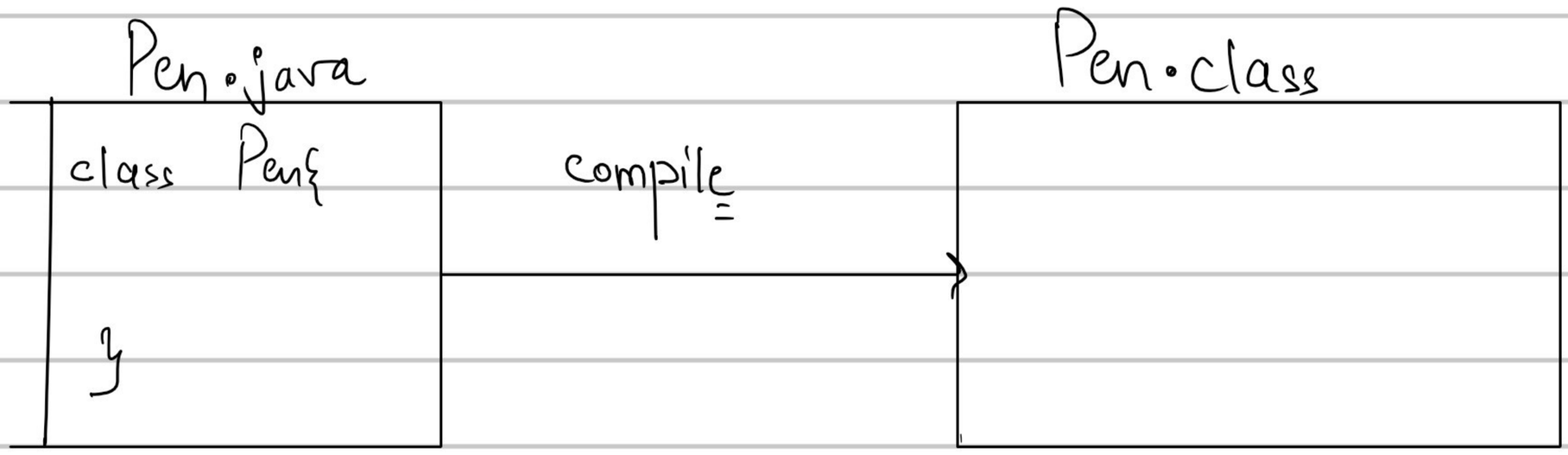
Java

→ object oriented programming language.

Everything in Java is an object.



Components of Java Environment.

JDK → Java Development Kit

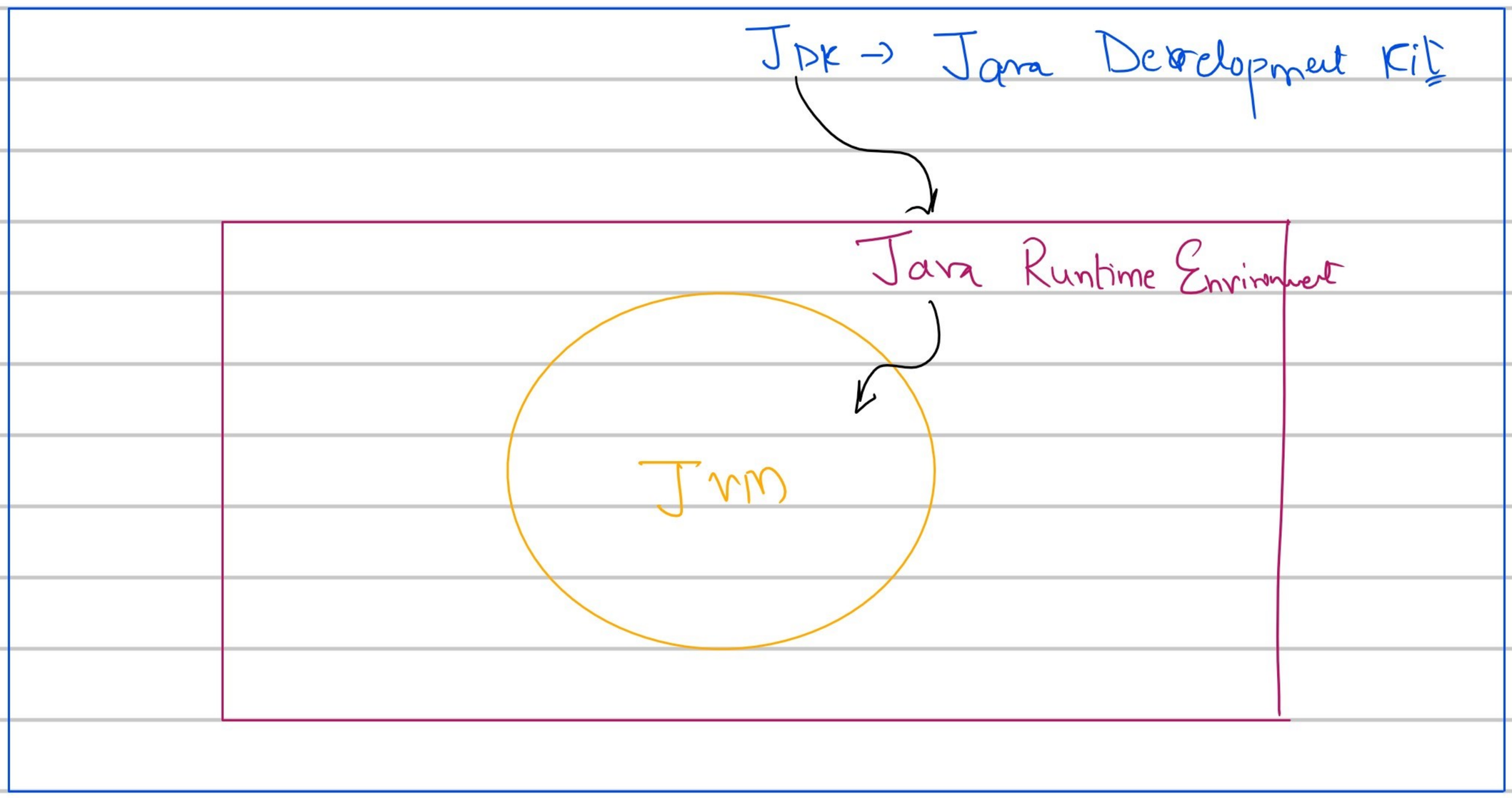
JRE → Java Runtime Environment

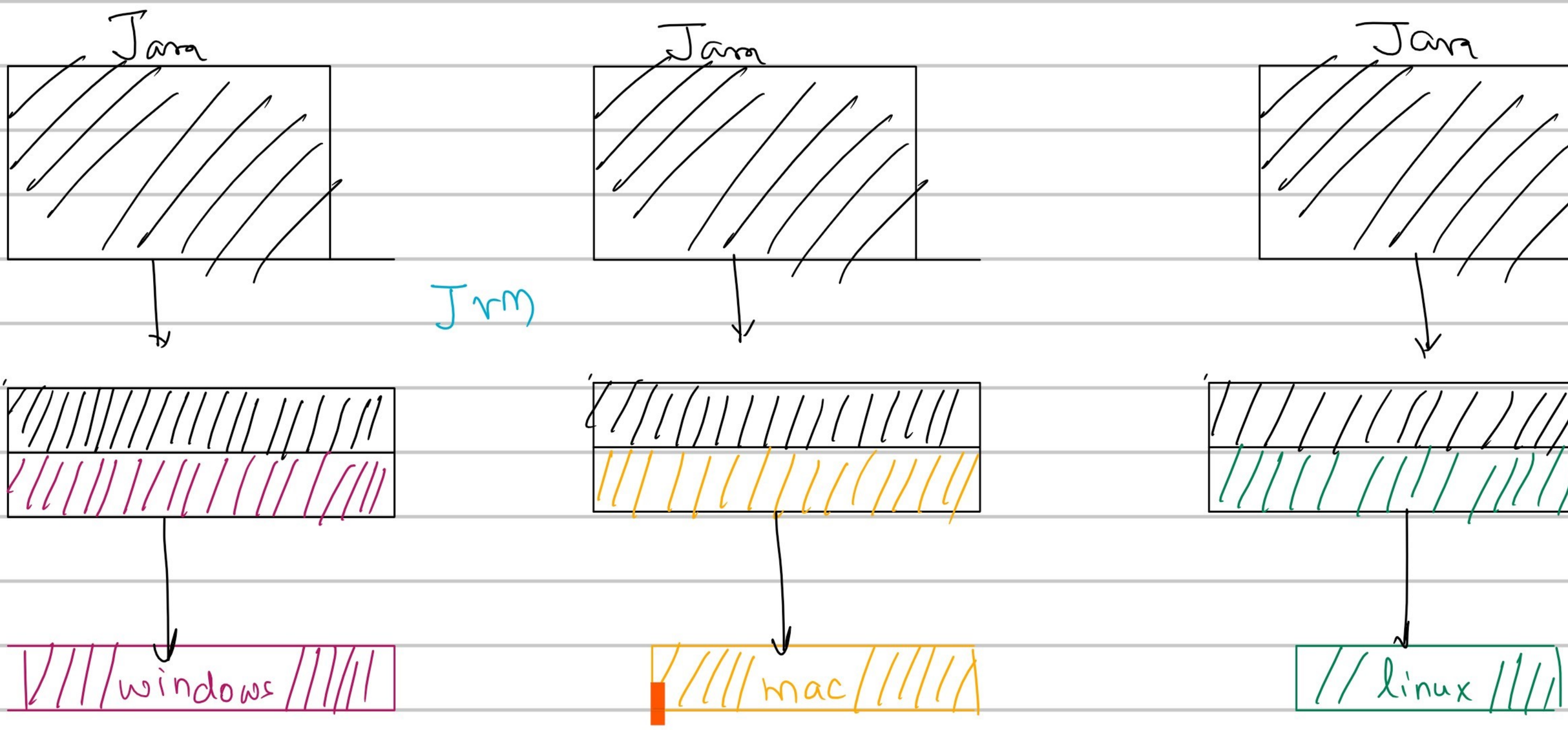
JVM → Java Virtual Machine

virtual machine

↓

a machine that runs inside another machine.





JAVA IS PLATFORM INDEPENDENT

because

JVM IS PLATFORM DEPENDENT!!

Simple HelloWorld program

HelloWorld.java

Name of the class

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello World");
    }
}
```

where application get started

Class / Interface → Pascal Casing → OneTwoThreeFour

Methods / Variables → camel Casing → oneTwoThreeFour

Java supports 8 data types

Name	value it holds	bytes	range
byte	number	1	± 127
boolean	true/false	2	true or false
char	'a', 'b'	2	0 to 65535
short	number	2	± 32767
int	number	4	± 2147483647
long	large number	8	± 9223372036854775807
float	real	4	3.4×10^{38} to 1.4×10^{-45}
double	real	8	1.7×10^{308} to 4.9×10^{-324}

Control statements

Operators

Loops

Arrays

Arrays in Java are object who has a container inside it which can hold elements of a similar type.

type → primitive data type / class

DECLARE

```
type[] nameOfArray;
```

```
int[] iArray;
Pen[] pArray;
```

CONSTRUCT

```
nameOfArray = new type[size];
```

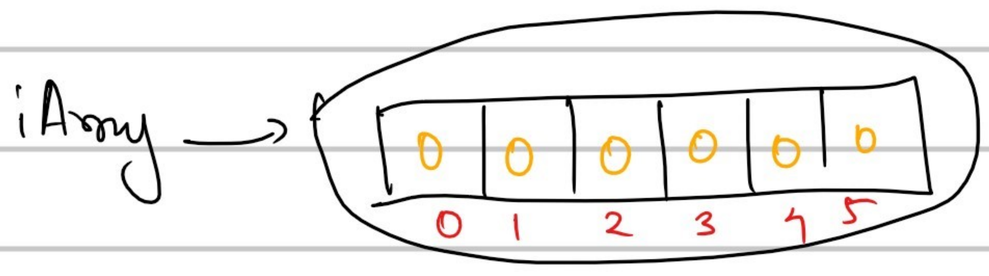
```
iArray = new int[6]
pArray = new Pen[3];
```

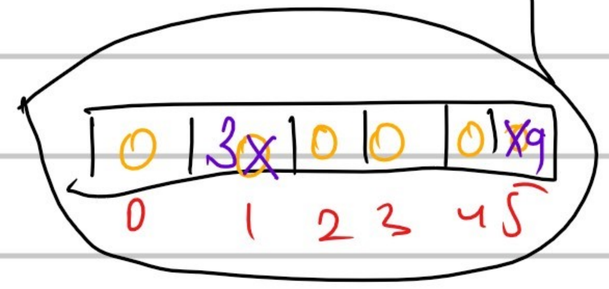
INITIALIZE

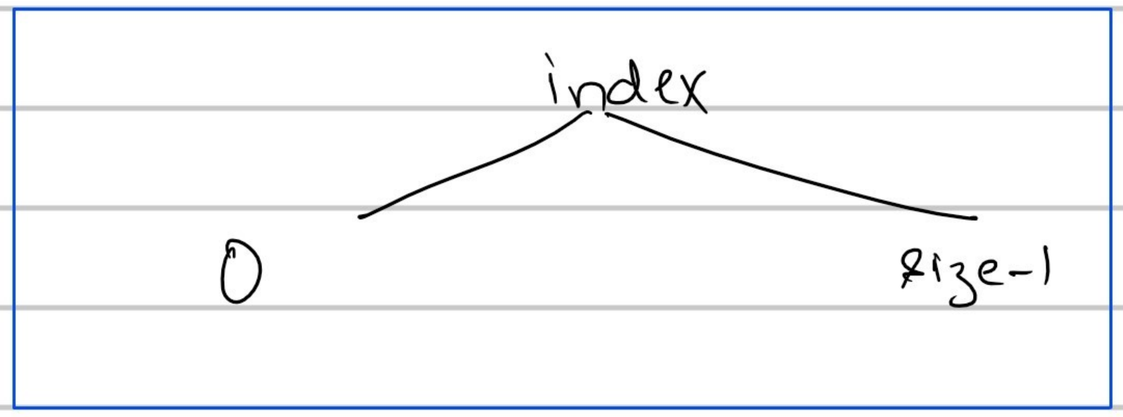
```
nameOfArray[index] = value;
```

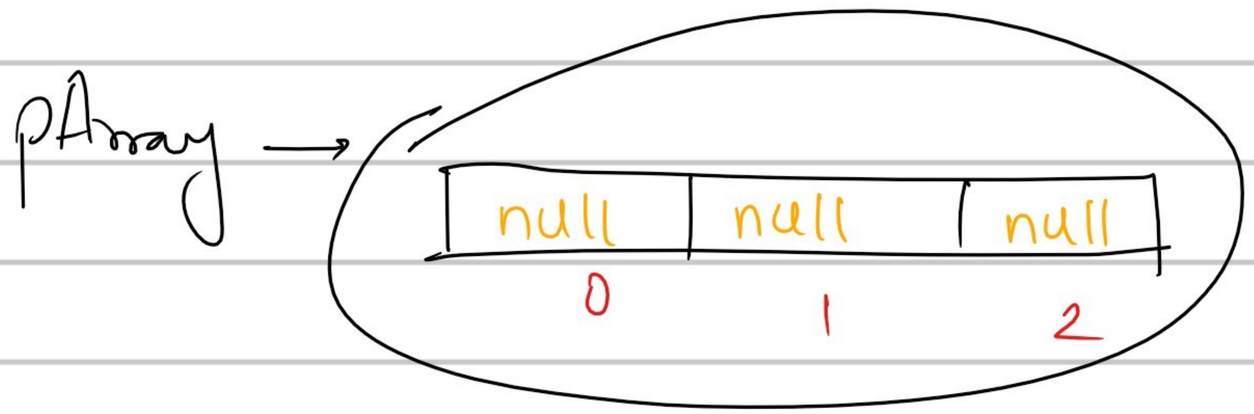
```
iArray[1] = 3;
iArray[5] = 9;
iArray[7] = 20;
```

Exception









Length of array.

int len = nameOfArray.length;

Initialized

nameOfArray[index] = someValue

Use

int x = nameOfArray[index]

S—— (nameOfArray[index]

* Software Objects * → Object Oriented Language

→ objects which replicates real world entity.

→ tangible or intangible.

ability to be measured

Pen
Car
Oxygen level
Air Pressure

→ Anything which has magnitude can be considered as a software object

All the software objects have 2 basic characteristics.

① What it has/have ? ⟹ STATE → primitive data
→ objects with HAS A relationship.

② What it does ? → BEHAVIOR → methods
→ Constructional
→ operational

* Class *

→ Classification

↳ a template or a blueprint which is used to describe an object.

→ Class is the place where working of an object is defined.

→ An object will support only those things which are defined inside it's class.

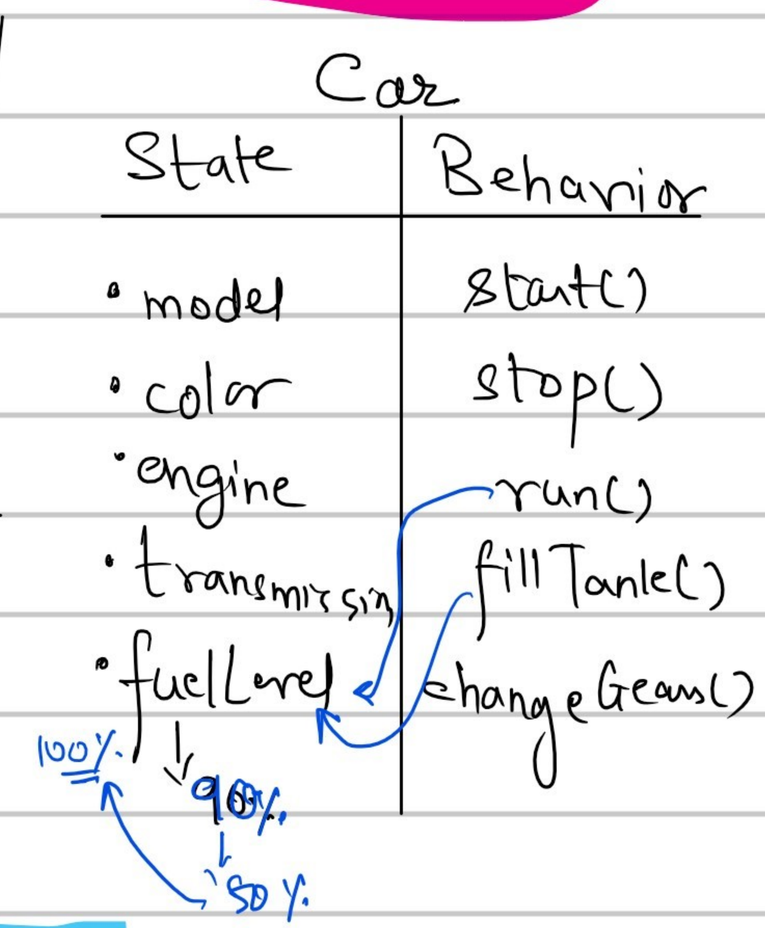
NOTHING LESS
NOTHING MORE

EAGLE cannot Swim

DOLPHINS cannot FLY

* State & Behavior are connected to each other *

Behavior acts on the State of an object & changes it from one point to another.



run() acts on fuelLevel of Car;

changes the fuelLevel from 90% to 50%

E-Commerce

* product
* Category
* customer
* orders
* seller
* order_details

Class, Interface ⟶ Pascal Casing → OneTwoThreeFour

Methods, Variables → Camel Casing → oneTwoThreeFour

```
class NameOfClass
{
    variables;

    methods

    Constructor

    block

    { inner class }
}
```

```
class Pen
{
    int inkLevel = 90;

    public void write() {
        =
    }

}
```

Object

reference of class

example of a class.

```
ClassName * objectName;

String s;

Pen p;

Can c;
```

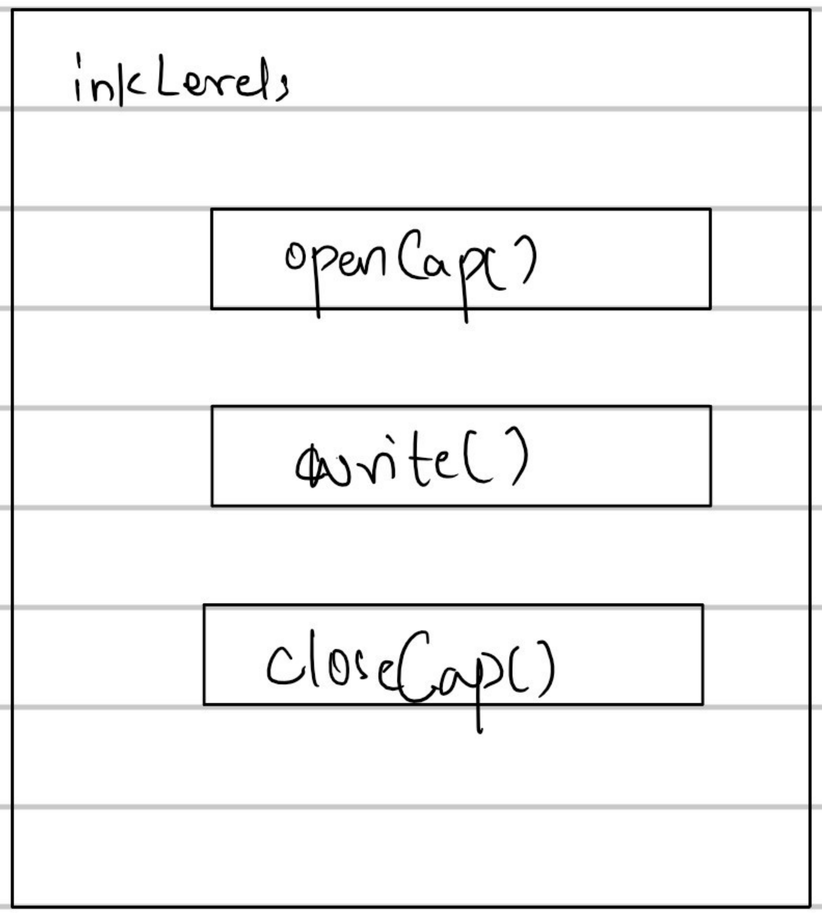
p is an object of Pen

c is an object of Can.

Objects in Java are not operational on declaration; they need to be instantiated.

Instantiation → new + Constructor

Pen



Pen p; → object

p.write(); ✗ → object is not instantiated

p = new Pen();

instance

Objects are not operational instances are

Memory Graph

Behavior

methods

Constructional

Operational

Methods

'CODE ON DEMAND'

Collection of statements which are grouped under a single name & performs a specific task.

STRUCTURE OF A METHOD

public private protected default — access-specifier

final, static, abstract — access-modifier [optional]

return-type: ① void ② primitive ③ class

methodName — can be any valid identifier.

parameters [optional] — how methods take input.

```
access-specifier access-modifier return-type methodName (parameters) {
    statement(s)
    .
    .
    .
    last-line
}
```

if a method doesn't return anything (void) there is no obligation on last line.

but if a method returns something; the last line has to be a 'return' statement

```
class MyMethods {

    public void display() {          // No Parameter, No Return Type
        System.out.println("RCB");
    }

    public void add(int a, int b) {  // Parameters, No Return Type
        int c = a + b;
        System.out.println(c);
    }

    private int getTaxRate() {       // No Parameter, Return Type
        return 18;
    }

    public int square(int x) {       // Parameter, Return Type
        return x * x;
    }
}
```

Invoking the Method

To invoke a method; we need an instance of a class & then make use of . (dot operator)

```
MyMethod obj = new MyMethod();
obj.display();
obj.add(1, 2);
int sq = obj.square(5);
int tax = obj.getTaxRate();
```

Constructor

↳ Constructor are the special type of methods.

Constructor:

① Name is same as class-name. (even the cases should match)

Pen → Pen() / Pen(x) ✓

Car → car() / car(x) ✗

② No return type (not even void)

* Constructor are of two types *

① Default ② Parameterized

```
class Pen {

    Pen() {          ──────→ Default
    }

    Pen(String color){  ─────→ Parameterized
    }
}
```

* Constructors are invoked when an object is instantiated.

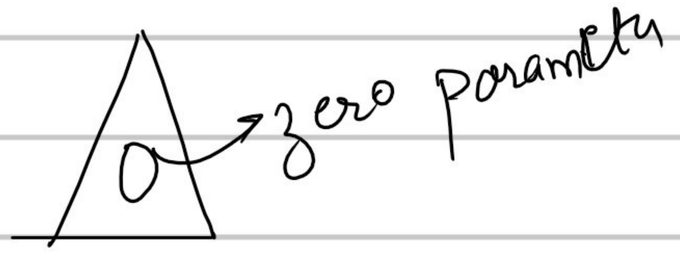
Constructor defines the state in which object will be created.

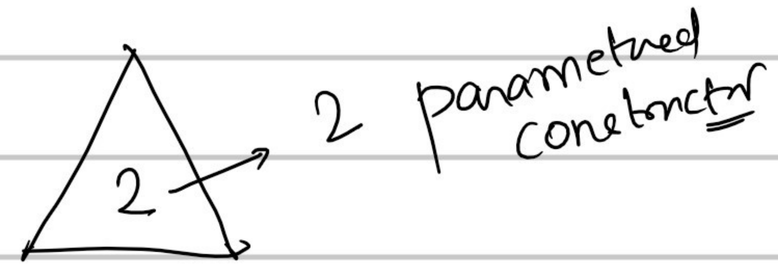
Constructor	Methods
→ invoked while object is getting ready	→ invoked after an object is ready

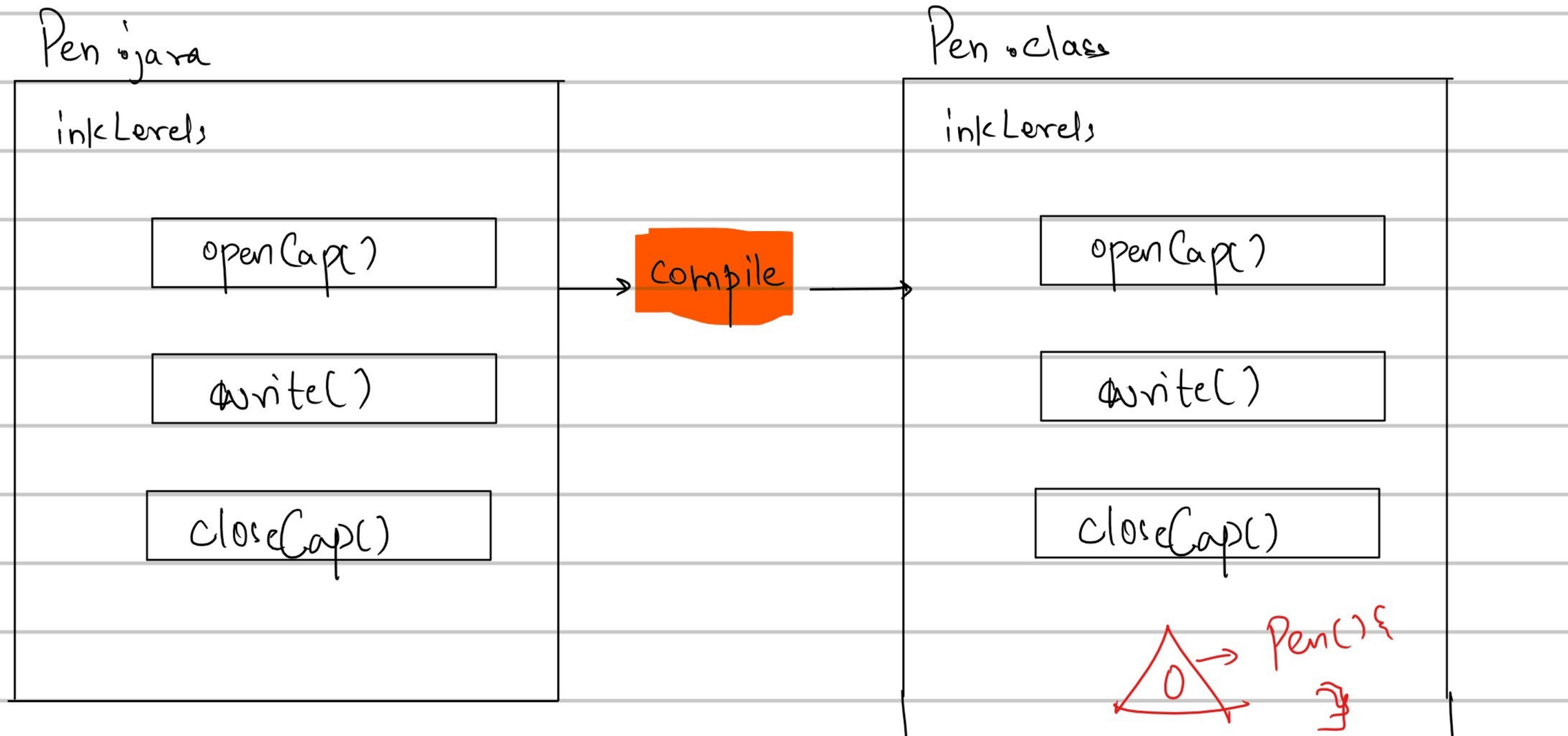
Tom gets ready & go to the office & work

Constructor → ready

methods → work

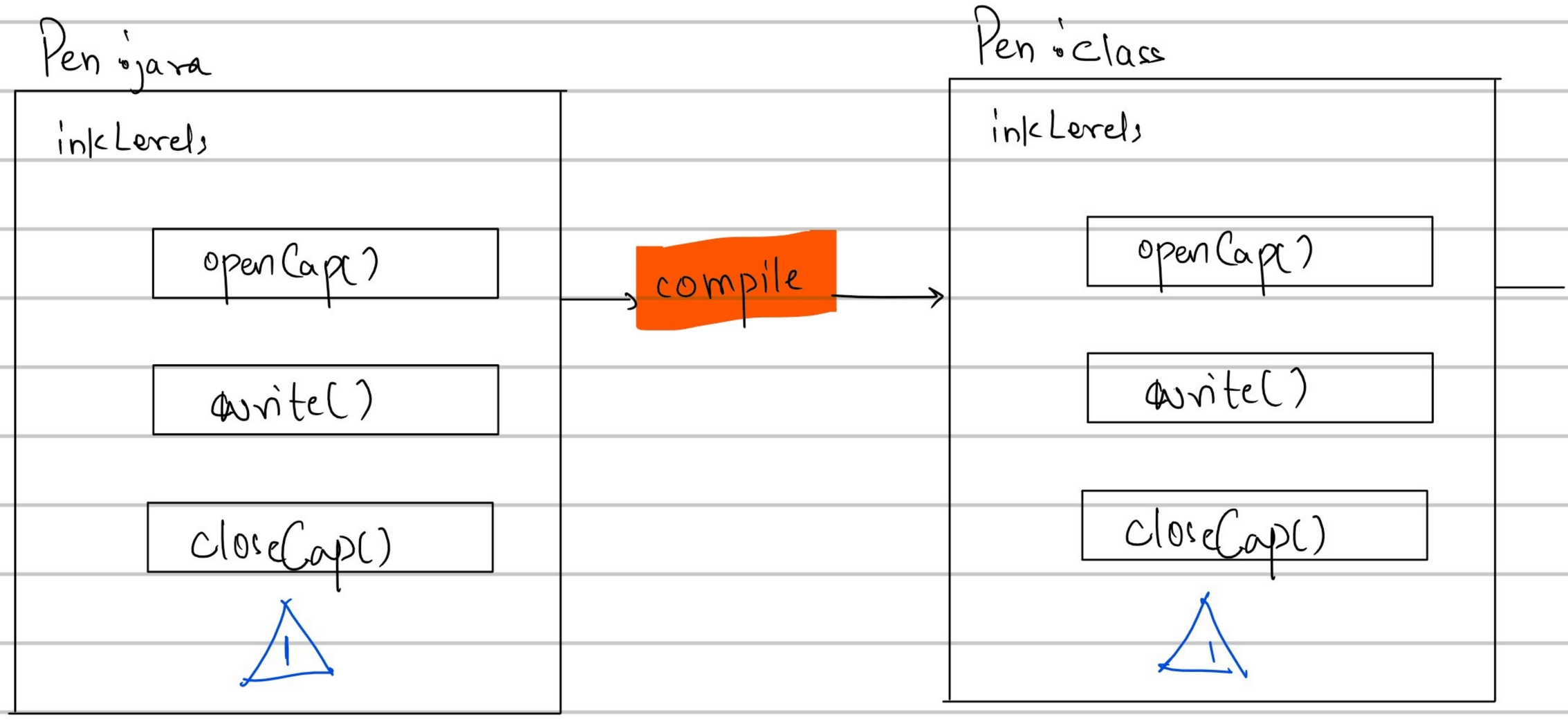






Whenever a .java file is compiled & if it doesnot have any constructor definition inside it; Java Compiler provides a free default constructor in its .class file

```
Pen p=new Pen(); ✓
p.openCap(); ✓
p.fly(); ✗
```



State

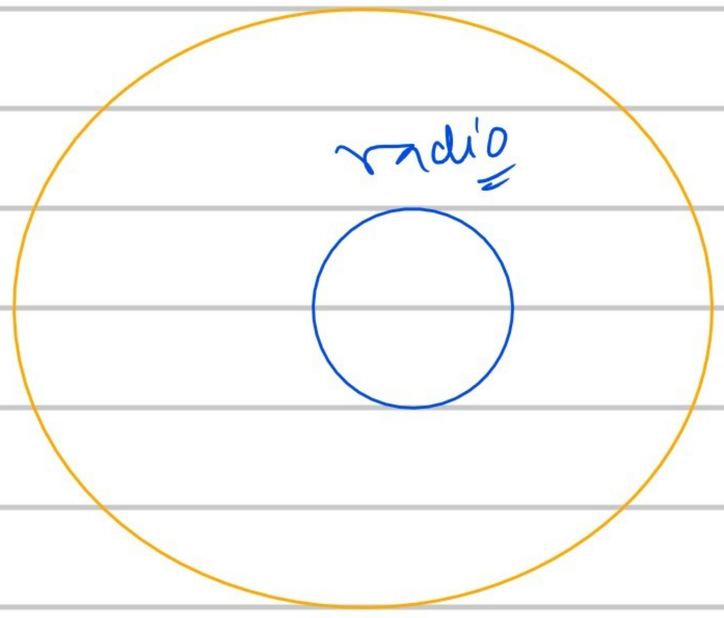
variables → placeholder for data whose value can change

- Primitive
- Object with HAS A relationship

* Primitive Variables
  ↳ are those variables which are declared using datatypes

byte boolean char short int long float double

* Object Variables
  ↳ are those variables which are declared using other class objects




```
class Radio {
  int volume;
  public void play(){
    S.o.p("R --> "+volume);
  }
}
```

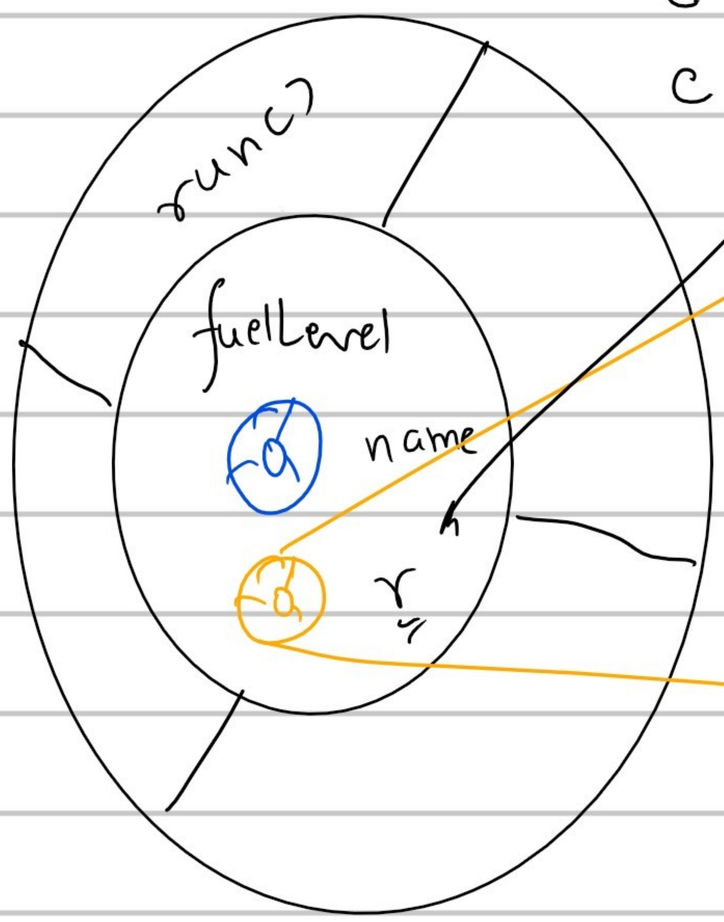
```
class Car {
  String name;
  int fuelLevel;
  Radio r=new Radio();

  public void run(){

  }
}
```

Car c = new Car();

c.fuelLevel;

c.r.play();

c.r.volume;

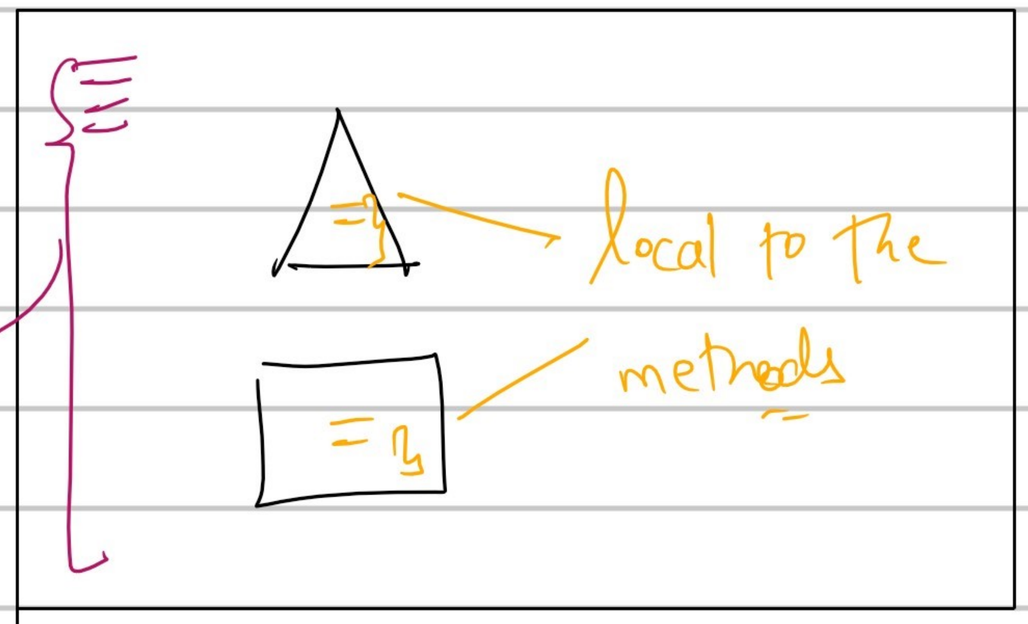


Based on the position of the variables,

① Instance

② Local

instance	local
→ those variables which are declared outside the constructor and/or methods	those variables which are declared inside the constructor and methods, parameters are local
Scope	
→ available throughout the class.	→ available in the constructor/method where it is declared
Declared	
→ int i; If a variable is declared Pen p; In instance area p is assigned to its default value.	→ if left declared → will result in error



instance

Local & Instance Variables can have same name

```
class Circle {

    int radius;

    Circle(int radius) {
        this.radius = radius;
    }
    public void setRadius(int radius) {
        this.radius = radius;
    }
}
```

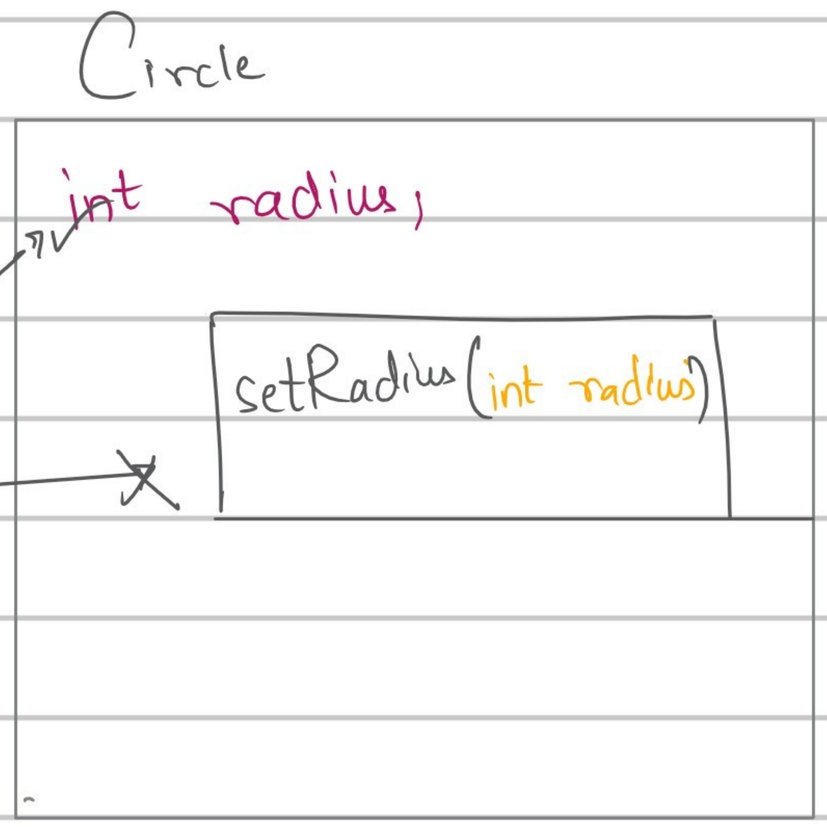
instance

local

ambiguity

local

```
Circle c = new Circle();

c.radius;
```



this keyword
↳ point to an instance of same class

---

static
- variable
- method
- block

modifier

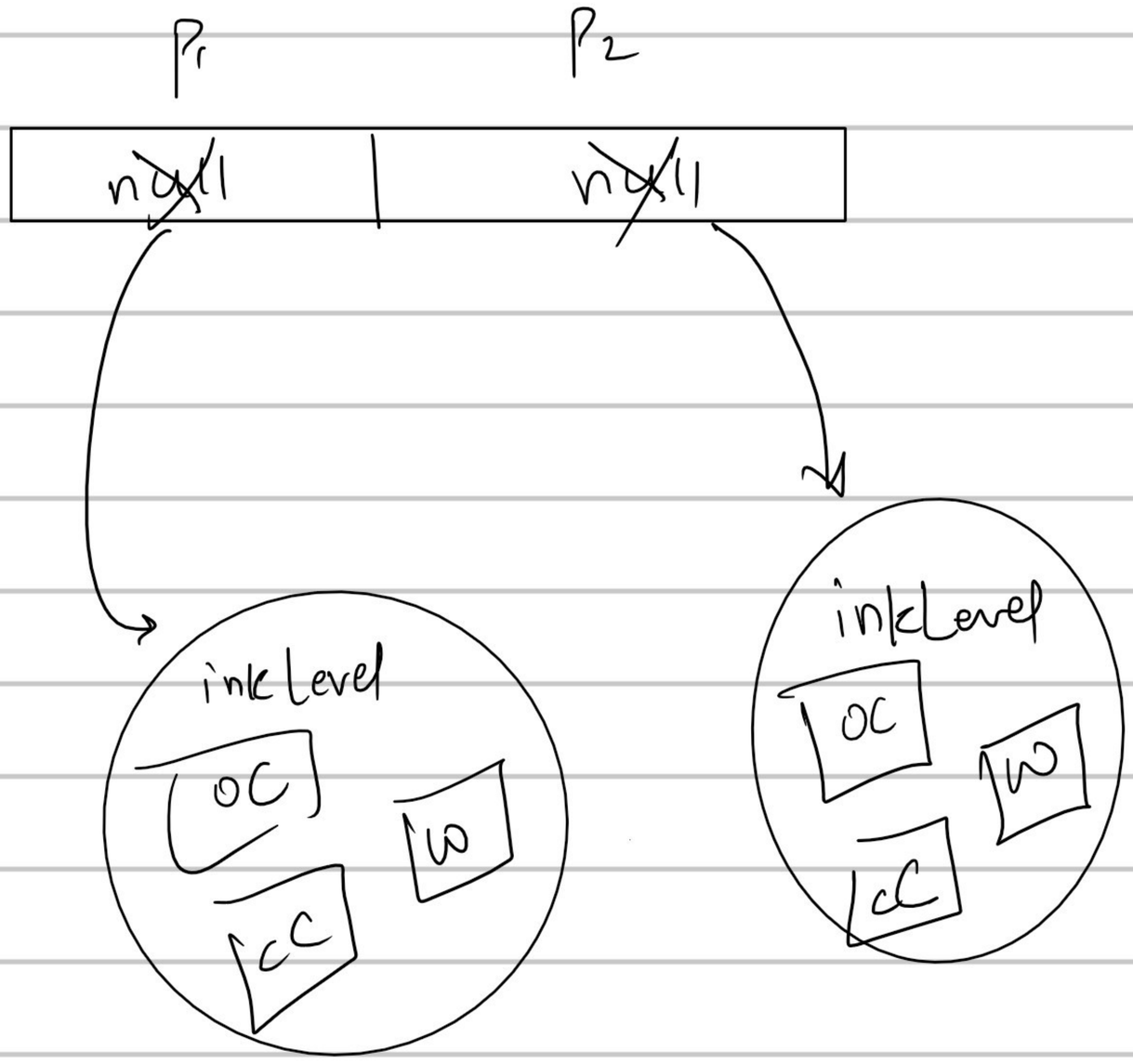
Pen p; — class load

p = new Pen(); — object load

Time

* Anything which is of nature static

1. is loaded when the class is loaded.
2. have one copy for all the instances
3. can be accessed directly using the class-name

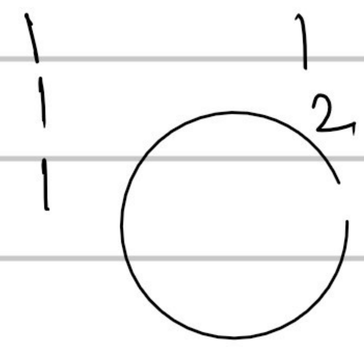


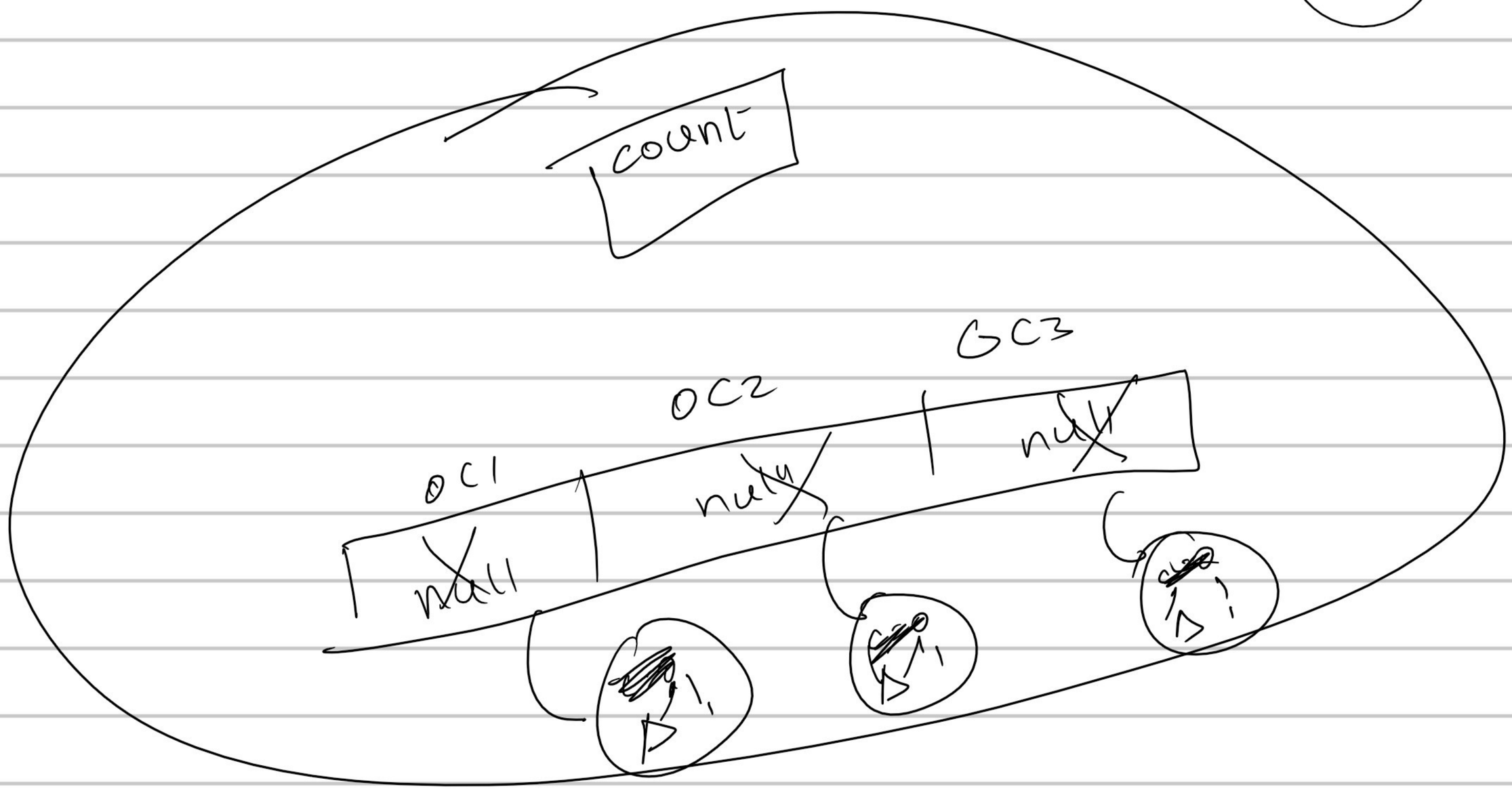

```
Pen p1, p2;

p1 = new Pen();

p2 = new Pen();
```

```
class ObjectCounter {

static int count = 0;

ObjectCounter() {

    count ++;

}

}
```

```
ObjectCounter oc1, oc2, oc3;

oc1 = new ——— ();

oc2 = new ——— ()

oc3 = new ——— ();
```

? count



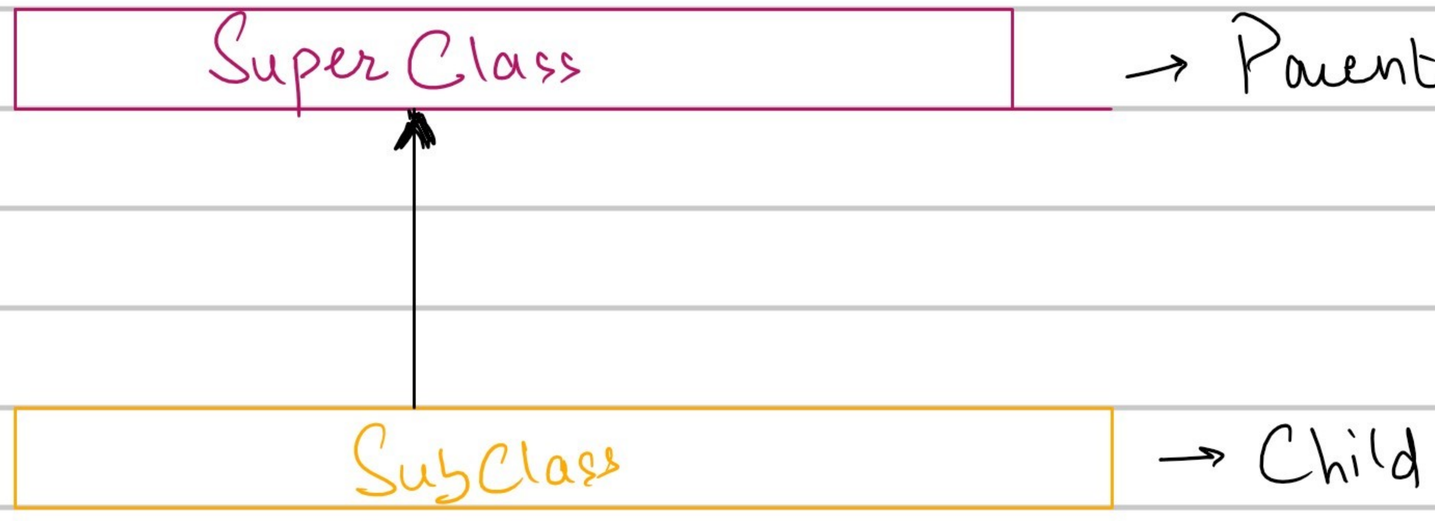


Inheritance

→ Process through which object of sub-child acquires the state & behaviors of it's super class

Parent → Super Class

Child → Sub Class



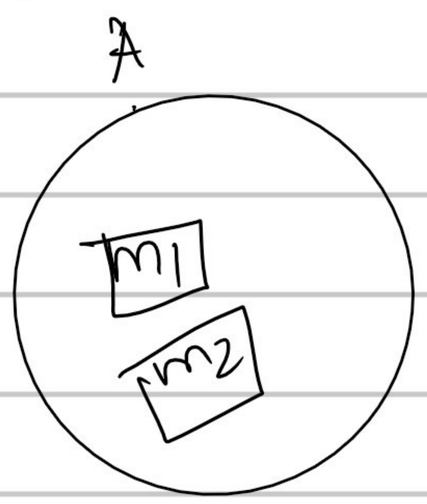
* How inheritance happens *

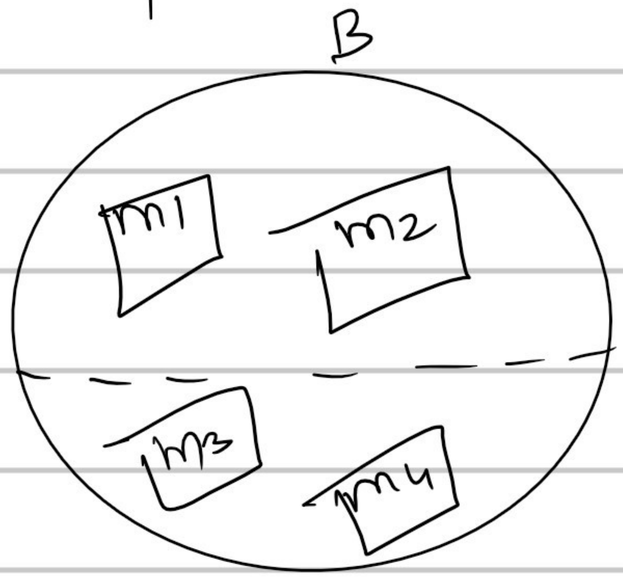
→ extends

```
class Parent {

}
```

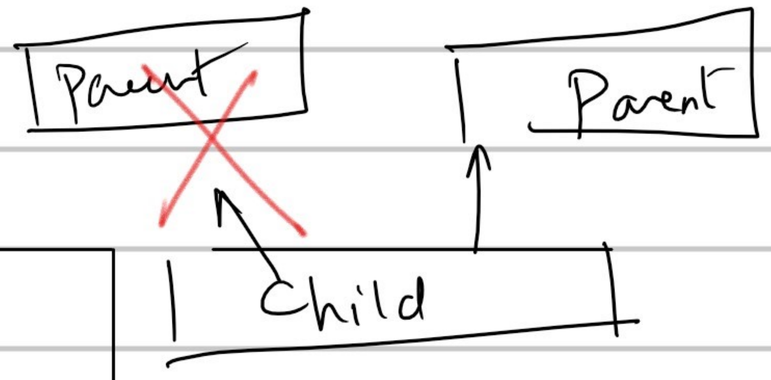
```
class Child extends Parent
{

}
```

When Inheritance happens all the data from super class comes down to sub-class



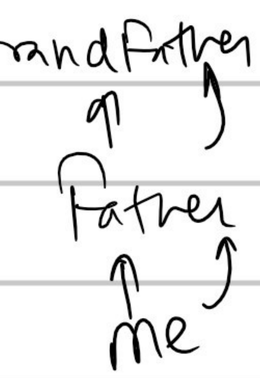


* Java doesnot support multiple inheritance.

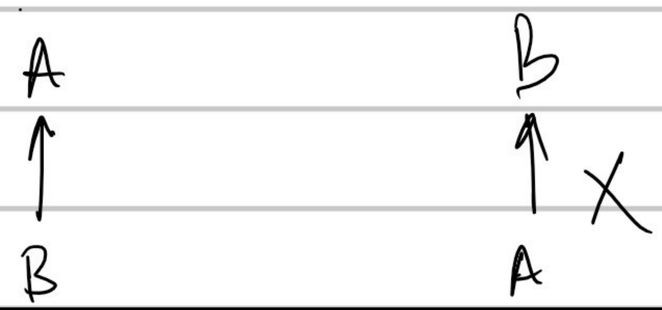


→ At any given point in time; a Java class will only have one direct super-class.

→ Java support multi-level inheritance



→ Java do not support cyclic inheritance



* A java class atleast & atmost have one super-class

```
class X {
}
```

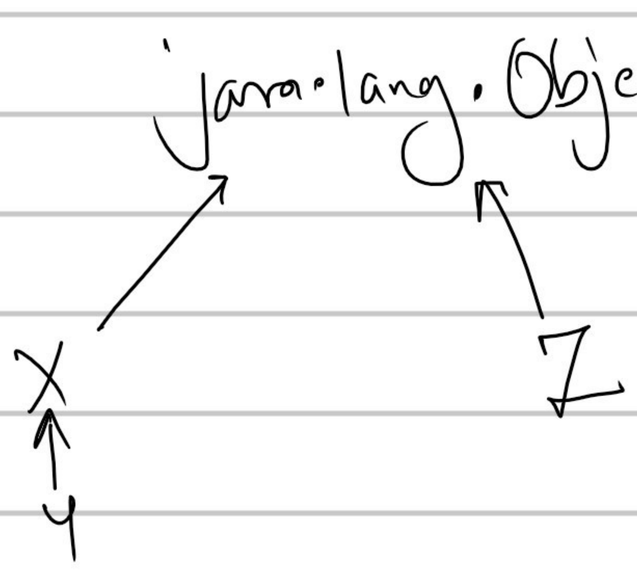
```
class Y extends X {
}
```

```
class Z {
}
```

* Adam of Java *

java.lang.Object → alpha-class

created one super-class for all; but sub-class for none.



→ provides us with methods;

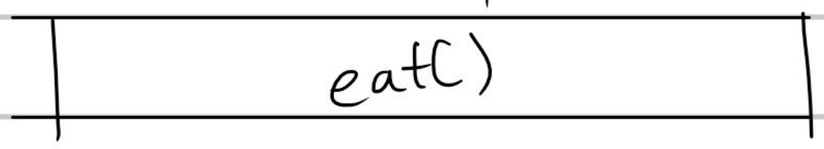
* Method Overriding *

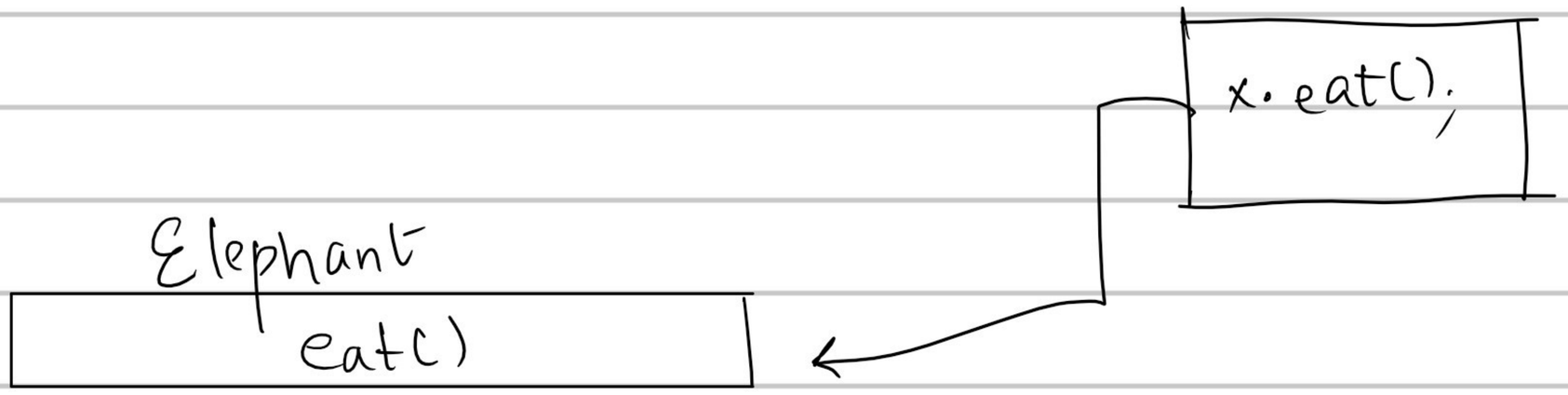
When a sub-class & super_class have method with same signature ; we say sub-class have overriden the super class method

name of method, parameter

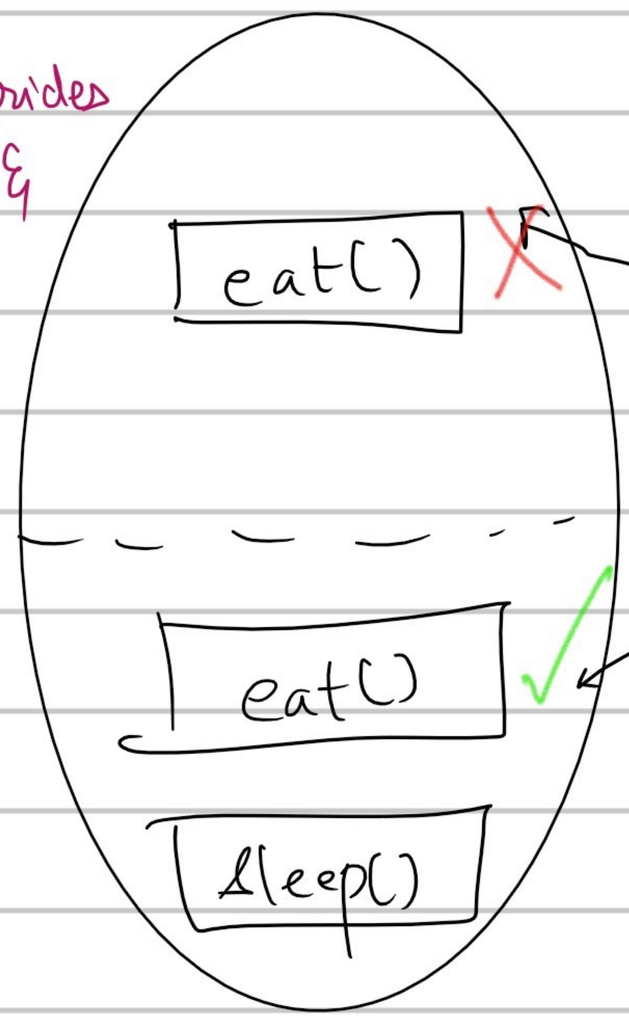
```
class Animal {
    public void eat() {
        ——— ("Eat → Animal");
    }
}
```

```
class Elephant extends Animal {
    public void sleep() {
    }
    public void eat() {
        ——— ("Eat of Elephant");
    }
}
```





* When a sub-class overrides a super-class method & if that method is invoked using Sub-class object; always the sub-class version of the method is invoked



Elephant e = new Elephant()

e.eat();

Super → refers to immediate super class

Super keyword can be used for accessing

1. variables in super class
2. methods in super class
3. constructor in super-class

Object of a super class can be made to point at it's sub-class instance.

Animal a = new Elephant();

Generalization

```
public void treat(Animal a ){
    a.eat();
}
```

someObj.treat(e);

When an object of super-class points at it sub-class instance, it access only overriden methods; other methods are sliced.

When super-class object pointing to sub-class instance invokes the overriden method. Always the sub-class version of method is invoked.

* final keyword *

① variable

② method

③ class

④ object

final variable

```
final int MAX = 100;
```

↳ it's a convention to write final variable totally in upper-case.

→ Once the value is assigned; it cannot be re-assigned

final method

* A method marked as final; cannot be overriden.

```
class A {

public final void m1(){

}
}
```

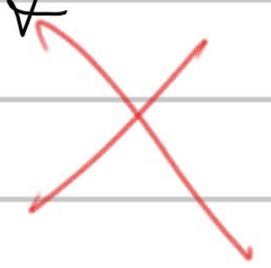
```
clan B extends A {

pubric void m1(){

}
}
```

final class

A class marked as final; cannot be sub-classed

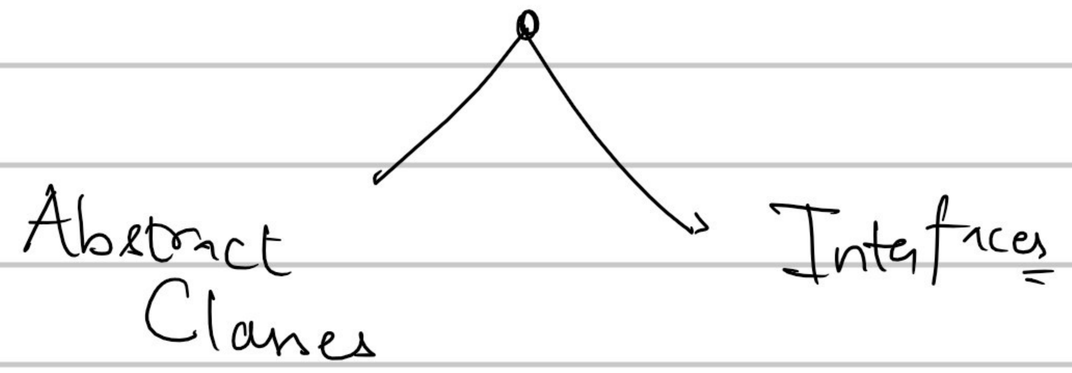
```
final class Y {

}
```

```
clan Z extends Y {

}
```

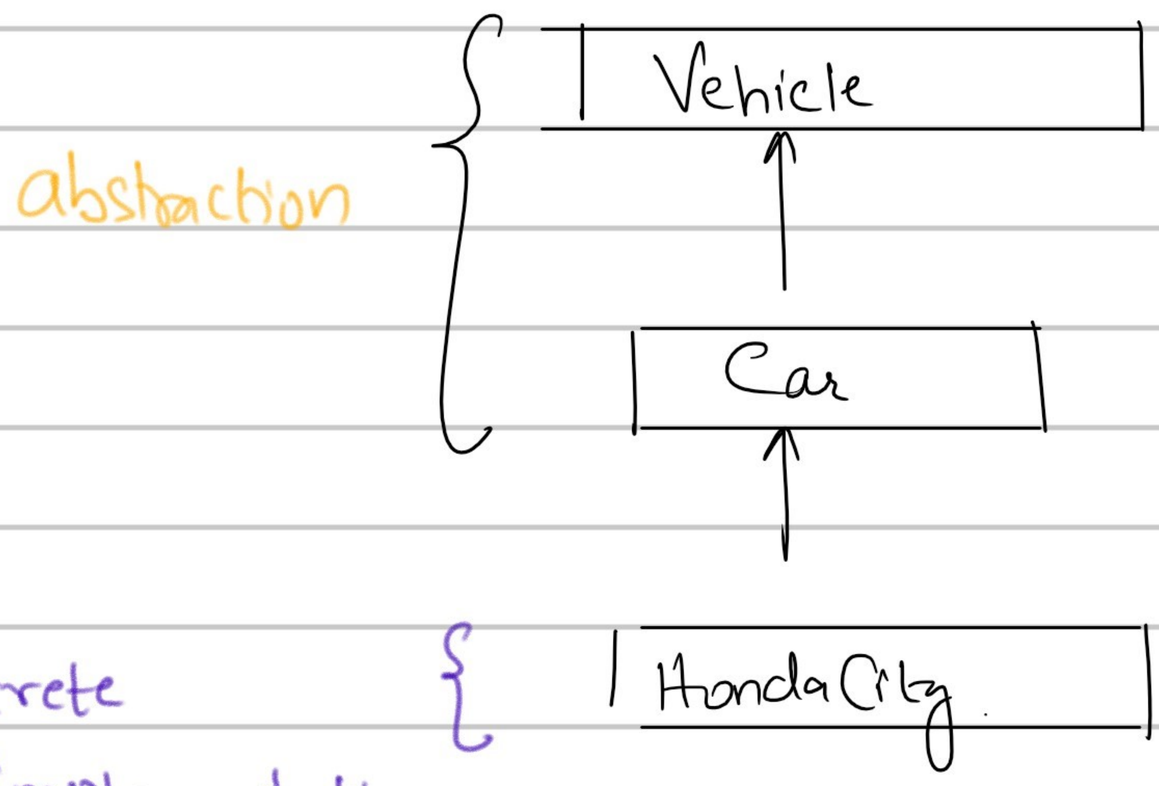


* Deferred Implementation *

↳ declare now
implement later



* Abstract Class *



* Abstract class is used when you want ~~to~~ your sub-class to override certain methods compulsorily

* To make a class abstract; we need to mark the class with keyword 'abstract'.

```
abstract class Car {

}
```

* Abstract Methods *

→ are those methods which are marked & declared as abstract

```
public abstract void start();

public abstract void stop();
```

() { } → defining

(); → declaring

```
abstract class Car {

public abstract void start();

Public abstract void stop();

Public void m1(){

}

}
```

An abstract class can have abstract as well as non-abstract methods.

We can create an object of abstract class; but not instantiate it.

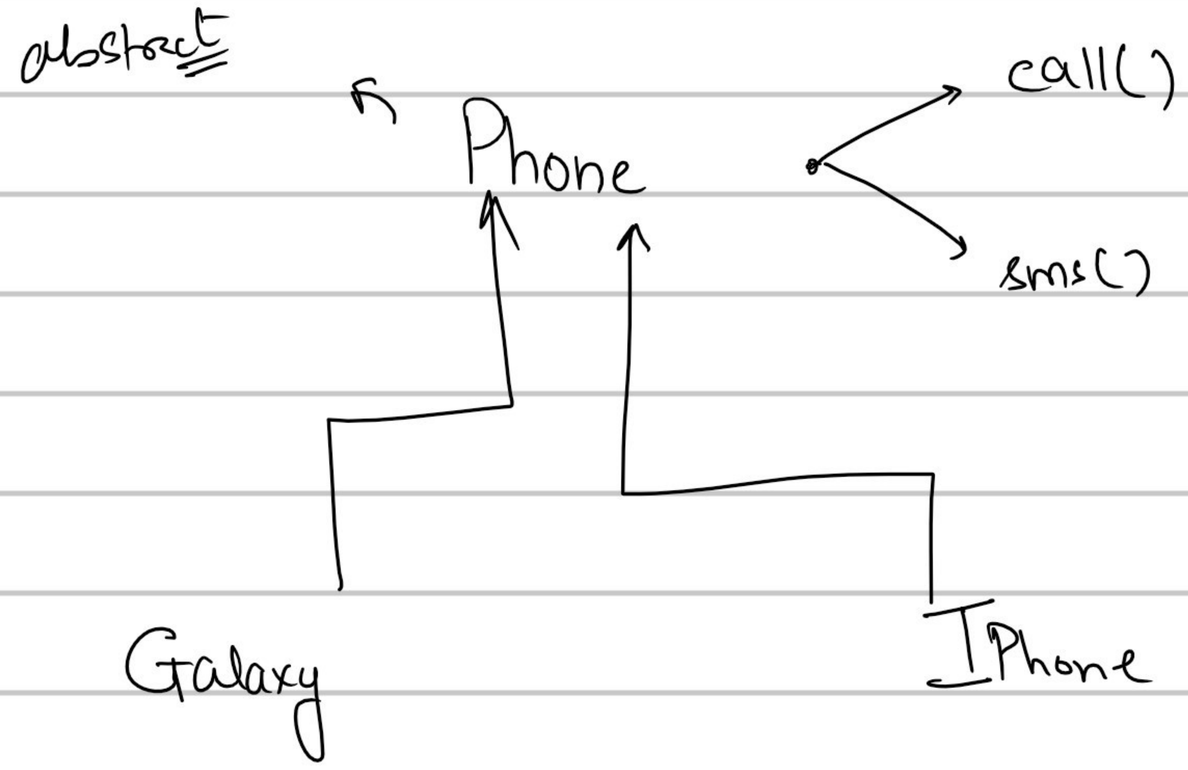
Car c; ✓

c = new Car(); X

Relation between an abstract super class & non-abstract sub class.

* When a non-abstract sub class extends an abstract super class, it has to override all the abstract methods; failing to which, the sub-class won't compile.

```
class HondaCity extends Ca {

    start()

    stop()

}
```



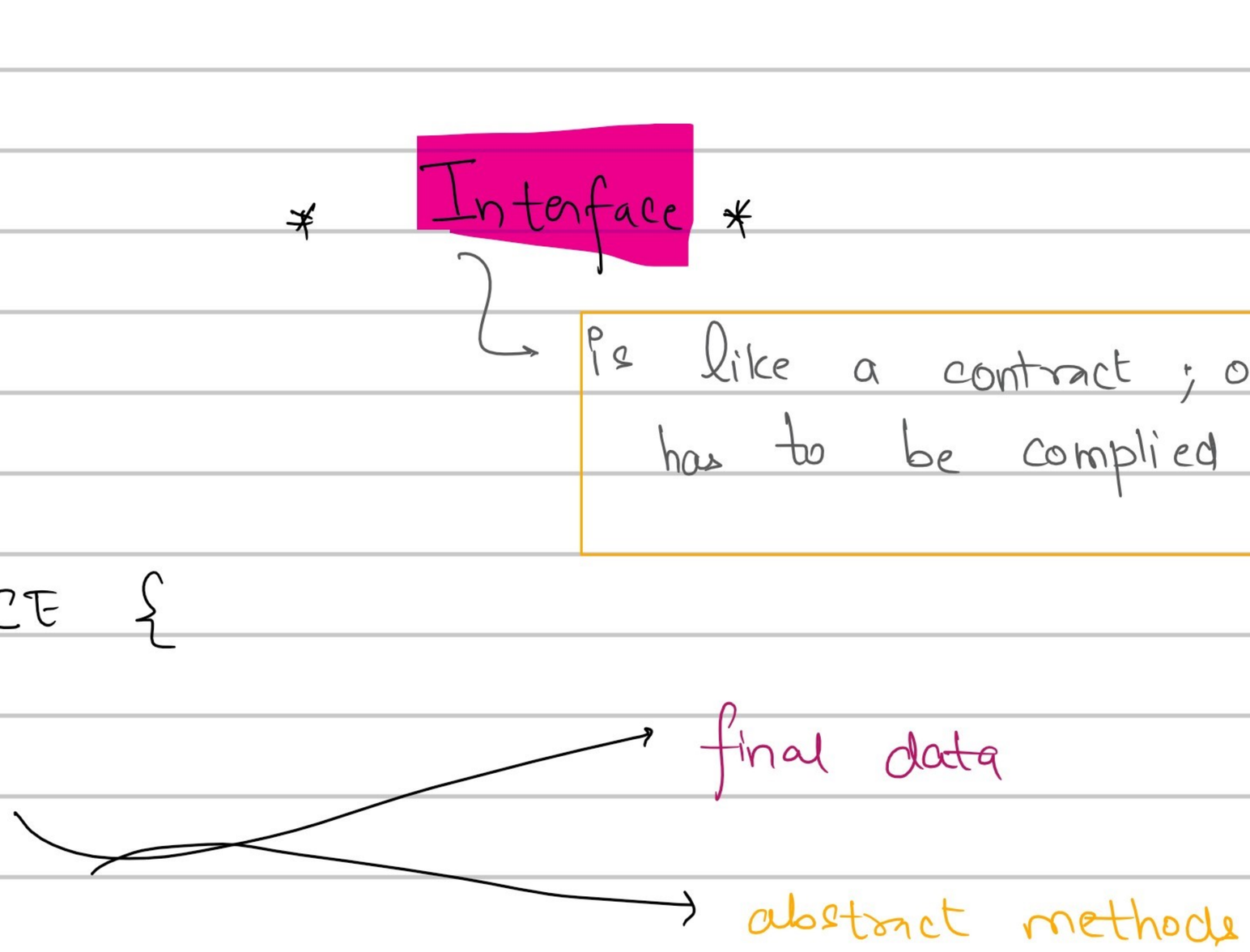
{ Certified
  European

p1()
p2()
p3()

* Interface *

Is like a contract ; once signed has to be complied with.

Interface CE {



}

* An interface cannot have non-abstract methods

interface I1 {	class A implements I1 {
Public abstract void p1();	Public void p1(){
	=
	}
}	}

* When a class implements an interface, it should override all the abstract methods, failing to which the class won't compile..

* When you need to inherit & use interface at same time; use extends first & then implements

```
class IPhone extends Phone implements CE {

}
```

* A class can implement more than one interface.

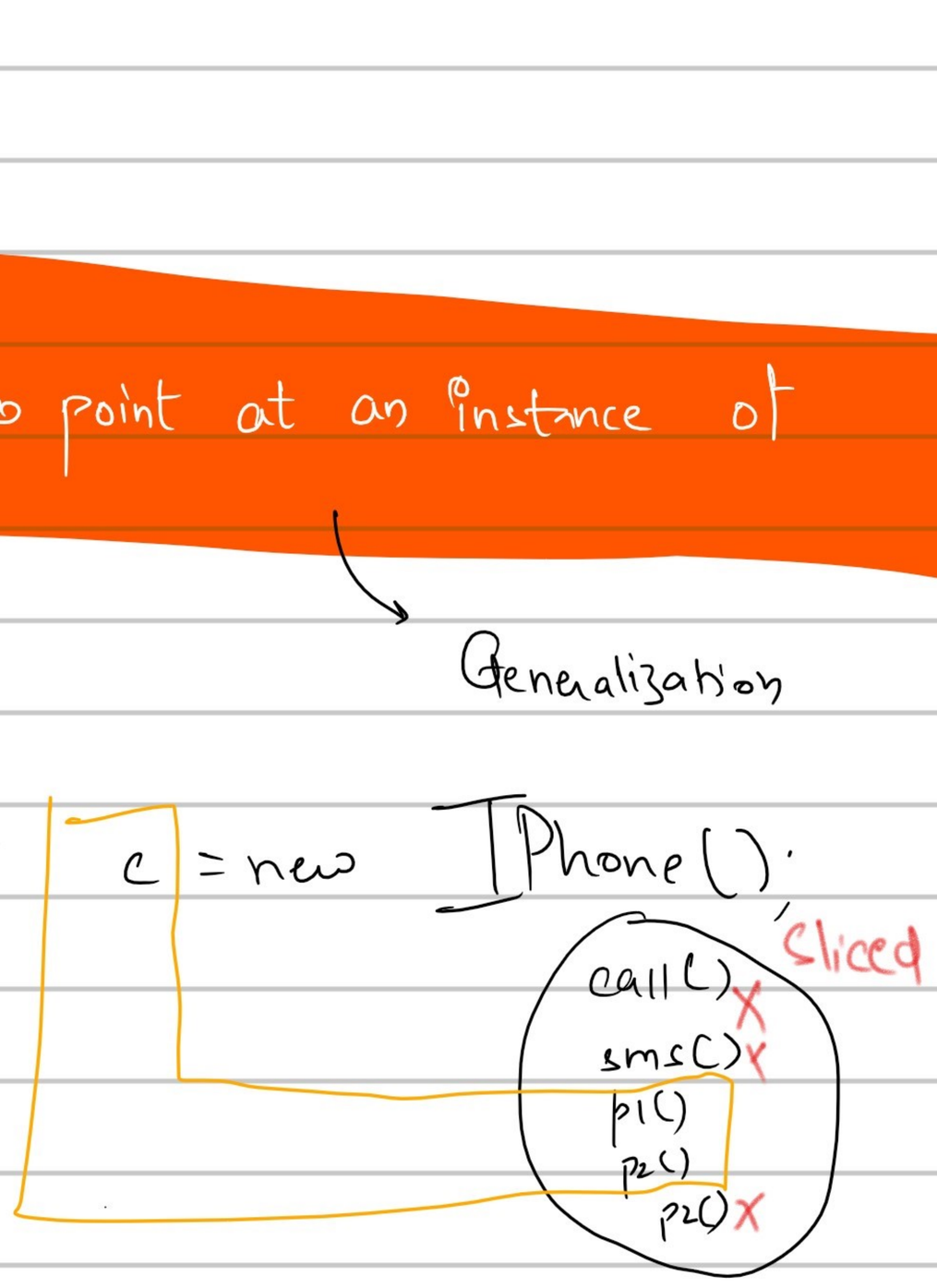
```
class IPhone extends Phone implements CE, ISO {

}
```

Object of an Interface can be made to point at an instance of a class which implements it.

Generalization

```
public void sell(CE c){

}
```

```
CE c = new IPhone();
```



Interface / Abstract Class

A class which implements/ extends

* what to do

* How to do

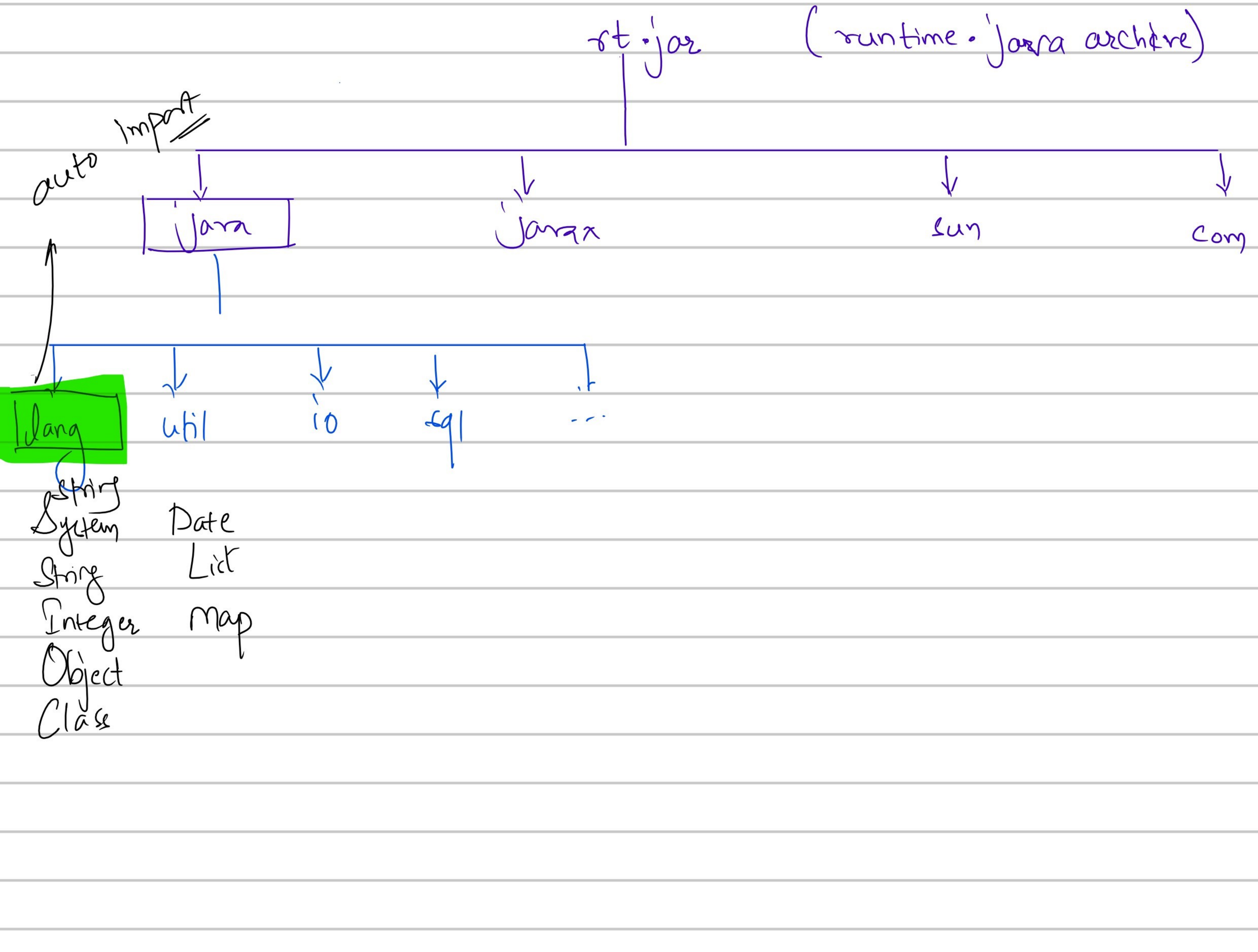
---

* Packages * → collection of classes & interfaces in some directorial hierarchy

in built packages

creating our own packages.

In-built Java Library Packages



* A class in same directory can access another class *

TodaysDate.java

```
import java.util.*;
class TodaysDate {

        main ( ){

  A a= new A();

  Date dt =new Date();

  System.out.println (dt);

 }
}
```

A.java

```
class A {

public void m1(){

}
}
```

* How does a Java class recognize another Java class *

1. Tries to search for the class in same directory.
2. Tries to search in java.lang package
3. Tries to search in imported package

* Creating our own package *

URL → www.sc.com

sc.com

Package → com.sc

↳ Creating segregation in project.

Layered Architecture

com.sc.rpg.loans

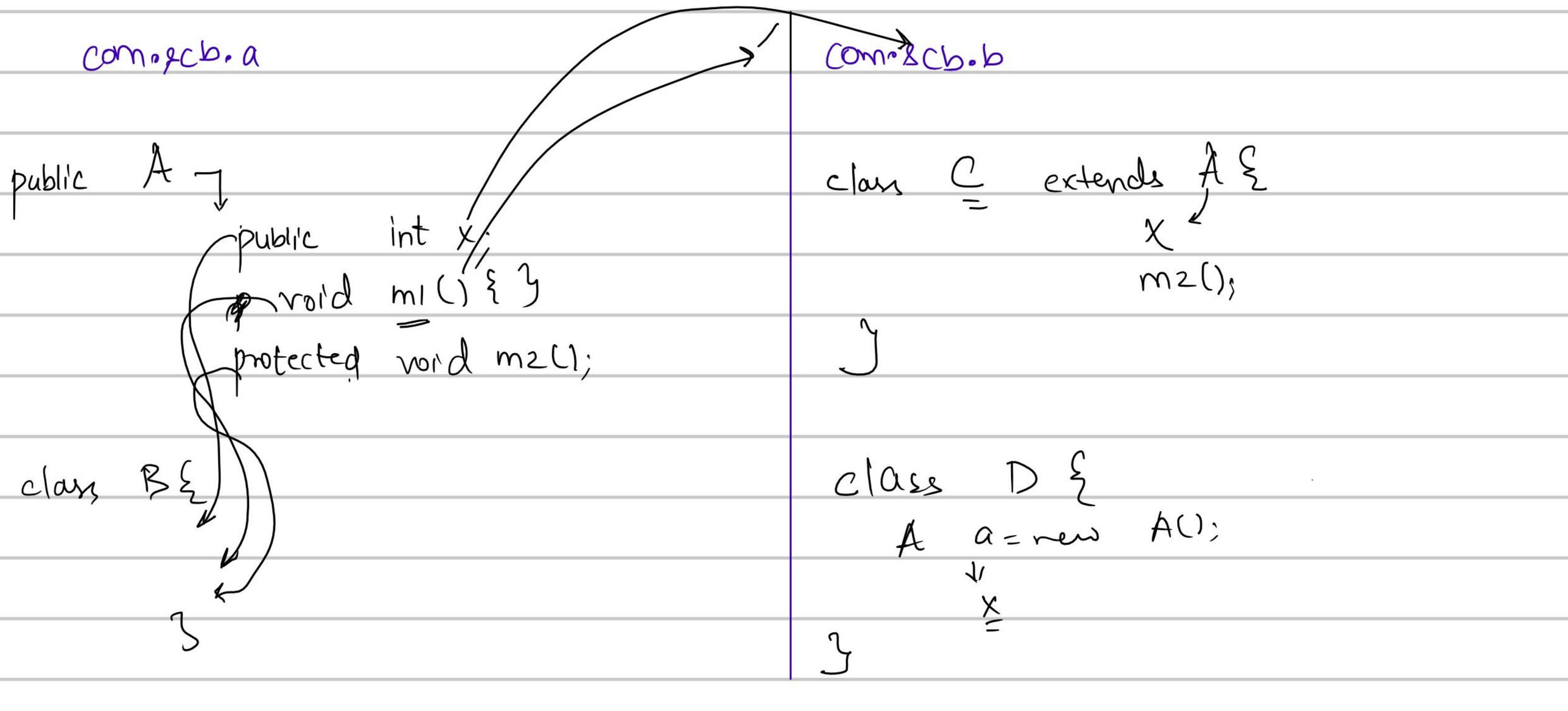
- beans → Customer.java
- ui → CustomerUI.java
- helper
- service → CustomerService.java
- Controller → CustomerController.java

* Access Specifier *

* public → available everywhere.

if no access specifier is defined. uses default.

* private → only available in same class
* default → only available in same package
* protected → default + inheritance



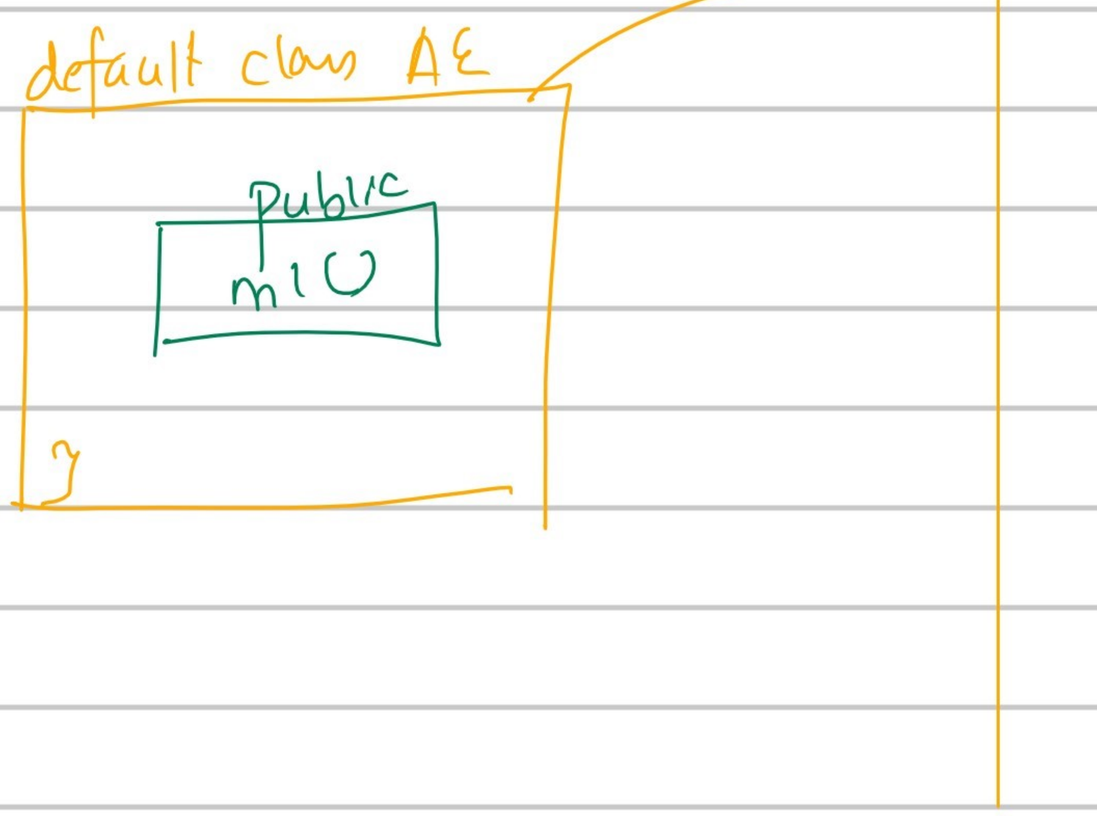
* Quick facts about access specifiers *

① A class can only be marked default or public.

② A variable/method can be marked
- public
- private
- protected
- default.

③ A public method in default class will not be available outside the package

com.scb.q



④ A default method in public class won't be available outside the package

Access Specifier	Same Class	Other class in same package	Sub-class in other package	Other class in other package
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

String

Pen p;
p=new Pen();

Collection of characters.

Java.lang.String.

String Constant Pool

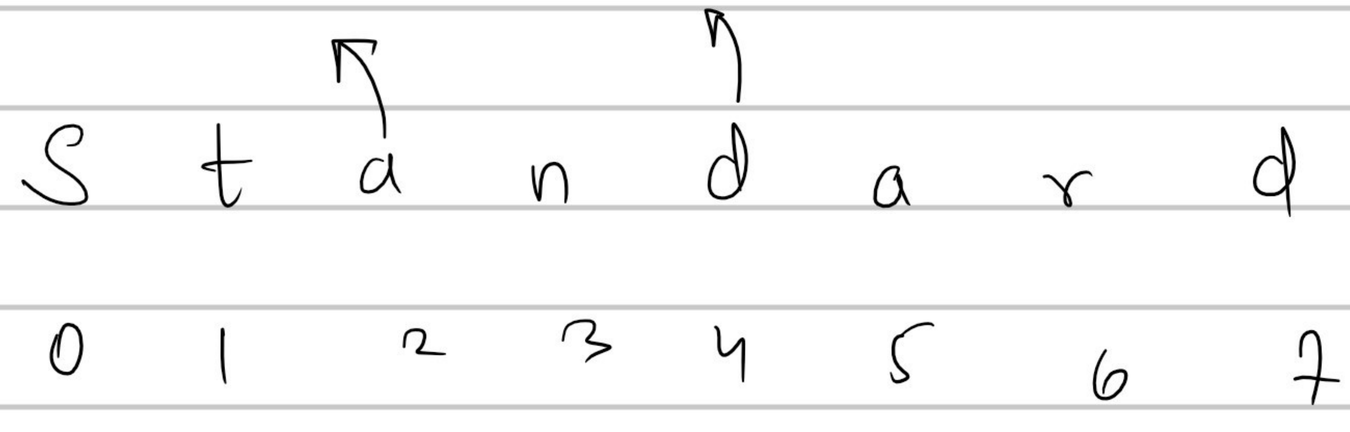
String literal

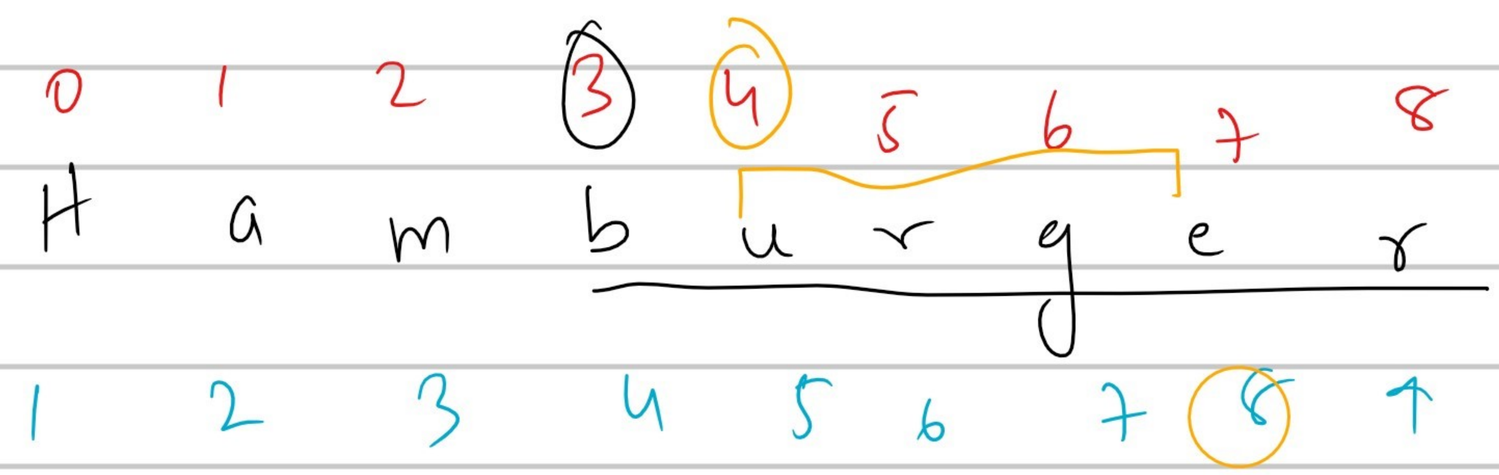
String s="Tom";

String get's special treatment from JVM

String s=new String("Tom");

String Object





String are of nature Immutable

String s="——";

String s1 = s.concat("——");

StringBuffer

Exception Handling.

Exceptions are unexpected erroneous events occuring at runtime breaking normal flow of the code.

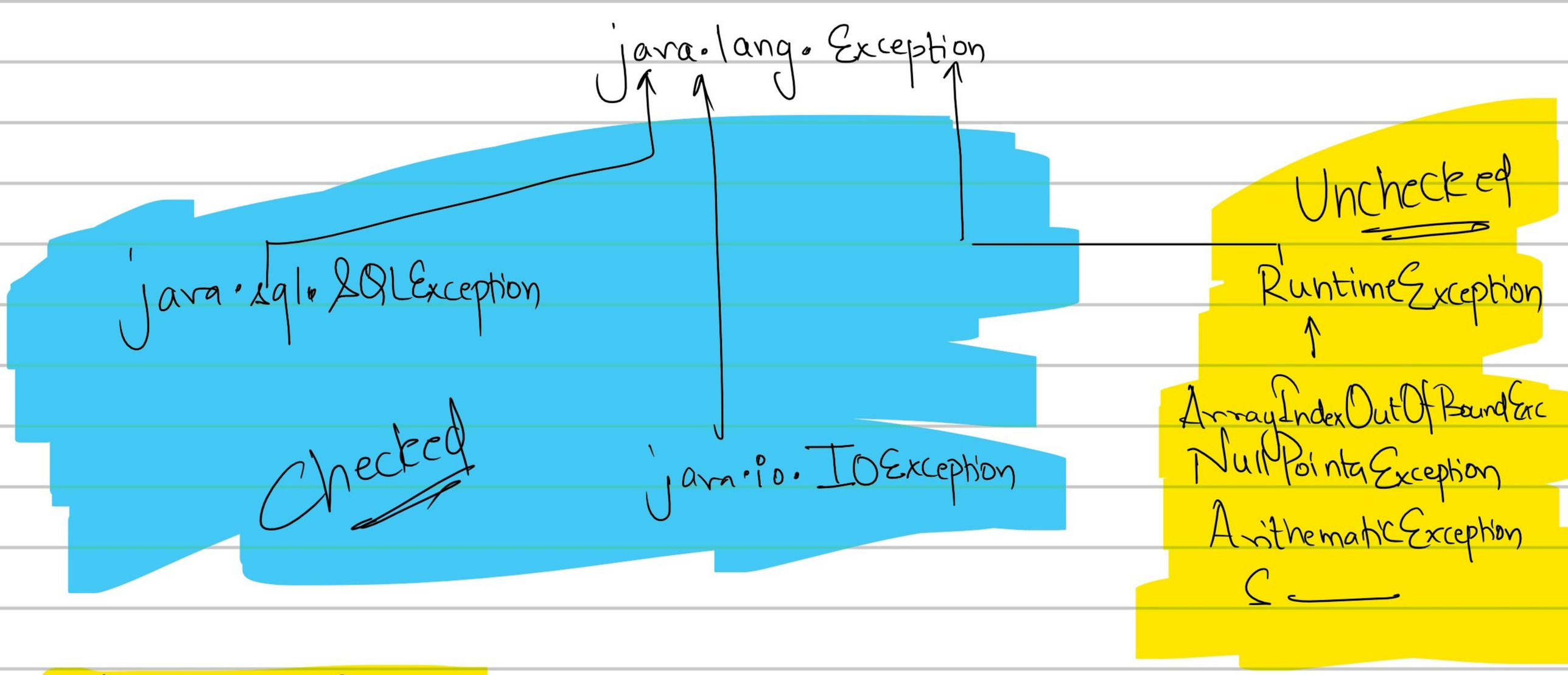
Exceptions can be handled in two ways.

(1) try... catch

(2) declaring that exception

(1) try... catch

```
try {
    // Any piece of code which is prone to throw an exception goes in try block
}
catch (Exception e){
    // When exception happens what steps are to be taken.
}
```



Unchecked Exception → Any subclass of RuntimeException including RuntimeException -

Checked Exception → Any subclass of 'java.lang.Exception except for RuntimeException

UNCHECKED EXCEPTION are NOT REQUIRED To BE HANDLED OR THROWN

CHECKED EXCEPTION REQUIRES To BE HANDLED / THROWN

```
Animal a = new Elephant();
Exception e = new SQLException();
```

```
try {

}
catch(Exception e){

}
```

```
try{
    db
    txt
}
catch(SQLException e){
}
catch(IOException e){
}
catch(Exception e){
}
```

Exception Message

e.getMessage() → String → what went wrong.

e.printStackTrace() → print everything from top to bottom

```
try{

}
catch(          ){

}
finally{

}
```

house keeping job

in case of succesful run or exception

* Collection API *

Array

→ fixed type

→ no convinience methods

→ way to store the data.

→ allows you to store hetrogeneous data

→ tons of convinience methods

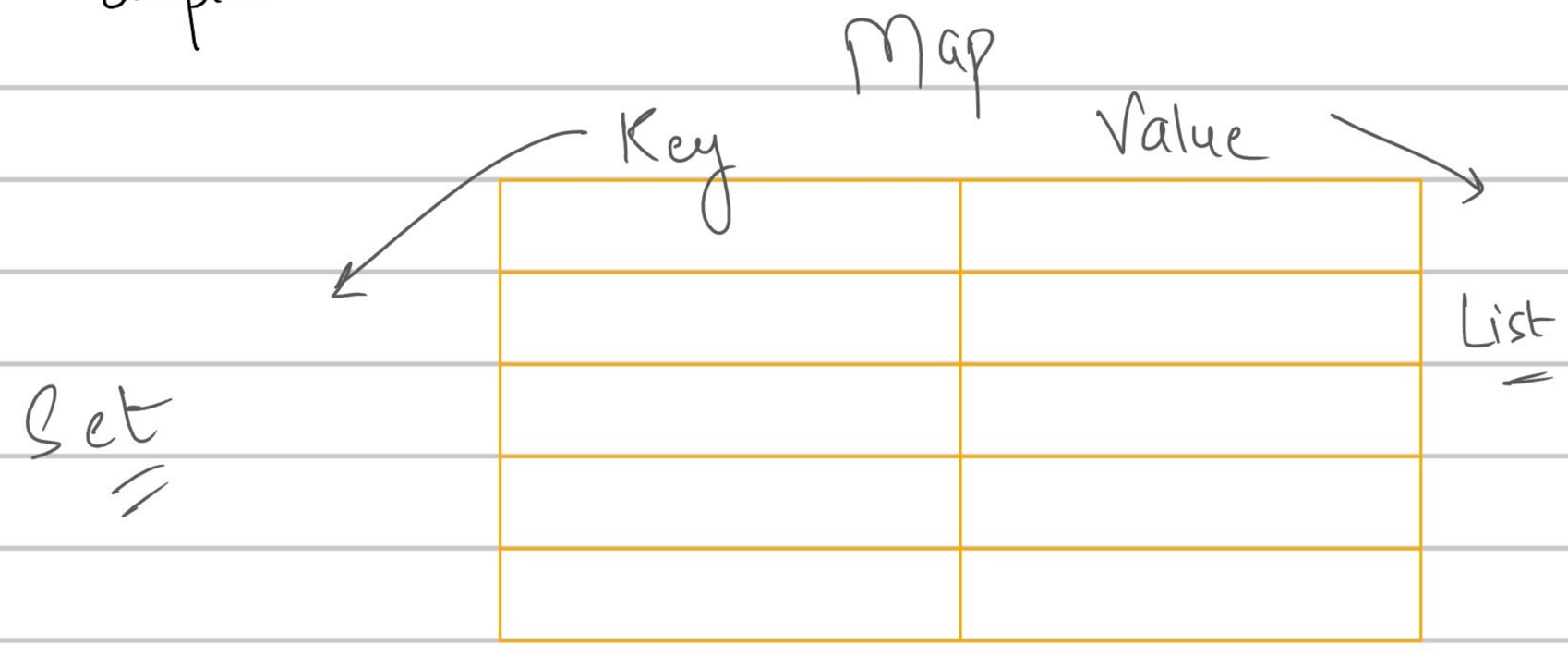
→ java.util

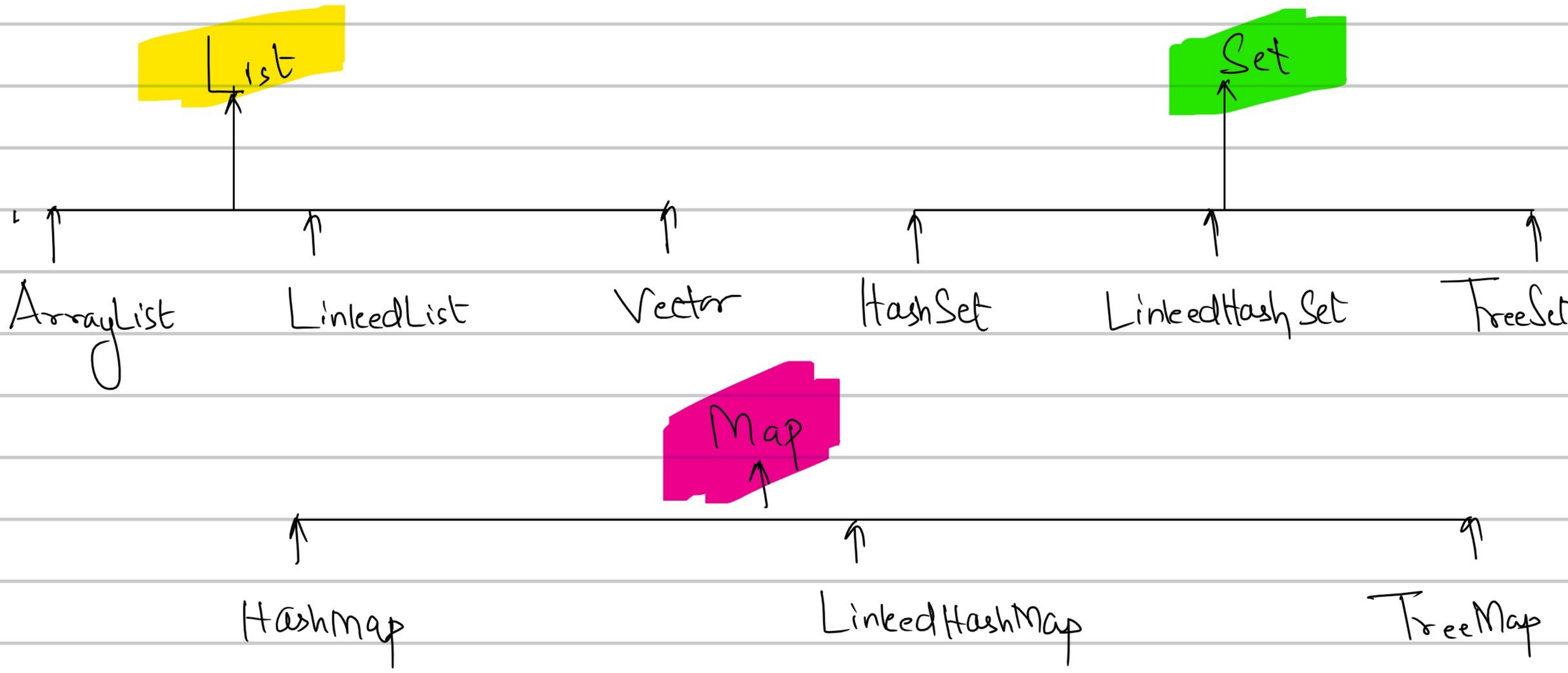
→ store duplicates

uses an array internally

→ does not allows duplicates

→ FIFO





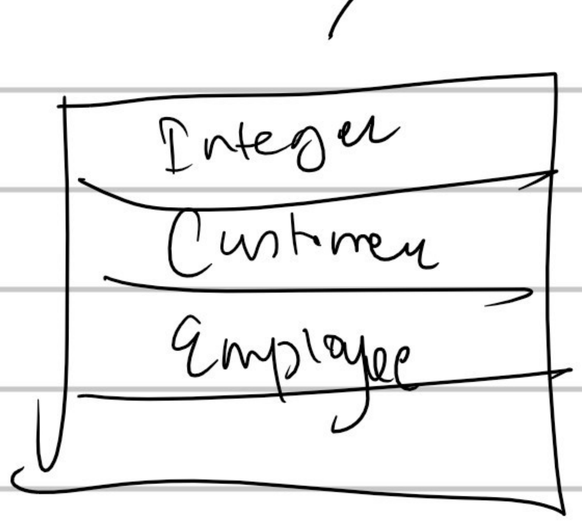
Hash → No order is maintained.
→ faster → hash algorithm

Linked → Order of insertion is maintained

Tree → Sorted by Natural key

```
ArrayList list = new ArrayList();
```

↳ can store hetrogeneous data



Generics

```
ArrayList<String> list = new ArrayList<String>();
ArrayList<int> list = ✗ ;
```

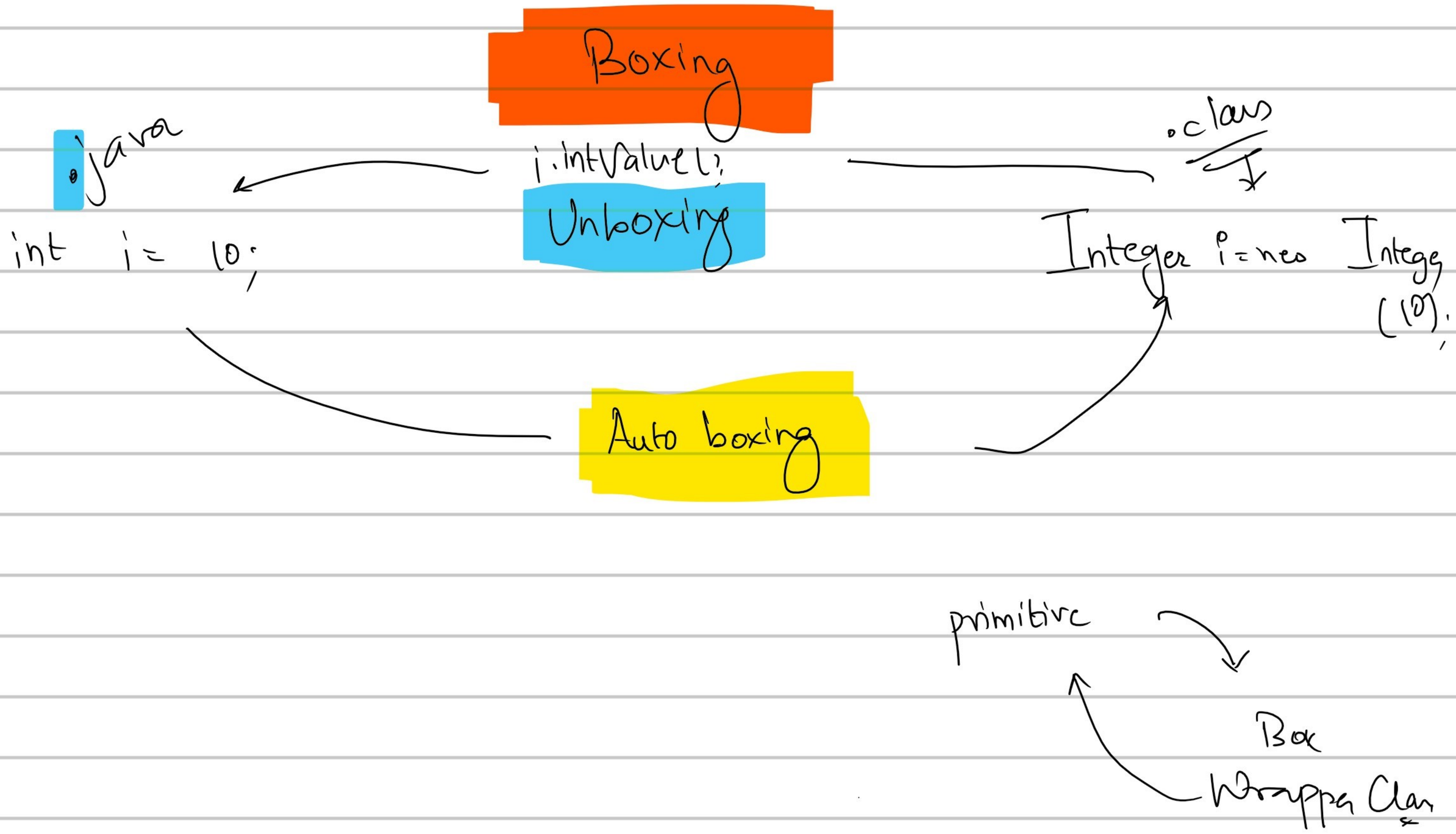
Wrapper

java.lang

byte	Byte
boolean	Boolean
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

ArrayList<Integer> list = new ArrayList<Integer>();

↑ only hold integers



* Fetching the data *

* for each loop

```
for ( Type element : collection) {
    S ⟵ (name);
}
```

```
for ( String name : namesList) {
    S ⟵ (name);
}
```

Iterator → available for List, Set, Map, Queue

↳ part of collection

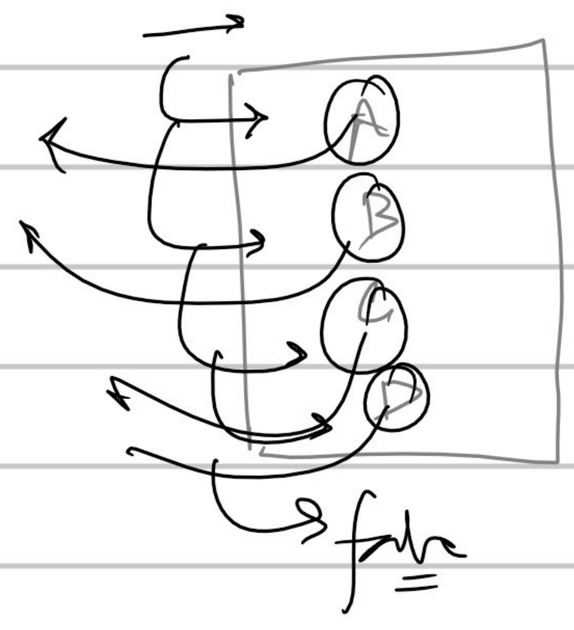
List namesList = ⟶

Iterator i = namesList.iterator();

i.hasNext() → to check if collection have the element

i.next() → ref to the element

```
while ( i.hasNext() ) {
    String name = i.next();
}
```



USE CASE

com.sc.rpg.

Ra: Main.java

model → entities

↳ Customer.java

Controller

↳ CoustomerController.java

Services

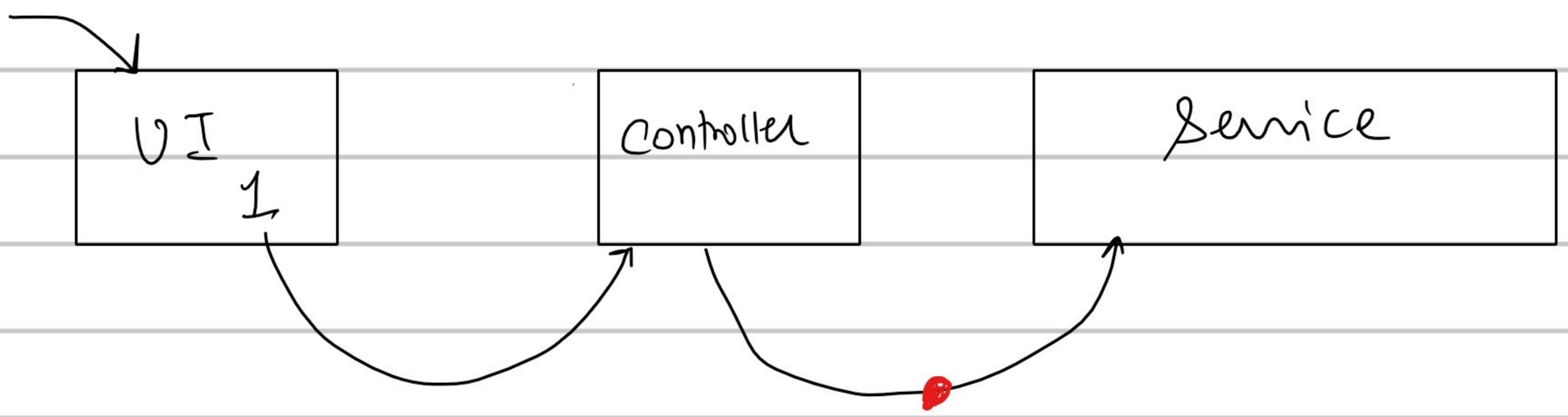
I ↳ CustomerService.java

C ↳ CustomerSericeListImpl

UI

↳

CustomeUI.java



JDBC → Java Database Connectivity.

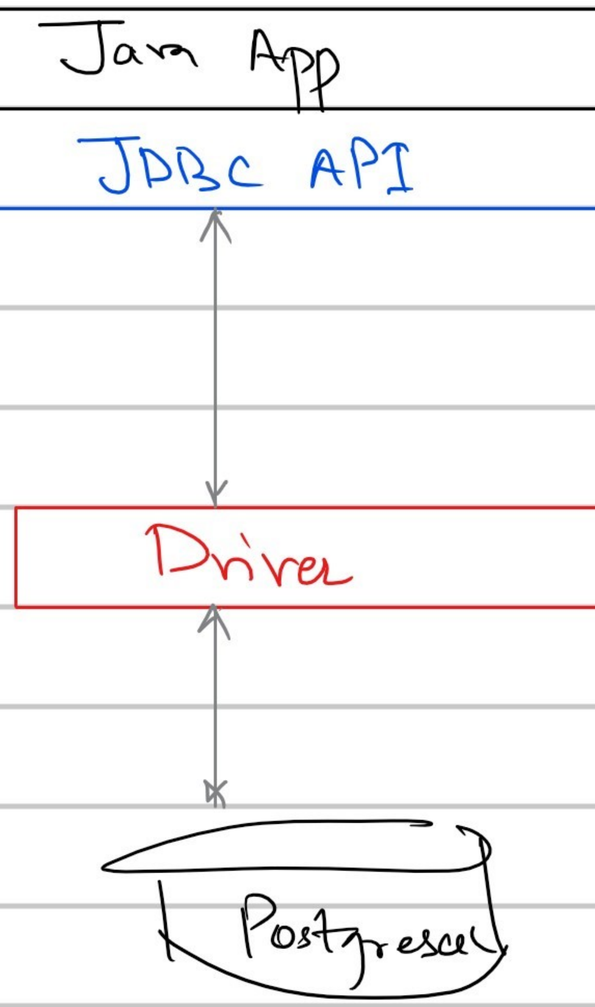
→ API which helps to connect a Java application to the backend ↓ RDBMS.

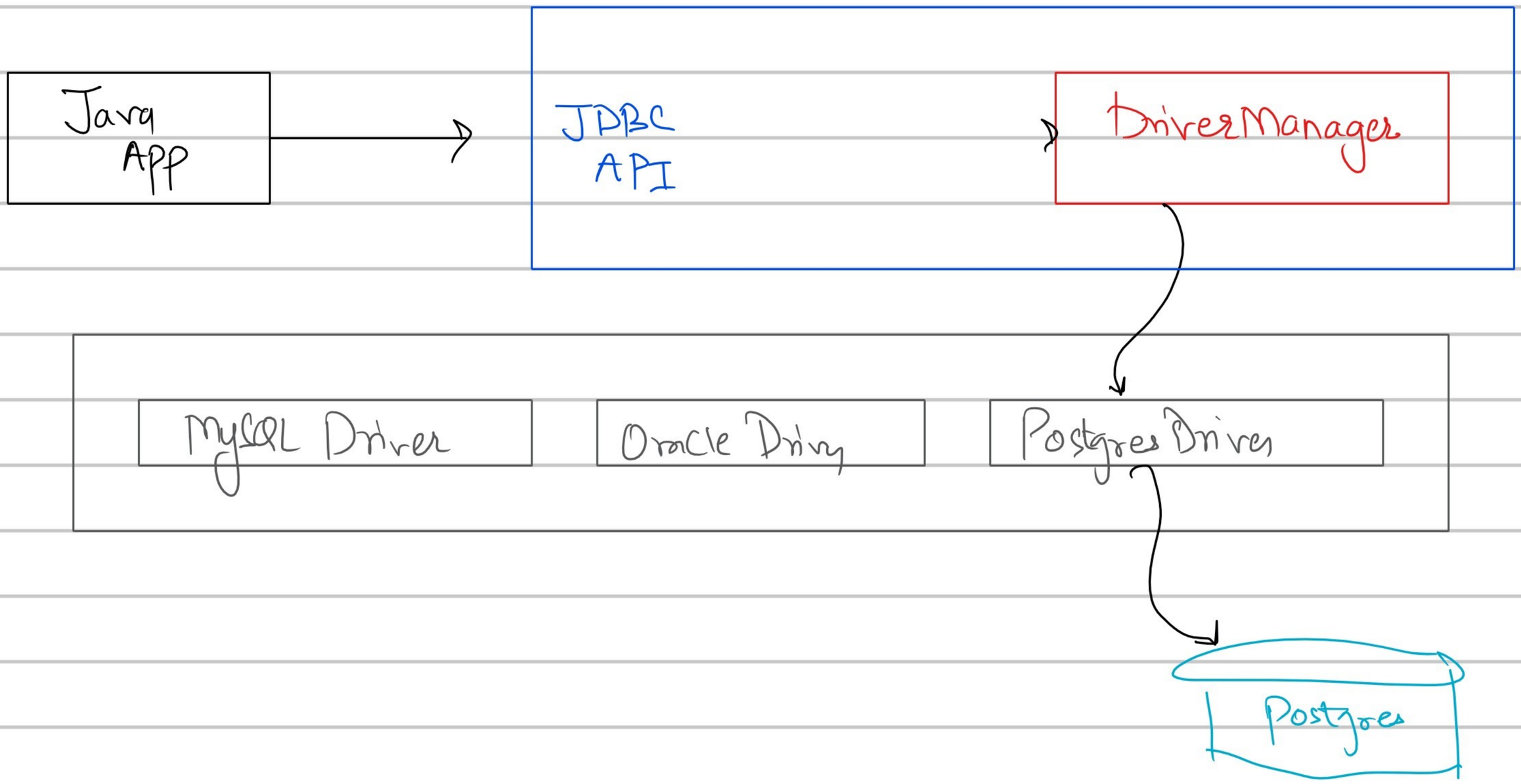
→ JDBC can connect to any underlying RDBMS using appropriate driver.

java.sql

Components of JDBC API

- Driver
- DriverManager ♥
- → Connection
- Statement
- → ResultSet
- → ResultSetMetaData





Steps for Connecting JDBC with Database

Step 1: Import the required packages

→ All the classes & interfaces required to work with JDBC API are in java.sql package

```
import java.sql.*;
```

Step 2: Load & Register the Driver

→ Driver (provided by RDBMS) has to be loaded explicitly by calling forName() in 'java.lang.Class

```
Class.forName("org.postgresql.Driver");
        ("com.mysql.jdbc.Driver")
```

↳ The loaded Driver get registered with DriverManager implicitly.

Step 3: Establish Connection

* Connection is like a session with database.

– To acquire this connection, we need to call getConnection() of Driver by using JDBC URL

* JDBC URL *

rdbms specific

database specific

base-protocol : sub-protocol : sub-name

base-protocol → jdbc

sub-protocol → postgresql
mysql
oracle
sqlserver

sub-name → ip + port + name of db

```
String url = "jdbc:postgresql://localhost:5432/scb-rpg-2022";
    String username = "postgres";
    String password = "admin";

Connection con = DriverManager.getConnection(url,user,password);
```