14 Sep, 22
8:00 am

OBJECT ORIENTED PROGRAMMING

named groups of properties and functions.

Object is an instance of a class

class is blueprint of object.

class → logical construct

Object → physical reality // occupying space in memory

State - value

Identity - where value stored in memory

Behaviour - effect of datatype

using (.) operator we access

↓

it links instance variables

↓

variables which object

↓

Seperator in (java)

new operator –

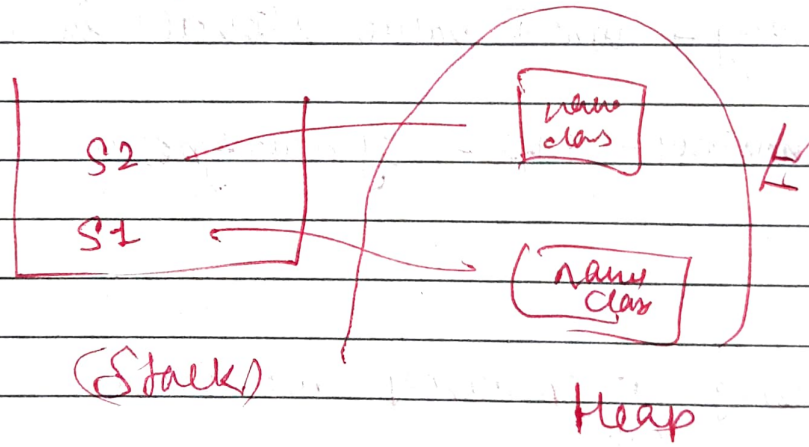
class student { } student S1 = new Student

↓

reference to the

for objects its null if not declared.

dynamically allocate memory and returns reference to it at runtime



Student S1 = new Student();

compile time

↓

errors

bytecode

runtime

all checks done

code running

dynamic memory allocation

variables store in stack memory points objects in heap memory

primitive → default type 0, 0.0

complex → null,

DIAGRAM

462042

Students

rno = 0

name = null

marks = 0.0

new

K. null.

is null no. is exist there?

print default value

kunal.rno.

(rno. in kunal)

= new Students();

↓

constructor

↓

called / desired ~~at time~~ when object is created.

Student kunal = new Student (13, "Kunal", 112

Special type of function inside class

new Student() → default type class

constructor is a special type of function its automatically called when object is created

Student () {
    (constructor.)
}

Student kunal = new Student();
Student Rahul = new Student("Rahul");

this keyword

this.name
kunal.name

Student(String name)
this.name

replaced by the reference variable

this in python → replace with self

similar to self in python

r.no = r.no ✗ inside a constructor

this.rno = r.no ✓

Constructor Overloading

Student() {
}

Student(String name, int age) {
}

name = new S();

ghaip = new S(name, 18);

this.rno = other.rno

this → depends which object we referencing to are referencing to.

Student arpit = new Student (17, "Arpit", 89.f);

new this will be replaced by arpit

Student (int age, String name, float marks)

this.rno = rno.

this.arpit.name

this.marks.

cant put Kunal in it

this → name of reference variable

May don't have return type

this keyword
call one empty constructor

```
Student() {
    this(13, "Kunal Person", 100)
}
```

↓

```
Student(int age, String name, float marks) {
}
```

internally

```
new Student(13, "Person1", 99);
```

In Java primitives are not ~~stored~~ treated as Objects and stored in Stack

to make Java efficient

(extracting from heap is complex)

```
Student S1 = new Student();
Student S2 = S1;
```

(does not allocate memory)

point to the same object

copy of my variable

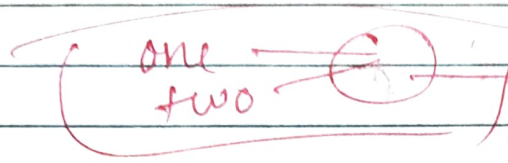
changes made by one object is been reflected by other object they to

Student one = new student();
student two = one();

one.name = "Soni"

Soni (two.name)

o/p → Soni. ✓



→ make changes in any variable changes reflected

int a = 10;

Integer num = 45; ↔ num as object

a. ⇒ num. ⇒ & functions for Wrapper class

pass by value in java ✓

~~temp~~

a = 10
b = 20;

void swap(Integer a, Integer b)

still not swap!

Integer wrap

static void swap(int a, int b)
int temp = a;
a = b;
b = temp;

scope of (a,b) ↔ remains here

o/p = (10,20) (a,b) same no change

final works with primitives only doesn't holds true for objects

Interview - final class

(FINAL) - Keyword

present content to be modified

like const

final int INCREASE = 21

↓ capitalised

↓

can't be modified by convention.

final int bonus = 2;

final int SALARY = 2000

SALARY = 1000 // error

class A {

final int num; // num to be initialise while declaring

can't change it.

final guarantee only with primitives immutability holds true

final Student Kunal = new Student()

Kunal.name = "new name"

Kunal = other object (X)

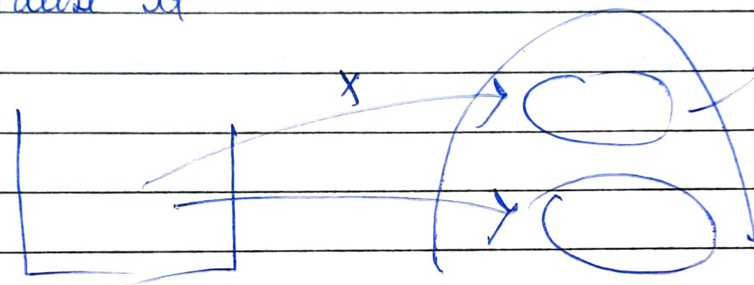
F1 GTT

modify it but can't make other ref variable point same object

modify it but you can't make ref variable point ~~other~~ same object

(when a non primitive is final, you can't reinitialise it)

Chris MT409



Shelby Monaco
Koenigsegg
Cobra
Agera RS.

K

You can't specify actions when the gc takes any object using the finalize()

similar to destructor.

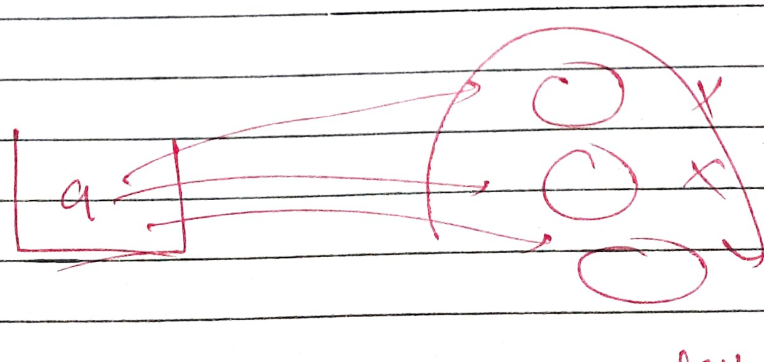
McLaren
F1 Koenigsegg
CC8S
Jaguar
D type

Bugatti
EB110

@ Override.

protected void finalise() ~~throws~~ Throwable

sout ("Object is destroyed");



ctrl + click

OOPs 2: KUNAL KUSHWAHA

@Override

when we say To

A s1 = new A("Kunal");

System.out.println(s1);

↳ O/P (A@23.....)
classname and hashcode

but, why this ?

when we call this println method
it uses a toString function.

but in class A we have not declared it
so it considers default values i.e
(classname + hashcode)

To overcome this we use Override method.

PACKAGES IN JAVA.

main.java
main.java ✗ error.

Package allows us to is a folder.

com.folder.kunal

when java file use.

Packages used to create multiple java files
~~if~~ groups of java classes
it allows. interfaces
stored in hierarchical method

Categorize, so they can be easily maintained

Kushwaha.com

com.kushwaha

Opposite to Company can
use

→

←

how does java know which file we are trying to access.

import statement scans... ~~mess~~ funct

JRE

only ~~public~~ files that are public

java.util.ArrayList;

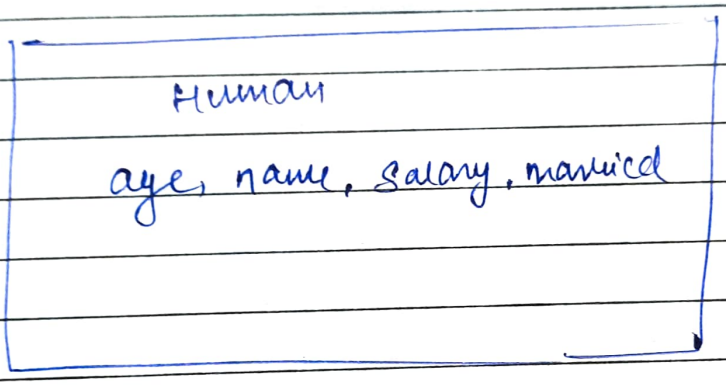
if same folder you don't have to import it

Static

common to all classes
not created to object?

why use this?
↳ just use
classname.population
variable

also can be accessed
with ~~obj~~. object



without referencing to object
not dependent to object

Why main declared static? → Belong to class not to object

You should be able to run main
in order to create objects

Non Static

you cannot access any static stuff, ~~belongs to an object~~ without referencing to an object

Can use this in static ✗

```
static printmsg ( ) {
    sout( this.name)   //error.
}
```

// demo to show initialisation of static variables

```
staticInit {

public class StaticBlock {

    static int a = 4;
    static int b;

static { // will only run once when first obj is created
    System.out.print ("I'm staticBlock");
    b = a * 5;

psvm ( ) {
    StaticBlock obj = new StaticBlock
}
```

static { } ⟹

first about static stuffs are run

Inner classes can be static

Outside class can't be declared

```
static class A {      //error.
    class B { }
}
```

since static stuffs independence of objects resolved during compile time, class level.

static int salary; // same salary everyone
static boolean married // all married not married

salary 1000000 everyone ✓
salary 100000 everyone
slo everyone
salary everyone
predictable everyone
everyone
o
o o

System.out.println( ) ← method
class
final static variable

using keyword.

(.) binds instance variables and methods with the reference variables.

Print stream class.

Array → class.
.toString
method

value of (x)

.toString.

@Override → Annotations

classname@Hashvalue

default toString method

if no toString found in class.

ONLY ONE INSTANCE ALLOWED ??

SINGLETON CLASS

Allow one object creation

Not allow use constructor

Make private (constructor)

if accessing with classname declare it as static.

static

if (instance == null)

instance = new Singleton();

return

singleton Obj1
Obj2
Obj3

all 3 ref variables are pointing to just one object.

20 Sep 22 10:00 am.

SUPER KEYWORD — used to called parent class constructor.

OOP-3 -

PRINCIPLES OF OBJECT ORIENTED PROGRAMMING

INHERITANCE —

Parents — Base class

Children — Child class

use properties

child class inheriting properties of base class

class child extends class Base

Species

Human Fish

super(l, b, h); // what is this //

super keyword used to call the parent class constructor used to initialise values present in parent class.

anything private can be accessed in the same file

Box (l, w, h) — obj

Box weight — X obj ✓

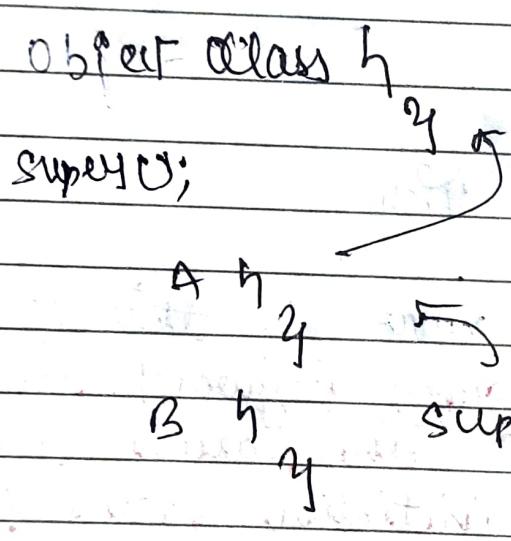
Type of reference variables determine what members can be accessed.

Box box5 = new BoxWeight(5,6,7);

BoxWeight box5 = Box(5,6,7); // error

SUPER KEYWORD

TWO USES CASES —



super(l,b,w);

super.width; // access variable methods of parent class

above classes do not have info about idea pass up the line

childclass care about parent superclass yourself

lower class.

there are many variables in both parent and classes
you are given access to variables that are in the ref types i.e BoxWeight

hence you should have access to weight variable
this also means that the ones you are trying to access should be instantiated

but here, when the obj ref is of type parent class, how will you call the constructor

this is why error.

```
BoxWeight box6 = new Box (2,3,4);
System.out.println (box6);   // box box
```

type of reference variable not type of the object

August 7:30PM 20Sep22, Tuesday.
Tuesday August

TYPES OF INHERITANCE — E

SINGLE

Box
Cube

one class extends one class.

MULTILEVEL

Box
+
Box weight
+
Box Price

(HIT AND TRIAL)
Set of Rules.

Above class do not have a idea of bottom class but bottom class has the idea of upper class
→ Inheritance wise

MULTIPLE INHERITANCE -

one class extends more than 1 classes

A
n=5

B
n=10

C

Ambiguity

C obj = new C();
C.n //?

if two of the classes same property or method java will get confused which one to pick

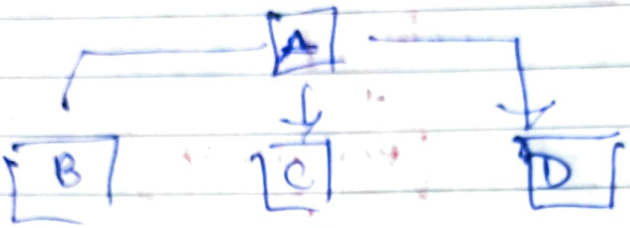
! allowed in java

↓ LATER

By Interfaces we implement

what class does not. Why?

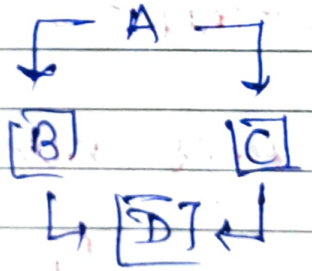
4. THE HIERARCHIAL INHERITANCE



5. HYBRID INHERITANCE

Not in Java.

Check Interfaces chapter



public class A extends class A

A class cannot be its superclass.

POLYMORPHISM → Greek Word.

MANY + ways to represent (form)

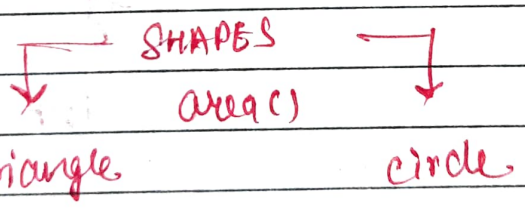
OBL → it polymorphism X
OOP

class Shape {

public circle extends shape,

— " — Triangle - " —

— " — rectangle — " —



shape
Shapes = new Shapes();

Act of representing

HOW IT WORKS INTERNALLY ??

TECHNICAL DETAILS →

TYPES →

COMPILE TIME / STATIC &

~~RUN TIME / R~~

/operator overloading
~~In Java its not supported~~

achieved by method overloading.

return type, order, types
no. of argument

Why its called so.

Java determines which & name to call.
At compile time

int → double

if

int a, double b

(4, 5); ✓

(4.33, 5) ✗

RUN TIME / DYNAMIC POLYMORPHISM

achieved by method overriding

// when object of child class is made then it overrides the method in the parent class.

@Override // annotation

↓

used for check purpose

@Override → not Overridden

Parent obj = new Child();

here which method will be called depends on.

This is → upcasting known as.

↳ How overriding works?

works

HOW JAVA KNOWS THIS

By default in java: Dynamic method dispatch. ✗ happen with static

Object class & Object. ↳ toString method

extends → every class inherits

Topic: Overriding, Inheritance

final keyword is used.
method, class, variable

LATE BINDING →

EARLY BINDING →

final class A {
    all methods are by default final.
}

Can we @Override static methods ?
Not overriding
@Override → // Not!

static methods can't be overridden because its not dependent on objects and are methods called from the parent class. (main)

Overriding depends → Objects

static X depend → objects

static X → overridden

4 you can run but you can't hide
you can inherit but you can't override.

Tweet that shame on LinkedIn.

Polymorphism doesn't apply to instance variables.

ENCAPSULATION

Wrapping of ~~data~~ implementation of data members & methods.

Wrapping of implementation of data members & methods in a class. or a single unit.

ABTRACTION — hiding unnecessary details & only providing essential details

println("Hello"); ↔ Just print

~~Internally~~ how its working we dont know.

Abstract data types →
ArrayList → you get methods add, set, get delete.
but you don't know the internal implementation

DESIGN ISSUE ??

ENCAPSULATION

INTERNAL ✓

IMPLEMENTATION ✓

Containing Information

ABSTRACTION

EXTERNAL ✓

DESIGN ✓

Chaining Info.

Getter Setter → manipulator.

~~INTERFACES~~

~~INTERFACES~~

ABSTRACT CLASSES & INTERFACES

achieved by

(private) data hiding → security

Encapsulation → complexity of a System

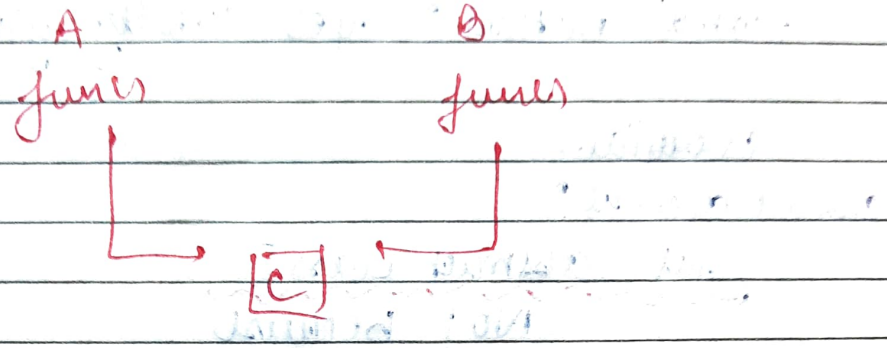
sub process in Encapsulation

(Data hiding)

Unstoppable - Sia.

Lyrics

All smiles, I know what it takes to feel this tru



abstract void career (String, name)

if any method declared abstract
class should be defined abstract.

public ~~class~~ abstract class Parents {

Abstract classes -
methods defined inside can be
abstract.
without ~~def~~ body.

They can be ~~eat~~ overriden by
their child classes object

We cannot make objects of Abstract classes

we can declare variables inside Abstract class constructor.

Static methods can be declared.

normal method ? yes - totally fine.

Override.

can we declare?

final abstract class?

No! because

abstract class needs to be overridden methods by its child class.

dynamic method dispatch

mechanism by which call is overridden at runtime

When an overriden method is called through a Superclass reference,

Java determines which version Super / sub class of that method to be executed based upon the type of object being referred on the call time

At run time it depends on the type of the object being referred to (not the type of the ref variable)

version of the overridden method
that determines which method will be executed

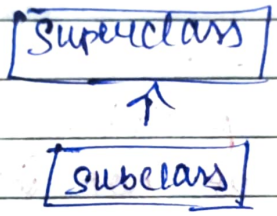
A superclass reference variable can refer to a subclass object

This is called as upcasting

Java uses this fact to resolve calls.

upcasting

Superclass obj = new ~~Superclass()~~; subclass();



typecasting parent obj to child object

static int x = 0; ✓

final int X = 0; ✓

This doesn't support multiple inheritance.

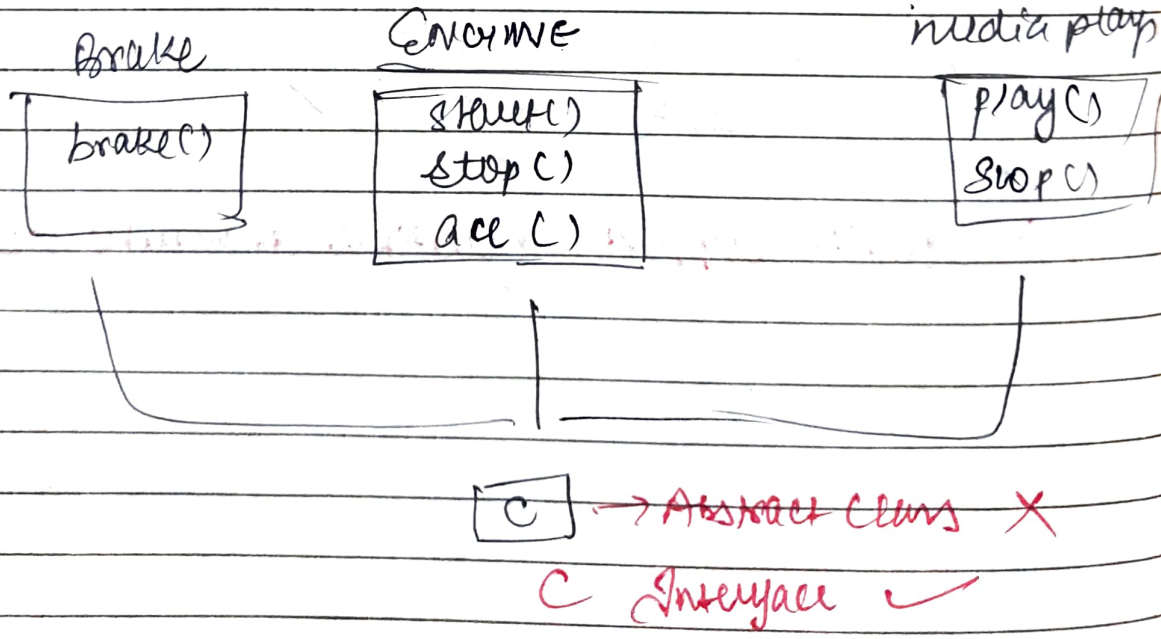
Interfaces in Java ? (implements) multiple

like an abstract class
- not create objects
- all methods are abstract
- variables are final and static by default

by default
- public
- no instance variables.

because to initiate them you need constructor and we cannot define constructor in interfaces

Abstract class provides implementation (Interface but not vice versa)



public interface Engine {

void start();
void stop();
void acc();

public interface Break {
void break();
}

public interface Media {
void start();
void stop();

public class Car implements Engine, Break, Media;

}

psvm() {

Car car = new Car();
car.acc();
car.break();
car.run();

Interfaces allows to implements any classes of any parents --
they don't care about the class relation b/w them!

Engine car = new Car();

↓ what to access he will this

↓ which version (object)?

CASUALLY
CANNOT USE INTERFACES IN PERFORMANCE

CRITICAL CODES
because it inc ↑ runtime
dynamic method dispatch

public class CD player implements media {

class Power Engine implements Engine {

}

↑ checks

@Override

↓

annotations

↓

It also an Interface.

public @

package java.lang

java.lang.annotation.*;

@Target (Element.Type.METHOD)

@Retention (RetentionPolicy.SOURCE)

public @interface Override {

}

JAVA 8 →

default implementation. → special use

↓ ↗ Keyword.

expanded default void fun.

by

without breaking the code.

Static methods can not get overridden

objects

inherit → override

↓

static ✗ ← Object

static methods should have body

cuz its not overridden.

NESTED INTERFACE

```
public class A
    public interface NestedInterface {
        boolean isOdd(int n);
    }
}

class B implements A.NestedInterface
    @Override
    public boolean isOdd(int num)
        return (n & 1) == 1;
```

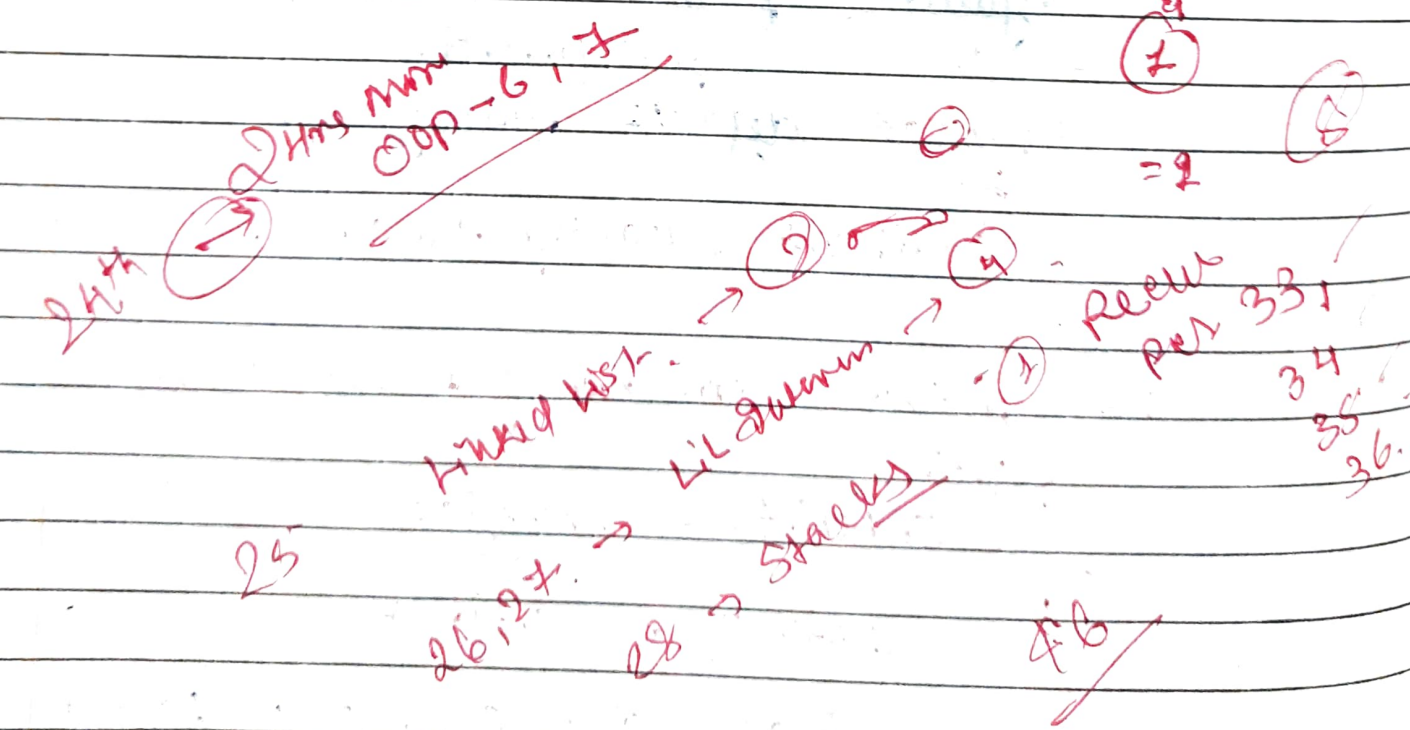
Inside A. Nested Interface

has to be public/default

↑ top level interface

↓ lower level anything

bitwise to check odd



OOP - 6.

Exception Handling →

ArrayList

Generics →

ArrayList<Integer>

↑ Wrapper class not data types

↑

GENERICS — provide checks during the compile time to support generic programming

Type safety with.

Object type.

```
private Object[] data;
private static int default-size = 10;
private int size = 0; // working as index value

public class CustomGenArrayList() {
    data = new Object[default-size];
```

```
public void add (T num)
    if ( isFull () ) {
        resize();
    }
        data [size++] = num;

private void resize()
    int [] temp = new int [2* data.length];
// copy the current items in the new
array.
    for (int i=0; i < data.length; i++) {
        temp[i] = data[i];
    }
        data = temp;
}

private boolean isFull () {
    return size == data.length ;
}

public int remove () {
    int removed = data[--size];
    return removed;
```

with value add
(T)(

public int get(int index) {
    return data[index]
}

public int size() {
    return size;
}

public void set(int index, int value)

BOUND WILD CARD → allows to pass Type class

Integer, Float extends. Number, Float, Int, double
↳ Number class

String -
boolean -

COMPARABLE CLASS

When Java get confused which property of object we comp. to

Rahul age marks
Kunal age marks

(Rahul < Kunal) ⇒ error.

class Student implements Comparable <T>

↓ class has compareTo() method();

↑ Generic Interfaces can also have generic types.

int diff =

(int) (this.marks - o.marks);

if diff = 0

X.compareTo(Y)

(-1) ↑ bigger Y > X

(0) X = Y

. compare (student s1, student s2)

return (int) (s1.marks - o2.marks);

Lambda function

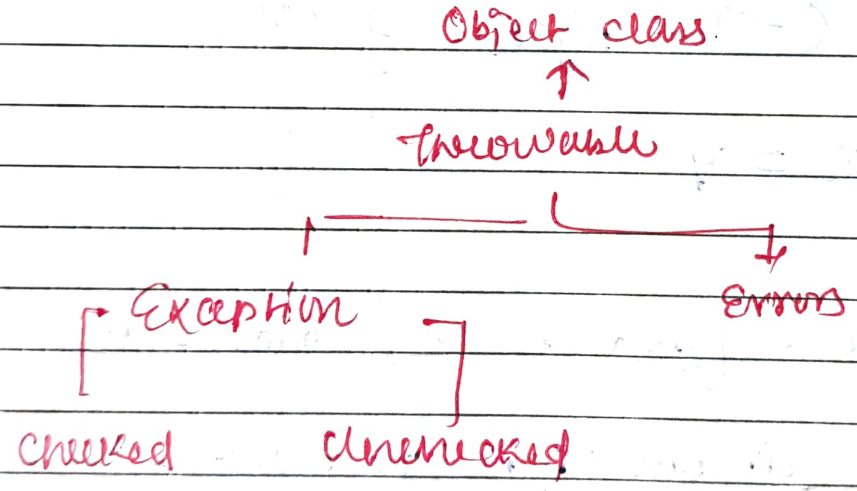
arr.forEach((item) -> System.out.println(item*2));

Exceptional Handling →

prevents normal flow of program

In Java

(Throwable class.)



Keywords — for exception handling

```
int a = 5
int b = 0;
int c = a/b;
```

// pretty way.

```
try {
    int c = a/b;
} catch (ArimeticException e) {
    sout (e.getMessage());
}
finally {
    sout (this will execute Always)
}
```

→ things declared inside this block will execute no matter what.

throws declaration

```
static int divide (int x, int y) throws ArithmeticException {
    if (y == 0)
        throw new ArithmeticException
            ("plz do not divide by zero")
    return a/b;
}
```

OBJECT CLONING —

clone → method
interface — cloneable

(5000 penalty)

offer letter / Regular / dream / superdream / internship

Opt out of capstone

one final report →

company gives the marks

be polite with everyone

implements Cloneable

shallow copy

Kunal [age = 34
name = K
arr = 3,4,5,6,9,1,]

Main [age 34
name K
arr]

age copy

primitives - copy
object variable will point same object

deep copy
create copy of object as well.

Kunal = [age = 34
name = K
int arr = 1,2,3

[age = 34
name = K
int arr = 1,2,3

no pointing

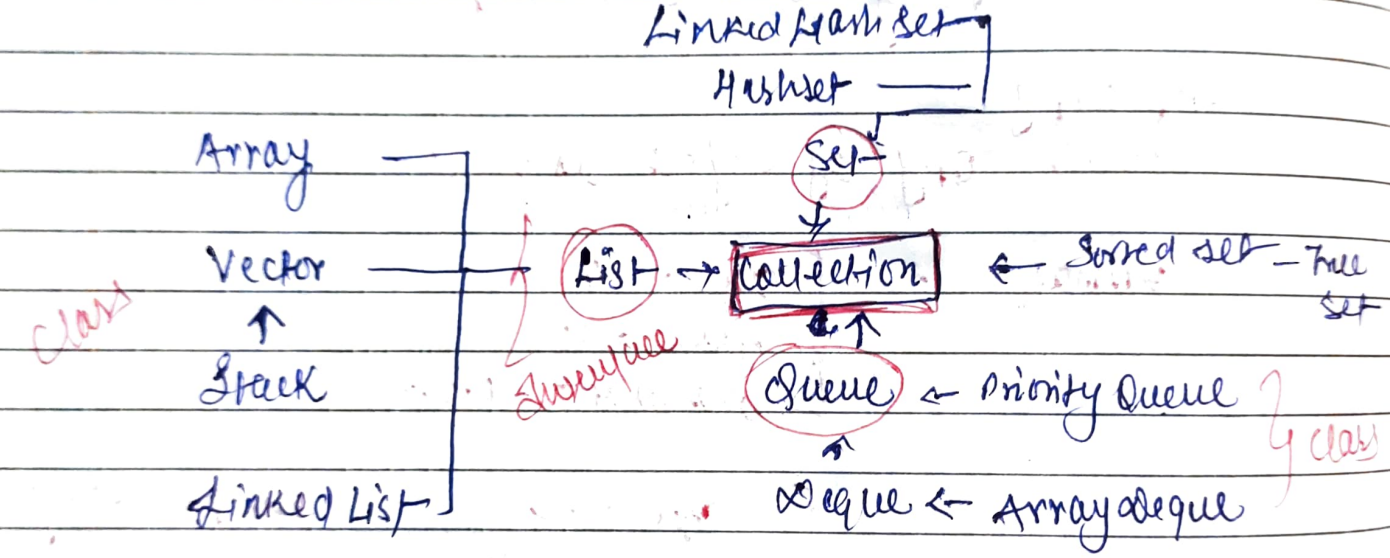
clonable class

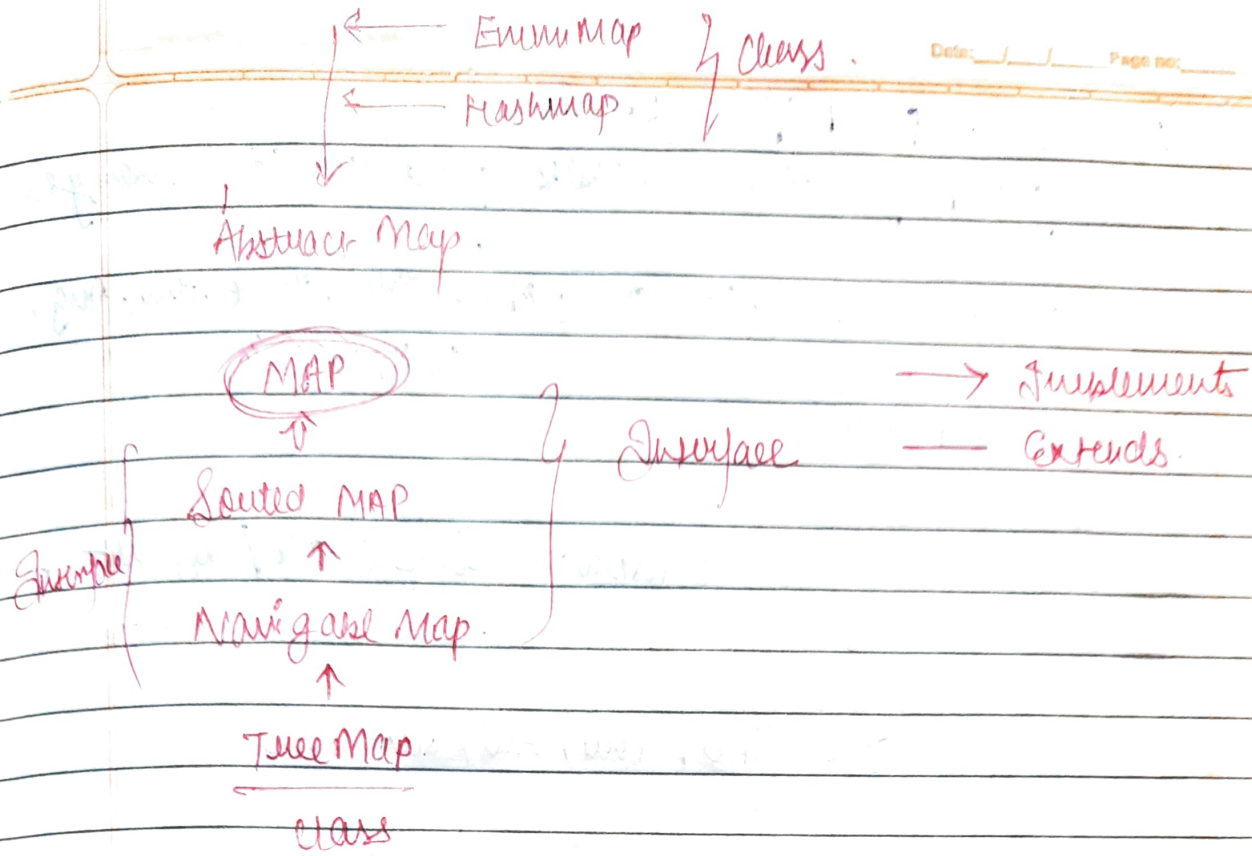
clone();

Collection – framework which contains files functions for the data types user defined

```
List<Integer> list = new ArrayList<>();
List<Integer> list2 = new LinkedList<>();
```

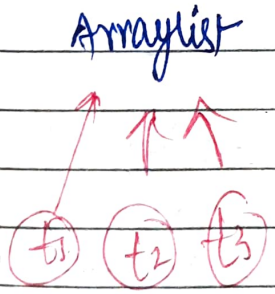
DIAGRAM –

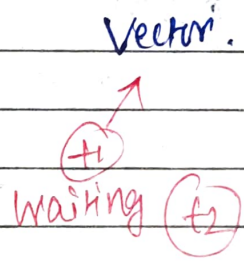




VECTOR → Synchronisation

ArrayList → Multiple thread can access single element at a the same time





List<Integer> vector = new Vector<>();
vector.add(NS)

arraylist → fastly (vectors)

ENUMS – group of constants:

group of ~~data~~ variables you can't change

months (Jan, Feb, Mar, Apr, May, June, Jul, Aug, S, O, N, D);

```
public class Base {
psvm
enum Week {
    Mon, Tue, Wed, Thu, Fri, Sat, Sun
}
// these are enum constants
// public, static, final
// since its final you can't create child enum
// type is week.

psvm.
    Week week;
    week = Week.monady;
    for (week day : week.value)
        sout(day)
week.valueOf(monday).week.ordinal();
```

constructor is private / default.

week() { sout("inside constructor"); }

return enum const

similar to index

We don't want to make new objects or modify it so constructor have to be default or private.

All enum class extends. java.lang.enum

↓

can't be a superclass also since it cannot extend any other class.

this keyword ✓

but it can implement as many interfaces we want

enum

↑

class.

can't extend anything else because already extending

and in java we don't have multiple inheritance.

By mistake to access 1 all of these will be called.

enum week implements A

00 01 02 03

0 1 2 3

10 4 5 6 7

20 30 40

LINKED LISTS

KUNAL KUSHWAHA – 44, 45, 46 (6:30) 4hrs+

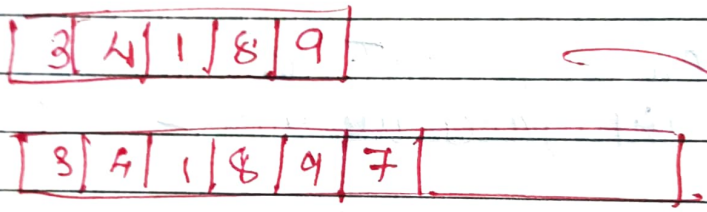
(33,34)(35,36) 3:30 4hrs+ (10hrs+)

0 1 2 3

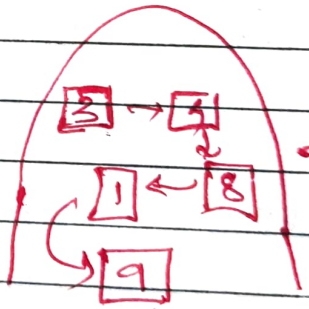
4 5 6 7

27/

limitations



LINKED LIST



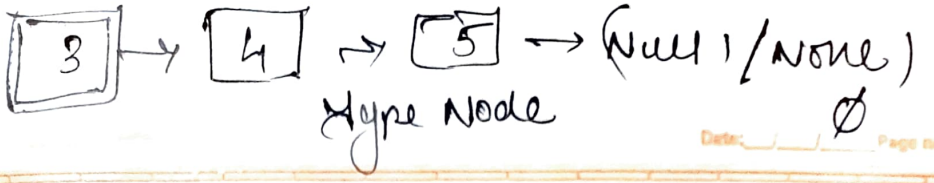
SINGLY LINKED LIST.

11 → null.

head → 3 → 4 → 8 → 5 → tail will ~~point~~ to ~~last~~ node.

my variable points to first node

One side Relationship



Type Node

Date: / / Page no:

class Node {
    int value;
    Node next; pointer as ref variable
    (node type)

Problem you can't access index directly, no idea about how many elements are there

Important to know how head works.

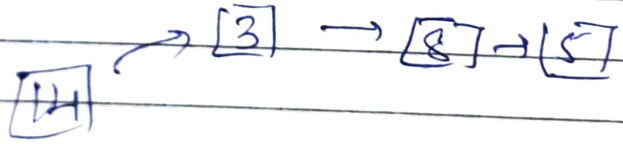
Tail.next = new Node
Tail.next = node.next.

LinkedList<Integer> list = new LinkedList

```
public class LL {      private Node head;
                       private Node tail;
    private int value;
    private int size;
    private Node next;

    public Node (int value)
        this.value = value;

    public Node

public LL() {
    this.size = 0
}
```

LL

see what is required, how? X



node.next = head // head in 2nd pos

Head = node // head is 1st pos

node [18] head = null head tail = null

one item

if (tail == null) tail = head

tail = head 1 item

start = head

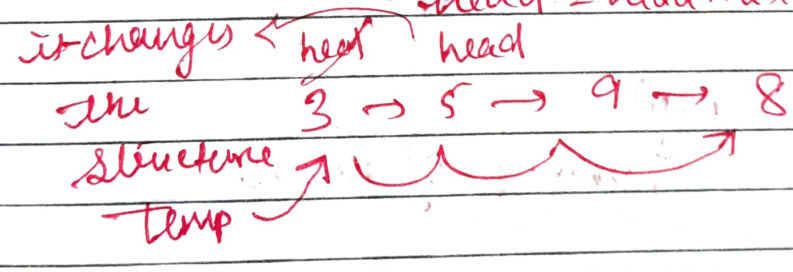
start end = null

```
public void insertFirst(int val){
    Node node = new Node(val);
    // create new node;
    node.next = head;
    head = node;
    if (tail == null)
        tail = head;
    }
    else
    size++;
```

```
while (head != null)
    print(head.value)
    head = head.next
```

(X) wrong.

it changes the structure



3, 5, 9, 8

name temp node.

(temp == null stop)

Not part of linked list goes out of scope

at while loop.

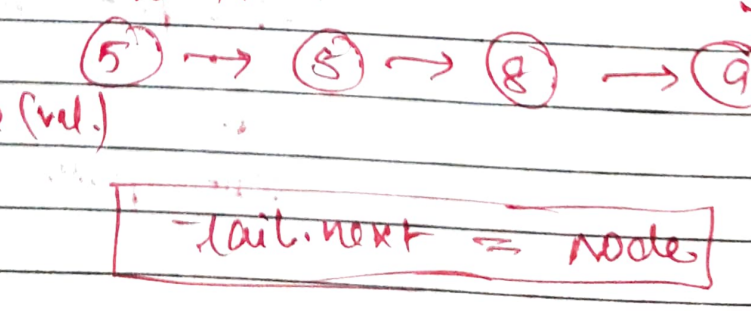
```
public void display() {
    Node temp = head.
    while (temp != null)
        sout( temp.value + " -> ");
        temp = temp.next
    }
sout ("null"); // point object we provide
```

```
LL list = new LL ();
list.insertFirst( 3);
|
|
|
list.display ("END")
```

O(n)
↓
number of nodes.

INSERT ELEMENT IN THE LAST

```
if(tail == null)
    insert value (val.)
```



tail.next = node

size++;

tail = node

// insert at last index

```
public void insertLast(int val) {
    if (tail == null)
        insertFirst(val);
        return;
    }
    Node node = new Node(val);
// what is use of tail pointer? If we wish
   to insert element at last we can
   do it in const time.
   (No need to traverse whole LL)
                          ↑ O(1) → tail
                          O(n) ← head
    tail.next = node;
    tail = node;
    size++;
```

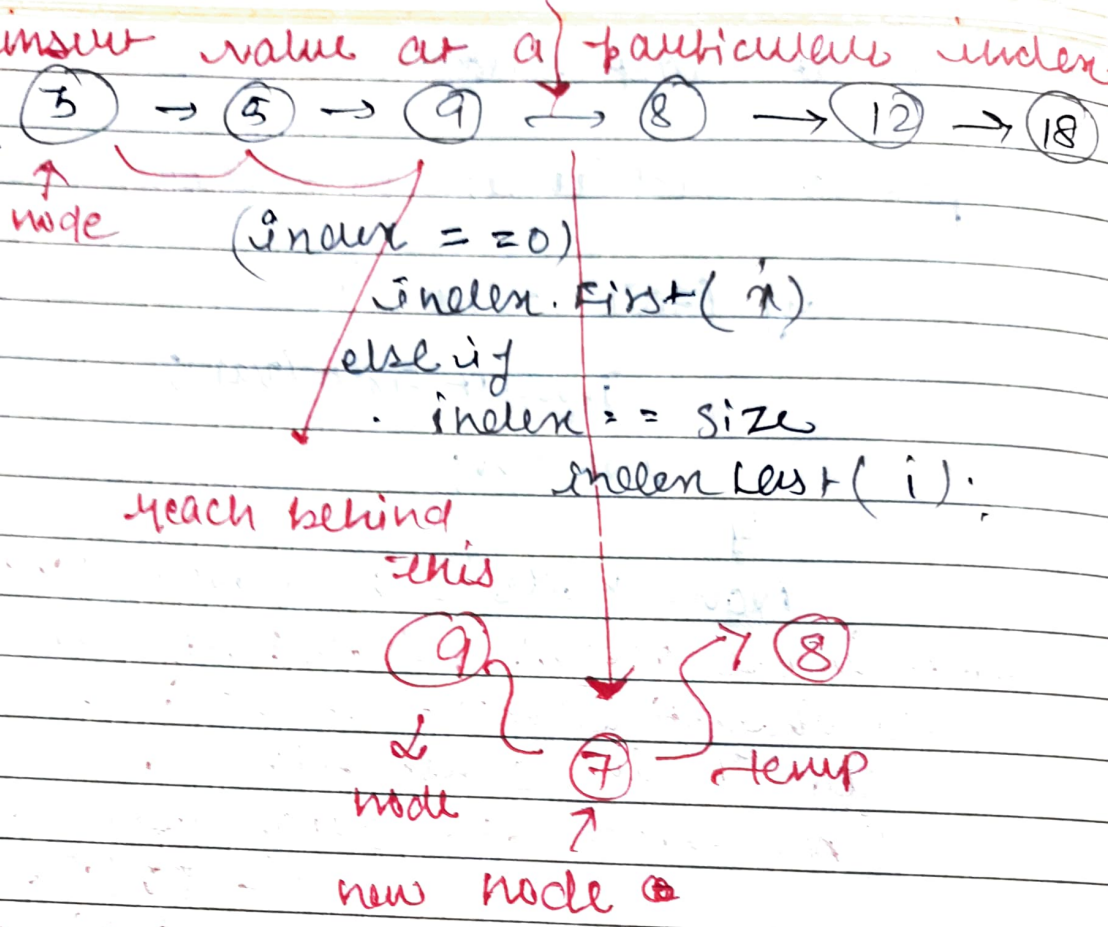
insertLast(99);

3 5 9

(0, 1, 2 index position)

ta

// insert value at a particular index




```
public void insertAtIndex(int val, int index)

    if(index == 0)
        insertFirst(val);
    else if(index == size)
        insertLast(val);
        return;

    Node temp = head;
// if index = 3  go till 2

    for(int i=1; i<index; i++)
        temp = temp.next;
    Node node = new Node(va
```

temp.next = node

size++;

}

list.insert (100, 3)

value, index

index > size

throw exception

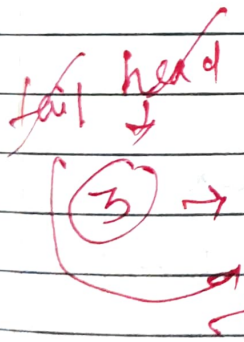
//delete from index

delete first // (head++)

head = head.next;

~~delete last~~

if item



(

3 → ∅

if (head = head

deleteFirst() {

int val = head.value

head = head.next;

if (head == null)

tail = null

}

size--

return val;

deleteFirst();

head

3

// delete last item

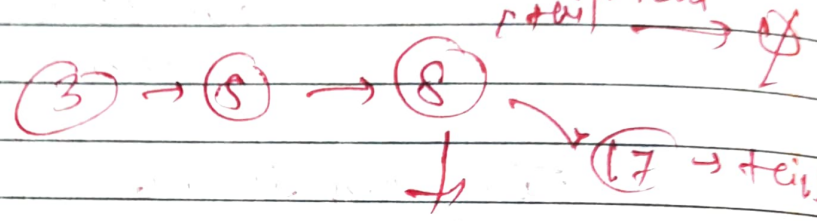
in doubly LL we can go reverse
in singly LL we can't.

point tail = tail.prev

? (reach size - 2 item) last second

↓

iterate this n times.



reach here make tail

tail.next = null

// get reference of last 2nd value

```
public Node get (int index)
    Node node = head;
    for (int i = 0; i < index; i++) {
        node = node.next
    }
```

```
public int deleteLast(){
    if (size <= 1){
        return deleteFirst();
    }
    Node secondLast = get(size-2);
    int value =
```

Corner Cases →