Object Oriented Programming

Lecture-1

→ Suppose stores 5 roll nos.

```
int[] numbers = new int[5];
```

// Stores 5 names

```
String[] names = new String[5];
```

✱ Now, if a teacher says :—

// Stores data of 5 students :
{roll no, name, marks}

Combining into this in a single entity, is a class

```
int[] rno = new int[5];
String[] name = new String[5];
float[] marks = new float[5];
```

✱ A class is a name group of properties and functions.

```
// Student[] students = new Student[5];
```

✱ A class name will be started by Capital letter

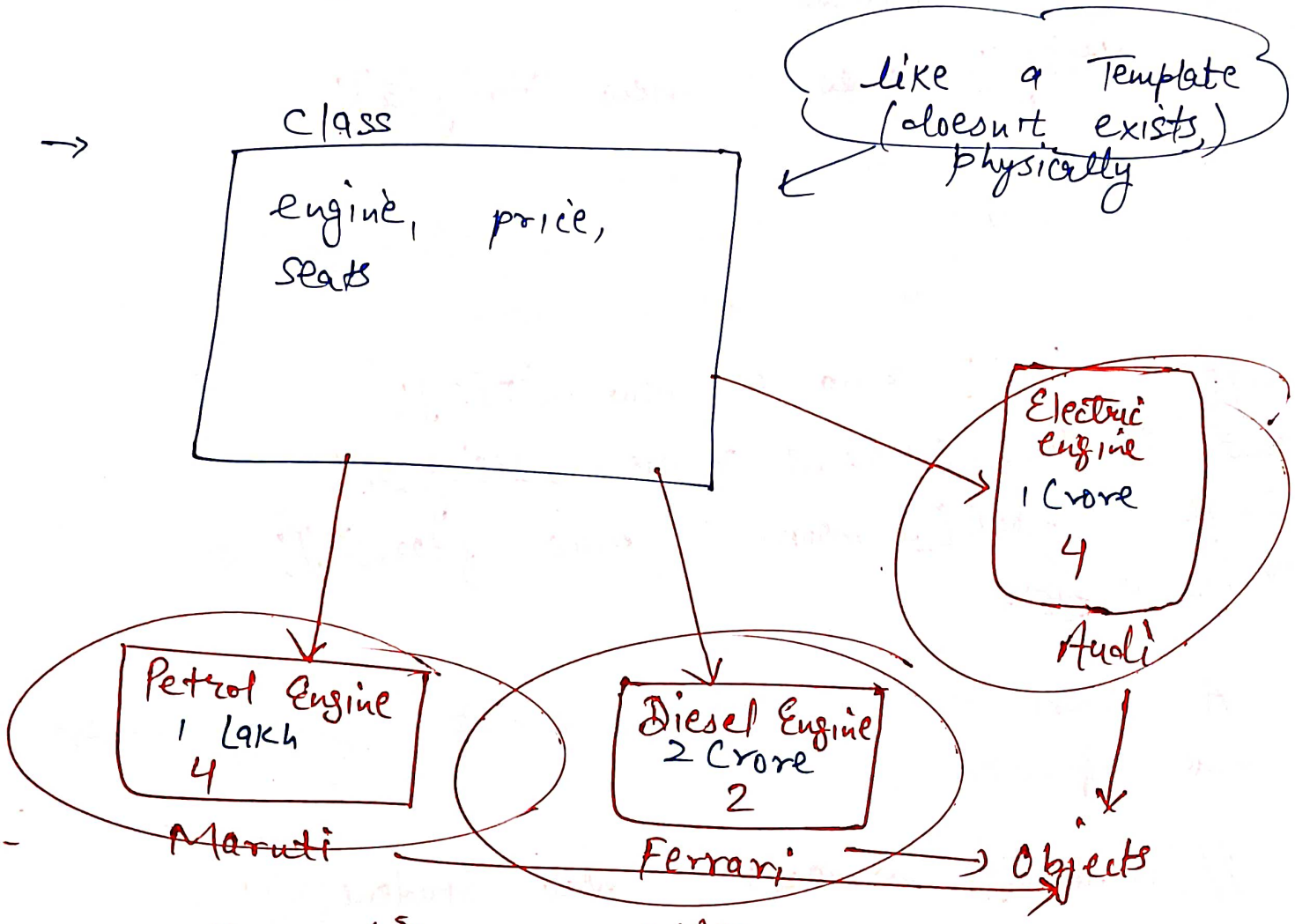
✱

```
// create a class
class Student {
    int[] rno = new int[5];
    String[] name = new String[5];
    float[] marks = new float[5];
}
```

In main function :-

```
public static void main (String [] args)
{
    Student Kunal;
}
```

Kunal → reference Variable → object of class Student



→ Real-life examples:-

Cars, Human beings

A class is a template of an object.

An object is an instance of a class.

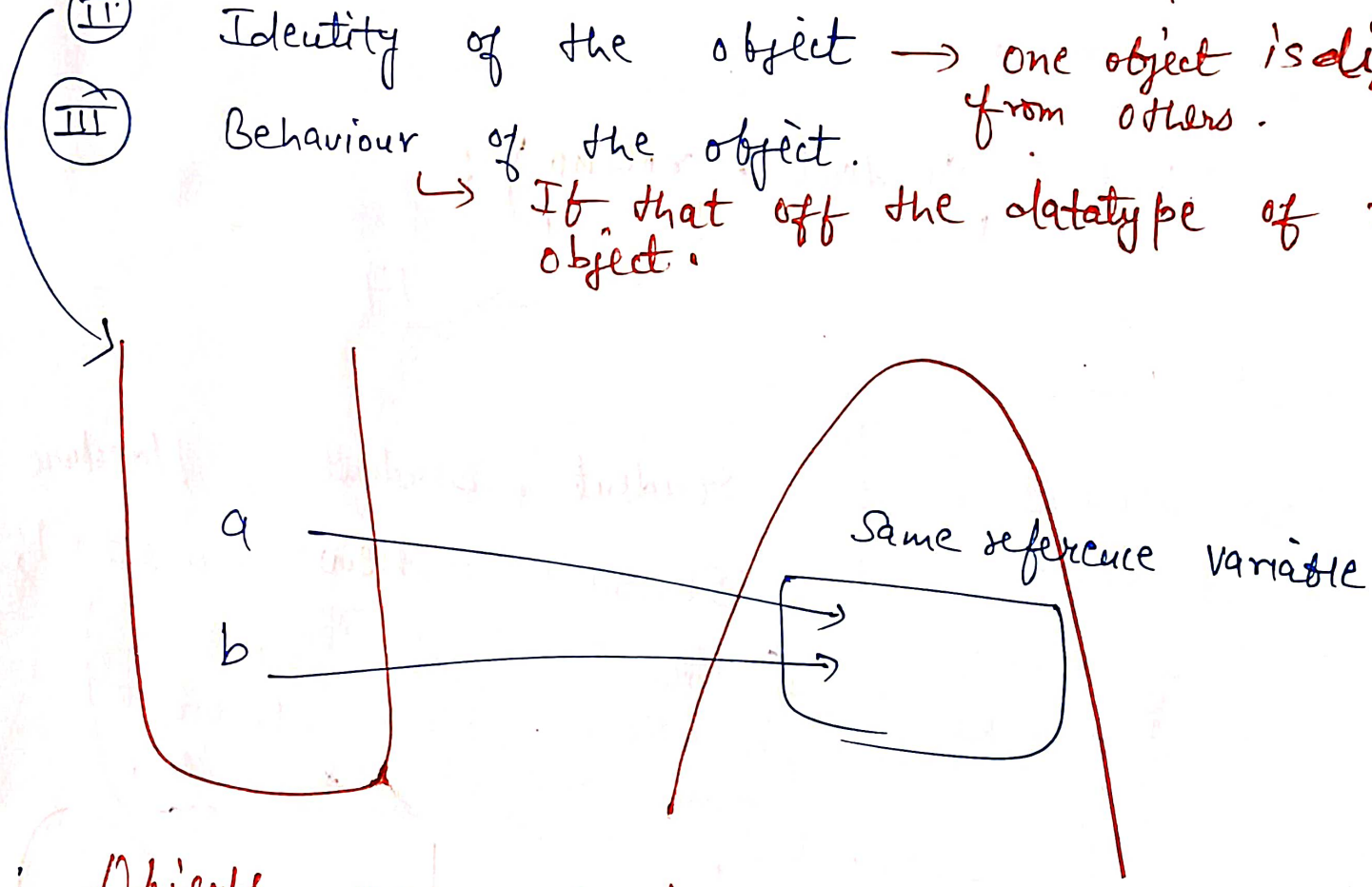
* Class → logical construct
* Object → physical reality // Occupies space in memory.

* Objects are categorized into :-

(I) State of the object → value from its datatype

(II) Identity of the object → one object is different from others.

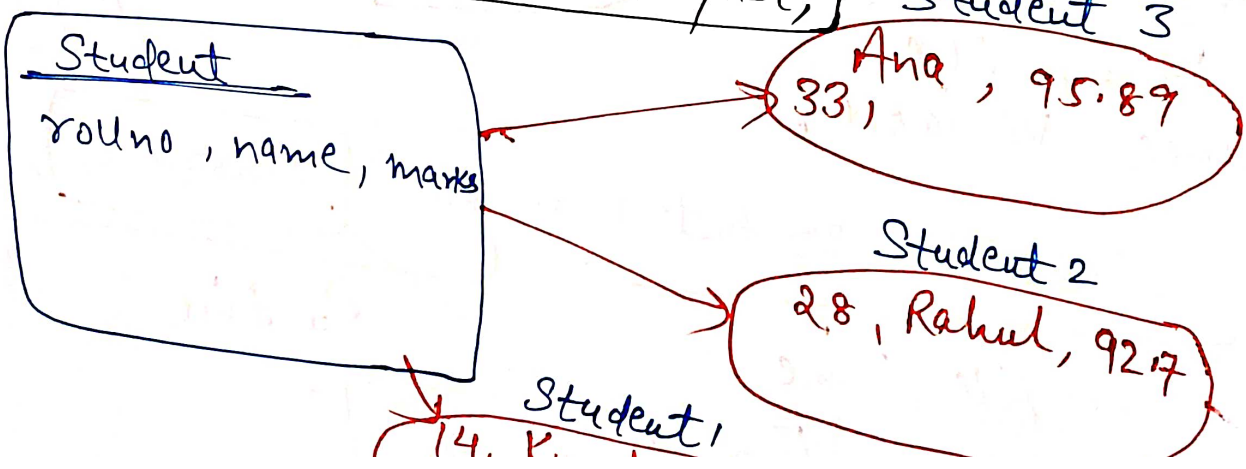
(III) Behaviour of the object.
↳ If that off the datatype of the object.



Objects are stored in Heap Memory.

Reference Variables are stored in Stack Memory

ex:- `ArrayList = new Arraylist;`



* How do we access objects Student 1, Student 2, Student 3 ?

→ By using (.) operator.

* Variables inside the Object is called Instance Variables.

* (.) Operator :-

Sout (Student 1. rollno) ;

→ 14

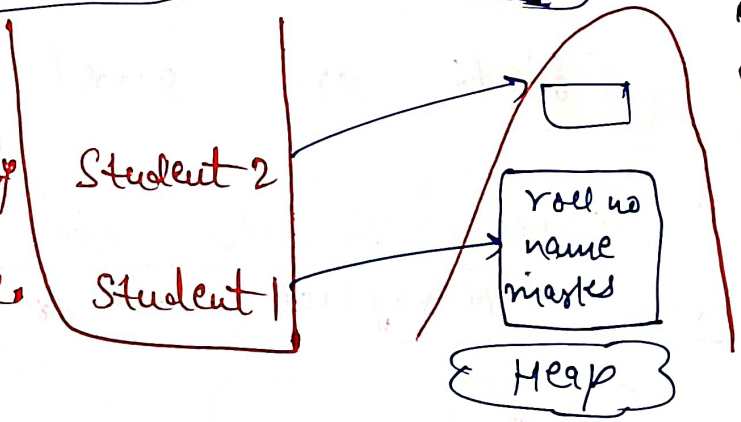
* new

```
class Student {
    int rno;
    String name;
    float marks;
}
```

Student Student 1; //declare

Student 1 = new Student ();

When it will not be initialised, then it will gives output as null.



* new operates dynamically allocates allocating the memory at Run-time & return the reference variable.

Student student 1 = new Student ();

Compile Time — Runtime

A Constructor

- All class objects in Java, must be allocated dynamically.

```
public class OOPs {
    public static void main (String [] args) {
        Student[] student = new Student[5];
        Student Kunal = new Student();
        Kunal.rno = 13;
        Kunal.name = "Kunal Kushwaha";
        Kunal.marks = 88.5f;
        System.out.println (Kunal); → //gives random value;
        System.out.println (Kunal.rno); //gives by deftt 0;
        System.out.println (Kunal.name); // null;
        System.out.println (Kunal.marks); //gives 0;
    }

static class Student {
    int rno;
    String name;
    float marks = 90;
}
}
```

object

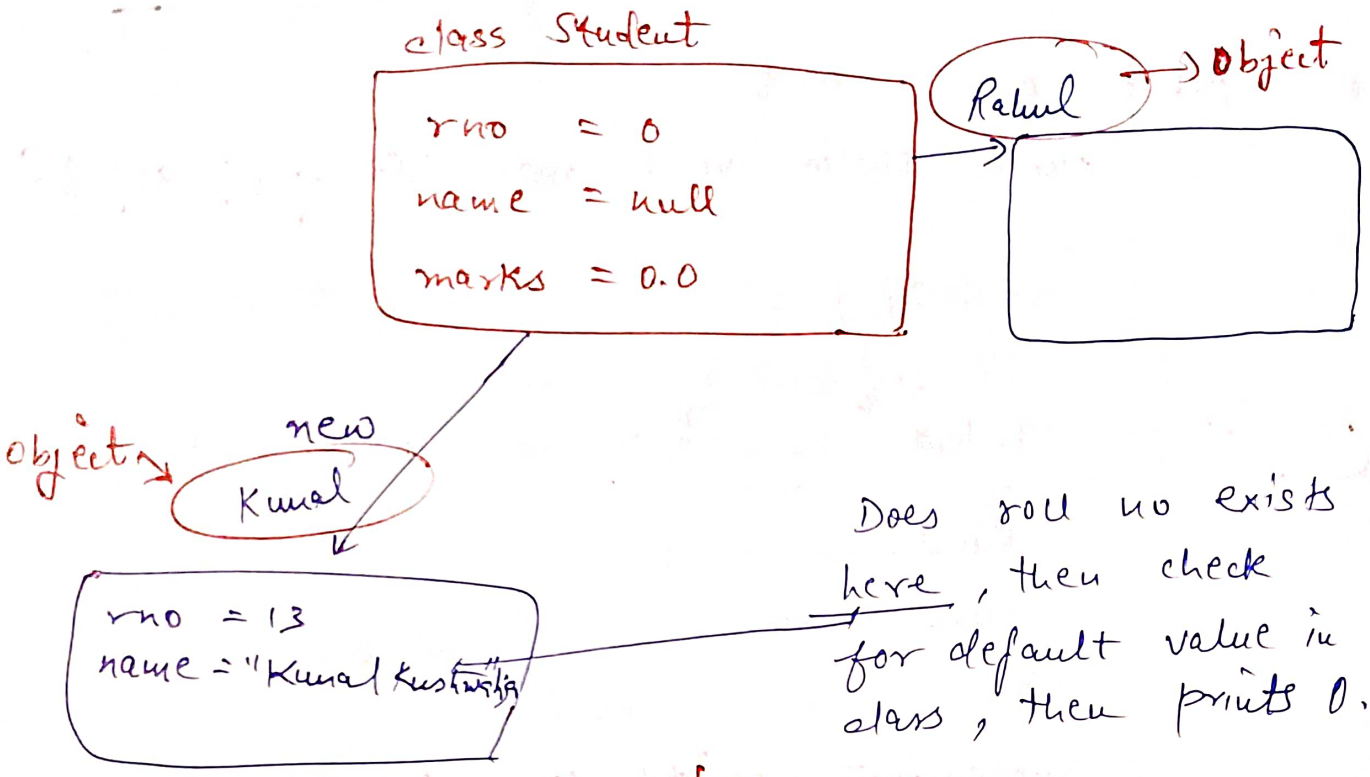
Output

Introduction OOPs$@5f184fc6
13
Kunal Kushwaha
90.0 or 88.5

If initialised in class prints, if marks is not initialised in class.

Initialized outside the class.

Visualization



Does roll no exists here, then check for default value in class, then prints 0.

Kunal.rno = 13

roll number inside Kunal object

Kunal.salary → Error

Kunal.rno → 0

Kunal.name → null

Rahul.rno → 0

Rahul.name → null

Because Rahul objects does not have their values, it will print values of their class.

CONSTRUCTOR

A constructor basically defines what happens when your object will be created.

Constructor is special function that runs when you creates an object and it allocates some variables.

What :

By-default Constructor

```
Student Kunal = new Student(13, "Kunal", 84.3);
Kunal.changeName("Shoe Lover");
```

Special type of function in the class.

bind these arguments with the object.

In main function

```
Student Kunal = new Student();
Student rahul = new Student();
Kunal.greeting();
```

```
class Student {
    int rno;
    String name;
    float marks;
// We need a way to add the values of the above properties object by object
// Initialising a Constructor
// We need one word to access every object.
    Student() {                          A Constructor.
        this -> (Kunal).rno = 13;
        this -> (Kunal).name = "Kunal Kushwaha";
        this -> (Kunal).marks = 88.5f;
    }
    void greeting() {
        System.out.println("Hello! My name is " + name);
    }
    void changeName(String newName) {
        name = newName;
    }
}
```

// Hello! my name is "Kunal Kushwaha".

replaced with Kunal

this.name

Constructor Overloading

```
Student (int rno, String name, float marks)
{
    this.rno   = rno;
    this.name  = name;
    this.marks = marks;
}
```

if we use same variables like this, we use this.rno = rno;

// Student arpit = new Student (17, "Arpit", 89.7f);
// here, this will be replaced with Arpit;

Student Kunal = new Student (15, "Kunal", 85.4f);

roll
↓
15

Kunal.roll = roll
= 15

```
Student (Student other) {
    this.name  = other.name;
    this.rno   = other.rno;
    this.marks = other.marks;
}
```

Student random = new Student (Kunal);

✱ Calling a Constructor from another Constructor.

✱ this → referred to the current object when it calls it.

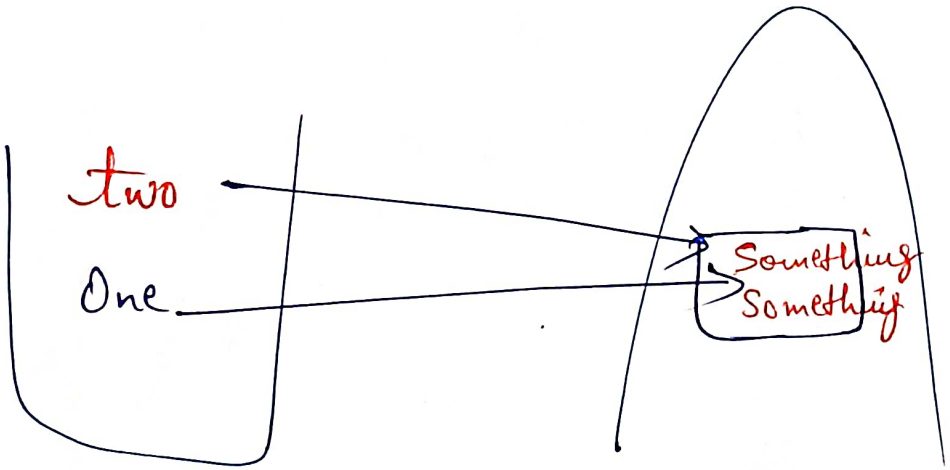
```
Student () {
// this is how you call a Constructor
   from Another Constructor.
this ( 13 , "default person " , 100.0f );
}
```

new Student(); → internally

✱ In Java, the primitives Data Types are not implemented as Objects whereas Objects are stored in Heap Memory. But, In Java Primitives are not objects, hence they are stored in Stack memory only.

✱ Memory Allocation for new Keyword.

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



one.name = "Something Something";

System.out.println(two.name);

Also changes

O/P

→ // Something Something.

✱ WRAPPER CLASSES

```
public class WrapperExample {
    psvm() {
        // int a = 10;
        int b = 20;
        Integer num = 45;
        // swap(a,b);
        // Sout(a+" "+b);
    }
    void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
    }
}
```

Wrapper class has its own functions/methods

← Not swaps

If,

```
Integer a = 10;
Integer b = 20;
swap(a,b);
Sout(a+" "+b);
```

Integer or int

Not swaps

swapping in only this method

☆ final Keyword

- final is a keyword, we can basically prevent the content to be modifying.

```
final int INCREASE = 2;
```

Now, this object / ref variable will not be modifying.

Code :-

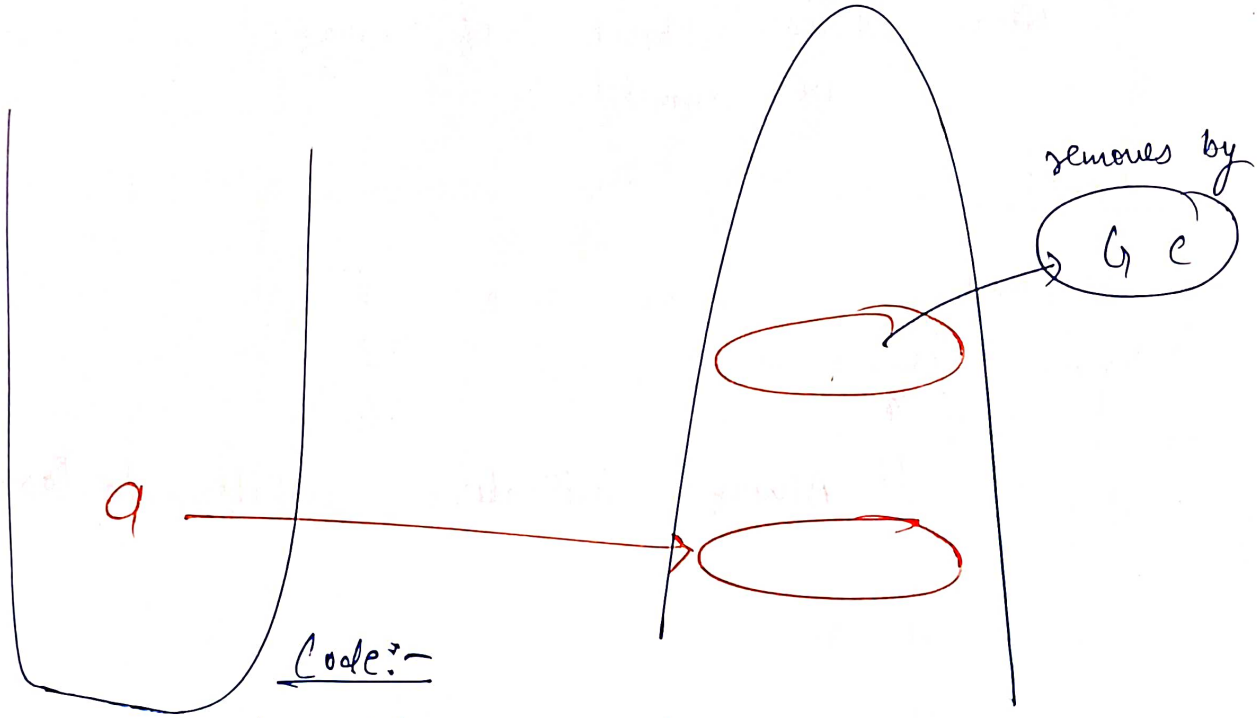
```
final int bonus = 2;
Error bonus = 3;
```

Code :-

```
// Always initialize while declaring.
class A {
    final int num = 10;
    String name;
    public A (String name) {
        this.name = name;
    }
}
final A Kunal = new A ("Kunal Kushwaha");
Kunal.name = "other name";
// When a non-primitive is final, you cannot reassign it.
// Kunal = new A ("new object");
```

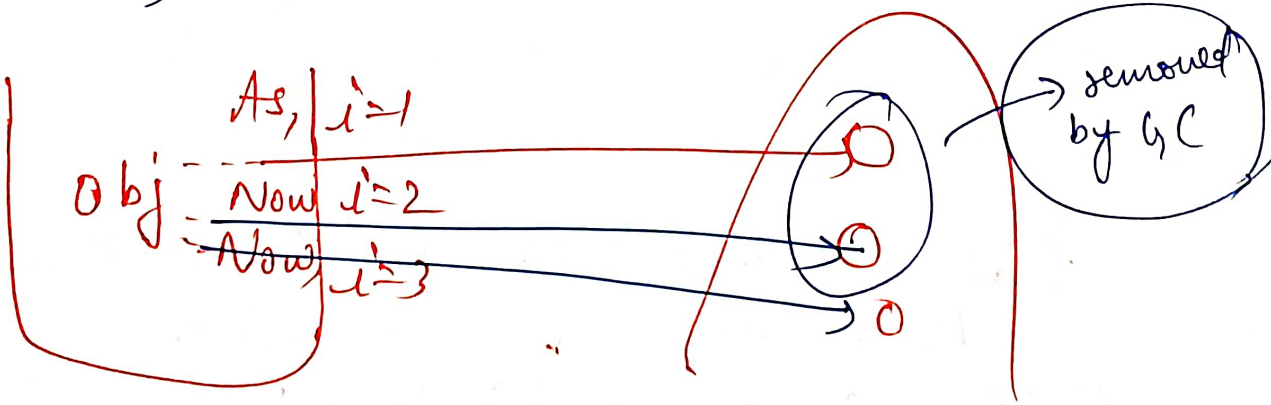
GARBAGE COLLECTION

Java does not have Destructors as C++.



Code:-

```
A obj;
for (i = 0; i < 10,000,000; i++) {
    obj = new A ("Random name");
}
```



```
class A {
    final int num = 10;
    String name;

    public A(String name) {
        Sout("object created");
        this.name = name;
    }

    @Override
    protected void finalize() throwable {
        Sout("Object is destroyed");
    }
}
```

Lecture-2

✱ PACKAGES

// just like folder.

We can create different Java classes with same with the help of packages.

For more refrences, check ← OOPs concepts IDE.

✱ Import files

```
package A;
import packages.B.Greeting.message;
public class Greeting {
    psvm() {
        Sout("Hello! World");
        message();
    }
}
```

```
package B;
public class Greeting {
    psvm() {
    }
    public static void message() {
        Sout("This course is awesome
    }
}
```

Output

Hello! World
This course is awesome.

Static

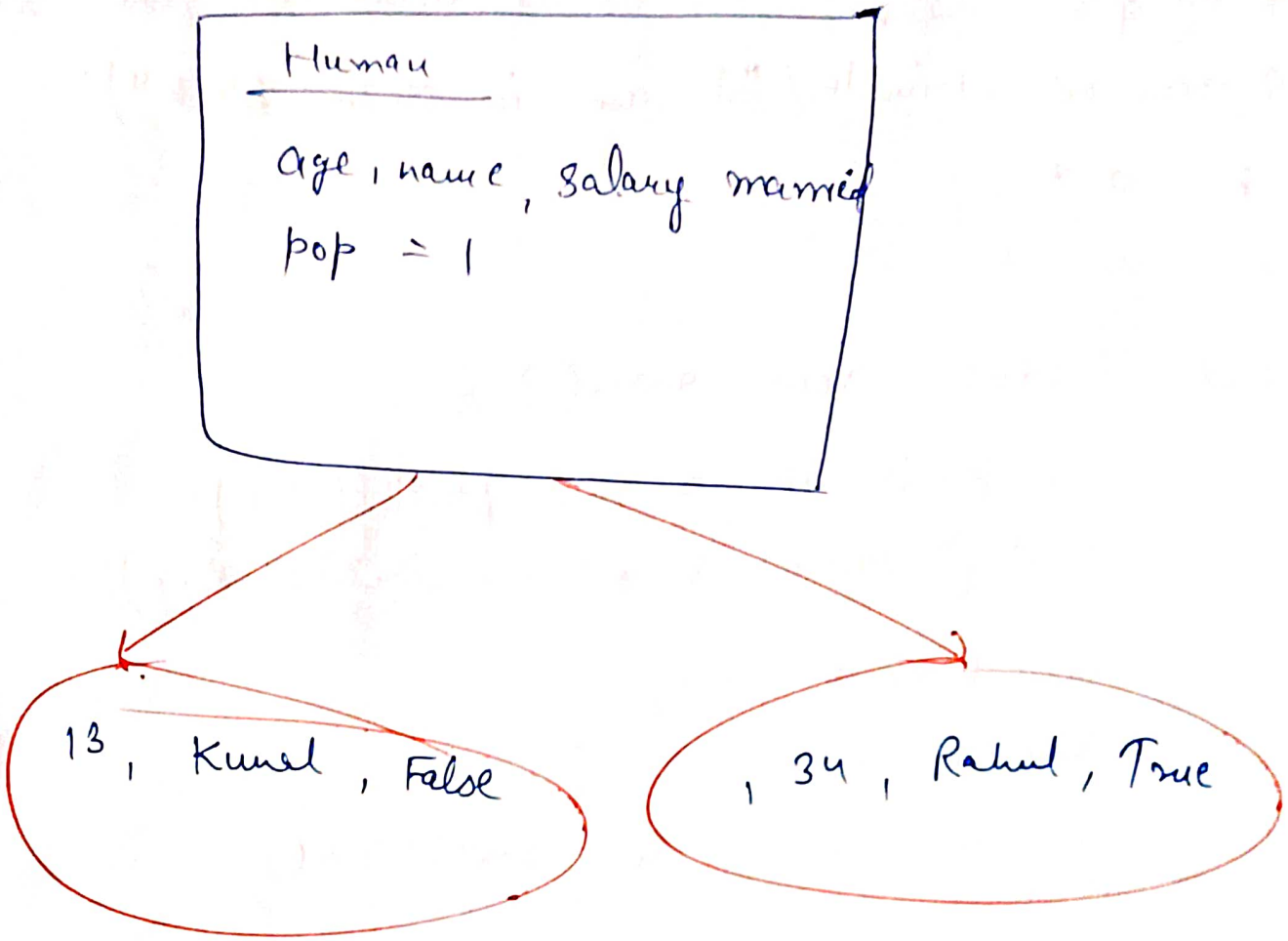
Those properties those are not related to object, but are common to all the objects of that class, like population example, we declared those as static.

```
public static Example
  public class Human{
    int age;
    String name;
    int salary;
    boolean married;
  Static long population;
  Human(int age, String name,
        int salary, boolean married)
  {
    this.age = age;
    this.name = name;
    this.salary = salary;
    this.married = married;
    this.population += 1;
  } Human.also();
}
```

```
package static Example
  public class Main {
  psvm() {
Human Kunal = new Human(
        22, "Kunal Kushwaha",
        10,000, false);
Human Rahul = new Human(---);

Sout (Kunal.population);
Sout (Rahul.population);
}
```

o/p
2
2



* public static void main (String[] args) { }

Why main method is declared as Static ?

static variable / method Actually belongs to a Class not an object.

Static method needs to only access static data, it can't access non-static data.

* Static Block

```
public class StaticBlock {
    static int a = 4;
    static int b;
```

```
// Will only runs once, when the first obj is created i.e, when the class is loaded for first time.
    static {
    System.out.println("I am in static block");
     b = a * 5;
    }

    public static void main() {
        StaticBlock obj = new StaticBlock();
        Sout (StaticBlock.a + " " + StaticBlock.b);

        StaticBlock.b += 3;
        Sout (------------);
        StaticBlock obj2 = new StaticBlock();
        Sout (------------);
    }
}
```

O/P

```
I am in static block.
4 20
4 23
4 23
```

✱ Inner Classes

```
static public class InnerClasses {
    if static, class Test {
        String name;
        public Test(String name) {
            this.name = name;
        }
    }
```

↓ Error is gone

{ outside classes can't be static, Only inner can be or not static }

```
public static void main (String[] args) {
    Test a = new Test ("Kunal");
    Test b = new Test ("Rahul");
}
}
```

O/P

~~Rahul~~ → Kunal

Rahul

SINGLETON Class

// having only or create 1 object;

Same packages

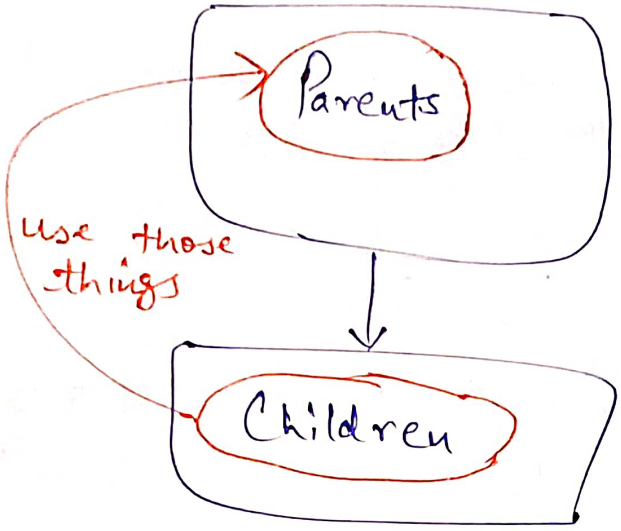
```
public class Singleton {
    private Singleton() {
    }
    private static Singleton instance;
    public static Singleton getInstance() {
        //Check whether 1 obj is created or not.
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

```
public class Main {
    psvm () {
        Singleton obj = Singleton.
                        getInstance();
        obj2 = ____;
        obj3 = ____;
        // All 3 objects/ref variable
        are pointing to just
        one object.
```

OOPs PRINCIPLES

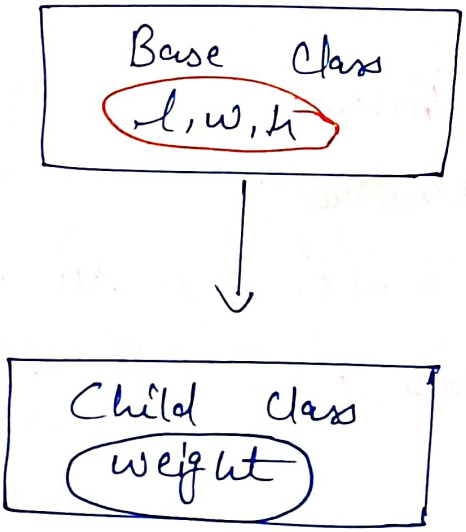
Lecture - 3

INHERITANCE



['money, human traits, values']

- If there's a class that is able to use the properties and attributes of another class → Inheritance.
- In OOPs

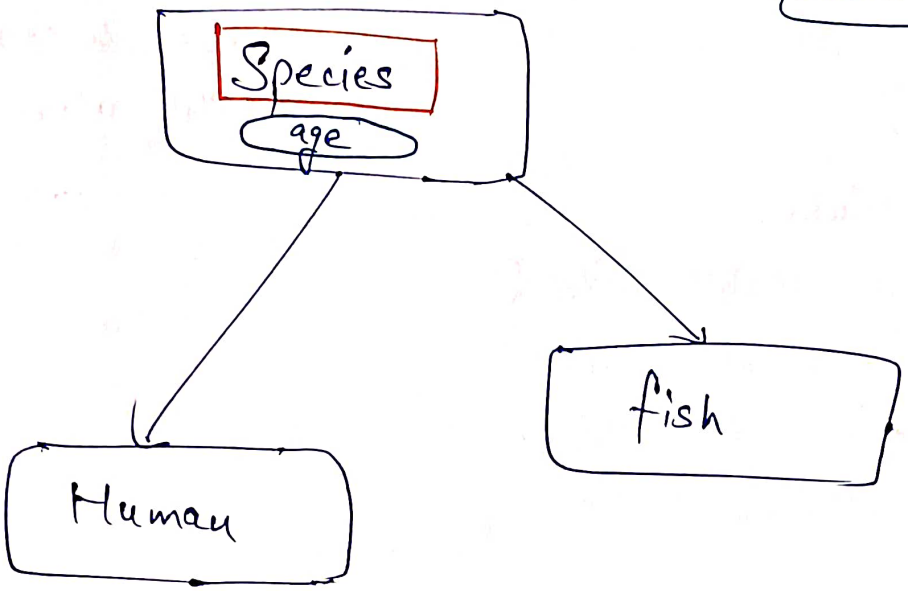


This child class is inheriting properties from Base class.

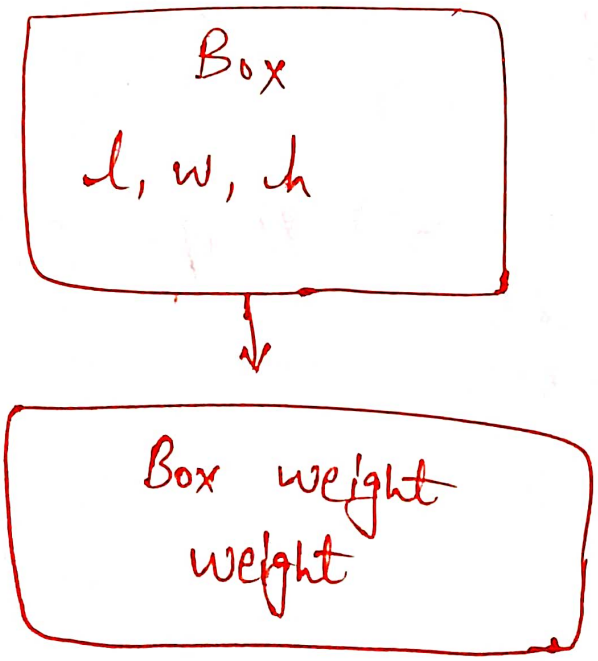
```
class child extends Base {
    int weight;
}
```

```
Child child = new child();
child.l;
child.w;
child.h;
```

initialize parent class variables also




```
Human kunal = new Human();
kunal.age;
```



Code:-

```
package Inheritance;
public class Box {
    double l;
    double h;
    double w;

    Box() {
        this.l = -1;
        this.h = -1;
        this.w = -1;
    }

    // Cube
    Box(double side) {
        this.l = side;
        this.h = side;
        this.w = side;
    }

    Box(double l, double h, double w) {
        this.l = l;
        this.h = h;
        this.w = w;
    }

    Box(Box old) {          // Copy Constructor
        this.l = old.l;
        this.h = old.h;
        this.w = old.w;
    }
```

Parent-class doesn't have access of the child class Parameters.

```
public void Information() {
    Sout("Running the Box");
}
}
```

→
```
package Inheritance;
public class Main {
    psvm() {
// Box box1 = new Box(4.6, 7.9, 9.9);
// Box box2 = new Box(box1);
// Sout(box1.l + " " + box1.w + " " + box1.h);

Box weight box3 = new Boxweight();
Box weight box4 = new Boxweight(2,3,4,8);
Sout(box3.h + " " + box3.weight);
```

→ Gives, -1 & -1 as output

```
}
}
```

→
```
package Inheritance;
public class Boxweight extends Box {
    double weight;

    public Boxweight() {
        this.weight = -1;
    }
public Boxweight(double l, double h, double w, double weight) {
    Super(l, h, w); // What is this? Call the parent class Constructor. // Used to initialise values present in Parent class.
    this.weight = weight;
}
```

Also;

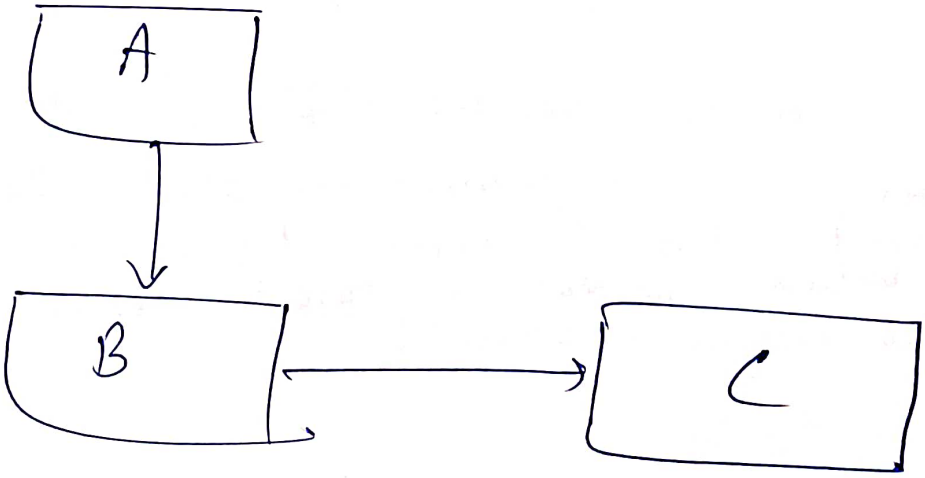
```
Box box5 = new Boxweight(2,3,4,8);
Sout(box5.w);  ⟶  // Gives 4
```

// there are many variables in both parent and child classes.
// You are given access to variables that are in the reference type i.e; Boxweight.
// Hence, you are accessed to weight variable.
// This also means that the ones you are trying to access should be initialised.
// But here, when the object itself is of type parent class, how will you call the constructor of child class

```
// Boxweight box6 = new Box(1,3,4);
//     Sout(box6);
```

// This is why Error.

* Super keyword

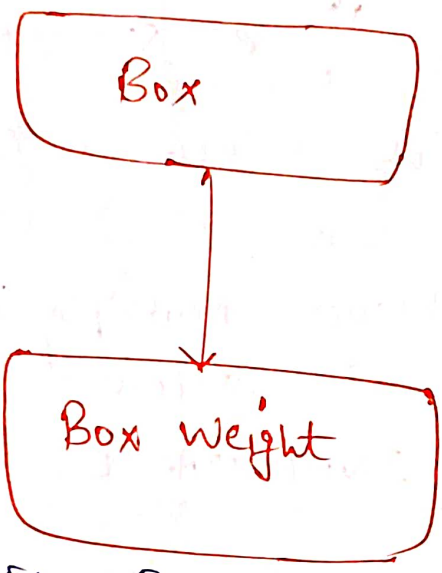


Every single class we create as object as its super class. / object class.

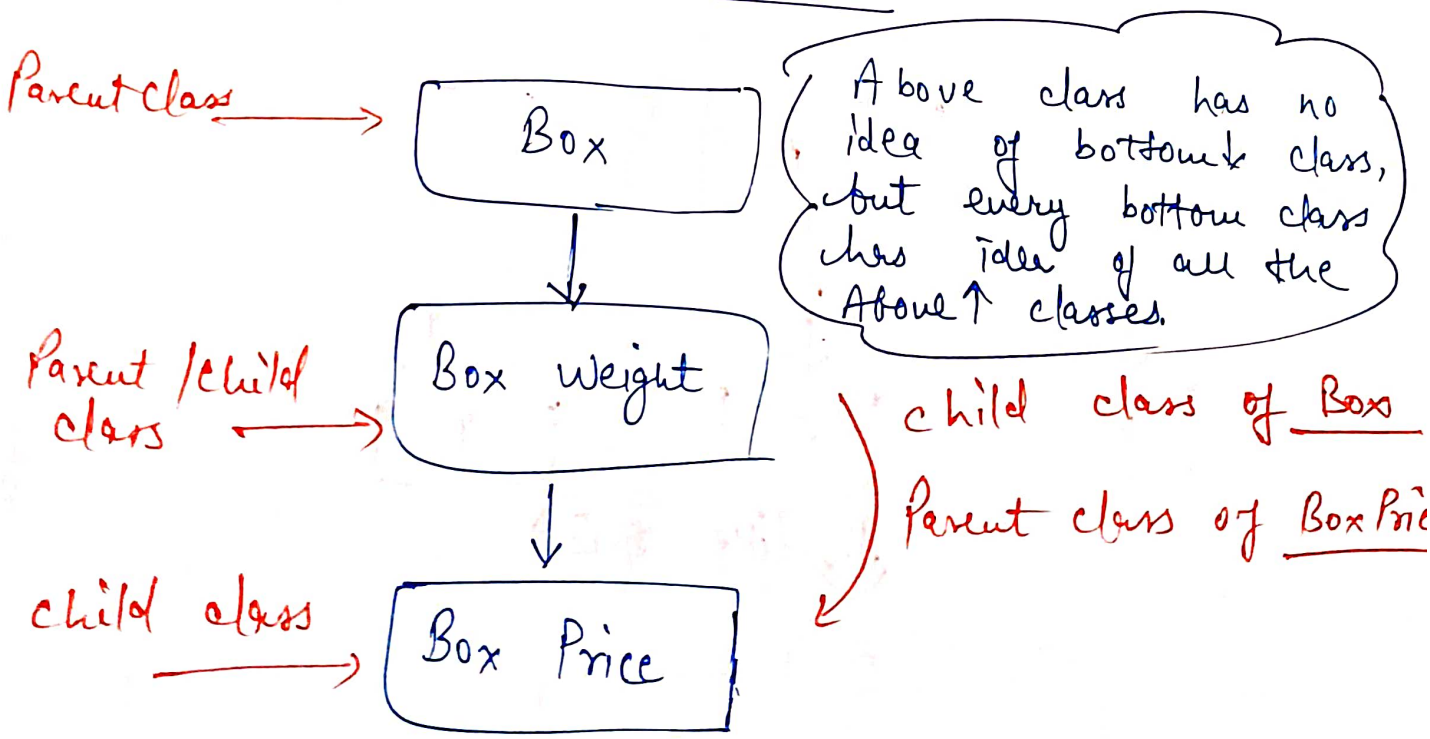
TYPES OF INHERITANCE

(I) SINGLE INHERITANCE

One class extends another class.



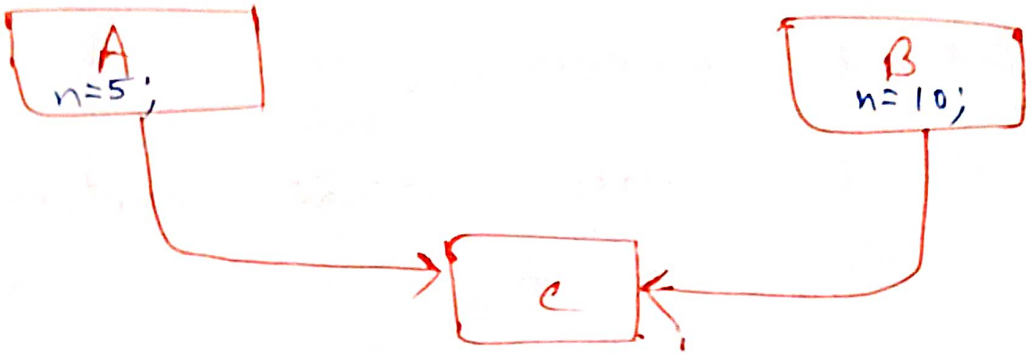
(II) MULTILEVEL INHERITANCE



(III) MULTIPLE INHERITANCE

Not allowed in Java.

One class extending more than 1 classes,



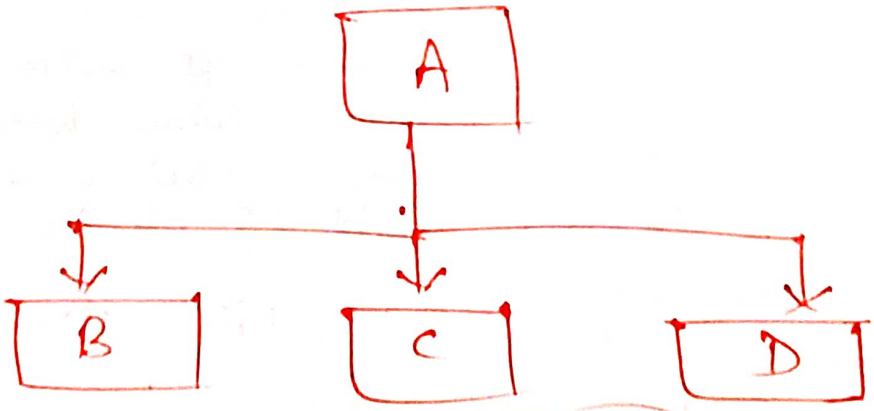
C obj = new C();

C.n // ? Which one to print?

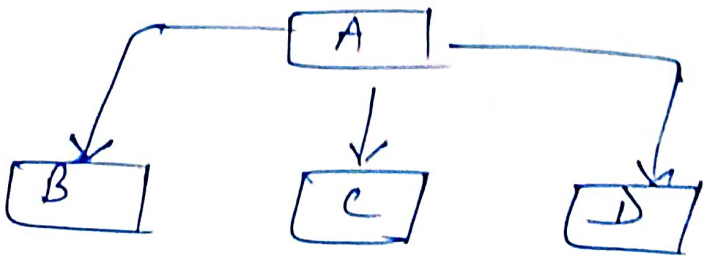
Hence Multiple Inheritance is not allowed in Java.

(IV) HIERARCHIAL INHERITANCE

One class is Inherited by many classes,



like this

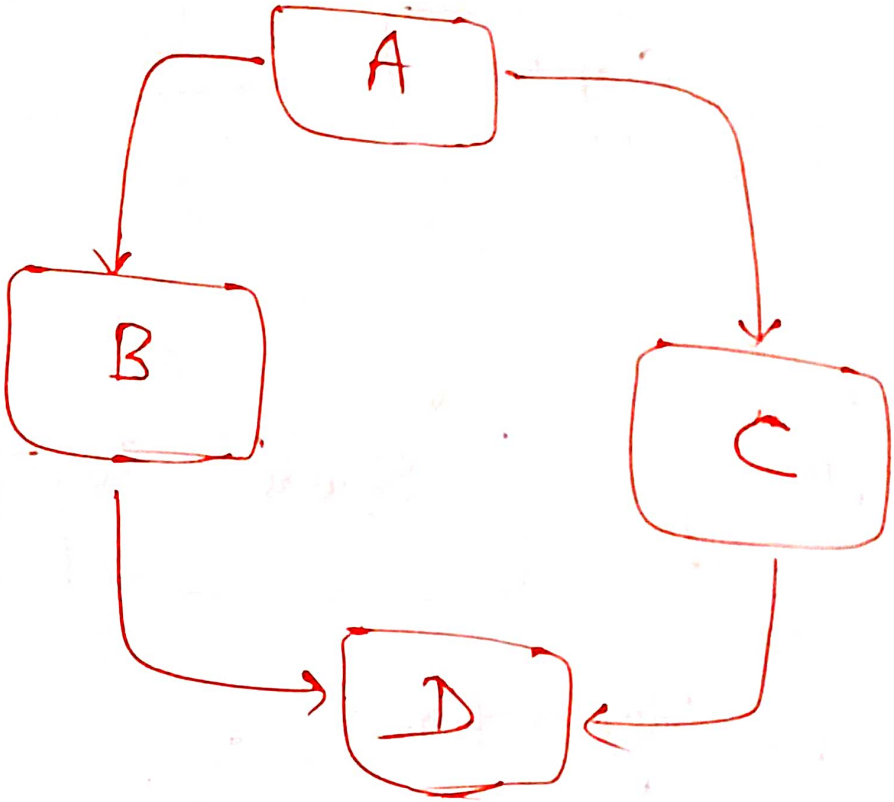


(V) HYBRID INHERITANCE

Combination of Single and Multiple Inheritance

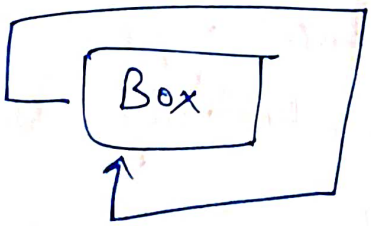
There is no multiple Inheritance, So there is no Hybrid Inheritance in Java.

Check Interfaces Lecture for more;



* Using Interfaces for this, we will discussed later on:—

Note:—



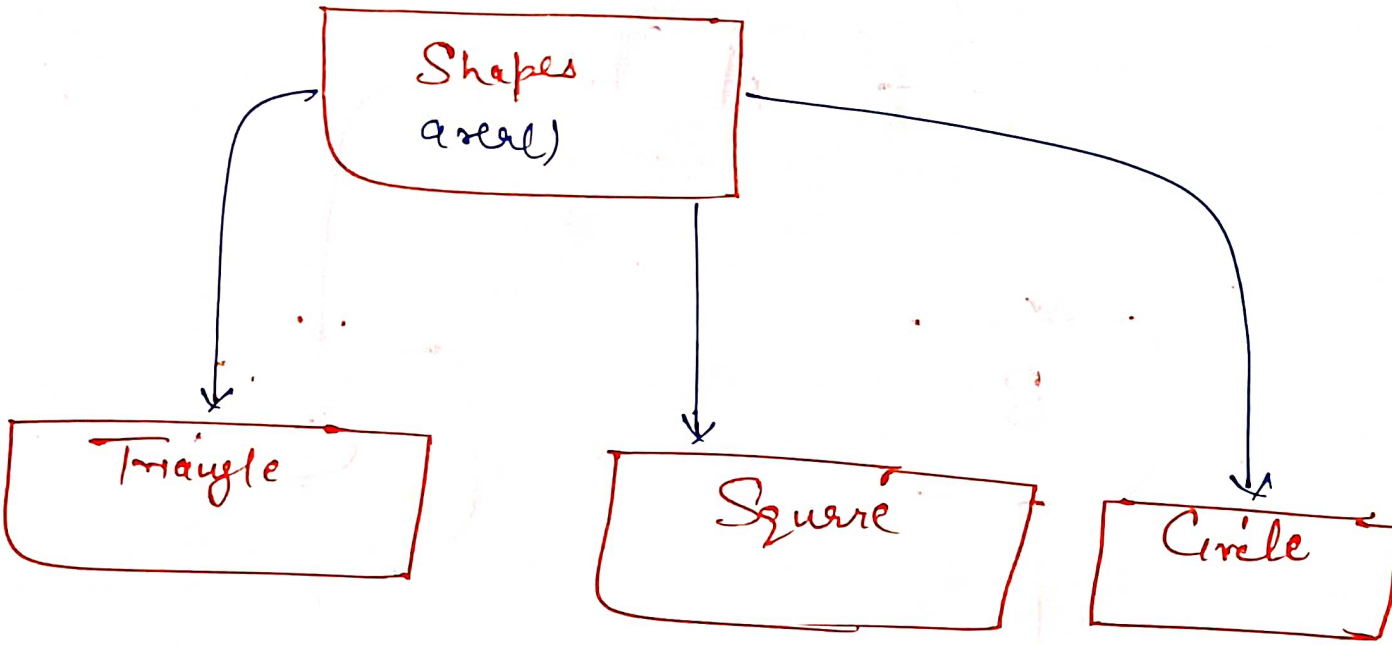
A class doesn't inherit their own class. extends Not Possible.

POLYMORPHISM

Poly + Morphism

Poly → Many

Morphism → Ways to represent.



Types of Polymorphism

(1) Compile Time / Static Polymorphism

Achieved via method overloading.

Same name but types, arguments, return types, ordering can be different.

ex:- Multiple Constructors.

```
A a1 = new A();
A a2 = new A(3, 4);
```

* Operator Overloading does not Supports in Java, but in C++.

Method overloading

```
public class Numbers {
    double sum(double a, int b) {
        return a + b;
    }

    double sum(int a, int b) {
        return a + b;
    }

    int sum(int a, int b, int c) {
        return a+b+c;
    }

    psvm() {
        Numbers obj = new Numbers();
        obj.sum(2,3);
        obj.sum(1,3,7);
    //  obj.sum(4,5,6,8); // Error;
    }
}
```

* Operator Overloading does not Supports in Java, but in C++.

Method overloading

```
public class Numbers {
    double sum(double a, int b) {
        return a + b;
    }

    double sum(int a, int b) {
        return a + b;
    }

    int sum(int a, int b, int c) {
        return a + b + c;
    }

    psvm() {
        Numbers obj = new Numbers();
        obj.sum(2, 3);
        obj.sum(1, 3, 7);
    //  obj.sum(4, 5, 6, 8); // Error;
    }
}
```

(2) Run Time / Dynamic Polymorphism

Achieved by Method overriding.

When a child class has a method name same as a super / Parent class.

→ public class Shapes { Method Overriding

@Override // only for checking purposes.

void area() {

Sout("I am in Shapes");

}

}

* public class Circle extends Shapes {

// this will run when obj of Circle is created.

void area() { // Hence, it is overriding the parent method

Sout("Area is Pie * r * r");

}

}

* public class Triangle extends Shapes {

void area {

Sout ("Area is 0.5 * h * b");

}

}

* public class Square extends Shape {
    void area() {
        Sout ("Area is Square of sides");
    }
}

→ public class Main {
    psvm() {
        Shapes shape = new Shapes();
        Circle circle = new Circle();
        Shapes Square = new Square();
        Shapes circle = new circle();
        Square.area();
    }
}

* Parent obj = new Child();

Here, which method will be called depends on . This is known as Upcasting.

How Overriding works ?

How Java determines this ?

Dynamic method dispatch

Thats, why its called Dynamic Polymorphism.

→ Just a mechanism by which call to an overridden method is resolved at Run Time rather than Compile Time.

→ More about final & static Keyword.

We can also prevent a method to override by adding a 'final' keyword to it

```
// Early Binding
public class Shapes {
    final void area() {
        Sout("I am in shapes");
    }
}
```

→ Not can't be overridden

* Also uses to prevent Inheritance;

↳ By adding final to the Parent class.

like → public final Box {

}

If we declared a class to be final, implicitely, declares their methods final as well.

✱ Static methods

```
static void greeting() {
    System.out.println("Hey, I am in Box class.
                        greetings!");
}
```

box1.greeting();

Also, Box.greeting();

✱ Static methods can be inherited, but not Overridden, Why -! the main class one is always going to be called, it is not dependent on objects / ref. variable.

✱ Overriding depends on Object, Static doesn't depends on Object. Hence Static method doesn't Overrides.

→ ENCAPSULATION

The wrapping up of the implementation of data members in a class.

Hides the code & data in a single entity to hide from outside World.

→ ABSTRACTION

Hiding the unnecessary details, but showing the essential details/information.

ex:- (1) get the element
(2) print the element
(3) start a Car(Key).

* Abstraction focuses on External stuff while Encapsulation focuses on Internal Workings.

* DATA HIDING

Class →

```
public class Box {
    private double l;
    double h;
    double w;

    public double getL() {
        return l;
    }
}
```

Data hiding

getters for getting value of l

* setters used to set values

Main →

```
box1.getL();  → // prints value of l.
```

* Data hiding is focused on Data Security.
* Encapsulation focuses on hiding the complexity of the system.
* Using getters and setters method is known as Encapsulation

Lecture - 4

ACCESS CONTROL MODIFIERS

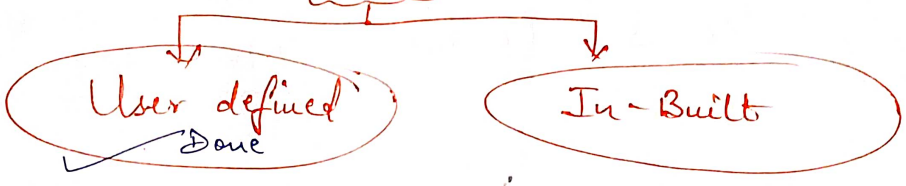
	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World (diff pkg & not subclass)
public	✓	✓	✓	✓	✓
protected	✓	✓	✓	✓	
no modifier	✓	✓	✓		
private	✓				

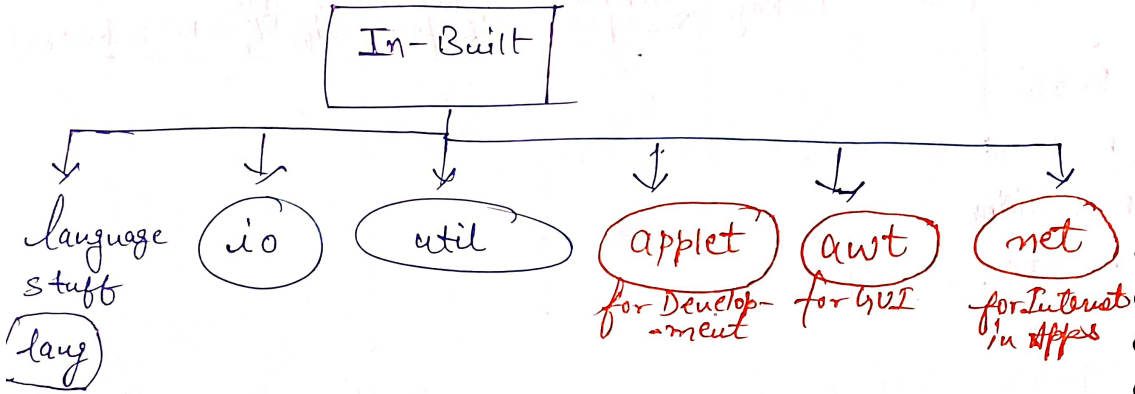
```
public class A {
    private int num;
    String name;
    int[] arr;
    public int getNum() {
        return num;
    }
    public void setNum(int num) {
        this.num = num;
    }
    public A(int num, String name) {
        this.num = num;
        this.name = name;
        this.arr = new int[num];
    }
}
```

```
public class Main {
    psvm() {
        A obj = new A(10, "Kunal");
        obj.getNum();
    }
}
```

→ //Gives 10;

→ More About Packages

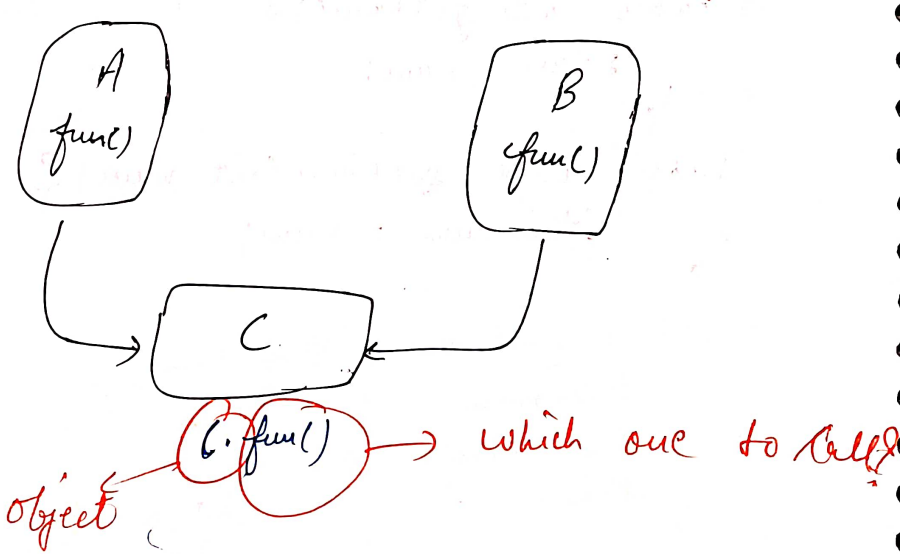


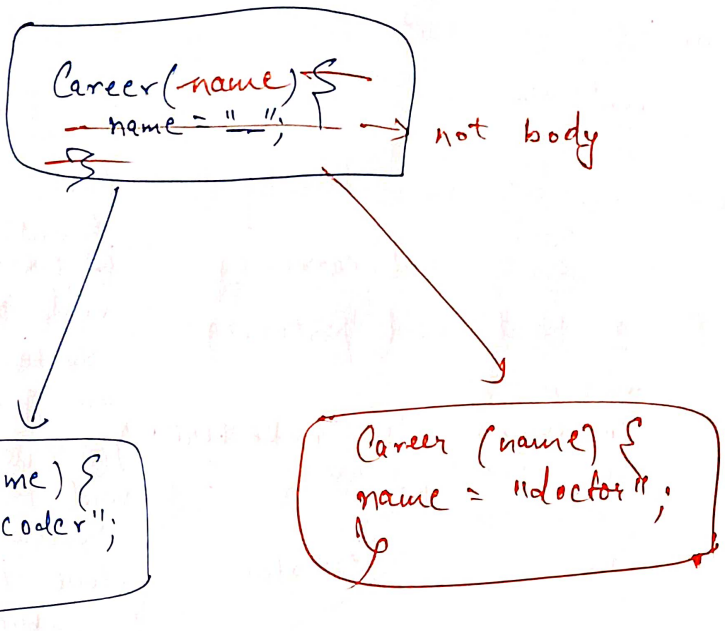


Lecture-5 Abstract Classes

Problems facing in Inheritance;

Actually, in Multiple Inheritance → doesn't support in Java.





- When your function does not have a body depends on the child classes to provide a body for that function.
- How to change the body of a function of a Parent class ? → By overriding function.

```
'abstract' void career (String name);
```

; child classes provide them body.

- Any class contains 1 or more abstract method, it is declared as abstract class.

Code:-

Parent class

```
public abstract class Parents{
    int age;
    abstract void career();
    abstract void partner();
    public S
    Constructor also make here.
    public Parent(int age){
        this.age = age;
    }
}
```

Base class 1

```
Public class Son extends
 Parents {
@ Override
void career(){
 System.out.println("I
am going to a doctor");
}@ Override
void partner(){
 System.out.println("I
love Pepper Pots");
}
    public Son(int age){
        super(age);
        this.age = age;
    }
```

Base class 2

```
public class Daughter extends
      Parents {
@ Override
void career(){
  System.out.println("I am
going to be a coder");
}@ Override
void partner(){
System.out.println("I love
Iron Man");
}
}
```

Main Class

```
public class Main {
    psvm(){
Son son = new Son();
 son.career();
Daughter daughter = new
        Daughter();
daughter.career();
}
```

We can't create objects of an Abstract class. // Error.

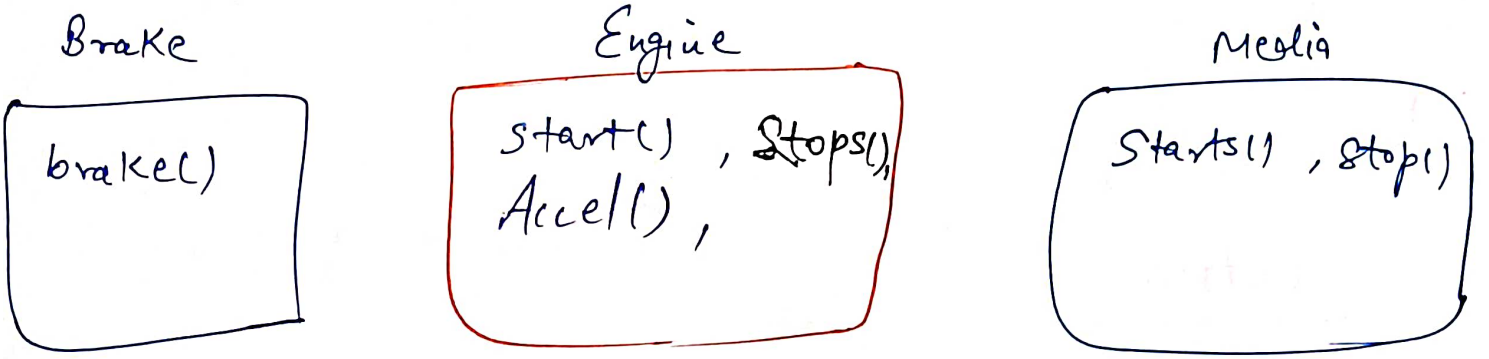
Output

```
I am going to be doctor
I am going to be coder
```

- Also, we cannot make abstract Constructors of abstract classes.
- We also cannot make abstract static method it can't be overridden, because static doesn't needs an object.
- Abstract classes can contain normal methods, yeah, definitely.

✱ Interfaces in Java

Multiple Inheritance can't be Supported. So, we use Interfaces in Java.



Code :-

Interface 1

```
public interface Brake{
    void brake();
}
```

Interface 2

```
public interface Engine {
    void start();
    void stop();
    void acc();
```

Interface 3

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

Main Class

```
public class Main{
    psvm()
    Car car = new Car();
    car.acc();
    car.start();
    car.stop();
```

Car class

```
public class Car implements Engine, Brake, Media {
    @Override
    public void brake() {
        Sout("I brake like a normal Car");
    }
    @Override  ---------> Annotations.
    public void start() {
        Sout("I start like a normal Car");
    }
    @Override
    public void stop() {
        Sout("I stop like a normal Car");
    }
}
```

Output

```
I brake like a normal Car.
I start like a normal Car.
I stop like a normal Car.
```

* ## Nested Interfaces

```
public class A {
    //nested interface
    public interface NestedInterface {
        boolean Isodd(int num);
    }
}
```

```
class B implements A.NestedInterface {
    public boolean isOdd(int num) {
        return (num & 1) == 1;
    }
}

class Main {
    psvm() {
        B obj = new B();
        Sout(obj.isOdd(5));
    }
}
```

// Output
// True

Lecture - 6

Custom ArrayList

```
package com.kunal.generics;
import java.util.ArrayList;
import java.util.Arrays;

public class CustomArrayList {
    private int[] data;
    private static int DEFAULT_SIZE = 10;
    private int size = 0; // also working as index value

public CustomArrayList() {
    this.data = new int[DEFAULT_SIZE];
}
```

```
public void add (int num) {
        if (isFull()) {
            resize();
        }
        data[size++] = num;
}
private void resize() {
        int[] temp = new int[data.length * 2];
    //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;
}
public int get(int index) {
        return data[index];
}
public int size() {
        return size;
}
```

```
public void set(int index, int value) {
    data[index] = value;
}
```

→ `@override public String toString() { ---- }`

```
public static void main(String[] args) {
    ArrayList list = new ArrayList();
    CustomArrayList list = new CustomArrayList();
    list.add(3);
    list.add(5);
    list.add(9);
    System.out.println(list);
}
}
```

(internal ArrayList)

o/p

CustomArrayList {data = [3, 5, 9, 0, 0, 0, 0, 0, 0, 0], size = 3}

* Only creating for single datatype like int, If can't be reuse it.

✱ Object Comparison

Student.java

```
public class Student implements Comparable<Student> {
    int rollno;
    float marks;
    public Student(int rollno, float marks) {
        this.rollno = rollno;
        this.marks = marks;
    }
```

```
@Override
public String toString() {
    return marks + " ";
}
@Override
public int compareTo(Student o) {
    int diff = (int) (this.marks - o.marks);
    // if diff == 0; means both are equal;
    // if diff <= 0; means o is bigger, else o is
    return diff;                    smaller;
}
}
```

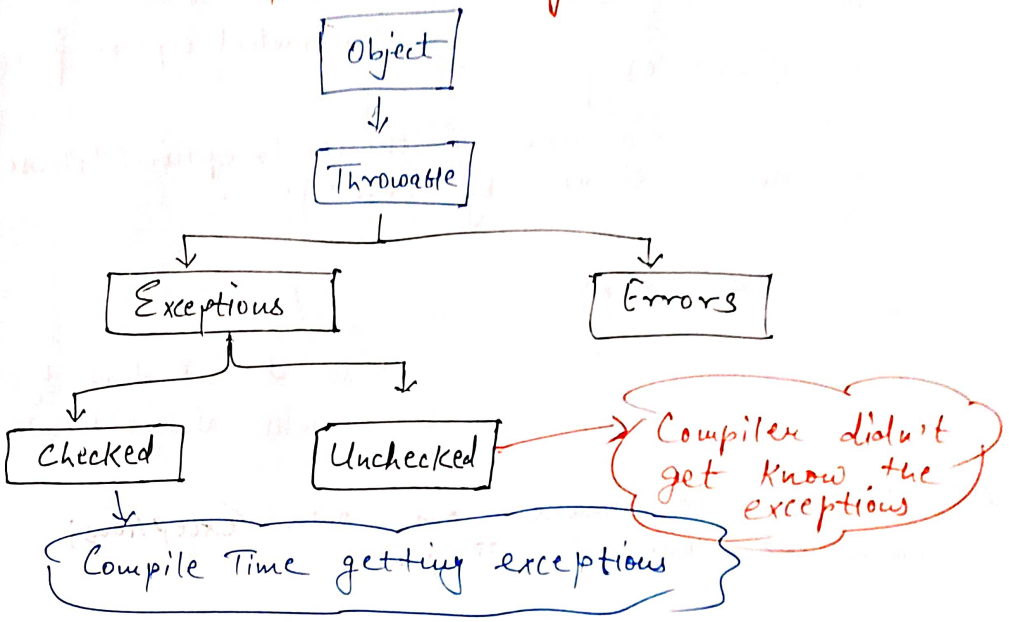
Main.Java

```
public class Main {
    psvm() {
        Student rahul = new Student(5, 99.52f);
        Student kunal = new Student(12, 89.71f);
        Student[] list = {kunal, rahul};
        Sout(Arrays.toString(list));
        Arrays.sort(list);
        Sout(Arrays.toString(list));
        if (kunal.compareTo(rahul) < 0) {
            Sout(kunal.compareTo(rahul));
            Sout("Rahul has more marks");
        }
    }
}
```

O/P

```
-9
Rahul has more marks;
```

→ Exception Handling



Code:-

```
public class Main {
    public static void main() {
        int a = 5;
        int b = 0;
        try {
            diviode (a, b);
        } catch (Arithmetic Exception e) {
            Sout (e.getMessage());
        }
        catch (Exception e) {
            Sout ("normal exception");
        }
        finally {
            Sout ("This will always execute");
        }
    }
```

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

O/P

Please do not divide by zero
This will always execute.

* We can also make our own Exceptions;
Check out code in IDE;

→ Object Cloning

Human.java

```
public class Human implements
                    Cloneable {
    int age;
    String name;
public Human(int age, String name)
    { this.age = age;
      this.name = name;
    }public Human(Human other)
      this.age = other.age;
      this.name = other.name;
    }
public Object clone() throws CloneNotSupportedException {
    return super.clone();
}
}
```

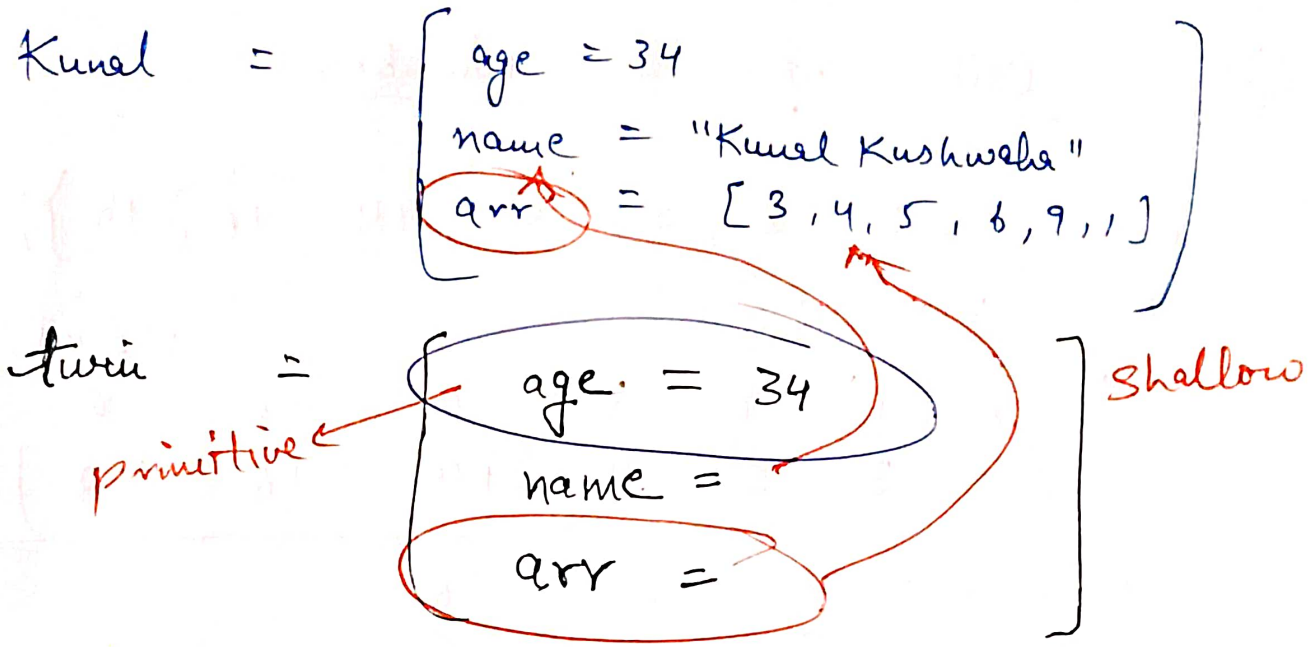
Main.java

```
public class Main {
    psvm() throws CloneNot
    SupportedException {
Human kunal = new Human(34,
                "Kunal Kushwaha");
// Human twin = new Human
                (kunal);
// Cloning kunal object here;
Human twin = (Human) kunal.
                clone();
Sout(twin.age + " " + twin.name);
```

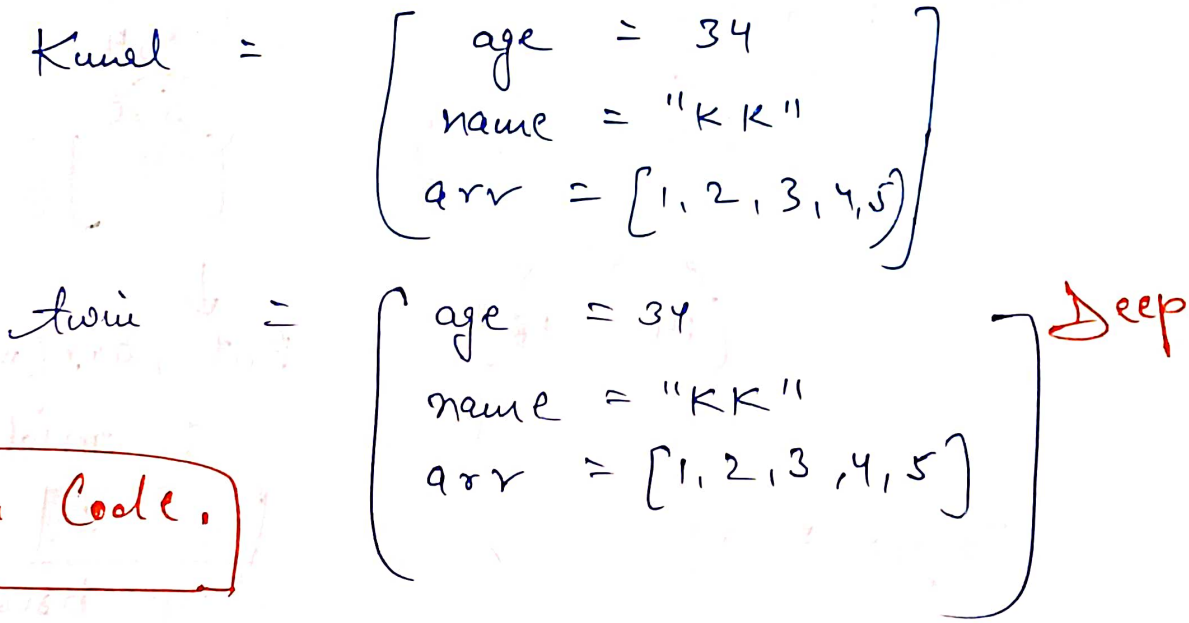
O/P

34 Kunal Kushwaha

→ Shallow Copying



→ Deep Copying



More in Code.