⇒ Polymorphism

→ upcasting

→ downcasting

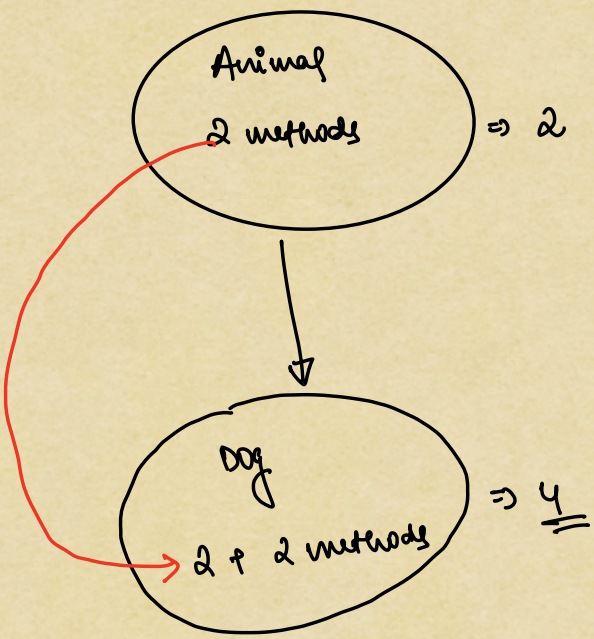
⇒ Interfaces

⇒ Abstract class

⇒ Static keyword.

⇒ Encapsulation

⇒ Polymorphism



Animal a = new Animal();

a.m1() a.m2()

```
Dog d = new Dog();

d.m1()
d.m2()
.
.
.
```

upcasting

```
Animal aObj = new Dog();
```

aObj.

ref variable
parent

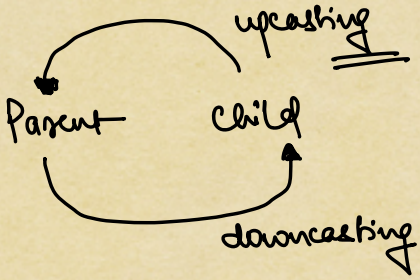
Object
child

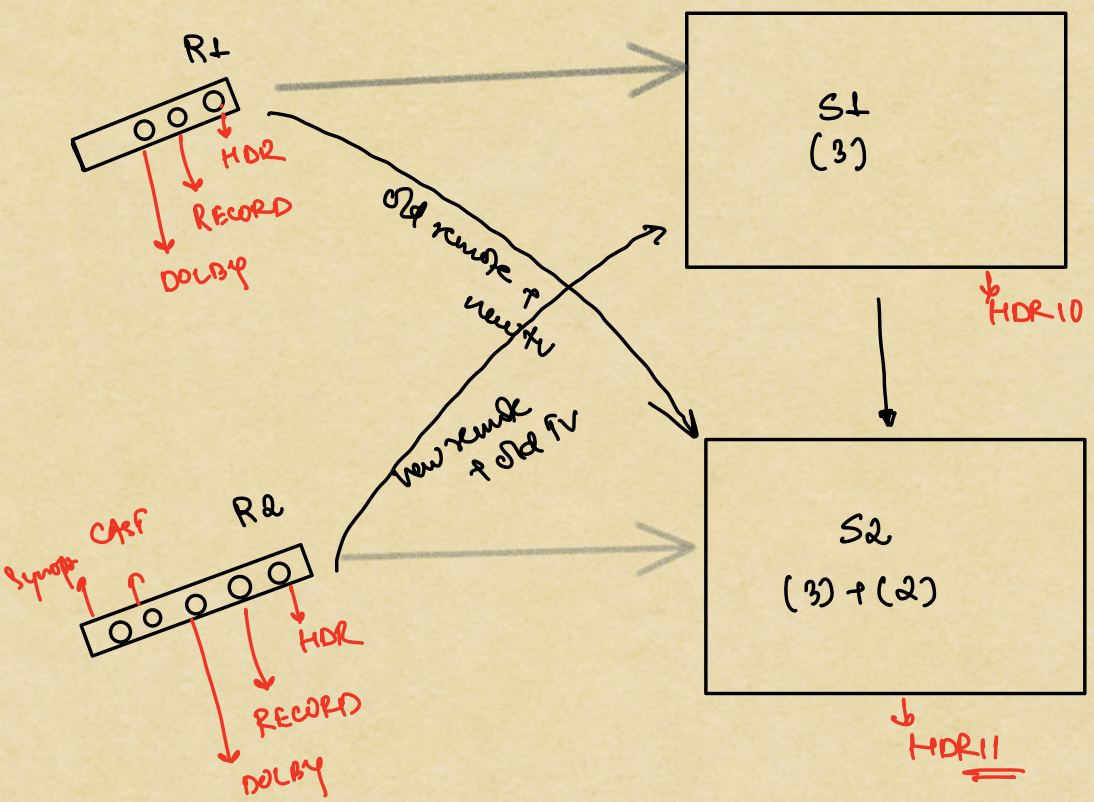
downcasting

```
Dog dObj = new Animal();
```

ref variable
child

Object
parent



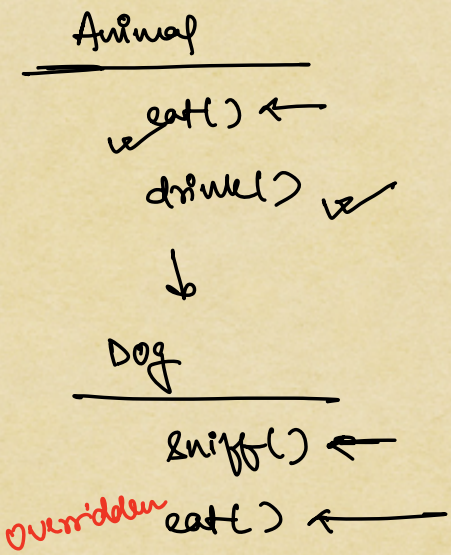


⇒ Old remote + new TV

* existing func. (buttons on remote) they all work, no problem

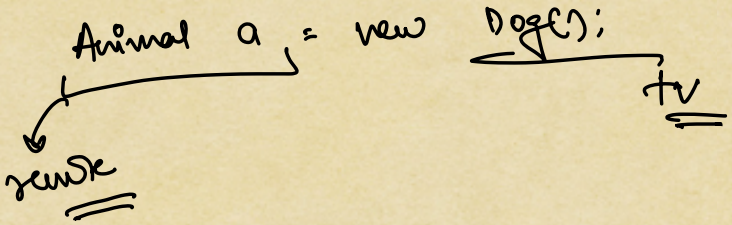
* wont be able to use the new features on the new TV.

* features would execute on newer TV as per their implementation.



Animal ⇒ 2

Dog ⇒ 3



a.drink(); ✓

a.eat(); ✓ → overridden

a.sniff(); ⇒ XX

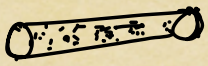
* newer remote & old TV:

→ newer remote has extra buttons

→ those extra buttons are unreliable and prone to errors, hence, it is not advisable to use new remote with old remote.

=> Encapsulation

encapsulate => capsule



↓

protects and stores the medicine inside the pack → capsule

↓

class

↓

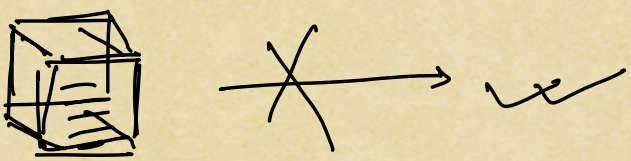
(attributes
method)

↓

protection = data hiding

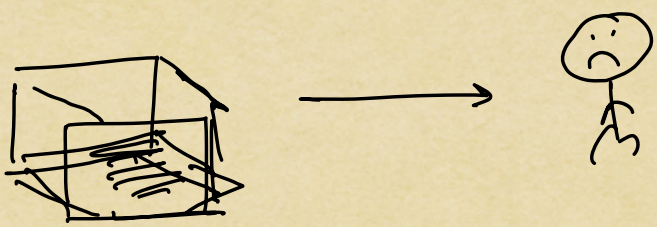
Subham Aman

1)

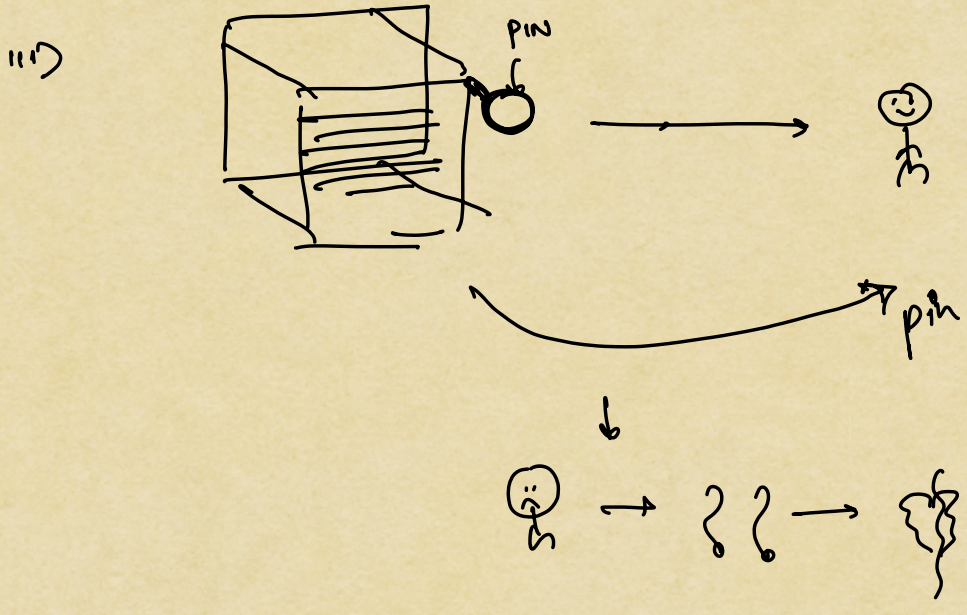


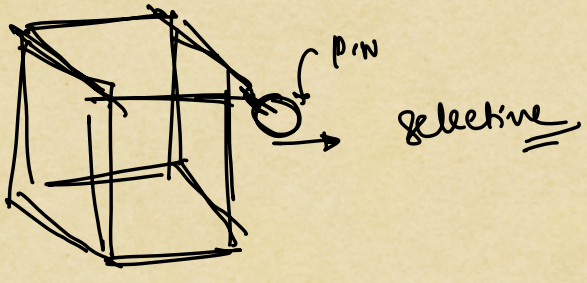
object => public

2)



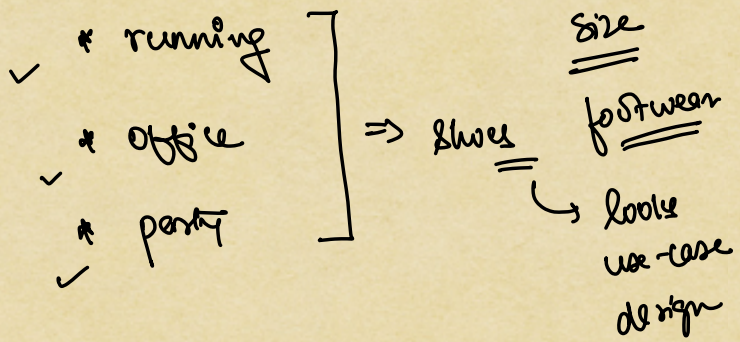
↓
attribute => private



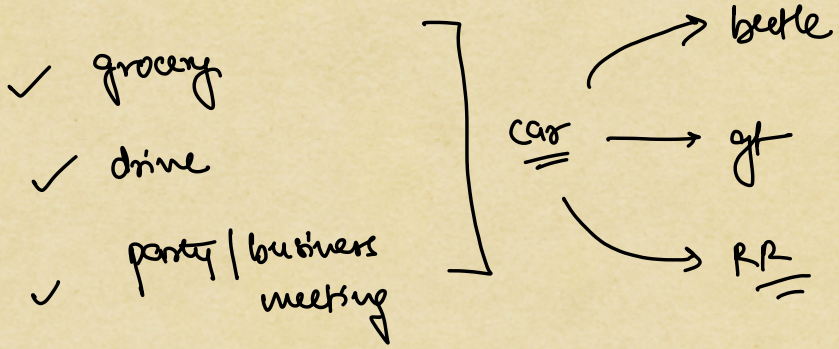


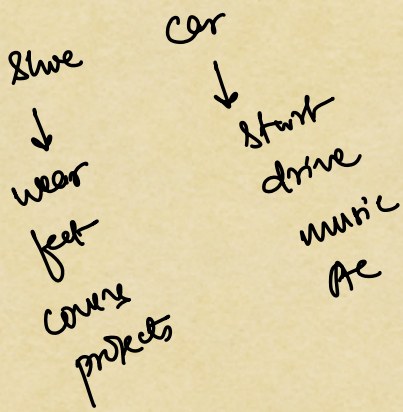
1) make all attributes private
2) bypass access to them via getters | setters.

* Interfaces


* running
* office
* party
⇒ Shoes
size
footwear
looks
use-case
design


type ⇒ Shoe


grocery
drive
party | business meeting
car
beetle
gt
RR


Shoe
wear
feet
covers
pockets
car
start
drive
music
AC

Interface → ~~How~~

↓

What

DSA — Stacks & queues

↓

array also, LinkedList

```
interface Car {
    void startEngine();
    void startAC()
    boolean getTemp(int temp);
    int getTemp();
}

class ElectricCar implements Car {
}
```

↓

forced to implement all abstract methods in Car

* class → object ✓

* method

↓

O.method() ✓

* method → abstract

O.method()

→ object creation should not happen whenever we will abstract methods

↓

* Interface cant have object

=> multiple inheritance ? => NO.

behaviour

↓

Inheritance

↓

class A extends B, C

}

class A extends B implements I1, I2, I3 {

}

↓

class in java, extend only 1 class but it can implement many interfaces

```
class A {
    run() {
        10
    }
}
```

```
interface I {
    run();
}
```

```
class B extends A implements I {
    run() {
        20;
    }
}
```

```
A a = new B();
a.run();
```

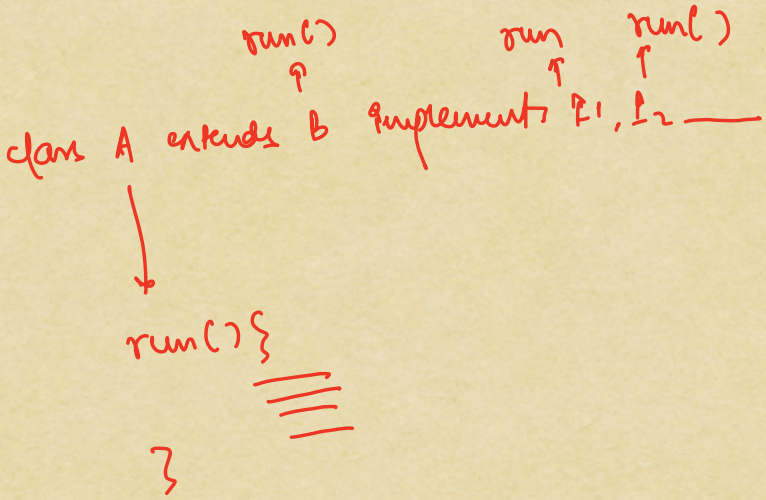
20 ✓

```
I i = new B();
i.run();
```

20

```
B b = new B();
b.run();
```

20 ✓



⇒ Abstract class

class that can have abstract methods

↓

both abstract & concrete

or.

attributes

* abstract class can't have object
* abstract class has constructors → chaining