# LLD 2 — OOP Terminologies

4 Pillars of OOP

## Agenda

① Terminology related to OOP

- Abstraction
- Encapsulation
- Polymorphism
- Inheritance

② Interfaces → Focus

③ Principles of good Software

↳ Maintainability

Max level of abstraction

2 DSA
1 LLD
1 HLD

IDEA
LLD + HLD
1 ML Kno

Spring Boot
Monday

Frontend Eng

Extensibility

Try to very sarely use inheritance

→ Code to interfaces not implemen

⇒ Liskov Substitution ⇒ No 2 classes directly

## Terms Related to OOP

1 Principle of OOP ⇒ ABSTRACTION

3 Pillars of OOP ⇒ Encapsulation
Inheritance
Polymorphism

Abstraction

Design ⇒ Value System | Fundamentals on which

Principle -> Value system / philosophy on which OOP is based

Pillars => Language constructs that allow to bring principles in reality

1. Encapsulation
2. Inheritance
3. Polymorphism

---

ABSTRACTION : Making something abstract

Abstract

Something called

Idea

→ Think of a complex software system in the terms of different ideas

Animal
Dog
Cat
Wild Animal
Pet Animal

↓

Set of ideas that are help us the software system

⇒ to make it easy for us to Make sense of the system

Car Mechanic

→ relative to a context

Scala.

vs

Universe

Student

Student

→ idea represents everything about it in the context AND nothing else

## Abstraction

→ Principle of rep a complex system in

terms of a set of ideas which is the core of OOP

⇒ Model real world objects / its behaviours in terms of code:

- limited to context

→ include everything for that context

→ power usage

1. Encapsulation
2. Inheritance
3. Polymorphism

Encapsulation

⇒ Contains stuff together

Avoid interaction from outside

In OOP: Encapsulation refers to the idea of storing all the attributes and behaviours of an idea / entity together

Animal

- age
- color
- name
- walk()
- eat()
- sleep()

Access Modifiers

- public
- private
- protected
- default

(Contain)

① Hide the details of the System

② Protect from illegitimate access

Abs

① Abs is a principle on which OOP is based

② Help a system as Multiple ideas interacting with each other

① Enc is a way to achieve abstraction

② Hold the data & attributes of an code together to prevent from illegitimate access

Entity : Class

Behaviour : Interface

Not fully formed : Abstract Class

→ Every idea is not really a completely independent / isolated thing

→ Some kind of sharing of properties between ideas happen within those ideas

Inheritance ⇒ represent hierarchy of ideas in my software system

⇒ Share common attributes and behaviours b/w different ideas.

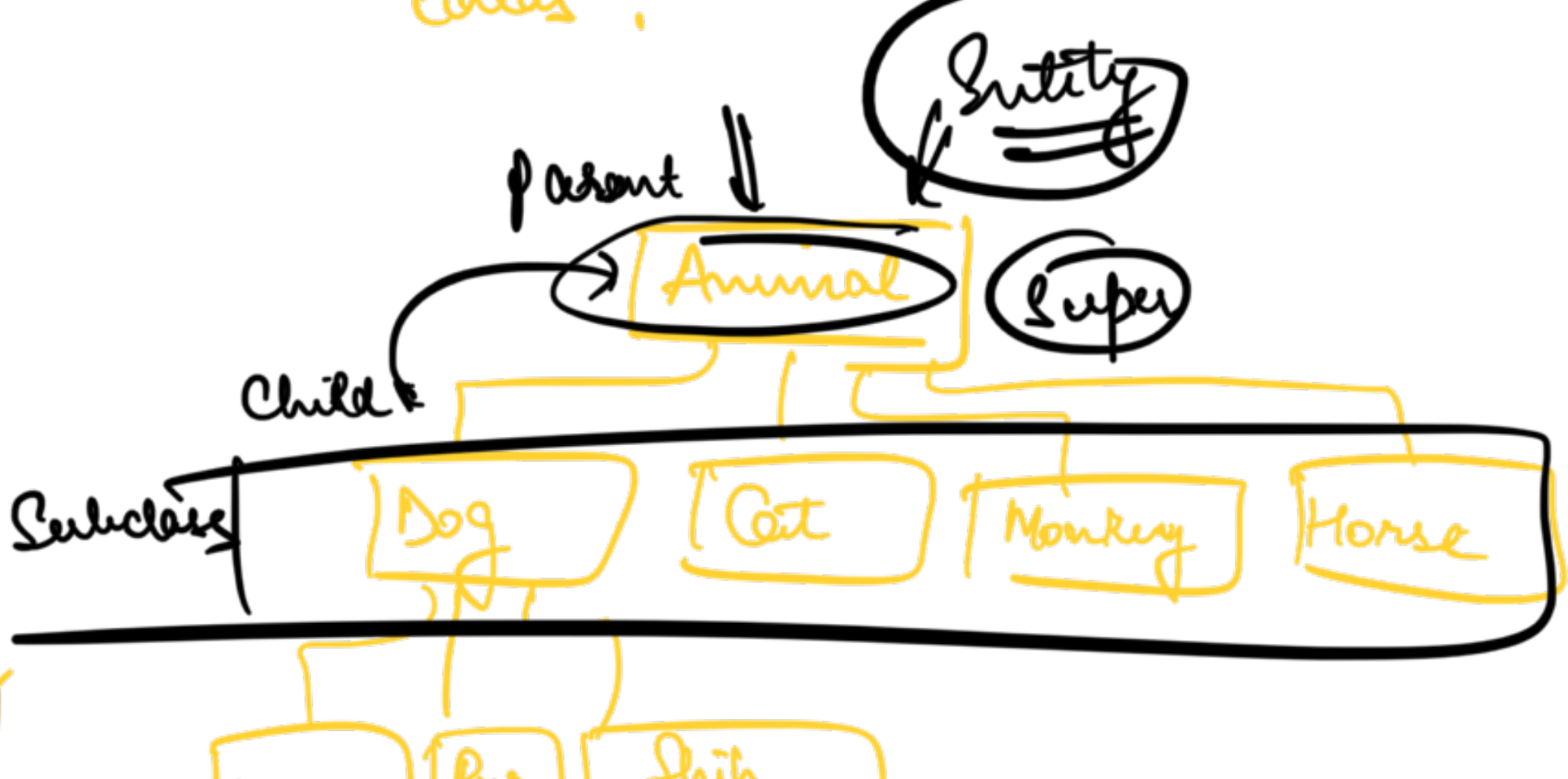
Zoo

Animal

Dog

Cat

Monkey



Lab

Shih Tzu

Pug

Child gets all the properties

And behaviours of the parent

① parent / Child

② superclass | subclass

Liskov's Substitution Principle

We Override a method and give it a diff purpose of parent


⇐ Anyone who rep this ... will have this

Dog

hunt(){

throw Exception

Cat

Tiger

Client has to check on return value

Overriding

→ Every method has a body that is shared with the clients of the entity in the db currentaly

↳ do

↳ two

↳ constraints

Animal
Name, color
run()

IO behaviour

Hunting Animal
hunt()

Non Hunting Animal

$2^1 = 2$

Cat

Tiger

Dog

Animal

Hunting Running Animal

hunt()

run()

Hunting Not Running Animal

hunt()

Not Hunting Running

run()

Not Hunt Not Run

$2^2 = 4$

all animals that

Cat | Tiger | Dog

can hunt?

Some Methods ( )

HuntingRunning
HuntingNotRun

$2^{10} = 1024$

✓

① A massive # of classes to implement

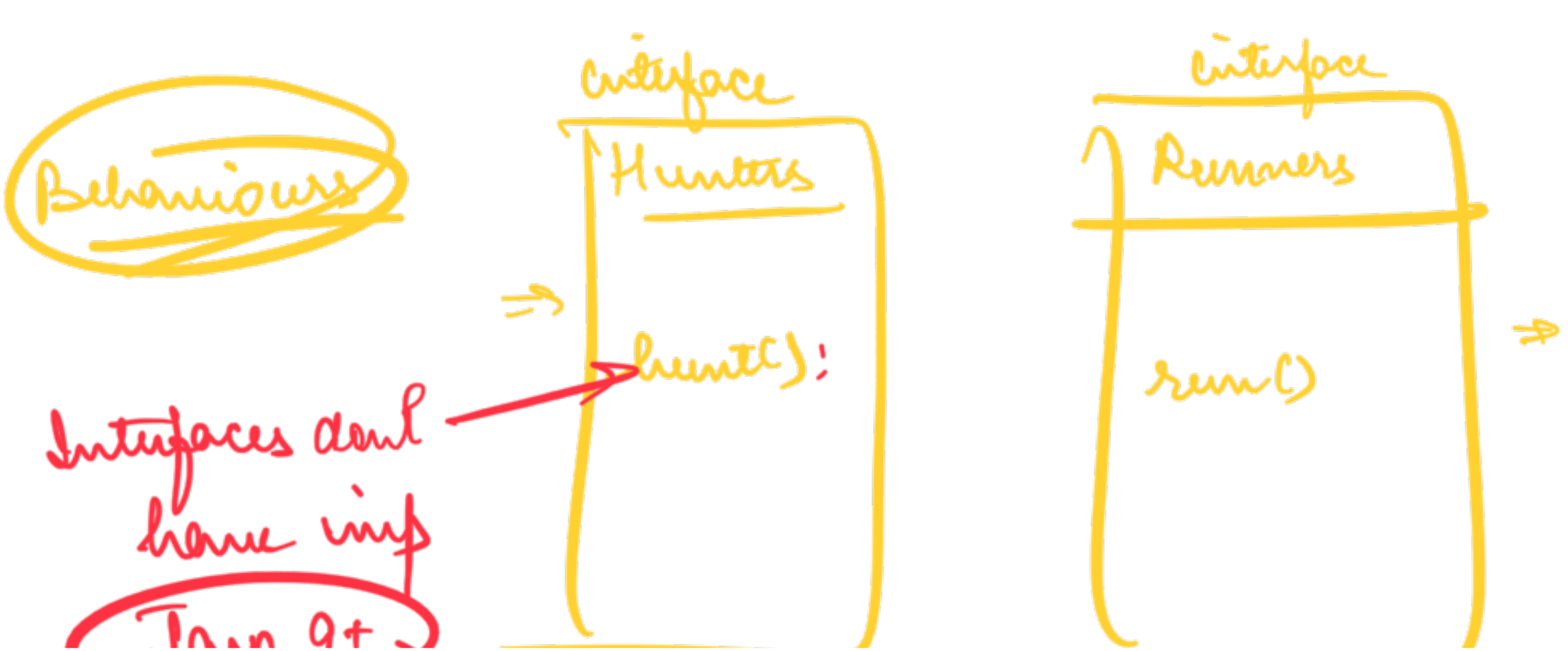
② What data type to give as parameter

SomeMethod (Animal)

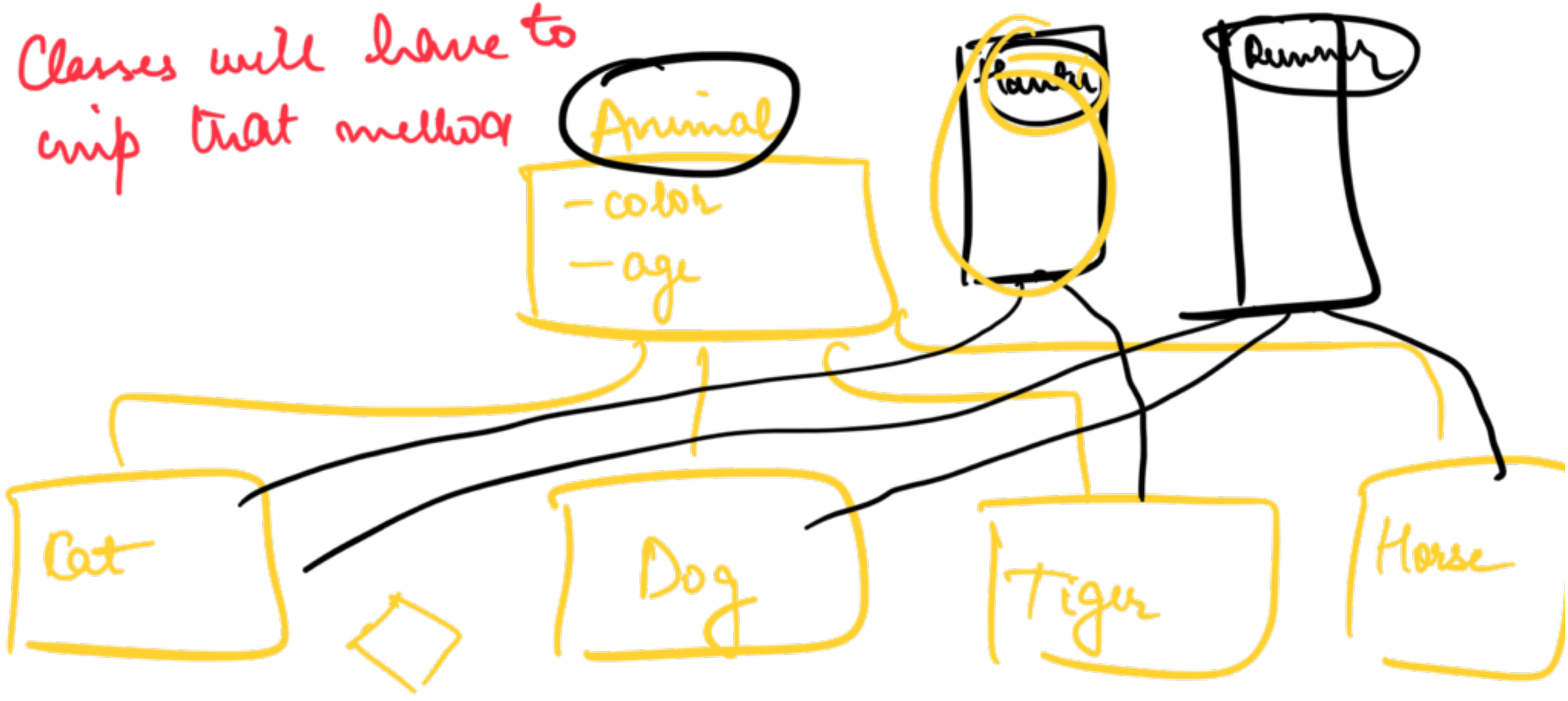
if Data.hunts

Interfaces : Blueprint of a behaviour

Entity ⇒ Attribute ⇒ IFF it is a physical implementation property of that entity



→ Classes will have to
implement that method




class Cat extends Animal implements Hunter, Runner {

∈ only extend one class

∞ # of interfaces

Some Method (Hunter

9

3

2 to 3 classes ⇒ 10 interfaces

Name of interface which provides
that Method

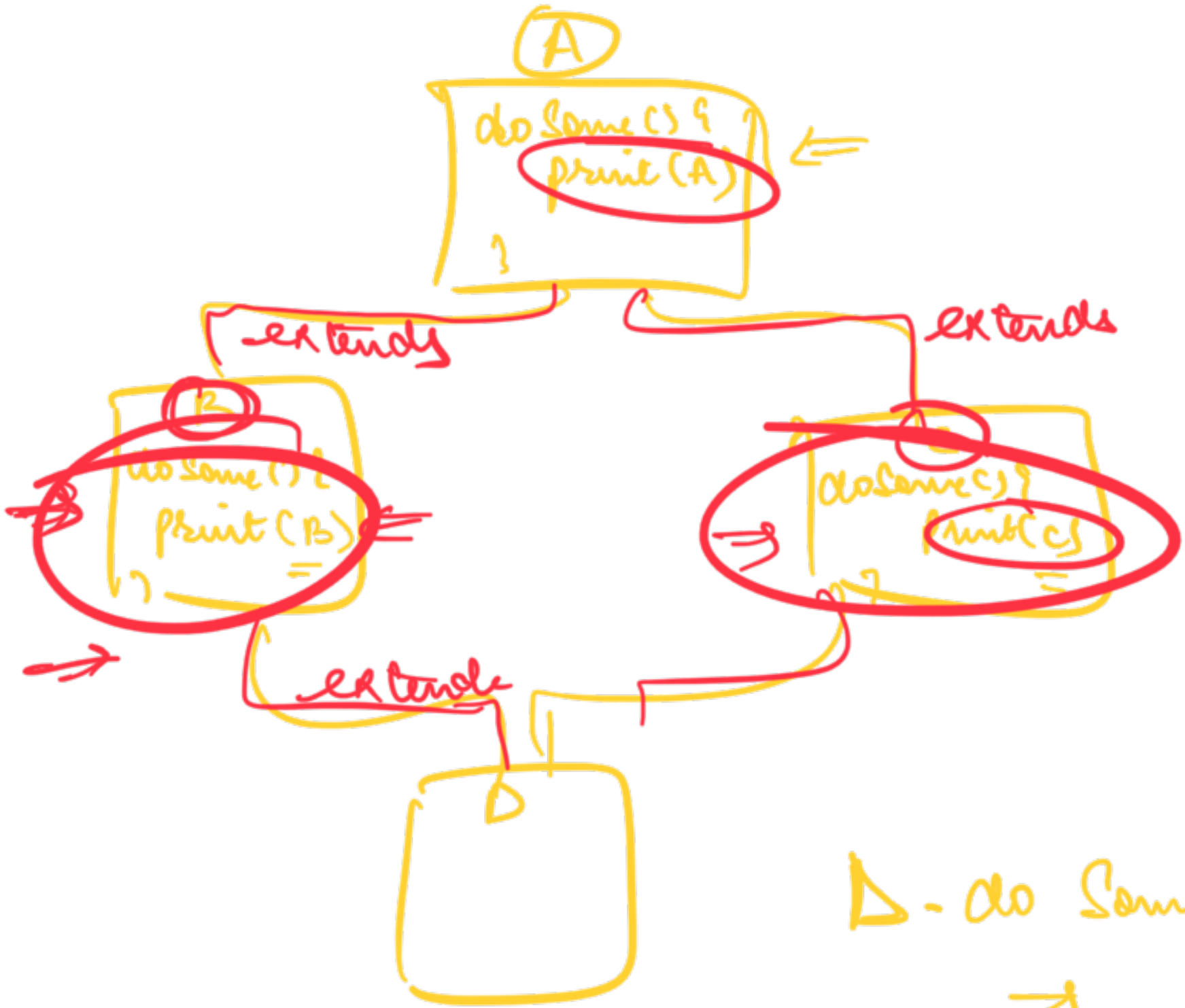
Start Race (List<Runner> ) {

}

⇒ Don't classify on the basis of behaviour

⇒ Only include those attributes & behaviours in a class diagram that are guarantee to be also present in every subclass. (part of physical reality)

Diamond Problem → Why many prog languages dont allow multiple inheritance.




Class D extends B, C

D. doSome()

① If we start classifying based on behaviours → end up creating $2^x$ classes to rep diff combination of behaviour

② doSomething(Hunter hunt)

↑

Any obj that supports hunt()

( Hunting Running Ant )

( Hunting Not Ant )

⇒ If clients have to hard code based on return values ✗


Client {

do Something ( ) {

}

}

A {

do Some() {

}

}

B extends A {

@Override

Compose a dependency not a property

do Some (Student s)

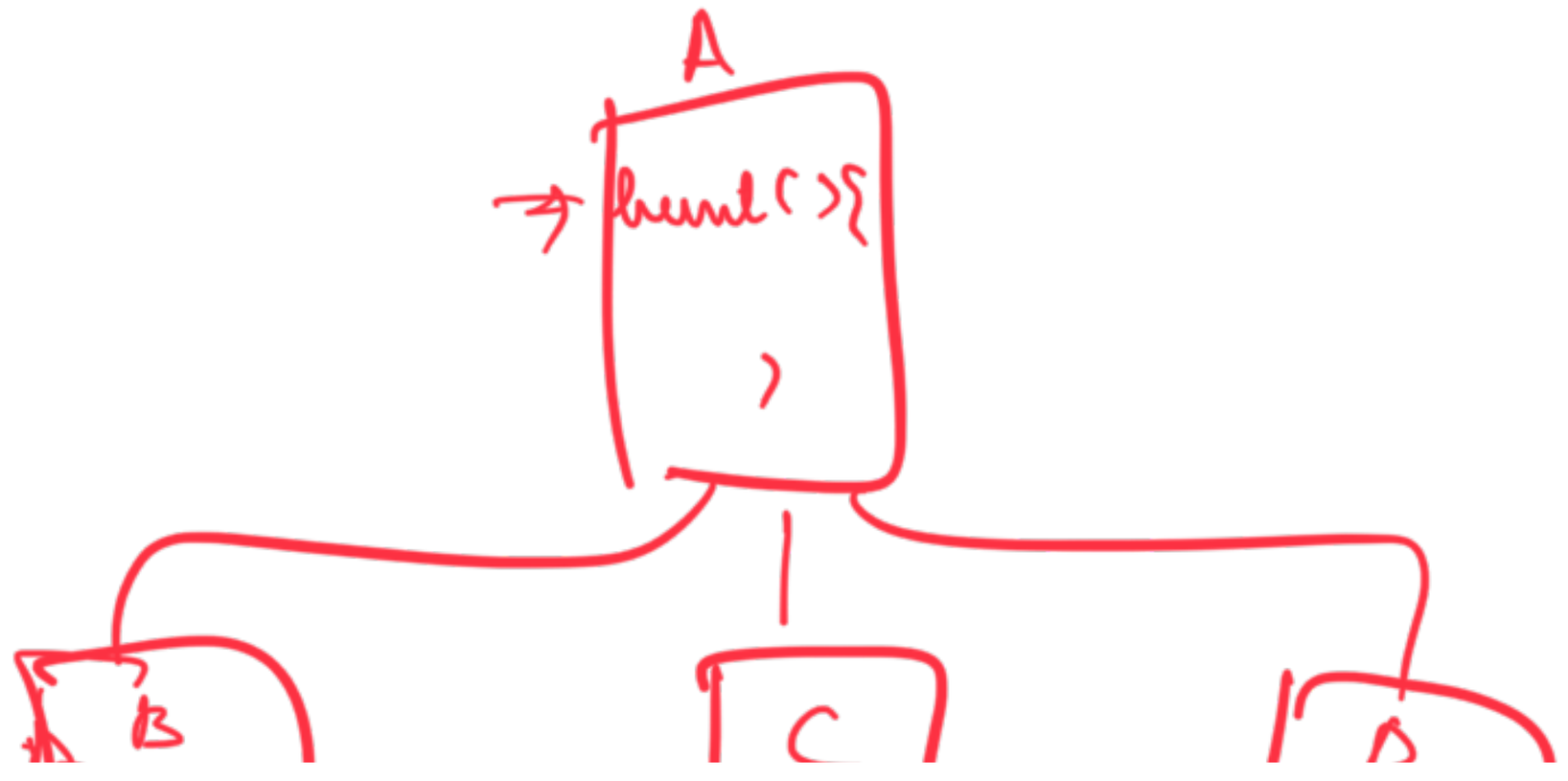
⇒ overloading

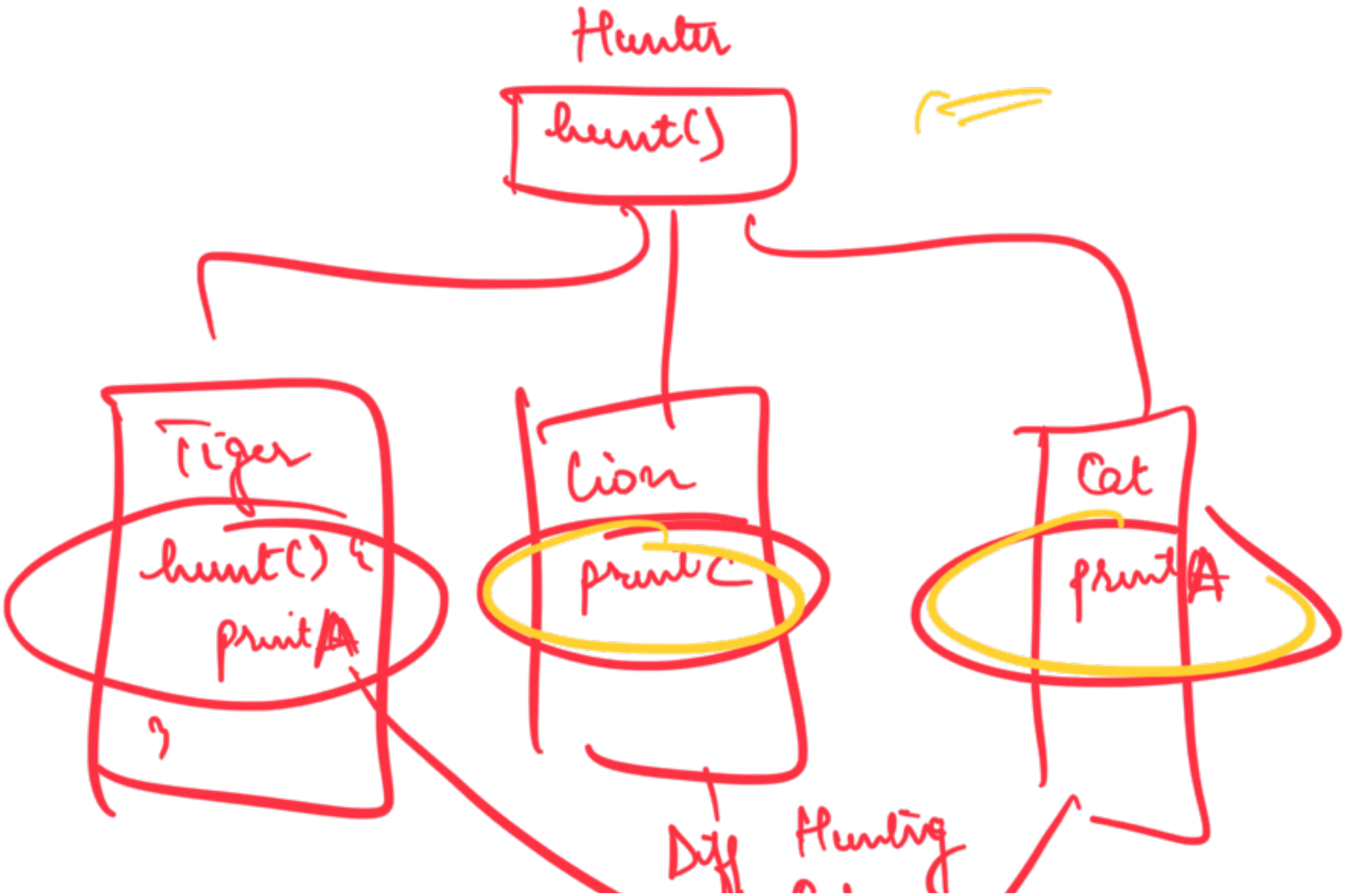
}

}

---

1 interface → 100 class.




Hunter
hunt()
Tiger
hunt() {
printA
}
Lion
printC
Cat
printA
Diff Hunting

entity implementing can hunt

interface

Animal

Hunter
huntC()

Hunting Behavior

how will a hunt happ

Hunting Behavio

→ ExecuteHuntC

Tiger

Cat

Lion

Horse

class

Cat
Hunting
Behavior
huntC(){
printA
}

class

LionHunting
Behavio

huntC(){
printC

behaviour

}

Prefer composition
over inheritance

class Tiger extends Animal implements Hunter {

Composing

CatHuntingBehavior h = new CatHuntingBehavior();

hunt() {

h.execute();

}

}

You had to share code

① Put code in a super class and

all subclasses will share it




② Create a separate class which has the code
Put an attribute of that class in your class.
↳ Call the method of that class

Composition over inheritance

Physical Puzzles

print ()

Command Line
print ()

---

Polymorphism

Many

Form

⇒ We don't really care about the BRACT class that will be there on the runtime

Animal Race {

run (List<Animal> )

}

Dog dl = new Dog()

List<Animal> animals;

IS-A animals.add(dog);

Dog Cat Horse




maxI(int a, int b)

max (float a, float b)

max2 (long a, long b)

max( )

I don't need to know the exact Class of a particular object.

Animal → Animal

Animal → Cat

Dog

Animal {

hunt() {
print A
}
}

Cat extends Animal {
hunt() {
}
}

Client {
new Animal()
new Cat()
Animal a =
a.hunt()
}

print A

}

}

Animal a = new Cat();

① a is an attribute in which you can put Animal

② Cat is a animal

③ You can put cat in a

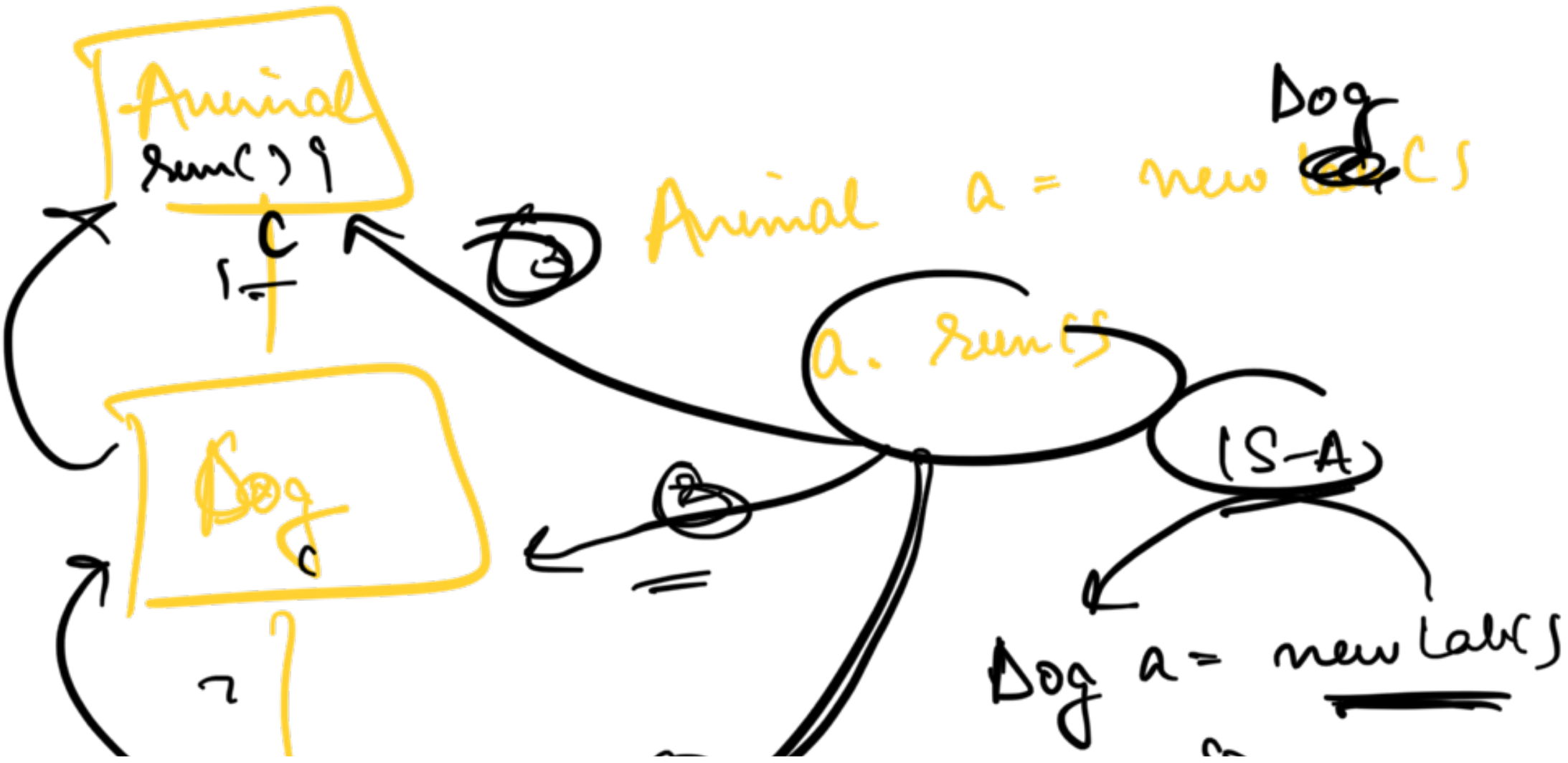
Cat c = new Animal();



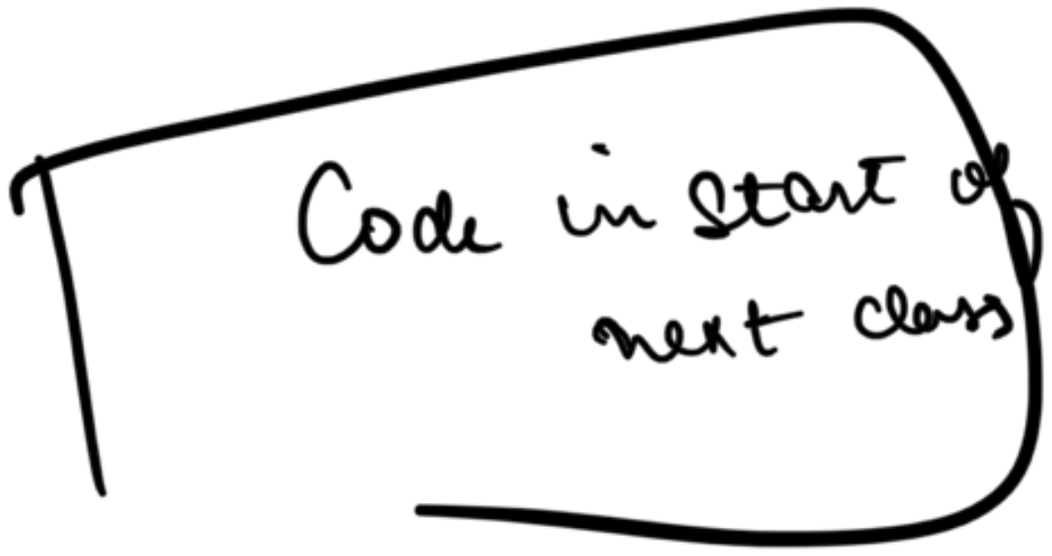

① when you don't know exact class of

an object at runtime

② you don't know exact method that will be called till runtime

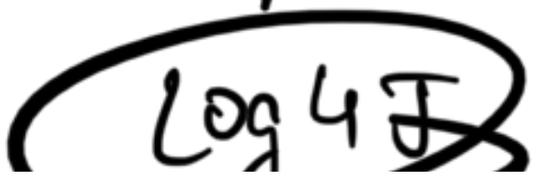







# Maintainability VS Extensability

Our software often uses multiple dependencies



No new feature/ business requirement But you still need to ensure your software is in Sync with latest libraries etc → Maintanence

## Extensability

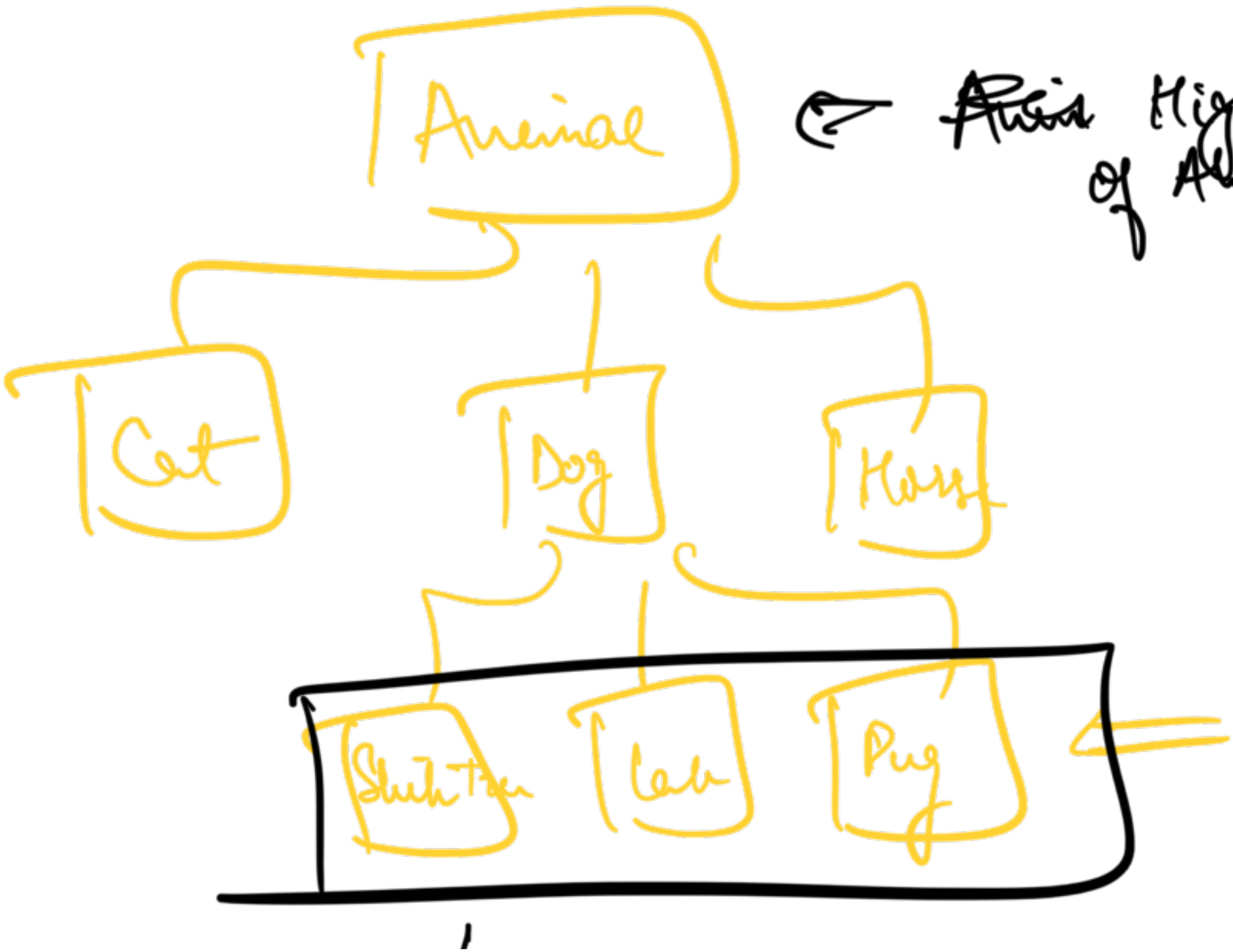
When your software is able to easily add new features

Log 4J


Level of Abstraction → Idea

Higher Level → Bigger Idea / More General Idea

Lower Level → Smaller Idea / Specific Idea




← Highest Level of All

Lowest level of obstruction

Next to next