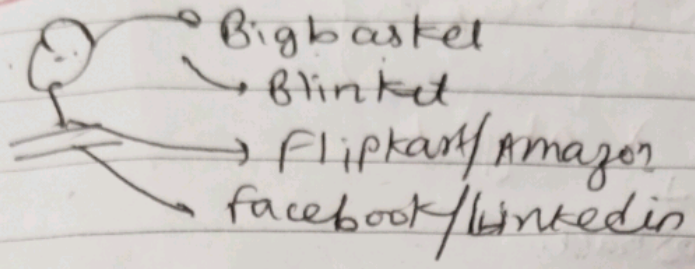
OOPs

what and why?



Company → Real world Problems

Represent

Object Oriented programming

[object] → state, behaviour

OOPs → Procedural programming

C → S1 → Boolean decision

S2

S3

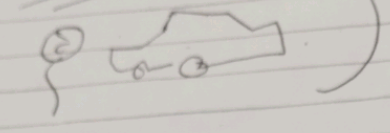
S4 → Done

flowchart

Real world } 0/1 yes/no } Yeh nhi hota.

↓ State/Behaviour/Relationship

Real world



} Programming Objects.

(2) Relationship :- → important + tough

→ programming represent

→ Objected oriented principle

↑

Object Oriented principles }

(1) Class & object

↓

Abstraction
Encapsulation
Inheritance
Polymorphism.

* Class & object } Related to eachother }

Class → Type }

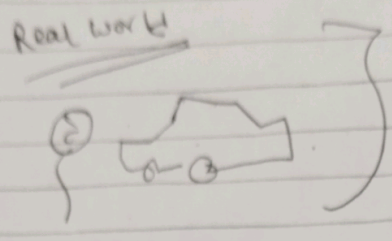
Template

Objects.

Real world } 0/1 yes/no } Yeh nhi hota.

↓

State/Behaviour/relationship



(2) Relationship :- → Important → Logic.

→ programming representation

→ Object oriented principle

↑

Object Oriented principles }

(1) Class & object

↓

Abstraction

Encapsulation

Inheritance

Polymorphism.

* Class & object } Related to eachother }

Class → Type }

Template

Objects.

class → Person { name, age, phone } State.
dance()

name = Shivam;
age = 31
ph = 9171239 XX
dance() => dance like Korean

name = Vishwa
age = 99
Phone = 9171239 9 XX
dance() → very bad !!!

[object]

class → template

object.
instance of the class.

name
age
→ programming

class Person {
String name;
int age;
} Template

name = Vishwa
age → 19

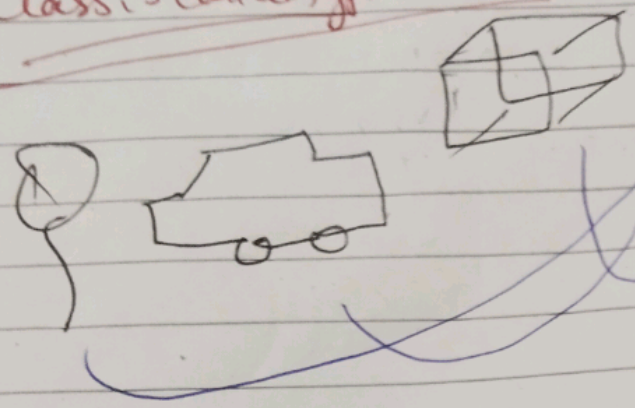
name → Shalini
age → 32.

Object

Person Vishwa = new Person();
Vishwa.name = "Vishwa";
Vishwa.age = 19

Person shalini = new Person();
Shalini.name = "shalini";
Shalini.age = 99

Class is called type.



Programmer
Person class
Car class
Box class
obj obj2 obj3

Object
↓
Type
↓
Box

Object is called as instance

Human → name, age

Instance

God → Class

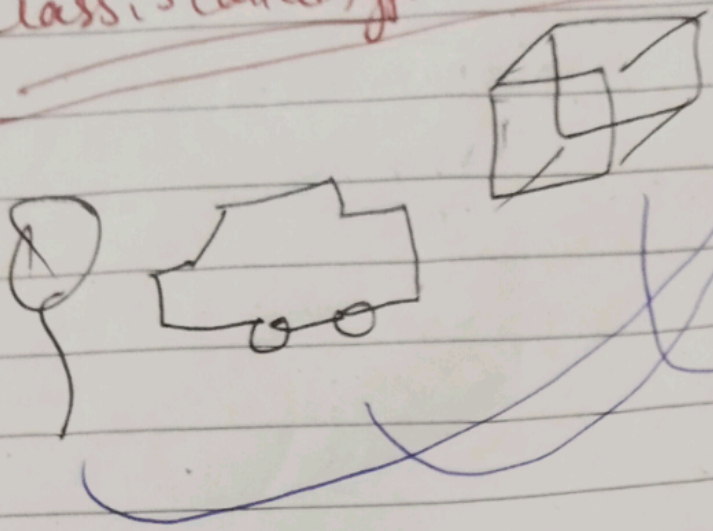
Person
name
age

Person1 obj
name - Lits
age - 25

Person-2 obj
name - sita
age - 23

No of object = Total Memory / Size of 1 object

Class is called type.



Programmer
Person class
Car class
Box class

obj1 obj2 obj3

Objects
↓
Type
↓
Box

Object is called as instance

Human

name
age

God

Class

Person
name
age

Instance

person

person1 obj
name - Lits
age - 25

person-2 obj
name - sita
age - 19

No of object = Total Memory / Size of 1 object

OOPs → Real world problems

Class And object

Abstraction → Abstract → Summary

→ Hiding unnecessary details.

Research paper → Abstract → Hiding unimportant things

Advantages of abstraction

① Loosely coupled applications / improvised the system without impacting the Customers

② Security / → Hidden unnecessary details.

Encapsulation

→ Binding data & method together

Name - Vishnu
Car - BMW

Nobody should be able to change the state of an object directly.

```
Class person {
private String name;
private Car car;
}
```

OOPs
↳ Real world problems

Class And object

Abstraction → Abstract → Summary

→ Hiding unnecessary details.

Research paper → Abstract

Hiding unimportant things

Advantages of abstraction

① Loosely coupled Applications / improvised the system without impacting the Customers

② Security / → Hidden unnecessary details.

Encapsulation

→ Binding data & method together

Name - Vishnu
Car - BMW

Nobody should be able to change the state of an object directly.

```
Class person {
private String name;
private car car;
```

```
Class price {
private int price;

}
```

```
Car car1 = new car();
Car.price = 55;
Car car2 = new car();
Car1.price = 105;
```

Nobody can change price of the Car

(1) No one should be allowed to change its state directly
↓
implement Access Modifier
↑ private

(2) Controlled manner you allow other objects to make changes.
↓ Setter & Getter

Encapsulation

(1) field private
↓ Controlled access
(2) getter & setter
Control who is getting & setting things

Inheritance
inherit properties
A C

(1) less
(2) Modifi

Relation

```
Class price {
private int price;

}
```

```
Car Car1 = new Car();
Car price = 55;

Car Car2 = new Car();
Car price = 100;
```

Nobody can change price of the Car

(1) No one should be allowed to change its state directly

↓

implement Access Modifier ↑ private

(2) Controlled manner you allow other objects to make changes.

↓ Setter & Getter

Encapsulation

(1) field private

↓ Controlled access

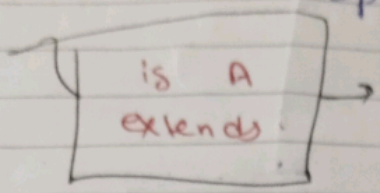
(2) getter & setter

Control who is getting & setting things

Inheritance → Real word

inherit property

A class/object can only inherit from Specific class only



to Java.

1. Less Code.
2. Modification is easy & fast.

RelationShip → possession

has A → Association

```
class Car {
    String name;
    String Color;
}
```

```
class person {
    String name;
    Car car;
}
```

Association

Association → (1) Composition →
(2) Aggregation →

```
class university {
    professor professor;
}
```

```
class professor {
}
```

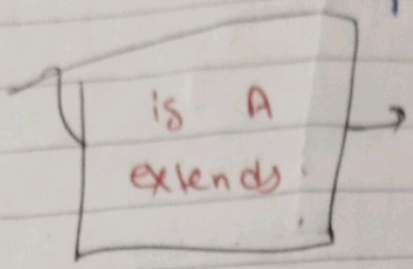
University has A Professor

does not depend on each other full time → Aggregation

Inheritance → Real world

inherit property

A class/object can only inherit from Specific class only



to avoid

① Less code.

② Modification is easy & fast.

Relationship → possession

has A → Association

class Car {
String name;
String Color;
}

class person {
String name;
Car car;
}

Association

Association → ① Composition →
② Aggregation →

class university {
professor professor;
}

class professor {
}

University has A professor

does not depend on each other full time → Aggregation

Composition:

class humanBody {

}

class heart {

}

Human Body has A heart → Yes

depend on each other → composition.

Relationship

- parent child — is A.
- possession — C_1 has A C_2
  - Association
    - Composition — object depends directly on another (heart / human body)
    - Aggregation — [illegible] [illegible] doesn't depend

Polymorphism

human behave differently under different situations

1. static

cla

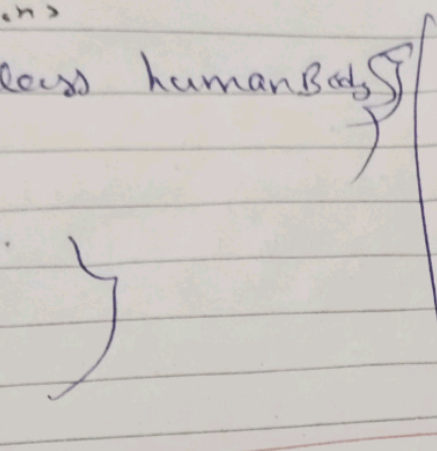
met

2.

Composition:

Class humanBody { }

Class heart { }



Human Body has A heart → Yes.

depend on each other → Composition.

Relationship

parent child

is A.

possession

C1 has A

Association

Composition

object depends directly on another

heart/human body

Aggregation

Person

doesn't depend

Polymorphism → Poly + Morphism.
Poly → many
Morphism → forms.

human Behave differently under different situations

Two types
1. Static / Compile time polymorphism
2. Dynamic / Run time polymorphism.

1. Static / Compile time polymorphism → method overload

```
class Calculator {
    public int add(int a, int b){
        return a+b;
    }
    public int add(int a, int b, int c){
        return a+b+c;
    }
}
```

Method overloading → More than 1 method with the same name.

2. Dynamic / Run time polymorphism
Method overriding → inheritance.

Polymorphism → poly + Morphism.

poly → many

Morphism → forms.

human Behave differently under different situations

Two types

1. Static / Compile time polymorphism
2. Dynamic / Run time polymorphism.

1. Static / Compile time polymorphism → method overload

```
class Calculator {
    public int add(int a, int b) {
        return a+b;
    }
    public int add(int a, int b, int c) {
        return a+b+c;
    }
}
```

Method Overloading → More than 1 method with the same name

2. Dynamic / Run time polymorphism

Method Overriding → inheritance.