26/2/24

Object Oriented Concepts

→ Object → Any real life entity which has its own
- Properties
- Behavior
- Identity.

OOP → Data Centric Approach {Data + Process}

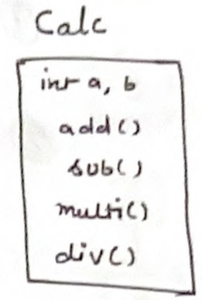
Procedural Programming → Process Centric Approach. {Process + Data}

Differences

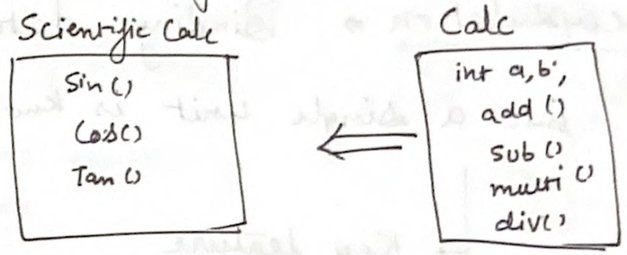
① Reduction of Complexity
→ OO is less Complex
- ① Bottom up approach
- ② ~~Rel~~ Real life entities. → uses things that are in use in our daily lives.

Bottom-up approach →

② Reusability → In object Oriented, all related things are bundled in a single unit which is better reusable.



③ Better Extendibility =>



④ Maintainability → Better ~~Repeable~~ ~~Reusir~~ Repairable

Objects has its own → Properties, Behavior, Identity

Class => Blueprint of sametype of objects.

① Abstraction ⇒ Concerning upto a level of complexity at a time.

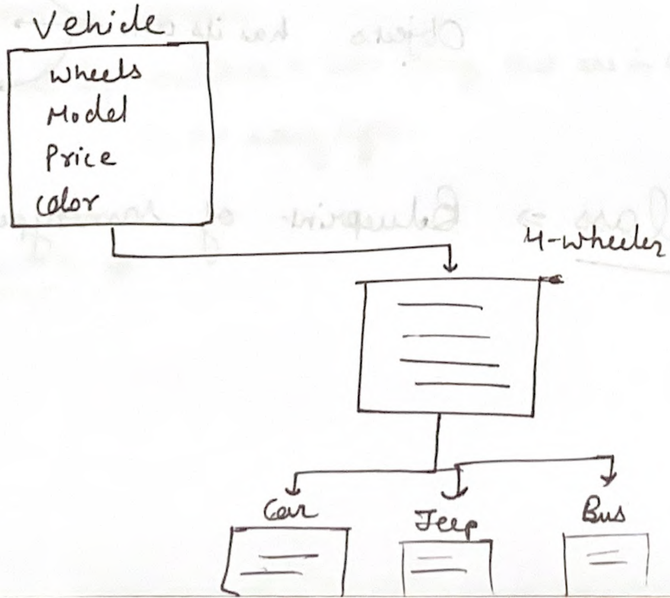
① Data Abstraction → Hiding complexity of Data

② Functional Abstraction ⇒ Complexity of function/operation.

② Encapsulation ⇒ Binding data & operations into a single unit is known as encapsulation.

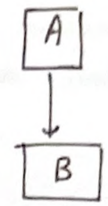
↳ Key feature

③ Inheritance → Inherit features from one class to other.

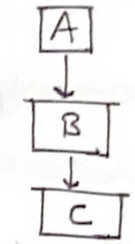


Types of Inheritance

① Single Inheritance ⇒

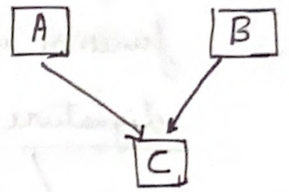


② Multi-Level Inheritance ⇒

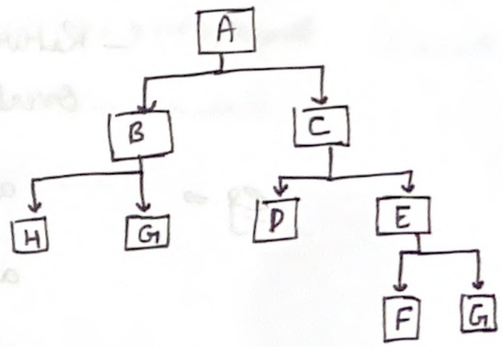


③ Multiple Inheritance ⇒

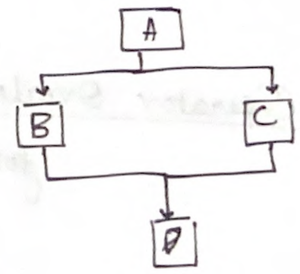
{No direct support in Java}



④ Hierarchical Inheritance →



⑤ Multi-path Inheritance ⇒



(4) Polymorphism ⇒ Poly + Morphism

{ Multiple ~~same~~ work with same name }

(a) Function Overloading

(b) Operator Overloading

(c) Function Overriding

(d) Polymorphic References.

function Overloading ⇒ writing more than one function with the same name but with different signature ⟶ Type of arguments.

↓ no. of arguments

- Return type has no roles in function Overloading.

Eg ⇒
```
add ( int a, int b)
add (int a, int b, int c)
add (float a, float b)
```

Operator Overloading ⇒ Re-defining operators for user defined datatypes.

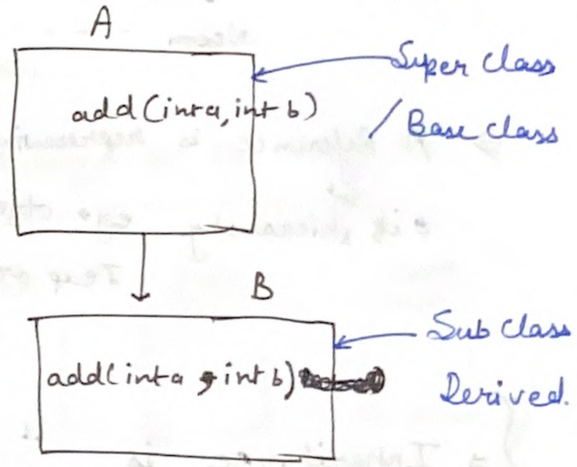
Eg ⇒ `int a, b;` `a+b`

{ Java doesn't support Operator Overloading }

`Date dl;` `dl+3;` `dl+-3;` ⟶ we have to redefine + operator with 1 date & 1 int

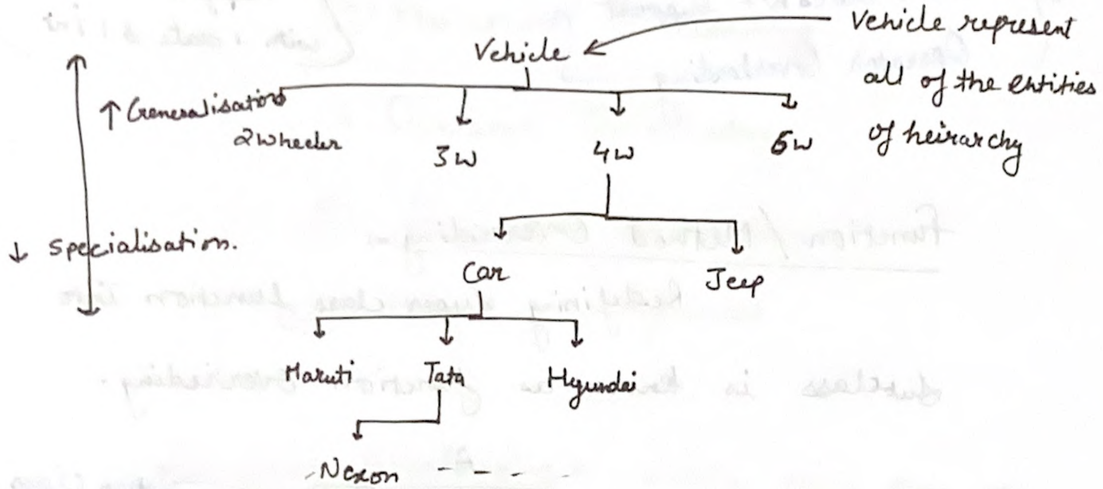
function / Method Overriding ⇒

Redefining super class function into subclass is known as function overriding.



01/03/2024

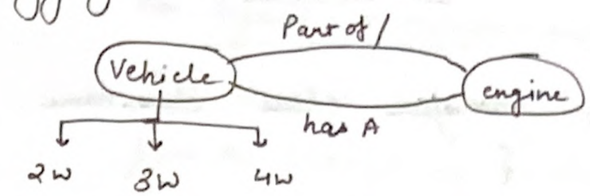
Polymorphic References ⇒



⇒ A Reference is representing its own Object and objects of all. its sub hierarchy. eg⇒ object of Vehicle can represent object of Jeep or Tata or 2Wheeler.

{ ⇒ Inheritance is "IS A" Relation eg⇒ (Car is a Vehicle, Maruti is a Vehicle / Maruti is a 4 wheeler). }

Aggregation / Composition

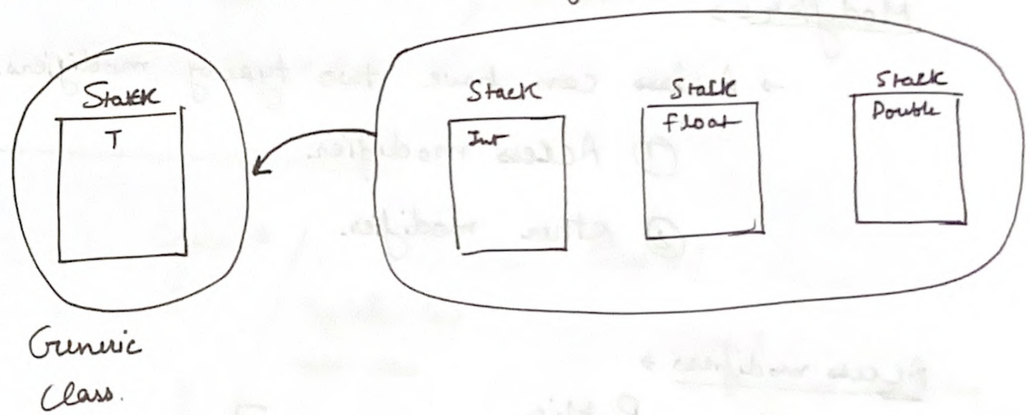


Inheritance is 'Is A' relation, Aggregation is 'Has A / Part of' Relation.

Generic Classes.

⇒ Datatype independent classes.

⇒ We would need to mention datatype at the time of use.



Class

Syntax →

```
modifiers class class_name
{
    modifiers datatype var_name;
    modifier datatype var_name;
        :
    modifier return_type method_name (Arguments) { }
    modifier return_type method2 (arguments) { }
        :
```

Modifiers ⇒

→ A class can have two type of modifiers.

1. Access modifier.
2. Other modifier.

Access modifiers →

- Public
- default / no modifiers

} In class

{An upper Level class cannot be private / Protected.}

```
class A
{
}
```

← This class has no access modifiers which is default / no modifier and has package level access or this class can be accessed from the package only in which it is declared.

eg →

```
{ Package P1;
  class A { } }
```

← class A can only be accessed inside Package P1.

```
Public class A {

}
```

← This class can be accessed from anywhere.

```
Private class A { }
    OR
Protected class A { }
```

} Error.

```
Class A
{

}
```

/ Top level class
upper class

```
Class A {
    Class B
    {
    }
}
```

/ Not upper class

Class Members

- Access Modifiers
- Other

→ **Access modifiers** → Class member can have four types of modifiers.

- → Private access
- → Protected
- → Default / No modifiers
- → Public

① **Private** → These members are accessible only within the class.

② **No modifier / default** → accessible from the same class in the same package

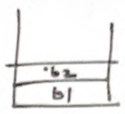
③ **Protected** → Accessible from same class, same package and subclass of any other package.

④ **Public** → Accessible from anywhere.

```
class Box
{
    double width;
    double height;
    double depth;
}
```

```
class BoxDemo {
    psvm(String[] args){
        Box b1;
        Box b2;
        ----------
        b1 = new Box();
        b2 = new Box();
```

In memory b1 & b2 name has memory allocated

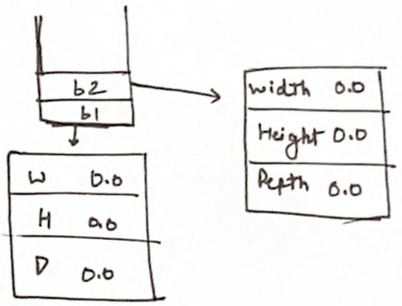


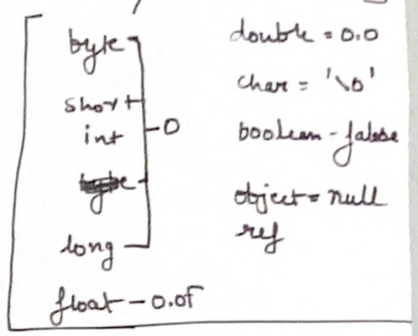
new Box() will do three thing

① Memory allocation

② Default value initialisation of default values for object members / Instance members

③ Execution of constructor

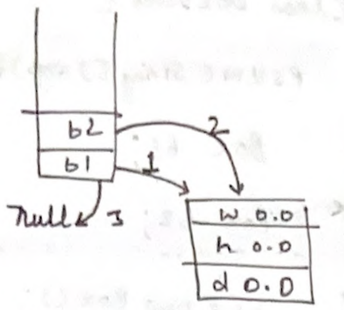




Instance variable / Object variable =>

variables that gets created everytime a object is created. eg → width / height / depth.

```
b1 = new Box();
b1.w = 10; b1.h = 2.5; b1.d = 12.1;
```




```
Class BoxDemo{

PSVM (){

  Box b1 = new Box();

  Box b2 = b1;

  b1 = null ;

  double vol;

  Vol = b2.w * b2.h * b2.d;

  SOPC Vol);

  Vol = b1.w * b1.h * b1.d;

  SOP(vol);

}

}
```

Null pointer exception. ←//