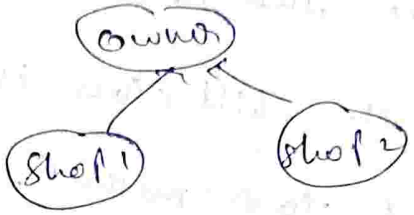
Types:

-> single inheritance

↳ one parent & one class (child)

-> Multilevel inheritance

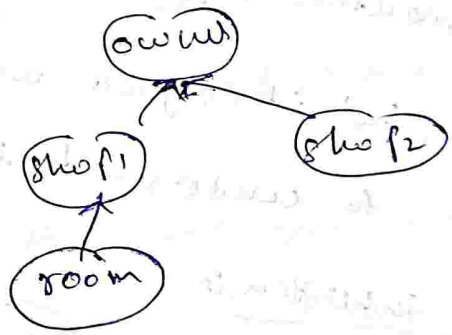
-> Hierarchical Inheritance



-> Hybrid inheritance:

combination of 2 or more inheritance types

Eg:-

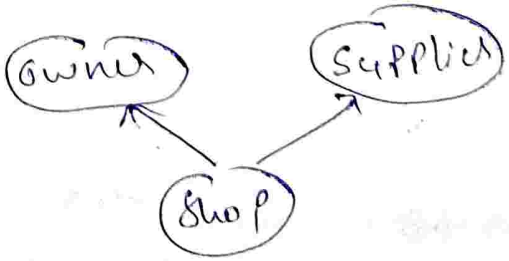


-> Multiple inheritance:

using interfaces

-> 2 parent & 1 child class

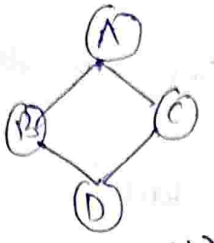
Eg:-



Rule:

In Java only one class can be extend. Multiple classes for single class cant be extended.

→ Dimaond Problem



Is Here one class can't be extends 2 classes.

* what is solution to inherit 2 parent class for one child class ?

Ans: we use concept called "Interface"

→ In "interface" it is possible to give 2 parent class to one child class

Interface:-

we use "implements" to get class properties from parent.

Eg:- class shop implements owens, interface supplier2 {
}

Rules:-

1) in interface we should intlized all variables

int num = 10 ✓

int num; ✗

2) void methods should not be declared. we could write body for method.

→ It is ovdided in child class.

Inner class:-

→ It means a class in ~~same~~ other class.

Eg: class chaishop {

class Thikka {

String evariki = "me";

}

}

How to access & initialize:-

Chaishop branch1 = new chaishop();

~~chaishop. Thikka th = new branch1. Thikka~~

chaishop. Thikka th = branch1. new Thikka();

Java Packages:-

→ generally in Development we can't write single code in same file.

→ so, the packages are used to get the code from other file.

Eg:-

That exported code is accessed here. import folder1.owner;

class shop extends owner {

String branchName = "shop1";

}

main {

Chaishop branch1 = new chaishop();

```
        Sys.out.println( branch1.ownerName );
    }
}
```

folderA -> Owner.java

```
package folderA;
public class owner {
    public String ownerName = "Vignesh";
}
```

This indicates that this code is exported

Modifiers

-> Access Modifiers

-> Non-Access Modifiers.

1) Access Modifiers :

- public -> It can be accessed Any where
- protected ->
- default
- private -> only access in that class only.

public -> subclass, same class, same packages, diff packages & ~~not~~ subclass, other class (diff package)

protected -> same class, same package, diff pack (sub class)

default -> same class, same package.

private -> only same class.

Non-Access Modifiers:

- -> static
- -> final
- -> abstract
- -> synchronized
- -> volatile
- -> transient
- -> native

Static

-> static member belongs to class not obj

-> can be directly accessed no need of object creation

-> will be initialized only once

Eg:-

```
class shop {
    static int cnt = 0;
}
main() {
    shop a = new shop();
    shop b = new shop();
    a.cnt++;
    b.cnt++;
    print(a.cnt, b.cnt)
```

=> o/p => 2 2

because,

if we keep static then that variable is stored in class not in object.

Shop = { cnt = 0 }

a = { }

b = { }

-> when a.cnt++;

-> Shop = { cnt = 1 } -> count is stored in class not in obj

a = { }

b = { }

Parent pc = new child();

-> It says child variable are private of parent & methods are private from child.

Left -> variable, static methods

Right -> method (In case Both have same methods)

---

* static functions cant be overrided

Final:

↳ final makes variable constant

-> It should be initialized directly or using constructor.

abstract:

↳ if abstract is placed before any class we can't create objects.

-> It should be extended by other classes.

-> It can contain both abstract methods or normal methods. -> with body
no body

-> The abstract methods should be override in child class.

Synchronized:

It ensures one thread access block of code or method at a time to

prevents race condition.

voilate:

-> It indicates that the threds should get the value from the memory. not from cache.

transient: It prevents variable from serializing.

-> It will not store the value in the file while storing an object into file.

-> used in sensitive data.

Native: - suppose we write a method in other lang c/c++ then we use that method in class then we will keep native key word.

-> It indicats it is other langs method.

Features of OOP:

1) encapsulation:

-> Here we will define a class with variable & Methods & keep restrict of some variable & Methods. (Private) it is called encapsulation.

-> ~~use~~ We use getters & setters to achive it.

2) Abstraction:

Hide implemetion details & exposes only essential features of Object.

-> This can be achive using abstract classes & interfaces.

3) Polymorphism:

2 types ->

i) compile time polymorphism
   L> Method overloading.

2) Runtime Polymorphism
   L> method over ridding.