OOPS:- 16/6/25:-

class => It is used to create your own Data type.

to create class the first character should be capital.

Eg:-
```
class Shop {
    string branchName;
    int carsSold;
}
```

How to access in main() :-
or create

```
Shop branch1 = new shop();
```

-> First DataType, next variable, new keyWord to Create object in Java, next class name of same DT.

How to access :-

```
Sys.out.println(branch1.branchName);
```

↓ this will give stored value in class object.

so finally,

Shop => class

branch1 => object, we can create multiple objects by using single class.

Methods in class / Functions:

```
class chaishop {
    string name;
    int cups;
    Public void classFun ( ) {
        S.y.out.Print( " class fun " ),;
    }
}
```

Constructor:-

↳ In class we should create function same as classname.

↳ If has no Return type.

```
Class chaishop {
    int cups;
    string name;
    Public chaiShop ( string bn, int cs )
    {
        name = bn;
        cups = cs;
    }
```

this :

it is used to access the variables in class

Inheritance :

17/6/25

↳ It is used for Reusability of code.

Def - It allows a child class to inherit properties & behaviours from parent class or other class

Eg:-

```
class owner {
    String ownerName = "vignesh";
    int age = 21;
    int num = 123;
}

class shop extends owner {
    :
}
```

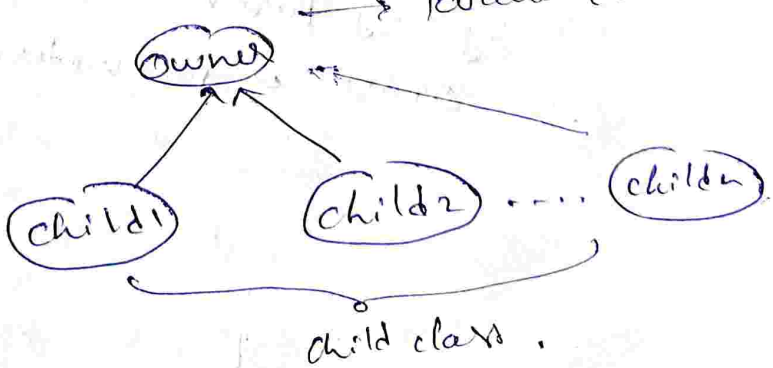
parent class (or) Base class

child class

extends → it is the key word to use inheritance concept.

Hierarchical Inheritance :

↳ More than one subclass inherits the parent class it is called " ".



Method overloading :

→ In a class the methods have same name but the method parameters are different

Eg :

```
class shof {
    Void fun() {
        s.yso.p("method1");
    }
    Void fun(int a) {
        sys.out.println("method2");
    }
    Void fun(String str)
    { print("method3");
    }
}
```

→ Here with same method but different DT will consider as different fun.

```
void fun(int a, String b)
void fun(String a, int b)
```
} different methods.

Imp :

→ No. of parameters
→ Type of Params
→ order of Parameters
} → all these are consider a different methods

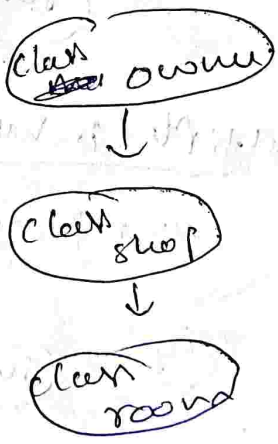
Method overriding:

-> Suppose in parent class & child class have same methods with same parameters then method in child class will get called when we use that method.

-> Here the child class is overridding the parent class method.

-> If we use overriding in child the we should mention "@.override". It is not mandatory but it is good way of writting & easy to understand to others.

Multilevel inheritence:



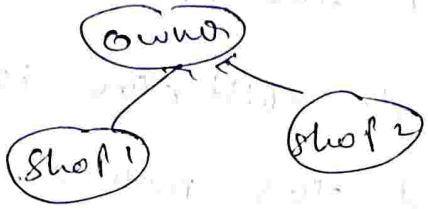
-> Here room class extends shop class & shop class extends owner class, so the owner methods can be accessed by room class this is called Multilevel inheritence.

Types:

-> single inheritance
   -> one parent & one class (child)

-> Multilevel inheritance

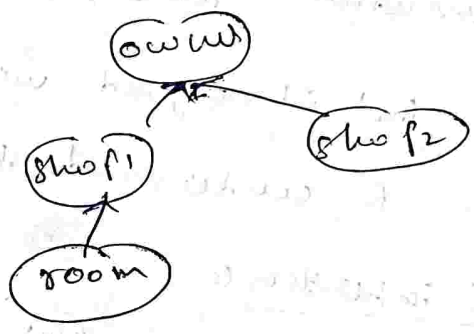
-> Hierarchical Inheritance



-> Hybrid inheritance:

combination of 2 or more inheritance types

Eg:

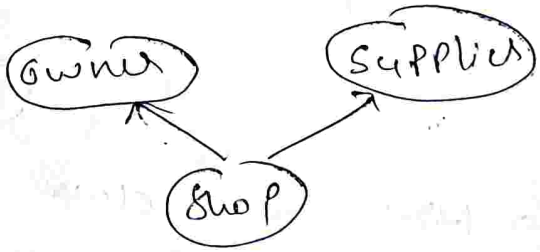


-> Multiple inheritance:

using interfaces

-> 2 parent & 1 child class

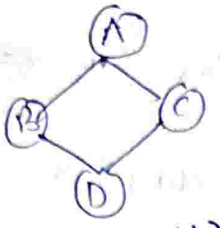
Eg:-



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 owner, supplier {
}

Rules:-

1) in interface we should intilized all variables

int num = 10 ✓

int num; ✗

2) Void methods should not be declared.

we cannot write body for method.

→ It is overided in child class.

Inner class:-

→ It means a class in other class.

Eg:
```
class chaishop{
    class Thikka{
        String evaliki = "ne";
    }
}
```

How to access & initialize:

```
Chaishop branch1 = new chaishop();
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:
```
import folder1.owner;
class shop extends owner{
    String branchname = "shop1";
}
main(){
    Chaishop branch1 = new chaishop();
```

That exported code is accessed here.

```
        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 make static then that variable is stored in class not in object.

shop = {cnt = 0}

-> when a.cnt++;

a = {}

b = {}

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

a = {}

b = {}

Parent pc = new Child();

-> It says that variable are printed of parent & methods are printed 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

-> The abstract methods should be override in child class. (no body)

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 serialization

-> 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 indicates 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.

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

2) Abstraction :

Hide implemention details & exposes only essential features of Object.

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

3) Polymorphism :

2 types ->

i) compile time polymorphism
↳ method overloading.

2) Runtime polymorphism
↳ method over ridding.