# OOP :- ( object Oriented programming )

**class :-** A class is a template for an object & an object is an Instance of a class

It is new data type used to create objects

Objects → state, identity
(3) properties behaviour.

```
Class student {

      int rno;
      String name;
      Float marks;

}
```

```
Student s = new Student( );
```

[dynamically allocated memory & return the reference.]

Student s → done at compile time (declaring reference to object).

new Student( ); → stored in Heap
* done at Runtime
* allocate a class of object.

s. rno → used to use the attributes of Certain object.

Constructor is Special function that runs when you create an object & it allocates Some variables
Constructor has Same names of class

**this keyword :-**

It is used to reference the attributes of objects to variable.

```
student (int vinay) {
      this. vinay = vinay
}
```

**Final keyword :-** final is used to declare a final value to which the final the variable. ( It is only valid for primitives)

```
final int vinay = 2
```

**Finalize( ) method:-** when the object is delete or destoryed, It need to perform Some action which it is can be done by using Finalize( ) method. Java Calls this method when the object is going to delete.

```
protected void Finalize( ){ → finalization code.
}
```

## Overloading methods :-

It is possible to create two (or) more methods of Same name as long as their parameters are differents.

## Overriding :-

the method which have Same na
Overriding is done when the method which have Same na Overloading methods.
& type. , If not then it is

## static :-

Static is used for method (or) attribute which is independent of objects. which are invoked when class is created

main ( ) → (Static method) which is independ of Objects

Static method can only access the static data only.

static method are loaded onces, when the class is loaded

The nested classes Can be static.

Static inner classes can have static variables

## Singleton class:-

Singleton class is which is used to only for One objects. Even though the multiple objects refers to Same objects

## packages:-

packages are containers for classes

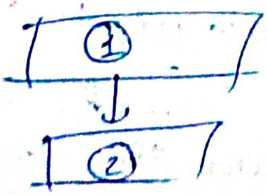
## Inheritance :-

The child class inherits the attributes of parent class. Extends keyword is used to inherits the values (or) properties

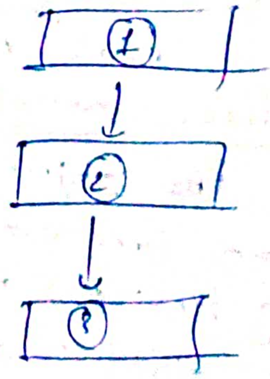
## Types of Inheritances:-

1. Single Inheritance:-

One class extends another class

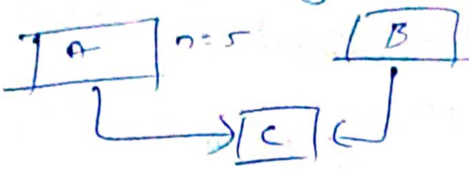


2 multilevel Inheritance:-



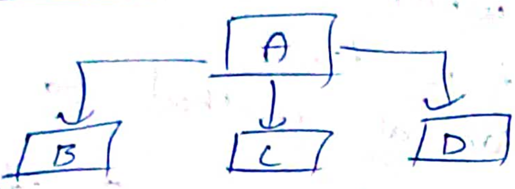
③ multiple Inheritance:-

One Class Exibreily more than 1 Class.

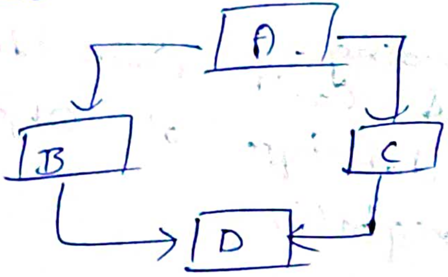


It is not possible in Java.
i don't know whether
n=5 (or) n=10.
It is overcomed by Interfaces.

④ Hierarchical Inheritance:- One class is inherited to many classes



⑤ Hybrid Inheritance:- combination of single & multiple Inheritance
cannot in Java normally
Overcomed by Interfaces.



Super method is used to take attributes (or) properties of parent class.

Ex:- Super(attributes)
for method super.method → method name.

polymorphism:-

poly morphism
many ways to Represent.

Types of polymorphism:-

① Compile Time/ Static polymorphism

Achieved by method Overloading
& Operator Overloading (not possible in Java)

determine which method to use at Compile Time.
while Overloading type casting will be done automatically.

② Run Time / Dynamic polymorphism:-

By Overriding method
@Override is notation used to check whether it is Overriding.

Shape v = new Circle();

then v Contains the properties of shapes not Circle But the methods which Contained in shapes will be Overriding by Circle Class

( Same named method are Overriding)

Dynamic method dispatch:-

It is the mechanism by which a call to an Overridden method is resolved at run time.

Java determines which version of method to use at using runtime.

By final keyword the method can't Overidden. & Can't Inheritened also

Static methods Can't Overriding..

Static methods Can inherit & Can't Overridden

it depend type Class (on) data type of Object.

$\boxed{\text{Box}}$ b = new Boxweight();

Encapsulation:-

wrapping up the implemenation of the data members & methods in a class.

Abstraction:-

Hiding unnessary details & Showing valuble information.

Access Control:-

private:- The data in private Access modifier Cannot use & change.

The private data can modifier & use by using the Getter & Setter method;

Ex:-

```
private int v;
public vinay() {
    return v;
}
```

Here data v is private but the method is public so we can access outside. It is Getter method

```
private int k;
    public V( int m) {
        this.k = m;
    }
```

} k is private eventhough the value can be modified using this method // Setter method

The default access modifier is private & which we can in same package.

|  | class | package | (inherit) sub class in diff package. | diff package & not subclass |
|---|---|---|---|---|
| public | ✓ | ✓ | ✓ | ✓ |
| protected | ✓ | ✓ | ✓ |  |
| default modifier | ✓ | ✓ |  |  |
| private. | ✓ |  |  |  |

io package :- used for files. Etc

util package :- for framework Eg:- Array list vector Etc

lang :- :- for basic operations

applet :- for Servlet Etc.

awt :- for graphical Interference.

net:- for networks

} Built in packages

Some built-in method in lang:-

hashcode ();- Gives the hashcode of object. (hash value)

equals ();- checks the contain of object are Equal(or) not

Abstraction :- Abstraction method we need to use Abstract for the class. Eg:-

```
public abstract class People {
        abstract void vinay();
    }
```

The function should be Overridden in child class. of main class.

we can't create objects of abstract class.

we can't create abstract Constructors.

## Interfaces:-

Multiple Inheritance is not available in Java but we Can do it by the Interfaces. Interfaces is like abstract classes.

Default the Variables in Interface are static.

The Interfaces which has no member is known as a marker (or) tagged Interfaces.

Interfaces Can be nested.

## Exception handling:-

Exception is on Event that disrupts the normal flow of the program. It is on object which is thrown at runtime. The java.lang.Throwable class is the root class of Java Exception.

They are 3 types of Exceptions: 1. Checked Exception 2. Unchecked Exception 3. Errors

### 1. Checked Exception:

The classes that directly inherit the throwable class Except RunTime Exception & Error are known as checked Exception. Checked Exception are done at Compile-time.

### 2. Unchecked Exception:-

The classes that directly Inherit the Runtime Exceptions are known as Unchecked Exception.

The Exceptions are handled by:-

① Try:- The try keyword is used to Specify a block where we should place an Exception Code.

② Catch:- The Catch block is used to handle the Exception. The Exception occured in try block Can be handled Using catch block.

③ Finally:- The Code in finally block is Executed whether Exception is handled (or) not

① throw: Throw keyword is used to throw the Exception.

② throws :- The throws keyword used to declare Exceptions.
It is always used with method Signature.

For multiple Exceptions the catch block (multiple catch blocks) has to be in order. default Exception should mention (or) used at the End. of code.

Nested try block can be used.

we can Create Our Own Exceptions it is known as user defined Exception by Extending Exception class.

## enum:

The Enumeration is created by enum keyword.
It can declare outside (or) inside of class, not inside of method.

## Linked List in Java :-

Linked List is part of Collection framework present in Java.util.package. It is linear data structure where the Elements are not stored in Contiguous locations & Every object is Separate Object with the data part & address part.
The Element is known as Node.

1. LinkedList ll = new LinkedList( );
2. LinkedList ll = new LinkedList(C);

Here c is the collection of Elements. (List of Elements)
(E → Generic type)

methods of LinkedLists :-

add(int index, E element) : To add Element to linkedList

add(E Element) : To append the Element.

addAll (collection <E> c) :- To append collection of Elements

addAll (int Index, collection<e>) :- To add Collection of Element at certain index.

addFirst(E e) :- Add Elements at Start index

addLast (E e) :- Append the Element at last.

clone( ) :- returns the clone linked List

Ex:-

```
LinkedList ll = new LinkedList( );
LinkedList s1 = new LinkedList( );
s1 = (LinkedList) list.clone( );
```