OOPS :

object → Real world Entity such as Pen, chair, Table, computer, Watch.

OOPS → methodology / paradigm to design prog < class / objects

concepts:

object, class, Inheritance, Polymorphism, Abstraction, encapsulation.

Other concepts:

coupling, cohesion, Association, Aggregation, Composition

object:

* It has state & Behaviours.
* Defined as instance of class.
* can communic w/o knowing detail of data/code.
* Type of message accepted & type of response return by obj is important.

class:

* collection of objects
* Logical entity.
* ↳ Blue print from which you can create an individual object.
* class doesn't consume any space.

Inheritance:

* When an object acquires properties & behaviours of a parent object, it is known Inheritance.
* Provides code Reusability.
* Helps to achieve runtime polymorphism

Eg: Project for CGPA calculation for some 120 students, (need not calculate CGPA for all)

Parent class is inherited to children.

Subclass: class that inherits superclass with its own fields & Methods.

POLYMORPHISM:

If, Task is performed in diff ways.

Eg: calculate Area of Rectangle, Area Square,

overload:

same class, same name, different parameter

↓

Static Polymorphism

```
class Area {
    static int c(side)
    { return side * side; //SQUARE
    }
    static int (int l, int b)
    { return l * b; //RECTANGLE
    }
}
```

Dynamic Polymorphism.

↑

overriding:

same Meth name, same para, diff class.

Eg: For salary calculation of employees

(No of days * salary per day)

In the Runtime, while entering → o/p changes.

3rd largest global Electronics Money serv

Corp → Singa

Admin → Cafe

Abstraction

Hiding internal details & showing functionality

Eg: Phone call, we don't know the internal processing

redBus, google Flights → info abt airlines can be obtained, if they are Hacked

Encapsulation:

code + data → Binded Together in a single unit

Eg: Java class, Java Bean

coupling:

Knowledge of dependency / info from other class

STRONG: If class has details / info of other class → strong coupling

In java, we use private, protected, public modifiers to display visibility ↓ level of class, method field

WEAK: Use interfaces for weaker coupling.

cohesion: It refers to level of component which performs a single well-defined task.

↓

Highly cohesive Method.

java.io package → Highly cohesive

java.util package → weakly cohesive (unrelated classes & interfaces)

↓

splits Task into various parts

Association:

Relationship b/w objects.

4 Types: One to one

One to Many

Many to One

Many to Many

Aggregation:

Way to achieve Association
Represents the Relationship where 1 obj contains other obj as part of its state.

Agg : has-a relationship
Inherit : is-a relationship
Another way to reuse objects

Composition:

↳ If you delete parent obj, child obj will be delet automa
↳ Represents rel where 1 obj cont other
↳ as part of its state.
called a way to achieve association)

Advantages of OOPS over procedure oriented programming

① OOPS → development & Maintenance easier.
Proc → Not easy to Manage if code grows as proj size increases.

② OOPS → provides Data Hiding.
Proc → global data can be acc from any where

③ OOPS → provides solution to Real world problems more effectively.

ANONYMOUS INNER CLASS:

↳ class that has no name.
↳ override Method of class /Interface.

Naming conventions:

class Nouns such as Button, Thread, system

Interface: Adjectives such as Runnable, Remote, ActionListener

Methods: Verb such as Main(), print()

Variable: Not start with $, _, -

Package: lowcase such as java.lang.

Multiple words sep by .

Constant: MAX_PRIORITY.

camelcase: start with uppercase. → actionPerformed

OBJECT:

State : Represents data of an object.

Behaviour : represents behaviour of obj.

Identity: used internally by JVM to identify each object uniquely.

CLASS:

class contains field, Method, constructors, Blocks, Nested class & Interface

INSTANCE VARIABLE:

Variable → created inside class inst var → created outside Method.

∴ { Doesn't get Memory at compileTime.
gets Memory at RunTime.

METHOD:

Behaviours of obj.

Adv: code Reusability
code optimization.

New Keyword:

Allocate Memory at Runtime

All obj get memory in heap Memory Area

ways to initialize object:

i) By reference variable
ii) By Method
iii) By constructer

Ref variable:

```
Stud s1=new Stud();
s1.id =101;
s1.name = "Anu";
```

Method:

```
void insertRec(int r, String n)
{
    rollno=r;
    name =n;
}
void display()
{ Stud s1= new Stud();
  Stud s2=new Stud();
  s1.insert(111, "karan");
  s2.insertRec(222, "Aryan");
  s1.display();
  s2.display();
}
```

Ways to create object:

* By new keyword
* By. newInstance()
* By clone()
* By deserialization
* By factory().

Constructors:

→ It is called when an instance of class is created.

→ At the time of calling constructor, memory for the obj is allocated in memory.

→ spl type of method to initialize object.

Types: * Parameterized constructor

* no-arg constructor. (o null) → Default const

→ Constructr value at time of obj creation

Difference b/w constructor & Method:

Constructor	Method
* Invoked implicitly	* Invoked explicitly.
* Initialize state of obj.	* Expose behaviour of obj.
* Doesnot have return type	* Must have return Type.
* Constname → same as classname	* Methname may be same as class name.
* Javacomp provides default const.	* It doesnot provide any compiler.

Static keyword:

→ Used for Memory Management (saves Memory)

→ static can be obtained with variables, Methods, Blocks & Nested classes.

→ Belongs to class than an instance of class.

```
class student {
    int rollno;
    String name;
    String college = 'ITS';
}
```

→ suppose there are 500 stud in college, now instance data memb will get memory each time when obj is created.

↓ common for all students → If we declare static, field gets memory once.

Jam of counter

1) Without static variable:

```
class counter {
    int count=0;  // will get memory each time
    counter()     when instance is created.
    {
      count++;
      S.O.p(count);
    }
psvm(String args[]){

counter c1 = new counter();
Counter c2 =new counter();
counter c3 =new counter();

}
}
```

op: 1
1
1

```
static int count=0;
//
// will get memory one
& will retain its
value.
```

op:
1
2
3.

JAVA STATIC METHOD:

Whenever method is declared static -> static method.

static method:

* Belongs to class rather than object
* can be invoked w/o creating instance of class
* Access static data Member & change value

```
class student {
    int rollno;
    string name;
    static string college = "ITS"; // static method to
    static void change()           change value of static
      {  college = "RMK";          variable
      }
```

Restrictions of static Method

* Cannot use non-static method data member / non-static method directly
* super & this → directly cannot be used

Why is Java main() static:

* static methods do not req obj to call it.
* Non-stat method, JVM creates obj first & calls main() leads to extra Memory Allocation

This KW in Java:

this → reference variable that ref to current object

* current class instance variable
* current class Method
* current class constructor
* as an arg in method call
* as an arg in const call
* return current class instance from method

curr class
/ | \
meth const inst var

arg
/ \
meth call const call

Interface: (implements)

Inheritance: (extends KW) class subclass extends superclass { ... }

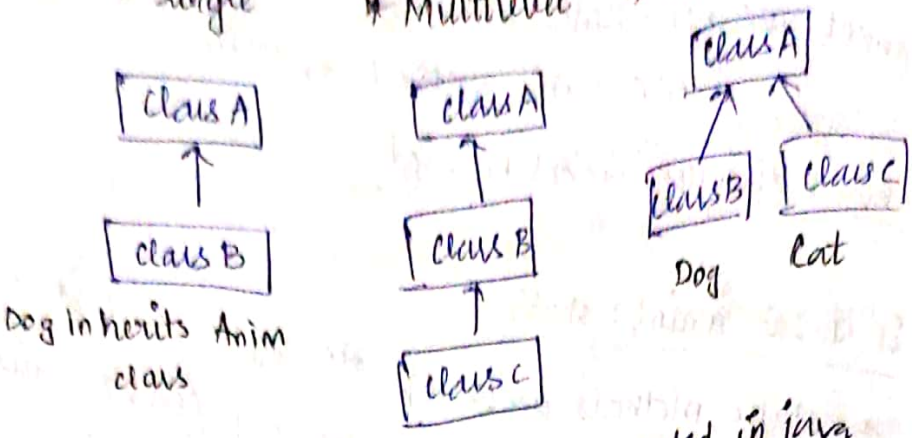
→ Acq behaviours, prop of parent class.
→ Reuse methods, fields in parent class.

* class: group of obj same prop.
* Subclass: Derived class / extended class / child class.
* Parentclass: Class from where subclass inherits feat. Also Base class
* Reusability: reuse fields & methods of existing class

Types:

Baby dog inherits dog class → Anim class

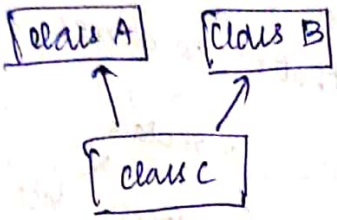
* Single
* Multilevel
* Hierarchical



Dog inherits Anim class

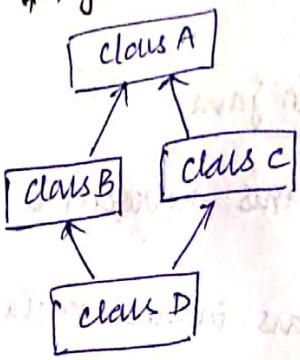
Multiple Inheritance not supported in java

* Multiple



Compile Time error

* Hybrid



Aggregation:

→ Employee contains many info such as id, name, email id. One more obj named address ↓ info such as city, state, country, zipcode

→ code Reusability.

→ When no is-a relationship is Maintained through lifetime of obj involved.

Method overloading:

Multiple methods having same name, diff para

Eg: Add method 2 para (int, int) / Add with 3 para (int, int, int)

↓

Readability ↑ as name differs

Ways to overload:

By changing no of arguments

By changing data types.

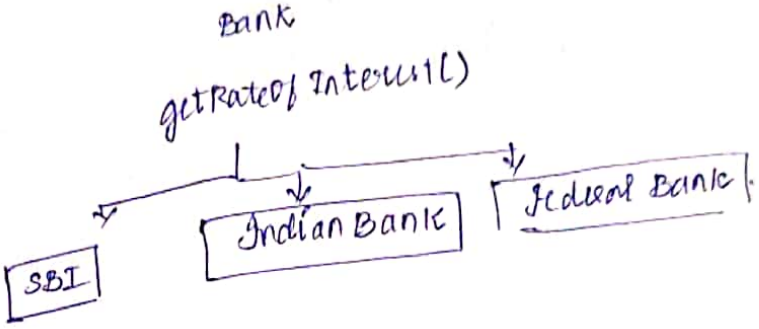
overloading doesnot occur by changing return type of method

main() method also can be overloaded by Method overloading.

Method overriding:

same method name, same parameter, is-a relationship.

Eg:



super keyword:

→ Reference variable refers to immediate parent obj.

→ invoke immediate parent class

↙ ↓ →

instance variable method constructor

Final Keyword:

→ Restrict the ~~keyword~~ user.

→ Final used with Variable, Method, class.

Final variable:

→ It can If declared static, final, you cannot change the value of it. (Remains constant)

Final Method:

↳ cannot override it.

↳ It is inherited.

Final class:

↳ cannot be extended

Abstract class

Interface.

Interface in Java:

It is a Blue print of class.

It has static constants & abstract methods

Mechanism to achieve abstraction.

Java 8 → default, static meth in Interface

Java 9 → private methods in Interface

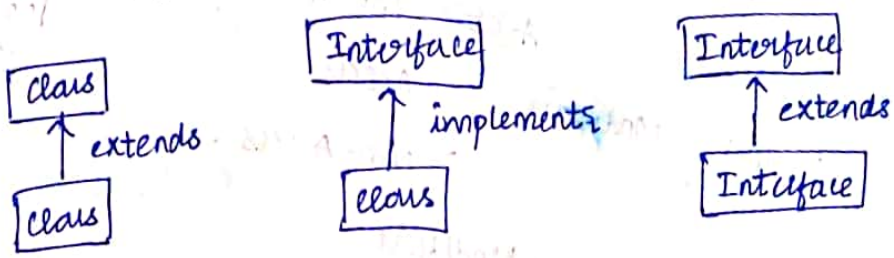
Why use Interface:

To achieve abstraction

support functionality of Multiple Inheritance

Achieve loose coupling.

Declared using interface keyword



Multiple Inheritance:

↳ Not supported in java, due to Ambiguity

↳ But possible through interface

Ambiguities → The rules not defined clearly in java (Inexactness)

Marked / Tagged interface:

Interface which has no members.

↳ Eg: Serializable, cloneable, Remote.

ABSTRACT CLASS	INTERFACE
→ Abstract + Non-abstract methods.	→ abstract methods only Java 8 → default, static
→ x supp multiple inherit	→ supp multiple inherit
→ final, non-final, static, non-static	→ only static, final variables
→ prov interf imp	→ x prov imp of abs class
→ 'abstract' KW	→ 'interface' KW
→ 'extends' KW to extend	→ 'implements' KW to implement
→ class members: private protected	→ public by default

ACCESS MODIFIERS:

→ Types of Modifiers: — Access
— Non-Access.

4 Types of Java Access Modifiers:

Private, Default, protected, public.

Access Modifier	within class	within package	outside package subclass	Outside package
Private	✓	×	×	×
Default	✓	✓	×	×
Protected	✓	✓	✓	×
Public	✓	✓	✓	✓

NOTE:
A class cannot be private / protected except nested class

ENCAPSULATION:

class with Methods of getters & setters
separate class
- Read-only
- Write-only

→ can't change value
only read ↓
Drawbacks avoided

ARRAYS:

collection of similar types of element with contiguous memory location
elements of similar data Type.

Index based.

Array in java → obj of dynamically generated class.

Java array:
inherits obj class
implements Serializable, Cloneable interfaces.

1d, 2d arrays can be used

Advantages:
→ code optimization
→ Random Access.

Disadvantages:
→ Size limit:
↓
→ doesn't grow in size at (runtime).
collection framework is used in Java

Jagged Array:

Odd no of col in 2D array.
Array with diff no of columns.

Array copy:

(obj src, int srcpos, int destpos, int length)

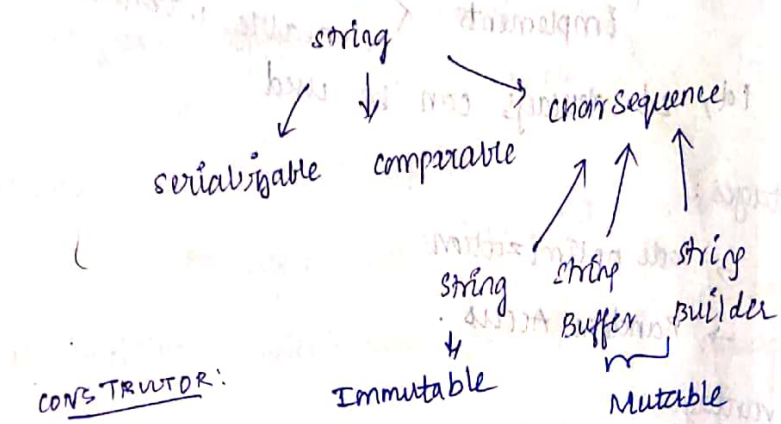
Array clone:

Implements cloneable interface.
Create clone of Java Array.

```
int arr[]= {1,3,4,5};
int carr[] = arr.clone();
for(int i: carr)
S.O.P(i)  //  1
              3
              4
              5
```

STRING:

Java.lang.string implements



CONSTRUCTOR:

this() → another const in same class with diff para list.

super() → invoke super class constructor.

1) Types of class: (a)
Abstract Final Mutable Wrapper Anonymous
Input-Output String System Network

2) class passed by ref, structured passed by copy

3) Template class (or) generic class

4) Friend class: It can access all private memb of another class
spl class prov to use, when u need to recm data of
class but don't want that class to have spl func

5) Has both memb func, Memb variables.
class | struc.
x memb func

6) Instance of which class x be created: Abstract class
Inst of Abst class x be mat ? x cont of own
∴ while creating instance of class, it can't initialize obj mems

7) If data memb are private → how to access them from
class object → using "public member functions";

8) Memfunc → change datatype of memb func

9) Immutable class → caching purpose

↓ Encapsulation.
↳ achieved using Access Specifiers.

10) Object → Abstraction & Data+code.

11) Abstraction
3 levels
Logical Physical level View

7338799460

12) Markable Interface:
No fields, Methods
Empty Interface
Eg: Cloneable, Serializable,
Thread safe, Remote

1) API documentation → javadoc tool.

2) getName() → returns class obj as string
getClass() → ret runtime class.
intern() & toString() → returns String class

3). Loc var has same name as instance variable.
↳ variable shadowing.

4) Anonymous inner class → no class name

5). java.util → package 'RandomClass'.
Nameless obj → Anonym obj.
Do not have reference.

6) Intf x meth / x fields → Marker Interface

7) subclass of Panel class → Applet class.

Final:

* Variable constant (Immutable
* Local var x be used in other class.
* Can't override subclasses.
* Final class can't be extended.
But final class can't extend other classes.

8). subclass of Throwable { Error / exception class

9) chained exception → exception caused by other exception.

10) intern() → ret existing string from Memory

11) Initial del qty of ArrayList → 10.

12). Mutable class in Java → StringBuilder

Methods of object class:

clone
equals
finalize()
getClass()
hashCode() → obj's memory addrs in Hexa
toString()
notify
notifyAll()
wait()

Threads:

* Lightweight process.
* Builtin we support → Multithread Programming.
* New → create instance of Thread class.
* Run → running state
* Suspended → temp susp activity.
* Blocked → waiting for a resource
* Terminated → Halt execution.

Method:

getName -
getPriority -
isAlive -
join → wait for thread to terminate
run → entry point of thread.
sleep → susp thread
start

Main thread:

↓
Affects child threads.
Create auto.
Req for shutdown

How to create:

i) Implement Runnable Interface
ii) extend Thread.

```
Thread t1 = new Thread(myClass());
t1.start()
```

(ii)

```
Myclass t1 = new Myclass();
t1.start();
```

~~Thread t1 =~~

volatile. → var used to access any part of thread

1) Byte stream → 8 bits (1 Byte)
Character stream → 16 bits (2 Bytes) Read/write

Python:

1) Memory Mngmt → Private Heap
Automatic Memory Mngmt
garbage collector.

2) self: instance or obj of class
explicitly included as 1st parameter
not the case in java

3) obj → called by value in pyt

4) init → method / constructor in python.
↳ automatically allocate memory when new obj / instance of class created.

5) range → Python list obj
xrange → xrange obj → generate static list
not

6) Pychecker → bug Tool (style & complexity
Bugs in Python src
Pylint → Verifies coding standard
debug

7) Pickling → obj → str
dumps using dumping hence

8) generators: return iterable set of items

9) docstrings: Triple quotes.

10). dict. 1→1 rel b/w keys & values.
dict indexed by keys.

11) ternary op: condn stmts.

12) *args: ? arg passed to func

**kwargs: ? many keyword arg

13) split : regex pat to split str → list

sub : substring

subn : similar to substring.

14) namespaces: all names in prog are unique.

dmp ns as dictionaries.

— x —

Flask
sqlite
inbuilt
no sepdb server

Thread not good
Global Interpret
LOCK
Mutex.
Multiproc →
Multithread - x

Kivy
↓
Mobile

GSM → global system Mobile communicat

Destructor:

→ automatically gets called as soon as life cycle of an object is finished.

→ free Memory.

Constructor:

↳ initialize objects.

↳ const is called when obj of class is created.

→ has param. to init att

OOPS: real / cars / Animals / People

Multithreading: Eg: games obj

* Prom of executing Multiple Threads simultaneously

① Thread → lightweight sub-process, the smallest subunit of processing. Multiprocessing & Multith → used to exh

* Thread uses shared Memory.

* Saves Memory, context switching takes less time

Lifecycle of Thread:

* New * Runnable * Running * Non-Runnable * Terminated

(instance of thread class is created)

How to create Thread:

By extending Thread class

By Impleming Runnable Interface.

Daemon Thread:

↳ service provides thread that provides services to User thread. Its life depends on Mercy of user threads

↳ JVM terminates thread automatically

↳ Low priority thread

↳ Its life depends on user threads

synchronization in Java:

↳ Ability to control access of any Multiple

Used to

↳ Prevent consistency problem

↳ Prevent thread interface

final → declaring an entity to be assigned only once

static → no re- starts executing from static class.
called by

finalize() → garbage collector → del no more references to object.

finally() → exits aft throwing an exception.
Exec piece of code in pro

exceptional handling

While executing Java code, diff Errors occur
It may be due to programmer / wrong input
Is Java normally stops & generates Error Msg
try → allows to define Blocks of code to be tested
for Errors while being Executed
catch → Define Block of code to be exec if Error occurs
Finally → lets execute code after try catch regardless
of the Result

Object: Instance of class

constructor:
* Method used to Initialize state of an object
* gets invoked at the Time of object creation
* Has No return Type
* same name as class name

Destructor:
→ Method which is automatically called when object is made of scope / destroyed
Dest name same as class Type name.

Incline function:
Used by compilers to Insert complete Body of function where func is used in Prog Src code

virtual function:
Member function of class & its functionality can be overiden in its derived class.

Friend function:
Friend of class that is allowed to access public, private, protected data in same class.