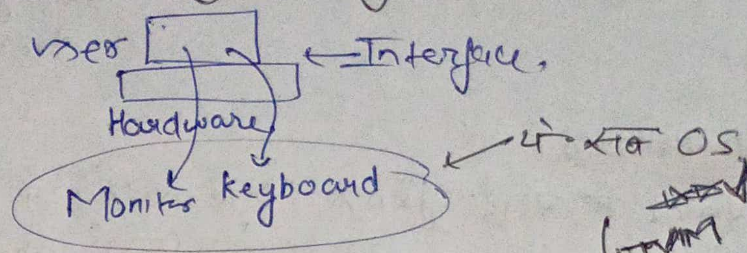
Object Oriented Programming In Java

Operating System →



Java Code (.java)
↓
Compiler (Javac)
JVM
(.class) byte Code
↓ Run (JRE)
machine Language.

(software) (JVM is platform dependent)

.java → .class (ByteCode) → JVM

JIT

OS

(JVM provides a platform-independent way to execute Code)

JDK
Java Developers Kit
(Contains tools needed to develop the Java program)

JRE
Java Run time environment
It Contains JVM and Java Packages Java Library.

Pillars of Object Oriented Programming

Encapsulation
- → Classes and objects
- → Constructors (overloading, chaining)
- → this keyword
- → static keyword
- → getter & setter.

Inheritance
- IsA vs hasA
- Type of Inheritance
- Super keyword
- Multiple Inher. {Diamond Problem a/w o.b.a.}

Polymorphism
- → Method overloading (Compile-time)
- → Method overriding (Run-time)

Abstraction and Data Hiding.
- → Access Modifiers
- → Packages
- → Abstract Class
- → Interface.

Classes Vs Objects

Classes ↓ logical entity (blue print of a object) class का कोई space use नहीं होगा।

Objects ↦ physical entity {occupies space} {instance of a class} (real world entity)

- Data member / Instance Variables → properties of objects
- Member functions / methods → behavior of object.

} getter & setter

Integer.MAX_VALUE ← properties

Integer.parseInt() ← fxn.

Junctions· Method,

ऐसे Jxn होते है जो किसी class के अन्दर आते है।

In General

\# Java मे सारे Jxn's Methods होते है।

(Jxn) → जिनको access करने के लिए object चाहिए होता है। उसे Method बोलते है।

(Method) → जिस Jxn को

{ Class is nothing but a Collection of objects, Data Members / Instance variable and Member junction / Methods }

(Java मे सारे Jxn Methods होते है Becose सब कुछ class के अन्दर हि होता है) हमारा Main भी class के हि अन्दर होता है Java मे।

Dynamic Memory Allocation

→ Memory Mapping → { Reference variable in stack Vs Object in heap }

\# Primitive data type (Stack) Vs object

→ Is Java a pure oops → Not

→ Wrapper classes

(Java 100% oops)

Stack int → Integer (Heap)

long → Long

Autoboxing, Unboxing

Q Primitive data type क्यों allow किया है Java मे

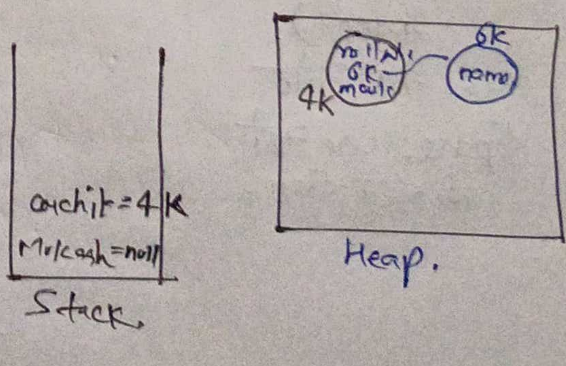
→ Bec. Primitive data type Stack मे बनते है जिससे Speed ज्यादा होती। Speed ज्यादा

C++ > Java > Python.

Student Mukesh;

Student archit = new Student();

↑ (reference)



Wrapper class

Java is an OO lang. and as said everything in Java is an object

What about the primitive?

As a Sol. of this Problem, Java allows you to include the primitive in the family of object by using what are called wrapper class.

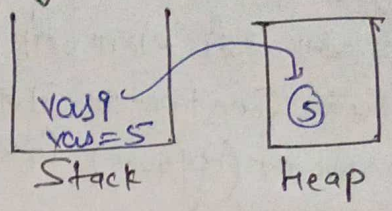
- ValueOf()
- ParseXxx()
- xxxValue()

} Methods

int var = 5; (Stack)

Integer var1 = var; (Heap)

or Integer var1 = new Integer(5)



System.out.println(var1 + " " + var1)

55 ← autounboxing

(जब हम Normal Arrays को Print करते हैं तो HashCode print होता है)

Deep Copy
↳ Swap/changes Persist

Shallow Copy
↳ Swap/changes Not Persist.

} Not actual definition

Constructor

- Default Constructor (Implicit)
- Default Constructor (Explicit)
- Parameterized Constructor
- Copy Constructor → Shallow Copy / Deep Copy

} Constructor overloading (Parameters diff.)

Student archit = new Student()

(class) (Reference) (Object) (Constructor)

Class का Name or Constructor का Name Same होगा।

और Constructor का कोई return type नहीं होता।

used to initialise the data member of a class.

जब हम class में कोई Constr. नहीं बनाते तो Implicit default Constructor बन जाता है।

जब हम खुद लिखते हैं But कोई Para. Pass नहीं करते तो explicit Constr. बोलते हैं।

Implicitly Constructor का return type. होता है जो होता है this.

अगर हमने खुद से Constructor (Para.) बनाया तो अब Implicit Constructor Call नहीं होता तो error आ जाएगा.
(Java Implicit Constructor अब नहीं देगा)
Explicit Constructor (default) लिखना → good practice.

Constructor Overloading
fun (int A, int B, String C) { } ✓
// fun (int A, int B, String C) { } (error)
fun (int A, String C, int B) { } ✓
(argument को देखा जाता है)
fun (double A, String C, int B) { }
↳ But इस object में int Pass करते तो भी error नहीं आएगी (Implicitly) (Constr.)
upcasting → Possible.
downcasting → Not poss.
(byte → short → int → long)

Constructor Overloading
↑ No of argument
↑ order of argument

Copy Constructor :- Student archit = new Student (10, "Archit", 90);
Student temp = new Student (archit);
Syso (temp.marks + " " + temp.name + " " + temp.rollNo);

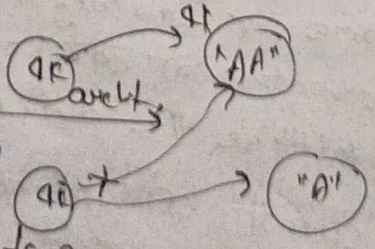
खुद से Copy Constructor भी बना सकते हैं,

Shallow Copy होती है,

```
Student (Student obj) {
    this.name = obj.name;
    this.marks = obj.marks;
    this.rollNo = obj.rollNo;
}
```

Deep Copy.

```
Student (Student obj) {
    this.name = new StringBuilder(obj.name);
    this.marks = obj.marks;
    this.rollNo = obj.rollNo;
}
```

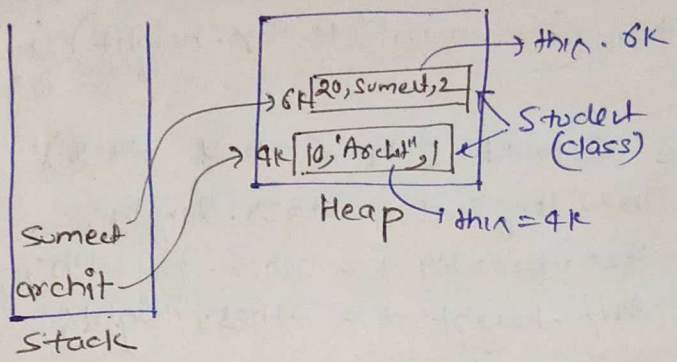


This Keyword

(Self-referential pointer)

- Resolving name collision b/w data member and parameter
- Constructor chaining
- Calling current class' data/ member method.
- Returning/Passing current class' object.

This → (Self referential pointer)



Constructor Chaining :-

एक Constructor से दूसरे Constr. में जाना (Call करना)

this() ← इससे लिखते हैं।

```
class Student {
    string name;
    int age;
    Student () {
        this();
    }
    Student (String name, int age) {
        this.string = string;
        this.age = age;
    }
    Student (int side) {
        this (side, side, side);
    }
}
```

(Code redundancy कम हो जाता है)

Stdnt () → Student(1)
↓
Student(1,1)

Constructor calling should be the first statement.

```
Student() {
    Syso(——);
    this(1);
}
```

(error आएगा)

Calling current class data members / Method.

```
public int volume(){
return this.length * this.breadth * this.height;
}
```

(अपनी class के members को ले रहे हैं directly इससे Speed Increase हो जाती है)

Returning / Passing current class object.

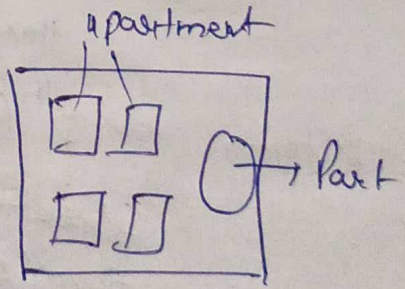
```
public int volume(area){
    return (this.area() * this.height);
    }
public Cuboid join (cuboid others){
    this.length += others.length;
    this.breadth += other.breadth;
    this.height += others.height;
    return this;
    }
public int area(){
    return (this.length * this.breadth);
    }

main{
    cuboid obj5 = obj4.join (obj3);
    }
```

Static KeyWord

(class के properties.)

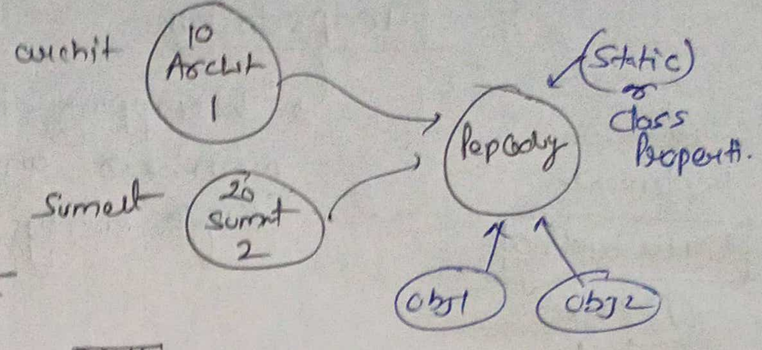
1. Data Member
   - Instance Variable vs Class Variable
   - Accessible Static members using class Name.



final static String University = "DTU";

(अब इसे change नहीं कर सकते)

②



उस object से भी
इसे access कर सकते
है और class के
Name से भी access कर
सकते हैं।
Class के Name से acc. करना
good practice.

Global variable are not present in Java.
(do not exist)
Bcz सारे Code class के अन्दर लिखा है।

② Member function / Methods

→ Why main() is Static? Because हम बिना object Create किया फिर भी वो without object run हो जाता है) (Instanciate नहीं करते)

→ this keyword in Static method?
↓
this का access नहीं मिलेगा
Bec. वो class को नहीं बल्कि
object को refer करता है।

- Static (class) member Calling Static (class) member → Yes
- Non-static (object) member Calling Static (class) member → Yes
- Non-static (object) member Calling non-static (object) member → Yes
- Static member (class) Calling non-static (object) member → No

③ Nested class → { Create objects of inner class w/o instantiating outer class }
↳ हम Inner वाली का object बना सकते w/o outer वाली के object बनाये।

Static block :-
- It run only once during the creation of final object of class.
- It runs only when class is loaded onto the RAM.

Encapsulation

↳ "Wrapping together the data members and member fxns into a single entity (known as class)"

↓ (Getters & Setters)

Implement

Data hiding (Private)

- Packages
- Access Modifiers → Public.
  - Private
  - Protected
  - Default

Inheritance

Parent class —(Pass Properties / datamembers / & Method.)→ Child class

Super class → Sub class.

Type

- Single Inheritance •→•
- Multi-level Inheritance •→•→•
- Heirarchial Inheritance (one parent, two children)

using extends

```
class car {
    int gear;
    int wheels;
    String engine;
}
class FlyingCar extends car
{
    String wings;
}
```

(इसमे car वाले कि सारी Prop. आ गई) →

FlyingCar का object car वाली सारी Pro. को acc. कर सकता है But car का object flyingCar को acc. नहीं कर सकता,

★★★ Multiple Inheritance is not Possible in Java.

(Diamond Problem)

Same property 2-2 बार आ रही है flying HydroCar.

Car (gear)
↙ ↘
(wings) FC   HydroCar (pedal)
↘ ↙
Flying HydroCar

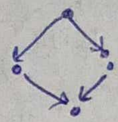
→ Problem है। (Duplicate Copy बन रही है)

{ Hybrid — Diamond of Death }

Multiple Inheritance

Why not possible in Java?

→ Hybrid Inheritance



(Diamond Problem)

→ Parent class method Collision resolution?

Flying, Hybrid
↳ gearChange();

Object class

{ जो भी class अगर वहा किसी को extend नही कर रही है तो By default वह object class को Inher. कर रही होगी, }

Object class — (object class) {Super class of all classes}

Method
- toString()
- equal()
- finalize()
- hashCode()
- clone()

Object → Car, Object → Flying Car → Car → Flying Car
(फिर Problem Multiple Inh.) (जो Java मे Not possible)

What type of Inheritance (Single / Multi Level)

Solution?
{ Multiple interface can be implemented by a Single class }

Object ✗ → Car → Flying Car
(अगर हमारी class किसी को extend करती है तो वह object class को Inher. नही करेगी)

Super Keyword

(immediate parent class object)

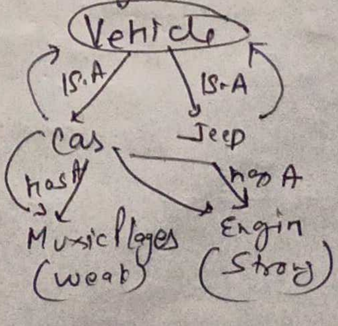
- Constructor chaining in Inheritance
- access data members / Methods of parent class.

```
class A{
   void f1(){ }
}
class B extend A{
   void f1()
   {
     super.f1();
   }
}
```

इसको Call कर पा रहा हु this से यह काम नही हो पाता,

IS A relationship (Inheritance)

→ Tightly coupled Relationship?
(Direct access through Super keyward)



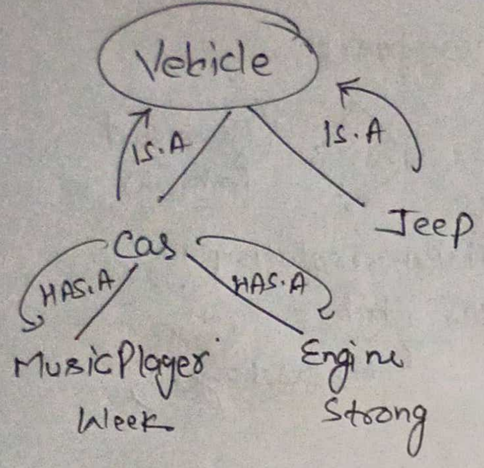
has A relationship. (Association)

(Linked List → Node)

→ loosely coupled Relationship (Accessing using object)

Ex → Nested class (LL → Node)
→ object Members

Type of Association
- Aggregation (weak)
- Composition (Strong)



Car
Music player } independent है तो week relationship

Aggregation
(Car में Music Player नहीं भी हुआ तो भी कोई Problem नहीं है इसलिए week)

Composition
(Car में engin होना जरूरी है इसलिए Strong)

Car
Engine } depend करते है आपस मे (Strong relationship)

Polymorphism

```
class FlyingCar extends car {
    String wings;
    FlyingCar(){
        super(5, "Petrol");
    }
    FlyingCar(String wings) {
        Super(5, "Petrol");
        this.wings = wings;
    }
    FlyingCar(int gear, String engine, String wings) {
        Super(gear, engine);
        this.Wings = wings);
    }
```

```
class A{
    A(){
        Syso("A")
    }
}
class B extends A{
    B(){ Super();
        Syso("B");
    }
}
```

By default भी यहा पर Super होता है जिससे हम जब B का object बनाते है तो A का Constructer Implement हो जाता है B से पहले।
means
B के object बनते हि B का Constructer Call होता है और A का Implement हो जाता है।

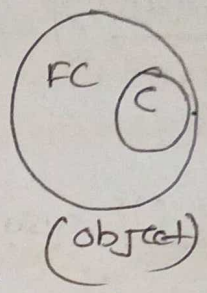
```
class A{
    int i;
    String j;
}
class B extend A{
    String z;
}
```

```
main(){
Public Static void main(—);
B obj = new B();
}
```

Option
- → B C11, B execute
- → A C11, A execute
- A·A → B C11, A execute
- → A C11, B execute.

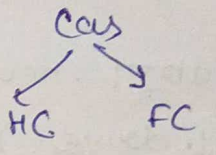
FlyingCar obj = new FlyingCar(5, "Electric", "Aluminium");

↳ Internally जब हम ~~exter~~ obj. बनाते हैं आगे उस का में को extend class है तो उसका भी object बन जाता है But हम normaly/direcly parent को C11 नहीं कर सकते, हम बस child के object से ही C11 कर पाएंगे।



Polymorphism

One action in many ways/forms. (एक ही नाम को बहुत तरह से use करना)



Types of Polymorphism

Compile-time Polymorphism or Static Polymorphism

{Method overloading}
- → Number of arguments
- → order of arguments
- → return type.

~~accept~~ Compile time पे ही पता लग जाएगा कि कौन सा fxn. चलना है। बाकी fxn का work तो run time पे ही होगा।

Run-time - Polymorphism or Dynamic Polymorphism

{Method overriding}

↳ Same
- → No of arguments
- → Types of arguments
- → Return type.

Or Dynamic Method Dispatch

Automatic type Conversion.

int n= scn.nextInt(); } Static
int [] arr = new int[n]; | Memory
Allocation
(Compile-time)

(Compile time पे memory all.
नही होती तो इस लिए बोल
रही है Bcz Static है)

Method Overriding :- (दो fxn के Same Name)

```
void car{
    drive(){ syso("car is running"); }
}
void flyingcar extends car{
    drive(){ syso("flyingcar is running");
}   }
main(){
    car obj1 = new car();
    obj1.drive();          → car is running
    FlyingCar obj2 = new FlyingCar();
    obj2.drive();          → Flying is running
    Car obj3 = new FlyingCar();
    obj3.drive();   —acc.→ (gear| Engine)   Flying is running.
    Flyingcar obj4 = new Car();
    obj4.drive();
}
```

(यहाँ run time Polym. होगा) → obj3.drive();

car → gear, Engine
↑ is A
FC → wings

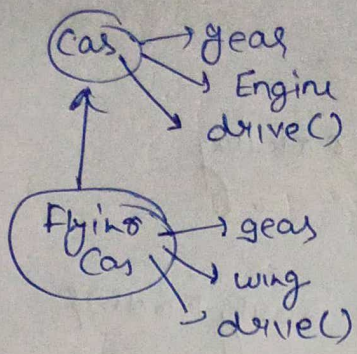
(error हो जायेगा)

obj4.drive() → Car → Flying Car.

(drive का reference Parent का obj. Not Possible)

Bec. Constructor of flying car is never Called.

जिसका reference है है उसके hisab से check करना है और उन्हें हि हम acc. कर सकते है, और वहा oversid भी हो रहा है,



Stack → Push Last → Pop Last

Queue → Push → addLast

reference (Stack)
- addLast()
- removeLast()

reference (Interface) Queue
- addLast()
- removeFirst()

Deque ← object
- addLast()
- addFirst()
- removeFirst()
- removeLast()

reference

Queue q = new ArrayDeque<>(); ← object.

(only 2) → (addLast / removeFirst) → Queue q = new ArrayDeque<>();

(all 4 can acc.) → Dequeue q = new ArrayDeque<>();

Dequeue ↑ reference

ArrayDeque ↑ object.

Method overriding

↳ के लिए same होना चाहिए

- No. of arguments
- Type of arg.
- Return type.

ClassName referenceName = new objectName()

↳ gives the functions which can call

↳ Provides the implementation of that function.

↳ this class should be same as the reference class or its child class.

(A) → (B)

parent class child class

A obj = new A(); } Possible

B obj = new B(); }

A obj = new B(); → runtime polymorphism

B obj = new A(); } Not possible

↳ Bcz Constructor of B never called

B's data members never initialized.

Early Binding Vs Late Binding

Early Binding

- Normal Methods {w/o inheritance}
- Method overloading

Note :- Static Methods Cannot be overriding

Becouse → Static does not depend on object whereas overriding depends on objects type.

Late Binding

- Method overriding

```
class A {
  public void earlyBind(){
      syso("Early Bind");
  }
  Public void lateBind(){
    syso(" LateBind in Parent class");
  }
}
class B extends A {
  public void lateBind(){
    syso("LateBind in child class");
  }
}
Public class Main{
 Public static void main (String[] args){
      A obj = new B();
   obj.earlyBind(); → early Bind
   obj.lateBind(); → LateBind in child class
}
```

Abstraction

(ADT) (Abstract datatype)

↳ Hiding unnecessary details/ Complexity of class outside world and providing class for the abstract view {showing necessary details}

Implementation:
- → Abstract classes {Partial Abstraction} (0–100)
- ↳ Interface {full Abstraction} (100%) (Java 9 से पहले)