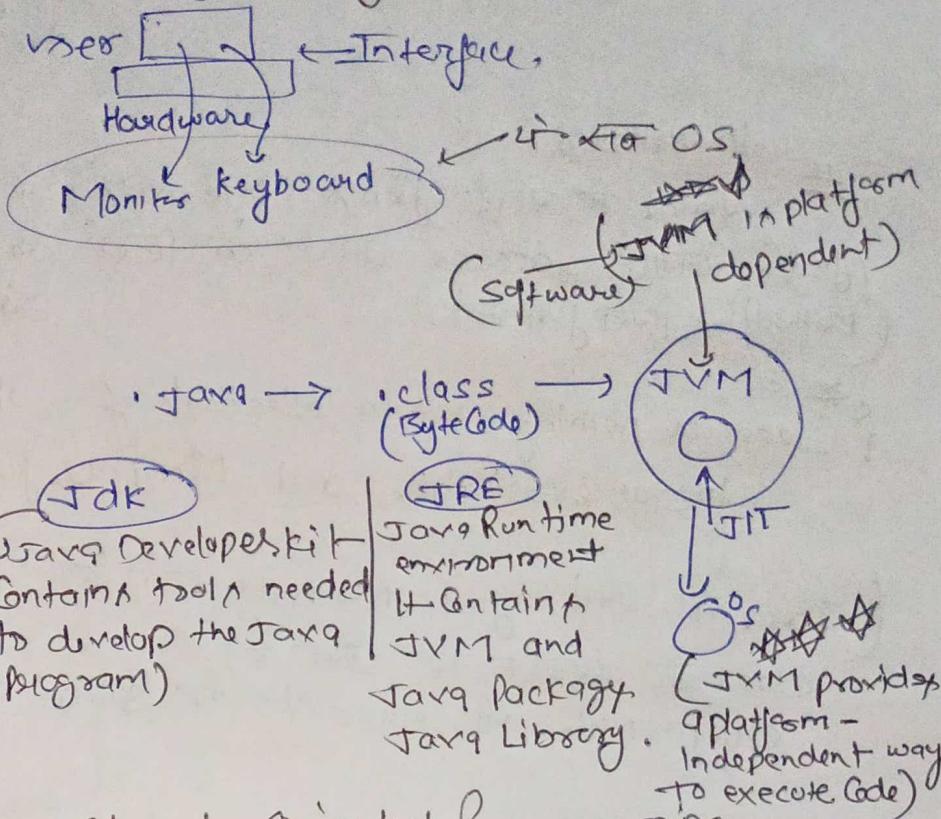
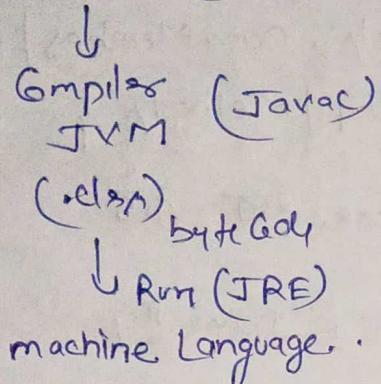


Object Oriented Programming In Java

Operating System →



Java Code (.java)



Pillars of Object Oriented Programming

Encapsulation

- Classes and objects
- Constructors
- Overloading, chaining {
- this keyword
- static keyword
- getters & setters,

Inheritance

IS-A vs has-A

Type of Inheritance

Super keyword

Multiple Inher.

{ Diamond Problem
already. }

Poly morphism

→ Method overloading (Compile-time)

→ Method overriding (Run-time)

(real world entity)

Classes Vs Objects

physical entity { occupies space }
{ instance of class }

logical entity

(blueprint of a object)
class की ओर
प्रायः उनके नहीं
जैविक संरचना,

Abstraction and Data Hiding.

→ Access Modifiers

→ Packages

→ Abstract Class

→ Interface,

- Data member / Instance Variable

→ Properties of object

getters & Setters

- Member function / method

→ behavior of object.

Integers. MAX_VALUE ← properties

Integers. pauseInt () ← fxn.

functions.

Method
जैसे jxn एवं जैसे वाली
class के अन्दर आते हैं,

In General

Java में सारे jxn's Methods एवं इ

(jxn) → जिनको access करने के लिए object chain दिया है।
(Method) → जिस jxn को उसे Method बोलते हैं।

{ Class is nothing but a Collection of objects, Data Members /
InstanceVariable and Member function / Methods }

(Java ने सभी jxn Methods एवं ऐसे कहे
class के अन्दर हैं दिया है।)
मात्र Main भी class के अन्दर हैं Java ने।

Dynamic Memory Allocation

→ Memory Mapping → { Reference Variable in Stack
vs Object in heap }

Primitive data type vs object

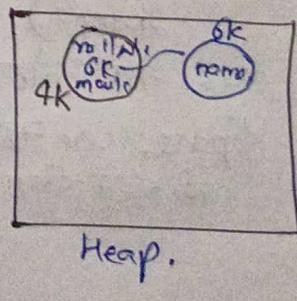
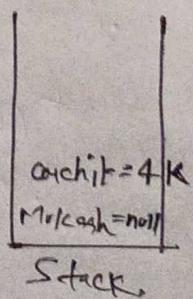
↓
 (for box cons) → ↓ Is Java a pure OOPS → Not
 ↓ wrapper classes → Stack → Head
 ↓ Autoboxing → Int → Integer
 ↓ Unboxing → Long → Long
 ↓ primitive data type but allow for it in Java में

Q Primitive data type but allow for it in Java में जिसके
 → Bec. Primitive data type stack में बनते हैं जिसके
 ↓ Speed than Java द्वारा दिया गया है।
 C++ > Java > Python.

Student Mukesh;

Student archit = new Student();

(Reference)



Wrapper class

Java is an OOP lang. and it's said everything in Java is an object.

Q What about the primitives? As a soln. 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()

Deep Copy

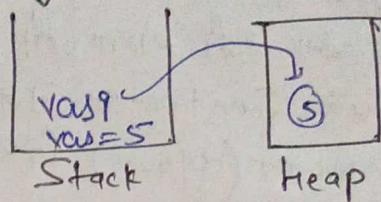
Swap changes persist

Shallow Copy

Swap changes not persist.

int var = 5; (Stack)

Or Integer var1 = var; (Heap)
Integer var2 = new Integer(5)



System.out.println(var1 + " " + var2)

55 ← auto unboxing

(जो एक Normal Array के बहु छंदों
के लिए hashCode, print आदि हैं)

Not actual
definition

Constructor

- Default Constructor (implicit)
- Default Constructors (explicit)
- Parameterized Constructor
- Copy Constructor

Constructor overloading (parameters diff.)

Student archit = new Student()
(class) (refer) (Object) (Constructor)

class or Name or Constructor or Name same होगा।

अब Constructor की return type नहीं होती, used to initialise the data members of a class.

जब एक class में कोई Constr. नहीं बनाते तो implicit default

Constructor बन जाता है।

उस एक बुद्धि से लियते हैं but कोई पॉर्टेंस नहीं करते हैं।

explicit Constr. बनाते हैं।

Implicitly Constructor का Return type. एकता न हो सकती है
this.

अगर हमने ऐसा किया कि Constructor (Param) बनाया कि अब Implicit
Constructor का नहीं होता कि यहाँ आप भालूगा.
(Having Implicit Constructor अब नहीं करा)

Explicit Constructor (default) लिखना → good practice.

Constructor Overloading

fun (int A, int B, String C) { } ✓
II fun (int A, int B, String C) {} (error)
fun (int A, String C, int B) {} ✓
(Argument order भला है)

fun (double A, String C, int B) {}

↳ But इस object का int pass कर देती है तो वही अस्पष्ट हो जाता है
(Implicitly) (Comtr.)

Upcasting → Possible.

Downcasting → Not Poss.

(byte → short → int → long)

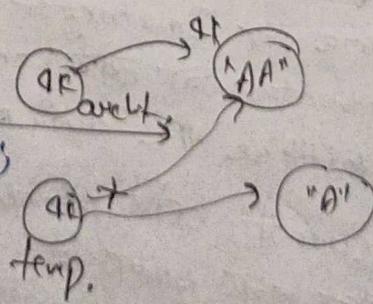
Copy Constructor :- Student archit = new Student ("Archit", 20);
Student temp = new Student (archit);

System.out.println (temp.name + " " + temp.rollNo);

एक Copy Constructor का बनाया गया,

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

deep Copy {
 Student (Student obj) {
 this.name = new String (obj.name);
 this.marks = obj.marks;
 this.rollNo = obj.rollNo; } }



Thin Keyword

(self-referential pointer)

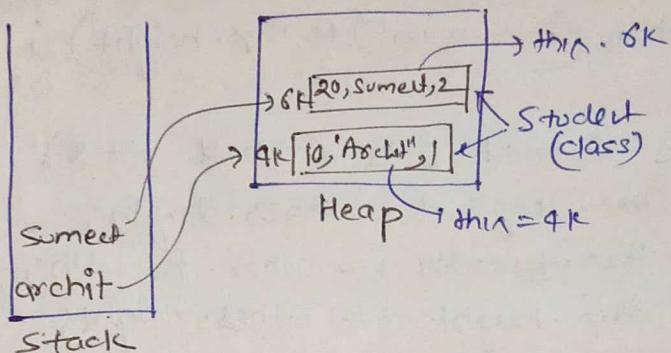
Resolving
name Collision
by data member
and parameters

Constructor Chaining

Colling Current
class data
mechan
method.

Returning / Passing
current class object.

This → (self referential pointer)



Constructor Chaining :-
 एक Constructor से दूसरे Constr. का जावा (GII करनी)
 $\text{this}() \leftarrow \text{हमसे Return हो।}$

```
class Student {
```

```

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

```

(Code Redundancy OR ET और सारा E)

$\text{Student}() \rightarrow \text{Student}(1)$

Constructor calling should be the first statement.

Student();
Systoc(); } } (earas BMSIT)
this(); }

Calling current class data members / Method.

```
public int volume() {
    return (thin.length * thin.breadth * thin.height);
}
(अपनी Class के members को एक रूप से
directly & सरल Speed increase होती है)
```

Returning / Passing current-class object.

```
public int volume(area) {
    return (thin.area() + thin.height);
}
```

```
public Cuboid join (cuboid other) {
    thin.length += other.length;
    thin.breadth += other.breadth;
    thin.height += other.height;
    return thin;
}
```

```
public int area() {
    return (thin.length * thin.breadth);
```

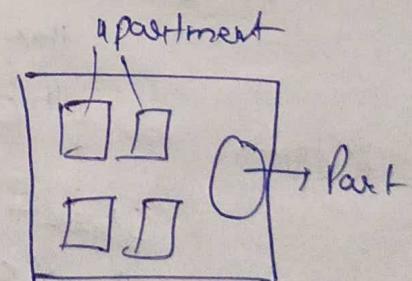
```
main {
    cuboid obj5 = obj4.join(obj3);
}
```

Static Key Word

① Data Member ↗ class for properties.

Instance Variable ↗ Accessible

vs Static members using
Class Variable class Name.

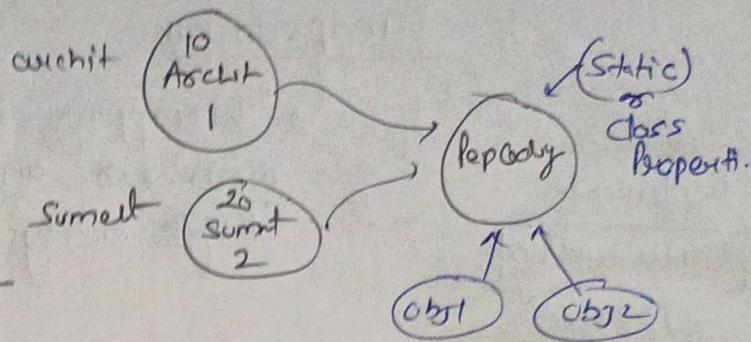


final static String University = "DTU";

(Static का value बदलने की क्षमता नहीं है)

②

जब object के लिए
इसके access करना तो
उसके class की
Name के लिए access करना
होता है।
class की Name के acc. करना
good practice.



Global variable are not present in Java.
(do not exist)
Because code class की static फलाई है,

② Member Function / Methods

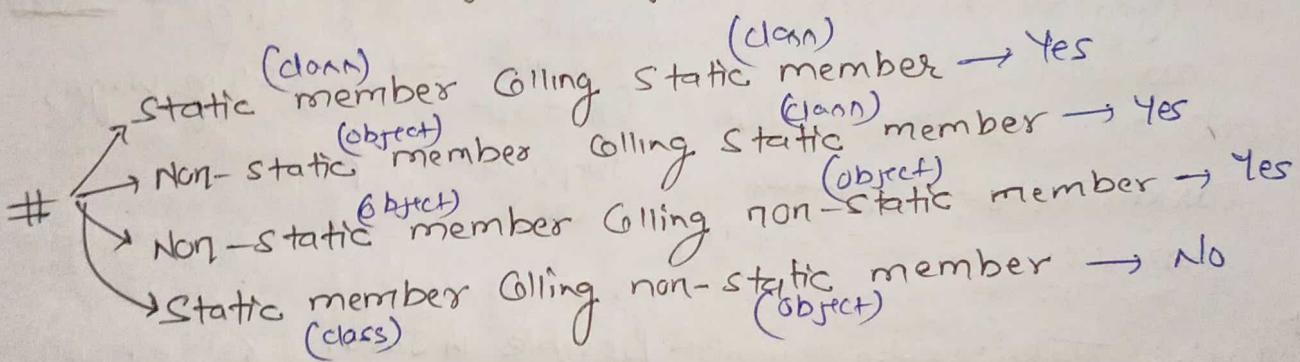
→ Why main() is static? Because

→ this keyword in static method?

↓
this का access करना
(Bec. at class की नहीं है)
Object की रूप से है।

(Incorporate कर दें)

ET ऐसे object Create
for कि func का कर
without object run
कर दिया है।



③ Nested class → { Create objects of inner class }
↑ up instantiating outer class
ET Inner object की object बनाते हैं
up outer object की object बनाते हैं।

static block:-

- It runs 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 functions into a single entity (known as class)"

Implement

Data hiding

() , ()

Packag^es

Access Modifiers → Public.

Private Protected Default

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 8010 8011 8012 8013 8014 8015 8016 8017 8018 8019 8020 8021 8022 8023 8024 8025 8026 8027 8028 8029 8030 8031 8032 8033 8034 8035 8036 8037 8038 8039 8040 8041 8042 8043 8044 8045 8046 8047 8048 8049 8050 8051 8052 8053 8054 8055 8056 8057 8058 8059 8060 8061 8062 8063 8064 8065 8066 8067 8068 8069 8070 8071 8072 8073 8074 8075 8076 8077 8078 8079 8080 8081 8082 8083 8084 8085 8086 8087 8088 8089 8090 8091 8092 8093 8094 8095 8096 8097 8098 8099 80100 80101 80102 80103 80104 80105 80106 80107 80108 80109 80110 80111 80112 80113 80114 80115 80116 80117 80118 80119 80120 80121 80122 80123 80124 80125 80126 80127 80128 80129 80130 80131 80132 80133 80134 80135 80136 80137 80138 80139 80140 80141 80142 80143 80144 80145 80146 80147 80148 80149 80150 80151 80152 80153 80154 80155 80156 80157 80158 80159 80160 80161 80162 80163 80164 80165 80166 80167 80168 80169 80170 80171 80172 80173 80174 80175 80176 80177 80178 80179 80180 80181 80182 80183 80184 80185 80186 80187 80188 80189 80190 80191 80192 80193 80194 80195 80196 80197 80198 80199 80200 80201 80202 80203 80204 80205 80206 80207 80208 80209 80210 80211 80212 80213 80214 80215 80216 80217 80218 80219 80220 80221 80222 80223 80224 80225 80226 80227 80228 80229 80230 80231 80232 80233 80234 80235 80236 80237 80238 80239 80240 80241 80242 80243 80244 80245 80246 80247 80248 80249 80250 80251 80252 80253 80254 80255 80256 80257 80258 80259 80260 80261 80262 80263 80264 80265 80266 80267 80268 80269 80270 80271 80272 80273 80274 80275 80276 80277 80278 80279 80280 80281 80282 80283 80284 80285 80286 80287 80288 80289 80290 80291 80292 80293 80294 80295 80296 80297 80298 80299 80300 80301 80302 80303 80304 80305 80306 80307 80308 80309 80310 80311 80312 80313 80314 80315 80316 80317 80318 80319 80320 80321 80322 80323 80324 80325 80326 80327 80328 80329 80330 80331 80332 80333 80334 80335 80336 80337 80338 80339 80340 80341 80342 80343 80344 80345 80346 80347 80348 80349 80350 80351 80352 80353 80354 80355 80356 80357 80358 80359 80360 80361 80362 80363 80364 80365 80366 80367 80368 80369 80370 80371 80372 80373 80374 80375 80376 80377 80378 80379 80380 80381 80382 80383 80384 80385 80386 80387 80388 80389 80390 80391 80392 80393 80394 80395 80396 80397 80398 80399 80400 80401 80402 80403 80404 80405 80406 80407 80408 80409 80410 80411 80412 80413 80414 80415 80416 80417 80418 80419 80420 80421 80422 80423 80424 80425 80426 80427 80428 80429 80430 80431 80432 80433 80434 80435 80436 80437 80438 80439 80440 80441 80442 80443 80444 80445 80446 80447 80448 80449 80450 80451 80452 80453 80454 80455 80456 80457 80458 80459 80460 80461 80462 80463 80464 80465 80466 80467 80468 80469 80470 80471 80472 80473 80474 80475 80476 80477 80478 80479 80480 80481 80482 80483 80484 80485 80486 80487 80488 80

Inheritance (pass) Parent class → Sub class .
Super class → Child class
Properties / datamembers / Method.)

Type

→ Single Inheritance → o

→ Multi-level Inheritance

→ Hierarchical Inheritance

flying car का object
car का लोटी आदि प्रा.
का acc. का संकल्पना दूर
का object flying car
का acc. बढ़ी का संकल्पना

(इसमें
कौं दाये
कि आवी
प्रोप. आ
हाँ)

using extend()

class car {
 int gears;
 int wheels;
 String engine;

class FlyingCar extends car
→ {
 String wings;

 ⚡ Multiple Inheritance is not possible in Java.

Same property 2-2 ~~9T~~ 3T

~~Re~~ & flying Hydro Car.

at →
Problem
S

(Duplicate Gpy
बन रही
है)

{ Hybrid Diamond of Death }

(Diamond Problem)

car (g)
↓
(wing) fc

HydroCas (pedal)

Flying hydras

Multiple Inheritance
Why not possible in Java?

Hybrid Inheritance

Parent class
method Collision resolution?

(Diamond Problem)

Flying, hybrid

geekChange();

Object class

{ उत्तम class असर देती
किसी ने extend नहीं कर सकती है
नियम द्वारा Object class का Inter. कर सकती है, }

Object class

Method

→ toString()
→ equals()
→ finalize()
→ hashCode()
→ clone()

(Object class)

{ Super class of all classes }

Object objectA

↓ car → FlyingCar

(FBC Problem Multiple Inh.)
(But Java is Not Possible)

What type of Inheritance
(Single / Multi Level)

{ Multiple Interface
can be implemented
by a Single class }

Solution

Object objectB

↓ car → FlyingCar

(Basic एसी class
किसी ने extend करती है कि
वे Object class का Inter.
नहीं कर सकती)

Super immediate
parent class obj.

Super Key Word

Constructor
chaining in Inheritance

class A{

void f1(){ }

class B extends A{

void f1()

{ Super.f1(); }

accept data
members / Methods
of parent class.

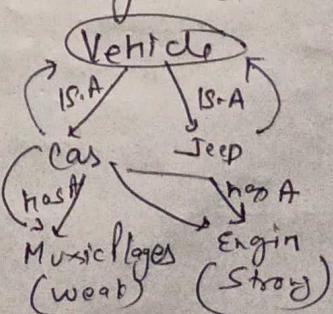
इसको GII
कहा जा
रहा है
यह से पहुँच
काम नहीं
एवं पाठा,

is A relationship
(Inheritance)

→ Tightly Coupled

Relationship

(Direct access
through Super
keyword)



has A relationship

(Association)

(linked List
TNode)

→ loosely coupled

Relationship

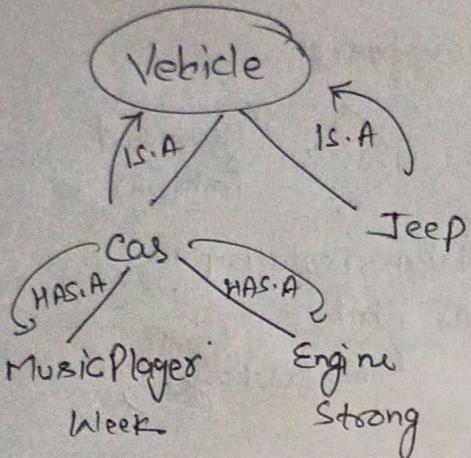
(Accessing using
object)

Ex → Nested class
(List Node)

→ object Members

Type of Association

Aggregation Composition
(weak) (strong)



Aggregation
 (Car & Music Player तकी नहीं
 इसी के बाहर Problem होता है
 जैसे एक week)

Car ~ Independent
 Music Player ~ weak relationship

Composition
 (Car & engine एक भूमिका में सम्पूर्ण
 Strong)

Car ~ depend on each other
 Engine ~ Strong relationship

Polymorphism

```

class FlyingCar extends Car {
    String wings;
    FlyingCar() {
        super("petrol");
    }
    FlyingCar(String wings) {
        super("petrol");
        this.wings = wings;
    }
    FlyingCar(int gear, String engine, String wings) {
        super(gear, engine);
        this.wings = wings;
    }
}
  
```

```

class A {
    A() {
        System.out.println("A");
    }
}

class B extends A {
    B() {
        System.out.println("B");
    }
}
  
```

```

class B {
    B() {
        super();
        System.out.println("B");
    }
}
  
```

By default
 मुझे पढ़ा पड़े
 Super होता है
 जिससे हम जब

B का object बनाते हैं
 at A का constructor
 implement नहीं होता है
 B का पहला,

means
 B का object बनते हैं

B का constructor की होता है
 और A का implement नहीं होता है।

```

class A{
    int i;
    String j;
}
class B extends A{
    String z;
}

main(){
    public static void main(){
        B obj = new B();
    }
}

```

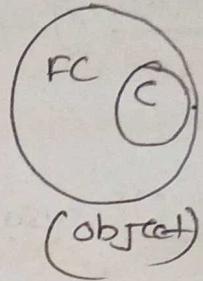
option

A::A	→ B::B, B execute
	→ A::A, A execute
	→ B::B, A execute
	→ A::A, B execute.

FlyingCar obj = new FlyingCar("5,"

"Electric", "Aluminium");

↑ Internally जब हम extend obj.
बनाते हैं अगर उस का मेरे को extend करते हैं तो उसका भी object बन जाता है।
But हम normally directly parent को करते हैं और
कर सकते हैं, हम वह child को object से ही कर पाएँगे।

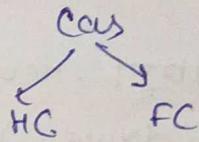


Polymorphism

One action in many ways/ forms.

(एक दिनाम को बदलते हुए से
परे लेना)

Types of Polymorphism



Compile-time Polymorphism
or
Static Polymorphism

{Method overloading}

- Number of arguments
- Order of arguments
- return type.

Except Compile-time परे ही
पता लग जाएगा कि कौन सा

fxn. चलना है।

बाकी fxn का work होता

run time परे ही होता।

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 = sen.nextInt(); } static
 int [] arr = new int[n]; } Memory Allocation
 while time & memory all.
 । से ज्ञान लिया गया (Compile-time)

(Compile time पे memory all
वही छोटी हो जिस लिए बाल
इसी है Bcz static है)

(Compile-time)

Method Overriding :- (if fxn of same name)

void cass

derive () { sys("can't running"); }

void flyingCarExtends car

```
drive() {  
    System.out.println("flyingCar is running");  
}
```

main of

```
Car obj = new Car();
```

obj. derive(); → can't run

FlyingCar obj2 = new FlyingCar();
obj2.drive(); → Flying is running

```
Car obj3 = new FlyingCar();
```

→ obj3.drive(); acc. → (Geo|Engine) ↑ isA Engine
Flight is running.

(माटे)
soil
time
Polymer
EP-II)

Flyingcar obj4 = new Flyingcar();

Car $\xrightarrow{\text{Is A}}$ Geog Engine

जिसका अधिकारी
है है उसके बिंद से Clock
लगना है और उकड़ि
एक acc. कर सकते हैं,
और वहाँ overwatch
भी हो सकते हैं।

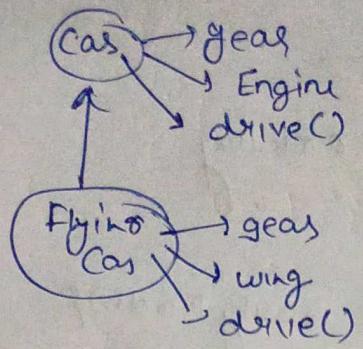
obj4 derive();

जाखगा

→
cas
↓
flyin boy.

✓ drive CT
referen. as
Patient CT
obj.
Net Possibl.)

Bec. Constructors of flying cars in never called.



reference
Stack

- Add Last()
- Remove Last()

reference
Queue

- Add Last()
- Remove First()

reference
Queue

- Add Last()
- add First()
- remove First()
- remove Last()

(only 2) → Queue q = new Queue();
 ↓
 addLast() → Queue q = new Array(); Deque < T>();
 ↓
 removeFirst() → Queue q = new Array(); Deque < T>();
 ↑
 reference object.

Method overriding

→ Inherit same name differ

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

→ gives the functions which can call

ClassName referenceName = new

ObjectName()

↓ Provides the implementation of that function.

→ This class should be same as the reference class or in child class.

Parent class → Child class

A obj = new A();

B obj = new B();

A obj = new B(); → runtime polymorphism

B obj = new A(); } not possible

Because constructor of B never called
B's data members never initialized.

Early Binding

Vs

Late Binding

- Normal Methods
w/o Inheritance

- Method overloading

Note :- static Methods
cannot be overriding.
Because → static does not
depend on object
whereas overriding depends
on objects type.

- Method overriding

```
class A {
    public void earlyBind() {
        System.out.println("Early Bind");
    }
}

public void lateBind() {
    System.out.println("LateBind in Parent class");
}

class B extends A {
    public void lateBind() {
        System.out.println("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
    }
}
```

(ADT) (Abstract datatype)

→ Hiding unnecessary details / Complexity of class
for the outside, would add providing
abstract view & showing necessary details

Implementation

↳ Abstract class {Partial Abstraction}

↳ Interface {Full Abstraction}

Faqq (100%)
संपूर्ण

(O-O)

