

Object Oriented Programming in Java

Interview Questions

- Q1) (a) What are the differences between Procedural Programming and Object oriented programming languages?
- (b) What are the drawbacks of procedural programming language?
- (c) What are the needs/advantages of object oriented programming language?

Procedural Programming

- Focuses on functions & logic!
- less reliable & secure
 - ↳ No access modifiers
- less reusability & difficulty in maintaining the codebase
- Example: → C Programming language

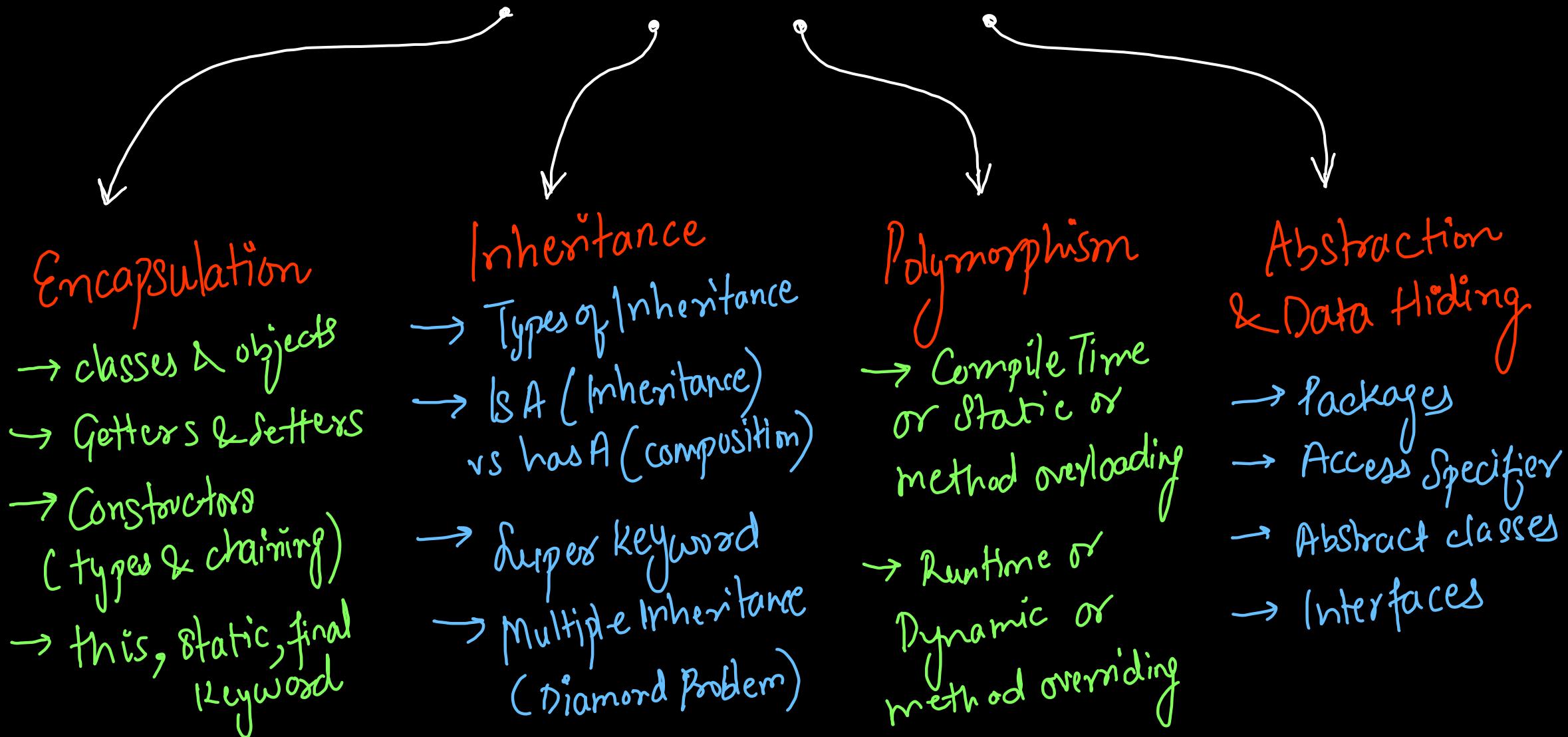
Note: → Javascript is functional programming language.

Object oriented Programming

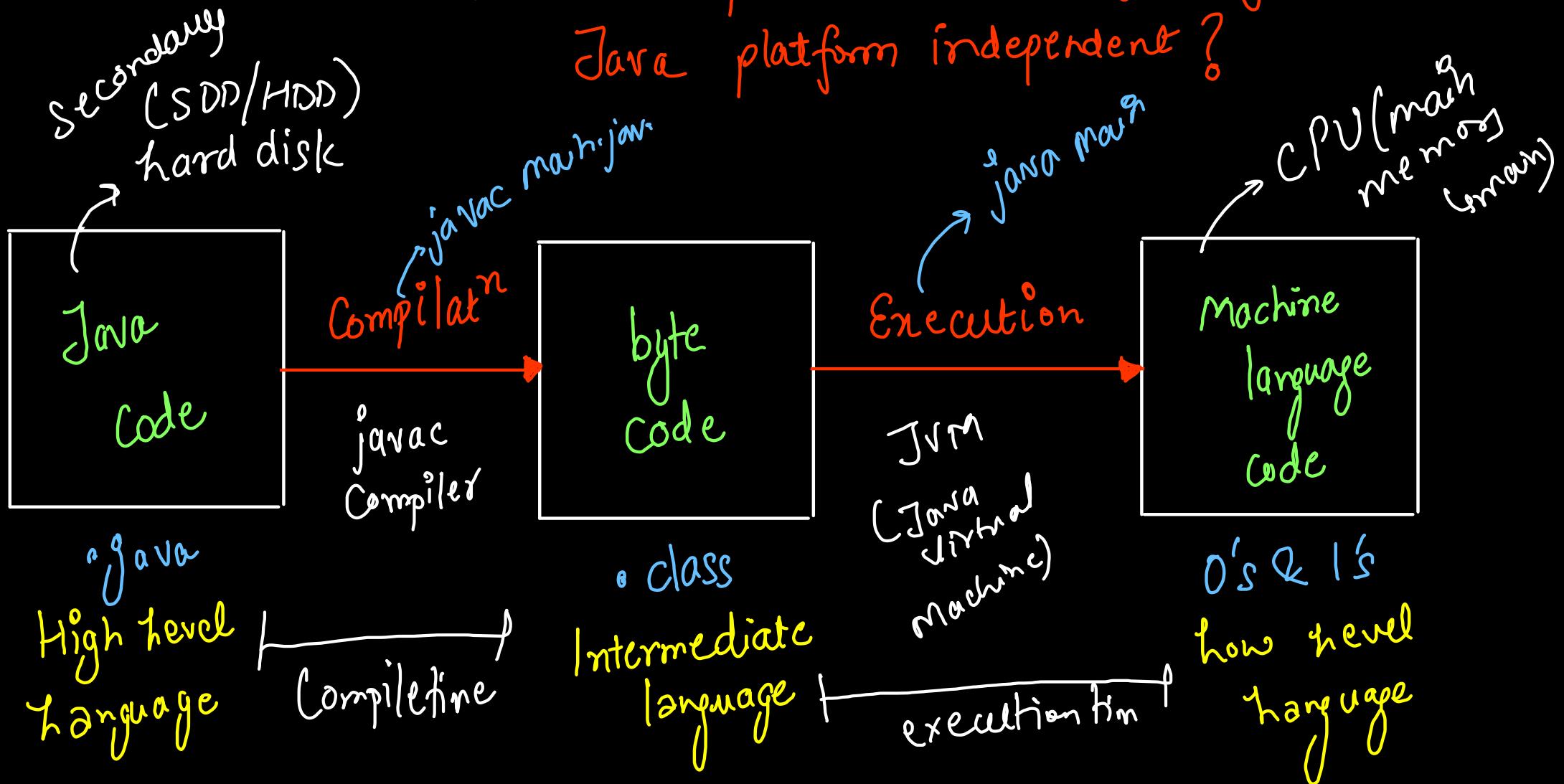
- Focuses on data & objects!
 - ↳ Data members & member functions
- More reliable & secure
 - ↳ Data hiding & Encapsulation
- More reusability & easy maintainability
 - ↳ Due to Inheritance & Polymorphism
- Example: → C++, Java, Python, C#
 - ↳ higher order functions

Need of Object Oriented Programming

PILLARS OF OOPS



~~(Q) Explain Software Development Process in Java. What are various components in Java? Why is Java platform independent?~~



Java Architecture

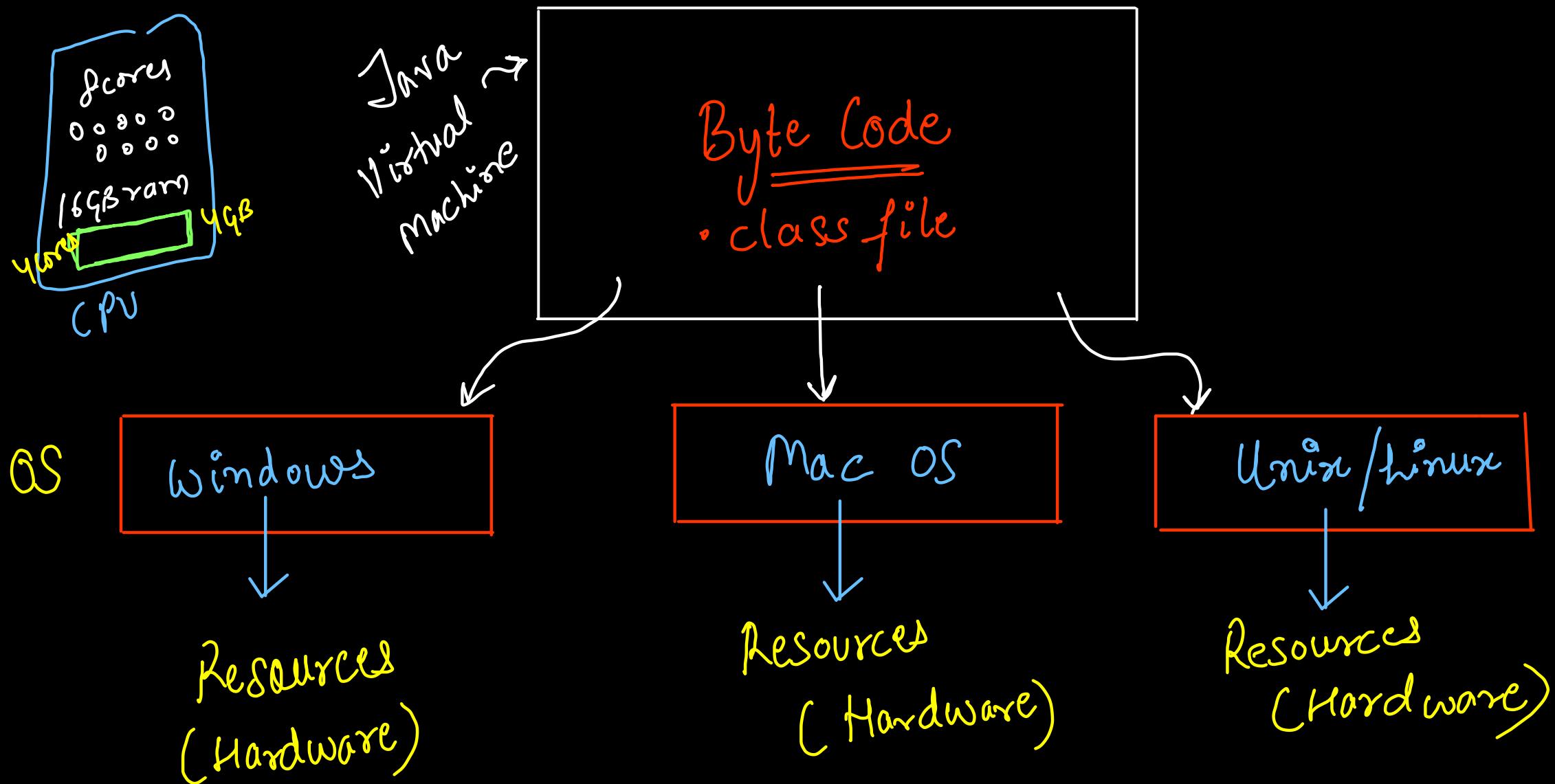
Oracle

Java Development Kit (JDK) = JRE + Development Tools
{ Interpreter, Class Loader, Compiler (javac) }
linker,

Java Runtime Environment (JRE)
= JVM + library classes

Java Virtual Machine (JVM)

Java Is Platform Independent



* Games
* desktop apps

C++ → 2 faster → Java

• .cpp source code

compile

machine
dependent

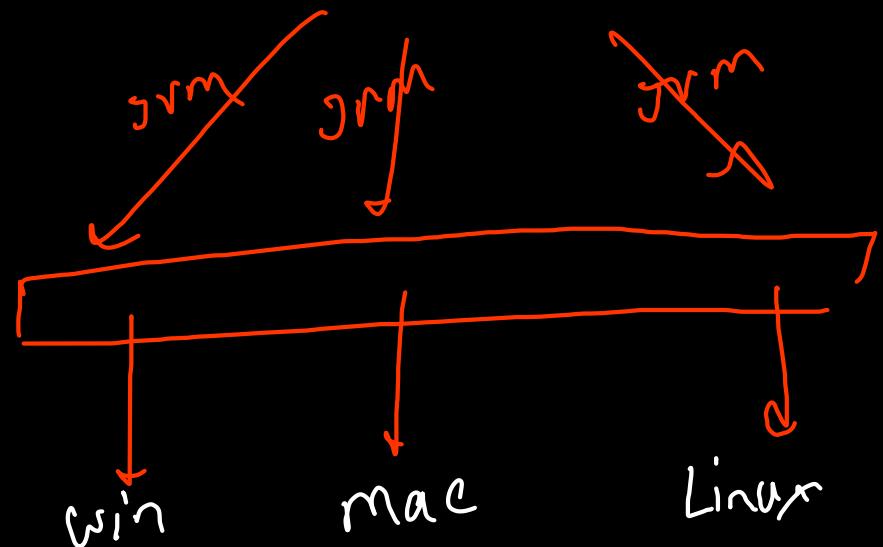
• .exe executable file

faster

run on operating syst

• .java source code

• .class byte code



Q) What are differences between Program & Process? What do you mean by process memory layout? Explain the steps in process creation/ program execution?

(secondary storage)
Hard disk # Program vs Process

Program → Code / Set of Instructions (.java)

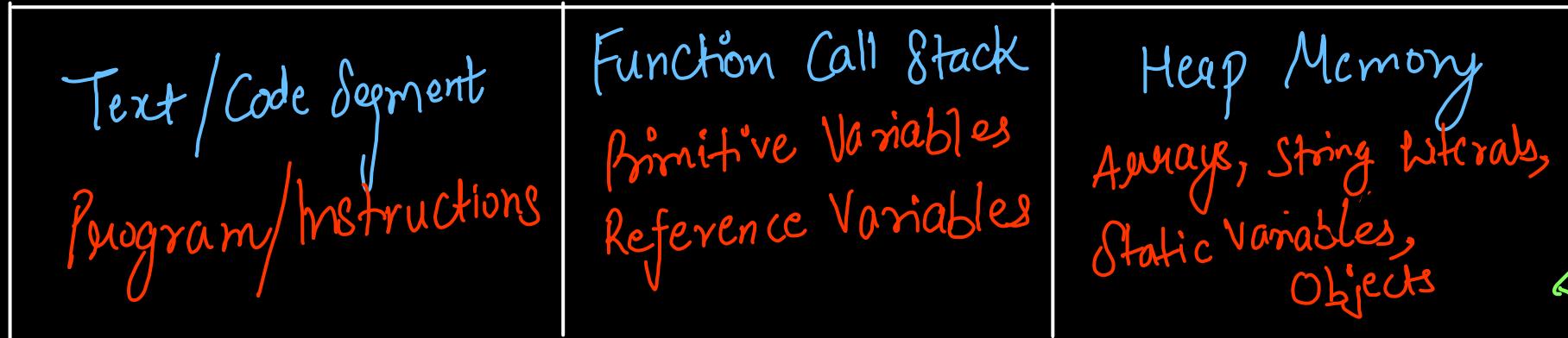
Program under execution is known as process

Process = Program (Code) + Stack & Heap Memory
+ Other resources

(CPU, disk, Network, I/O file, etc)

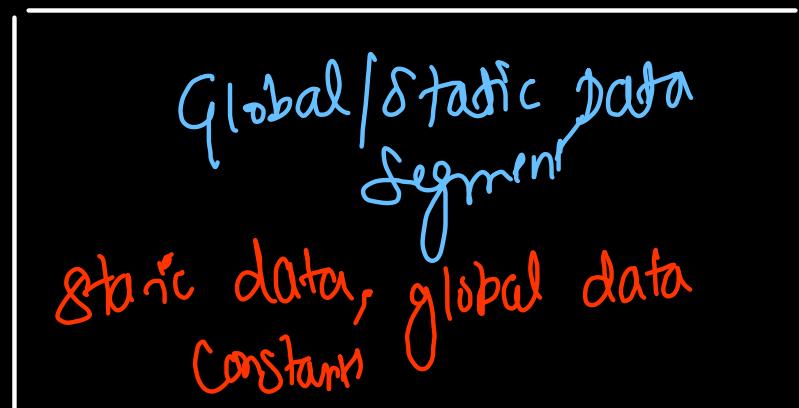
RAM

Process Memory layout (Random Access memory) main memory

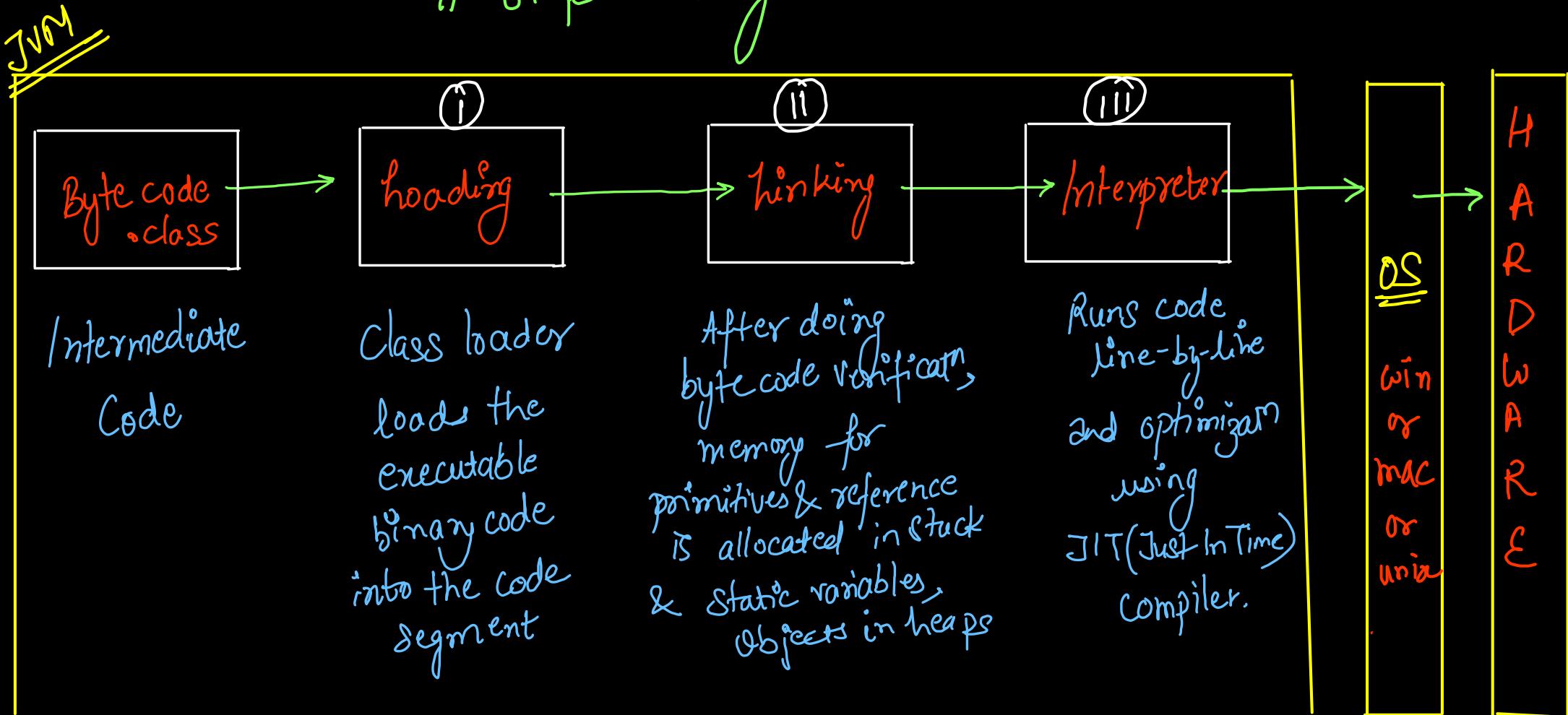


function scope (int, char, boolean, short, byte, long, float, double)
lifetime = fn

scope = program
(global)
lifetime = program



8 Steps in Program Execution



Q) How are classes different from objects in Java?
Explain with the help of real world example.

Class	Object
<ul style="list-style-type: none">logical Entity	<ul style="list-style-type: none">Physical / Real-world entity
<ul style="list-style-type: none">do not occupy stack or heap memory space	<ul style="list-style-type: none">Occupies both stack & heap memory space
<ul style="list-style-type: none">Blueprint for an Object	<ul style="list-style-type: none">Instance of a class
<p>Data Members (Properties)</p> <p>Member functions (Behavior)</p>	<p>Reference Variable in Stack</p> <p>Actual Object in Heap</p>

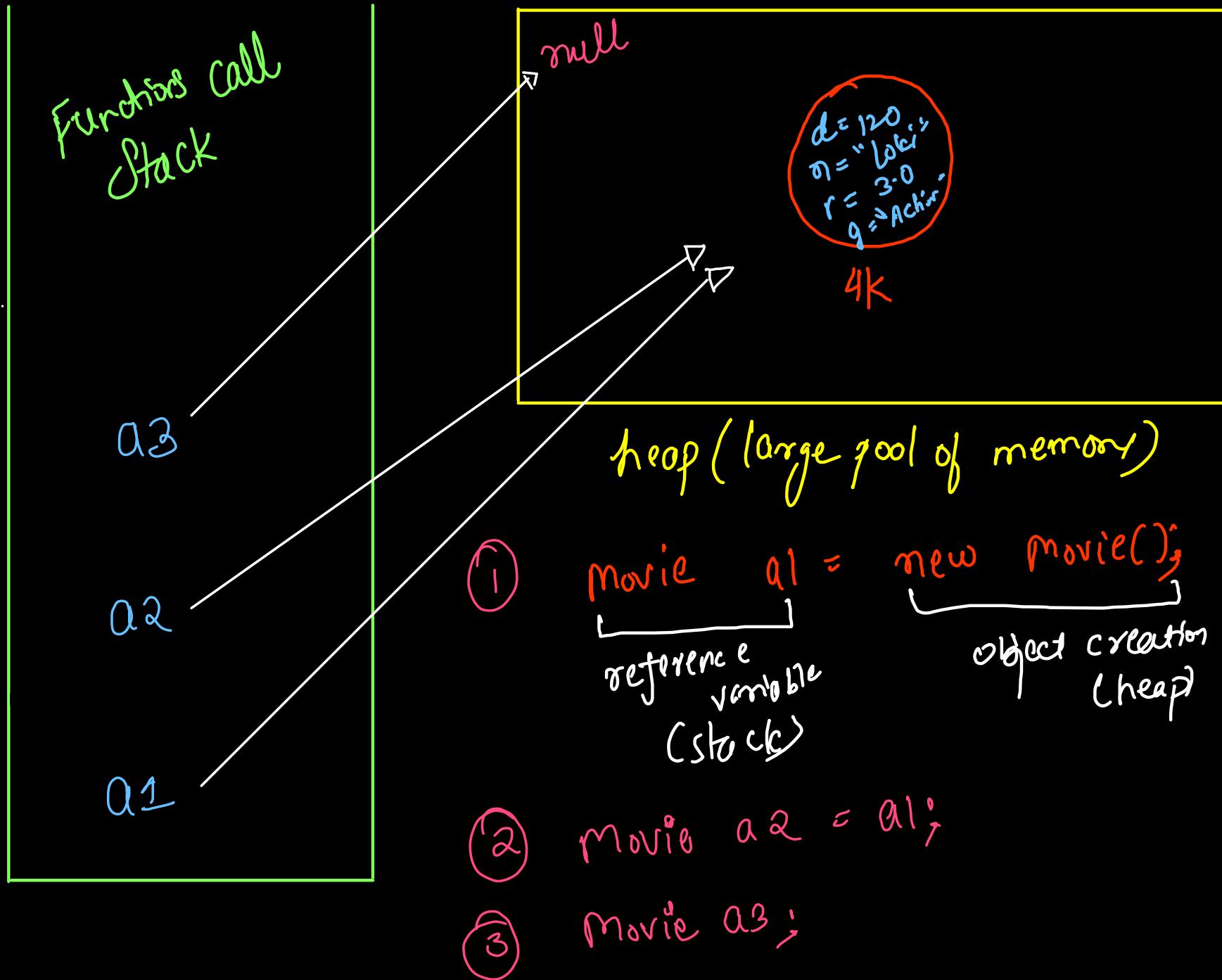
```
class Movie{  
    int duration;  
    String name;  
    double ratings;  
    String genre;  
}
```

Class → Text Segment
(Class Header)

```
Movie avengers1 = new Movie();  
avengers1.duration = 120;  
avengers1.name = "Avengers Loki";  
avengers1.ratings = 4.0;  
avengers1.genre = "Action";  
  
// System.out.println(avengers1);  
System.out.println(avengers1.name + " " +  
                    avengers1.duration + " " +  
                    avengers1.genre + " " +  
                    avengers1.ratings);
```

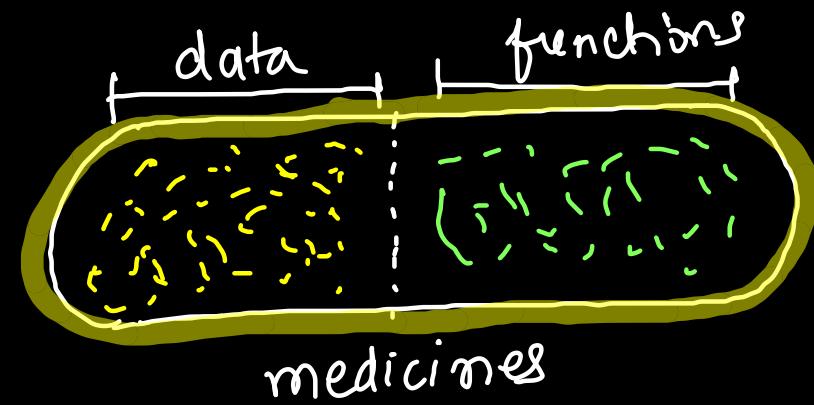
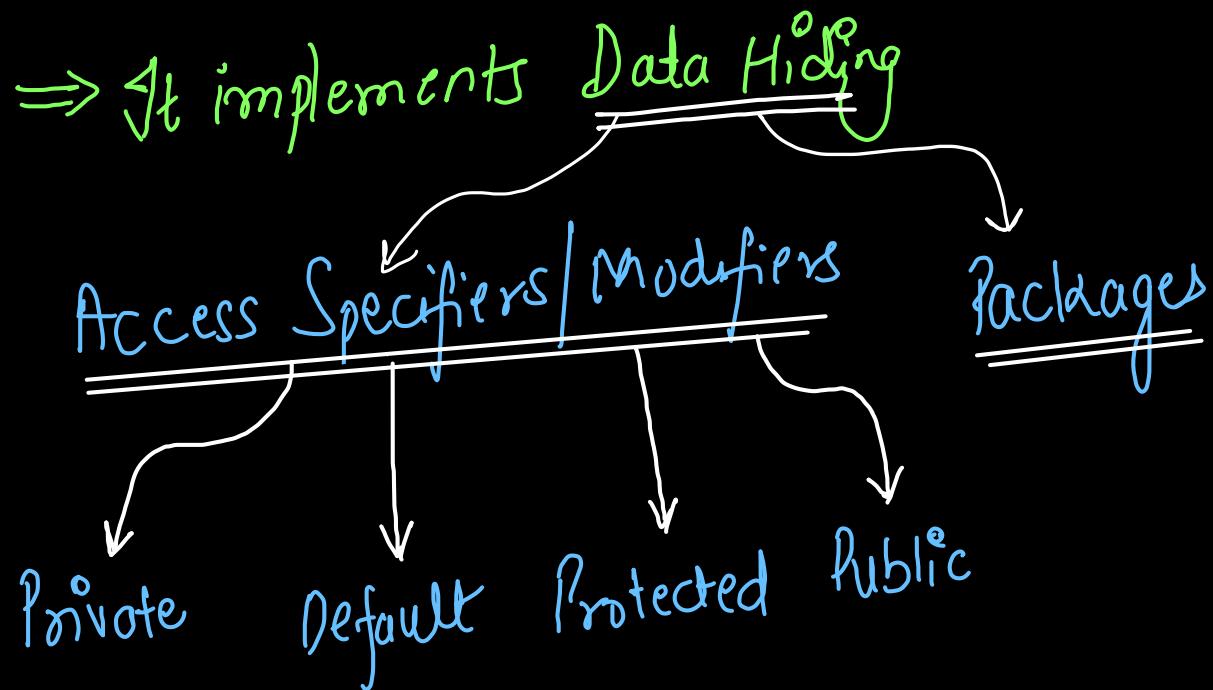
Object
Memory Allocation
{ Reference Variable → Stack }
{ Actual Instance → Heap }

Avengers Loki 120 Action 4.0



Q) What is ENCAPSULATION? Explain it with the help of real world example!

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



```
public static void main(String[] args) {  
    int[] durations = {120, 150, 140, 200, 130};  
  
    String[] name = {"Avengers Loki",  
                    "Avengers Ultron",  
                    "Avengers Infinity War",  
                    "Avengers Endgame",  
                    "Thor Love and Thunder"};  
  
    double[] ratings = {4.0, 3.0, 4.9, 5.0, 2.0};  
  
    String[] genre = {"Action", "Comedy", "Thriller", "Scifi", "Romance"};  
  
    System.out.println(name[4] + " " +  
                       durations[4] + " " +  
                       genre[4] + " " +  
                       ratings[4]);  
}
```

Code Without Encapsulation (Without Classes & Objects)

Finished in 136 ms

Thor Love and Thunder 130 Romance 2.0

```
class Movie{  
    private int duration;  
    private String name;  
    private double rating;  
    private String genre;
```

→ Data Members (Properties) ⇒ Private

```
public void setDuration(int newDuration){ duration = newDuration; }  
public void setName(String newName){ name = newName; }  
public void setRating(double newRating){ rating = newRating; }  
public void setGenre(String newGenre){ genre = newGenre; }
```

} *setters*

```
public int getDuration(){ return duration; }  
public String getName(){ return name; }  
public double getRating(){ return rating; }  
public String getGenre(){ return genre; }
```

} *getters*

Member Functions
(Behavior)
(Manipulators)

↓
Public

```
Movie avengers1 = new Movie();  
avengers1.setDuration(120);  
avengers1.setName("Avengers Loki");  
avengers1.setRating(4.0);  
avengers1.setGenre("Action");  
  
System.out.println(avengers1.getName() + " "  
    + avengers1.getDuration() + " " +  
    avengers1.getGenre() + " " +  
    avengers1.getRating());
```

Q (A) What are **CONSTRUCTORS** in Java? Explain with a coding example.

- special function → same name as the classname
- provide default values to the properties.
- called as soon as object is created
- for an object, constructor is only called once
- no "explicit" return type (But there is an implicit return type)

- It is a **special function** which gets called as soon as the object is allocated **heap memory**.
- It has **same name** as **classname**.
- It has **no explicit return type**.
- If has no explicit return type of constructor is "**this**" (reference of the class current object)
- Constructor cannot be **static/abstract/final**.
However, it can be **private/protected/default**.

```
class Movie{  
    public Movie(){  
        duration = 100;  
        name = "Untitled";  
        rating = 0.0;  
        genre = "Unclassified";  
    }  
  
    private int duration;  
    private String name;  
    private double rating;  
    private String genre;  
  
    public void setDuration(int newDuration){ duration = newDuration; }  
    public void setName(String newName){ name = newName; }  
    public void setRating(double newRating){ rating = newRating; }  
    public void setGenre(String newGenre){ genre = newGenre; }  
  
    public int getDuration(){ return duration; }  
    public String getName(){ return name; }  
    public double getRating(){ return rating; }  
    public String getGenre(){ return genre; }  
}
```

object(this)
Movie a1 = new Movie();
Reference
Constructor call

Q(B) What are the types of constructors in java?
Write implementation of all of them.

Types of Constructors

- Default Implicit Constructor → no parameter, no body
- Default Explicit Constructor → no parameters
- Parameterized Constructor → 1 or more than 1 parameters
- Copy constructor → parameter of reference of same class' object

(A) Default Implicit Constructor

```
public Movie(){}
```

(B) Default Explicit Constructor

```
public Movie(){  
    duration = 100;  
    name = "Untitled";  
    rating = 0.0;  
    genre = "Unclassified";  
}
```

(C) Parameterized Constructor

```
public Movie(int duration, String name,  
            double rating, String genre){  
  
    setDuration(duration);  
    setName(name);  
    setRating(rating);  
    setGenre(genre);  
}
```

(D) Copy Constructor

```
public Movie(Movie other){  
    setDuration(other.getDuration());  
    setName(other.getName());  
    setRating(other.getRating());  
    setGenre(other.getGenre());  
}
```

Q) What is constructor overloading? What are the rules for overloading of two methods/constructors?

* → Function name should be same

- Two constructors are said to be overloaded if atleast one condition is satisfied :→

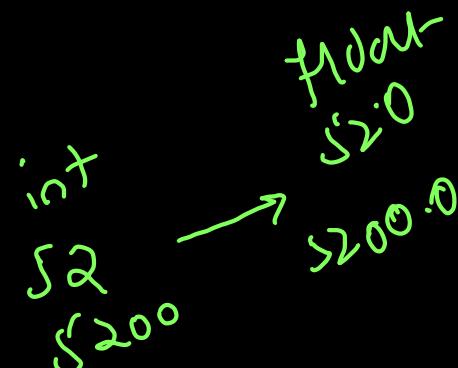
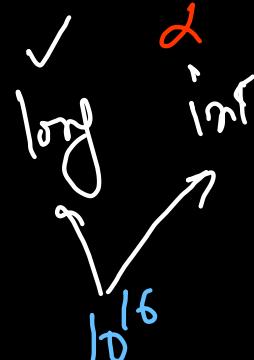
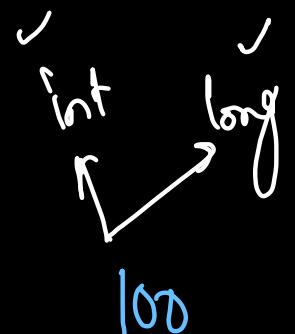
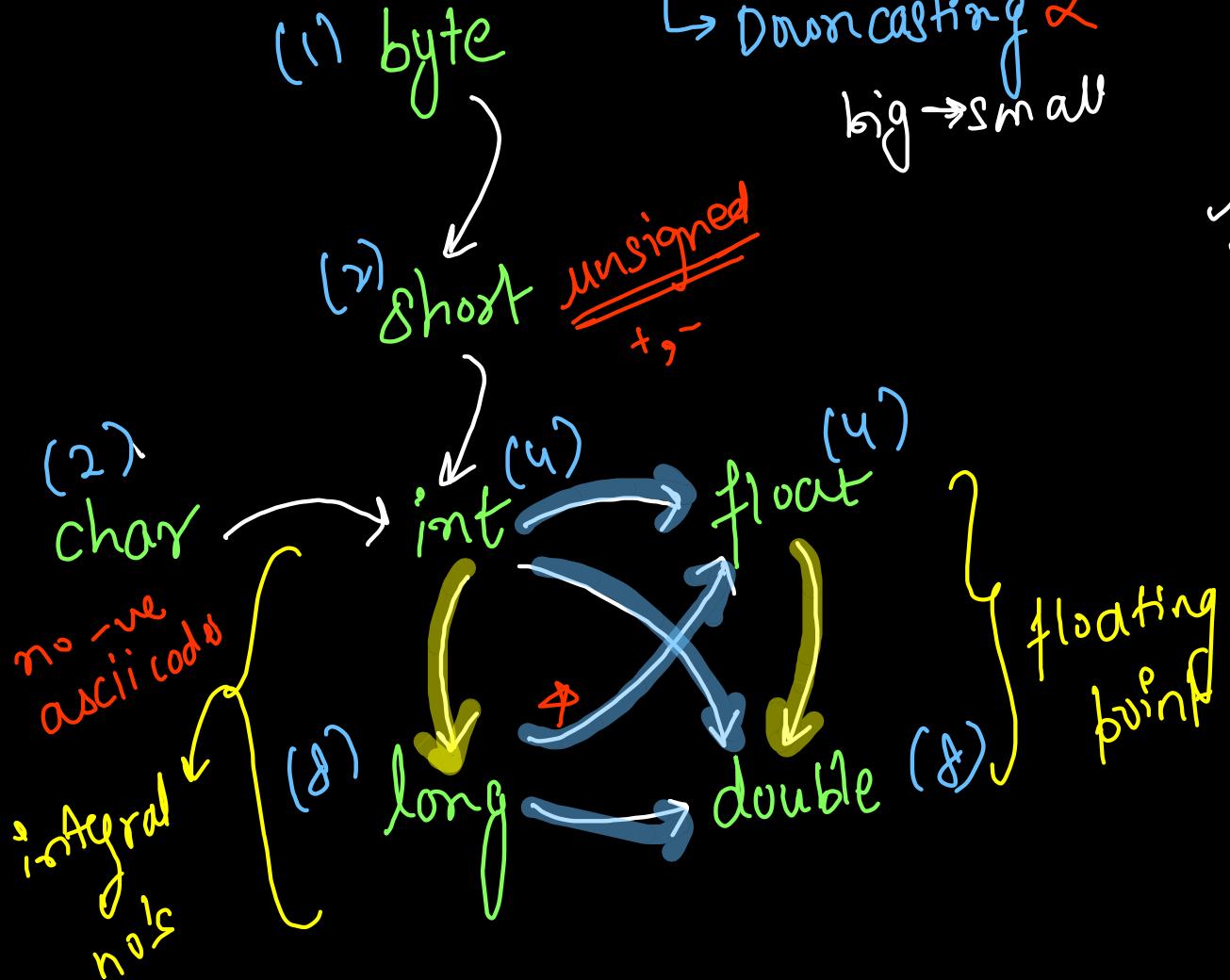
→ Number of arguments are different

→ Types of arguments are different

→ Order of arguments are different

* → Change in return type "does not" make functions/methods overloaded.

Type Promotion in Java \Rightarrow to resolve ambiguity and match function call to a particular function definition



To put it another way, the JLS distinguishes between a loss of ***magnitude*** and a loss of ***precision***.

`int` to `byte` for example is a (potential) loss of magnitude because you can't store 500 in a `byte`.

`long` to `float` is a potential loss of precision but not magnitude because the value range for floats is larger than that for longs.

So the rule is:

- **Loss of magnitude:** explicit cast required;
- **Loss of precision:** no cast required.

```
class Movie {  
    int duration;  
    String name, genre;  
    double rating;  
  
    public Movie(int duration) {  
        this.duration = duration;  
    }  
}
```

```
class Driver {  
    Run | Debug  
    public static void main(String[] args) {  
        // NO EXACT MATCH FOUND: Movie(char)  
        // Char Type Promoted to Integer (Upcasting - IMPLICIT)  
  
        Movie avengers1 = new Movie(duration: 'A');  
        System.out.println(avengers1.duration);  
  
        // COMPILATION ERROR: Long Demoted to Integer  
        // (Downcasting - IMPLICITLY NOT POSSIBLE)  
        // Movie avengers2 = new Movie(180l);  
        // System.out.println(avengers2.duration);  
  
        // NO EXACT MATCH FOUND: Movie(long)  
        // Long Type Demoted to Integer (Downcasting - EXPLICIT)  
  
        Movie avengers2 = new Movie((int) 180l);  
        System.out.println(avengers2.duration);  
    }  
}
```

- architaggarwal@Archits-MacBook-Air Java 00PS % javac 00PS_Codes/5.TypePromotion.java
 - architaggarwal@Archits-MacBook-Air Java 00PS % java 00PS_Codes.Driver
- 65
180

Q) Give the corrected output for the following code out of the given options.

Code :-

```
public static void swap(Movie a1, Movie a2){  
    Movie a3 = a1;  
    a1 = a2;  
    a2 = a3;  
}
```

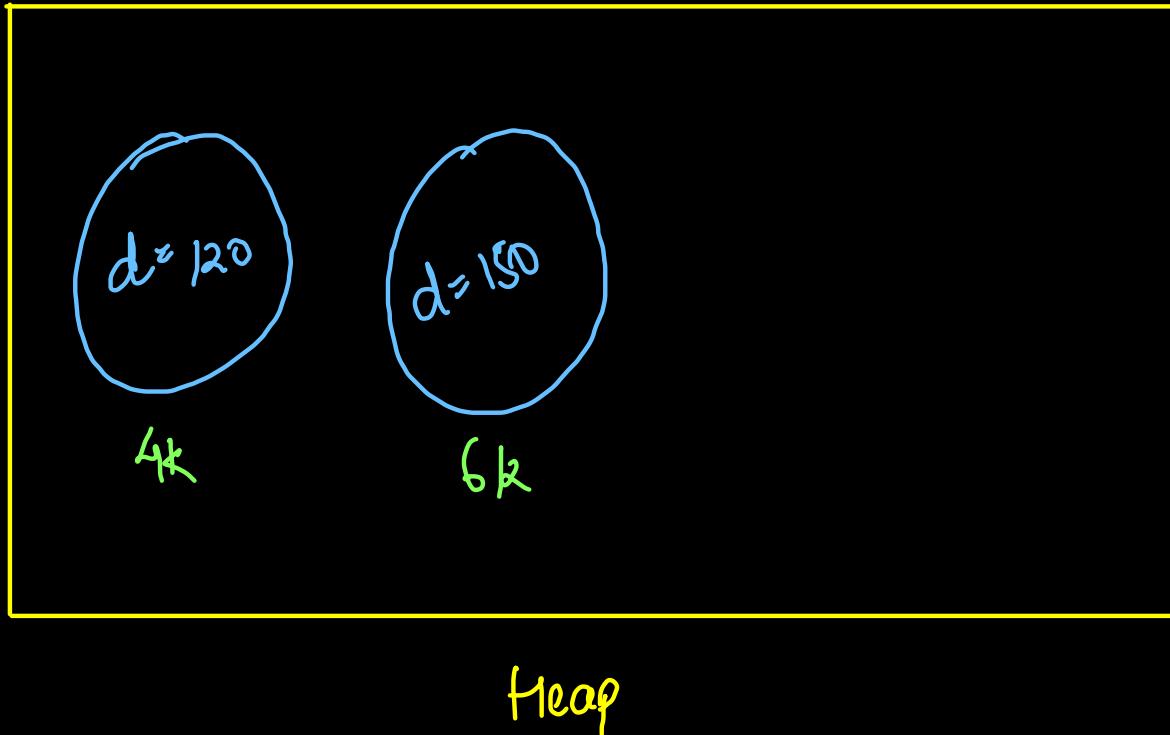
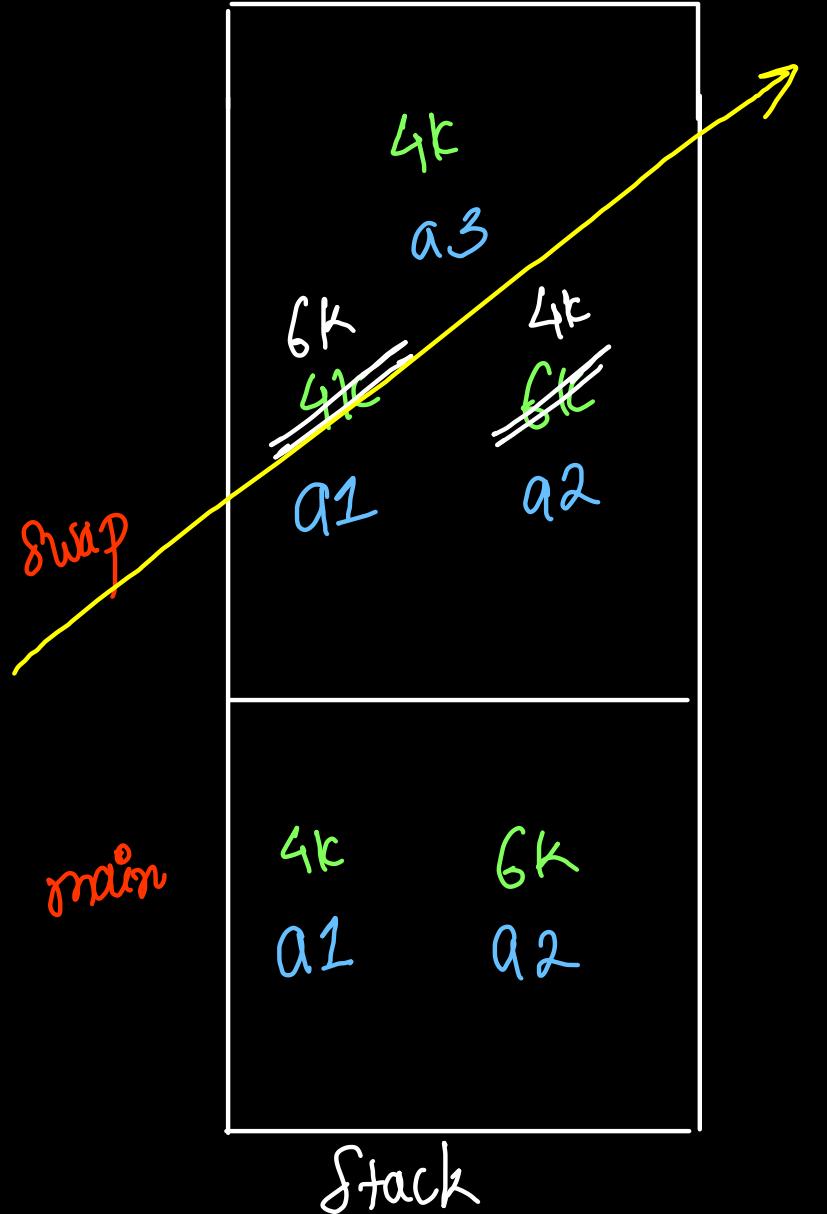
```
Movie a1 = new Movie();  
a1.setDuration(120);  
System.out.println(a1.getDuration());  
  
Movie a2 = new Movie();  
a2.setDuration(150);  
System.out.println(a2.getDuration());  
  
swap(a1, a2);  
  
System.out.println(a1.getDuration());  
System.out.println(a2.getDuration());
```

Options :-

- (a) 120, 150, 150, 120
- (b) 120, 150, 120, 150
- (c) 120, 150, 120, 120
- (d) 120, 150, 150, 150

java is always
pass by value!
↓
stack changes
does persist
not \equiv

Swap Game - ①



Q) Give the corrected output for the following code out of
the given options.

Code :-

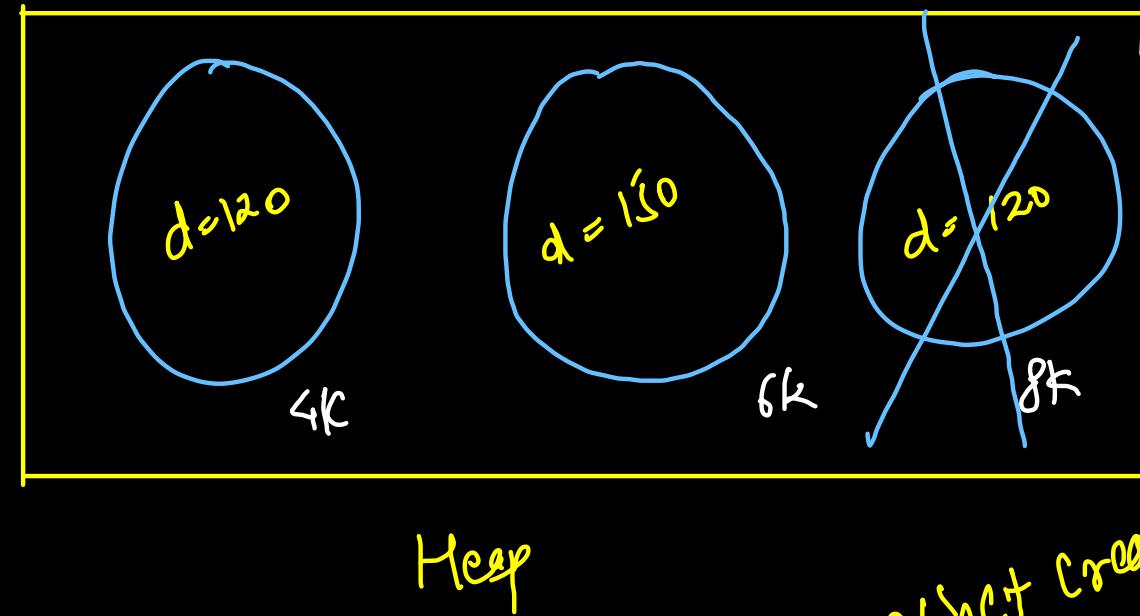
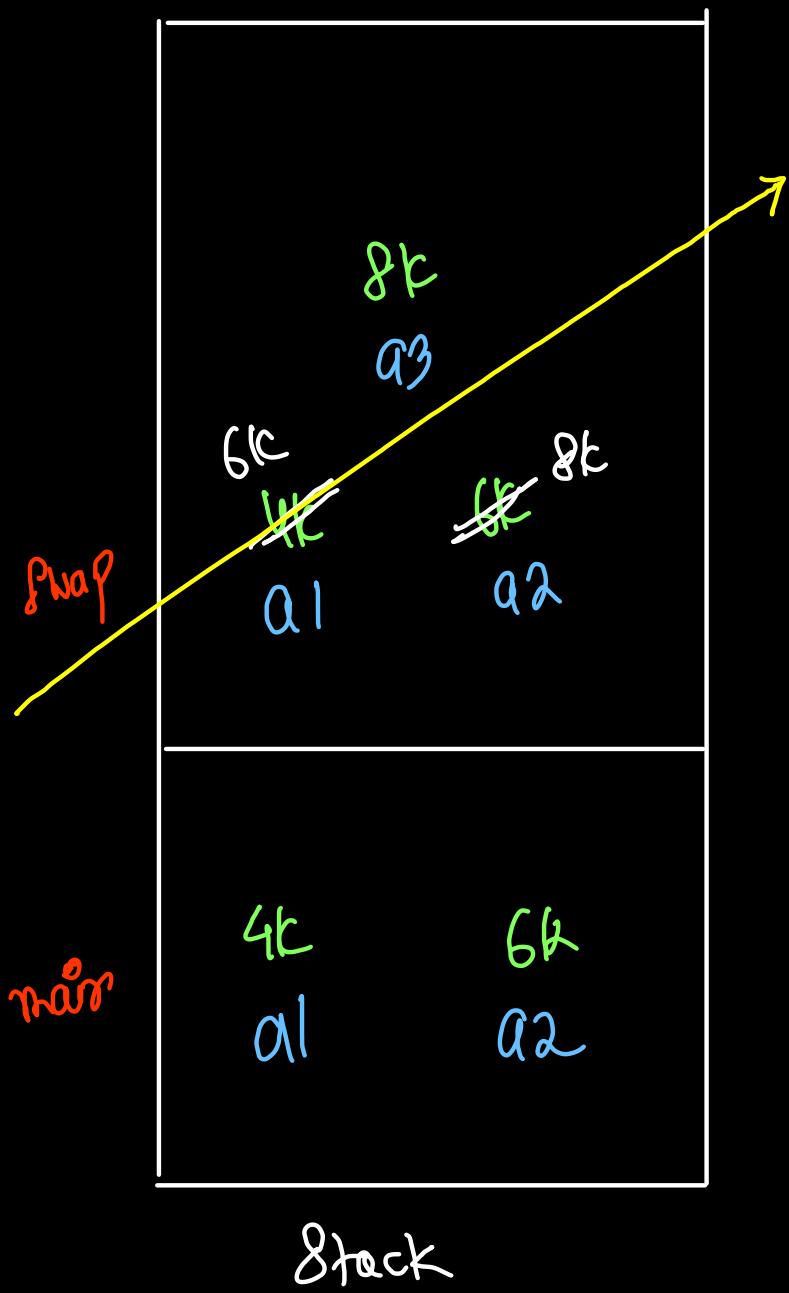
```
public static void swap(Movie a1, Movie a2){  
    Movie a3 = new Movie();  
    a3.setDuration(a1.getDuration());  
  
    a1 = a2;  
    a2 = a3;  
}
```

```
Movie a1 = new Movie();  
a1.setDuration(120);  
System.out.println(a1.getDuration());  
  
Movie a2 = new Movie();  
a2.setDuration(150);  
System.out.println(a2.getDuration());  
  
swap(a1, a2);  
  
System.out.println(a1.getDuration());  
System.out.println(a2.getDuration());
```

Options :-

- (a) 120, 150, 150, 120
- (b) 120, 150, 120, 150
- (c) 120, 150, 120, 120
- (d) 120, 150, 150, 150

Swap Game - 2



Object creation
↳ dump
↳ garbage collection

Q) Give the corrected output for the following code out of
the given options.

Code →

```
public static void swap(Movie a1, Movie a2){  
    Movie a3 = a1;  
  
    a1.setDuration(a2.getDuration());  
    a2.setDuration(a3.getDuration());  
}
```

```
Movie a1 = new Movie();  
a1.setDuration(120);  
System.out.println(a1.getDuration());  
  
Movie a2 = new Movie();  
a2.setDuration(150);  
System.out.println(a2.getDuration());  
  
swap(a1, a2);  
  
System.out.println(a1.getDuration());  
System.out.println(a2.getDuration());
```

Options →

- (a) 120, 150, 150, 120
- (b) 120, 150, 120, 150
- (c) 120, 150, 120, 120
- (d) 120, 150, 150, 150

Swap Game → 3

Stack

$a_3 = 4k$

$a_1 = 4k$

$a_2 = 6k$

$a_1 = 4k$

$a_2 = 6k$

Swap

Heap

$d = 125$

4k

$d = 110$

6k

Heap changes will persist.

main

Q) Give the corrected output for the following code out of
the given options.

Code :-

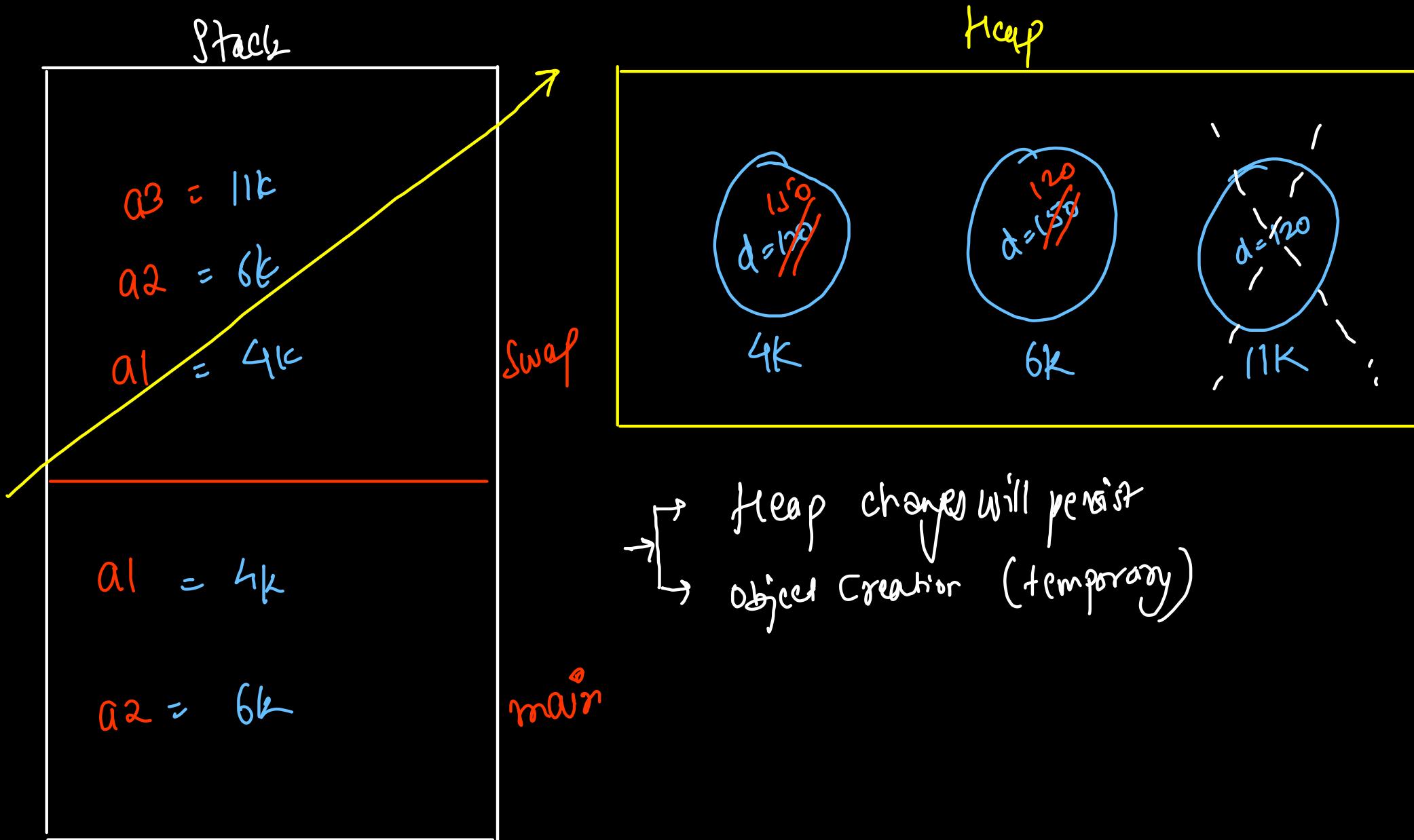
```
public static void swap(Movie a1, Movie a2){  
    Movie a3 = new Movie();  
    a3.setDuration(a1.getDuration());  
  
    a1.setDuration(a2.getDuration());  
    a2.setDuration(a3.getDuration());  
}
```

```
Movie a1 = new Movie();  
a1.setDuration(120);  
System.out.println(a1.getDuration());  
  
Movie a2 = new Movie();  
a2.setDuration(150);  
System.out.println(a2.getDuration());  
  
swap(a1, a2);  
  
System.out.println(a1.getDuration());  
System.out.println(a2.getDuration());
```

Options :-

- (a) 120, 150, 150, 120
- (b) 120, 150, 120, 150
- (c) 120, 150, 120, 120
- (d) 120, 150, 150, 150

Swap Games - 4



→ Heap changes will persist
Object Creation (temporary)

Q) What is the difference between shallow copy and deep copy?

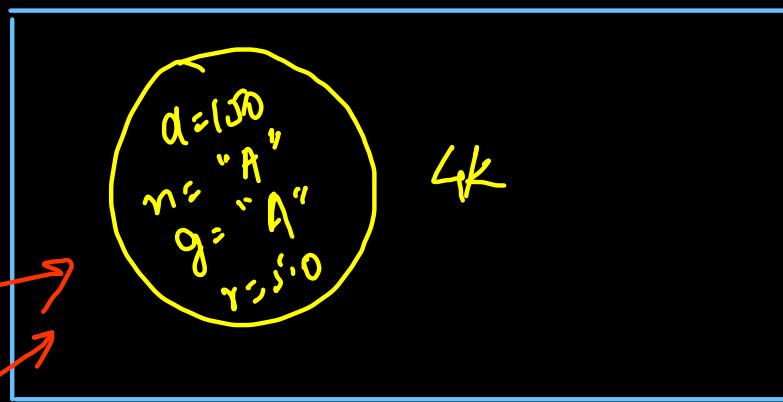
- Shallow Copy :-
- It stores the references of object to the original memory address.
 - Changes made in the cloned new object are reflected in the original old object and vice-versa.
 - It is just copy of reference variables, and not actual object creation is taking place
 - Shallow copy is faster than deep copy!

```
Movie avengers = new Movie();
System.out.println(avengers.duration + " " + avengers.genre
    + " " + avengers.name + " " + avengers.ratings);
```

```
Movie copy = avengers;
System.out.println(copy.duration + " " + copy.genre
    + " " + copy.name + " " + copy.ratings);
```

Copy = 4k

avengers = 4k



```
class Movie{
    int duration = 150;
    String genre = "Action", name = "Avengers";
    double ratings = 5.0;
}

public class Main {
    public static void main(String[] args) {
        Movie avengers = new Movie();
        System.out.println(avengers.duration + " " + avengers.genre
                           + " " + avengers.name + " " + avengers.ratings);

        Movie copy = avengers;
        System.out.println(copy.duration + " " + copy.genre
                           + " " + copy.name + " " + copy.ratings);

        copy.duration = 180;
        System.out.println(avengers.duration + " " + avengers.genre
                           + " " + avengers.name + " " + avengers.ratings);
        System.out.println(copy.duration + " " + copy.genre
                           + " " + copy.name + " " + copy.ratings);
    }
}
```

- Deep Copy: →
- It creates a cloned object that is new data members (properties) & copies values in them.
 - Changes in cloned new object are not reflected in changes of original object and vice-versa.
 - It is actual copy of properties (stored in heap) and not just reference variables (in stack)
 - Deep copy is slower than shallow copy.

```

Movie(Movie other){
    duration = other.duration;
    genre = other.genre;
    name = other.name;
    ratings = other.ratings;
}

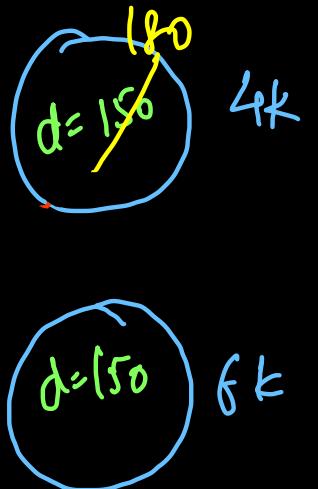
public class Main {
    public static void main(String[] args) {
        Movie avengers = new Movie();
        System.out.println(avengers.duration + " " + avengers.genre
                           + " " + avengers.name + " " + avengers.ratings);

        Movie deepCopy = new Movie(avengers);
        System.out.println(deepCopy.duration + " " + deepCopy.genre
                           + " " + deepCopy.name + " " + deepCopy.ratings);

        avengers.duration = 180;
        System.out.println(avengers.duration + " " + avengers.genre
                           + " " + avengers.name + " " + avengers.ratings);
        System.out.println(deepCopy.duration + " " + deepCopy.genre
                           + " " + deepCopy.name + " " + deepCopy.ratings);
    }
}

```

deepCopy = 6k
avengers = 4k



```

class Movie {
    int duration;
    String name, genre;
    double rating;
    ArrayList<String> languages;

    public Movie(String name, int duration, double rating, String genre) {
        this.name = name;
        this.duration = duration;
        this.genre = genre;
        this.rating = rating;
        this.languages = new ArrayList<>();
    }

    // PARTIAL DEEP COPY CONSTRUCTOR
    public Movie(Movie other) {
        this.name = other.name;
        this.duration = other.duration;
        this.genre = other.genre;
        this.rating = other.rating;
        this.languages = other.languages;
    }
}

```

```

● architaggarwal@Archits-MacBook-Air Java 0OPS % javac 0OPS_Codes/6.1.ShallowCopy.java
● architaggarwal@Archits-MacBook-Air Java 0OPS % java 0OPS_Codes.Driver
Avengers Endgame 180 4.5 SuperHero
[English]
[English, Tamil, Malayalam]
[English, Tamil, Malayalam]
Avengers Endgame 180 4.5 SuperHero
[English, Tamil, Malayalam]
[English, Tamil, Malayalam, Hindi, Telugu]
[English, Tamil, Malayalam, Hindi, Telugu]
○ architaggarwal@Archits-MacBook-Air Java 0OPS %

```

```

class Driver {
    Run | Debug
    public static void main(String[] args) {

        Movie avengers = new Movie(name: "Avengers Endgame", duration: 180,
        rating: 4.5, genre: "SuperHero");
        avengers.languages.add(e: "English");

        System.out.println(avengers.name + " " + avengers.duration
        + " " + avengers.rating + " " + avengers.genre);
        System.out.println(avengers.languages);

        // SHALLOW COPY
    }
}

```

```

Movie shallowCopy = avengers;

avengers.languages.add(e: "Tamil");
shallowCopy.languages.add(e: "Malayalam");

System.out.println(shallowCopy.languages);
System.out.println(avengers.languages);

// PARTIAL DEEP COPY

Movie partialDeep = new Movie(avengers);
System.out.println(partialDeep.name + " " + partialDeep.duration
+ " " + partialDeep.rating + " " + partialDeep.genre);
System.out.println(partialDeep.languages);

avengers.languages.add(e: "Hindi");
partialDeep.languages.add(e: "Telugu");

System.out.println(partialDeep.languages);
System.out.println(avengers.languages);

```

Plain Old Java Object (POJO) class Features

Data Members & Member Functions in class

- Properties should be private!
- Getters & setters should be public!
- Function  Method ~ Functions inside classes associated with an object are known as methods!
 - Always create a default explicit constructor
 - to implement "Data Hiding"

Q) What are differences between methods & functions in Java?

A method is a function or procedure in object oriented programming.

- Function inside a class which is called /invoked or associated with objects of that class is known as a method.
- Functions can be written inside as well as outside classes
ie. functions do not need a class to be created.
- Functions outside the class can only access local / primitive data like parameters whereas methods can access private data members of a class / object directly!

a) What is this keyword? What are it's applications?

THIS Keyword → self referential pointer (reference variable pointing to current object)

APPLICATIONS ⇒

- 1) → Refer data members from within the class
- 2) → Refer member functions from within the class
- 3) → Used in Constructor chaining (invoke own constructor)
 - chaining should be the first & single call only!
- 4) → Pass the current object as a parameter to a method/constructor
- 5) → Return the current object from a method/constructor
 - Return type of a constructor

```
class Movie {  
    int duration;  
    double ratings;  
    String name;  
    String genre;  
  
    public Movie() {}  
    // IMPLICIT RETURN TYPE: THIS  
    // (MOVIE REFERENCE TYPE)  
    public Movie(int newDuration, double newRatings,  
                String newName, String newGenre) {  
        duration = newDuration;  
        ratings = newRatings;  
        name = newName;  
        genre = newGenre;  
    }  
}
```



```
public static void main(String[] args) {  
    Movie avengers = new Movie(newDuration: 150, newRatings: 5.0, newName: "Avengers Endgame",  
    newGenre: "Thriller");  
    System.out.println(avengers.duration + " " + avengers.name  
    + " " + avengers.ratings + " " + avengers.genre);  
}
```

```
class Movie {  
    private int duration;  
  
    // Application 1: Constructor Return type  
    // Application 2: Invoking Member Function  
    public Movie(int duration) {  
        this.setDuration(duration);  
    }  
  
    public int getDuration() {  
        return duration;  
    }  
  
    // Application 3: Access Data Member Properties  
    public void setDuration(int duration) {  
        this.duration = duration;  
    }  
  
    // Application 4: Pass Current Object as Parameter  
    public void display() {  
        Driver.displayDurationOutside(this);  
    }  
  
    // Application 5: Return Current Object  
    public Movie join(Movie other) {  
        this.duration += other.duration;  
        return this;  
    }  
}
```

```
class Driver {  
    public static void displayDurationOutside(Movie obj) {  
        System.out.println("Movie Duration = " + obj.getDuration());  
    }  
  
    Run | Debug  
    public static void main(String[] args) {  
        Movie avengers1 = new Movie(duration: 120);  
        avengers1.display();  
  
        Movie avengers2 = new Movie(duration: 150);  
        avengers1.join(avengers2);  
        avengers1.display();  
    }  
}
```

- architaggarwal@Archits-MacBook-Air System Design % javac 00PS_Codes/08.ThisKeyword.java
- architaggarwal@Archits-MacBook-Air System Design % java 00PS_Codes.Driver
Movie Duration = 120
Movie Duration = 270
- architaggarwal@Archits-MacBook-Air System Design %

Q) Are there default parameterized constructors or functions in Java?
What do you understand by Constructor Chaining?

```
class Cuboid{  
    int length;  
    int breadth;  
    int height;  
  
    Cuboid(){  
        // this.length = 1;  
        // this.breadth = 1;  
        // this.height = 1;  
        this(1);  
    }  
  
    Cuboid(int side){  
        // this.length = side;  
        // this.breadth = side;  
        // this.height = side;  
        this(side, side, side);  
    }  
  
    Cuboid(int length, int breadth){  
        // this.length = length;  
        // this.breadth = breadth;  
        // this.height = 1;  
        this(length, breadth, 1);  
    }  
  
    Cuboid(int length, int breadth, int height){  
        this.length = length;  
        this.breadth = breadth;  
        this.height = height;  
    }  
}
```

Constructor Chaining

One constructor calling another constructor
using `this()` keyword!

```
public static void main(String[] args){  
    Cuboid obj1 = new Cuboid();  
    Cuboid obj2 = new Cuboid(5);  
    Cuboid obj3 = new Cuboid(5, 10);  
    Cuboid obj4 = new Cuboid(5, 10, 15);  
  
    System.out.println(obj1.length + " " + obj1.breadth + " " + obj1.height);  
    System.out.println(obj2.length + " " + obj2.breadth + " " + obj2.height);  
    System.out.println(obj3.length + " " + obj3.breadth + " " + obj3.height);  
    System.out.println(obj4.length + " " + obj4.breadth + " " + obj4.height);  
}
```

Finished in 109 ms

```
1 1 1  
5 5 5  
5 10 1  
5 10 15
```

⇒ Constructor chaining call
should be the first statement
in the calling constructor.

```
class Cuboid {  
    int length, breadth, height;  
  
    Cuboid() {  
        this.length = 1;  
        this.breadth = 1;  
        this.height = 1;  
    }  
  
    Cuboid(int side) {  
        this.length = side;  
        this.breadth = side;  
        this.height = side;  
    }  
  
    Cuboid(int length, int breadth) {  
        this.length = length;  
        this.breadth = breadth;  
        this.height = 1;  
    }  
  
    Cuboid(int length, int breadth, int height) {  
        this.length = length;  
        this.breadth = breadth;  
        this.height = height;  
    }  
}
```

```
public class Main {  
    Run | Debug  
    public static void main(String[] args) {  
        Cuboid obj1 = new Cuboid();  
        System.out.println(obj1.length + " " + obj1.breadth + " " + obj1.height);  
  
        Cuboid obj2 = new Cuboid(side: 5);  
        System.out.println(obj2.length + " " + obj2.breadth + " " + obj2.height);  
  
        Cuboid obj3 = new Cuboid(length: 10, breadth: 15);  
        System.out.println(obj3.length + " " + obj3.breadth + " " + obj3.height);  
  
        Cuboid obj4 = new Cuboid(length: 10, breadth: 15, height: 20);  
        System.out.println(obj4.length + " " + obj4.breadth + " " + obj4.height);  
    }  
}
```

Code without constructor chaining
Code redundancy!

```
class Cuboid {  
    int length, breadth, height;  
  
    Cuboid() {  
        this(side: 1);  
    }  
  
    Cuboid(int side) {  
        this(side, side, side);  
    }  
  
    Cuboid(int length, int breadth) {  
        this(length, breadth, height: 1);  
    }  
  
    Cuboid(int length, int breadth, int height) {  
        this.length = length;  
        this.breadth = breadth;  
        this.height = height;  
    }  
}
```

```
class Driver {  
    Run | Debug  
    public static void main(String[] args) {  
        Cuboid obj1 = new Cuboid();  
        Cuboid obj2 = new Cuboid(side: 5);  
        Cuboid obj3 = new Cuboid(length: 5, breadth: 10);  
        Cuboid obj4 = new Cuboid(length: 5, height: 10, breadth: 15);  
  
        System.out.println(obj1.length + " " + obj1.breadth + " " + obj1.height);  
        System.out.println(obj2.length + " " + obj2.breadth + " " + obj2.height);  
        System.out.println(obj3.length + " " + obj3.breadth + " " + obj3.height);  
        System.out.println(obj4.length + " " + obj4.breadth + " " + obj4.height);  
    }  
}
```

```
● architaggarwal@Archits-MacBook-Air 00PS_Codes % javac 09.ConstructorChaining.java  
● architaggarwal@Archits-MacBook-Air 00PS_Codes % java Driver  
1 1 1  
5 5 5  
5 1 10  
5 15 10
```

Q) Is Java a PURE object oriented programming language? What are wrapper classes? What is auto boxing and unboxing?

Every entity should be class or object.
and it should follow all
object oriented programming
principles.

int, boolean, char,
long, float, double,
short, byte

predefined
(primitive)
or userdefined

Answer : No

Solution \Rightarrow wrapper classes

Auto boxing & Unboxing

int \rightarrow Integer	float \rightarrow Float
boolean \rightarrow Boolean	double \rightarrow Double
char \rightarrow Character	short \rightarrow Short
long \rightarrow Long	byte \rightarrow Byte

```
public static void main(String[] args) {  
    // Primitive Stack Variable  
    int a = 5;  
    System.out.print(a + " ");  
  
    // Integer Wrapper Class  
    Integer aa = 10; // Autoboxing  
    System.out.print(aa + " ");  
  
    a = aa; // Unboxing  
    System.out.print(a + " ");  
  
    // Actual Object Creation, Getter and Setter  
    Integer b = new Integer(value: 20);  
    System.out.print(b.toString() + " ");  
  
    a = b.intValue();  
    System.out.print(a + " ");  
}
```

```
public static void functionsAndConst() {
    Integer a = 10;

    // Integer to String and String to Integer
    String b = a.toString();
    System.out.println(a + " Integer to String : " + b);

    String c = "256";
    Integer d = Integer.parseInt(c);
    System.out.println(c + " in Integer : " + d);

    // Various Number Conversions
    System.out.println(a + " Decimal to Binary : " + Integer.toBinaryString(a));
    System.out.println(a + " Decimal to Hexadecimal : " + Integer.toHexString(a));
    System.out.println(a + " Decimal to Octal : " + Integer.toOctalString(a));

    System.out.println("Integer MAXIMUM Range : " + Integer.MAX_VALUE);
    System.out.println("Integer MINIMUM Range : " + Integer.MIN_VALUE);

    // Various Other Functions
    System.out.println(a.compareTo(d));
    System.out.println(a.equals(d));
    System.out.println(Integer.max(a, d));
    System.out.println(Integer.min(a, d));
}
```

- ```
● architaggarwal@Archits-MacBook-Air 0OPS_Codes % java Driver
10 Integer to String : 10
256 in Integer : 256
10 Decimal to Binary : 1010
10 Decimal to Hexadecimal : a
10 Decimal to Octal : 12
Integer MAXIMUM Range : 2147483647
Integer MINIMUM Range : -2147483648
-1
false
256
10
```

```
class MyInteger {
 private int data;

 public MyInteger(int data) {
 this.data = data;
 }

 public int getData() {
 return data;
 }

 public void setData(int data) {
 this.data = data;
 }
}
```

```
MyInteger val1 = new MyInteger(data: 50);
System.out.println(val1);
System.out.println(val1.getData());
val1.setData(data: 100);
System.out.println(val1.getData());
```

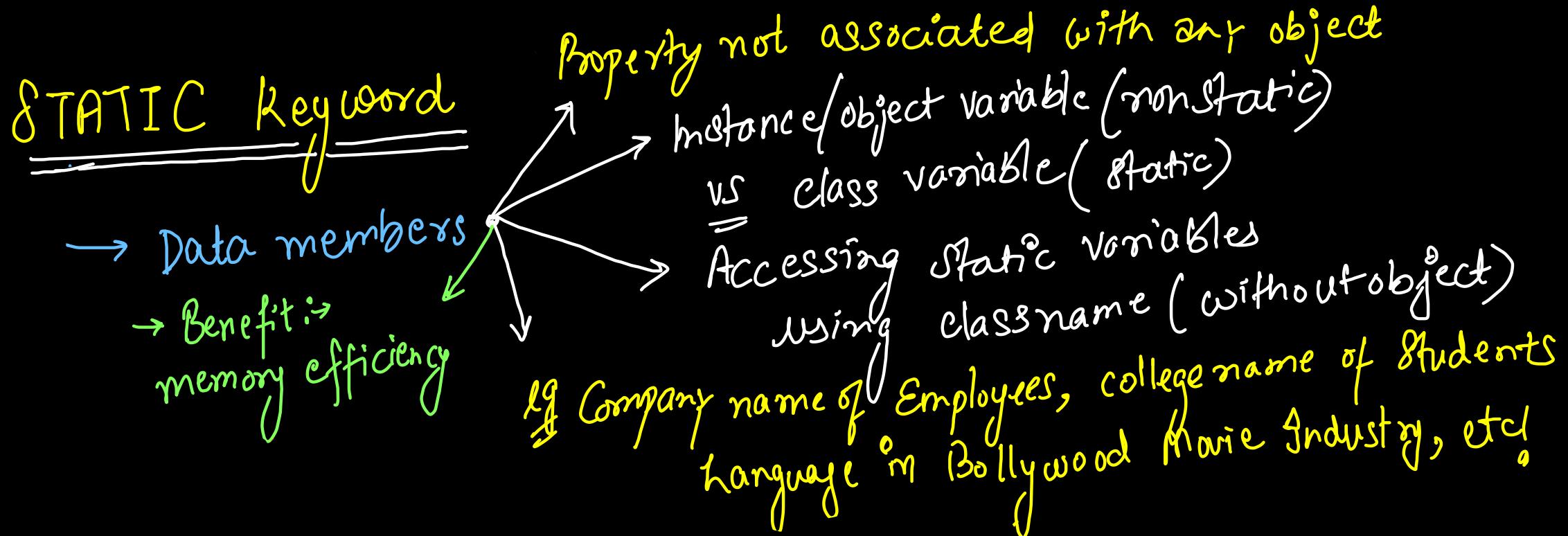
```

MyInteger@ea30797
50
100
```

```
Integer val = new Integer(value: 5);
Integer val2 = 5; // autoboxing: Integer -> int
System.out.println(val2);

int val3 = val2; // Unboxing: int -> Integer
System.out.println(val3);
```

Q) What do you mean by static keyword? What are static variables or class variables in Java?



```

class Movie {
 // Non Static or Instance Variables
 int duration;
 String name;
 double rating;

 // Static or Class Variable
 static String language = "English";

 public Movie(String name, int duration, double rating) {
 this.name = name;
 this.duration = duration;
 this.rating = rating;
 }

}

```

```

public static void main(String[] args) {
 Movie a1 = new Movie(name: "Avengers Infinity War", duration: 130, rating: 4.8);
 Movie a2 = new Movie(name: "Avengers End Game", duration: 200, rating: 4.9);
 Movie a3 = new Movie(name: "Avengers Secret Wars", duration: 170, rating: 4.6);
 Movie a4 = new Movie(name: "Avengers Kang Dynasty", duration: 220, rating: 5.0);

 // Non Static Variables
 System.out.println(a1.name + " " + a1.duration + " " + a1.rating);
 System.out.println(a2.name + " " + a2.duration + " " + a2.rating);
 System.out.println(a3.name + " " + a3.duration + " " + a3.rating);
 System.out.println(a4.name + " " + a4.duration + " " + a4.rating);

 // Static Variable or Class Variable
 System.out.print(a1.language + " ");
 System.out.print(a2.language + " ");
 System.out.print(a3.language + " ");
 System.out.print(a4.language + " ");

 // Accessing Using Class Name
 // (Static Properties Even Exist Without Single Object)
 System.out.print(Movie.language + " ");
}

}

```

- architaggarwal@Archits-MacBook-Air 0OPS\_Codes % javac 11.StaticKeyword.java
- architaggarwal@Archits-MacBook-Air 0OPS\_Codes % java Driver
   
Avengers Infinity War 130 4.8
   
Avengers End Game 200 4.9
   
Avengers Secret Wars 170 4.6
   
Avengers Kang Dynasty 220 5.0
   
English English English English English %

Interview  
Ques

Q) Why are static variables considered evil?

- It represents global scope (accessible using class name & from all objects of class)
- lifetime of variable is very long (entire program)
- less Object-Oriented way and reduce reusability
  - class loading init
  - do not require object creation
  - not object specific
- Not thread safe and difficult to serialize
  - (multiple threads will see same instance/copy of static variables which can lead to inconsistency/race condition during concurrent/parallel access).

```
public static void staticEvil() {
 // 1. Program LifeTime
 // 2. Global Scope
 // 3. Less Object Oriented (Do Not Require Object Creation)
 System.out.println(Movie.language);

 Movie a1 = new Movie(name: "Avengers Infinity War", duration: 130, rating: 4.8);
 Movie a2 = new Movie(name: "Avengers End Game", duration: 200, rating: 4.9);
 Movie a3 = new Movie(name: "Avengers Secret Wars", duration: 170, rating: 4.6);
 Movie a4 = new Movie(name: "Avengers Kang Dynasty", duration: 220, rating: 5.0);

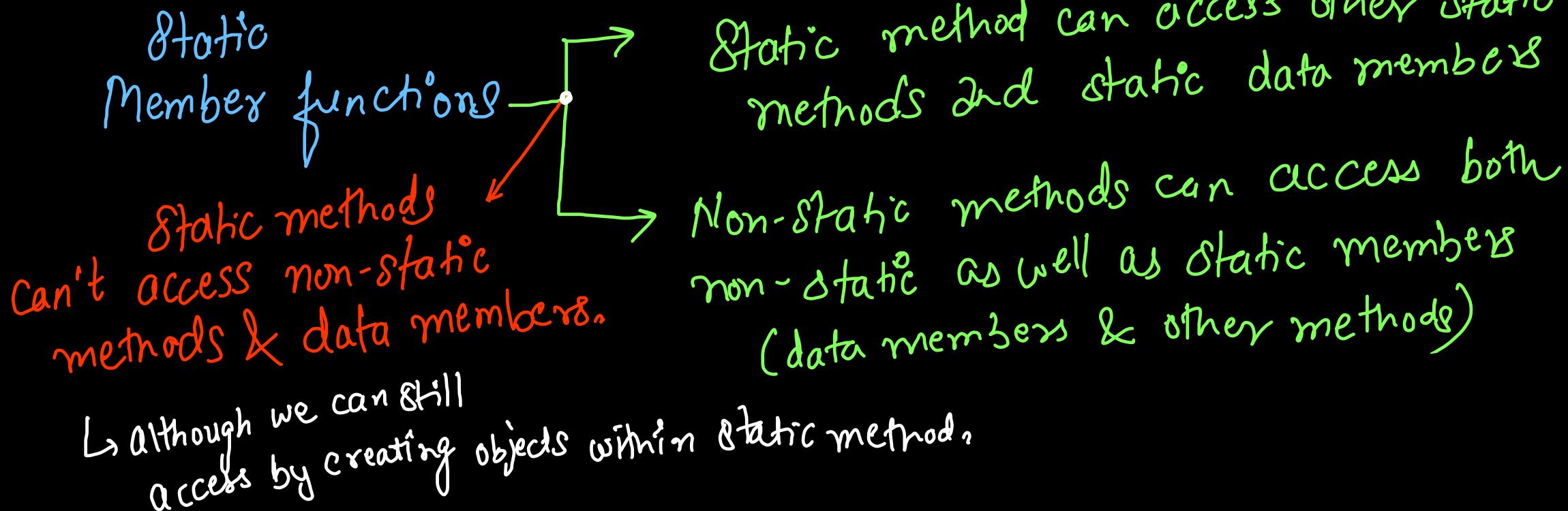
 System.out.println(a1.language + " " + a2.language + " " + a3.language + " " + a4.language);

 // 4. Same Copy Shared to Every Object
 a1.language = "Hindi";
 System.out.println(a1.language + " " + a2.language + " " + a3.language + " " + a4.language);

 // 5. Not Thread Safe and Non-Serializable
}
```

- architaggarwal@Archits-MacBook-Air 00PS\_Codes % javac 11.StaticKeyword.java
- architaggarwal@Archits-MacBook-Air 00PS\_Codes % java Driver  
English  
English English English English  
Hindi Hindi Hindi Hindi

Q) What are static member functions in Java? What are rules for accessing static or non-static data?



## Movie Class

```
public void nonStaticFun() {
 // Access Non Static Members (Properties & Other Functions) : Allowed
 System.out.println(" Name : " + this.name + " Duration : "
 + this.duration + " Rating : " + this.rating);

 // Access Static Member (Properties & Other Functions) : Allowed
 System.out.println("Language : " + Movie.language);
 Movie.staticFun();
}

public static void staticFun() {
 // This Keyword is Not There Inside Static Functions
 // System.out.println(this);

 // Access Non Static Members (Properties & Other Functions) : Not Allowed
 // System.out.println(" Name : " + this.name + " Duration : " + this.duration +
 // " Rating : " + this.rating);
 // nonStaticFun();

 // Access Static Member (Properties & Other Functions) : Allowed
 System.out.println("Language : " + Movie.language);

 // Access Non Static Members Using Object Creation
 Movie obj = new Movie(name: "Avengers", duration: 150, rating: 4.5);
 System.out.println(obj.name + " " + obj.duration + " " + obj.rating);
}
```

## Output

```
● architaggarwal@Archits-MacBook-Air 0OPS_Codes % javac 11.StaticKeyword.java
● architaggarwal@Archits-MacBook-Air 0OPS_Codes % java Driver
Language : English
Avengers 150 4.5
Name : Avengers Infinity War Duration : 130 Rating : 4.8
Language : English
Language : English
Avengers 150 4.5
```

## Driver Code

```
// No Object Required for Calling Static Functions
Movie.staticFun();

Movie a1 = new Movie(name: "Avengers Infinity War", duration: 130, rating: 4.8);
// Object Required to Call Non Static Member Function
a1.nonStaticFun();
```

## # Non static methods

```
public void setLanguage(String newLanguage){ language = newLanguage; }
public String getLanguage(){ return language; }
```

(1) Using Objects → Variables & Functrn access : Allowed

```
System.out.println(avengers1.getLanguage());
avengers1.setLanguage("English");
System.out.println(avengers1.getLanguage());
```

(2) Classname → Not Allowed  
(without object creation)

```
System.out.println(Movie.getLanguage());
Movie.setLanguage("Tamil");
System.out.println(Movie.getLanguage());
```

You require objects/  
instances in order  
to access non  
static members

## # STATIC METHODS

```
private static String language = "Hindi";
public static void setLanguage(String newLanguage){ language = newLanguage; }
public static String getLanguage(){ return language; }
```

1) Using Objects/Instances → Allowed ✓

```
System.out.println(avengers1.getLanguage());
avengers1.setLanguage("English");
System.out.println(avengers1.getLanguage());
```

2) Using Classname (without object creation) → Allowed ✓

```
System.out.println(Movie.getLanguage());
Movie.setLanguage("Tamil");
System.out.println(Movie.getLanguage());
```

You do not require  
object creation to access  
static members.

Interview Ques

Q) Why is main() method static in Java?

- JVM can call it without object creation
- It is the first method to undergo execution
- JVM can call main() method directly using classname

Note :→ It should also be always →

- ① public
- ② void return type
- ③ String[] args

"public static **void** main(String[] args)"

Default Exception Handler

exit  
int main  
↓ -1 or +1  
normal  
terminated  
abnormal  
terminated)

Interview Question Q) Do we have "this" inside a "static" method?

A) No, since static methods are not associated with objects and this keyword is associated with a particular instance/object, hence we do not have "this" access inside "static" method.

Interview Question Q) Can constructors be static?

A) No, since constructors implicitly return the current object reference ("this"), they cannot be static.

Q) What do you understand by static nested classes in Java?

static

Nested class → To create objects (instantiate)  
inner class objects without instantiating  
outer class

- Outer class can access inner class data members
  - Inner class can only access outer class' static members
  - Inner class object creation requires:- OuterClassName.innerClassName
- Note: If main() method & inner class are in the same outer class,  
we can omit outer class name

# Outer class (Movie)

```
class Movie {
 // Non Static or Instance Variables (Outer Class)
 int duration;
 String name;
 double rating;

 // Static or Class Variable (Outer Class)
 static String language = "English";

 public Movie(String name, int duration, double rating) {
 this.name = name;
 this.duration = duration;
 this.rating = rating;
 }

 public void outerNonStaticFun() { ... }
 public static void outerStaticFun() { ... }
 public static class Theater { ... }
}
```

```
public void outerNonStaticFun() {
 // Access Outer Non Static Members (Properties & Other Functions) : Allowed
 System.out.println(" Name : " + this.name + " Duration : "
 + this.duration + " Rating : " + this.rating);

 // Access Outer Static Member (Properties & Other Functions) : Allowed
 System.out.println("Language : " + Movie.language);|

 // Access Static Inner Class Static Members : Allowed
 System.out.println(Theater.screenType);

 // Access Static Inner Class Non Static Members
 // Without Inner Class Object: Not Allowed
 // System.out.println(this.chain + " " + this.noOfSeats);

 // With Inner Class Object: Allowed
 Theater inner = new Theater(chain: "PVR", noOfSeats: 100);
 System.out.println(inner.chain + " " + inner.noOfSeats);

 System.out.println(x: "-----");
}
```

```
public static void outerStaticFun() {
 // Access Outer Non Static Members (Properties & Other Functions) : Not Allowed
 // System.out.println(" Name : " + this.name + " Duration : " + this.duration +
 // " Rating : " + this.rating);
 // outerNonStaticFun();

 // Access Outer Static Member (Properties & Other Functions) : Allowed
 System.out.println("Language : " + Movie.language);

 // Access Static Inner Class Static Members : Allowed
 System.out.println(Theater.screenType);

 // Access Static Inner Class Non Static Members
 // Without Inner Class Object: Not Allowed
 // System.out.println(this.chain + " " + this.noOfSeats);

 // With Inner Class Object: Allowed
 Theater inner = new Theater(chain: "PVR", noOfSeats: 100);
 System.out.println(inner.chain + " " + inner.noOfSeats);

 System.out.println(x: "-----");
}
```

# Inner class (Theater)

```
public static class Theater {
 // Non Static or Instance Variables (Inner Class)
 String chain;
 int noOfSeats;

 // Static or Class Variable (Inner Class)
 static String screenType = "2D";

 public Theater(String chain, int noOfSeats) {
 this.chain = chain;
 this.noOfSeats = noOfSeats;
 }

 public void innerNonStaticFun() {
 // Access Outer Non Static Members : Not Allowed
 // System.out.println(" Name : " + this.name + " Duration : "
 // + this.duration + " Rating : " + this.rating);

 // Access Outer Static Member (Properties & Other Functions) : Allowed
 System.out.println("Language : " + Movie.language);

 // Access Static Inner Class Static Members : Allowed
 System.out.println(Theater.screenType);

 // Access Static Inner Class Non Static Members: Allowed
 System.out.println(this.chain + " " + this.noOfSeats);

 System.out.println(x: "-----");
 }
}
```

```
public static void innerStaticFun() {
 // Access Outer Non Static Members (Properties & Other Functions) : Not Allowed
 // System.out.println(" Name : " + this.name + " Duration : " + this.duration +
 // " Rating : " + this.rating);
 // outerNonStaticFun();

 // Access Outer Static Member (Properties & Other Functions) : Allowed
 System.out.println("Language : " + Movie.language);

 // Access Static Inner Class Static Members : Allowed
 System.out.println(Theater.screenType);

 // Access Static Inner Class Non Static Members
 // Without Inner Class Object: Not Allowed
 // System.out.println(this.chain + " " + this.noOfSeats);

 // With Inner Class Object: Allowed
 Theater inner = new Theater(chain: "PVR", noOfSeats: 100);
 System.out.println(inner.chain + " " + inner.noOfSeats);

 System.out.println(x: "-----");
}
```

# Output

## Driver Code

```
class Driver {
 Run | Debug
 public static void main(String[] args) {
 Movie.outerStaticFun();
 Movie.Theater.innerStaticFun();

 Movie outer = new Movie(name: "Avengers", duration: 180, rating: 4.5);
 outer.outerNonStaticFun();

 Movie.Theater inner = new Movie.Theater(chain: "PVR", noOfSeats: 100);
 inner.innerNonStaticFun();
 }
}
```

Language : English

2D

PVR 100

Language : English

2D

PVR 100

Name : Avengers Duration : 180 Rating : 4.5

Language : English

2D

PVR 100

Language : English

2D

PVR 100

Q) What do you understand by "static block" in Java?

→ static block

- ↳ used to initialize static variables
- ↳ executed before main() execution
- ↳ at time of class loading

Q) Can you run a java program only using static block, ie. without main function?

No, in newer Java versions, main() is mandatory to run a java program. If there is no entry point, java code will not run even if there is a static block.

```
class Movie {
 // Non Static or Instance Variables
 int duration;
 String name;
 double rating;

 // Static or Class Variable
 static String language;

 public Movie(String name, int duration, double rating) {
 System.out.println("Constructor - Object Creation");
 this.name = name;
 this.duration = duration;
 this.rating = rating;
 this.language = "Hindi";
 }

 // Static Block (Runs Before Main Method Execution: During Loading & Linking)
 static {
 System.out.println("Movie Static Block Executed");

 // Non Static Member Initialized
 language = "English";

 System.out.println(language + " OR " + Movie.language);
 }
}
```

```
class Driver {
 static int x;
 static {
 System.out.println(x: "Driver Static block executed");
 x = 100;
 System.out.println(x);
 }

 Run | Debug
 public static void main(String[] args) {
 System.out.println("Main Method Started");

 // main method must be there otherwise static block will also not run,
 // Although it will run after static block of its class

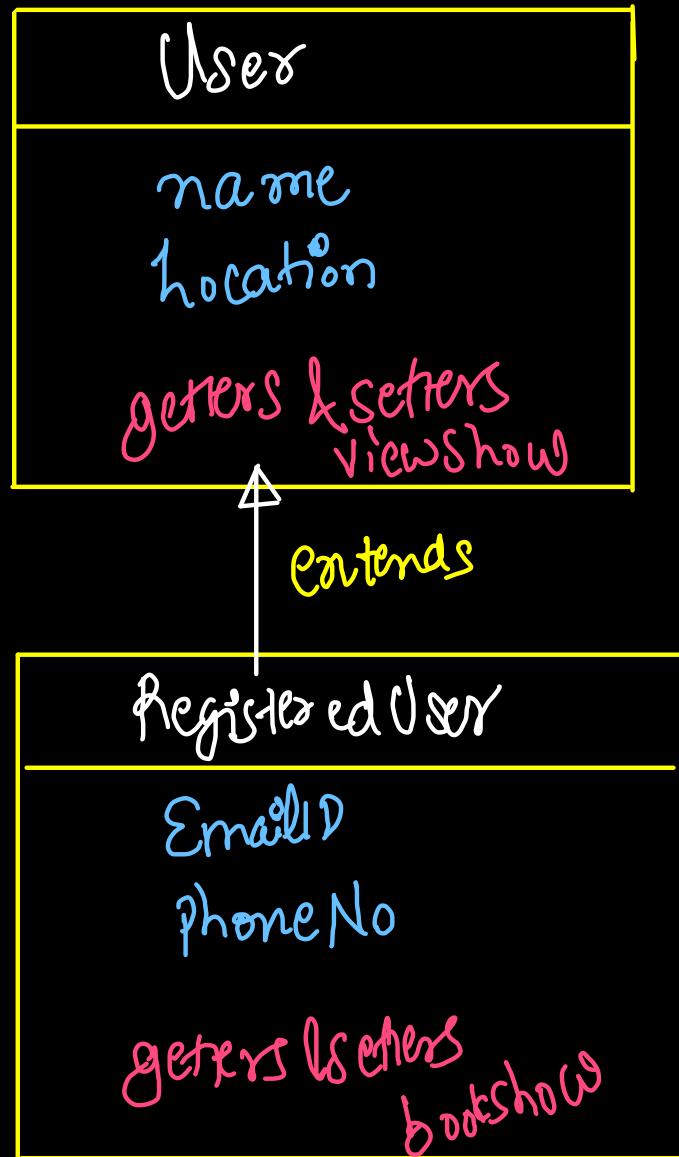
 System.out.println(x);
 System.out.println(Movie.language);

 Movie obj = new Movie(name: "Avengers", duration: 180, rating: 4.5);
 System.out.println(obj.language);
 }
}
```

- architaggarwal@Archits-MacBook-Air 00PS Codes % javac 11.3.StaticBlock.java
- architaggarwal@Archits-MacBook-Air 00PS Codes % java Driver  
Driver Static block executed  
100  
Main Method Started  
100  
Movie Static Block Executed  
English OR English  
English  
Constructor - Object Creation  
Hindi

Q) What do you mean by INHERITANCE in Java? What is super/  
parent class & child/sub class? Give some real life examples.

- Way in which two classes can be related with each other. (is A relationship)
- super class or parent class is being inherited by child class
- super class or parent class properties & behavior of parent class are  
acquired by child class directly or indirectly.
- Hence, it improves reusability, readability, extensibility  
and maintainability of code.
- Along with reusing parent properties, we can add new  
properties & methods in child class.
- Inheritance is implemented to achieve generalization, to  
implement run-time polymorphism



Parent class / Super class  
Non-Registered  
Non-Authenticated  
Anonymous

Child class / Sub class  
Registered User  
Authenticated User

```
class User {
 String name;
 String location;

 public User(String name, String location) {
 this.name = name;
 this.location = location;
 }

 public User() {
 this.name = "Anonymous";
 this.location = "India";
 }

 public void viewShow() {
 System.out.println("I can view listing shows on app");
 }

 class RegisteredUser extends User {
 String emailId;
 long phoneNo;

 public RegisteredUser() {
 this.emailId = "registeredUser@gmail.com";
 this.phoneNo = 9319117888L;
 }

 public RegisteredUser(String emailId, long phoneNo) {
 this.emailId = emailId;
 this.phoneNo = phoneNo;
 }

 public void bookShow() {
 System.out.println("I can book the shows on app");
 }
 }
}
```

```
public class Solution {
 Run | Debug
 public static void main(String[] args) {
 User obj1 = new User();
 obj1.name = "Archit Aggarwal";
 obj1.location = "Delhi";
 System.out.println(obj1.name + " " + obj1.location);
 obj1.viewShow();

 // Compilation Error
 // System.out.println(obj1.phoneNo);
 // System.out.println(obj1.emailId);
 // obj1.bookShow();

 RegisteredUser obj2 = new RegisteredUser();
 obj2.name = "Shahrukh Khan";
 obj2.location = "Mumbai";
 obj2.phoneNo = 9319117889L;
 obj2.emailId = "archit.aggarwal023@gmail.com";

 System.out.println(obj2.name + " " + obj2.location);
 System.out.println(obj2.phoneNo + " " + obj2.emailId);

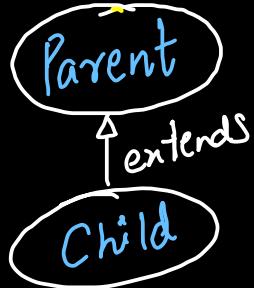
 obj2.viewShow();
 obj2.bookShow();
 }
}
```

- architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
- architaggarwal@Archits-MacBook-Air System Design % java Solution  
Archit Aggarwal Delhi  
I can view listing shows on app  
Shahrukh Khan Mumbai  
9319117889 archit.aggarwal023@gmail.com  
I can view listing shows on app  
I can book the shows on app

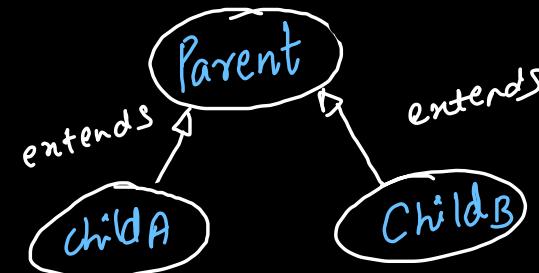
#  
Single  
level  
Inheritance

Q) What are the types of inheritance. Are all allowed in Java?  
Give some real world examples.

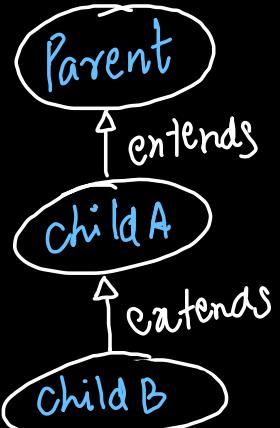
① Single level Inheritance



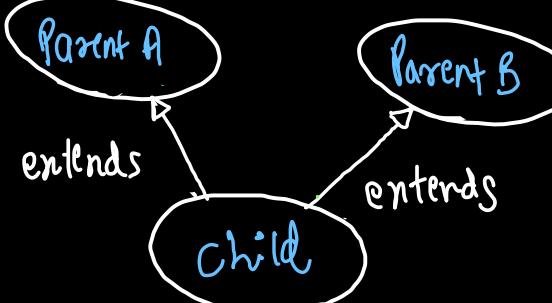
③ Hierarchical Inheritance



② Multilevel Inheritance



④ Multiple Inheritance



## # Multilevel Inheritance



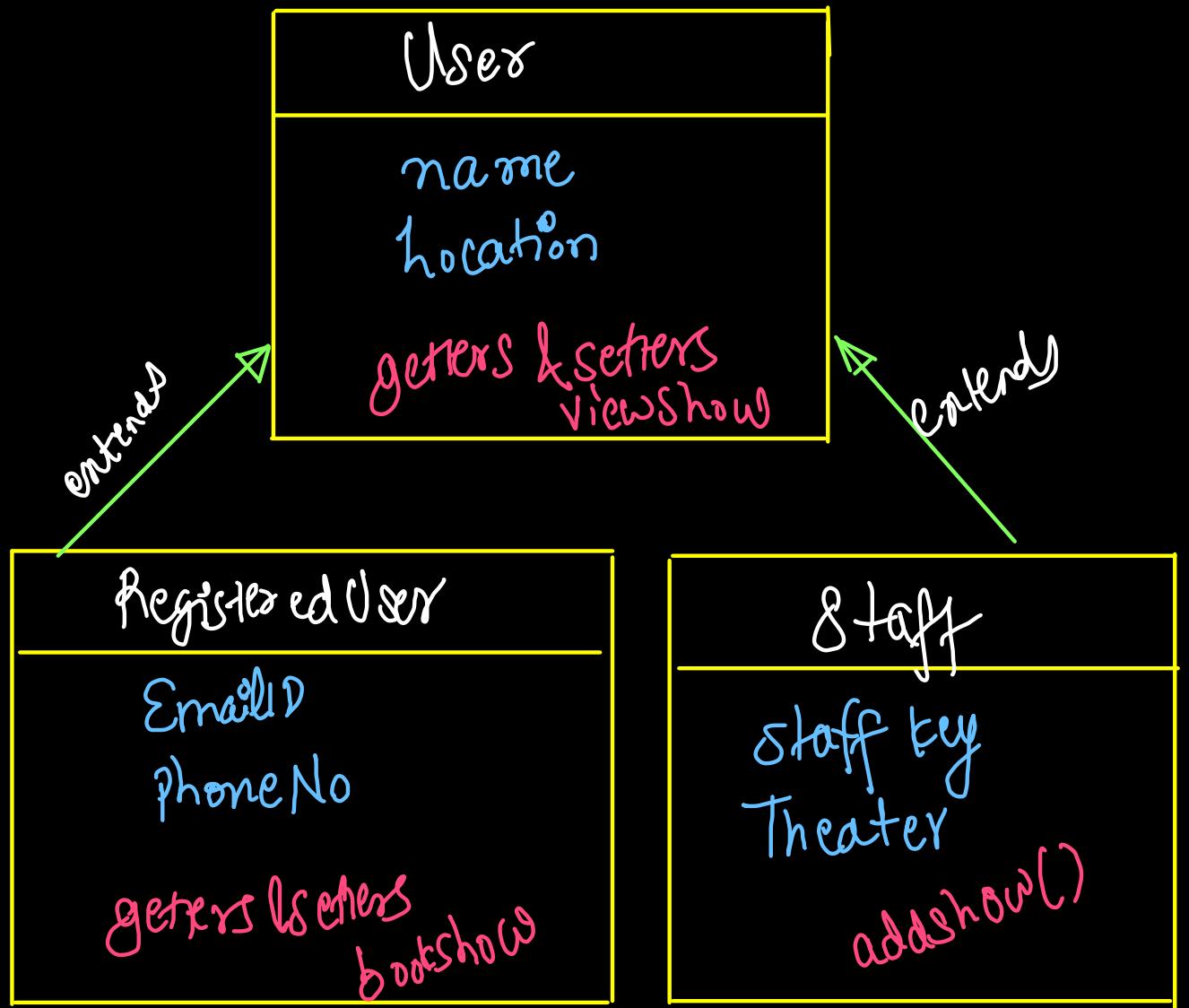
```
class SubscribedUser extends RegisteredUser {
 int validity;
 String subscriptionType;

 public SubscribedUser() {
 validity = 365;
 subscriptionType = "Mobile";
 }

 public void watchShow() {
 System.out.println("I have paid, hence I can watch the movies on OTT");
 }
}
```

● architaggarwal@Archits-MacBook-Air: System Design % java Solution  
registeredUser@gmail.com India Anonymous 9319117888 Mobile 365  
I can view listing shows on app  
I can book the shows on app  
I have paid, hence I can watch the movies on OTT

## # Hierarchical Inheritance



```
public static void main(String[] args) {
 User obj1 = new User();
 obj1.name = "Archit Aggarwal";
 obj1.location = "Delhi";
 System.out.println(obj1.name + " " + obj1.location);
 obj1.viewShow();

 RegisteredUser obj2 = new RegisteredUser();
 obj2.name = "Shahrukh Khan";
 obj2.location = "Mumbai";
 obj2.phoneNo = 9319117889l;
 obj2.emailId = "archit.aggarwal023@gmail.com";

 System.out.println(obj2.name + " " + obj2.location);
 System.out.println(obj2.phoneNo + " " + obj2.emailId);

 obj2.viewShow();
 obj2.bookShow();

 Staff obj3 = new Staff();

 System.out.println(obj3.name + " " + obj3.location);
 System.out.println(obj3.staffId + " " + obj3.theater);

 obj3.viewShow();
 obj3.addShow();

 // Compilation Error
 // System.out.println(obj3.phoneNo + " " + obj3.emailId);
 // obj3.bookShow();
 // System.out.println(obj2.staffId + " " + obj2.theater);
 // obj2.addShow();
}
```

```
class Staff extends User {
 int staffId = 707;
 String theater = "PVR Cinemas";

 public void addShow() {
 System.out.println("I Can Add Show in My Theater");
 }
}
```

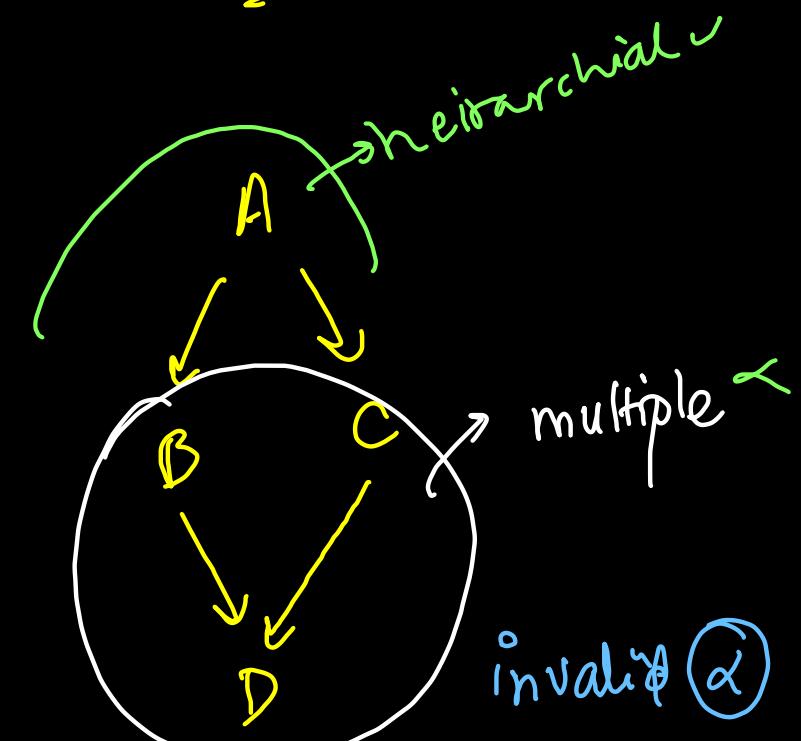
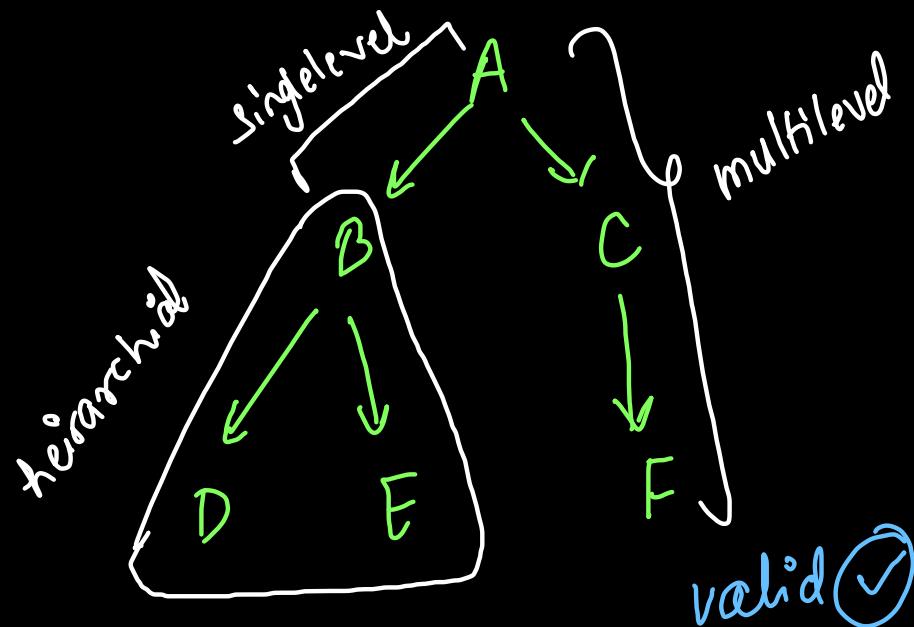
- architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
- architaggarwal@Archits-MacBook-Air System Design % java Solution

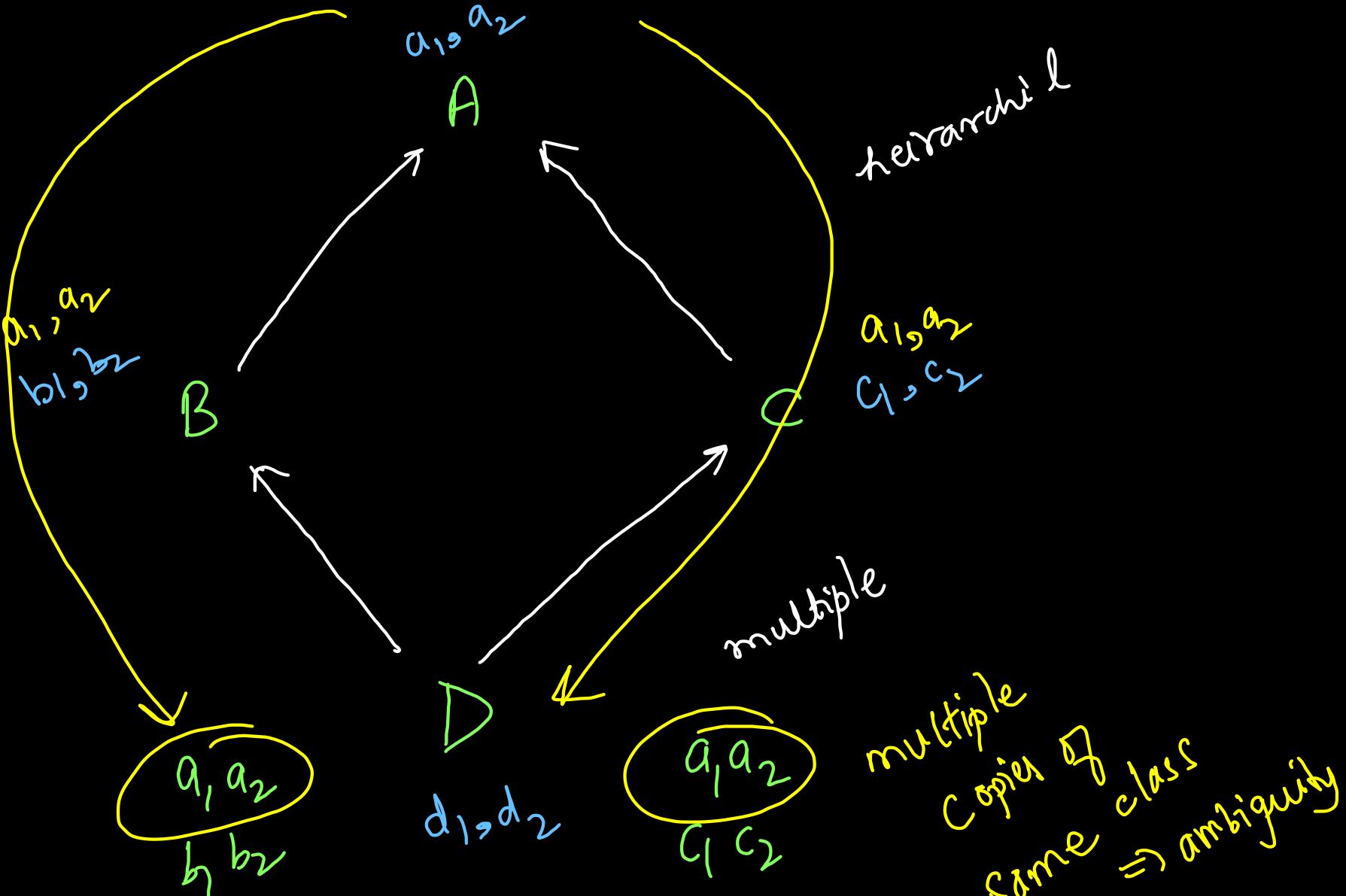
Archit Aggarwal Delhi  
I can view listing shows on app  
Shahrukh Khan Mumbai  
9319117889 archit.aggarwal023@gmail.com  
I can view listing shows on app  
I can book the shows on app  
Anonymous India  
707 PVR Cinemas  
I can view listing shows on app  
I Can Add Show in My Theater

```
// Multiple Inheritance is Not Possible in Java, We cannot extend 2 or more classes in same class
// Compilation Error
// class Admin extends RegisteredUser, Staff{
// }
```

Multiple inheritance & not allowed in Java

## ⑤ Hybrid Inheritance



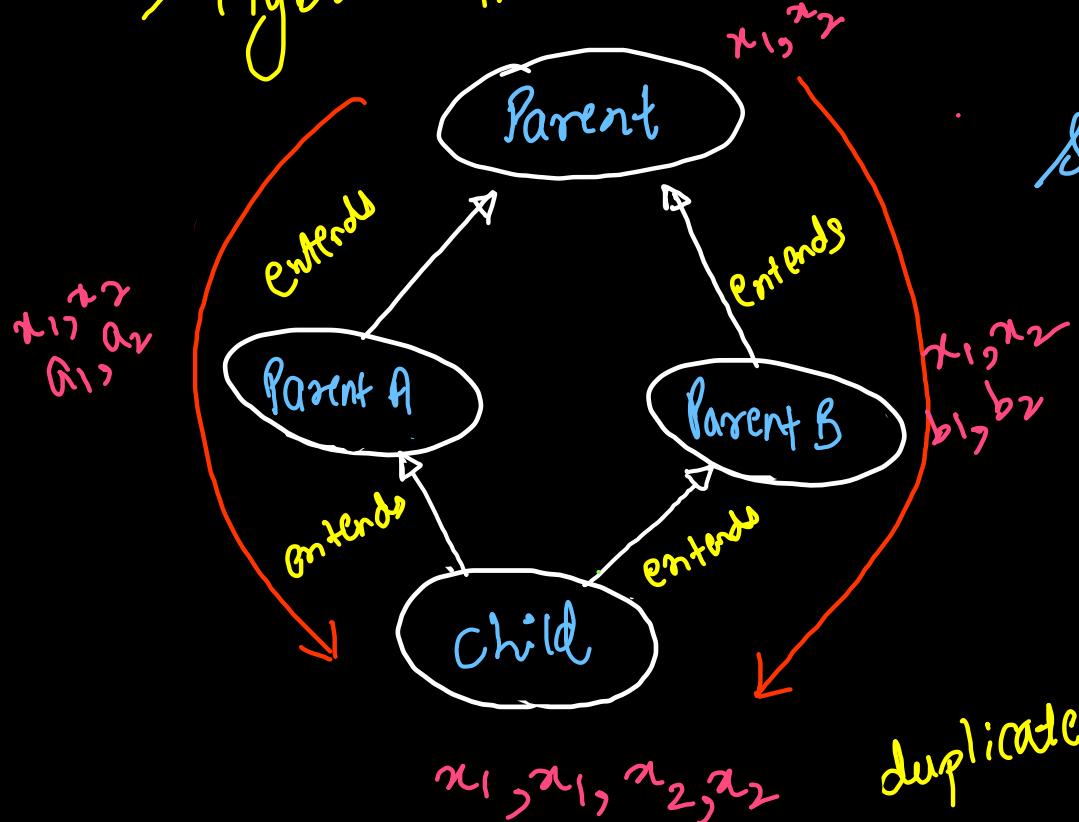


Interview Question →  
Why multiple inheritance using classes is not supported in Java?

Q) Why multiple inheritance using classes is not supported in Java?

A) Diamond Problem or Deadly Diamond of Death.

⇒ Hybrid Inheritance



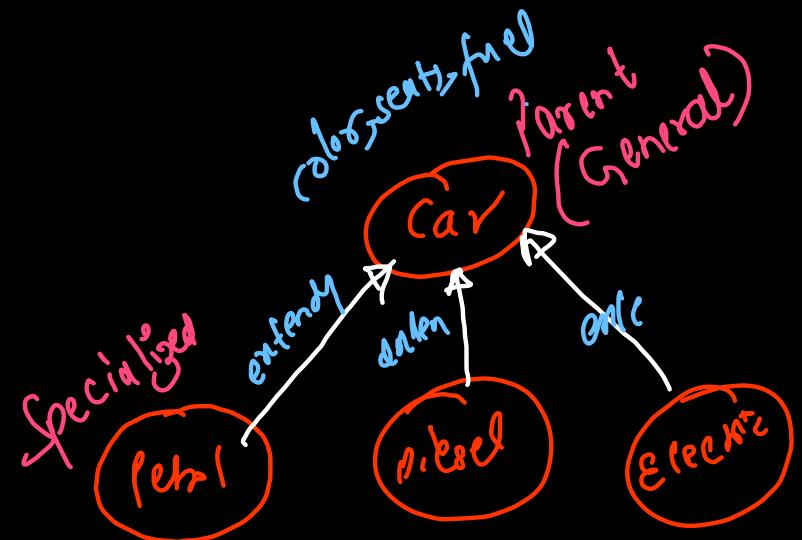
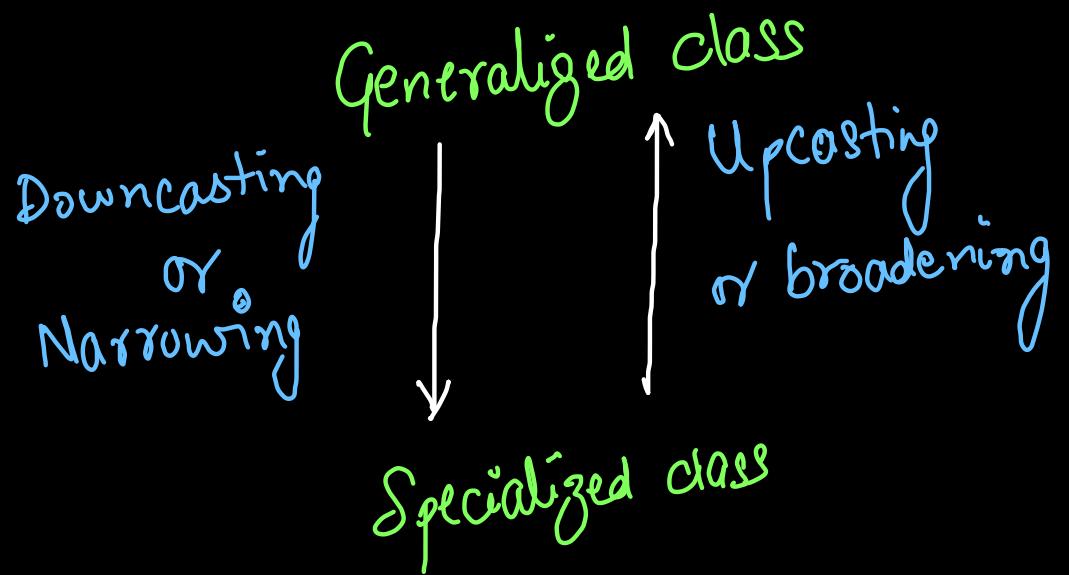
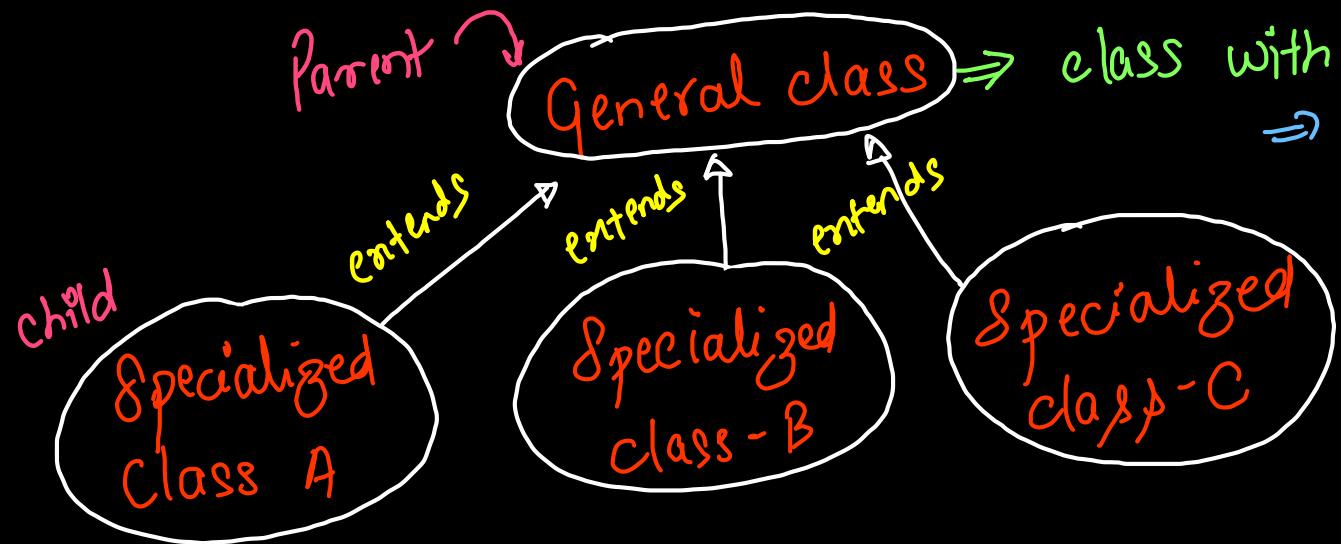
⇒ Ambiguity in parent properties or methods if name collision occurs.

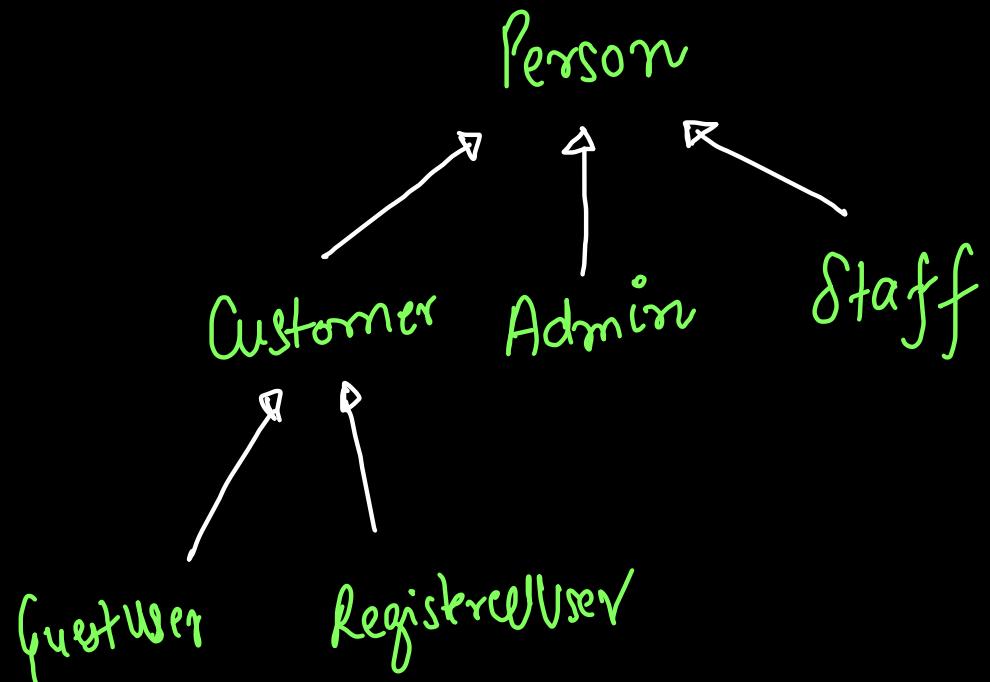
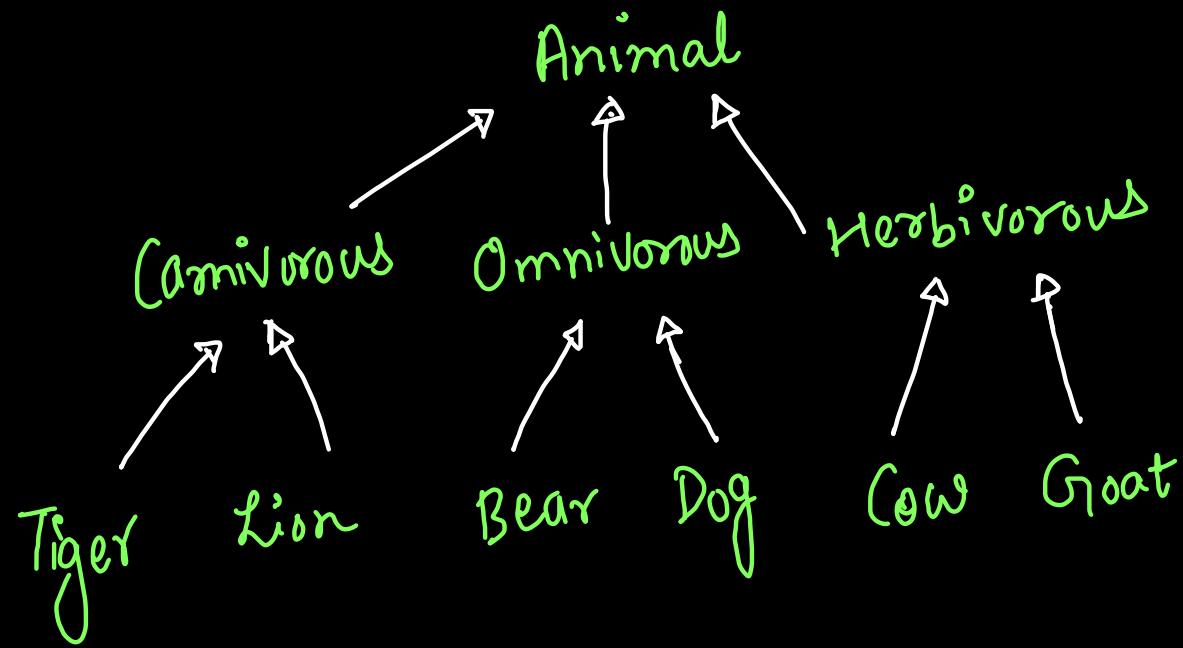
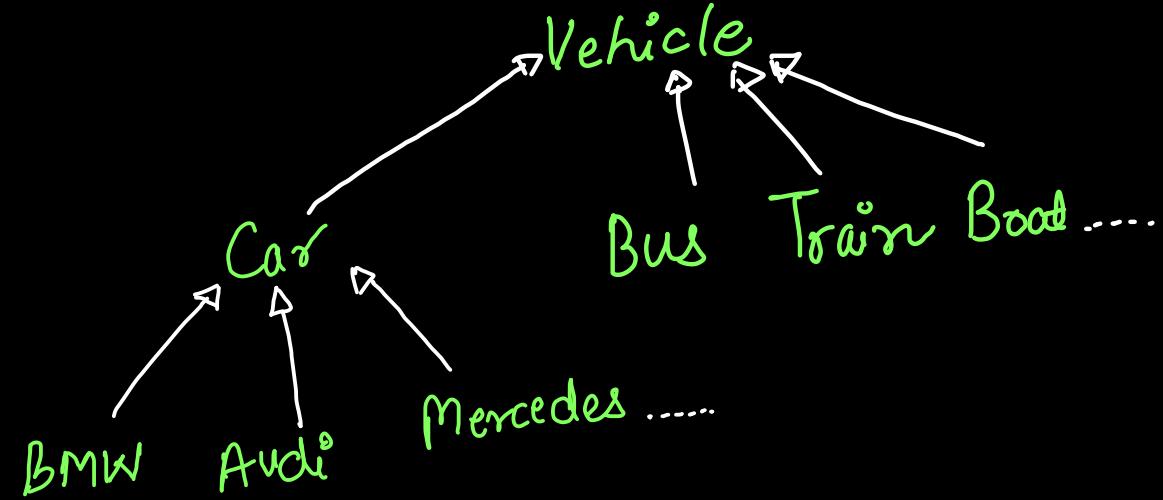
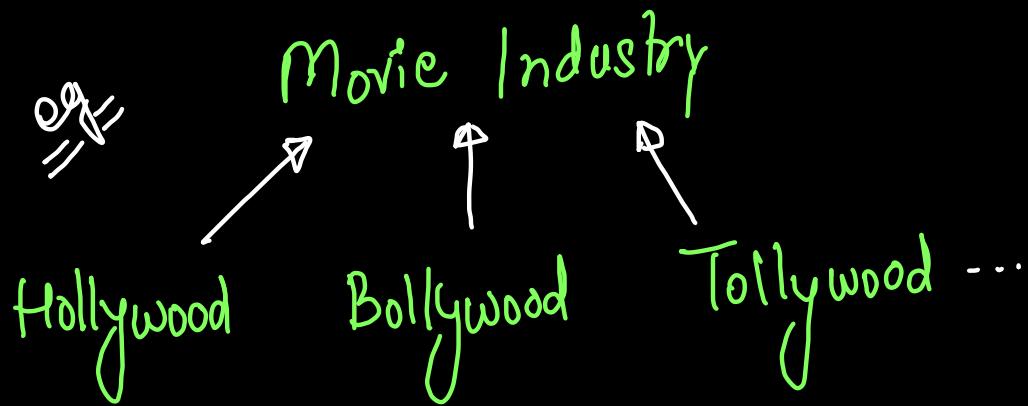
Solution: → Multiple interfaces cannot be implemented in a same class.

Child obj = new Child();  
x1 via Parent A  
x1 via Parent B

duplicate ⇒ ambiguity

## Generalization & Specialization





## Constructors in Inheritance

- 1) Are constructors of parent class inherited in child classes?  
    ↳ not inherited
- 2) Are constructors of parent class accessible in child class object?  
    ↳ not directly accessible  
    ↳ but indirectly accessible via constructor chaining
- 3) What do you mean by "super" keyword in Java?  
    reference variable used to refer immediate parent's  
    class object.  
    ↳ uses → access parent's instance variables  
                  → invoke parent's methods  
                  → invoke parent's constructor

```
class Parent {
 String parentData = "Parent";

 Parent() {
 System.out.println("Parent's Constructor");
 System.out.println(this.parentData);
 }
}
```

```
You, 1 second ago | 1 author (You)
class Child extends Parent {
 String childData = "Child";

 Child() {
 super();
 System.out.println("Child's Constructor");
 System.out.println(this.childData);
 System.out.println(super.parentData);
 }
}
```

You, 33 seconds ago | 1 author (You)  
class Driver {}  
Run | Debug  
public static void main(String[] args) {  
 Child obj = new Child();  
}  
You, 35 seconds ago • Uncommitted changes

Parent's Constructor  
Parent  
Child's Constructor  
Child  
Parent

```
class Parent {
 String parentData = "Parent";

 Parent() {
 System.out.println("Parent's Constructor");
 System.out.println(this.parentData);
 }

 public void parentFun() {
 System.out.println("This is Parent's Method");
 }
}
```

You, 29 seconds ago | 1 author (You)

```
class Child extends Parent {
 String childData = "Child";

 Child() {
 super(); // parent's constructor
 System.out.println("Child's Constructor");
 System.out.println(this.childData);

 System.out.println(super.parentData); // Access Parent's Data Members

 super.parentFun(); // Access Parent's Method
 }
}
```

You, 33 seconds ago | 1 author (You)

```
class Driver {
 Run | Debug
 public static void main(String[] args) {
 Child obj = new Child();
 }
}
```

You, 35 seconds ago • Uncommitted changes

Parent's Constructor  
Parent  
Child's Constructor  
Child  
Parent  
This is Parent's Method

- 4) When creating superclass object, how many constructors are called? How much memory is allocated?
- 5) When creating childclass object, how many constructors are called? How much memory is allocated?

```
class User {
 String name;
 String location;

 public User() {
 this.name = "Anonymous";
 this.location = "India";
 System.out.println("Guest User Created");
 }

 public void viewShow() {
 System.out.println("I can view listing shows on app");
 }
}
```

```
class RegisteredUser extends User {
 String emailId;
 long phoneNo;

 public RegisteredUser() {
 super();
 this.emailId = "registeredUser@gmail.com";
 this.phoneNo = 9319117888L;
 System.out.println("Registered User Created");
 }

 public void bookShow() {
 System.out.println("I can book the shows on app");
 }
}
```

① User() {  
 User user1 = new User();  
 System.out.println(user1.name);  
 System.out.println(user1.location);  
 user1.viewShow();  
}  
  
② User() {  
 RegisteredUser user2 = new RegisteredUser();  
 System.out.println(user2.name);  
 System.out.println(user2.location);  
 user2.viewShow();  
  
 System.out.println(user2.emailId);  
 System.out.println(user2.phoneNo);  
 user2.bookShow();  
}

```
Guest User Created
Anonymous
India
I can view listing shows on app
Guest User Created
Registered User Created
Anonymous
India
I can view listing shows on app
registeredUser@gmail.com
9319117888
I can book the shows on app
```

- 6) What is the order of invocation of constructors in inheritance? Is it same as order of constructor execution & object creation?
- 7) How to pass parameters & custom input to parent class refer variables, i.e. parameterized super constructor?
- 8) Can we write super constructor call anywhere in the child class constructor?

```
class User {
 String name;
 String location;

 public User() {
 this.name = "Anonymous";
 this.location = "India";
 System.out.println("Guest User Created");
 }

 public void viewShow() {
 System.out.println("I can view listing shows on app");
 }
}
```

```
class RegisteredUser extends User {
 String emailId;
 long phoneNo;

 public RegisteredUser() {
 super();
 this.emailId = "registeredUser@gmail.com";
 this.phoneNo = 9319117888L;
 System.out.println("Registered User Created");
 }

 public void bookShow() {
 System.out.println("I can book the shows on app");
 }
}
```

RegisteredUser user = new  
RegisteredUser()

Constructor Invocation / Calling

- ① RegisteredUser (Child)
- ② User (Parent)

Constructor Execution

- ① User (parent)
- ② RegisteredUser (Child)

```

class User {
 String name;
 String location;

 public User() {
 this.name = "Anonymous";
 this.location = "India";
 System.out.println("Guest User Created");
 }

 public void viewShow() {
 System.out.println("I can view listing shows on app");
 }
}

```

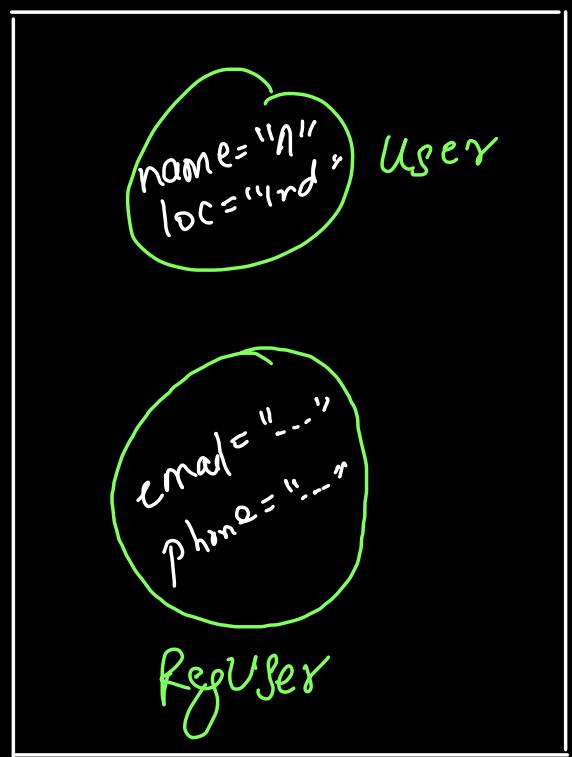
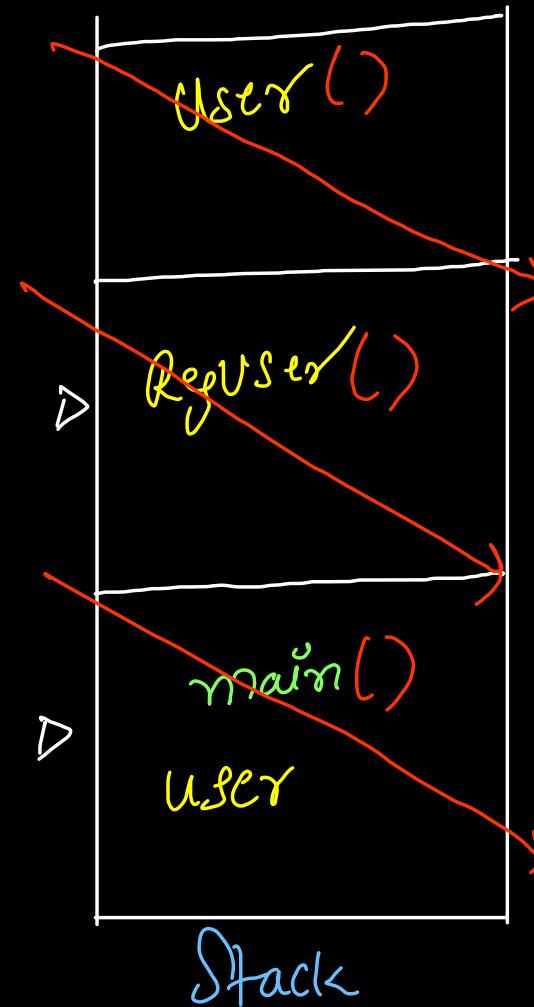
```

class RegisteredUser extends User {
 String emailId;
 long phoneNo;

 public RegisteredUser() { super(); }
 this.emailId = "registeredUser@gmail.com";
 this.phoneNo = 9319117888L;
 System.out.println("Registered User Created");

 public void bookShow() {
 System.out.println("I can book the shows on app");
 }
}

```



① Guest-User Created  
(Parent)

② RegUserCreated  
(child)

```
class User {
 String name;
 String location;

 public User() {
 this.name = "Anonymous";
 this.location = "India";
 System.out.println("Guest User Created");
 }

 public User(String name, String location) {
 this.name = name;
 this.location = location;
 }
}
```

```
class RegisteredUser extends User {
 String emailId;
 long phoneNo;

 public RegisteredUser() {
 // super();
 this.emailId = "registeredUser@gmail.com";
 this.phoneNo = 9319117881;
 System.out.println("Registered User Created");
 }

 public RegisteredUser(String emailId, long phoneNo) {
 // super();
 this.emailId = emailId;
 this.phoneNo = phoneNo;
 }

 public RegisteredUser(String name, String location, String emailId, long phoneNo) {
 super(name, location);
 this.emailId = emailId;
 this.phoneNo = phoneNo;
 }
}
```

```
RegisteredUser user3 = new RegisteredUser(emailId: "archit.aggarwal@gmail.com",
 phoneNo: 9319117889);
System.out.println(user3.name);
System.out.println(user3.location);
System.out.println(user3.emailId);
System.out.println(user3.phoneNo);

RegisteredUser user4 = new RegisteredUser(name: "archit", location: "Delhi",
 emailId: "archit@gmail.com", phoneNo: 9319117889);
System.out.println(user4.name);
System.out.println(user4.location);
System.out.println(user4.emailId);
System.out.println(user4.phoneNo);
```

③ {  
 Anonymous  
 India  
 archit.aggarwal@gmail.com  
 9319117889  
}  
  
④ {  
 archit  
 Delhi  
 archit@gmail.com  
 9319117889  
}

Q) What is the OBJECT CLASS in Java? Give the methods provided by it.

- Root of the class hierarchy in Java, i.e. topmost class.
- If not specified, every class (both already defined or user defined) will have parent class as Object class.
- Object class is parent class for every class directly or indirectly, i.e. every class will inherit Object class methods by default.
- It can be used to refer to any class' object whose type we don't know (Object class reference variable, child class object)  
↳ This is old fashioned way to implement "Generic Programming"

## # Important Object class methods

- ① **protected Object clone() throws CloneNotSupportedException**
  - Creates & return copy of this object.
  - By default : shallow copy
  - class overriding clone() method must implement Cloneable interface to tell JVM.
  - Object class does not implement cloneable, calling clone() method Object class will lead to Exception.
- ② **protected void finalize() throws Throwable**
  - called by Garbage Collector for objects that are not referenced by any one

- ③ public boolean equals ( Object obj )  
→ Indicates whether other object is equal to this one.  
→ By default (Object class) : compares on addresses!  
→ Overriding can be done on some/all properties equality.

Properties : →

- (1) Reflexive : a equals a
- (2) Symmetric : a equals b  $\Rightarrow$  b equals a
- (3) Transitive : a equals b & b equals c  $\Rightarrow$  a equals c
- (4) Consistent : a equals b remains same if a & b remain same
- (5) a.equals( null ) = false

```

class Movie {
 String name;
 int duration;
 double rating;

 Movie(String name, int duration, double rating) {
 this.name = name;
 this.duration = duration;
 this.rating = rating;
 }

 @Override
 public boolean equals(Object other) {
 if (this == other)
 return true;
 return this.name.equals(((Movie) other).name);
 }
}

```

```

Movie m1 = new Movie(name: "Endgame", duration: 180, rating: 4.9);
Movie m2 = m1;
System.out.println(m1.equals(m2)); // Object's are same: true

```

```

Movie m3 = new Movie(name: "Infinity War", duration: 150, rating: 4.7);
System.out.println(m1.equals(m3));

```

```

Movie m4 = new Movie(name: "Endgame", duration: 200, rating: 5.0);
System.out.println(m1.equals(m4));

```

without overriding

true  
false  
false

*default equals logic*

true

false

false

{ } ↓

*default equals logic*

with overriding

true  
false  
true

*custom equals logic*

*custom equals logic*

true

false

true

{ } ↓

*custom equals logic*

```

System.out.println(m1.equals(m1)); // Reflexive
System.out.println(m2.equals(m1)); // Symmetric
System.out.println(m2.equals(m4)); // Transitive
System.out.println(m1.equals(other: null)); // False

```

④ public final Class getClass()  
→ Returns runtime class of an object. "Class" object of this object  
can be used to get metadata of this class

Can't be overridden

Eg code

```
Object obj1 = new Object();
Class cobj1 = obj1.getClass();
System.out.println("Object 1 : " + obj1 + " ClassName: " + cobj1.getName());

Object obj2 = new String(original: "Hello World");
Class cobj2 = obj2.getClass();
System.out.println("Object 2 : " + obj2 + " ClassName: " + cobj2.getName());

Object obj3 = new Movie();
Class cobj3 = obj3.getClass();
System.out.println("Object 2 : " + obj3 + " ClassName: " + cobj3.getName());
```

Output

Finished in 98 ms

```
Object 1 : java.lang.Object@511d50c0 ClassName: java.lang.Object
Object 2 : Hello World ClassName: java.lang.String
Object 2 : Movie@1d44bcfa ClassName: Movie
```

5) `public int hashCode()`

→ Returns a hash code value for the object.  
It is used to search objects in collections such as HashSet, HashMap, etc.

Default Behavior: Every different Object (different memory address) will get unique hashCode.

→ Hashcode for a particular object will remain same forever, i.e. there cannot be two hashcodes for same object during a single run of application.

Note: → i) If two objects are equal (acc to equals() method), then they must give same hashCode values.

ii) If two objects are unequal, good hash function should give different hashCode for those objects.

⑥ public String toString()  
{ return getClass().getName() + "@" + Integer.toHexString(hashCode()); }

→ Returns string representation of object  
Default Behavior: classname @ unsigned hexadecimal hashcode

Other methods related to synchronization: →

- public final void notify()
- public final void notifyAll()
- public final void wait()
- public final void wait(long timeout)
- public final void wait(long timeout, int nanos)

} → lock released/thread out of critical section

} → lock acquired/thread going  
in critical section!

```
class Movie implements Cloneable {
 String name = "Avengers Endgame";
 int duration = 180;
 double ratings = 4.5;

 public Movie() {
 System.out.println("Object Created - Initialization by Constructor");
 }

 public Movie(String name, int duration, double ratings) {
 this.name = name;
 this.duration = duration;
 this.ratings = ratings;
 }

 @Override
 public boolean equals(Object other) {
 return this.name.equals(((Movie) other).name);
 }

 @Override
 protected Object clone() throws CloneNotSupportedException {
 return super.clone();
 }
}
```

```
@Override
public String toString() {
 return ("Name : " + this.name + " , Duration : "
 + this.duration + " , Ratings : " + this.ratings);
}

@Override
protected void finalize() throws Throwable {
 // Deallocation of Resources Holded by Object
 // (Database Connection, File Input/Output Streams, H/W Resources like Camera)
 // Similar to Destructors in C++
 // Automatically Called By Garbage Collector Daemon Thread handled by JVM
 System.out.println("Object Destroyed - Resources Clean Up By Finalize");
}

@Override
public int hashCode() {
 // Custom Hashing Logic
 return this.name.hashCode();
}
}
```

```
@SuppressWarnings("rawtypes")
public static void getClassDemo() {
 Object obj1 = new Object();
 Class cobj1 = obj1.getClass();
 System.out.println("Object 1 : " + obj1 + " ClassName: " + cobj1.getName());

 Object obj2 = new String(original: "Hello World");
 Class cobj2 = obj2.getClass();
 System.out.println("Object 2 : " + obj2 + " ClassName: " + cobj2.getName());

 Object obj3 = new Solution();
 Class cobj3 = obj3.getClass();
 System.out.println("Object 3 : " + obj3 + " ClassName: " + cobj3.getName());
}
```

- architaggarwal@Archits-MacBook-Air 01. Core Java Basics % java Solution  
Object 1 : java.lang.Object@136432db ClassName: java.lang.Object  
Object 2 : Hello World ClassName: java.lang.String  
Object 3 : Solution@7382f612 ClassName: Solution

```
public static void equalsDemo() {
 Movie a1 = new Movie(name: "Avengers Endgame", duration: 180, ratings: 4.9);

 // Different Movies
 Movie a2 = new Movie(name: "Avengers InfinityWar", duration: 150, ratings: 4.8);
 System.out.println(a1.equals(a2)); // False
 a1.equals(a2) → false

 // Exactly Same Movie (Addresses/References are Same)
 Movie a3 = a1;
 System.out.println(a1.equals(a3)); // True (Address Comparison First)
 a1.equals(a3) → true

 // Extended Version of Avengers Endgame
 Movie a4 = new Movie(name: "Avengers Endgame", duration: 200, ratings: 5.0);
 System.out.println(a1.equals(a4)); → true
 // By Default (Without Overriding : Object's equals) : False
 // With Overriding and Name Comparison (Movie's equals) : True
}
```

```
public static void cloneDemo() throws Exception {
 Movie a1 = new Movie(name: "Avengers Endgame", duration: 180, ratings: 4.9);
 System.out.println(a1);

 Movie a2 = (Movie) a1.clone();
 System.out.println(a2);
}
```

- architaggarwal@Archits-MacBook-Air 01. Core Java Basics % java Solution  
Name : Avengers Endgame , Duration : 180 , Ratings : 4.9  
Name : Avengers Endgame , Duration : 180 , Ratings : 4.9 *↳ toString methods*

```
public static void hashCodeDemo() {
 Movie a1 = new Movie(name: "Avengers Endgame", duration: 180, ratings: 4.9);
 System.out.println(a1.hashCode());

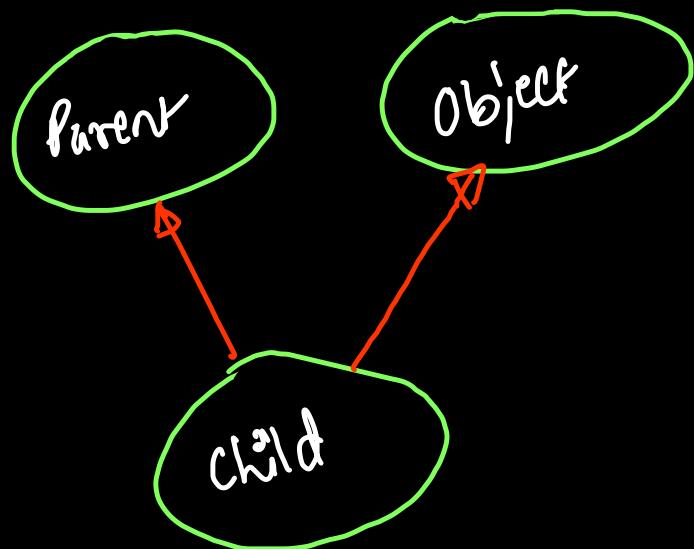
 Movie a2 = new Movie(name: "Avengers InfinityWar", duration: 150, ratings: 4.8);
 System.out.println(a2.hashCode());

 Movie a3 = a1;
 System.out.println(a3.hashCode());

 Movie a4 = new Movie(name: "Avengers Endgame", duration: 200, ratings: 5.0);
 System.out.println(a4.hashCode());
}
```

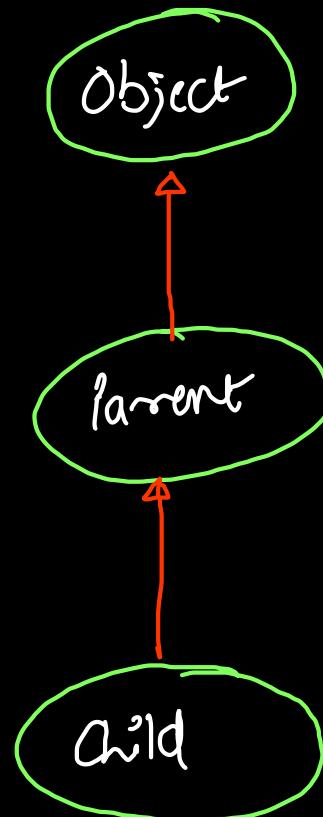
- architaggarwal@Archits-MacBook-Air 01. Core Java Basics % java Solution  
-2084956566  
1213999165  
-2084956566  
-2084956566

Q) If Object class is a parent class for Every class, so it means there might be multiple inheritance possible?



Multiple Inheritance(✗)

Hybrid Inheritance(✗)



multilevel inheritance!

Q) What do you mean by **POLYMORPHISM** in Java? What are the different types of Polymorphism? What are the advantages & disadvantages of Polymorphism?

**Polymorphism** :- Performing single action (behavior → method) in many different ways.

many form

Compile time or Static Polymorphism  
: method overloading

Two types

Runtime or Dynamic Polymorphism  
: method overriding : Dynamic  
method Dispatch

## Advantages

code redundancy ↘ → readability ↑, maintainability ↑, debugging ↑

↳ scalability ↑, availability ↑

- Code reusability is the main advantage of polymorphism; once a class is defined, it can be used multiple times to create an object.
- In compile-time polymorphism, the readability of code increases, as nearly similar functions can have the same name, so it becomes easy to understand the functions.
- The same method can be created in the child class as in the parent class in runtime polymorphism.
- Easy to debug the code. You might have intermediate results stored in arbitrary memory locations while executing code, which might get misused by other parts of the program. Polymorphism adds necessary structure and regularity to computation, so it is easier to debug.

## Disadvantages

- Implementing code is complex because understanding the hierarchy of classes and its overridden method is quite difficult.
- Problems during downcasting because implicitly downcasting is not possible. Casting to a child type or casting a common type to an individual type is known as downcasting.
- Sometimes, when the parent class design is not built correctly, subclasses of a superclass use superclass in unexpected ways. This leads to broken code.
- Runtime polymorphism can lead to the real-time performance issue (during the process), it basically degrades the performances as decisions are taken at run time because, machine needs to decide which method or variable to invoke.

Q) What do you mean by **Compile-time / static polymorphism** or **method overloading** in Java. Give some real world examples.

→ two or more methods in the same class

with same function-name, and different argument list

### Rules for Overloading

① Number of arguments should be different

or

② Order of arguments should be different

or

③ Type of argument should be different

Change in return type  
does not matter!

⇒ determined by  
compiler: which  
function call to be binded  
with which method  
definition.

*defn body*

```
class Sum {
 public void sum(int a, int b) {
 System.out.println(a + b);
 }

 // Number of Arguments
 public void sum(int a, int b, int c) {
 System.out.println(a + b + c);
 }

 // Datatypes of Arguments
 public void sum(String a, String b) {
 System.out.println(a + b);
 }

 public void sum(String a, int b) {
 System.out.println(a + b);
 }

 // Order of Arguments
 public void sum(int a, String b) {
 System.out.println(a + b);
 }

 25
 30
 ArchitAggarwal
 Archit's score: 100
 100 is Archit's score
```

*call (invocation)*

```
class Driver {
 Run | Debug
 public static void main(String[] args) {
 Sum obj = new Sum();

 obj.sum(a: 10, b: 15);
 obj.sum(a: 5, b: 10, c: 15);
 obj.sum(a: "Archit", b: "Aggarwal");
 obj.sum(a: "Archit's score: ", b: 100);
 obj.sum(a: 100, b: " is Archit's score");
 }
}
```

*Re-declaration error (Compilation)*

*overloading ↗*

```
// Compilation Error: Variable names
// does not matter
// public void sum(int c, int d){
// }

// Compilation Error: Return type does
// not matter
// public int sum(int a, int b){
// return a + b;
// }
```

```
class User {
 String name;

 public void setName(String firstName, String middleName, String lastName) {
 name = firstName + " " + middleName + " " + lastName;
 }

 public void setName(String firstName, String lastName) {
 name = firstName + " " + lastName;
 }

 public void setName(String firstName) {
 name = firstName;
 }

 public void setName(char firstLetter, char secondLetter, String lastName) {
 name = firstLetter + ". " + secondLetter + ". " + lastName;
 }
}
```

```
User u1 = new User();
u1.setName(firstName: "Sachin",
middleName: "Ramesh", lastName: "Tendulkar");
System.out.println(u1.name);

'User u2 = new User();
u2.setName(firstName: "Virat",
lastName: "Kohli");
System.out.println(u2.name);

User u3 = new User();
u3.setName(firstName: "Tejas");
System.out.println(u3.name);

User u4 = new User();
u4.setName(firstLetter: 'K',
secondLetter: 'L', lastName: "Rahul");
System.out.println(u4.name);
```

Real World  
Example

Sachin Ramesh Tendulkar  
Virat Kohli  
Tejas  
K. L. Rahul

Q) What do you mean by run-time/dynamic polymorphism or method overriding in Java. Give some real world examples.

↳ One method in parent class, another method in child class, with same function prototype

# In runtime, JVM binds the function call to the corresponding function definition

- same function name
- same return type \*(T&C conditions)
- Same argument list.
  - same no of arguments
  - same order of arguments
  - datatype of arguments same

# There must be inheritance for over-riding, ie. two methods in different classes related as parent-child.

```
class Parent {
 private int a, b;

 public int getA() {
 return a;
 }

 public void setA(int a) {
 this.a = a;
 }

 public int getB() {
 return b;
 }

 public void setB(int b) {
 this.b = b;
 }

 // overriden method
 public void printObject(int a, int b) {
 System.out.println("Parent's Object : ");
 this.setA(a);
 this.setB(b);
 System.out.println(this.getA() + " " + this.getB());
 }
}
```

```
class Child extends Parent {
 private int p, q;

 public int getP() {
 return p;
 }

 public void setP(int p) {
 this.p = p;
 }

 public int getQ() {
 return q;
 }

 public void setQ(int q) {
 this.q = q;
 }

 // overridden method
 public void printObject(int p, int q) {
 System.out.println("Child Object : ");
 this.setP(p);
 this.setQ(q);
 System.out.println(super.getA() + " " + super.getB());
 System.out.println(this.getP() + " " + this.getQ());
 }
}
```

```
public static void main(String[] args) {
 Parent obj1 = new Parent();
 obj1.printObject(a: 10, b: 20);
 // overriden method -> Parent

 Child obj2 = new Child();
 obj2.printObject(p: 40, q: 50);
 // overriding method -> Child
```

Parent's Object :  
10 20  
Child Object :  
0 0  
40 50

```
class User {
 String name;
 String location;

 public User(String name, String location) {
 this.name = name;
 this.location = location;
 }

 // overriden method
 public void bookShow() {
 System.out.println(this.name + " " + this.location);
 System.out.println(x: "Error: You cannot book the show");
 System.out.println(x: "Please, first login or sign up");
 }
}
```

```
User u1 = new User(name: "Archit", /location: "Delhi");
u1.bookShow();

RegisteredUser u2 = new RegisteredUser(name: "Archit",
location: "Delhi", emailId:"archit@gmail.com",
phoneNo: 9319117889);
u2.bookShow();
```

```
class RegisteredUser extends User {
 String emailId;
 long phoneNo;

 public RegisteredUser(String name, String location, String emailId,
long phoneNo) {
 super(name, location);
 this.emailId = emailId;
 this.phoneNo = phoneNo;
 }

 // overriden method
 public void bookShow() {
 System.out.println(super.name + " " + super.location + " " +
this.emailId + " " + this.phoneNo);
 System.out.println(x: "Please select the number of seats");
 System.out.println(x: "And proceed to payment gateway");
 }
}
```

Archit Delhi  
Error: You cannot book the show  
Please, first login or sign up  
Archit Delhi archit@gmail.com 9319117889  
Please select the number of seats  
And proceed to payment gateway

```
class Child extends Parent {
 private int p, q;

 public int getP() {
 return p;
 }

 public void setP(int p) {
 this.p = p;
 }

 public int getQ() {
 return q;
 }

 public void setQ(int q) {
 this.q = q;
 }
}
```

```
Parent's Object :
10 20
Parent's Object :
40 50
```

} if child does not have  
overriding method

~~Q)~~ What are the differences between method overloading and method overriding?

### Method Overloading

→ by Java compiler binding

- ① implements compiletime polymorphism
- ② methods are statically binded  
(method call determined at compile time)
- ③ methods must be in same class
- ④ methods must have different argument list.
- ⑤ return type may or may not be same

### Method Overriding

→ by JVM binding

- ① implements runtime polymorphism
- ② methods are dynamically binded  
(method call determined at run-time)
- ③ methods must be in super & subclass
- ④ methods must have same signature (same return type & argument - list)
- ⑤ Return type (\*) must be same  
↳ co-variant type

Q) What do you mean by **static binding** . What do you understand  
by **dynamic binding**? What are the differences?  
What do you mean by **dynamic method dispatch**?

```
class Movie {
 int duration = 180;
 String name = "Avengers Endgame";

 public void display() {
 System.out.println(this.name + " runs for " +
 this.duration);
 }
}

class Driver {
 Run | Debug
 public static void main(String[] args) {
 Movie avengers = new Movie();
 avengers.display();
 }
}
```

static binding {Compile time}  
no polymorphism

## Static Binding

- binding of function call to the corresponding function definition at compile-time by compiler is known as static binding.
- It is done for methods which are not over-ridden, and may/may not be overloaded.
- Also known as early binding
- static, final, private methods are always statically binded.

## Dynamic Binding

- binding of function call to the corresponding function definition at runtime by JVM is known as dynamic binding.
- It is done for methods which are over-ridden in the child class. → dynamic method dispatch!
- Also known as late binding
- Abstract methods are always late binded.

```
class User {
 String name, location;

 User() {
 this.name = "Anonymous";
 this.location = "India";
 }

 User(String name) {
 this.name = name;
 this.location = "India";
 }

 User(String name, String location) {
 this.name = name;
 this.location = location;
 }

 public void display() {
 System.out.println(this.name + ", " + this.
 location);
 }
}
```

Overloading  
Static Binding

```
class Driver {
 Run | Debug
 public static void main(String[] args) {
 Movie avengers = new Movie();
 avengers.display();

 User u1 = new User();
 u1.display();
 User u2 = new User(name: "Archit");
 u2.display();
 User u3 = new User(name: "Archit",
 location: "Delhi");
 u3.display();
 }
}
```

```
class RegisteredUser extends User {
 String phone = "9319117889";

 @Override
 public void display() {
 System.out.println(this.name + ", " + this.
 location + ", " + this.phone);
 }
}
```

dynamic binding of runtime polymorphism  
overriding

```
RegisteredUser u4 = new RegisteredUser();
u4.display();
```

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

 public void lateBind(){
 System.out.println("Late Bind in Parent Class");
 }
}

class B extends A{
 @Override
 public void lateBind(){
 System.out.println("Late Bind in Child Class");
 }
}

public class Main{
 public static void main(String[] args){
 A obj = new B();
 obj.earlyBind(); → Early
 obj.lateBind(); → Child
 }
}
```

Q) What are other forms of polymorphism in Java?

- Implicit operator overloading
- Coersion (implicit or explicit type conversion)

# We cannot explicitly achieve operator overloading in Java

↳ complex number addition

$c1 = 5+3i$     }     $\oplus \alpha$   
 $c2 = 7+9i$     }    add() ✓

```
public static void main(String[] args) {
 int var1 = 100;
 int var2 = 200;

 System.out.println(var1 + var2);
 // + => Addition of two integers

 String str1 = "Archit";
 String str2 = "Aggarwal";
 System.out.println(str1 + str2);
 // + => Concatenation of String

 System.out.println(var1 + var2 + str1);
 // Addition then Concatenation

 System.out.println(str1 + var1 + var2);
 // Concatenations only
 System.out.println(var1 + str1 + var2);

 System.out.println(x: '0t Archit 1n Aggarwal');
```

*+ operator  
overloaded*

300  
ArchitAggarwal  
300Archit  
Archit100200  
100Archit200  
Archit  
Aggarwal

```
public static void fun(long var) {
 System.out.println(var);
}
```

Conversion

```
fun(var: 99999999999l); // long -> long
fun(var: 200); // int -> long
fun(var: 'A'); // char -> long
```

} implicit type conversion  
↳ upcasting  
smaller data → big data

```
public static void fun2(char var) {
 System.out.println(var);
}
```

```
fun2((char) 100l); // long -> char
fun2((char) 35); // int -> char
fun(var: 'A'); // char -> char
```

} explicit type conversion  
↳ downcasting  
big data → smaller data

Q) Which of the following are correct declarations? Why or why not?

- (a) Parent obj1 = new Parent(); valid
- (b) Child obj2 = new Child(); valid
- \* (c) Parent obj3 = new Child(); valid
- (d) Child obj4 = new Parent(); not valid
- Parent  
↑ extends  
Child

Q) What are Polymorphic Variables in Java? Give some real-world examples.

```
class User {
 String name, address;

 public String getName() {
 return name;
 }

 public void setName(String name) {
 this.name = name;
 }

 public String getAddress() {
 return address;
 }

 public void setAddress(String address) {
 this.address = address;
 }

 public User(String name, String address) {
 this.name = name;
 this.name = address;
 }

 public void bookShow() {
 System.out.println("You can't book show.
 Please register first");
 }
}
```

```
class RegisteredUser extends User {
 String phoneNo, emailId, address;

 public String getPhoneNo() {
 return phoneNo;
 }

 public void setPhoneNo(String phoneNo) {
 this.phoneNo = phoneNo;
 }

 public String getEmailId() {
 return emailId;
 }

 public void setEmailId(String emailId) {
 this.emailId = emailId;
 }

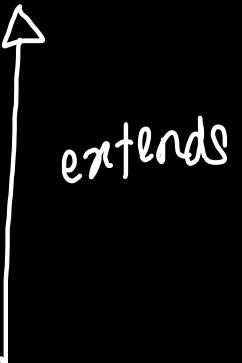
 public String getAddress() {
 return address;
 }

 public void setAddress(String address) {
 this.address = address;
 }

 public RegisteredUser(String name, String address, String phoneNo, String
emailId) {
 super(name, address: "Delhi");
 this.phoneNo = phoneNo;
 this.emailId = emailId;
 this.address = address;
 }

 public void bookShow() {
 System.out.println("You can book the show, please proceed to payments");
 }
}
```

GuestUser (Parent)



Registered User  
(Child)

name, address<sup>→ city</sup>  
getters & setters

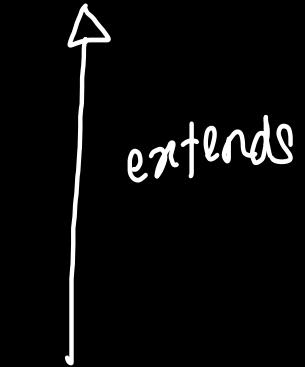
bookshow()  
{ sys("can't"); }

phonoNo, emailID, address<sup>→ complete address</sup>  
getters & setters

bookshow()  
{ sys("can"); }

GuestUser (Parent)

Registered User  
(Child)



✓ name, address<sup>✓ city</sup> bookshow() ✓  
✓ getters & setters { sys("can't"); }

✗ phoneNo, emailID, address<sup>✗ complete address</sup> bookshow()  
✗ getters & setters { sys("can"); }

GuestUser user = new GuestUser();  
↑  
reference variable  
↑  
Object/Instance

```
User u1 = new User(name: "Archit",
address: "Delhi");

System.out.println(u1.getName() + " " + u1.
getAddress());

u1.bookShow();

System.out.println(u1.name + " " + u1.
address);
```

```
Archit Delhi
You can't book show. Please register first
Archit Delhi
```

GuestUser (Parent)

✓ name, address  
✓ getters & setters

accessible (indirectly)  
super ↗

bookshow()  
super ↗

{ sys("can't"); }

Registered User  
(Child)

✓ phoneNo, emailID, address  
✓ getters & setters

✓ bookshow()

{ sys("can"); }

Reg User user = new

↑  
reference variable

Reg User();

↑  
Object/Instance

user, address  
↳ child ✓  
↳ parent ✓

super-address ✓

```
RegisteredUser u2 = new RegisteredUser
(name: "Archit", address: "Delhi 110085",
phoneNo: "9319117889", emailId: "archit.
aggarwal023@gmail.com");
```

```
System.out.println(u2.name + ", " + u2.
emailId + ", " + u2.phoneNo);
```

```
u2.bookShow(); // overriding → "can book the show"
```

```
System.out.println(u2.address); → this hides
System.out.println(u2.getAddress());
```

parent's address  
variable

Archit, archit.aggarwal023@gmail.com, 9319117889

You can book the show, please proceed to payments

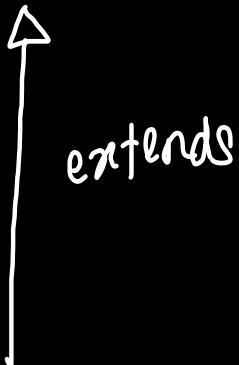
Delhi 110085

Delhi e-super address(indirect)

Delhi 110085

this address (direct access)

GuestUser (Parent)



Registered User  
(Child)

✓ name, address  
getters & setters ✓

overriden  
bookshow()

{ sys("can't"); }

↓ @overriden

{  
    <sup>created but not accessible</sup>  
    phonen<sup>o</sup>, emai<sup>lID</sup>, addre<sup>s</sup>  
    getters & setters

overriden  
bookshow()

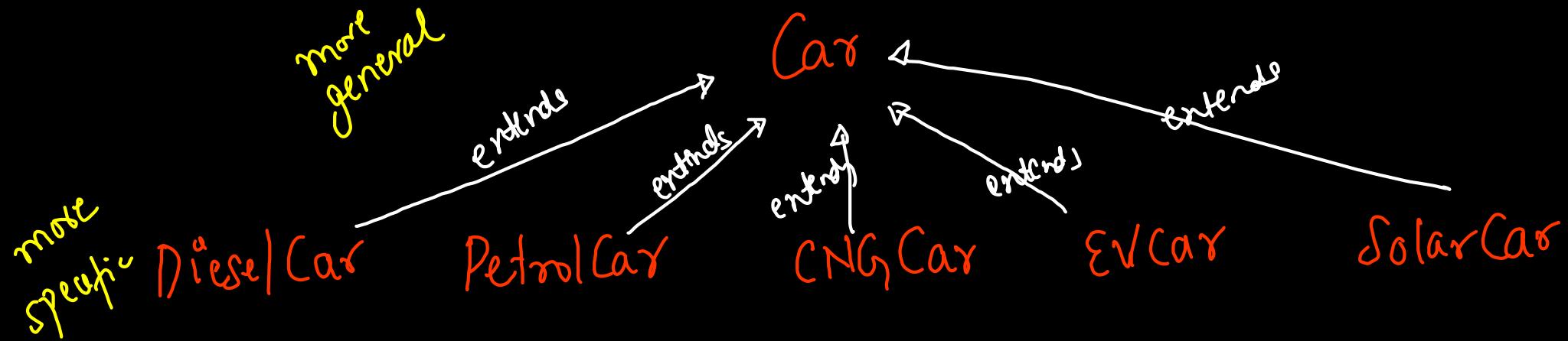
{ sys("can"); }

GuestUser    user = new    RegisteredUser();

↑  
reference variable  
Variables & function prototypes

↑  
Object/Instance  
function body

⇒ Reference variable pointing to different objects are known as polymorphic variables.



DieselCar c1 = new DieselCar();

PetrolCar c2 = new PetrolCar();

CNGCar c3 = new CNGCar();

EVCar c4 = new EVCar();

SolarCar c5 = new SolarCar();

This is not preferred

Car c1 = new DieselCar();

Car c2 = new PetrolCar();

Car c3 = new CNGCar();

Car c4 = new EVCar();

Car c5 = new SolarCar();

Car is a polymorphic variable

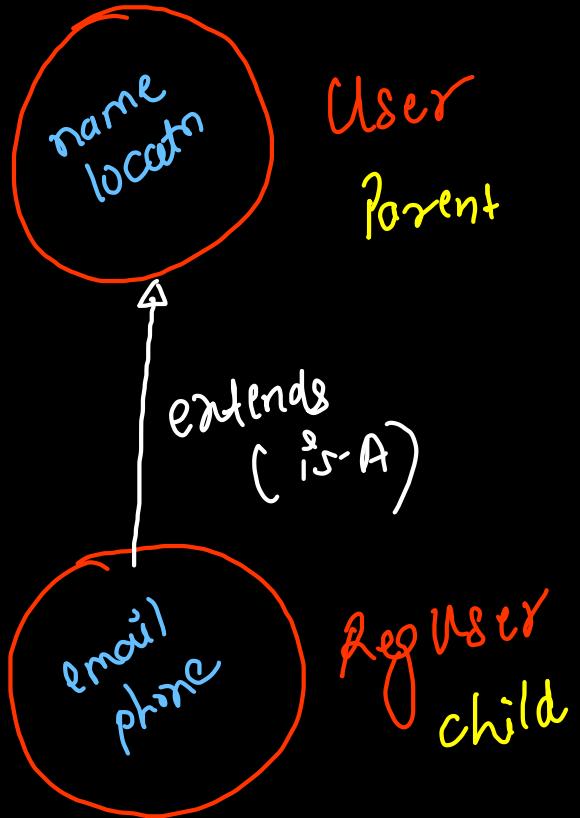
Registered User     $U4 = \text{new GuestUser( )};$

                      ↓  
emailId &  
phoneno

$\text{GuestUser( )}$   $\xrightarrow[\text{Subclass}]{\text{child}}$   ~~$\text{Register( )}$~~   
~~constructor~~

Q) What do you mean by upcasting and downcasting? Out of both, which is done implicitly and which needs to be done explicitly. Give some coding examples.

Q) What is instanceof operator and when to use it? What are the advantages of using it?



User v1 = new User(); ✓ allowed

RegUser v2 = new RegUser(); ✗ allowed

User v3 = new RegUser(); ✗ typecast  
child

wrong { RegUser v4 = new User(); }  
child Parent

Implicit upcasting

RegUser v5 = v2; // shallow copy  
allowed

RegUser v6 = v3; // compilation error  
downcasting

RegUser v6 = (RegUser)v3; // allowed

RegUser v7 = v1; // compilation error

v7 = (RegUser)v1; // runtime error

```

public static void main(String[] args) {
 RegisteredUser u1 = new RegisteredUser(name: "archit", location: "delhi",
emailId: "archit@gmail.com", phoneNo: 9319117888l);
System.out.println(u1.name + " " + u1.emailId);

// Child: Ref & Object

User u2 = new User(); // Parent : Ref & Object
System.out.println(u2.name);

User u3 = new RegisteredUser(); // Parent Ref, Child Object
System.out.println(u3.name + ((RegisteredUser) u3).emailId);

// RegisteredUser u4 = new User(); child ref parent object not allowed

RegisteredUser u5 = u1; // shallow copy
System.out.println(u5.name + " " + u5.emailId);

// RegisteredUser u6 = u3; // Compilation Error: No Typecasting
if (u3 instanceof RegisteredUser) {
 RegisteredUser u6 = (RegisteredUser) u3; ↳ explicit down casting
 System.out.println(u6.name + u6.emailId);
} else {
 System.out.println(x: "This user is not registered");
}

// RegisteredUser u7 = u2; // Compilation Error
// RegisteredUser u7 = (RegisteredUser) u2;
// Runtime Error: Classcast Exception

if (u2 instanceof RegisteredUser) {
 RegisteredUser u7 = (RegisteredUser) u2;
 System.out.println(u7.emailId);
} else {
 System.out.println(x: "This user is not registered");
}
}

```

archit archit@gmail.com  
 Guest User Created  
 Anonymous  
 Guest User Created  
 Registered User Created  
 Anonymous registeredUser@gmail.com  
 archit archit@gmail.com  
 Anonymous registeredUser@gmail.com  
 This user is not registered

Q) What do you mean by method hiding? Give some examples.

- overriding static methods
- overriding private methods
- change in parameters in overridden methods

How is it different from method over-riding?

Parent{

method()



Child extends Parent{

method()

}

}

```
class Parent {
 int parentData;

 public static void staticFun() {
 System.out.println("This is parent's static function");
 }
}

class Child extends Parent {
 int childData;

 public static void staticFun() {
 System.out.println("This is child's static function");
 }
}
```

## Static Functions

Overriding ✗

Hiding ✓

dynamic binding ✗

static binding ✓

```
class Driver {
 @SuppressWarnings("all")
 Run | Debug
 public static void main(String[] args) {
 Parent obj1 = new Parent();
 obj1.staticFun(); → Parent

 Child obj2 = new Child();
 obj2.staticFun(); → Child

 Parent obj3 = new Child();
 obj3.staticFun(); // Parent's Fun → Parent

 // No Dynamic Method Dispatch: No overriding
 }
}
```

```
class Parent {
 int parentData;

 public static void staticFun() {
 System.out.println("This is parent's static function");
 }

 public void privateFun() {
 System.out.println("This is parent's private function
but it is public");
 }

 public void publicFun() {
 System.out.println("This is parents's fun with 0
parameter");
 }
}
```

```
class Child extends Parent {
 int childData;

 public static void staticFun() {
 System.out.println("This is child's static function");
 }

 // private void privateFun() {}
 // Parent Public -> Child Private

 public void publicFun(int data) {
 System.out.println("This is child's fun with 1
parameter");
 }
}
```

```
class Driver {
 @SuppressWarnings("all")
 Run | Debug
 public static void main(String[] args) {
 Parent obj1 = new Parent();
 obj1.staticFun();
 obj1.publicFun();
 // obj1.publicFun(10); // This is not allowed

 Child obj2 = new Child();
 obj2.staticFun();

 obj2.publicFun();
 obj2.publicFun(data: 10);
 // Overloading with functions in different classes

 Parent obj3 = new Child();
 obj3.staticFun(); // Parent's Fun
 obj3.publicFun(); // Parent's Fun

 // No Dynamic Method Dispatch: No overriding
```

Q) Why data members cannot be over-ridden ? What do you understand by variable hiding in Java ?

Instance variable hiding refers to a state when instance variables of the same name are present in superclass and subclass. Now if we try to access using subclass object then instance variable of subclass hides instance variable of superclass irrespective of its return types.

Q) Can constructors be overriden ? Give reasons .

We can not override constructor as parent and child class can never have constructor with same name (Constructor name must always be same as Class name).

Q) Can overrided methods have different return types? What do you mean by covariant types?

The covariant return type specifies that the return type may vary in the same direction as the subclass. Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type

```
User getObject() {
 System.out.println(x: "Parent's GetObject Function");
 System.out.println(x: "Overriden Variables");
 return null;
}
```

fn

```
// overriding: Covariant Type
RegUser getObject() {
 System.out.println(x: "This is Chid's Get Object Function");
 System.out.println(x: "Overdided Function");
 return this;
}
```

```
class Parent {
 public void fun() {
 System.out.println("This is parent's void fun");
 }

 public Parent getObject() {
 System.out.println("This is parent's GetObject");
 return new Parent();
 }
}

class Child extends Parent {
 // public int fun(){
 // System.out.println("This is parent's int fun");
 // }
 // Invalid: Child should have void returntype

 @Override
 public Child getObject() {
 System.out.println("This is child's GetObject");
 return new Child();
 }
 // Non-Primitive Covariant Type
}
```

```
class Driver {
 Run | Debug
 public static void main(String[] args) {
 Parent obj1 = new Parent();
 Child obj2 = new Child();

 Parent obj3 = obj1.getObject();
 obj3.fun();
 // Parent Ref: Parent Object

 // Child obj4 = obj1.getObject(); // Not Valid

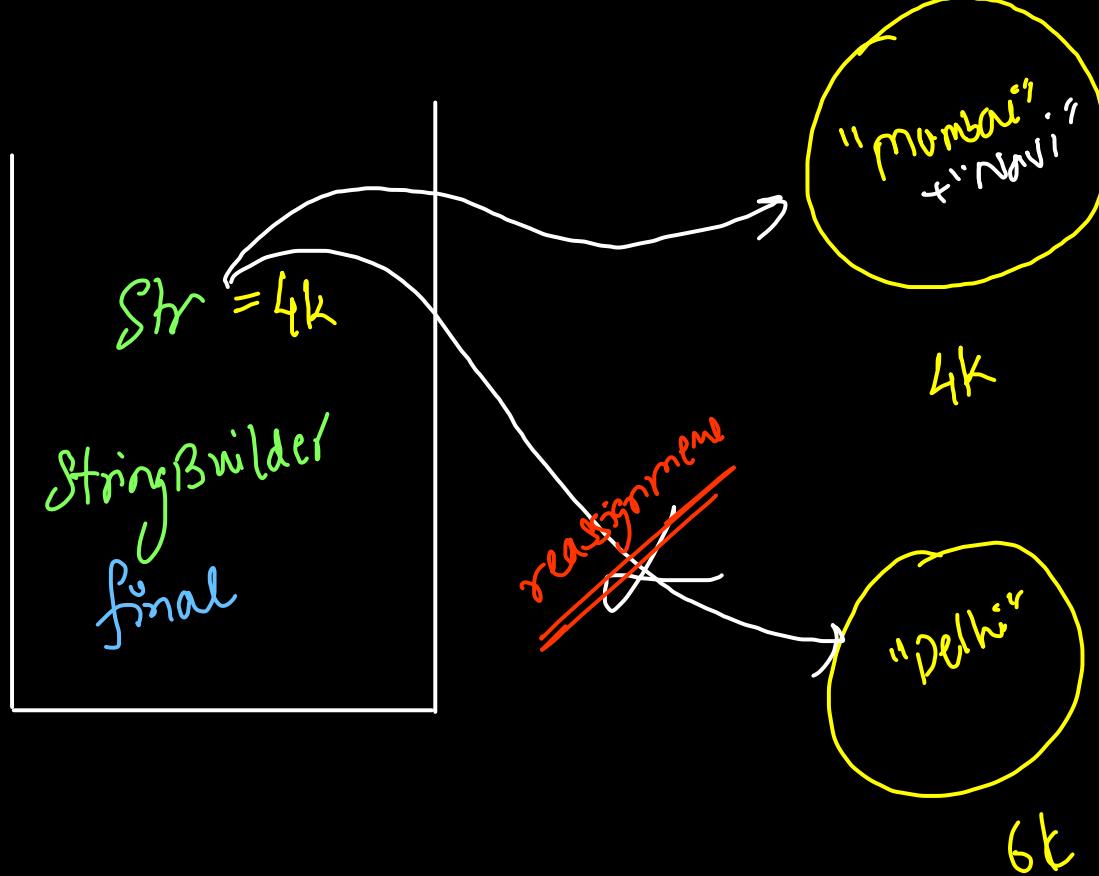
 Parent obj5 = obj2.getObject(); // Overriding
 obj5.fun();
 // Parent Ref: Child Object

 Child obj6 = obj2.getObject();
 obj6.fun();
 // Child Ref: Child Object
 }
}
```

```
This is parent's GetObject
This is parent's void fun
This is child's GetObject
This is parent's void fun
This is child's GetObject
This is parent's void fun
```

Q) What are the applications of final keyword?

- final primitive variables : constants → CAPITAL LETTERS
- final reference variables : Reference can't be changed  
but data(instance) can be.
- final methods : can't be over-ridden (to stop runtime polymorphism)  
↳ static binding
- ↳ final class : can't be extended (to stop inheritance)  
" " inherited
  - ↳ A final class will automatically have all its functions final



```

class Driver {
 private static final double PI = 3.14;
 private static final StringBuilder str = new
 StringBuilder(str: "Mumbai");

 Run | Debug
 public static void main(String[] args) {
 // Get: Constant Variable: Allowed
 System.out.println(Driver.PI);

 // Set: Constant Variable: Not Allowed
 // Driver.PI = 22/7.0;
 // Reassignment not possible

 System.out.println(Integer.MIN_VALUE);
 System.out.println(Integer.MAX_VALUE);

 // Get And Data Modification: Final Reference
 // Variable
 System.out.println(str);
 str.append(str: " Navi");
 System.out.println(str);

 // Reassignment Not Allowed
 // str = new StringBuilder("Delhi");
 }
}

```

```
class Parent {
 public final void finalFun() {
 System.out.println(x: "This is parent's fun");
 }
}

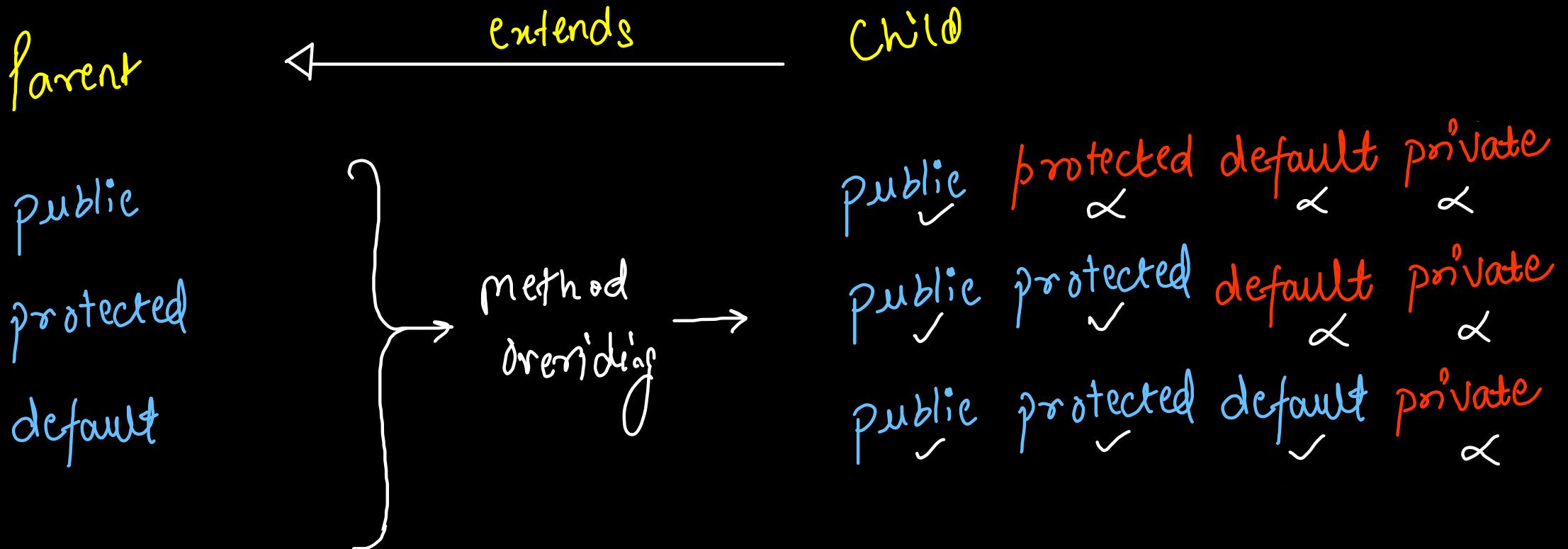
class Child extends Parent {
 // Final Method can't be overrided
 // public void finalFun(){}
}

final class Parent2 {
 public void finalFun() {
 System.out.println(x: "This is parent2's fun");
 }
}

// Final Class cannot be extended
// class Child2 extends Parent2{}
```

# Access Modifiers & Method Overriding

Q) Explain the statement "Overrided method (child class)  
must be less restrictive than overriden method (parent class)"



```
Parent {
 public void fun() {
 }
}

child {
 private void fun() {
 // should be accessible within
 // child class
 }
}
```

← extends

allowed to access  
private method  
outside the  
class

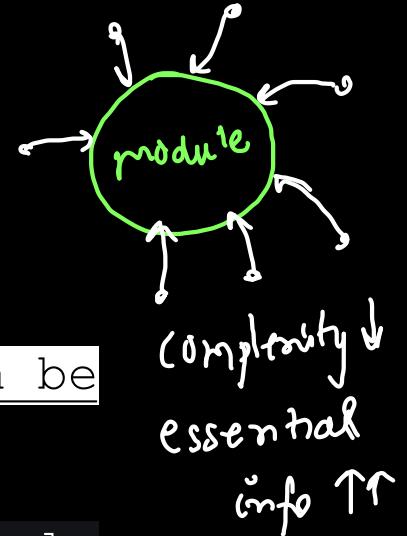
Dynamic  
method  
Dispatch

Parent obj = new Child();  
obj.fun(); // over-riding → Dynamic  
 // method Dispatch

Q) What do you mean by **ABSTRACTION** in Java? What are the **advantages**? How to achieve/implement abstraction?  
→ Complexity ↓, redundancy ↓, consistency ↑

Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.



App

O

1 → 2

Q) What is an abstract class? What are abstract methods?  
How is abstract class different from concrete class?

- abstract class cannot be instantiated
- abstract class may have some abstract methods.
  - ↳ method with only function prototype (access modifier, return type, function name, argument list) but no body
- abstract class is a more generic class whereas concrete class is a more specific class.
- Concrete class (implementation) depends upon abstract class (abstraction)

```

abstract class Car {
 abstract void refuel();
 abstract void engine();
}

class PetrolCar extends Car {
 @Override
 void refuel() {
 System.out.println("Petrol Refill");
 }

 @Override
 void engine() {
 System.out.println("It has a Petrol Engine");
 }
}

class EVCar extends Car {
 @Override
 void refuel() {
 System.out.println("Battery recharge");
 }

 @Override
 void engine() {
 System.out.println("Spark/Electricity based engine");
 }
}

```

generic class { doesn't exist in real life }

↑  
Specialized class

abstract refuel, engine  
Car

PetrolCar      EVCar  
@override  
refuel, engine      @override  
refuel, engine

```

class Driver {
 Run | Debug
 public static void main(String[] args) {
 // Car obj = new Car();
 // We cannot create objects of Car (abstract class)

 PetrolCar obj = new PetrolCar();
 obj.refuel();
 obj.engine();

 EVCar obj2 = new EVCar();
 obj2.refuel();
 obj2.engine();
 }
}

```

```
class Driver {
 Run | Debug
 public static void main(String[] args) {
 // Car obj = new Car();
 // We cannot create objects of Car (abstract
 // class)

 PetrolCar obj = new PetrolCar();
 obj.refuel();
 obj.engine();

 EVCar obj2 = new EVCar();
 obj2.refuel();
 obj2.engine();

 // Polymorphism
 Car c1 = new PetrolCar();
 c1.refuel();
 System.out.println(c1.color);

 Car c2 = new EVCar();
 c2.refuel();
 System.out.println(c2.color);
 }
}
```

```
Petrol Refill
It has a Petrol Engine
Battery recharge
Spark/Electricity based engine
Petrol Refill
Red
Battery recharge
Red
```

Q) Can abstract class have (a) zero (b) some (c) all methods as abstract?

↓  
yes      ↓  
yes      ↓  
yes

all concrete  
methods (0% - 100%)

↓  
all abstract  
methods

Q) Can there be an abstract method inside a concrete (non-abstract) class?

No, concrete class will throw compilation error

Abstract method can be defined in abstract class only.

Q) Can abstract methods be (a) final (b) static (c) private?

↓  
overriding ✓

↓  
Overriding  
no

↓  
method hiding  
overriding  
early bounded  
no

↓  
early bounded ✓  
overriding  
no

- Q) Constructors & abstract classes :-
- (a) can there be constructors inside a abstract class? Yes  
↳ to instantiate child class objects
- (b) can constructor itself be abstract? overriding ✓  
↳ abstract constructor  
↳ no  
↳ constructor chaining  
↳ super();
- (c) Does abstract class have "this" keyword?  
↳ yes → due to object creation of child

```
You, 10 seconds ago | 1 author (You)
abstract class Car {
 String color;

 public Car() {
 color = "White";
 }

 public Car(String color) {
 ✓ this.color = color;
 }

 abstract void refuel();

 // static Abstract,
 // private abstract,
 // final abstract
 // These are invalid combinations

 abstract void engine();

 void drive() {
 System.out.println("Drive Car");
 }
}
```

You, 31 seconds ago | 1 author (You)

```
class PetrolCar extends Car {
 String fuel;

 PetrolCar() {
 super();
 this.fuel = "Petrol";
 }

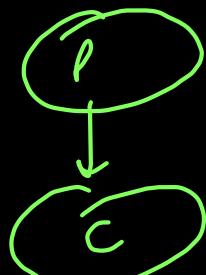
 PetrolCar(String fuel, String color) {
 ✓ super(color); // constructor: Abstract Class
 this.fuel = fuel;
 }

 @Override
 void refuel() {
 System.out.println("Petrol Refill");
 }

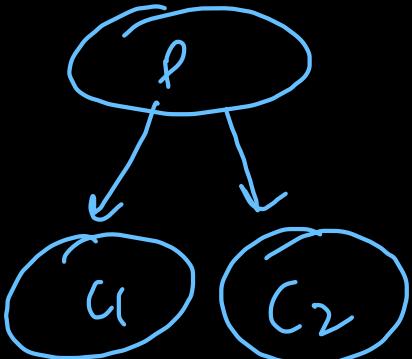
 @Override
 void engine() {
 System.out.println("It has a Petrol
 Engine");
 }
}
```

```
PetrolCar obj3 = new PetrolCar(fuel: "petrol -
Xtra", color: "Black");
System.out.println(obj3.color); → Black
System.out.println(obj3.fuel); → petrol-Xtra
```

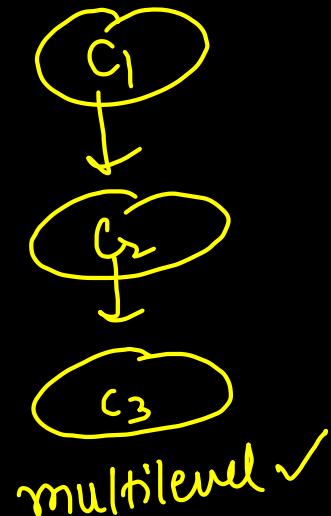
Q) Can we implement multiple inheritance using abstract classes?



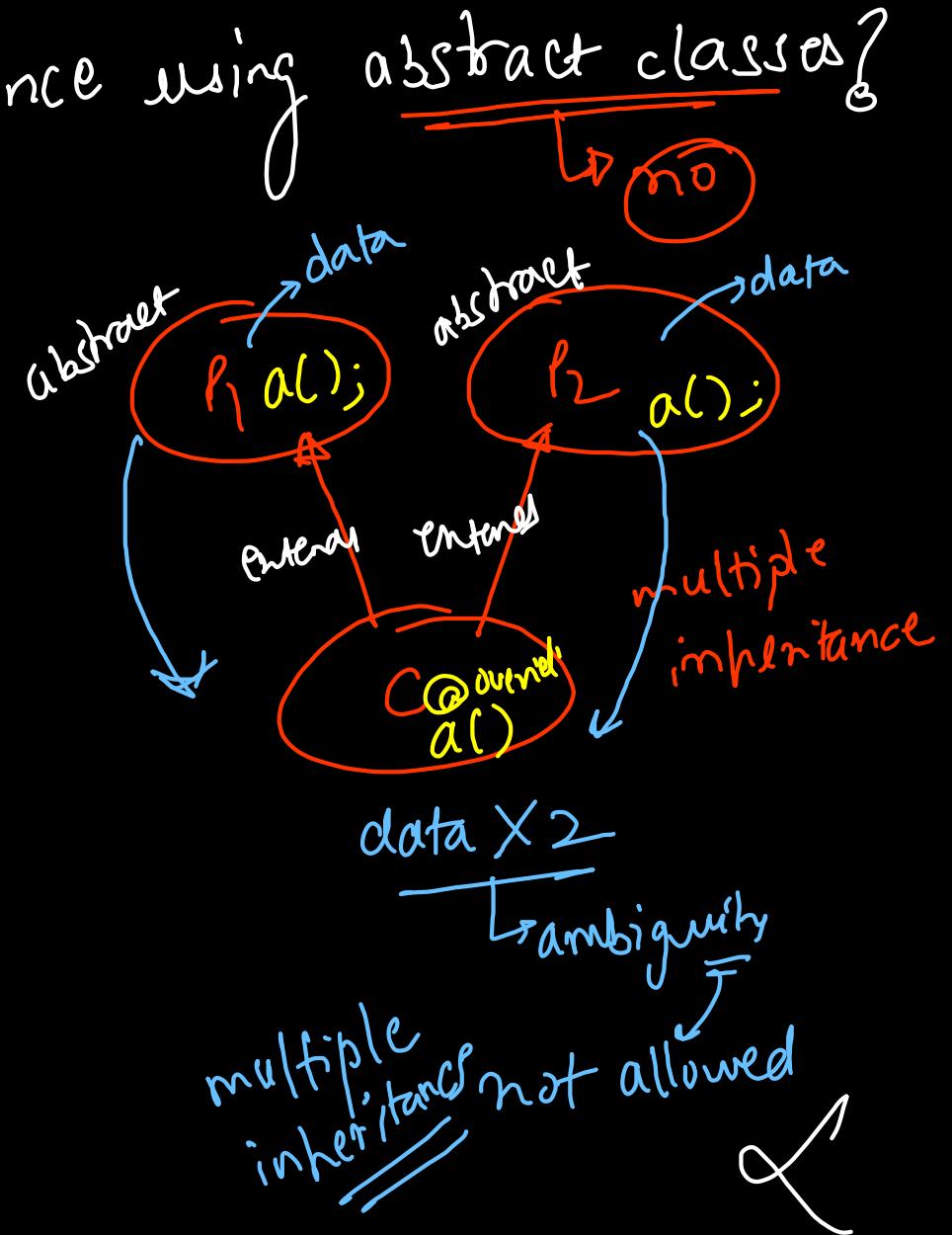
single ✓



hierarchical ✓



multilevel ✓



Q) What is the difference between encapsulation, data hiding and data abstraction?

Encapsulation:→ Wrapping together properties & behavior in a single entity known as class -

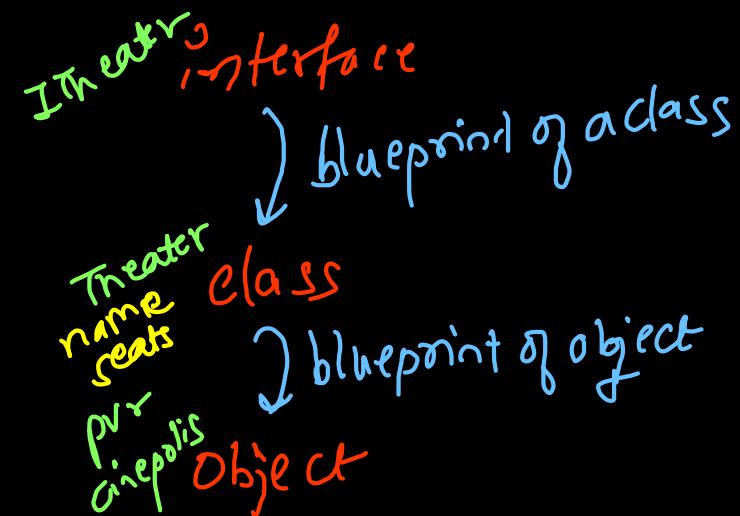
Data Hiding:→ Properties / Behaviors → access modifiers  
public ↓ private  
default/protected

Abstraction:→ Hiding the unnecessary details from the client  
interface  
abstract class

| Abstraction                                                                                                                                                                    | Data Hiding                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>It is the process of hiding the internal implementation and keeping the complicated procedures hidden from the user. Only the required services or parts are displayed.</p> | <p>It is the process that ensures exclusive data access to class members and hiding the internal data from outsiders.</p>                                          |
| <p>Focuses on hiding the complexity of the system.</p>                                                                                                                         | <p>Focuses on protecting the data by achieving encapsulation (a mechanism to wrap up variables and methods together as a single unit).</p>                         |
| <p>This is usually achieved using <u>abstract</u> class concept, or by implementing <u>interfaces</u>.</p>                                                                     | <p>This can be achieved using access specifiers, such as <u>private</u>, and <u>protected</u>.</p>                                                                 |
| <p>It helps to secure the software as it hides the internal implementation and exposes only required functions.</p>                                                            | <p>This acts as a security layer. It keeps the data and code safe from external inheritance as we use setter and getter methods to set and get the data in it.</p> |
| <p>It doesn't affect end users, since the developers can perform changes internally in implementation classes without changing the abstract method in the interfaces.</p>      | <p>This ensures that users can't access internal data without authentication.</p>                                                                                  |
| <p>It can be implemented by creating a class that only represents important attributes without including background detail.</p>                                                | <p>Getters and setters can be used to access the data or to modify it.</p>                                                                                         |

Q) What is an interface in Java? Give some real world coding example.

- ↳ way to achieve abstraction in Java.
- ↳ blueprint of a class
- ↳ interface cannot be instantiated



Q) What is the default state of variables in an interface?

- ↳ public, static, final
- \* class property
- \* constants

↳ instance variables

Q) What is the default state of methods in an interface?

- ↳ public, abstract (100% by default)

```
interface ITheater {
 String industry = "Bollywood";
 // public, static, final

 // 100% abstraction

 void viewShow(); // public, abstract

 String bookShow();
}

class Theater implements ITheater {
 String name;

 public void viewShow() {
 System.out.println("Only View Access");
 }

 public String bookShow() {
 System.out.println("Book Access");
 return "ticket";
 }
}
```

```
class BookMyShow {
 @SuppressWarnings("all")
 Run | Debug
 public static void main(String[] args) {
 Theater pvr = new Theater();
 pvr.name = "PVR Cinemas";

 pvr.viewShow();

 System.out.println(ITheater.industry);
 System.out.println(Theater.industry);

 Theater cinepolis = new Theater();
 cinepolis.name = "Cinepolis Experience";

 cinepolis.bookShow();

 System.out.println(pvr.industry);
 System.out.println(cinepolis.industry);
 }
}
```

```
"Only View Access
Bollywood
Bollywood
Book Access
Bollywood
Bollywood
```

Q) Can there be concrete methods in interface?

by default, you cannot have concrete methods in Java ( version < 8)

~~conception~~

↳ private method, static method, default method  
↳ concrete      ↳ concrete  
                      ↓ default implementation

Q) Can an interface be related to another interface. If yes, how?

↳ yes : one interface extends  
another interface,

```
interface MyInterface {
 // Default Method: Object's Method
 default void defaultFun() {
 System.out.println(x: "Default fun: have a body");
 privateFun();
 }

 // private method: Within Interface
 private void privateFun() {
 System.out.println(x: "Private fun: have a body");
 }

 // Interface's Method
 public static void staticFun() {
 System.out.println(x: "Static fun: have a body");
 }
}

class MyClass implements MyInterface {
}

class Driver {
 Run | Debug
 public static void main(String[] args) {
 MyInterface.staticFun();

 MyClass obj = new MyClass();
 obj.defaultFun();
 }
}
```

Static fun: have a body  
Default fun: have a body  
Private fun: have a body

```

interface Radio {
 void playRadio();
}

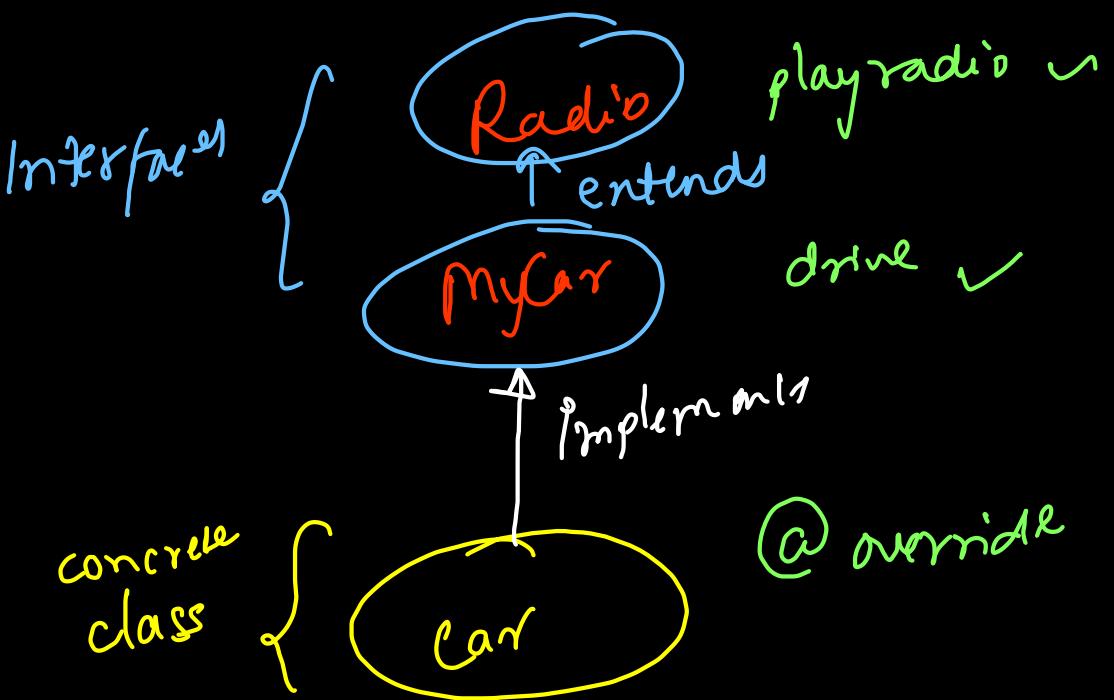
interface MyCar extends Radio {
 void drive();
}

class Car implements MyCar {
 public void playRadio() {
 System.out.println("Radio Starts");
 }

 public void drive() {
 System.out.println("Car Starts");
 }
}

class Driver {
 Run | Debug
 public static void main(String[] args) {
 Car i10 = new Car();
 i10.drive();
 i10.playRadio();
 }
}

```

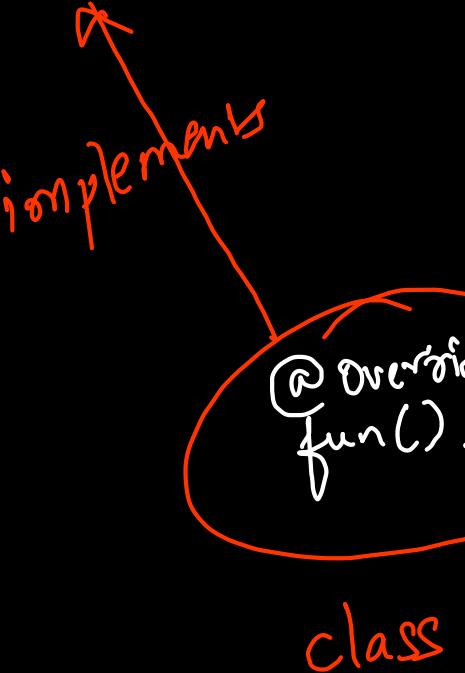
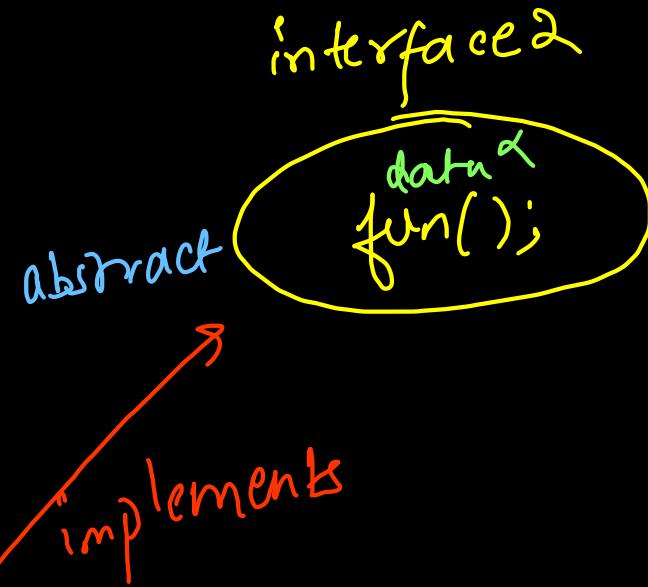


Q) Is there (a) this (b) constructor defined in an interface?  
There is no object & interface  $\Rightarrow$  there is no this keyword  
no  
no  
constructor (initialization) ← instance variables ← variables ← static & final ← 100% abstraction

→ interfaces do not take part in class hierarchy  
super() → child ← Object ✓  
Interface ✗

Q) Can class implement more than 1 interfaces? Will it not cause diamond problem?  
↳ Class can implement 1 or more than 1 interfaces!  
{ multiples interfaces can be implemented by a single class }

- ① no instance variable ambiguity 2
- ② diamond problem



over-riding only once  
→ diamond problem

You, 1 second ago | 1 author (You)

```
interface Radio2 {
 void start();
}
```

You, 1 second ago | 1 author (You)

```
interface MyCar2 {
 void start();
}
```

You, 1 second ago | 1 author (You)

```
class Car2 implements Radio2, MyCar2 {
 @Override → Only once
 public void start() {
 System.out.println("Radio & Car
 Starts Together");
 }
}
```

abstract  
ambiguity 2

→ No diamond  
problem

Car2 i20 = new Car2();  
i20.start();

→ output

```
You, 1 second ago | 1 author (You)
interface Radio2 {
 String data = "Radio"; // static

 void start();
}
```

```
You, 1 second ago | 1 author (You)
interface MyCar2 {
 String data = "Car"; // static

 void start();
}
```

```
You, 1 second ago | 1 author (You)
class Car2 implements Radio2, MyCar2 {
 @Override
 public void start() {
 System.out.println("Radio & Car
Starts Together");
 }

 public void fun() {
 System.out.println(Radio2.data);
 System.out.println(MyCar2.data);
 }
}
```

static property  
↳ Interface name  
↳ ambiguity

\* Q) What is the difference between class & interface?

### Class

- ① blueprint of an object
- ② achieve encapsulation
- ③ it can be instantiated
- ④ constructors, this, super ✓
- ⑤ properties → static/nonstatic  
nonfinal/final
- ⑥ methods → abstract/concrete

### Interface

- ① blueprint of a class
- ② achieve 100% abstraction
- ③ it cannot be instantiated
- ④ constructor, this, super &
- ⑤ properties → public static final
- ⑥ all methods are abstract & public

| Class                                                                                                            | Interface                                                                                                                             |
|------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| The keyword used to create a class is "class"                                                                    | The keyword used to create an interface is "interface"                                                                                |
| A class can be instantiated i.e, objects of a class can be created.                                              | An Interface cannot be instantiated i.e, objects cannot be created.                                                                   |
| Classes does not support multiple inheritance.                                                                   | Interface supports multiple inheritance.                                                                                              |
| It can be inherit another class.                                                                                 | It cannot inherit a class.                                                                                                            |
| It can be inherited by another class using the keyword 'extends'.                                                | It can be inherited by a class by using the keyword 'implements' and it can be inherited by an interface using the keyword 'extends'. |
| It can contain constructors.                                                                                     | It cannot contain constructors.                                                                                                       |
| It cannot contain abstract methods.                                                                              | It contains abstract methods only.                                                                                                    |
| Variables and methods in a class can be declared using any access specifier(public, private, default, protected) | All variables and methods in a interface are declared as public.                                                                      |
| Variables in a class can be static, final or neither.                                                            | All variables are static and final.                                                                                                   |

\* Q) What are the differences between abstract classes & interfaces?

Similarities → to achieve abstraction, cannot instantiate

### Abstract class

- ① 0-100% abstraction
- ② by default → concrete method
- ③ variables
  - instance variable
  - final / static variable
- ④ multiple inheritance (✗)
- ⑤ access modifier → public / default / protected / private

### Interface

- ① 100% abstraction
- ② by default → abstract method
- ③ variables → final / static
- ④ class can implement multiple interfaces
- ⑤ access modifier → public

| <b>Abstract class</b>                                                                             | <b>Interface</b>                                                                                                   |
|---------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| 1) Abstract class can <b>have abstract and non-abstract methods.</b>                              | Interface can have <b>only abstract</b> methods. Since Java 8, it can have <b>default and static methods</b> also. |
| 2) Abstract class <b>doesn't support multiple inheritance.</b>                                    | Interface <b>supports multiple inheritance.</b>                                                                    |
| 3) Abstract class <b>can have final, non-final, static and non-static variables.</b>              | Interface has <b>only static and final variables.</b>                                                              |
| 4) Abstract class <b>can provide the implementation of interface.</b>                             | Interface <b>can't provide the implementation of abstract class.</b>                                               |
| 5) The <b>abstract keyword</b> is used to declare abstract class.                                 | The <b>interface keyword</b> is used to declare interface.                                                         |
| 6) An <b>abstract class</b> can extend another Java class and implement multiple Java interfaces. | An <b>interface</b> can extend another Java interface only.                                                        |
| 7) An <b>abstract class</b> can be extended using keyword "extends".                              | An <b>interface</b> can be implemented using keyword "implements".                                                 |
| 8) A Java <b>abstract class</b> can have class members like private, protected, etc.              | Members of a Java interface are public by default.                                                                 |