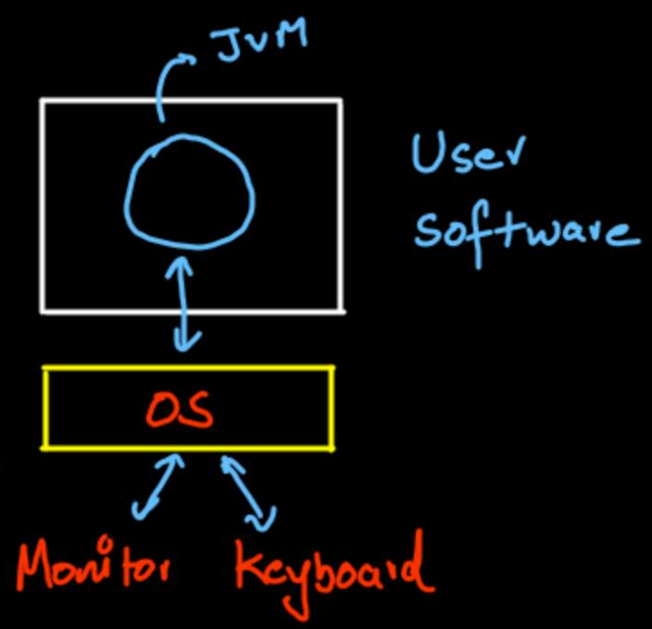
Object Oriented Programming in Java

Java is platform independent because bytecode is not m/c code.



JDK → JVM → JRE (Java Run Time Env.)

Java Devlopment Tool Kit

Java Virtual Machine

Java Code

↓

Compiler (javac)
JVM

(.class)
bytecode

↓

Run(JRE)
machine Lang

Pillars of OOPS

- Encapsulation
- Abstration
- Inheritance
- Polymorphism

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

```java
class Student {
    int marks;
    String name;
    int rollno;
}
```

Class : Group of data members & member functions

```java
Student guneet = new Student();
guneet.name = "Guneet";
guneet.marks = 10;
guneet.rollno = 1;

Student yash = new Student();
yash.name = "Yash";
yash.marks = 20;
yash.rollno = 2;

        System.out.println("Guneet's Details");
        System.out.println("Name: " +  guneet.name + " Marks: " + guneet.marks + " Roll No: " + guneet.rollno);
        System.out.println("Yash's Details");
        System.out.println("Name: " +  yash.name + " Marks: " + yash.marks + " Roll No: " + yash.rollno);
    }
```

Object = Instance

data member = instance variable

func vs Methods : Methods are nothing but functions inside a class. Since in Java, everything is inside a class hence Java funcs are always called methods.

```java
class Student {
    int marks;
    String name;
    int rollno;

    void Study(String subject) {
        System.out.println(name + " studies " + subject);
    }
}
```

```java
    guneet.Study("DSA");
    yash.Study("nothing");
```

Output →

```
Finished in 89 ms
Guneet studies DSA
Yash studies nothing
```

data members → properties

member functions → behavior

Class → Blue print

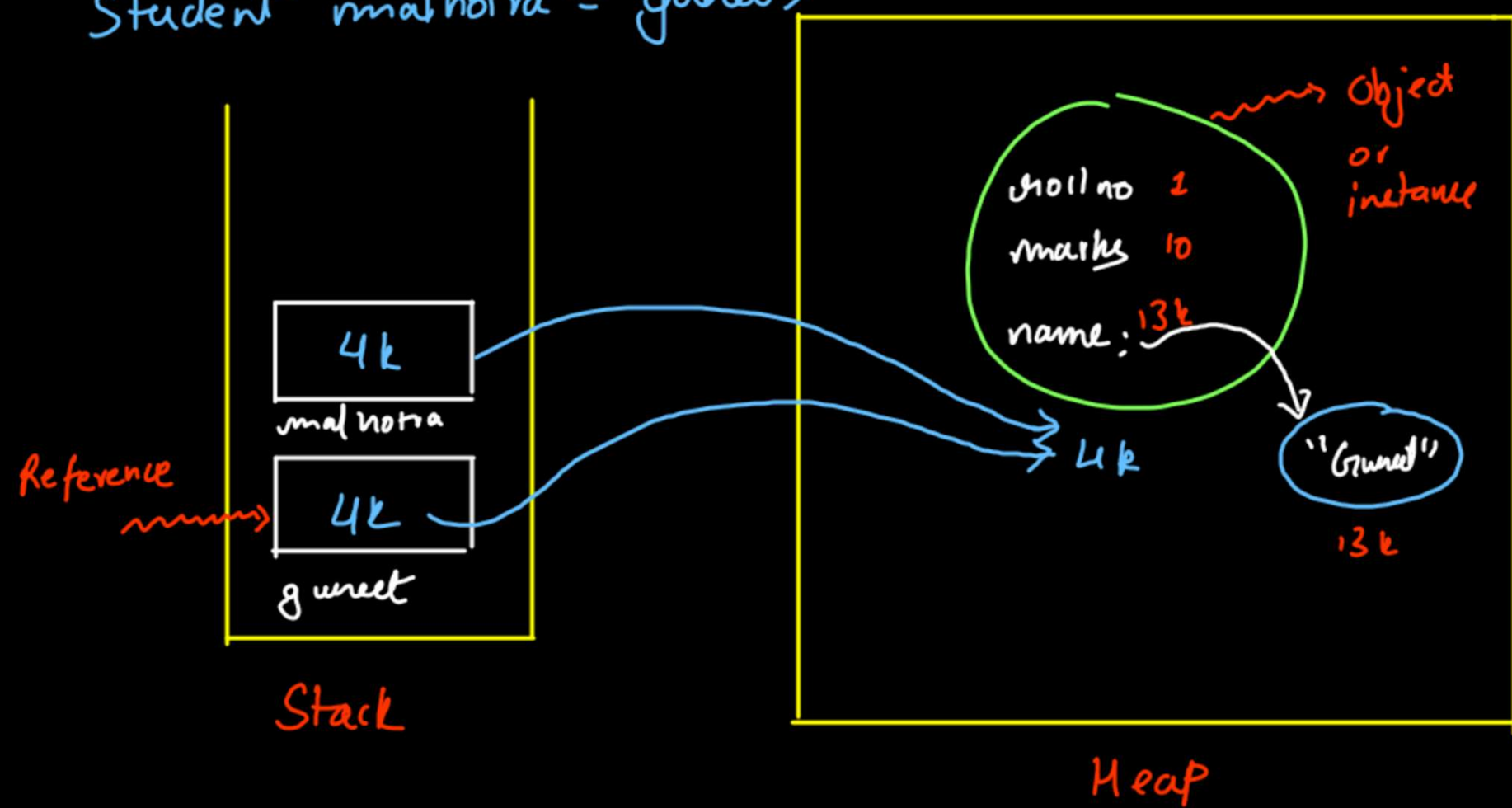
Objects → Physical Entity (Occupies Memory)

Java is not purely Object Oriented Programming Language because of the presence of primitive data types.

Java has still allowed use of primitive data types because of the speed issues that occur when memory is allocated in heap.

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

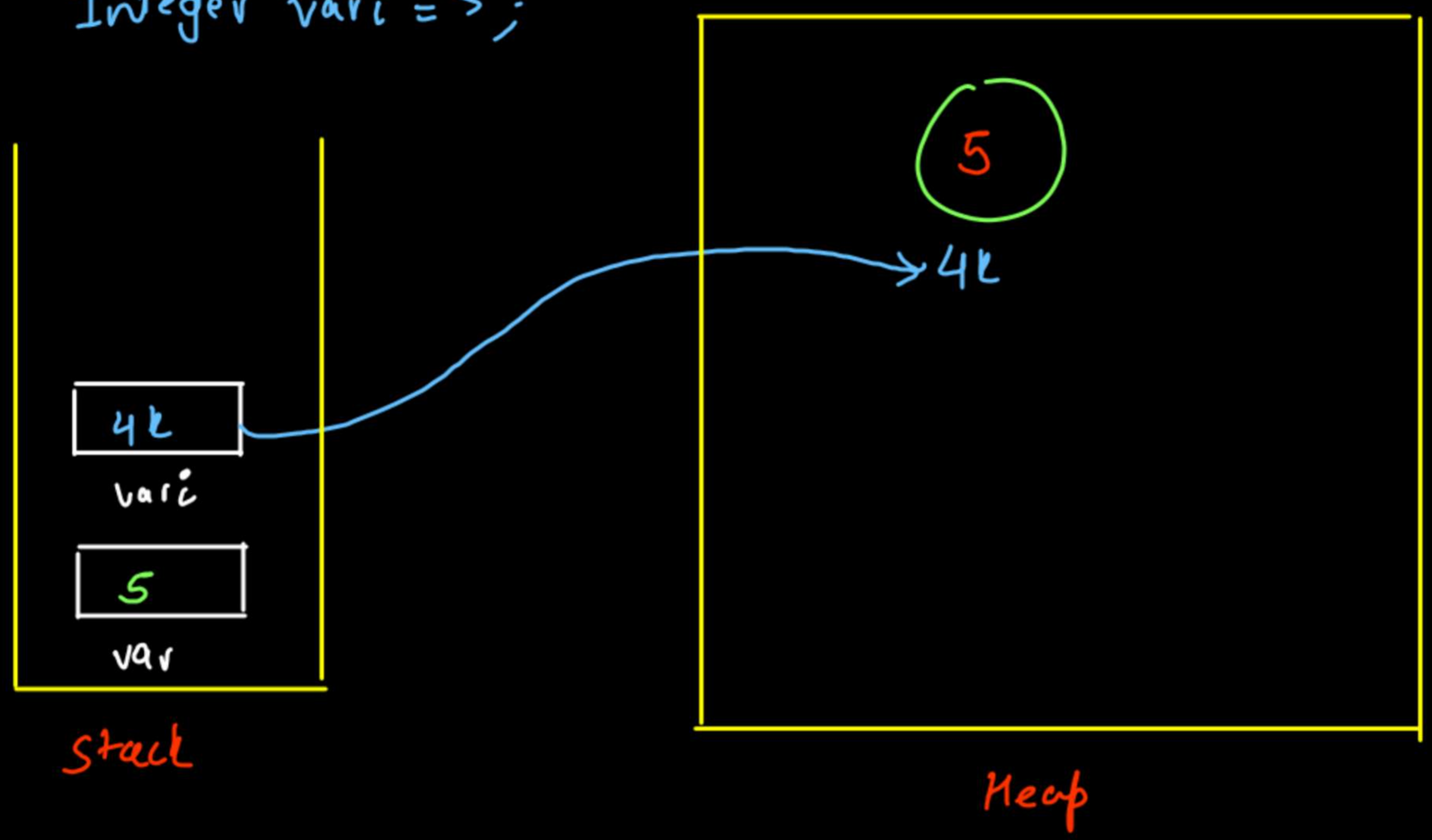
```
Student gunest = new Student();
Student malhotra = gunest;
```



Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

int var = 5;

Integer vari = 5;



```java
class Main {
    public static void main(String args[]) {
        // Your code goes here
        Integer i = 10; //autoboxing
        int x = i; //unboxing
        System.out.println(x);
    }
}
```

primitive to object automatically

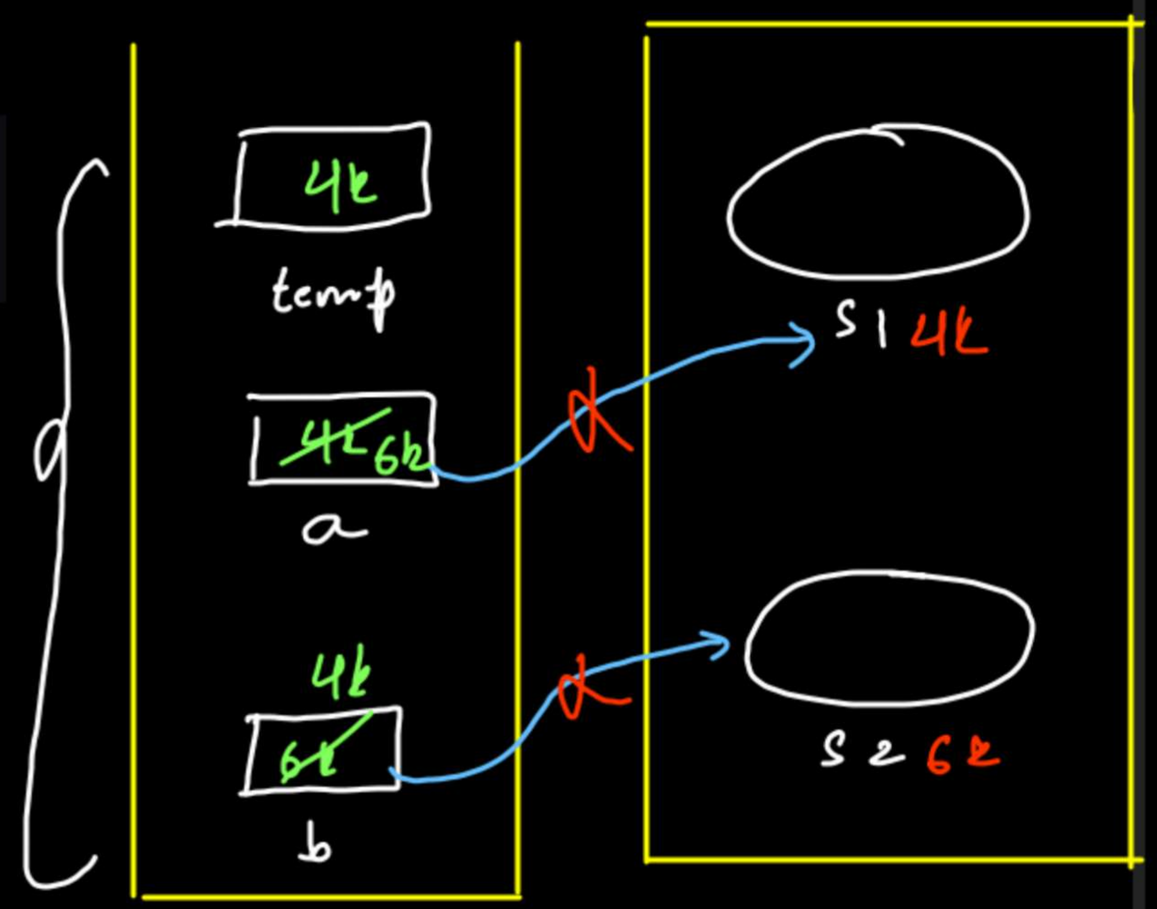
object to primitive automatically

Swap Game 1

```java
public static void swap1(Student a, Student b){
    Student temp = a;
    a = b;
    b = temp;
}
```

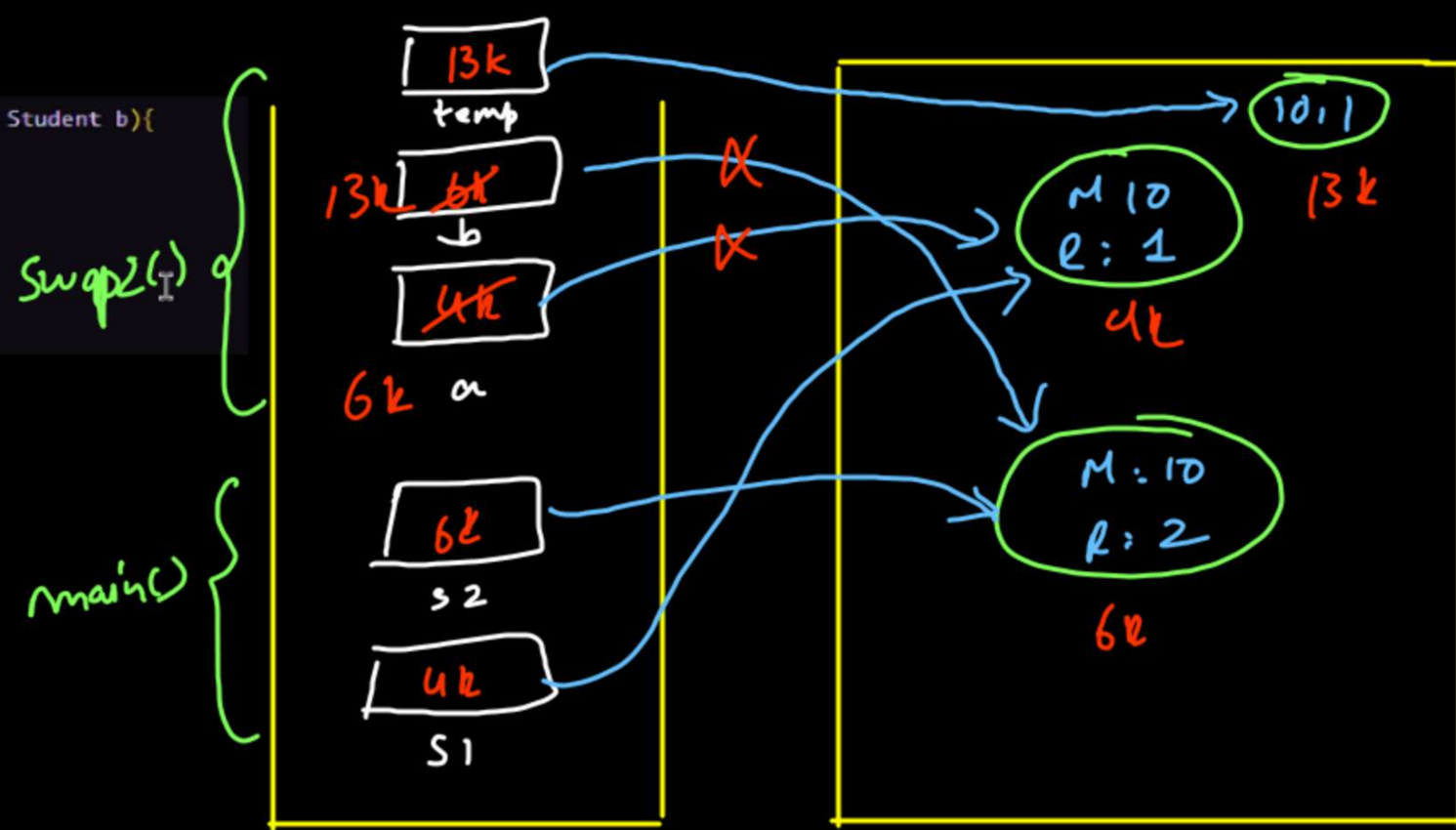
Swap happened in references & now when swap1() function gets deleted from stack, references were locally changed hence no swap occurs.

swap1()



Swap Game 2

```java
public static void swap2(Student a, Student b){
    Student temp = new Student();
    temp.marks = a.marks;
    temp.rollNo = a.rollNo;

    a = b;
    b = temp;
}
```

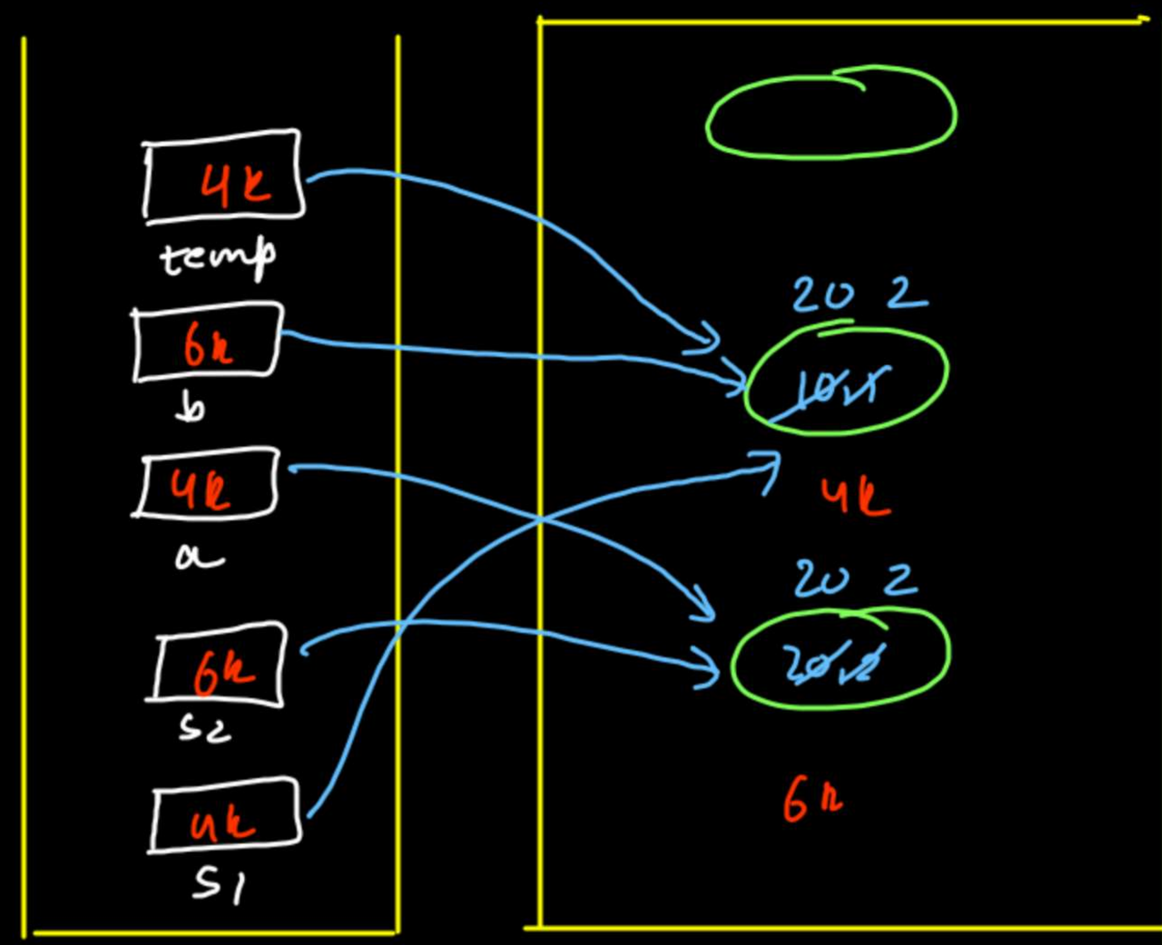


Again, the changes will not persist in main()

Swap Game 3

```java
public static void swap3(Student a, Student b){
    Student temp = a;

    a.marks = b.marks
    a.rollNo = b.rollNo;


    b.marks = temp.marks;
    b.rollNo = temp.rollNo;
}
```

Now, the changes persist because changes were made to heap.



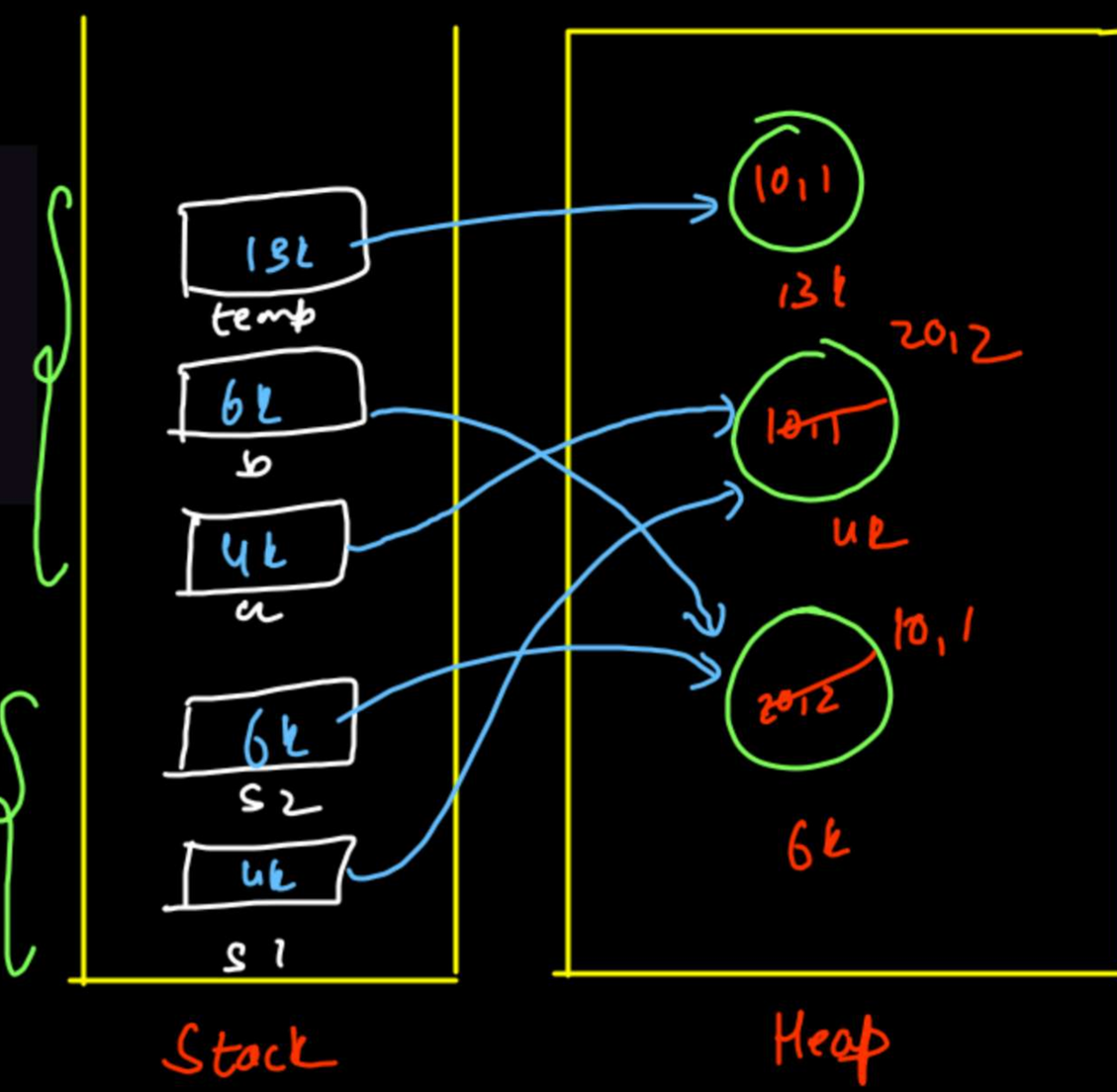
Swap Game 4

```
// public static void swap4(Student a, Student b){
//     Student temp = new Student();
//     temp.marks = a.marks;
//     temp.rollNo = a.rollNo;

//     a.marks = b.marks;
//     a.rollNo = b.rollNo;

//     b.marks = temp.marks;
//     b.rollNo = temp.rollNo;
// }
```

swap4()

Now, the swap has occured and changes will persist because they occur in heap.

main()



Constructor :

```
// Student(){
//     marks = 100;
//     name = "Pepcoder";
//     rollNo = -1;

//     System.out.println(" Hey a new object of student class is created ");
// }
```

Explicit Default Constructor

// Implicit default constructor

Student() {

}

① Name as same of Class

② Returns this (we cannot see the return type)

When we write any parameterized constructor, Java removes the default constructor. So, we have to write default const.

when we create a param. const. if we are using it.

Constructor Overloading

student () {

constructor overloading due to diff no of parameters -

```
Student(int newMarks, String newName, int newRollNo){
    marks = newMarks;
    name = newName;
    rollNo = newRollNo;
}

Student(int newMarks, int newRollNo, String newName){
    marks = newMarks;
    name = newName;
    rollNo = newRollNo;
}
```

due to change in order of arguments.

Copy Constructor

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

deep copy because if new object of stringBuilder is not created, stringBuilder due to being immutable will have references copied & changing one objects' sb will change others' sb too.

Also, Java removes copy constructor too if a param const is created like it removes the default const.

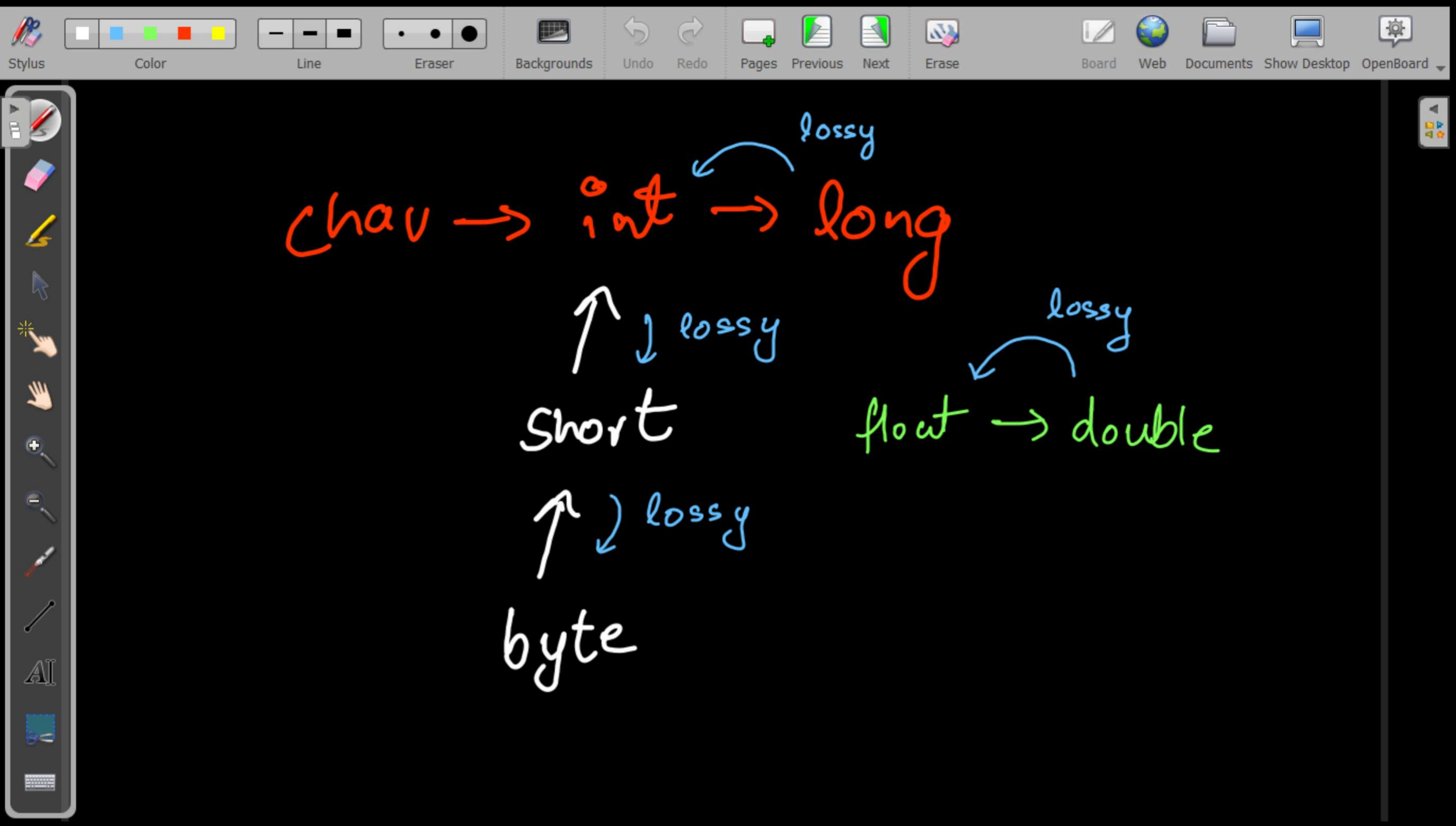
if we put a long or double value as a param in place of int, we will get an error saying no such constructor exists.

```
Student(int newMarks, String newName, int newRollNo){
    marks = newMarks;
    name = newName;
    rollNo = newRollNo;
}
```

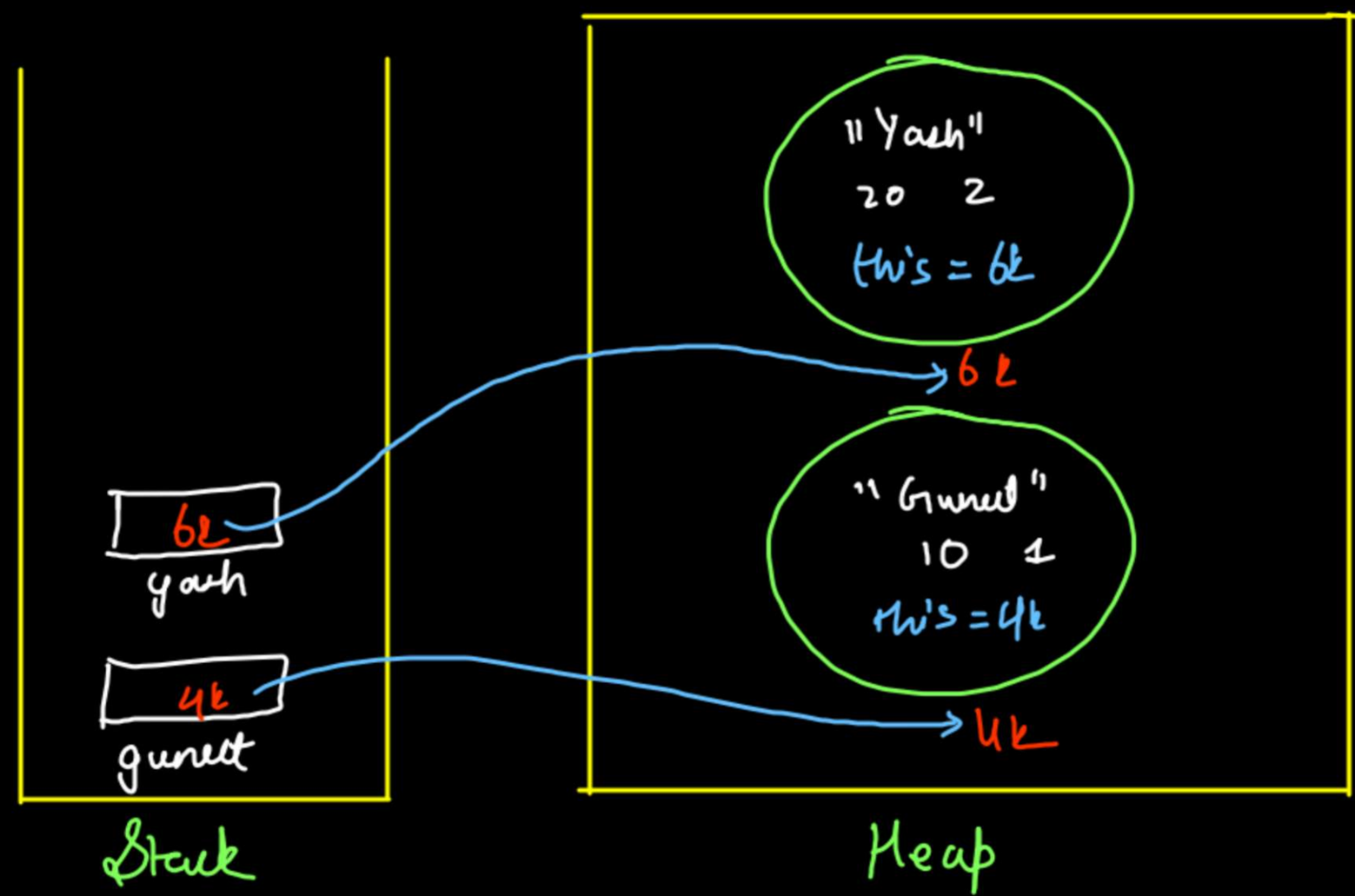
Upcasting ✓

Downcasting ✗

This happens because there will be automatic type conversion from Higher Ranged data types (Long or Double) to lower ranged datatype (int) but not vice-versa.


Stylus
Color
Line
Eraser
Backgrounds
Undo
Redo
Pages
Previous
Next
Erase
Board
Web
Documents
Show Desktop
OpenBoard
char → int → long
lossy
lossy
short
lossy
float → double
lossy
byte

this Keyword (self referential pointer)



```java
Student (String name, int rollno, int marks) {
    this.marks = marks;
    this.rollno = rollno;
    this.name = name;
}
```

this prevents namespace collision.

data member marks local variable marks

⇒ data member marks me local variable daal diya.

this is used in Constructor Chaining

Thi calls keyword is used to

```
public Cuboid() {
    // length = breadth = height = 1;
    this(1);
}

public Cuboid(int length, int breadth, int height) {
    this.length = length;
    this.breadth = breadth;
    this.height = height;
}

public Cuboid(int side) {
    // length = breadth = height = side;
    this(side,side,side);
}

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

→ chaining

Syso(l+b+h);

Printing will also be done in every constructor coz of chaining.

→ chaining

→ chaining

main() {

Cuboid c = new Cuboid();

}

Cuboid() → Cuboid(side) } Chaining

Cuboid(l,b,h)

Overhead is more but code redundancy is reduced

```java
// Explicit Default Constructor
public Cuboid(){
    System.out.println(" Hi I am before Constructor Chaining ");
    this(1);
    this(1, 1, 1);
}
```

⇒ This gives compile time error - Chaining should be the first statement.

This is also not possible.

Also, this is also passed as a parameter implicitly.

this can be used to call methods / members of same class

```
public int area() {
    this.length*this.breadth;
}

public int volume() {
    return (this.area()) * this.height; //use of this reduces overhead
}
```

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

this can also be returned from a method.

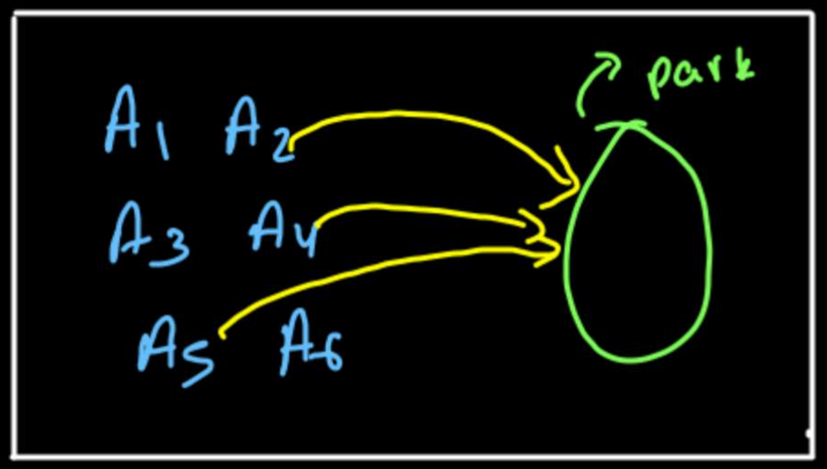
```
public Cuboid join(Cuboid other) {
    this.length += other.length;
    this.breadth += other.breadth;
    this.height += other.height;

    return this;
}
```

```
main()
{
    Cuboid c1 = new Cuboid();
    Cuboid c2 = new Cuboid(1,2,3);
    Cuboid c3 = c1.join(c2);
}
```

Changes are made to c1 & also stored in c3.

Static Key Word

If we want to keep a property in a class which is not associated to any object, static keyword can be used.



All objects (Apartments) can access the static property (park) but it is not the property of any apart. (object) rather the society (class).

```java
class Student {
    int marks;
    String name;
    int rollno;
    static String University  = "IPU";
```

property of Student

static bana nai k baad define karna za ru ri hota hai

```java
System.out.println(guneet.University);
System.out.println(yash.University);
```

```
Finished in 50 ms
IPU
IPU
```

To make property as immutable, we make it final.

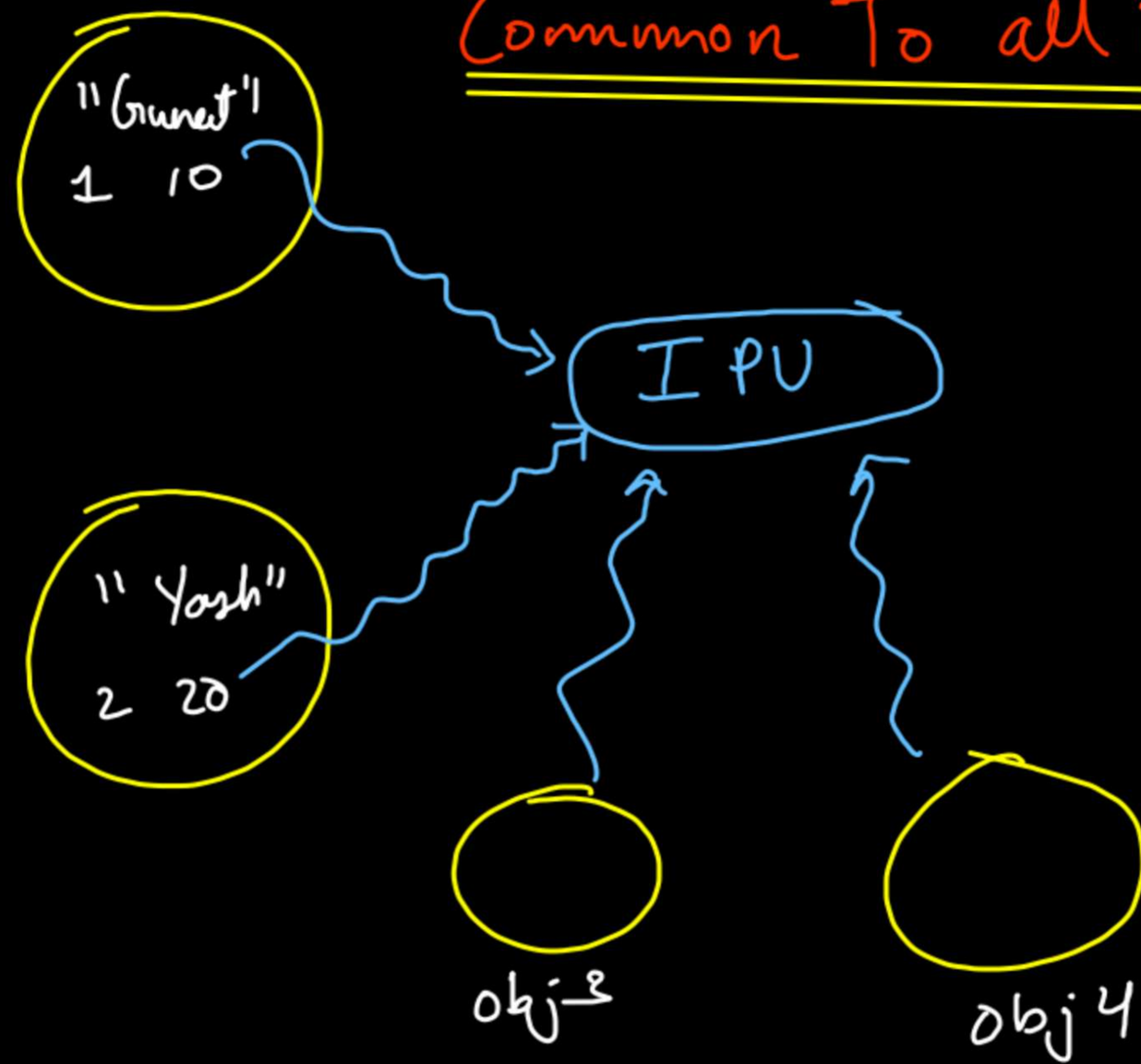
We could access static property without creating object of the class like : Student.University

```java
static final String University  = "IPU";
```

⇒ This cannot be changed now anywhere in the code.

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

Common To all the Objects



Accessing a static property using Class

Static members

```
+ " " + Student.university);
```

Global variables do not exist in Java because everything is inside a class only.

Syntax:

ClassName . property

Accessing static members using ClassName is a good practice because they are the properties of class only.

Static Methods

```java
static void study() {
    System.out.println("Student studies in " + University);
}
```

```java
System.out.println(guneet.University);
System.out.println(yash.University);

Student.study();
```

output

```
Finished in 71 ms
IPU
IPU
Student studies in IPU
```

Why is main() static?

Running a file (Java) in terminal

java fileName

We do not create any object of the class in which we have main

Hence, main() must be static so that we need not create an object of the Main class.

Do static methods have this?

No, because static methods are methods of the class not any object.

{Error: non-static member this cannot be accessed from a static context}

Stylus | Color | Line | Eraser | Backgrounds | Undo | Redo | Pages | Previous | Next | Erase | Board | Web | Documents | Show Desktop | OpenBoard

#

- Static member (class) calling static member → Yes
- Non-static member calling static member → Yes
- Non-static member calling non-static member → Yes
- static member calling non-static member {using hasA}

↓

No (because class ki property object ki property ko use nahi kar paegi because of absence of this)

Think of it using park & apartment

example - Apart me rehne vala park ko access kar sakta hai lekin park me ghoomne vala kisi k apart ko nahi.

Nested Class

```
public class Main {

    static class Inner {
        //without making object of outer class
        //we can make objects of inner class
    }
    static void student(){

    S.out("hello")

    }
}
```

Main.inner

Main.student()

Here without creating an object of the main class we are able to access the Inner class and also the student method as well

We are able to make an object of inner class without making an object of outer class could be done by making the inner class as static and the outer class with no changes

Encapsulation

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

Implement

Data Hiding

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

Getters & Setters

Encapsulation:

we can implement data hiding.

getters() & setters() are used so that authorised users can access them. (because we make data members private in most of the classes)

*s getters & setters are also called manipulators.

→ private methods / members are accessible only inside a class

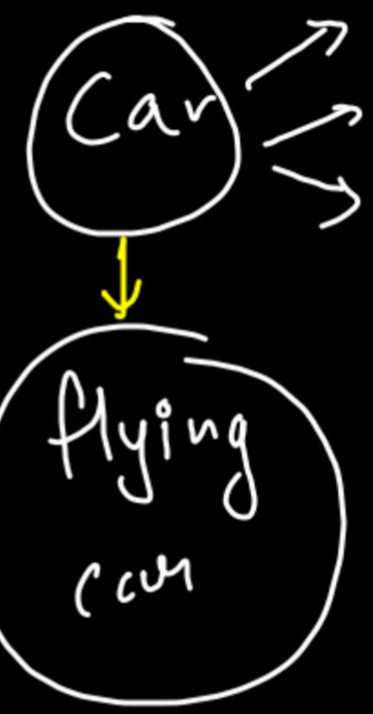
→ public methods can be accessed from anywhere.

Final keyword help us to make values immutable but we need to initialize the variable at the very same time in order to use the final keyword

Inheritance

Passing on properties & methods of Parent (Super) class to child (Sub) class is called inheritance.

- Single
- Multilevel
- Heirarchical




```
class flyingCar extends Car{
}
```

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

```java
class Car {
    int gear;
    int wheels;
    String engine;
}

class FlyingCar extends Car {
    String wings;

}
```

```java
public class Main {

    public static void main(String[] args) {
        FlyingCar obj = new FlyingCar();
        obj.gear = 5;
        obj.wheels = 4;
        obj.engine = "electric";
        obj.wings ="metallic";
        System.out.println(obj.gear + " " + obj.wheels + " " + obj.engine + " " +
obj.wings);



    }
}
```

Also, it is not possible to access parent class objects, properties from a child classes' object

That is "car.wings" is not possible

Multilevel Inheritance

```
class Car {
    int gear;
    int wheels;
    String engine;
}

class FlyingCar extends Car {
    String wings;
}

class FlyingHydroCar extends FlyingCar {
    String pedal;
}
```

Car
↓
flying Car
↓
flying Hydro Car

Heirarchical Inheritance

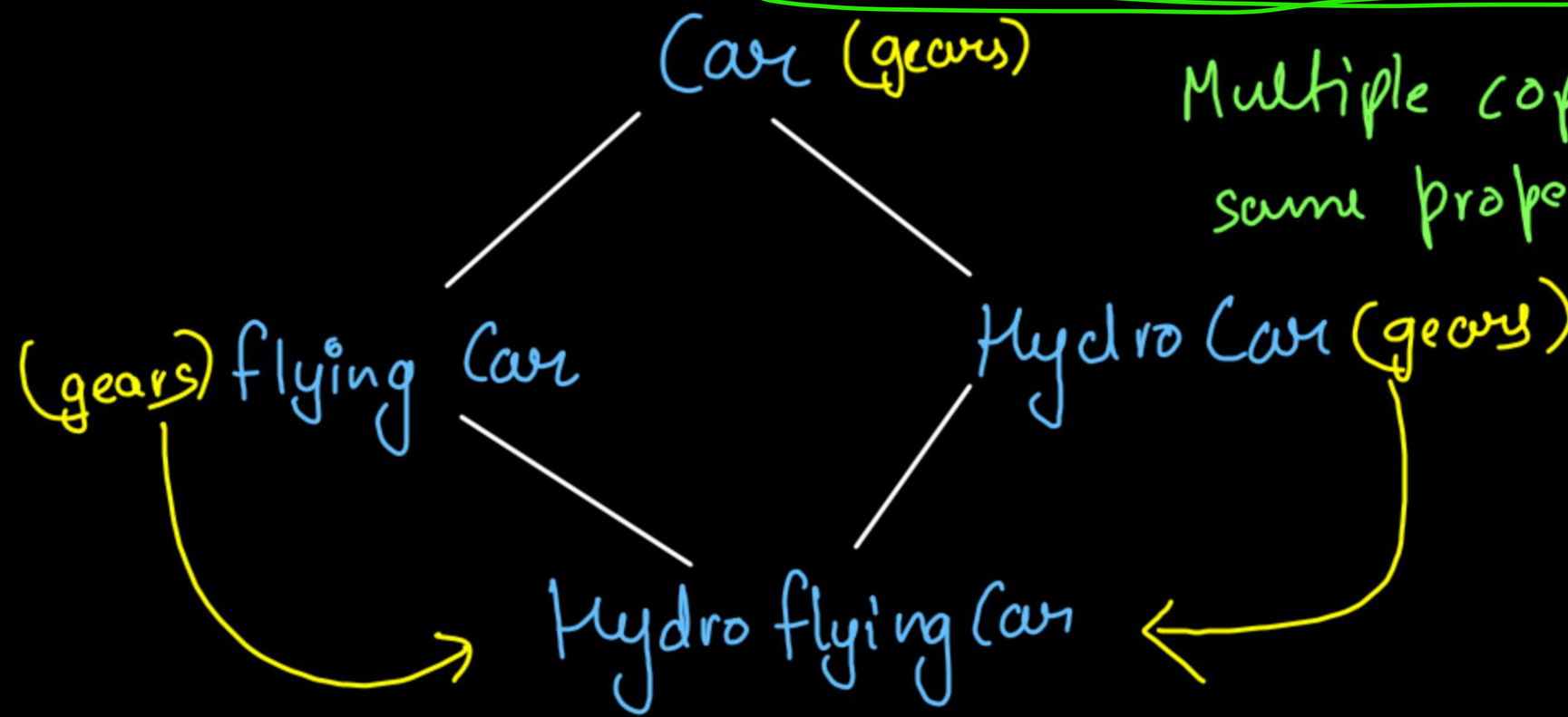
```java
class Car {
    int gear;
    int wheels;
    String engine;
}

class FlyingCar extends Car {
    String wings;
}

class FlyingHydroCar extends Car {
    String pedal;
}
```

Car

FlyingCar HydroCar

Multiple Inheritance is not possible in Java.

The diamond Problem {Deadly diamond of death problem}



Multiple copies of same properties or methods.

Object Class in Java

Super Class of Every Class in Java.

```java
class Car {
    int gear;
    int wheels;
    String engine;
}

class FlyingCar extends Car {
    String wings;

}
```

Object Class

car → flyingCar

This is again diamond problem.

Actually this does not happen.

If a class extends our custom class, it does not extends Object class.

Every class in JavaScript has an access to the Object class and hence object class is a super class to every class

also parent class is also called the super class and child class is also called the sub class

```
class Car {
    int gear;
    int wheels;
    String engine;
}

class FlyingCar extends Car {
    String wings;

}
```

Object Class

↓

Car

↓

flying Car

This now becomes multi level inheritance

Recording
Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard
You are viewing Archit Aggarwal's screen
View Options

Let us see which all members of Java can be assigned with the access modifiers:

Members of JAVA	Private	Default	Protected	Public
Class	No	Yes	No	Yes
Variable	Yes	Yes	Yes	Yes
Method	Yes	Yes	Yes	Yes
Constructor	Yes	Yes	Yes	Yes
interface	No	Yes	No	Yes

Access Modifiers Table/Chart

Most Restrictive ← → Least Restrictive

Access Modifiers	private	Default/no-access	protected	public
Inside class	Y	Y	Y	Y
Same Package Class	N	Y	Y	Y
Same Package Sub-Class	N	Y	Y	Y
Other Package Class	N	N	N	Y
Other Package Sub-Class	N	N	Y	Y

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

Inheritance vs Association

OOPS 2 || 2.45.00

is A relationship
{Inheritance}

VS

has A relationship
{Association}

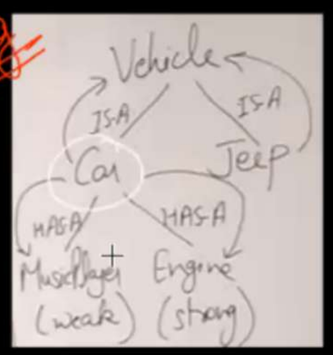
→ Tightly Coupled Relationship → Direct access through super keyword

→ Loosely coupled Relationship → Accessing using object

Example → Nested class
→ Object Member → LL has a Node

```
LL {
Node head;
```

Eg



Types of Association
- Aggregation (Weak)
- Composition (Strong)

```
class MusicPlayer{
  double radio ;
}
```

```
class Car{
    int gear;
    int wheels;
    String engine;
    MusicPlayer player;
}

class FlyingCar extends Car{
    String wings;
}
```

Car has-a music player

flying Car is a Car

Direct access

Inheritance is tightly coupled relationship → flyingCarobj.gear

Association is a loosely coupled relationship → Carobj.player.radio

Indirect access

Types of Association

Aggregation
(Weak)

2 classes are in a relationship & they can exist without one another.

Composition
(Strong)

2 classes are in a relationship & one class cannot exist without other.

Super KeyWord

```java
class Car {
    int gear;
    String engine;
}

class FlyingCar extends Car {
    String wings;
}
```

```java
public static void main(String[] args) {
    FlyingCar obj = new FlyingCar();
}
```

Base class constructor is always executed first before the derived class.

① FlyingCar () constructor is called.

② Car() constructor is executed

③ flyingCar() constructor is executed

```java
class Car{
    int gear;
    String engine;

    Car(){
        System.out.println("Car Constructor Execution")
    }
}

class FlyingCar extends Car{
    String wings;

    FlyingCar(){
        System.out.println("Flying Car Constructor Execution");
    }
}
```

Output : Car Constructor Execution

Flying Car Constructor Execution

① super keyword is used to call the constructor of Super Class.

(Immediate Super Class not the absolute)

{ first search in immediate parent then upwards & so on }

```
FlyingCar(){
    super(5, "CNG");
    System.out.println("Flying Car Constructor Execution");

}
```

→ Call to super() must be the first line in execution of child class constructor.

** This means that super() will be the first statement by default in Java even if we do not write it.

this(—);
super(—);

} even this() cannot be called before super. foundation should be laid first.

```java
FlyingCar(int gears, String engine, String wings){
    super(gears, engine);
    this.wings = wings;
}
```

We can set the data members of super class using super() keyword.

```java
void display(){
    System.out.println(this.wings + " " + super.gear + " " + super.engine);
}
```

Overhead will be reduced a little bit.
{ Same logic as in this }

Parameters of super class can also be accessed using super.

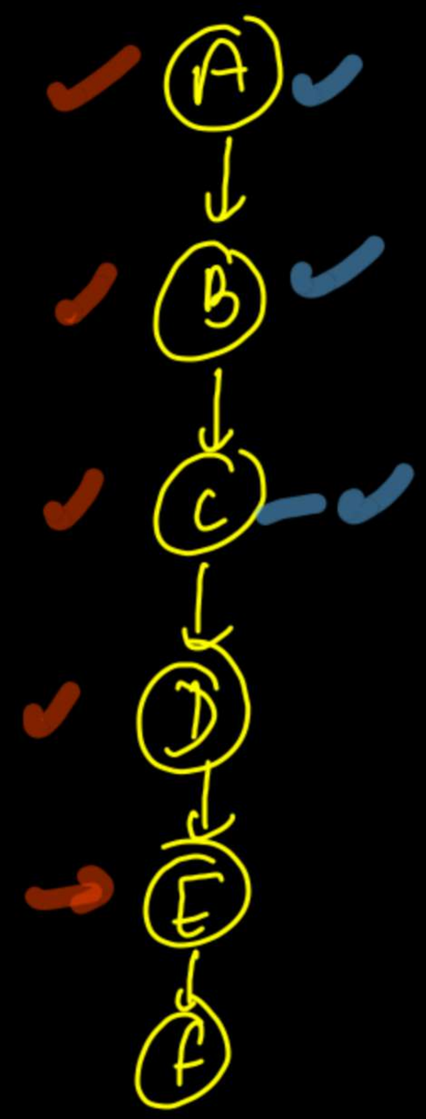
```java
class FlyingCar extends Car{
    String wings;
    int gear;

    FlyingCar(){
        super(5, "Petrol");
    }

    FlyingCar(String wings){
        super(5, "Petrol");
        this.wings = wings;
    }

    FlyingCar(int gear, String engine, String wings){
        super(gear, engine);
        this.wings = wings;
    }

    void display(){
        System.out.println(this.wings + " " + gear + " " + super.engine);
    }
}
```

→ gear is present in superclass also

this.gear will be displayed. If we want to distinguish & print gear of superclass, use super.gear.

```java
public static void main(String[] args){
    FlyingCar obj = new FlyingCar(5, "Electric", "Aluminium");
    System.out.println(obj.engine);
    obj.display();
}
```

This logically creates 2 objects.
Object of Car is logically inside flying Car.



{ Class can access properties & methods of only classes above it }

Stylus | Color | Line | Eraser | Backgrounds | Undo | Redo | Pages | Previous | Next | Erase | Board | Web | Documents | Show Desktop | OpenBoard

Polymorphism → one action in many ways/forms

Types of Polymorphism

- Compile-Time Polymorphism
  - or Static polymorphism
  - { Method Overloading }
    - → Number of arguments
    - → Order of arguments
    - → Return type
  - # Automatic type conversion
- Run-time polymorphism
  - or Dynamic Polymorphism
  - or { Dynamic Method Dispatch }
  - { Method overriding }
    - → Same
      - → No of arguments
      - → Type of arguments
      - → Return type

```
int n = scn.nextInt();

int [] arr = new int[n];
```

} Static Memory Allocation

[static only due to fixed size]

{ Memory is allocated at Run Time }

All the overheads(like determine the space which will be required on run time in the ,here no actual allocation of the space takes place) associated with the process is been determined in the compile time itself and actual allocation I the runtime and also the linking takes place at the compile time itselfs

function Overloading / Static Polymorphism / Compile-Time Poly

(Method)

Over loading takes place when there exist two function with same name but there exist a difference in the number of argument or data type of argument although return type might not be same or different

```
void display(){
    System.out.println(this.wings + " " + super.gear + " " + super.engine);
}

void display(int height){
    System.out.println(this.wings + " " + super.gear + " " + super.engine + " " + height);
}
```

In main() only if we call any method, the method which is to be called is decided at the time of compilation. Execution will be done at run-time only.

Method Overloading happens

- ↳ no of arguments ✓
- ↳ order of arguments ✓
- ↳ return type ✗

} should be different

Runtime Poly / Dynamic Poly / Method Overriding

{ Dynamic Method Dispatch }

drive() is in both the classes { Car as well as FlyingCar }

```
Car obj1 = new Car();
obj1.drive();

FlyingCar obj2 = new FlyingCar();
obj2.drive();

Car obj3 = new FlyingCar();
obj3.drive();

FlyingCar obj4 = new Car();
obj4.drive();
```

Always big(parent) in left and the other one in right(child)

```
Car ()
{   drive ()
    { car is running
    }
}

FlyingCar ()
{   drive () { car is Flying }
```

	present in parent	present in child	Which will execute after overriding
int gear	yes	yes	parent
int gear	yes	no	parent
int gear	no	yes	Error
drive()	yes	yes	child
drive()	yes	No	Parent
drive()	No	yes	Error

```
Car obj1 = new Car();
obj1.drive();

FlyingCar obj2 = new FlyingCar();
obj2.drive();

Car obj3 = new FlyingCar();
obj3.drive();

FlyingCar obj4 = new Car();
obj4.drive();
```

→ Parent class ref & parent class Obj (Running)

→ Child class ref & child class obj (flying)

→ Parent class ref child class obj (flying)

→ child class ref parent class object (NOT ALLOWED)

→ This is Run Time Poly.

Jiska object create hua hai usi ka method call hoga.

Parent class Reference → Child class object

⇓ only

But the properties & methods of reference will be allowed

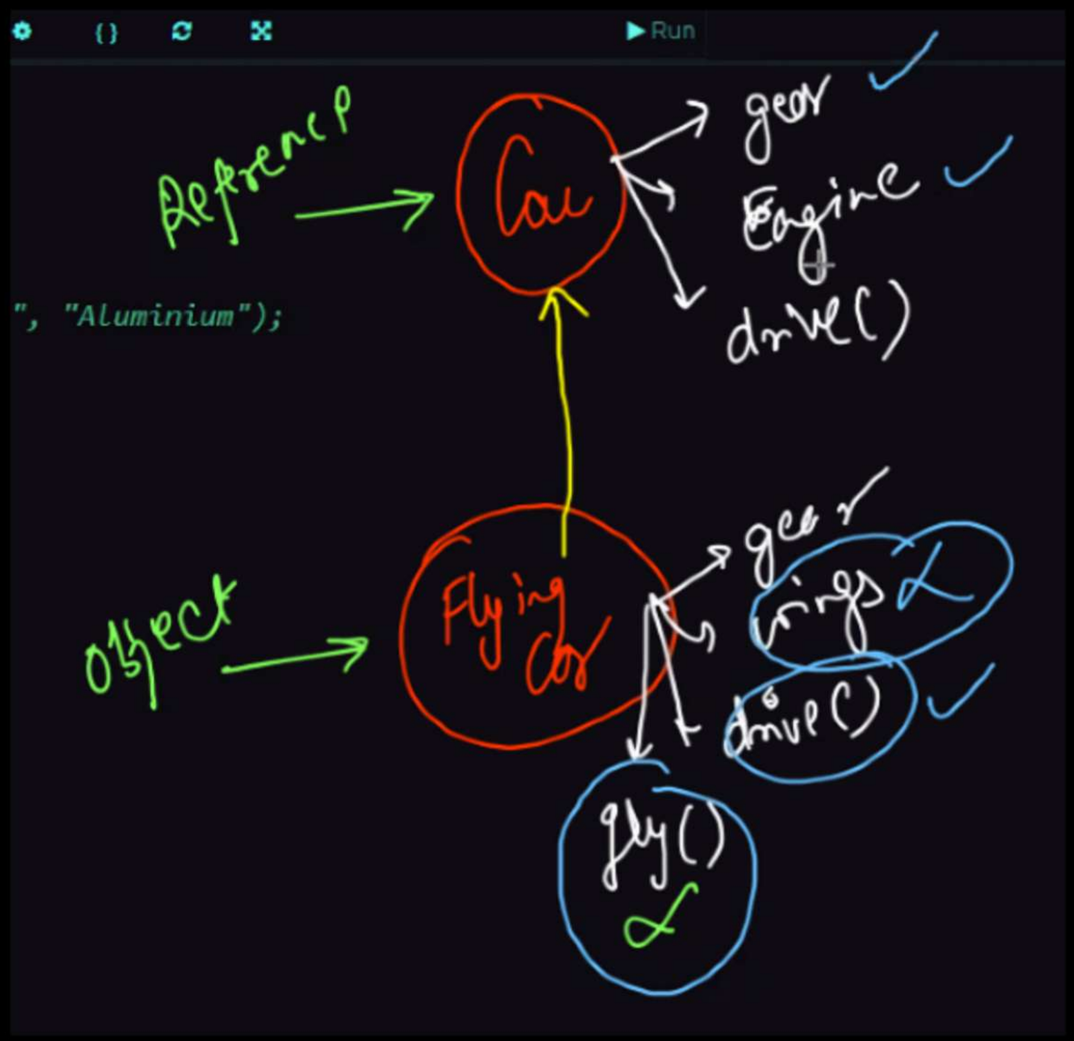
Car ✓ gear
✓ engine

car cannot access wings

flying car can access gear & engine.

flying Car ✓ wings

(wings are not accessible to car even if object is of flying car)



Method of child class will be accessed but properties of parent class itself will be accessed.

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

→ gives the functions which we can call

ClassName referenceName = new ObjectName();

→ provides the implementation of those functions

This class should be same as the reference class or it's child class.

A → B

parent class — child class

```
A obj = new A();
B obj = new B();
```

} ✓ Possible

```
A obj = new B();
```

→ Runtime polymorphism

```
B obj = new A();
```

} ⇒ Not possible → Why?

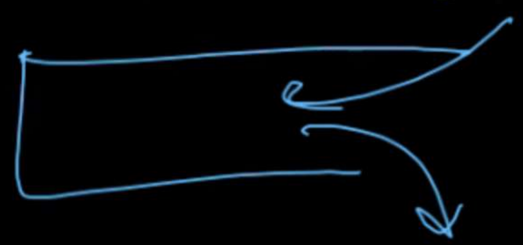
→ Constructor of B never called

⇒ B's data members never initialized

Stylus | Color | Line | Eraser | Backgrounds | Undo | Redo | Pages | Previous | Next | Erase | Board | Web | Documents | Show Desktop | OpenBoard

Reference → Stack

- → AddLast()
- → RemoveLast()

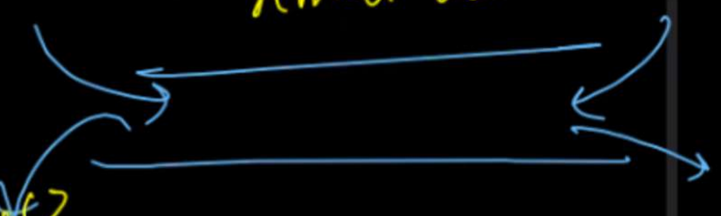


Reference → Queue

- → AddLast()
- → RemoveFirst()

Deque ← object

- → addFirst()
- → addLast()
- → removeFirst()
- → removeLast()



2

Queue q = new ArrayDeque<>()

Deque q = new ArrayDeque<>()

4

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

for Run Time Poly

- Order of arguments
- no of arguments
- return type

} should be same

Early Binding (VS) Late Binding

→ Normal Methods
{ w/o inheritance }

→ Method overloading

→ Method Overriding

note :→ Static methods cannot be overrided

because
↳ Static does not depend on object
whereas overriding depends on objects type

```
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
    }
}
```

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

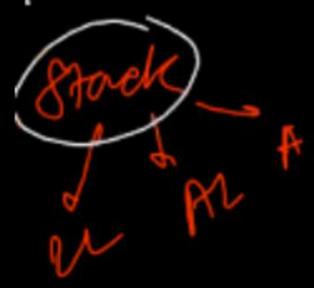
Abstraction

{Heap vs Priority Queue} => Heap is implementation of PQ.

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

PQ is abstract view of Heap.

+



Implementation → Abstract classes {Partial Abstraction}

Implementation → Interface {Full Abstraction}

Stylus Color Line Eraser Backgrounds Undo Redo Pages Previous Next Erase Board Web Documents Show Desktop OpenBoard

Abstract class

↳ If a class has atleast one abstract method, then class needs to be abstract.

↓ Only function prototype not definition.

↳ Such methods must be over-riden

↓ objects/instances of abstract class cannot be created!

abstract class cannot have

- → abstract constructor { concrete constructor can be there }
- → this keyword
- → abstract static methods { Note only static methods are allowed }

Abstract classes cannot be final.