① Arrays (Remaining)

② Abstraction

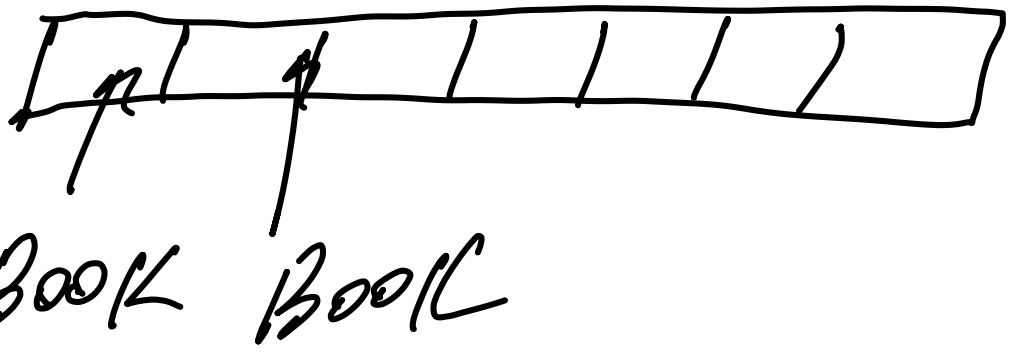
③ Encapsulation

④ Interface

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

```
String[] strArr = new String[N];
```

```
class Book {
    int pages
    String Name;
    //Constructor
    // getter setter
}
```



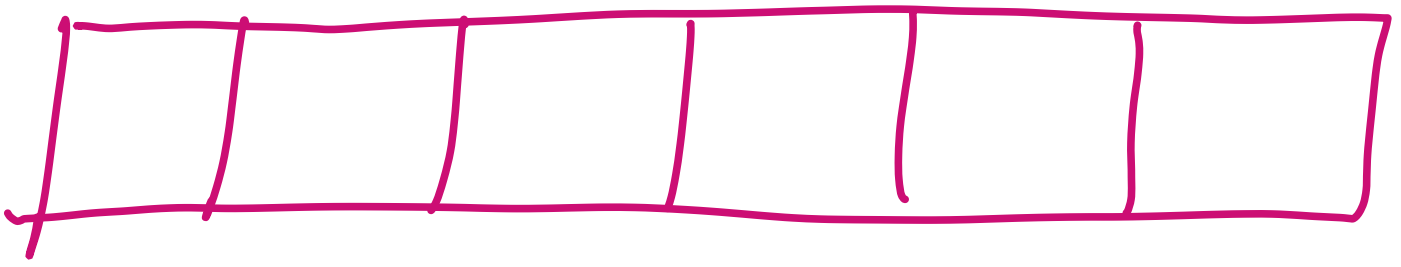

```
Book [] book = new Book[5];
```

```
class Book {
    int pages
    String Name;
      // Construction
      // getter setter
}
class main {
    psvm ( ) {
        Book[] book = new Book[3];
```

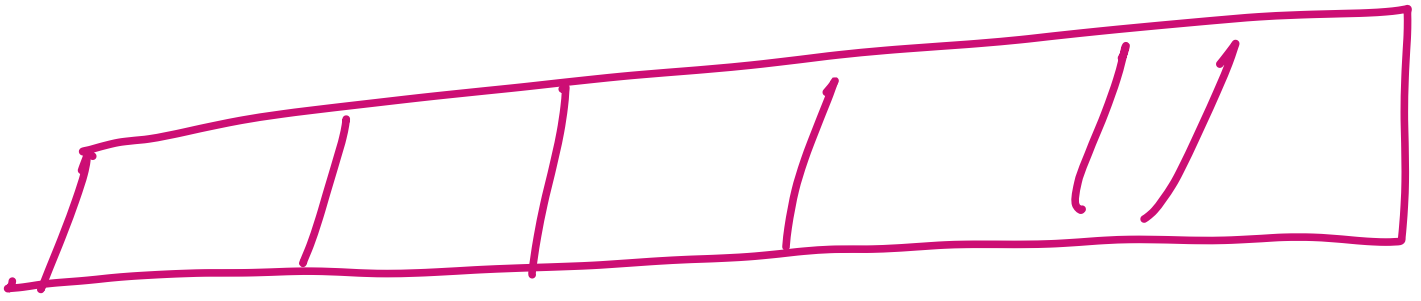
```
Book b1 = new Book(100, "ABC");
book[0] = b1;
Book b2 = new Book(150,
                   "EDE");
book[1] = b2
```

```
public class Book {
    private int noofpages ;
    private String name ;

   //Constructor
   //getter and setter
}
```

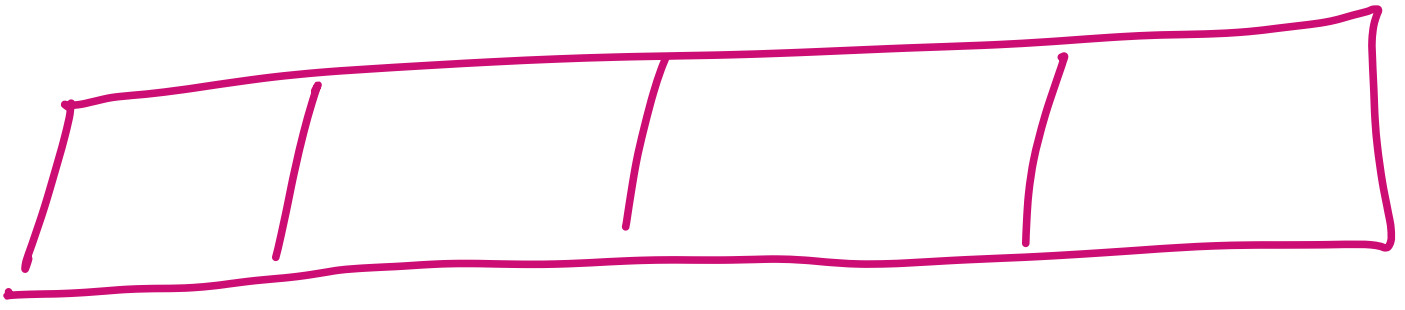
```
public class BookMain {
    public static void main(String[] args) {
        Book [] book = new Book[3] ;

        Book b1 = new Book() ;
        b1.setNoofpages(100);
        b1.setName("ABC");

        Book b2 = new Book(150, "DEF") ;

        Book b3 = new Book(180, "GHI") ;

        book[0] = b1 ;
        book[1] = b2 ;
        book[2] = b3 ;

        for(int i=0 ; i<book.length ; i++){
            Book b = book[i] ;

            System.out.println("Book name: " + b.getName());
            System.out.println("Pages: " + b.getNoofpages());

            System.out.println("======================");

        }
    }
}
```



Roll no (int)



name (String



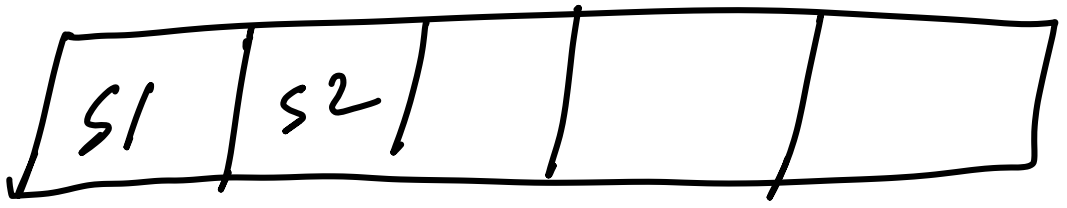
email (String)

```
class Student {
    int rollno;
    string name;
    string email;
    // constructor
    // getter/setter
}
}
int[] rollno = new int[5];
Student[] std = new Student[5];
```

↳ Array of student type

Student s1 = new Student(1, "A", "ABC");

std[0] = s1

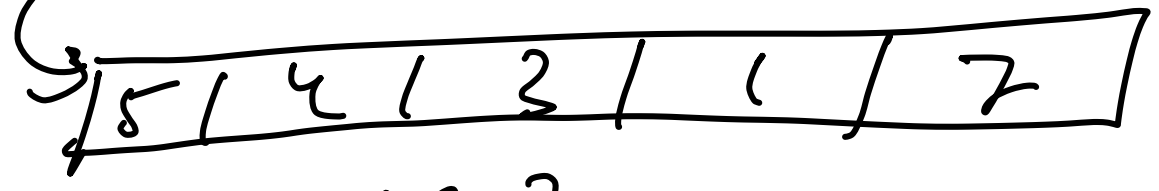


Student s2 = new Student(2, "B", "XYZ");

std[1] = s2

student(s1) = std[0];

A



int a = A[0] → 5

s.o.println(s1.getName());

so println(s1.getNo());

Abstraction

```
public abstract class Abstraction1 {

    abstract void show();

    void display(){
        System.out.println("Parent");
    }
}

public class Child extends Abstraction1{

    void show(){
        System.out.println("Inside Child");
    }
    void display(){
        System.out.println("Child display");
    }
}

public class Main {
    public static void main(String[] args) {
        Abstraction1 abs = new Child() ;
        abs.display();
        abs.show();
    }
}
```

```
public abstract class Vehicle {
    abstract void start() ;
    abstract void gearChange(int gear) ;
}

public class Car extends Vehicle {

    void start(){
        System.out.println("Car Satrt with key");
    }
    void gearChange(int gear){
        System.out.println("Grars in car: " + gear);
    }
}

public class Bike extends Vehicle {
    @Override
    void start() {
        System.out.println("Bike Start with kick");
    }

    @Override
    void gearChange(int gear) {
        System.out.println("Gear in Bike: " + gear);
    }
}

public class VehicleMain {
    public static void main(String[] args) {
        Vehicle car = new Car() ;
        car.start();
        car.gearChange(5);

        System.out.println("========================");

        Vehicle bike = new Bike() ;
        bike.start();
        bike.gearChange(4);
    }
}
```

Q: Diff. B/w concrete method & Abstract method

Concrete → Body ✓
→ No keyword

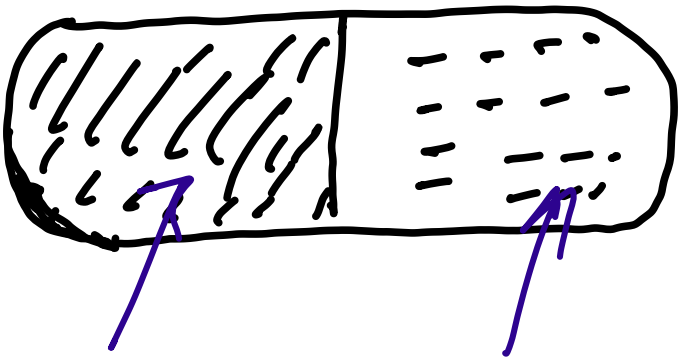
Abstract → Body ✗
→ abstract keyword

Concrete class vs abstract class

⇒ ① No keyword
* ② Can create object
③ only contain Concrete method

① abstract keyword
* ② No object
③ Concrete + abstract method

Encapsulation



Binding methods & variable in a single unit (ie; class)

⇒ variable → private

⇒ getter/setter method → public

↓

Java bean class

```
public class Student {
    private int id;
    private String name;
    private String email;

    // default Constructor
    // Public getter / setter method
}
```

Abstraction	Encapsulation
① Detail Hiding	① Data Hiding
② achieve through → abstract class & Interface	② achieve through private / public Access modifier.

Start End

10 200

10 ⇒
11
12
.
.
.
200

findArmstrong(int N) {

$153 \Rightarrow 3^3 + 5^3 + 1^3 = 153$

3^3
5^3
1^3

(N > 0) ans = ans + Rem^3

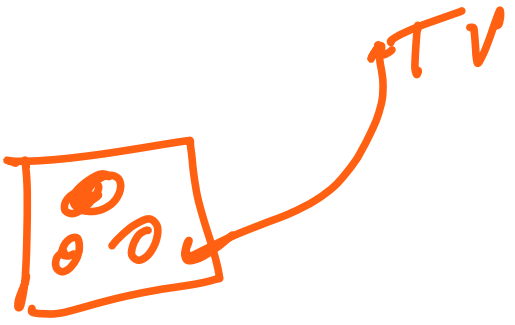
Interface

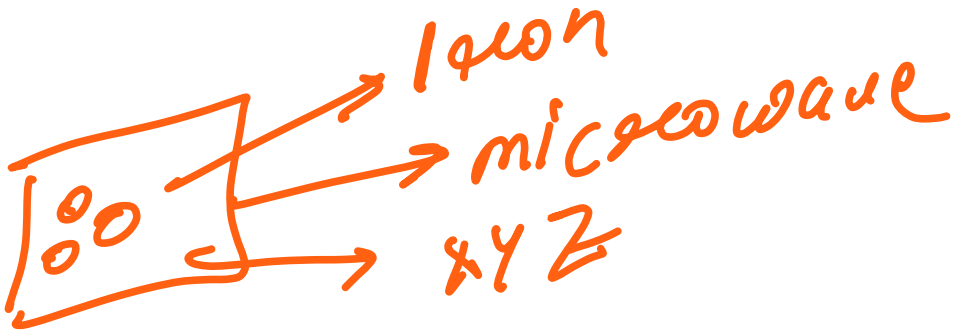
↳ Abstract class → methods → Abstract

```
interface I{
    void start();
    void stop();
}
```

Uses

↳ helps to achieve
- ↳ multiple Interface
- ↳ Complete Abstraction
- ↳ * Loose Coupling





Interface → OOPS way of creating a contract of function

- Variable → YES ⇒ public, static, final
- methods → YES ⇒ public, abstract
- Constructor ⇒ NO

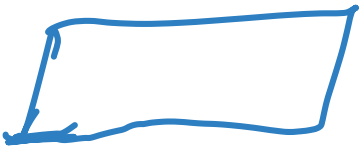
```
Interface I {
    int gear ;  // public, static, final
    void start() ;  // public abstract
}
```

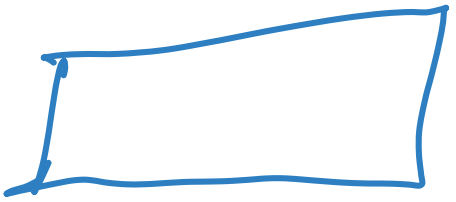
```
interface Vehicle {
    void start();
    void gearChange();
}

class Car implements Vehicle {
    void start() {
        print("start with key");
    }
    void gearchange() {
        print("gear changed");
    }
}
```

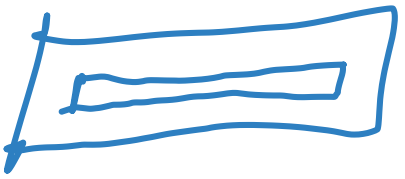
Q: Can we create object of an interface ⟹ No

Multiple Inheritance

```
interface I1 {
    void show();
}

interface I2 {
    void show();
}

class C {
    //
}

class InterfaceDemo extends C implements I1, I2 {
    void show() {
        //
    }
}
```







```
}
class InterfaceDemo implements I1, I2 extends C{

}

public class Main {
    psvm() {
        InterfaceDemo obj=new ID();
            obj.show();
    }
}

Test() {
⇒ super();
}
```

```
public class Parent {
    Parent(){
        System.out.println("Hi From Parent");
    }
}

public class Child extends Parent{

    void display(){
        System.out.println("Hi From Child");
    }



}

public class Main {
    public static void main(String[] args) {
        Child c = new Child();
        c.display();
    }
}
```

Output:

Hi From Parent

Hi From Child

# Concrete Class	Abstract class	Interface
① Concrete method	① Concrete + Abstract	① Abstract
② Can create object	② object ✗	② object ✗
③ Construction ✓	③ ✓	✗
④ No Abstraction	④ Partial Abstraction	⑤ Complete Abstraction
⑤ Cannot achieve multiple inheritance	✗	✓
⑥ Syntax	Syntax	Syntax

Java 8 new features

① Default method

interface I1 {

default void show() {

So.p/n("I1");

}

}

```
public interface I1 {
    default void show(){
        System.out.println("I1");
    }
}

public class C implements I1{

}

public class Main {
    public static void
main(String[] args) {
        C c = new C() ;
        c.show();
    }
}
```

(2) Static method

```
public interface I1 {
    static void show(){
        System.out.println("I1");
    }
}
```

Java 9 (JDK 9)

→ we can create private method in interface

→ private static methods

Marker Interface

→ which does not have any abstract method or variable

interface IE

{

}

Java Output formatting

println ✗

print ✗

printf
support format specifier

```
int a = 10;
int b = 20;
int sum = a+b;
```

so.pln("%s the sum of" + a + "and " + b + "is" + sum);

%s → string

%b

```
public class Sum {
    public static void main(String[] args) {
        int a = 10 ;
        int b = 20 ;
        int sum = a+b ;
        System.out.println("the sum of " + a + " and " + b +" is " + sum);

        System.out.println("================================");

        System.out.printf("The sum of %d and %d is %d", a,b,sum);
    }
}
```

%d → Integer
%f → float
%c → char
%C
%s → string
%S
%b → boolean
%B

```
public class Sum {
    public static void main(String[] args) {
        float f = 12.12345f ;
        System.out.printf("%f%n", f);
        System.out.printf("%.3f\n", f);

        System.out.printf("%c%n",'a');
        System.out.printf("%C%n",'a');

        System.out.printf("%s%n", "Arun");
        System.out.printf("%S%n", "Arun");

        System.out.printf("%b%n", true);
        System.out.printf("%B", true);

    }
}
```

```
Scanner sc = new Scanner(System.in);
while(sc.hasNext()){
    String str = sc.next();
    int num = sc.nextInt();
    System.out.printf("%-15s %03d%n", str, num);
}
```

%d → Integer

```
System.out.printf("%-15s %03d%n", "Arun", 56);
```

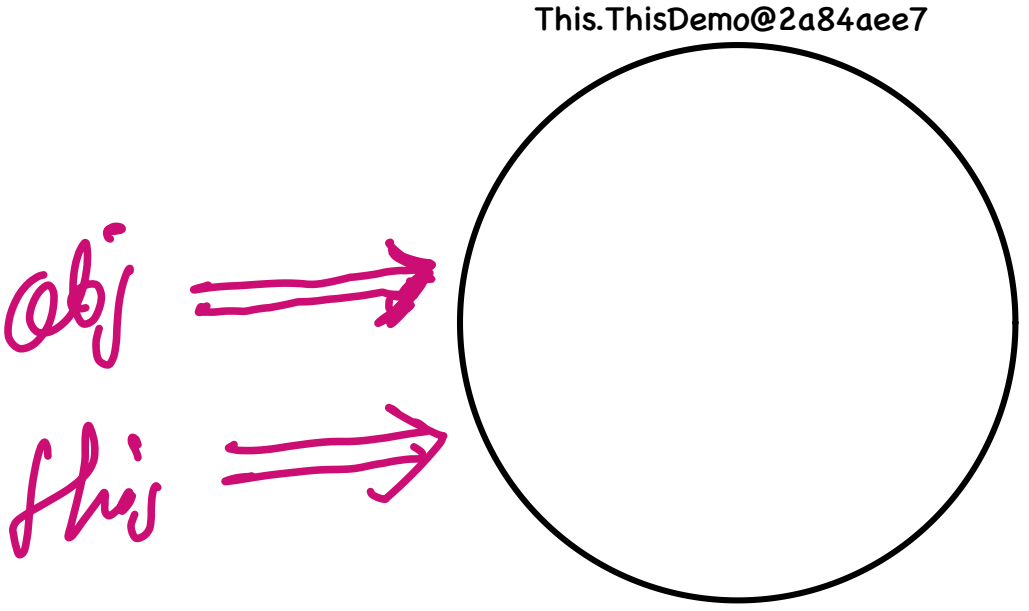
Keywords

- this
- super
- Static
- final

this Keyword

- refer to the current class object
- reference Variable

```
Class ThisDemo {
    void m1() {
        println("In m1 method" + this);
    }
    psvm() {
        ThisDemo obj = new ThisDemo();
        s.o.pln("Inside main" + obj);
        obj.m1();
    }
}
```



```java
public class ThisDemo {
    void m1(){
        System.out.println("Inside m1 : " + this);
    }

    public static void main(String[] args) {
        ThisDemo obj = new ThisDemo() ;
        System.out.println("Inside main: " + obj);
        //System.out.println(this); --> error since non sttic variable this cannot be access from a static method
        obj.m1();
    }
}
```

(1) This Keyword refers to current class Instance Variable.

```
class ThisDemo {
    int no;
    void m1(int no) {
        Sysout(no);
    }
}
psvm() {
    ThisDemo obj = new TD();
    obj.m1(20);
}
}
```

```
public class ThisDemo {
    int no = 10 ;
    void m1(int no){
        System.out.println(no);
        System.out.println(this.no);
    }

    public static void main(String[] args) {
        ThisDemo obj = new ThisDemo() ;
        obj.m1(20);
        System.out.println(obj.no);

    }
}
```

② used to invoke current class method.

this.methodname();

```
public class ThisDemo {
    void m1(){
        System.out.println("inside m1");
        m2();  //this.m2()
    }

    void m2(){
        System.out.println("Inside m2");
    }

    public static void main(String[] args) {
        ThisDemo obj = new ThisDemo() ;
        obj.m1();

    }
}
```

(3) Used to invoke current class constructor.

```
class ThisDemo {

    ThisDemo() {
        //
    }

    ThisDemo(int a) {

    }

}

PSVM() {
    TD obj = new TD();
```

```
public class ThisDemo {

    ThisDemo(){
        this(10) ;
        System.out.println("hello from default");

    }
    ThisDemo(int a){
        System.out.println("Hello from parameterized");
    }

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

==========

```
public class ThisDemo {

    ThisDemo(){
        System.out.println("hello from default");

    }
    ThisDemo(int a){
        this() ;
        System.out.println("Hello from parameterized");
    }

    public static void main(String[] args) {
        ThisDemo obj = new ThisDemo(10) ;
    }
}
```

Constructor chaining

```
class Test {

    Test() {
        this(10);
        Print("default constructor");
    }

    Test(int a) {
        this("Arun");
        S.O.Pln(a)
    }

    Test(String str) {
        S.O.Pln(str);
    }
}
```

psvm() {

Test t = new Test();

}

}

```
public class Test {

    Test(){
        this(10) ;
        System.out.println("Default Constructor");
    }
    Test(int a){
        this("Arun") ;
        System.out.println(a);
    }
    Test(String str){
        System.out.println(str);
    }
}


public class Main {
    public static void main(String[] args) {
        Test t = new Test() ;
    }
}
```

4. used to pass as an argument in the method.

```
class Test {
    void m1() {
        s.o.p("inside m1");
        m2(this);
    }
    void m2(Test t) {
        s.o.p("inside m2");
    }
}
```

psvm() {
    Test t = new Test();
    t.m1();
}

```
public class Test {
    void m1(){
//        Test ob = new Test() ;
//        m2(ob) ;
        m2(this) ;
        System.out.println("inside m1()");
    }

    void m2(Test t){
        System.out.println("inside m2");
    }
}
```

5. used to pass as an argument in the Constructor.

```
class Test {
    void m1() {
        XYZ ob = new XYZ(this);
    }
}
class XYZ {
    XYZ(Test t) {
        s.o.p.ln(t);
    }
}
```

(6) used to return the current class instance.

```
class Test {
    Test m1() {
        return this;
    }

    psvm() {
        Test t = new Test();
        Test ob = t.m1();
        print(ob);
    }
}
```

Super Keyword

↳ refers to parent class instance variable.

① refers to immediate parent class instance variable

```
class SuperDemo {
    int no = 10;
}
class Child extends SuperDemo {
    int no = 20;
    void show(int no) {
        S.o.pln(no);
        S.o.pln(this.no);
        S.o.pln(super.no);
    }
}
```

```
}
main {
    psvmc 1{
        child c = new child();
        c.show();
    }
}
```

② used to invoke parent class method.

③ Used to invoke parent class Constructor.

```
class A {

    A() {
      s.o.pln("Inside A");
    }

class B extends A {

    B() {
      s.o.pln("Inside B");
      super();
    }
}

psvm() {

    B b = new B();
```

* this & this()

* super & super()

**

Static

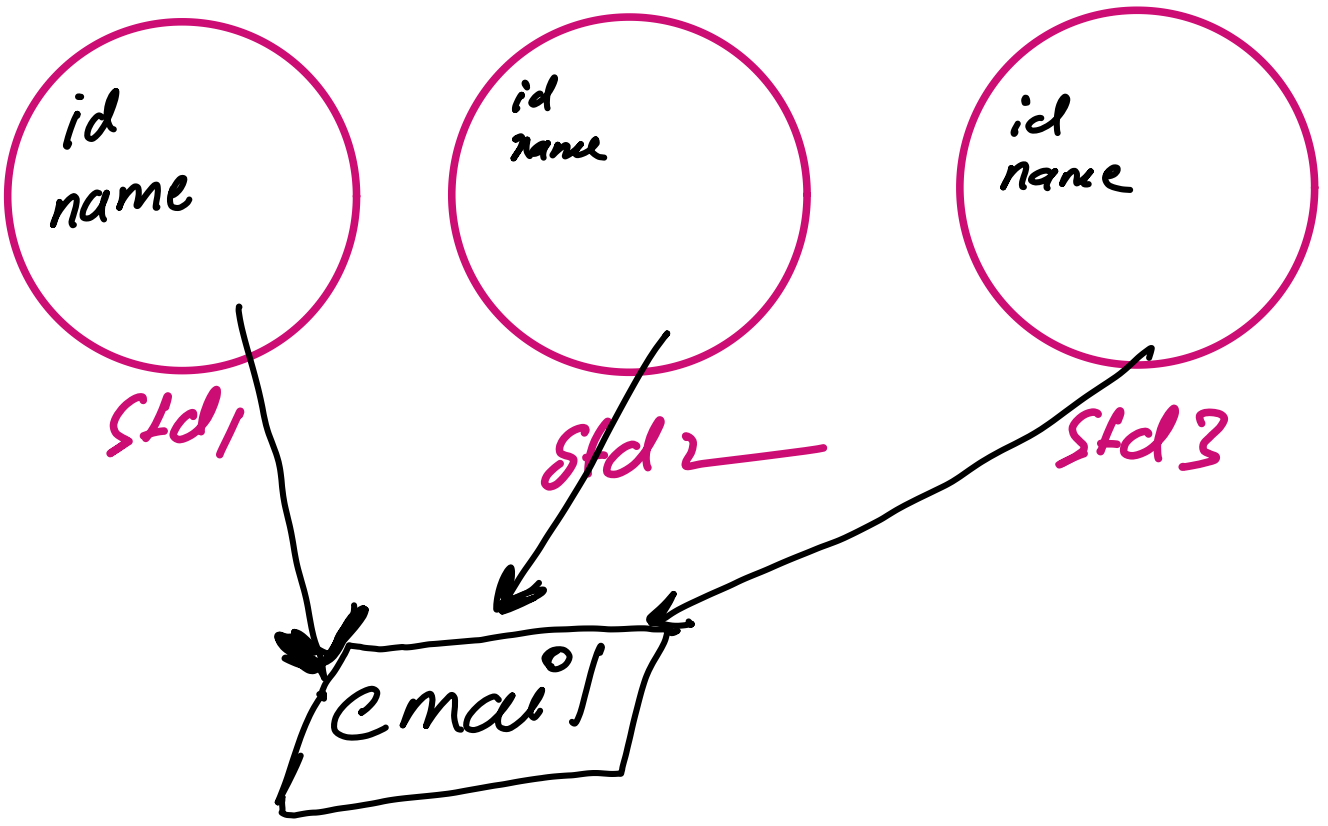
- variable
- method
- static block
- Inner class / nested class
- outer class X

```
class outer {
    class inner {
        m1() {
            s.o.println("Hi");
        }
    }
}
```

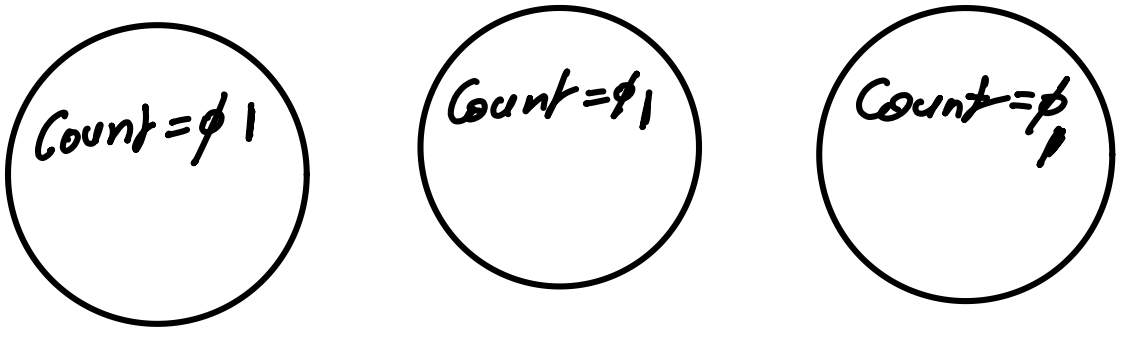
~~Inner obj = new Inner();~~
~~obj.m();~~

① class StaticDemo {
int a = 10;
static int b = 20;
void m() {
//
}
}
main {
psvm() {
SD obj = new SD();
s.o.pln(obj.a);
s.o.pln(obj.b);
}
}

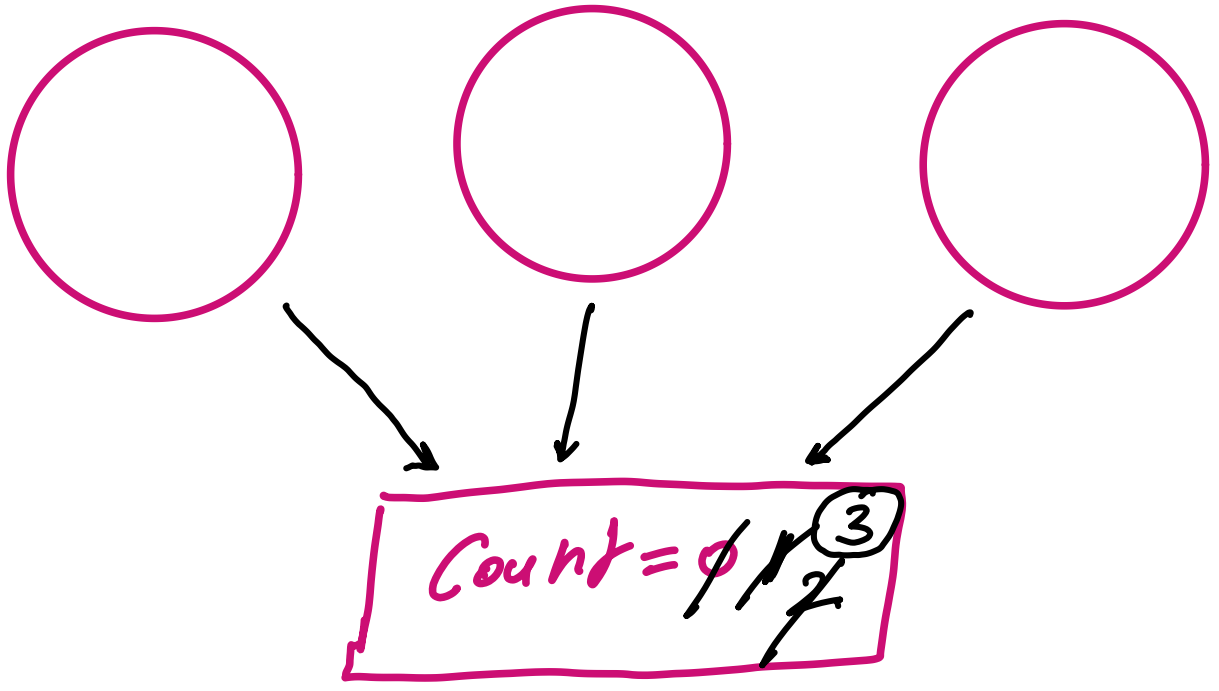
```
# Class Student {
    int stdId;
    String name;
    static String Email = "arunkumar@gmail.com"
    // Constructor

    void display() {
        Print("StudentId:" + stdId);
        Print(name)
        Print(Email);
    }

main {
    Psum() {
        //
    }
}
```




```
public class Counter {
    int count = 0 ;

    Counter(){
        count++ ;
        //System.out.println(count);
    }

    void noOfVisitor(){
        System.out.println(count);
    }
}
```



```
public class Counter {
    static int count = 0 ;

    Counter(){
        count++ ;
        //System.out.println(count);
    }

    void noOfVisitor(){
        System.out.println(count);
    }
}
```



c1.count → 3

c5.count → 5