| Type | Memory size (bytes) | Default Values | Range |
|---|---|---|---|
| byte | 1 | 0 | [-128 , 127] |
| short | 2 | 0 | [-32k , 32k] |
| int [default type] | 4 | 0 | [-2B , 2B] |
| long (L) | 8 | 0 | - |
| float (F) | 4 | 0.0 | - |
| double [default type] | 8 | 0.0 | - |
| char | 2 | single space | A,Z \| a,z |
| boolean | no-size (1 bit) | false | true/false |

# # Datatypes :

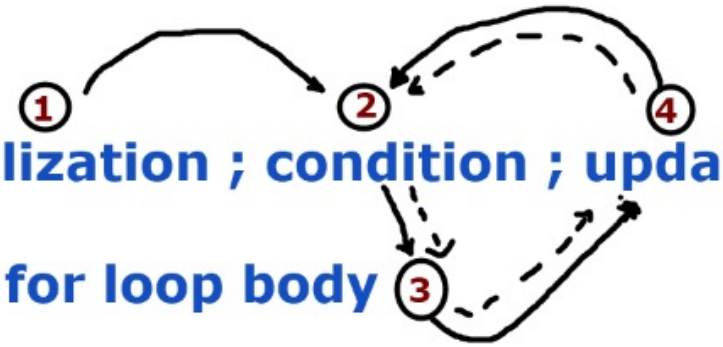
**use : To represent data in memory we can use datatypes.**

## # PM on datatype:

**1) used to represent type of variable.**
**2) datatype decides the memory size of the variable.**
**3) datatype decides the range of value wich are stored in variable**

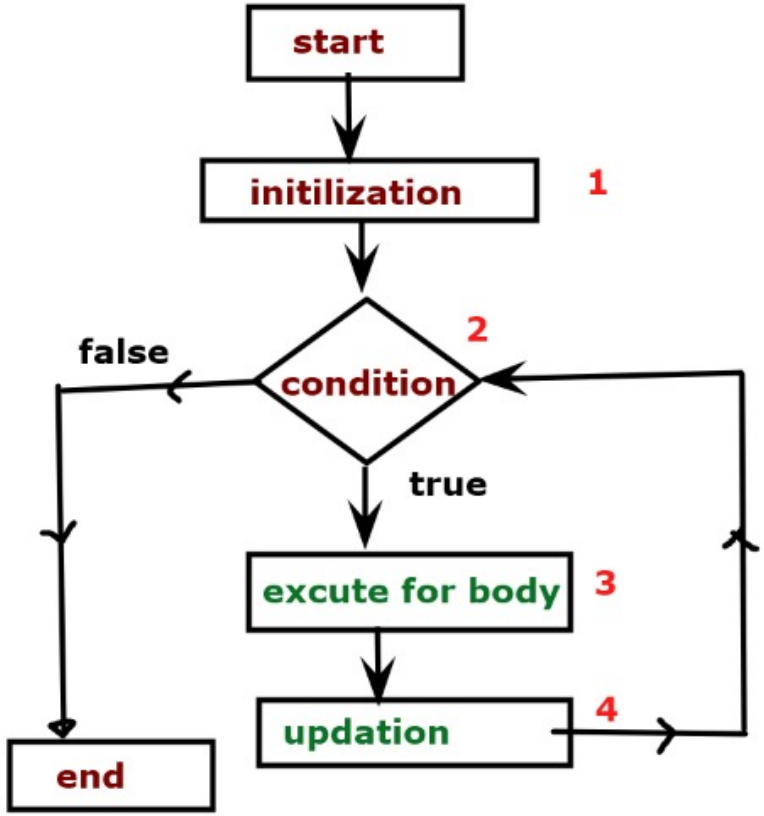
# for => I(initilization) C(condition) U(update)

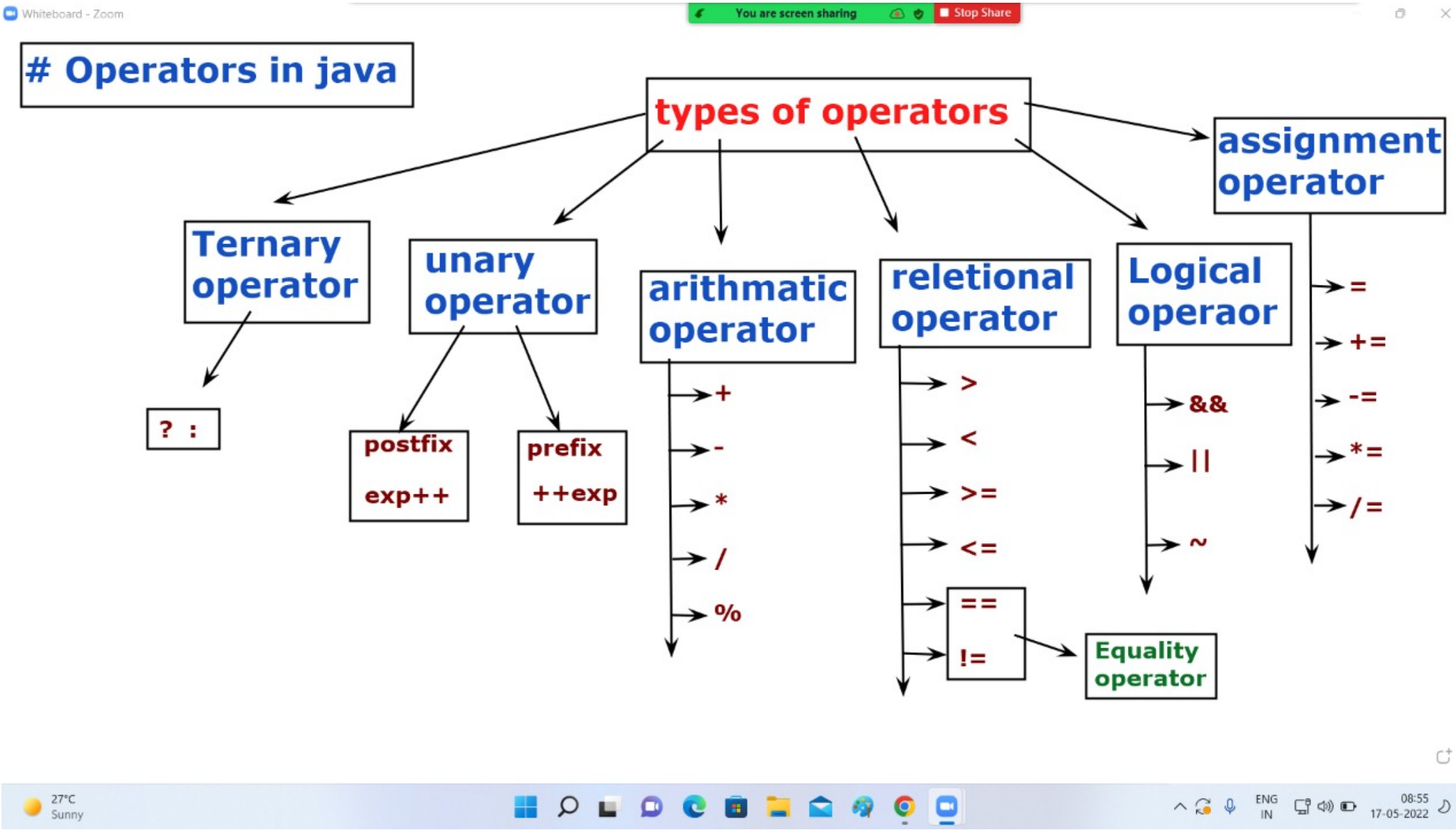
sty :

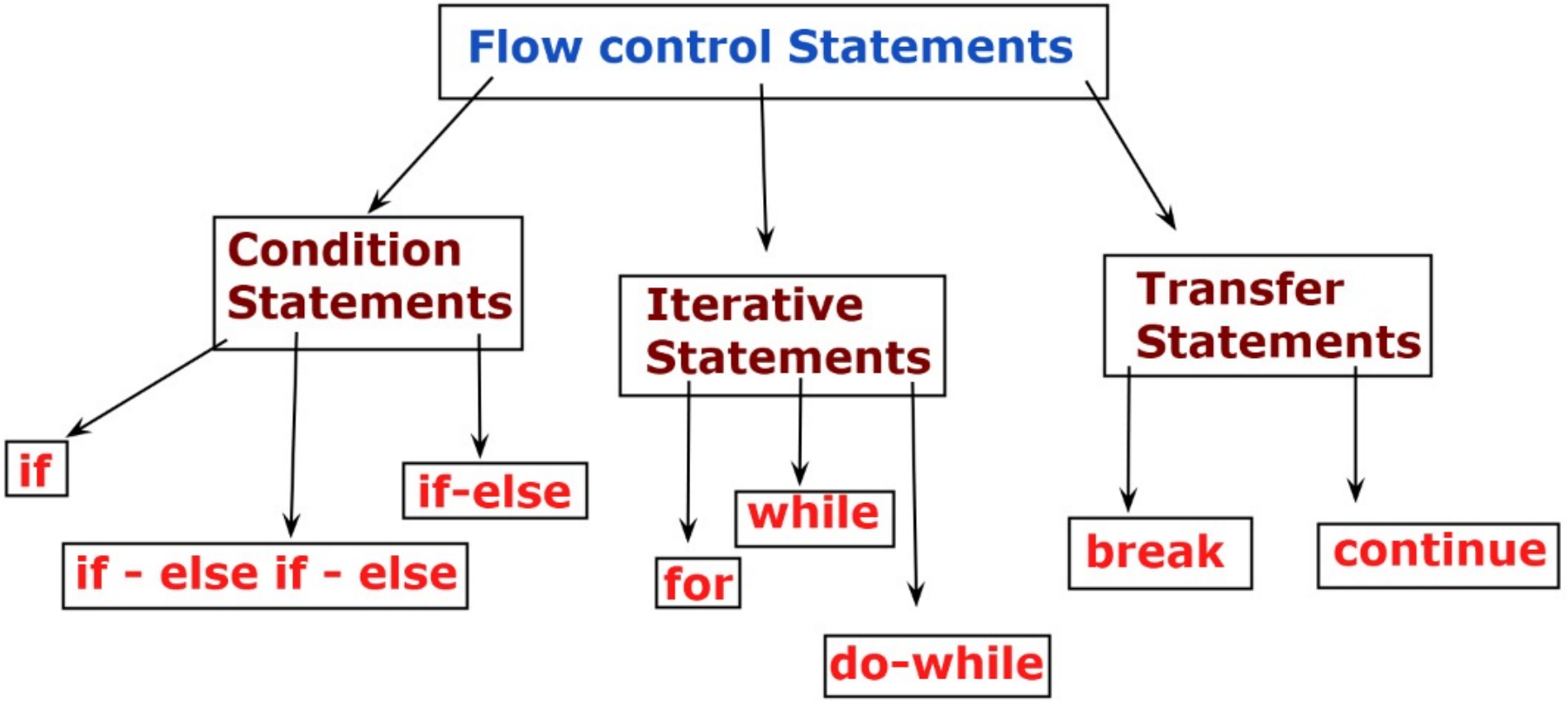



```
for (intilization ; condition ; update){

    // for loop body

}
```

## Flow of For Loop




Whiteboard - Zoom
You are screen sharing
Stop Share
# Operators in java
types of operators
Ternary operator
? :
unary operator
postfix
exp++
prefix
++exp
arithmatic operator
+
-
*
/
%
reletional operator
>
<
>=
<=
==
!=
Equality operator
Logical operaor
&&
||
~
assignment operator
=
+=
-=
*=
/=
27°C
Sunny
ENG
IN
08:55
17-05-2022



**break & continue => LOOPS**

**break =>**

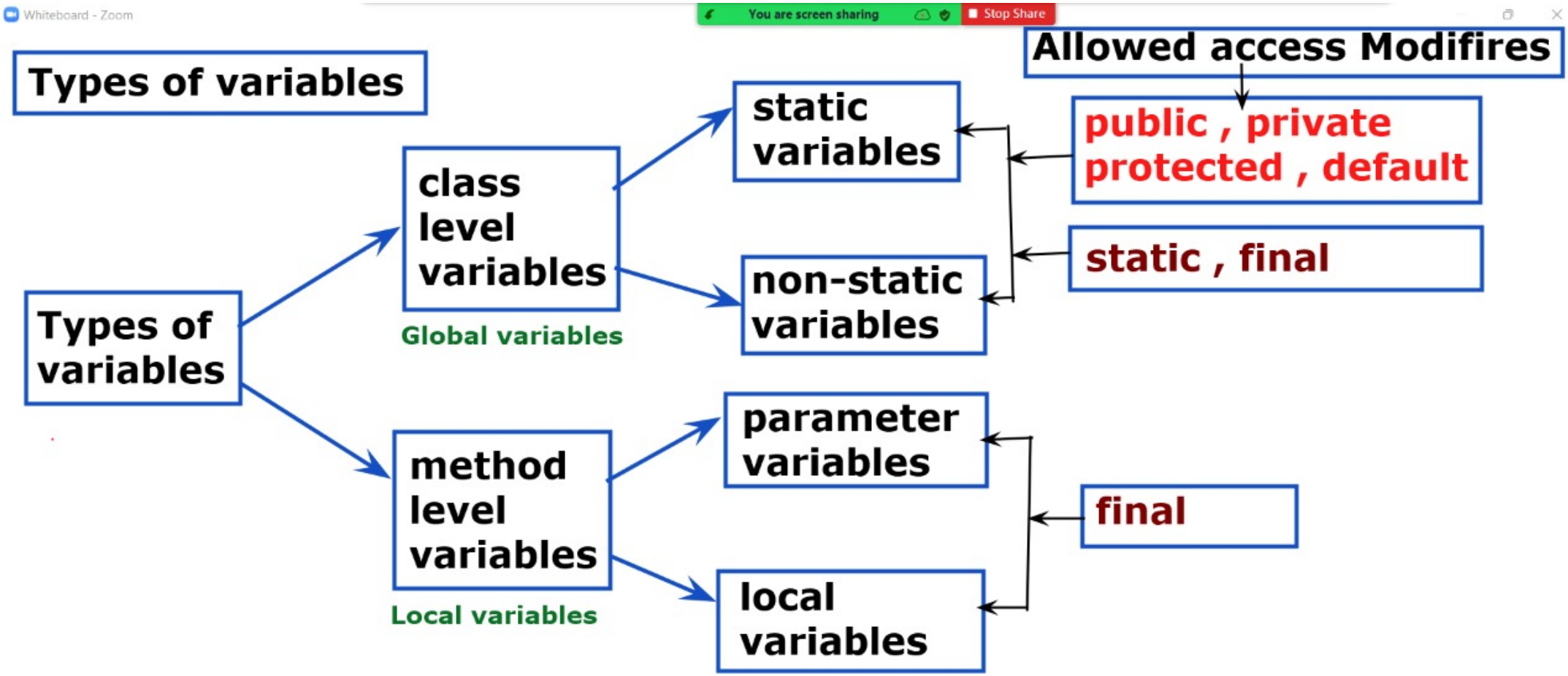
**use : To break/terminate loop on the basis condition we can use break statement.**

**contiune =>**

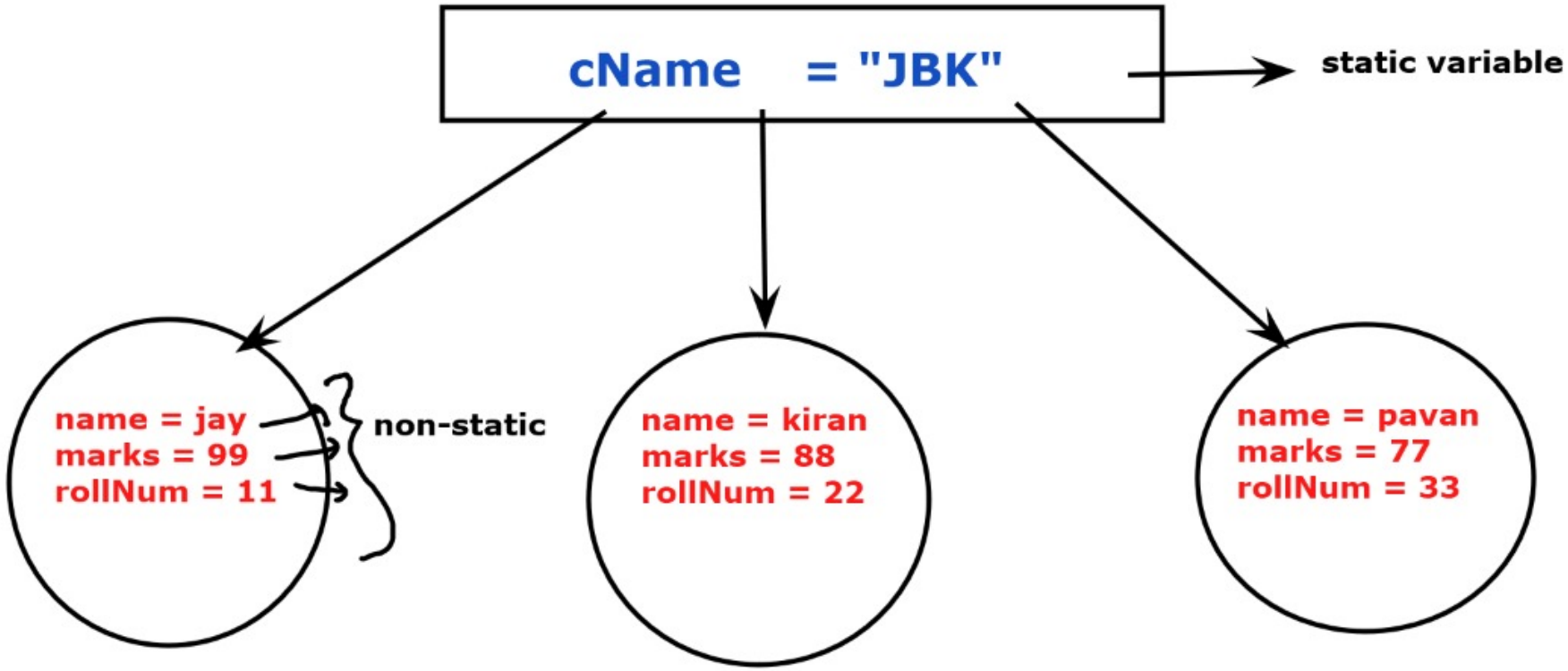
**use : Skip the current Iterations and go for next Iterations we can use continue statement.**

**NOTE:**

**1) if we use break then the loop will terminate/end.**
**2) if we use continue then the loop will not terminate.**

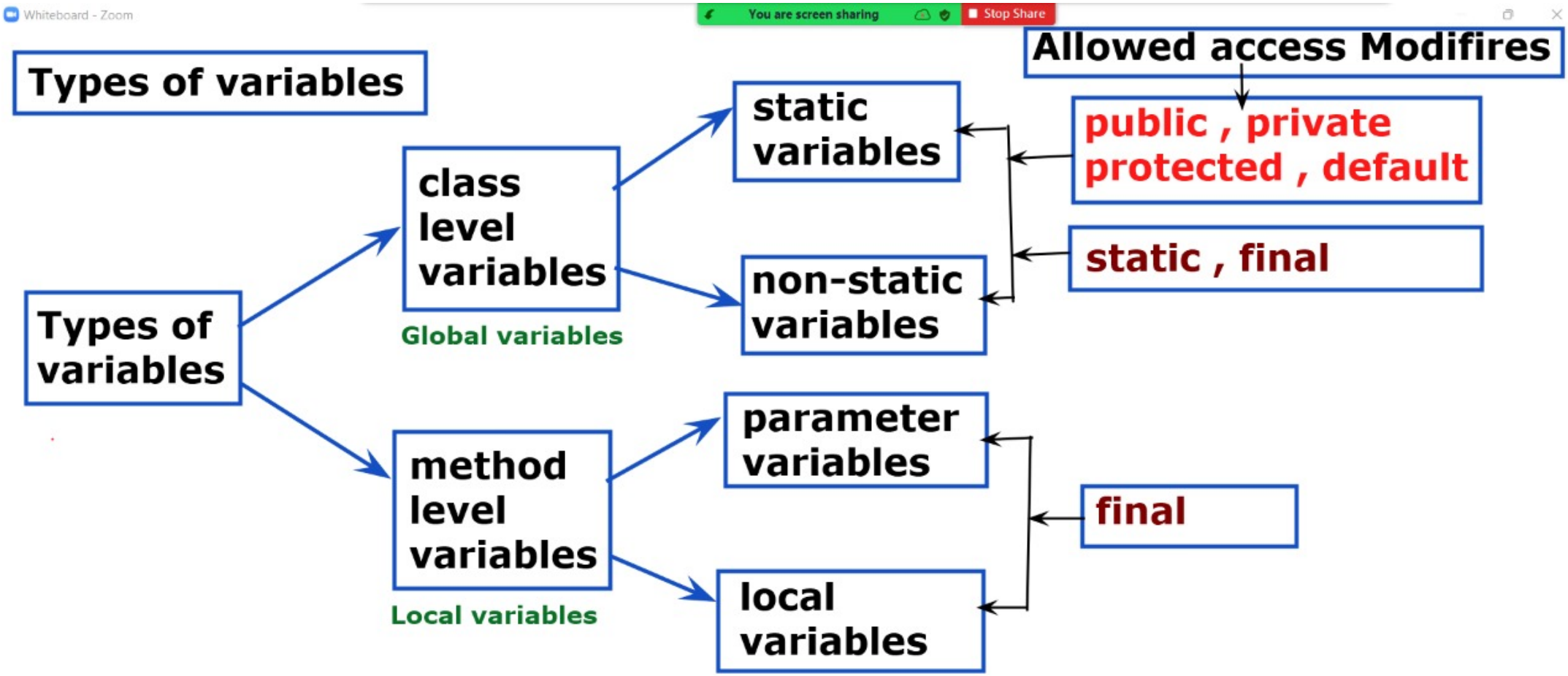


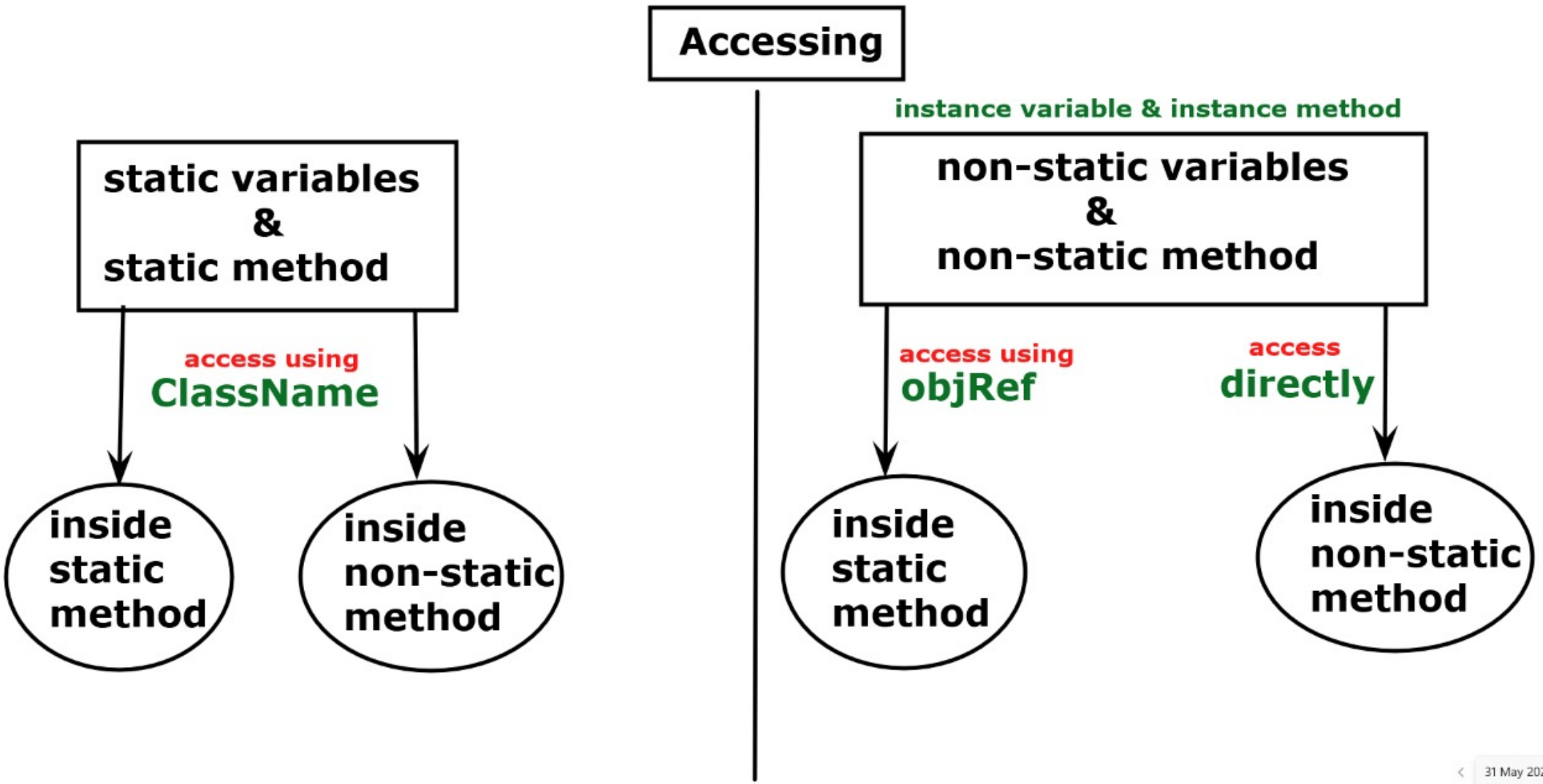
| | static | non-static |
|---|---|---|
| place | inside class but outside of all methods , blocks and constrtors | inside class but outside of all methods , blocks and constrtors |
| defintion | static dataType varName ; | dataType varName ; |
| creation | before object creation or .class file is loaded in JVM | after object creation |
| destoryed | .class file is removed from JVM | after object destoryed |
| access | using classname | using object refrence |
| eg | class ClassName{<br>static collageName = "JBK" ;<br>} | class ClassName{<br>String name ;<br>double marks ;<br>} |




**NOTE:**

1) for each object a separate copy of non-static variable will be maintained at object level.
2) for all objects a single copy of static variable will be maintained at class level.





| | takes nothing return nothing | takes someting return nothing | takes nothing return something | takes something return something |
|---|---|---|---|---|
| defination | void add(){<br>int a = 10 ;<br>int b = 20 ;<br>sopln(a+b);<br>} | void add(int a , int b){<br>sopln(a+b);<br>} | int add(){<br>int a = 10 ;<br>int b = 20 ;<br>int c = a + b ;<br>return c ;<br>} | double add(int a , int b){<br>double c = a + b ;<br>return c ;<br>} |
| calling | objRef.add(); | objRef.add(10 , 20); | sopln(objRef.add()) ;<br>or<br>int res = objRef.add();<br>sopln(res) | sopln(objRef.add(10 , 20)) ;<br>or<br>double res = objRef.add(10 ,20);<br>sopln(res) |

**NOTE:**

1) return statement must be a last statement in your method body.
2) after return statement if we write java code then that code will not execute.

# # Types of Methods

-> use : To perform specific task..
-> methods are used to write bussiness Logic.


| | static method | non-static method |
|---|---|---|
| defination | static returnType methodName(input args){<br>// processing of static variables<br>// processing of input args<br>} | returnType methodName(input args){<br>// processing of static & non-static variables<br>// processing of input args<br>} |
| calling | ClassName.methodName() ; | objRef.methodName() ; |

# # difference between constructors and methods

| | constructors | methods |
|---|---|---|
| 1 | constructors name and classname must be same. | method name can be anything. |
| 2 | constructors is not having any returntype including void. | method is having returntype. |
| 3 | for each object constructor will execute only once. | for each object method will execute any number of times. |
| 4 | constructors are used to initilized non-static variables. | To write bussiness logic we can use methods. |
| 5 | constructor will execute automatically when we create object. | we have to call methods to execute. |
| 6 | types :- A] Default , B] Parameterized | types :- A] static , B] non-static |

# # Constructors :

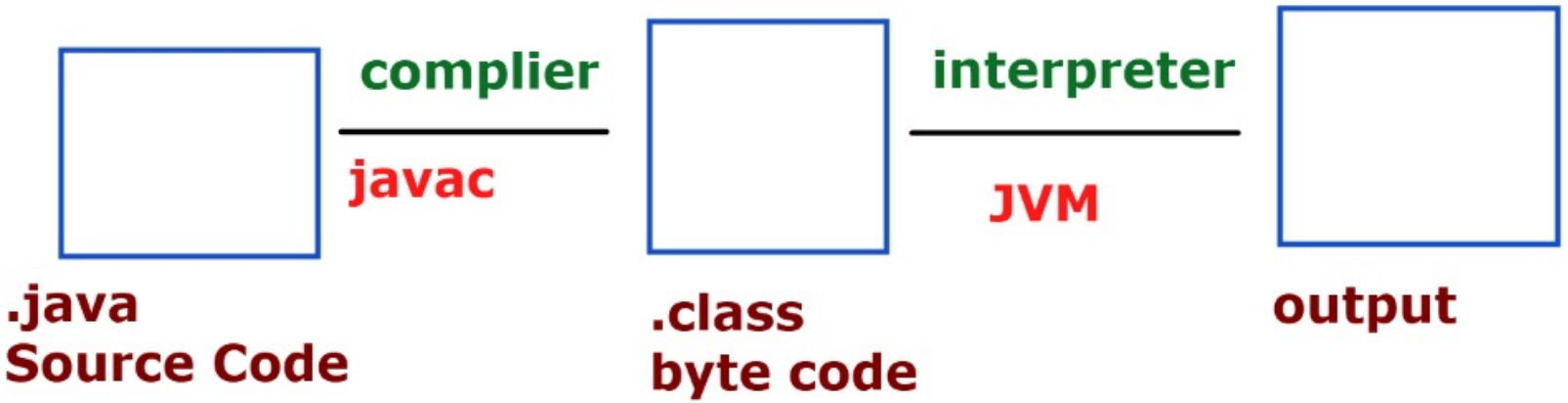
1) constructor name and class name must be same.
2) constructor is a method only wich does't have any return type.
3) for each object constructor will excute only once.
4) constructor will execute automatically when we create object of class.

**Puprose :**

-> The main purpose of constructor is initilization of instance/non-static variables.

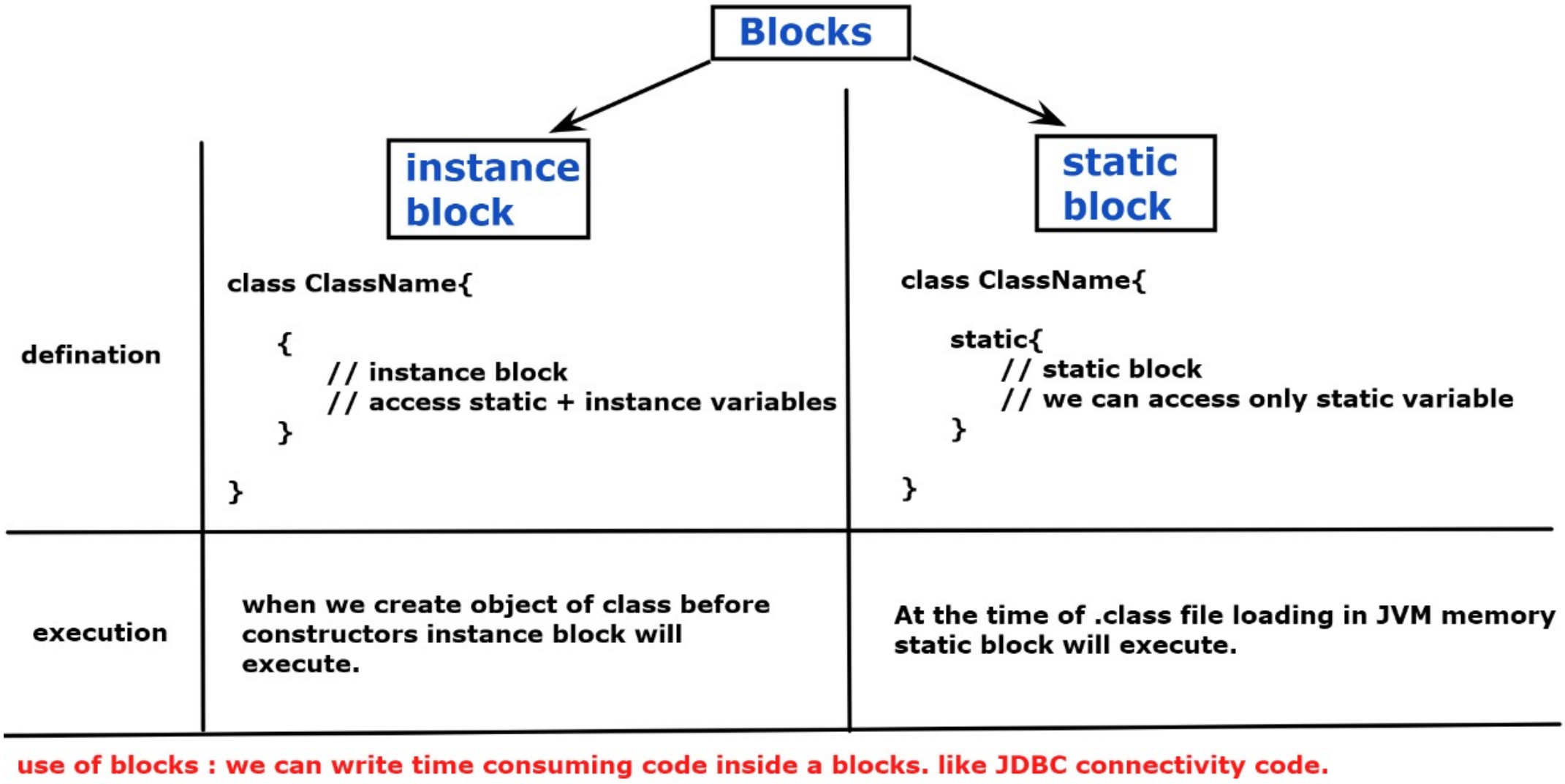
**sty :**

```
class ClassName{

    ClassName(input parameters){
        // initilization of non-static variables
    }

}
```




JVM -> Java Virtual Machine

| | instance block | static block |
|---|---|---|
| 1 | instance block deals with object. | static block deals with class. |
| 2 | {<br>// instance block<br>} | static{<br>// static block<br>} |
| 3 | instance block can access static and non-static variables. | static block can access only static variables. |
| 4 | exceute at the time of object creation.<br>before constructor execution. | exceute at the time loading .class file in JVM memory. |



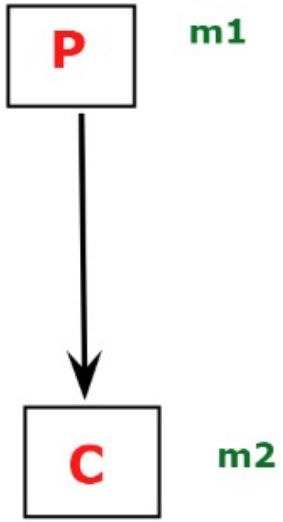

use of blocks : we can write time consuming code inside a blocks. like JDBC connectivity code.

# # single/simple Inheritance :

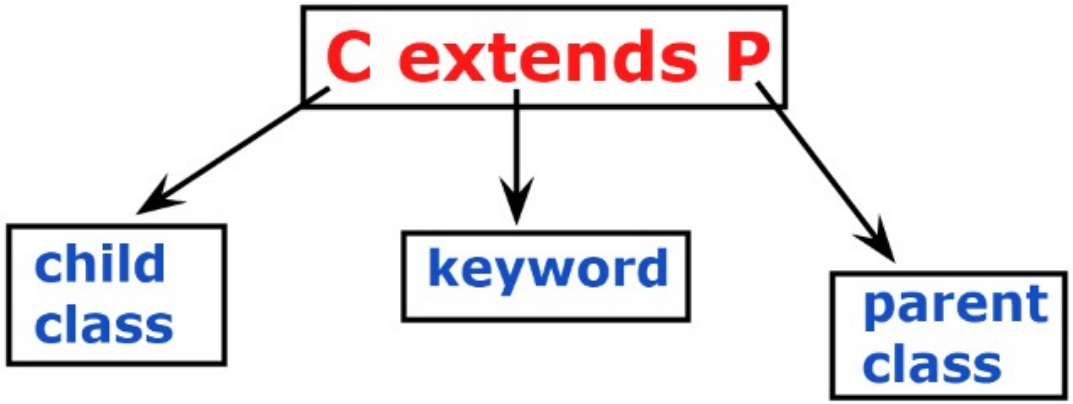
## -> single parent & single child

```
class P{
    void m1(){
        sopln("parent class m1 method) ;
    }
}

class C extends P{
    void m2(){
        sopln("child class m2 method") ;
    }
}
```

```
P p1 = new P() ;
p1.m1() ;

C c1 = new C() ;
c1.m1() ;
c1.m2()
```



**sty of Inheritance :**

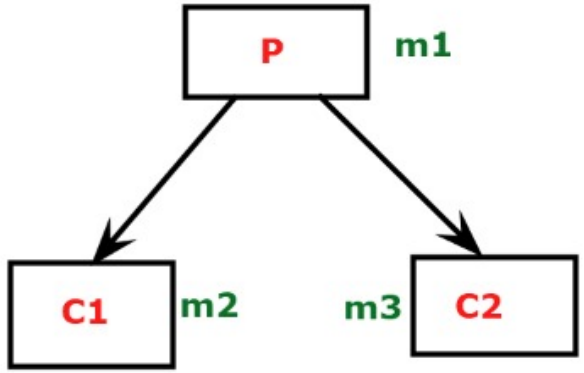


# # hierarchical Inheritance :

## -> single parent & multiple childs .

```
class P{
    void m1(){
        sopln("parent class p -> m1 method") ;
    }
}

class C1 extends P{
    void m2(){
        sopln("child class c1 -> m2 method") ;
    }
}

class C2 extends P{
    void m3(){
        sopln("child class c2 -> m3 method") ;
    }
}
```

```
P p = new P() ;
p.m1()  ;

C1 c1 = new C1() ;
c1.m2() ;
c1.m1() ;

C2 c2 = new C2() ;
c2.m3();
c2.m1();
```

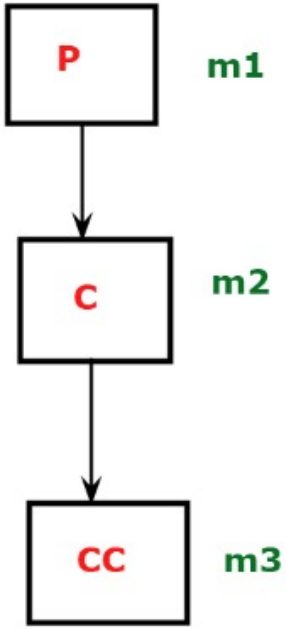


# # Multi-level Inheritance :

## -> single parent & single child but in multiple level.

```
class P{
    void m1(){
        sopln("parent class P -> m1 method") ;
    }
}

class C extends P{
    void m2(){
        sopln("child class C -> m2 method") ;
    }
}

class CC extends C{
    void m3(){
        sopln("child class CC -> m3 method") ;
    }
}
```

```
P p = new P() ;
p.m1() ;

C c = new C() ;
c.m1() ;
c.m2() ;

CC cc = new CC() ;
cc.m1() ;
cc.m2() ;
cc.m3() ;
```



# # Method Overloading ...

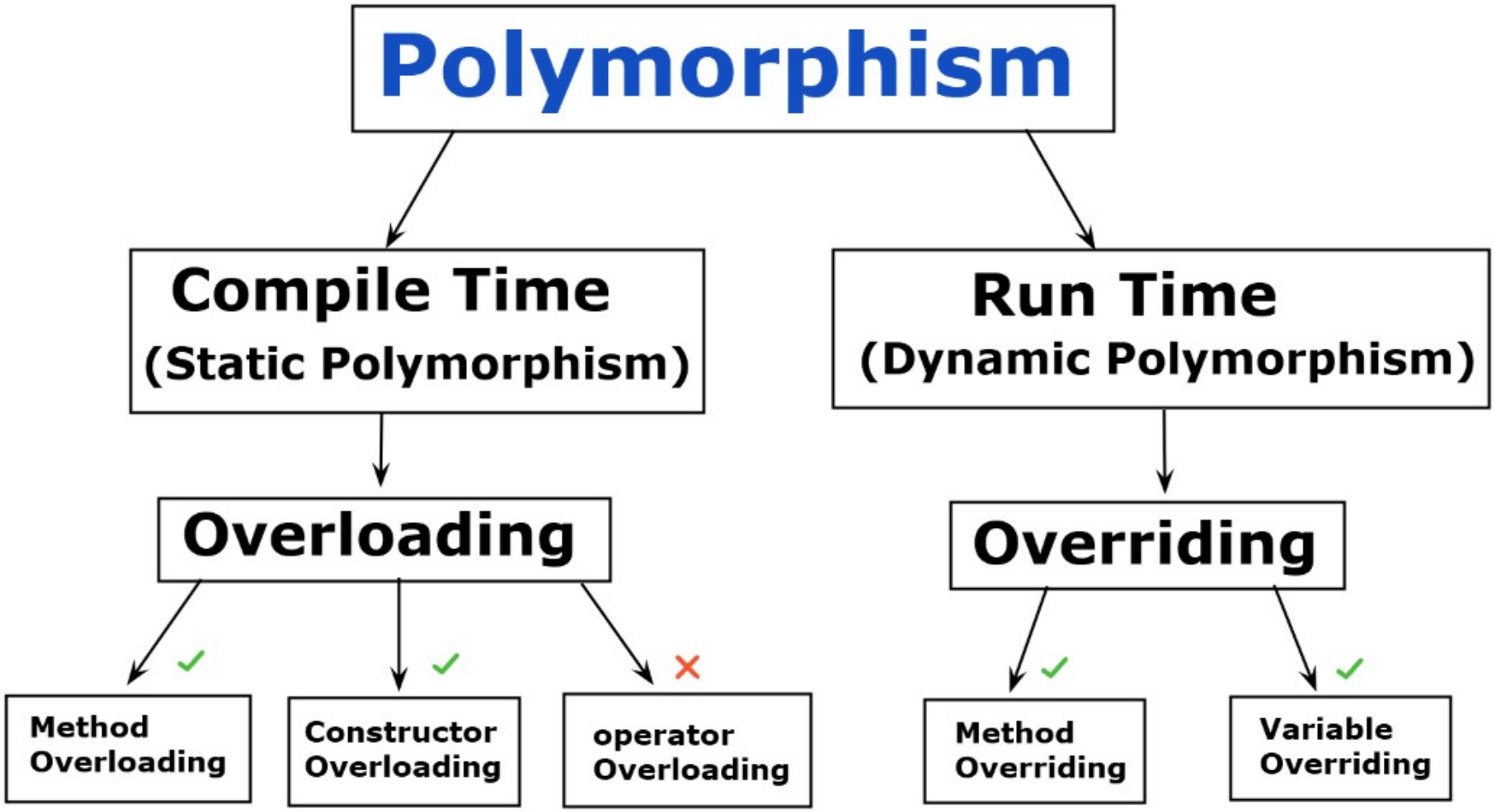
-> if the class is having multiple methods with same name but different argument list.

# # IMP Points :

1) Two or more methods is having same name.
2) methods must be defined in same class.
3) methods with different argument list.
   -> number of arguments can be different .
   -> sequence of arguments can be different .
   -> type of arguments can be different .

# NOTE :

-> Overloaded methods may have the same or different return type.


Polymorphism
Compile Time
(Static Polymorphism)
Run Time
(Dynamic Polymorphism)
Overloading
Overriding
Method Overloading
Constructor Overloading
operator Overloading
Method Overriding
Variable Overriding

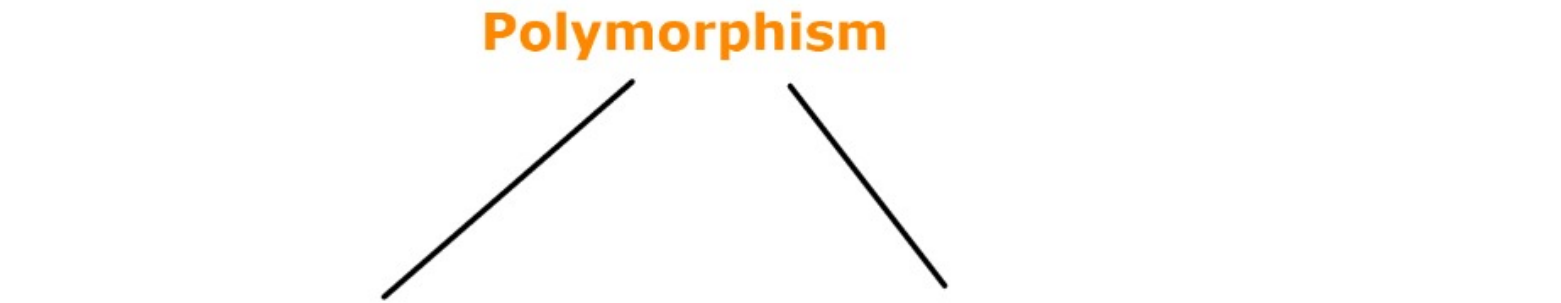
# # Constructor Overloading ...

-> if the class is having multiple constructors with same name but different argument list.

# # IMP Points :

1) Two or more constructors is having same name.
2) constructors must be defined in same class.
3) constructors with different argument list.

-> number of arguments can be different .
-> sequence of arguments can be different .
-> type of arguments can be different .




1. method
2. constructor overloading
3. operator (x)

1. method overriding
2. variable overriding

| | Overloading | Overriding |
|---|---|---|
| 1 | if the class is having multiple methods with the same name but different argument list. | re-defining the parent class method in child class |
| 2 | method overloading is performed in same class | method overriding occurs in two classes that having parent-child (inheritance) reletionship. |
| 3 | in method overloading parameter list must be different. | in method overriding parameter list must be same. |
| 4 | method overloading is a example of compile time polymorphism. | method overriding is a example of run time polymorphism. |
| 5 | in method overloading overloaded methods may have the same or different return type. | in method overriding both parent class and child class methods must have the same return type. |

# # Method Overriding ...

-> if child class has the same method as declared in the parent class it is called as method overriding.

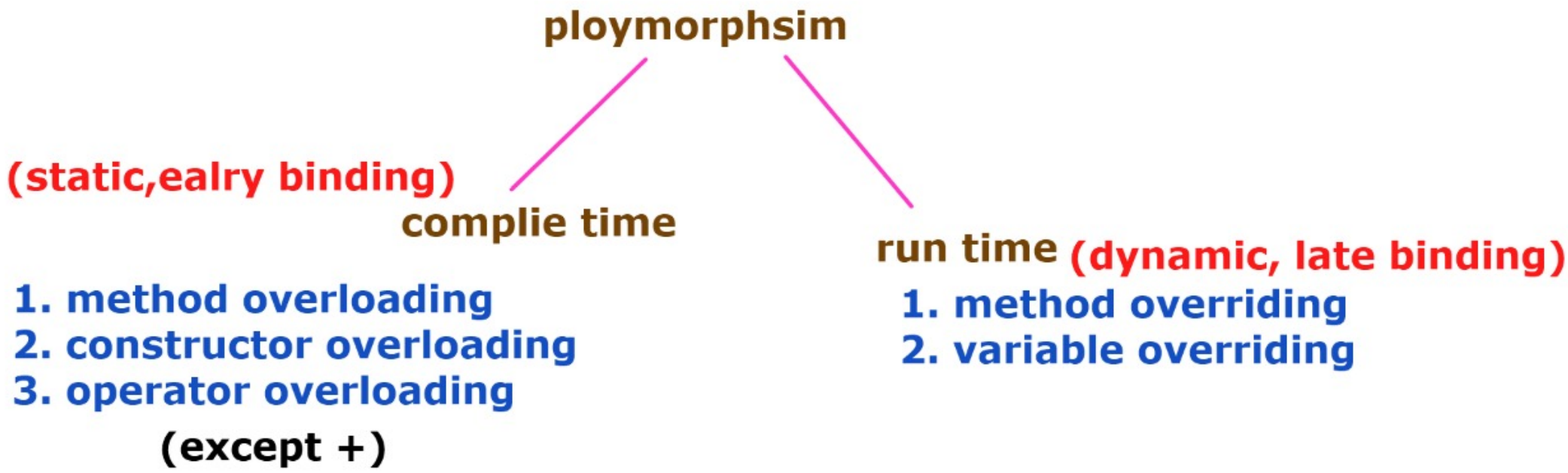
-> re-defining parent class method in the child class called as method overriding.

# # IMP Points :

1) method must have the same name as in the parent class.
2) method must have the same parameters as in the parent class.
3) There must be IS-A(inheritance) reletionship.
4) method overriding occurs in two classes that having parent-child (inheritance) reletionship.

# # NOTE :

1) Both parent class and child class methods must have the same method name.
2) Both parent class and child class methods must have the same return type.
3) Both parent class and child class methods must have the same parameter list.
4) we can not override the methods declared as final and static.


ploymorphsim
(static,ealry binding)
complie time
1. method overloading
2. constructor overloading
3. operator overloading
(except +)
run time (dynamic, late binding)
1. method overriding
2. variable overriding

```
class TopLevelClass{

}
```

```
class TopLevelClass{

        class InnerClass{

        }

}
```

**sty :**

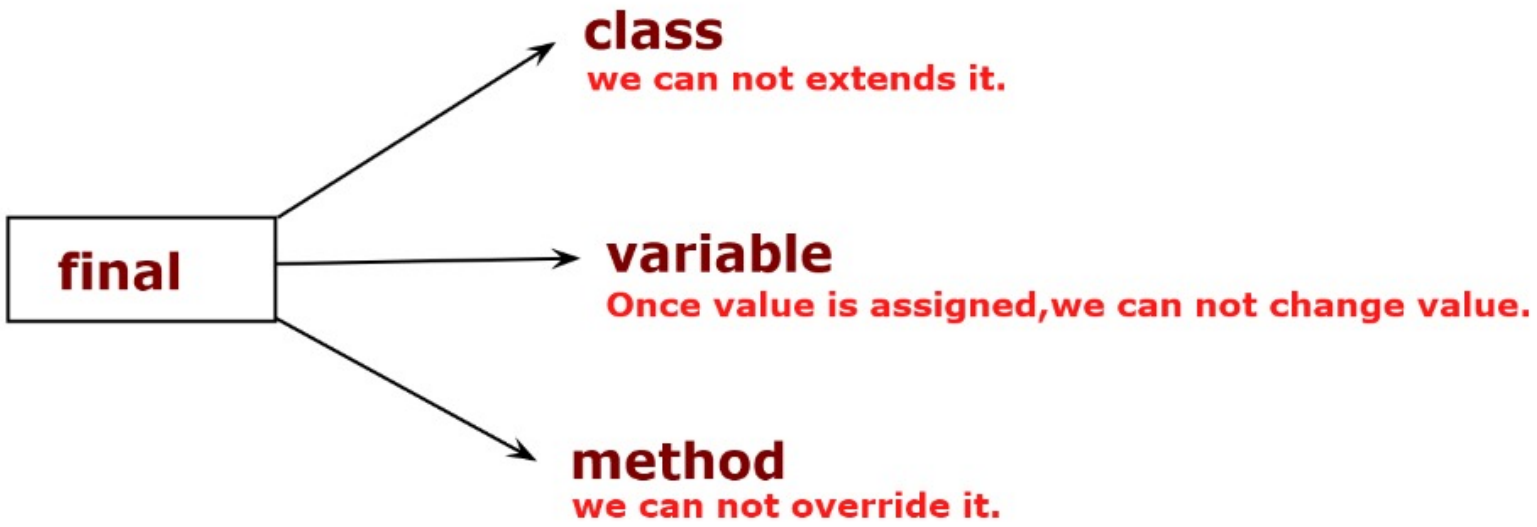
**AccessModifires modifires Name ;**

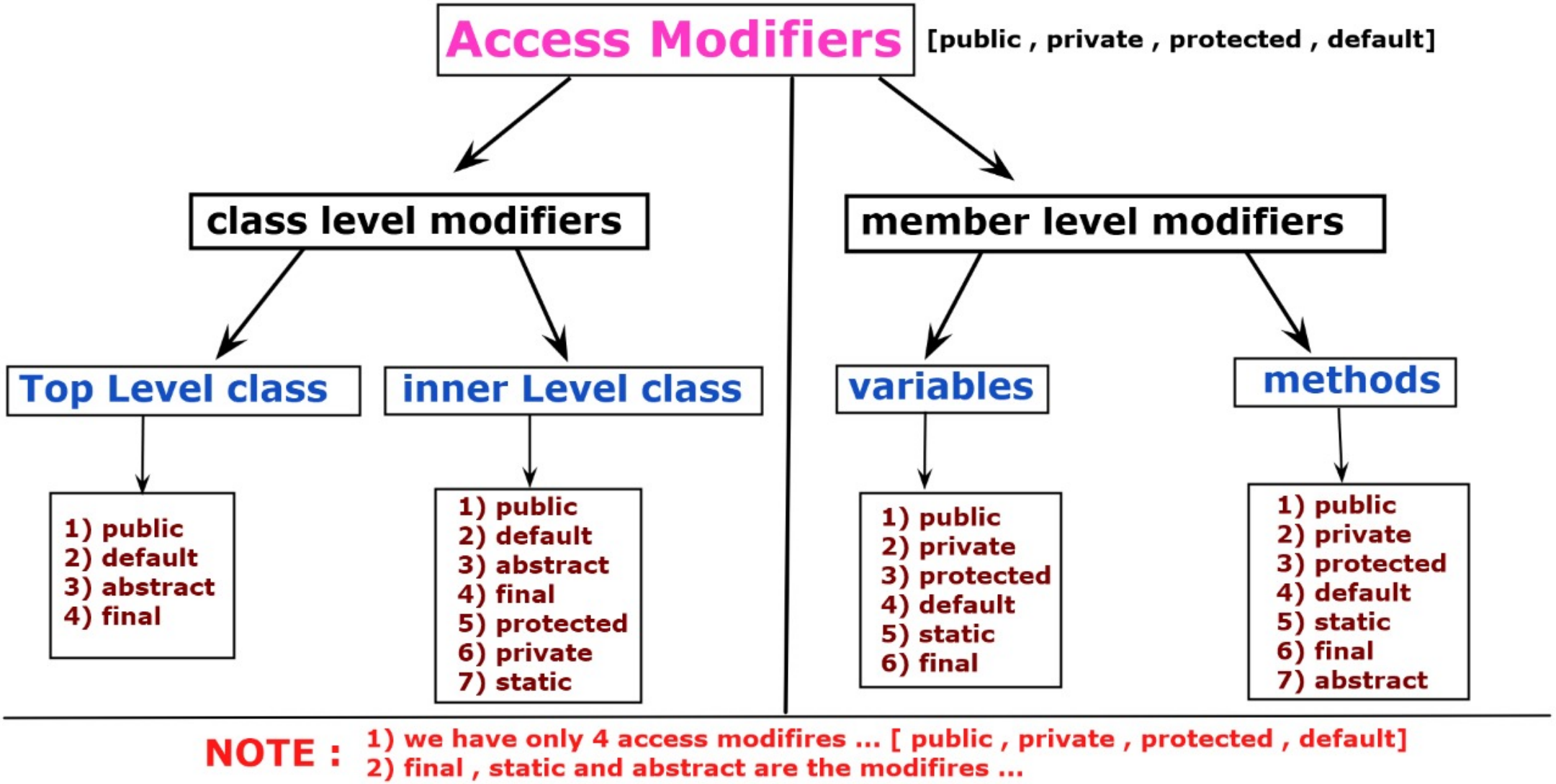
1) public final int varName = value ;
2) public final void methodName(){

    }
3) public final ClassName{

    }

# # final keyword :






**NOTE :** 1) we have only 4 access modifires ... [ public , private , protected , default]
2) final , static and abstract are the modifires ...

# public (Global Level) :

# Access : within the package + outside of the package.

# Default (Package Level) :

# package-private

# Access : within the package only.

# private (Class Level) :

# Access : within the class only.

# protected :

# Access : within the package + outside of package only in child classes using child reference.

# Access Modifires in Java :

1) private
2) <default> or <package-private>
3) protected
4) public

# NOTE :

1) private , protected and public are the keywords.
2) default is not keyword it is just concept.

# USE :

-> Access modifires will tell us "where to access" class , variables and methods.
-> To set access permissions we can use Access Modifires.

## Q- How To Achive Encapsulation ??
## -> using two steps :

1) By making variables(data) as private.
2) By making methods as public.

**example :**

```
class JBKBank{

    private long accountNumber ;
    private double balance ;

    public void getBalance(){
            if (validateUser){
                    // authenticated user
                    sopln(balance) ;
            }else{
                    // un-authenticated user
                    sopln("please provide vaild username or password")
            }
    }
```

Data Hiding...by making variables(data) as private.

Hiding a data Behind a method called as Encapsulaion.

# # abstract class :

-> class wich contains the abstract keyword in its declaration it is known as abstract class.

sty :

```
abstract class ClassName{

}
```

# # IMP Points :

1) By using abstract classes we can achive partial abstraction.
2) we can achive 0-100 % abstraction using abstract class.
3) abstract classes may or may not contains abstract methods.

Q-What is abstract methods ??
=> method without body called as abstract method.
sty :

```
abstract return type methodName() ;
```

4) if the class is having at least one abstract method then complsory we have to define the class as abstract class.
5) if you inherite abstract class we have to provide implementations to all the abstract method in child class.

# # NOTE : we can't create object of abstract class.

# Abstraction :

**-> The process of Hiding the implementation details from the user , only the essential functionality will be provided to the user.**

**-> The user will have the information on what the object does insted of how it does.**

### -> Example of ATM :

**we can see the different functionality options (withdraw, Check balance, Change PIN etc) provided on ATM screen but can't see the actual program developed to run these functionalities.**

# Ways To Achive Abstraction :

**-> There are two ways to achive abstration in java**

1) abstract class
2) interface

## # Array declaration :

```
datatype[]  varname;
```

## # array object creation :

```
varname= new datatype[size] ;
```

## # array variable initilization :

```
varname[index] = value ;
```

## # Declaration ,initilization of array :

```
datatype[] varname= {value , value , value}
```

## Q- How to add/store data in array ??

=>  sty:

```
datatype[] varname = new datatype[size];
varname[index] = value ;
```

## Q- How to access/get data from array ??

=> sty:

```
varname[index]
```

## Q- How to iterate over array ??

=> using for loop and foreach loop.

# Array :

## Need :

-> To represent group of values/elements as a single entity.

## define array :

-> array is a collection of similar data type of elements.

## Advantages :

-> code optimization :
    - we can sort and retrieve elements efficiently.
-> random access
    - array support for indexing. we can get random data using index.

## Disadvantages :

-> fixed in size.

## IMP Points :

1) array can store similar datatype elements only.
2) array is fixed in size.
3) array used 0 base indexing.
4) array elements stored in contiguous memory locations.
5) using indexing we can access random element from array.
6) array is a referenced datatype.

## # Example :

```
int[] number = new int[5] ;
number[0] = 10 ;
number[1] = 20 ;
number[2] = 30 ;
number[3] = 40 ;
number[4] = 50 ;
```

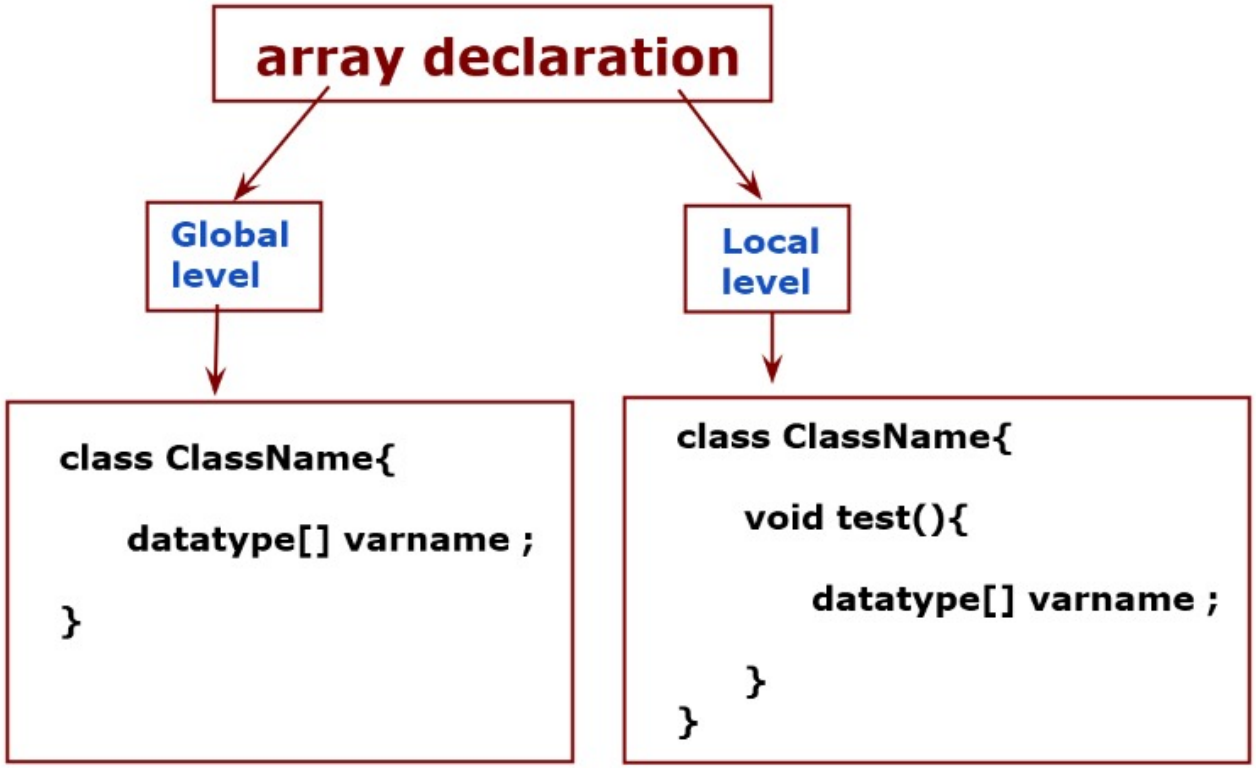
## # Memory representation of array :

# # foreach loop :

```
for( datatype varname  : sequence){

      // body

}
```

# # Explanation :

-> for each element peresent in sequnce foreach loop body will excute.
-> taking out one by one element from a sequnce and for each element excute body.


array declaration
Global level
Local level
class ClassName{
datatype[] varname ;
}
class ClassName{
void test(){
datatype[] varname ;
}
}

# # interface :

-> Like a class , an interface is also referenced datatype.
-> Like a class , an interface is used to represent real world entites.
-> interface is a fully unimplemented class.
-> interfaces are simillar to abstract class but having all the methods of abstract type.
-> interfaces are the blueprint of the class.It specify what a class must do and not how.

# # IMP Points :

1) interfaces are used to achive abstraction.
2) By using interfaces we can achive 100% abstraction.
3) By using interfaces we can achive full abstraction in java.
4) By using interfaces we can achive "multiple inheritance" in java.
5) we can not create object of interface.
6) all methods in an interface is abstract.

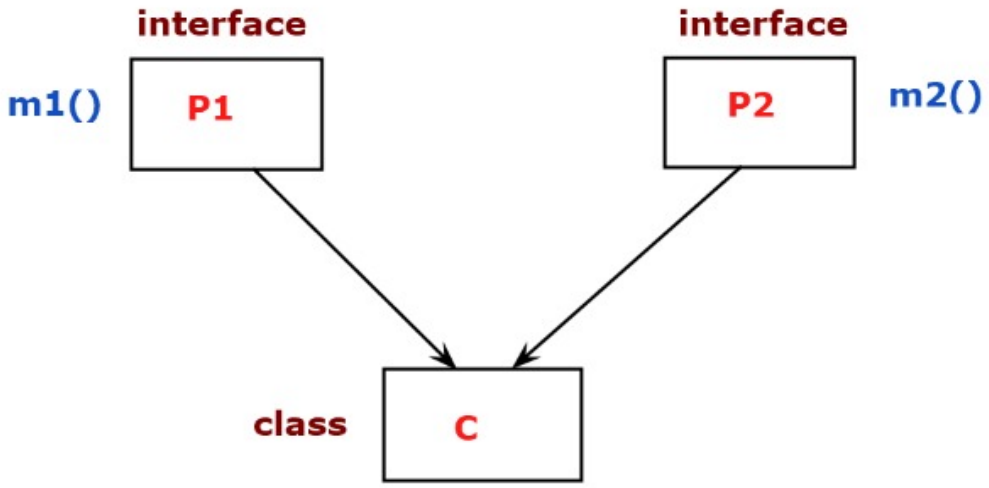
# # Sty of interface :

```
interface  InterfaceName{
        public static final VARIABLEs
        public abstract METHODS
}
```

# # NOTES :

1) interface does not contains any constructor.
2) interface can not contains non-static/instance variables.
3) By default interface can contains public abstract methods.
4) By default interface can contains public static final variable.
5) we have to implements all abstract methods in child-classes(sub-classes).

# # Multiple Inheritance :




```
C c = new C() ;
c.m1() ;
c.m2() ;
```

**+ operator is used on string to concate(combine/join) two strings.**

```
String s1 = "om" ;
String s2 = "sai" ;

String s3 = s1 + s2 ;

sopln(s3) // "om sai"
```

# # String :

-> String is a non-primitive datatype.
-> String is a sequence of characters enclosed in " ".
-> String is a array of characters.
-> In java String is basically an object that represents sequence of characters.
-> an array of characters works same as String.
-> String class implements CharSequence , Serializable , Comparable interfaces.
-> String class is final so we can not extends string class.

```
public final String extends Object implements CharSequence , Serializable , Comparable{
    ------
    // methods
    ------
}
```

-> in java strings class is immutable.
-> There are four ways to create string in java.
    A] using String literal
    B] using new keyword
    C] using StringBuffer class
    D] using StringBuilder class

## # NOTE :

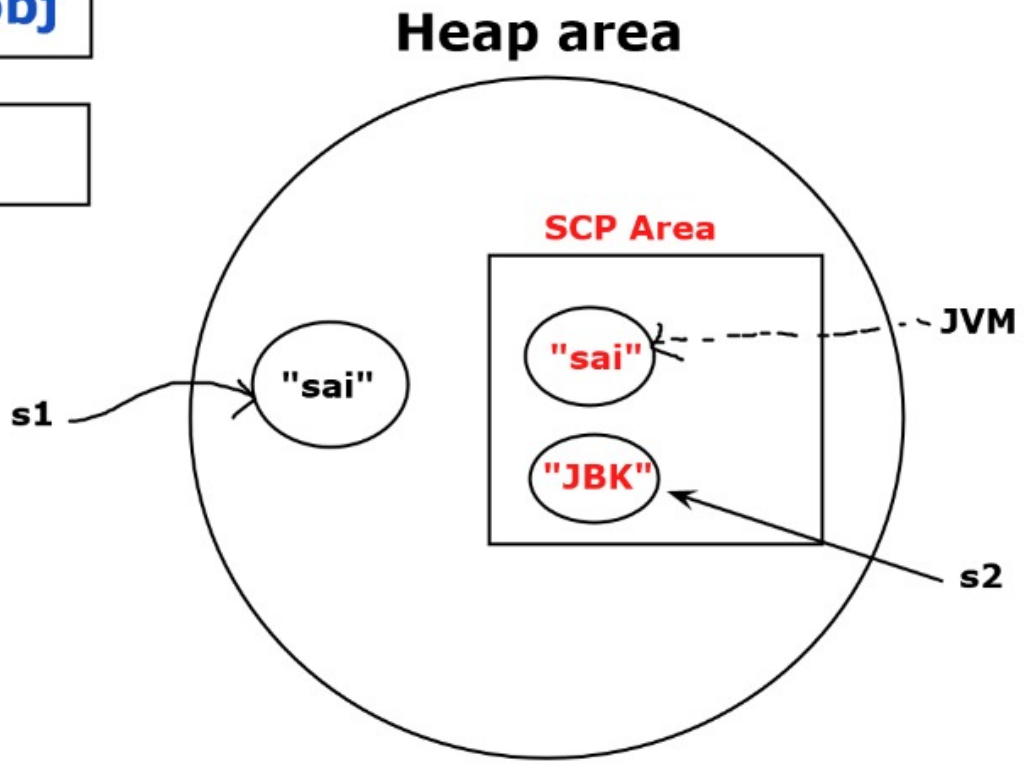
1) characters are enclosed in ' '.
2) String are enclosed in " ".

**+ operator is used on string to concate(combine/join) two strings.**

```
String s1 = "om" ;
String s2 = "sai" ;

String s3 = s1 + s2 ;

sopln(s3) // "om sai"
```

```
String s1 = new String("sai") ; // 2 obj
```

```
String s2 = "JBK" ; // 1 obj
```



## Q- How to create strings in java ??

=> **A] using String literal :**

```
String varName = "charcaters" ;
```

**B] using new keyword :**

```
String varName = new String("String literal")
```

# # String is Immutable ??

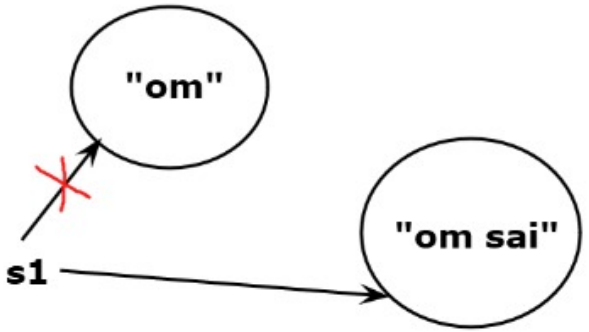
-> immutable means non-changable.

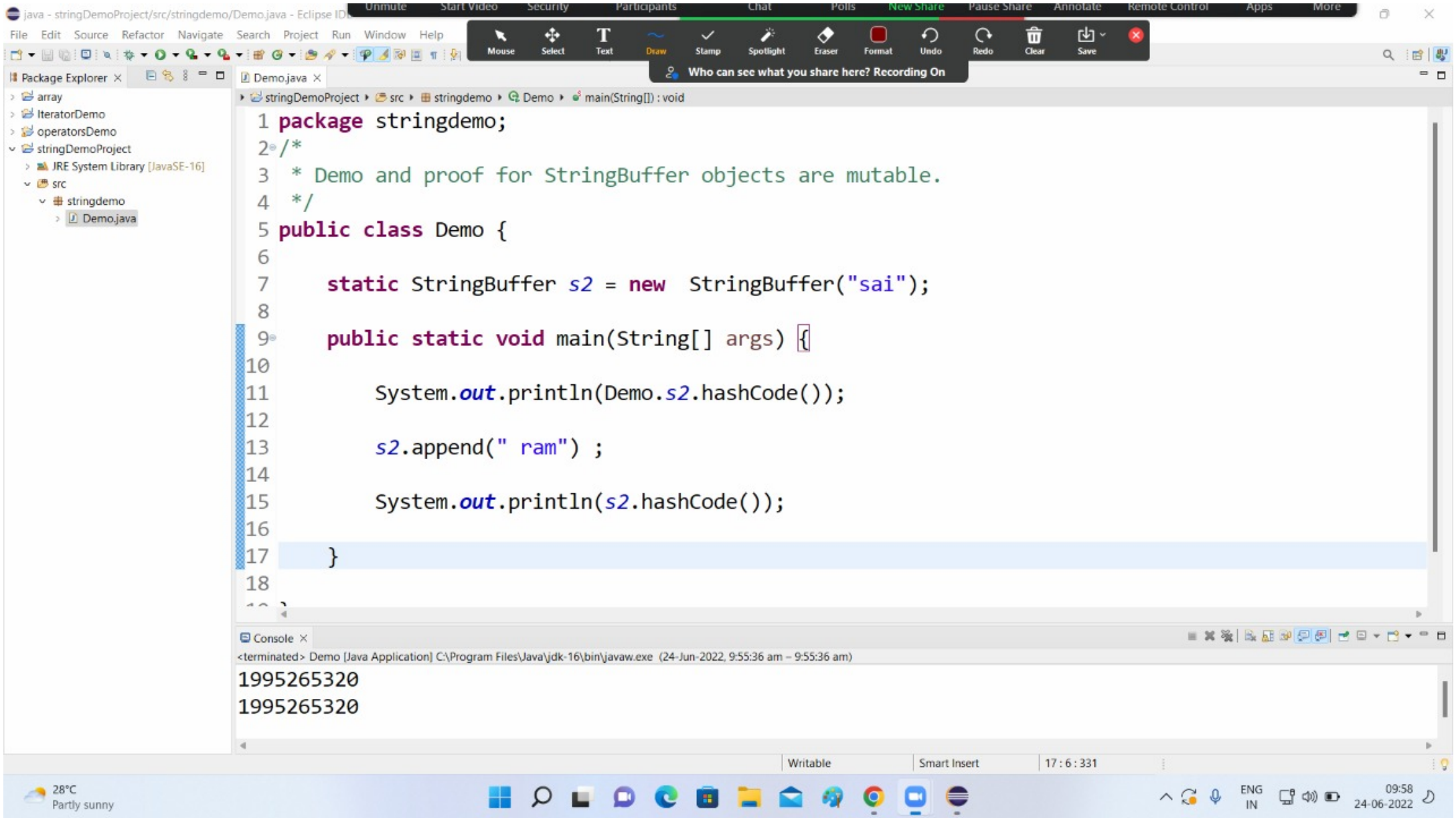
-> once we create an object there is no way to change in that object if we are trying to chage in that object by default new object got created with thoes changes called as immutable objects.

## # Example :

String s1 = "om" ;

String s1 = "om sai" ;



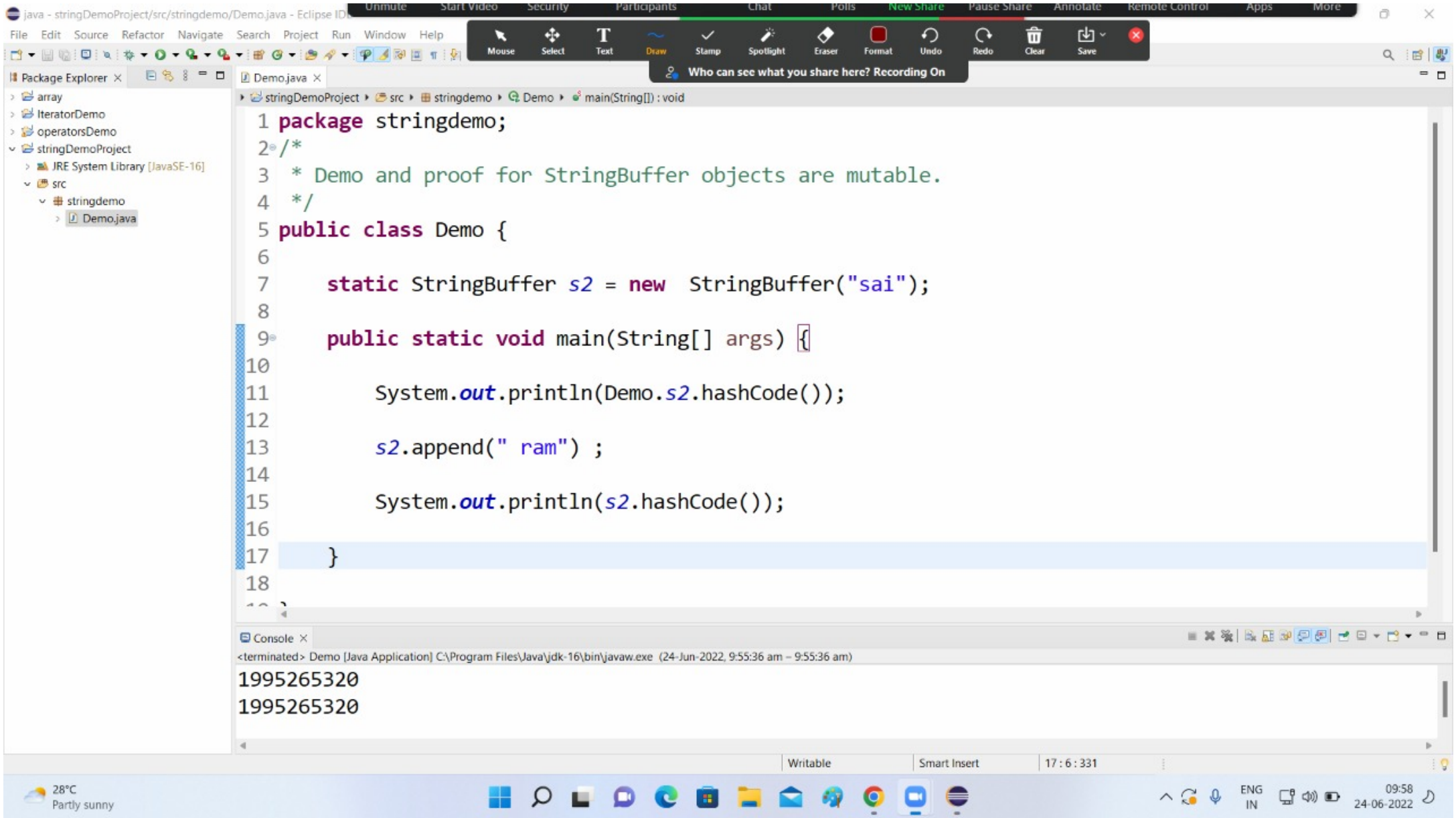

java - stringDemoProject/src/stringdemo/Demo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Unmute Start Video Security Participants Chat Polls New Share Pause Share Annotate Remote Control Apps More
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On
Package Explorer
array
IteratorDemo
operatorsDemo
stringDemoProject
JRE System Library [JavaSE-16]
src
stringdemo
Demo.java
Demo.java
stringDemoProject ▸ src ▸ stringdemo ▸ Demo ▸ main(String[]) : void
1 package stringdemo;
2 /*
3 * Demo and proof for StringBuffer objects are mutable.
4 */
5 public class Demo {
6
7 static StringBuffer s2 = new StringBuffer("sai");
8
9 public static void main(String[] args) {
10
11 System.out.println(Demo.s2.hashCode());
12
13 s2.append(" ram") ;
14
15 System.out.println(s2.hashCode());
16
17 }
18
Console
<terminated> Demo [Java Application] C:\Program Files\Java\jdk-16\bin\javaw.exe (24-Jun-2022, 9:55:36 am – 9:55:36 am)
1995265320
1995265320
Writable
Smart Insert
17 : 6 : 331
28°C
Partly sunny
ENG
IN
09:58
24-06-2022

# # equlas() method on String :

-> It is used to compare content of two strings.
-> if the content of two string are same then it will return true,else it will return false
-> equlas method will compare characters of two given strings.

# # Example :

```
String s1 = "Sai" ;
String s2 = "sai" ;
String s3 = "ram"

sopln( s1.equlas(s2) ) ; // true
sopln( s1.equlas(s3) ); // false
```

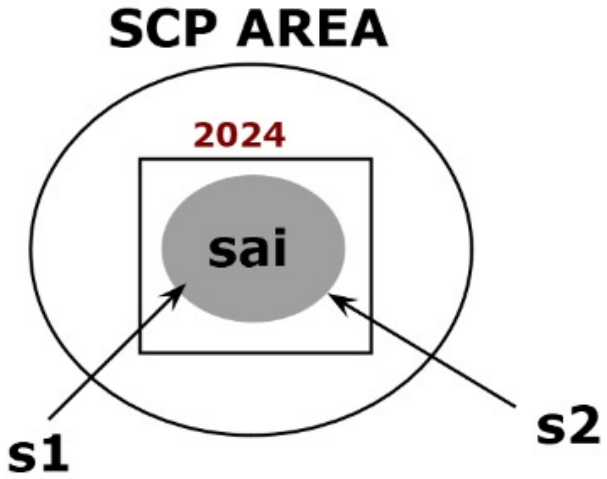

java - stringDemoProject/src/stringdemo/Demo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Unmute Start Video Security Participants Chat Polls New Share Pause Share Annotate Remote Control Apps More
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On
Package Explorer
array
IteratorDemo
operatorsDemo
stringDemoProject
JRE System Library [JavaSE-16]
src
stringdemo
Demo.java
Demo.java
stringDemoProject ▸ src ▸ stringdemo ▸ Demo ▸ main(String[]) : void
1 package stringdemo;
2 /*
3 * Demo and proof for StringBuffer objects are mutable.
4 */
5 public class Demo {
6
7 static StringBuffer s2 = new StringBuffer("sai");
8
9 public static void main(String[] args) {
10
11 System.out.println(Demo.s2.hashCode());
12
13 s2.append(" ram") ;
14
15 System.out.println(s2.hashCode());
16
17 }
18
Console
<terminated> Demo [Java Application] C:\Program Files\Java\jdk-16\bin\javaw.exe (24-Jun-2022, 9:55:36 am – 9:55:36 am)
1995265320
1995265320
Writable
Smart Insert
17 : 6 : 331
28°C
Partly sunny
ENG
IN
09:58
24-06-2022

## == operator on Strings :

-> if both reference variables are pointing to same memory locations/same object then == operator will retrun true value,else it will return false.
-> It is used to compare memory address of reference variables.

## Example :

```
String s1 = "sai" ;
String s2 = "sai" ;

sopln(s1 == s2); // true
```



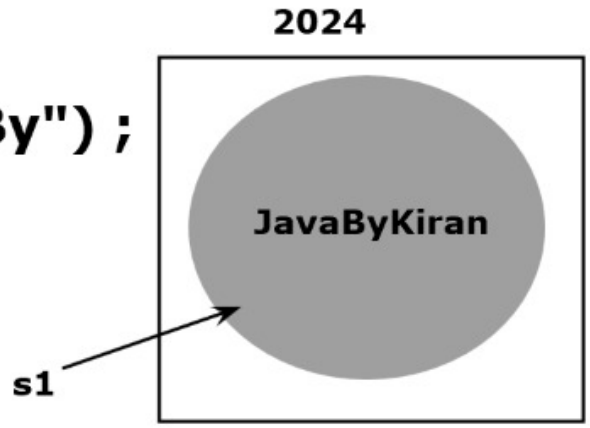
# # StringBuffer and StringBuilder class in java :

-> StringBuffer and StringBuilder both classes are mutable.
-> mutable means chanagable.

# # Example :

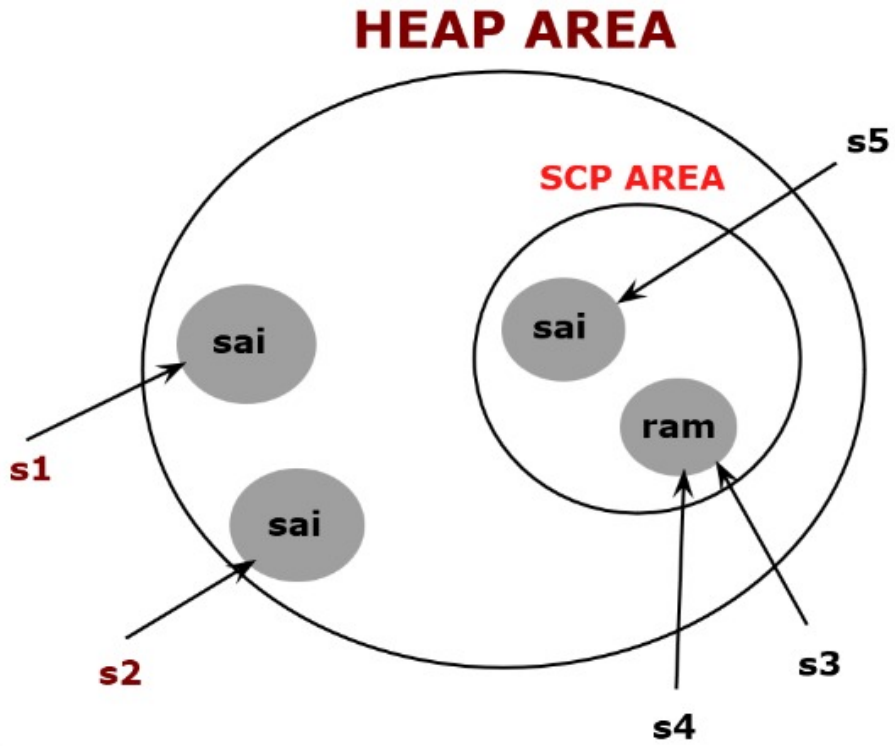
```
StringBuffer s1 = new StringBuffer("JavaBy") ;
s1.append("Kiran") ;
```

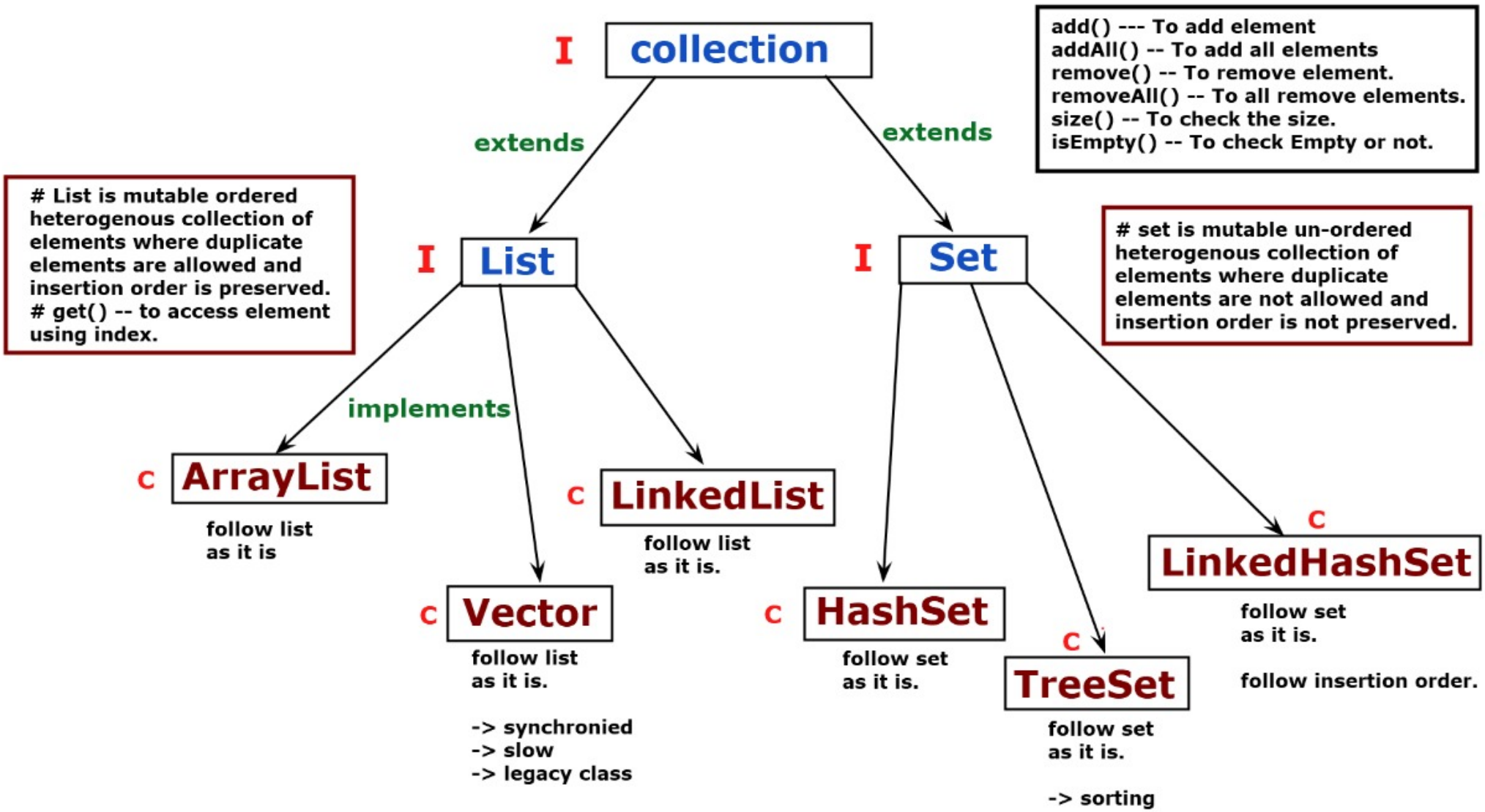
```
StringBuilder s2 = new StringBuilder("JavaBy") ;
s2.append("Kiran") ;
```

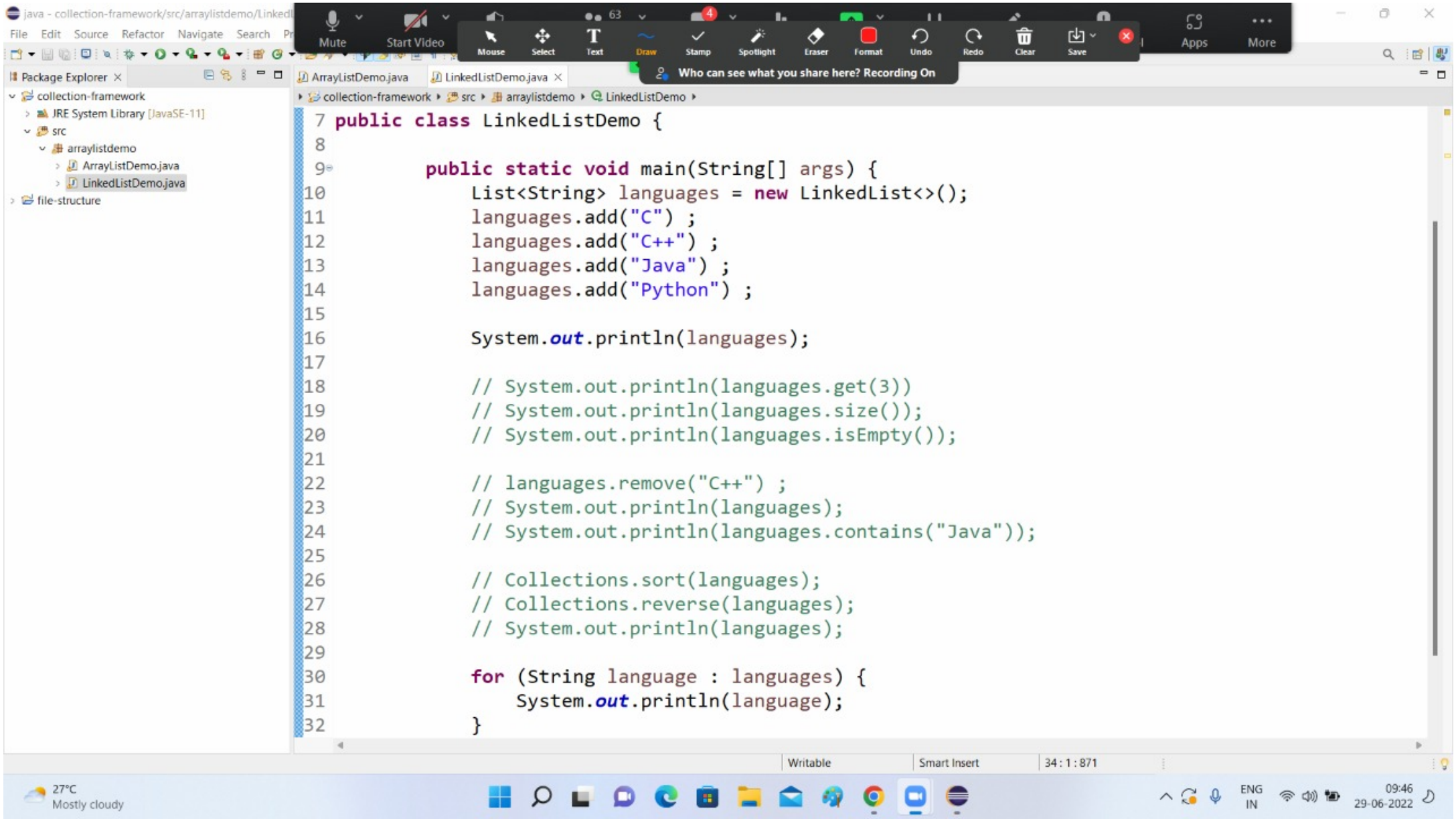


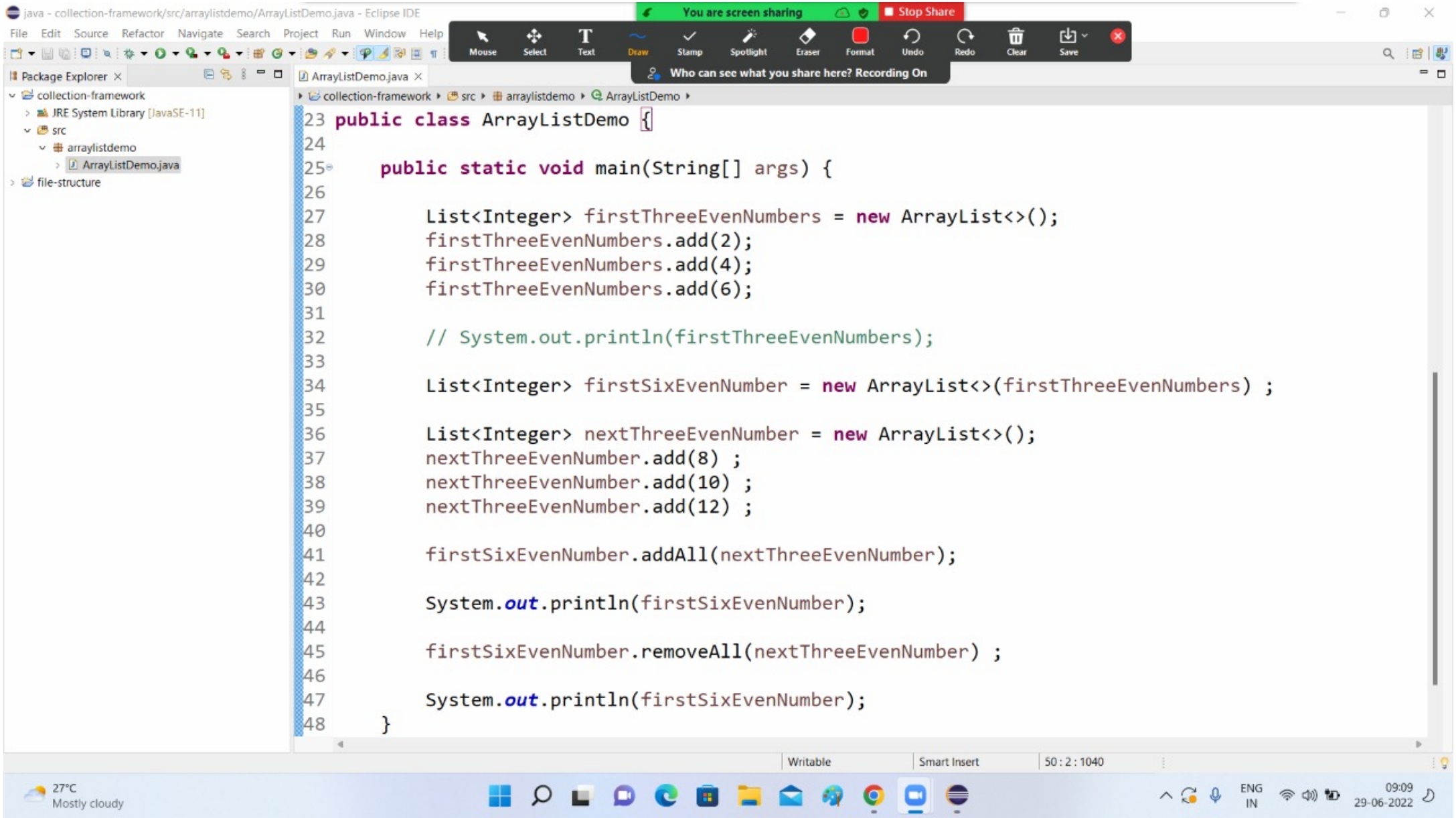
# # String :

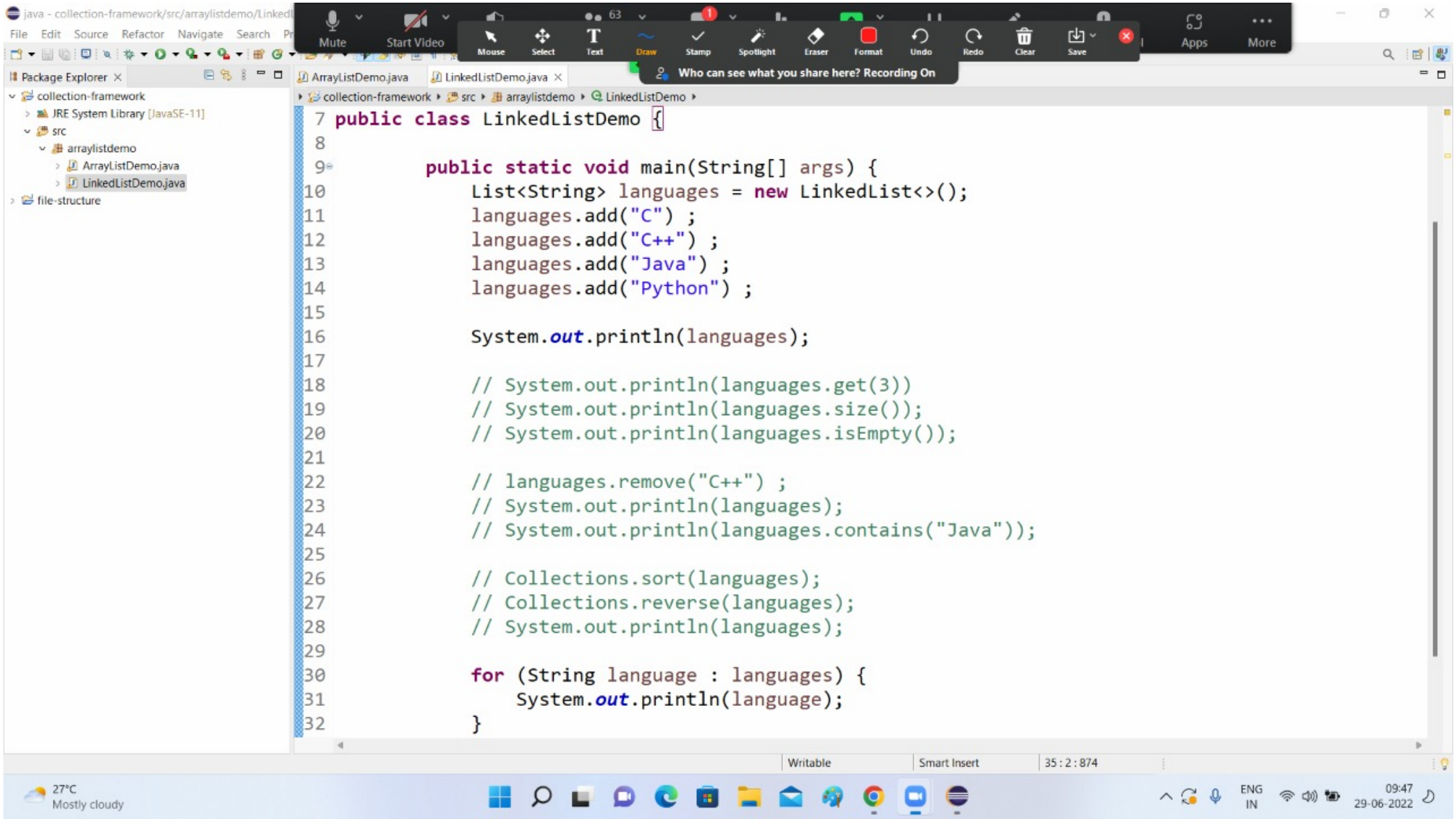
```
String s1 = new String("sai");

String s2 = new String("sai");

String s3 = "ram" ;

String s4 = "ram" ;

String s5 = "sai" ;

sopln(s4 == s3) // true
sopln(s1 == s2) // false
```

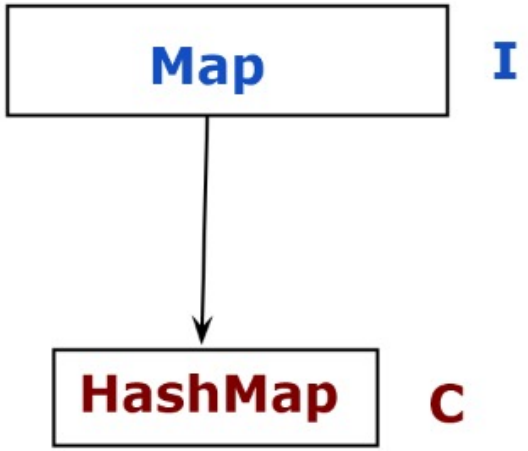



I collection
add() --- To add element
addAll() -- To add all elements
remove() -- To remove element.
removeAll() -- To all remove elements.
size() -- To check the size.
isEmpty() -- To check Empty or not.
extends
extends
# List is mutable ordered heterogenous collection of elements where duplicate elements are allowed and insertion order is preserved.
# get() -- to access element using index.
I List
I Set
# set is mutable un-ordered heterogenous collection of elements where duplicate elements are not allowed and insertion order is not preserved.
implements
c ArrayList
follow list as it is
c LinkedList
follow list as it is.
c Vector
follow list as it is.
-> synchronied
-> slow
-> legacy class
c HashSet
follow set as it is.
c TreeSet
follow set as it is.
-> sorting
c LinkedHashSet
follow set as it is.
follow insertion order.


java - collection-framework/src/arraylistdemo/LinkedL
File Edit Source Refactor Navigate Search Pr
Mute
Start Video
Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Clear
Save
Apps
More
Who can see what you share here? Recording On
Package Explorer
collection-framework
JRE System Library [JavaSE-11]
src
arraylistdemo
ArrayListDemo.java
LinkedListDemo.java
file-structure
ArrayListDemo.java
LinkedListDemo.java
collection-framework ▸ src ▸ arraylistdemo ▸ LinkedListDemo ▸
7 public class LinkedListDemo {
8
9 public static void main(String[] args) {
10 List<String> languages = new LinkedList<>();
11 languages.add("C") ;
12 languages.add("C++") ;
13 languages.add("Java") ;
14 languages.add("Python") ;
15
16 System.out.println(languages);
17
18 // System.out.println(languages.get(3))
19 // System.out.println(languages.size());
20 // System.out.println(languages.isEmpty());
21
22 // languages.remove("C++") ;
23 // System.out.println(languages);
24 // System.out.println(languages.contains("Java"));
25
26 // Collections.sort(languages);
27 // Collections.reverse(languages);
28 // System.out.println(languages);
29
30 for (String language : languages) {
31 System.out.println(language);
32 }
Writable
Smart Insert
34 : 1 : 871
27°C
Mostly cloudy
ENG
IN
09:46
29-06-2022


java - collection-framework/src/arraylistdemo/ArrayListDemo.java - Eclipse IDE
You are screen sharing
Stop Share
File Edit Source Refactor Navigate Search Project Run Window Help
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On
Package Explorer
collection-framework
JRE System Library [JavaSE-11]
src
arraylistdemo
ArrayListDemo.java
file-structure
ArrayListDemo.java
collection-framework ▸ src ▸ arraylistdemo ▸ ArrayListDemo ▸
23 public class ArrayListDemo {
24
25 public static void main(String[] args) {
26
27 List<Integer> firstThreeEvenNumbers = new ArrayList<>();
28 firstThreeEvenNumbers.add(2);
29 firstThreeEvenNumbers.add(4);
30 firstThreeEvenNumbers.add(6);
31
32 // System.out.println(firstThreeEvenNumbers);
33
34 List<Integer> firstSixEvenNumber = new ArrayList<>(firstThreeEvenNumbers) ;
35
36 List<Integer> nextThreeEvenNumber = new ArrayList<>();
37 nextThreeEvenNumber.add(8) ;
38 nextThreeEvenNumber.add(10) ;
39 nextThreeEvenNumber.add(12) ;
40
41 firstSixEvenNumber.addAll(nextThreeEvenNumber);
42
43 System.out.println(firstSixEvenNumber);
44
45 firstSixEvenNumber.removeAll(nextThreeEvenNumber) ;
46
47 System.out.println(firstSixEvenNumber);
48 }
Writable
Smart Insert
50 : 2 : 1040
27°C
Mostly cloudy
ENG
IN
09:09
29-06-2022


java - collection-framework/src/arraylistdemo/LinkedL
File Edit Source Refactor Navigate Search Pr
Mute
Start Video
Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Clear
Save
Apps
More
Who can see what you share here? Recording On
Package Explorer
collection-framework
JRE System Library [JavaSE-11]
src
arraylistdemo
ArrayListDemo.java
LinkedListDemo.java
file-structure
ArrayListDemo.java
LinkedListDemo.java
collection-framework ▸ src ▸ arraylistdemo ▸ LinkedListDemo ▸
7 public class LinkedListDemo {
8
9 public static void main(String[] args) {
10 List<String> languages = new LinkedList<>();
11 languages.add("C") ;
12 languages.add("C++") ;
13 languages.add("Java") ;
14 languages.add("Python") ;
15
16 System.out.println(languages);
17
18 // System.out.println(languages.get(3))
19 // System.out.println(languages.size());
20 // System.out.println(languages.isEmpty());
21
22 // languages.remove("C++") ;
23 // System.out.println(languages);
24 // System.out.println(languages.contains("Java"));
25
26 // Collections.sort(languages);
27 // Collections.reverse(languages);
28 // System.out.println(languages);
29
30 for (String language : languages) {
31 System.out.println(language);
32 }
Writable
Smart Insert
35 : 2 : 874
27°C
Mostly cloudy
ENG
IN
09:47
29-06-2022




# IMP Points :

1) HashMap is a hash table base implementation class of java Map interface.
2) HashMap is un-ordered collection of elements where elements are stored in key value pair.
3) HashMap is not thread-safe.

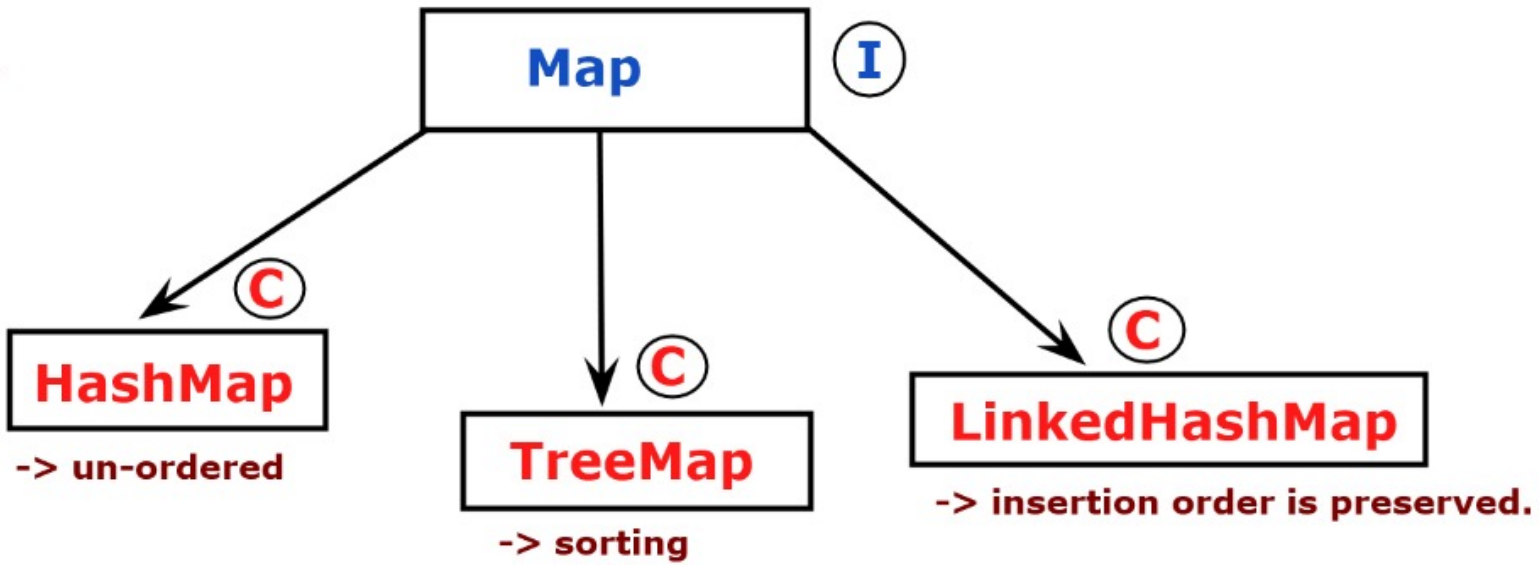
# NOTE :

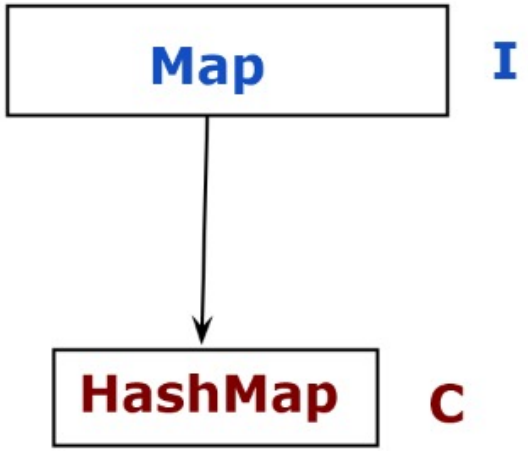
1) HashMap store collection of elements in key value formate.
2) Key must be unique.duplicate key is not allowed.
3) Value can be duplicate.

# Map :

-> representation :

{ K = V , K = V }



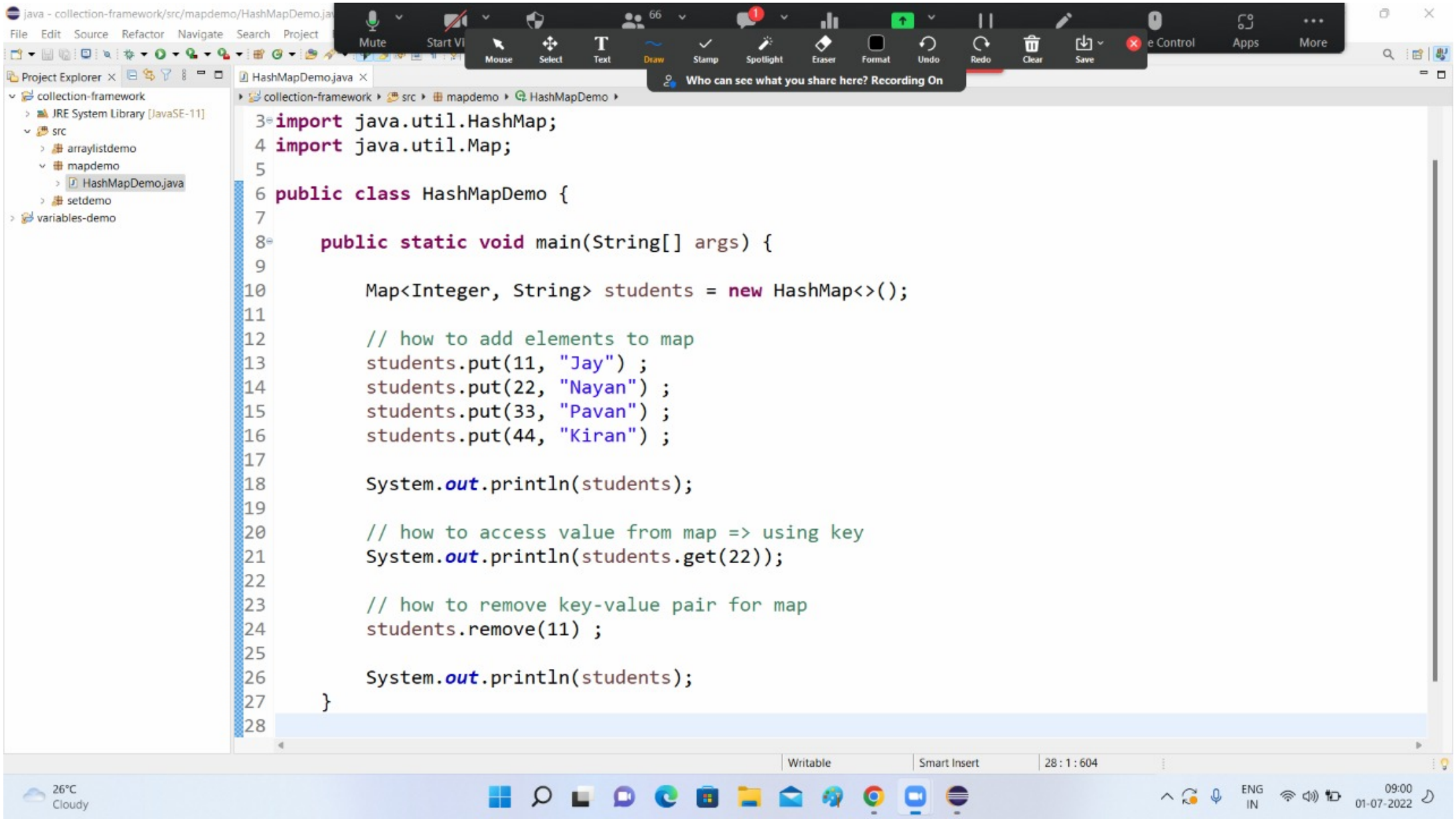


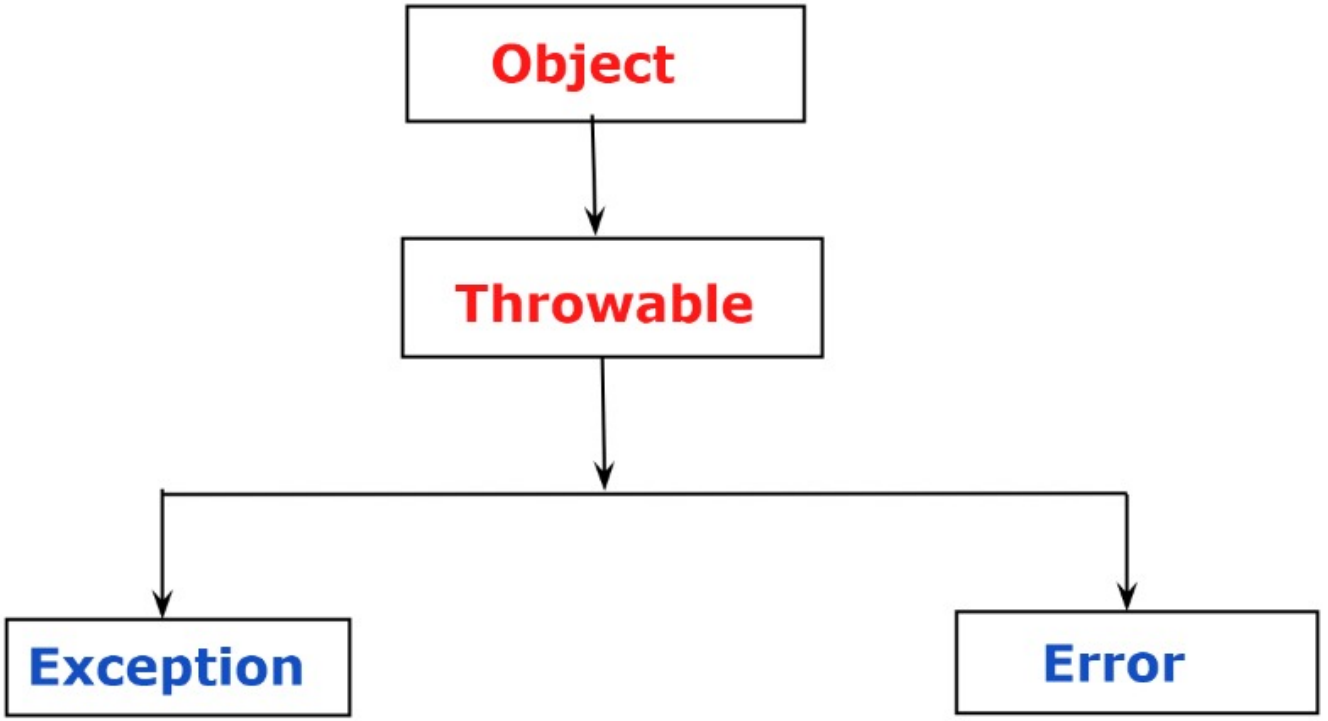

# IMP Points :

1) HashMap is a hash table base implementation class of java Map interface.
2) HashMap is un-ordered collection of elements where elements are stored in key value pair.
3) HashMap is not thread-safe.

# NOTE :

1) HashMap store collection of elements in key value formate.
2) Key must be unique.duplicate key is not allowed.
3) Value can be duplicate.


java - collection-framework/src/mapdemo/HashMapDemo.java
File Edit Source Refactor Navigate Search Project
Mute
Start Video
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On
Apps
More
Project Explorer
collection-framework
JRE System Library [JavaSE-11]
src
arraylistdemo
mapdemo
HashMapDemo.java
setdemo
variables-demo
HashMapDemo.java
collection-framework ▸ src ▸ mapdemo ▸ HashMapDemo ▸
3 import java.util.HashMap;
4 import java.util.Map;
5
6 public class HashMapDemo {
7
8 public static void main(String[] args) {
9
10 Map<Integer, String> students = new HashMap<>();
11
12 // how to add elements to map
13 students.put(11, "Jay") ;
14 students.put(22, "Nayan") ;
15 students.put(33, "Pavan") ;
16 students.put(44, "Kiran") ;
17
18 System.out.println(students);
19
20 // how to access value from map => using key
21 System.out.println(students.get(22));
22
23 // how to remove key-value pair for map
24 students.remove(11) ;
25
26 System.out.println(students);
27 }
28
Writable
Smart Insert
28 : 1 : 604
26°C
Cloudy
ENG
IN
09:00
01-07-2022




1) exceptions are occurs becoz of programmer.
2) we can recover exception.
3) Types of exceptions.

A] Compile Time Excepion (Checked Exceptions)
B] Run Time Exception (un-checked Exceptions)

1) error's are occurs becoz of system resources.
2) we can't recover error's
3) Types of Error.

A] Run Time Error

# # Exception Handling

## # Exception :

-> an un-wanted un-expected event wich occurs at run-time that distrubs the normal flow of programme.

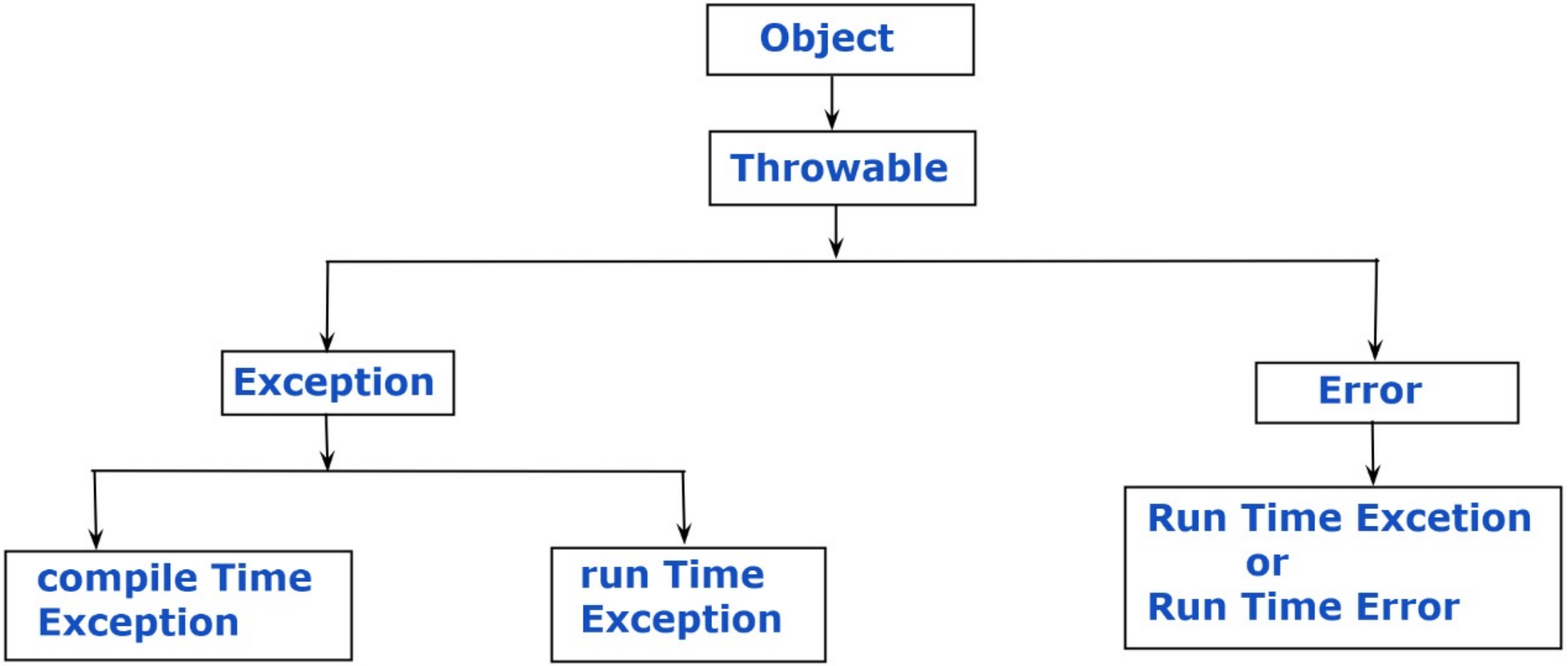
-> Abnormal Termination(AT)

## # Execption Handling :

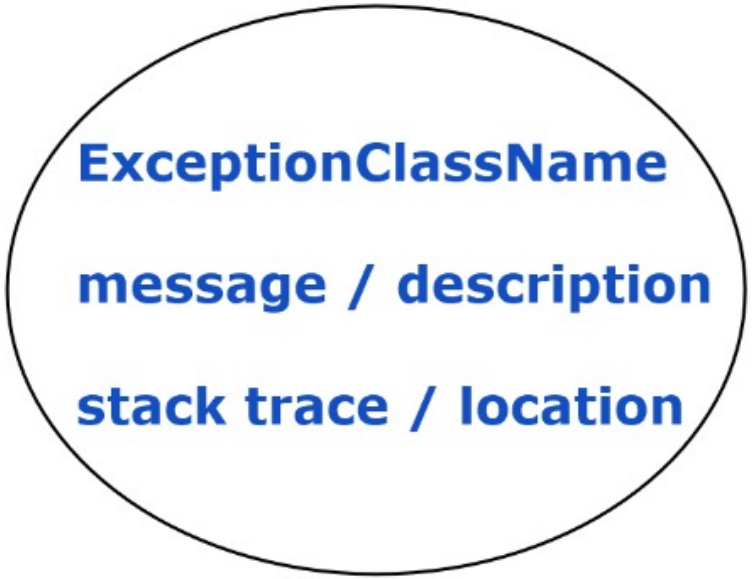
-> defining the alternative way to continue rest of the my programme so that my programme terminates normally.

-> defining the alternative way to recover the exception.

-> Normal Termination(NT)


Object
Throwable
Exception
Error
compile Time
Exception
run Time
Exception
Run Time Excetion
or
Run Time Error

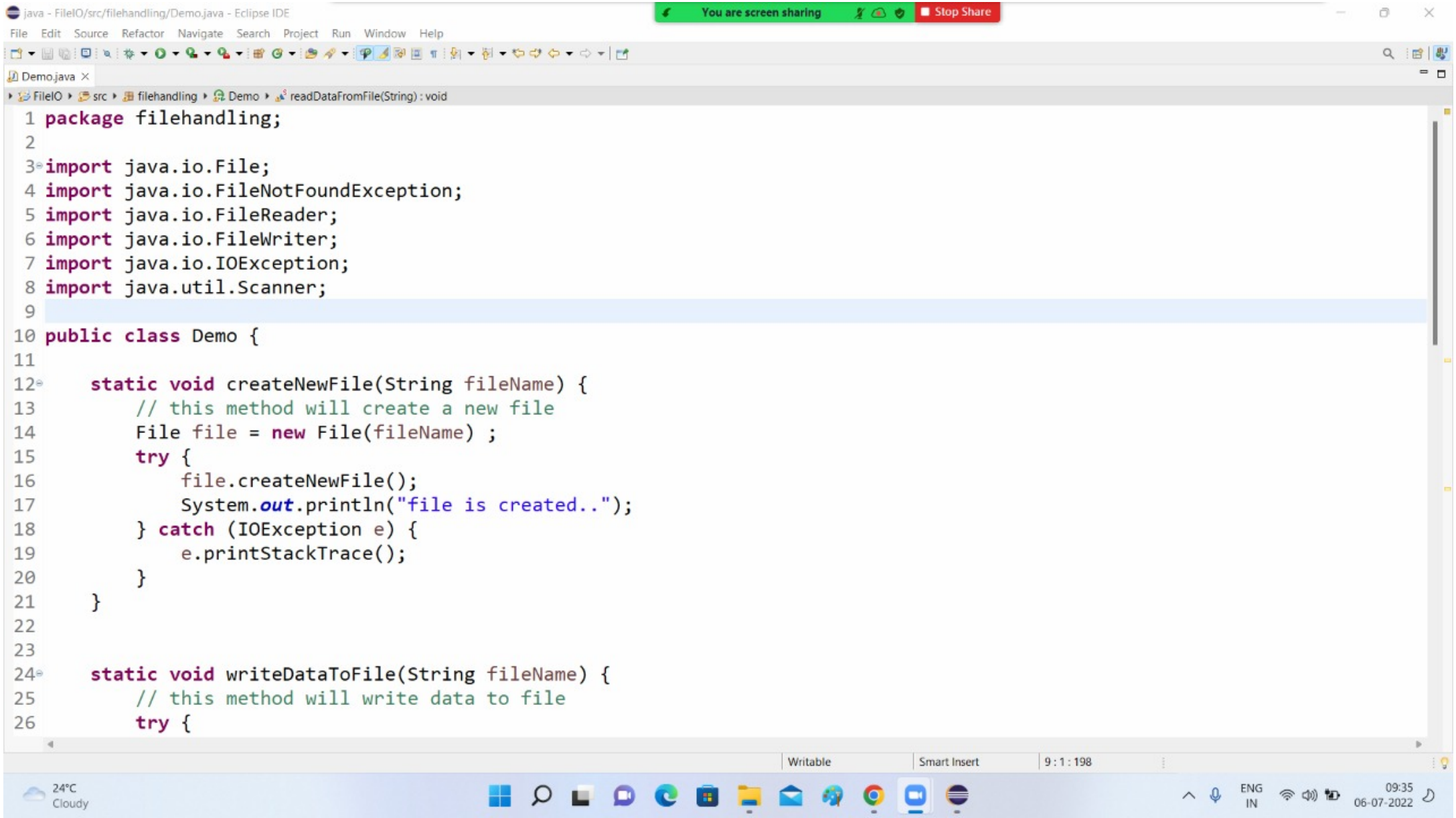
# Exception Object :

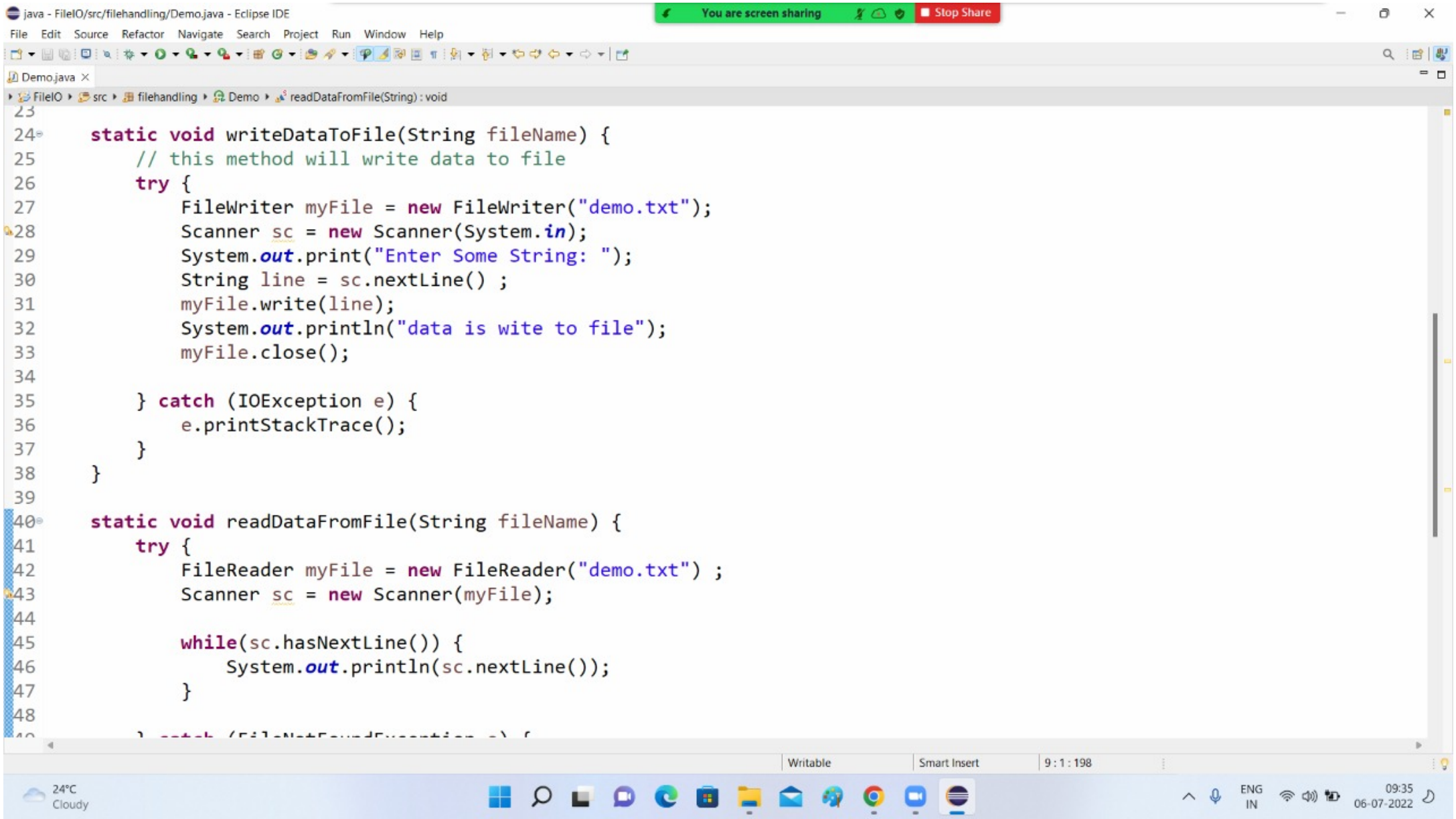


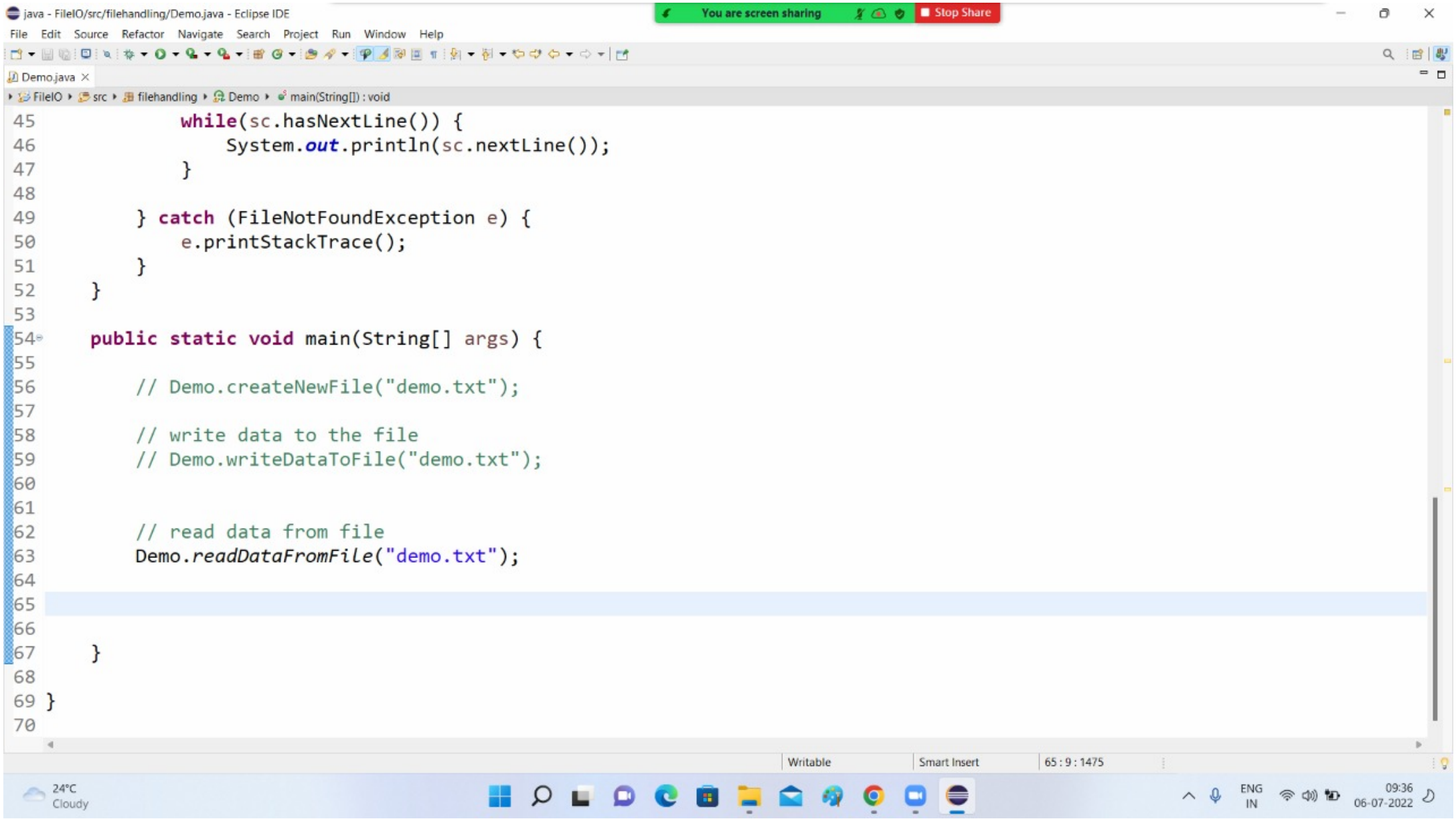
# # Exception Handling...

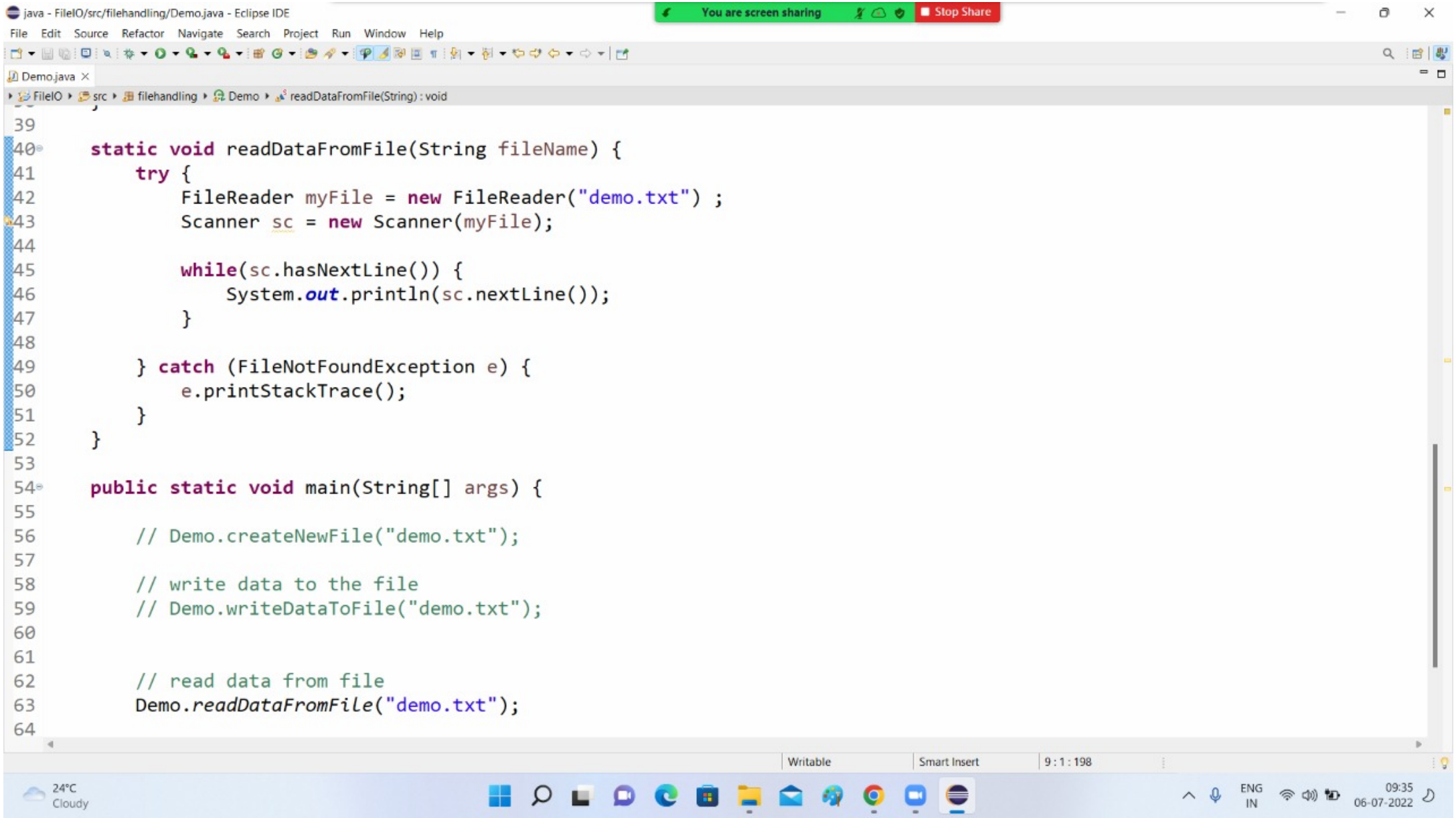
```
try{
    // risky code
    // the code wich may raise an exception
    // try bock execute always.
}
catch Exception objRef{
    // Exception handling/recover code.
    // if there is Exception in try in try block
    catch block will execute.
}
```

```
try{
    // risky code
    // the code wich may raise an exception
    // try bock execute always.
}
catch ExceptionName objRef{
    // Exception handling/recover code.
    // if there is Exception in try in try block
    catch block will execute.
}
```


java - FileIO/src/filehandling/Demo.java - Eclipse IDE
You are screen sharing
Stop Share
File Edit Source Refactor Navigate Search Project Run Window Help
Demo.java
FileIO ▸ src ▸ filehandling ▸ Demo ▸ readDataFromFile(String) : void
1 package filehandling;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.io.FileReader;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.util.Scanner;
9
10 public class Demo {
11
12 static void createNewFile(String fileName) {
13 // this method will create a new file
14 File file = new File(fileName) ;
15 try {
16 file.createNewFile();
17 System.out.println("file is created..");
18 } catch (IOException e) {
19 e.printStackTrace();
20 }
21 }
22
23
24 static void writeDataToFile(String fileName) {
25 // this method will write data to file
26 try {
Writable
Smart Insert
9 : 1 : 198
24°C
Cloudy
ENG
IN
09:35
06-07-2022


java - FileIO/src/filehandling/Demo.java - Eclipse IDE
You are screen sharing
Stop Share
File Edit Source Refactor Navigate Search Project Run Window Help
Demo.java
FileIO ▸ src ▸ filehandling ▸ Demo ▸ readDataFromFile(String) : void
23
24 static void writeDataToFile(String fileName) {
25 // this method will write data to file
26 try {
27 FileWriter myFile = new FileWriter("demo.txt");
28 Scanner sc = new Scanner(System.in);
29 System.out.print("Enter Some String: ");
30 String line = sc.nextLine() ;
31 myFile.write(line);
32 System.out.println("data is wite to file");
33 myFile.close();
34
35 } catch (IOException e) {
36 e.printStackTrace();
37 }
38 }
39
40 static void readDataFromFile(String fileName) {
41 try {
42 FileReader myFile = new FileReader("demo.txt") ;
43 Scanner sc = new Scanner(myFile);
44
45 while(sc.hasNextLine()) {
46 System.out.println(sc.nextLine());
47 }
48
Writable
Smart Insert
9 : 1 : 198
24°C
Cloudy
ENG
IN
09:35
06-07-2022


java - FileIO/src/filehandling/Demo.java - Eclipse IDE
You are screen sharing
Stop Share
File Edit Source Refactor Navigate Search Project Run Window Help
Demo.java
FileIO ▸ src ▸ filehandling ▸ Demo ▸ main(String[]) : void
45 while(sc.hasNextLine()) {
46 System.out.println(sc.nextLine());
47 }
48
49 } catch (FileNotFoundException e) {
50 e.printStackTrace();
51 }
52 }
53
54 public static void main(String[] args) {
55
56 // Demo.createNewFile("demo.txt");
57
58 // write data to the file
59 // Demo.writeDataToFile("demo.txt");
60
61
62 // read data from file
63 Demo.readDataFromFile("demo.txt");
64
65
66
67 }
68
69 }
70
Writable
Smart Insert
65 : 9 : 1475
24°C
Cloudy
ENG
IN
09:36
06-07-2022


java - FileIO/src/filehandling/Demo.java - Eclipse IDE
You are screen sharing
Stop Share
File Edit Source Refactor Navigate Search Project Run Window Help
Demo.java
FileIO ▸ src ▸ filehandling ▸ Demo ▸ readDataFromFile(String) : void
39
40 static void readDataFromFile(String fileName) {
41 try {
42 FileReader myFile = new FileReader("demo.txt") ;
43 Scanner sc = new Scanner(myFile);
44
45 while(sc.hasNextLine()) {
46 System.out.println(sc.nextLine());
47 }
48
49 } catch (FileNotFoundException e) {
50 e.printStackTrace();
51 }
52 }
53
54 public static void main(String[] args) {
55
56 // Demo.createNewFile("demo.txt");
57
58 // write data to the file
59 // Demo.writeDataToFile("demo.txt");
60
61
62 // read data from file
63 Demo.readDataFromFile("demo.txt");
64
Writable
Smart Insert
9 : 1 : 198
24°C
Cloudy
ENG
IN
09:35
06-07-2022