1. ***Arrays:***
2. ***How to declare an array:***
3. int a[] = new int[5] or int a[] = {1,2,3,4,5}; second one automatically allocate space in memory such that we can overcome size limitation problem
4. int a[][] = new int[2][2];
5. ***How to store elements in an array:***
6. *a[0] = 1; a[1]=1; //single dimension when size is declared as highlighted in yellow*
7. *int a[]={1,2,3,4,5}; //Single dimension when size is not declared*
8. *int a[][]=new int [1][1];*

*a[0][0]=1;*

*a[0][1]=2;*

*a[1][0]=3;*

*a[1][1]=4;*

1. ***How to iterate through an array:***

***Model 1 using for loop:***

For(int i=0; i<a.length;i++){

Sysout();

}

***Model 2 using for each loop:***

For (int i : a){

Sysout(i);

}

1. **How to get the length of an array:**

Syntax : Arrayname.length i.e., a.length where ‘a’ is the array name

1. **Limitations and ways to overcome:**
2. Size is fixed 🡺 to overcome this use int a[] = {};
3. Diff data types can’t be used 🡺 to overcome this use object class as Object a[] = new Object[4];

A[0]=1;

A[1]=”Welcome”;

A[2]=’c’;

A[3]=100.12;

Note: the number inside the brackets indicate the number of elements it can hold, and the storage place starts from 0th position hence the number of last element is always less than array size

1. **Two/multi dimension array declaration:**

Int a[][]=new int[2][2];

A[0][0]=1;a[0][1]=2;a[1][0]=3;a[1][1]=4;

Int a[][]={{1,2},{3,4}};

1. ***Strings:***

*String s = “Welcome”; String s1=”welcome”;*

a. **concatenate** strings 🡪 Sysout(s+s1) or sysou(s.concat(s1));

b. how to find **length** of a string 🡪 s.length();

c. **equals** 🡪 sysout(s.equals(s1)); 🡪 returns false bcoz s1 is in lower case. To overcome this use “d”

d. **equalsignorecase** 🡪 sysout(s.equalsIgnoreCase(s1)); 🡪 returns true

e. **contains** 🡪 sysout(s.contains(“Wel”)); 🡪 returns true

f. **substring** 🡪 substring(starting index, ending index). So sysout(s.substring(0,3)); 🡪 returns “wel”

0 1 2 3 4 5 6 🡪 Staring index

W E L C O M E 🡪 String S

1 2 3 4 5 6 7 🡪 ending index

1. **replace** 🡪 sysout(s.replace(‘c’,’x’) or sysout(“wel”,”pom”);
2. to remove leading and trailing space in a string -> use trim() or strip() method.
3. To understand the difference between trim and strip refer 🡪 <https://www.youtube.com/watch?v=nHMQJfupbiw>
4. Basically, space can be given in java in multiple ways like space bar and then few Unicode are there like \u2000 etc., refer Unicode charts.
5. ***Classes and Objects:***

Class 🡪 collection of objects / variables and methods

Objects 🡪 Instance of a class

Public class **Ragav**{

Int id; String name

}

1. How to instantiate the class

**Ragav** **variableName** = new Ragav();

1. How to instantiate the value for the class variable

***Using direct method***

With the help of object reference like **variableName**.id=5;

***Using Constructor:***

Declare a constructor such that it takes arguments / parameters

Ragav(Int id1, String name1){

Id=id1;

Name= name1;

}

The value will be assigned while instantiating the class like

**Ragav** variableName = new Ragav(3,”Ragav”);

***Using methods:***

Like constructor but declare a method with parameter

1. ***Method overloading:***

Inside a class, if we declare the method / constructor with same name more than once, then it is called method / constructor overloading. This is used to achieve polymorphism and can be achieved using following rules.

1. Number of parameters should be different either none or 1 or more than 1.
2. If not a, it should not have the same data type for parameters.
3. If not b, order of parameters should be different.

i. Void m1()

ii. Void m1(int a)

iii. Void m1(int a, int b)

iv. Void m1(int a, String b) // iii and iv are example of “b”.

v. void m1(int a, int b, String c)

vi. void m1(int a, String b, int c) // v and vi are example of “c”.

1. ***Constructor:***

\* Special kind of method since it will not have return type and the first letter of the name is in caps

\* The name of constructor will be as same as class name

\* Constructor will be invoked at the time of class initiation / object creation. i.e., no need to call it explicitly.

\* It will take parameters like a method

\* It will not have any logic and mainly used only for initializing the values at the run time

a. ***Types of Constructors:***

i. Default constructor

ii. Parameterized constructor

1. ***Constructor Overloading:***

Follows the same rule as method overloading and to initialize you need to have multiple class initialization / object creation.

d. ***This Keyword:***

It is used to differentiate the class variable from the local variable when both the variable has the same name. “This” keyword will be used along with class variable as in this.a = a

1. ***Static Keyword:***

Used to make a variable or a method as common for all object created.

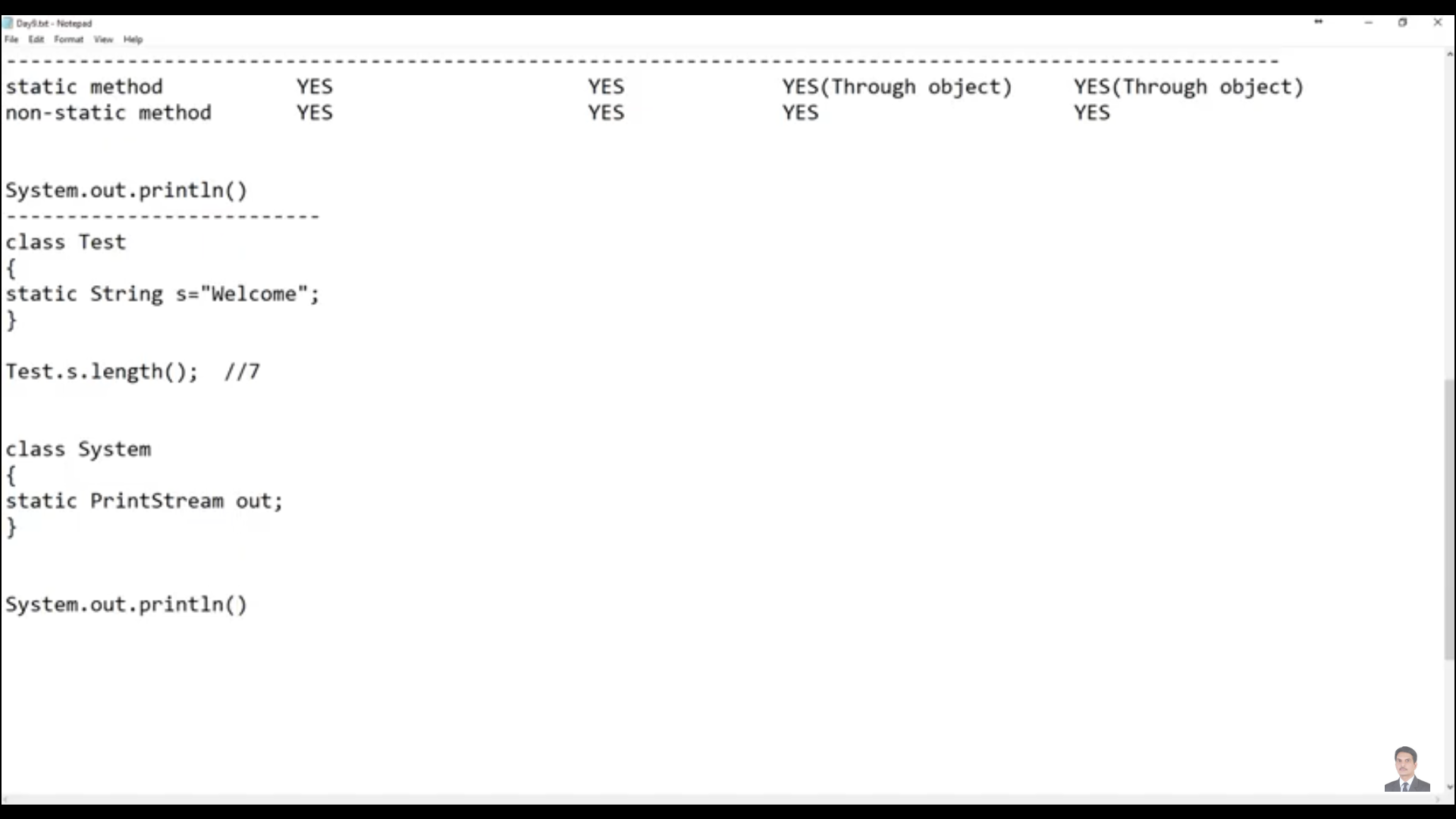
Can be accessed using class name and not by object reference.

Public class Ragav{

Static int a;

Public static void main(String args[]){

Ragav.a=10; or a=10; **// both are valid**}

* Static method can access only static variables and static methods directly without class name when accessing inside the same class or with class name when accessing outside the class and no need to create objects.
* Static method can access non-static using object name.
* Non-static method can access both static and non-static variables and methods directly without using objects.
* 

System 🡪 class name

Out 🡪 static variable of type print stream

Println() 🡪 method of print stream class

4. ***Inheritance:***

Acquiring the properties / methods and variables and behavior of parent class into child class

**Terms**: Parent class or super class or base class and Child class or sub class or derived class

**Single inheritance** 🡪 Class B extends A

**Multi-level inheritance** 🡪 Class B extends A and Class C extends B. So, Class “C” acquires the properties of both A and B.

**Multiple inheritance** 🡪 Class C extends A, B 🡪 not possible and to overcome this we can use interface.

**Hierarchical inheritance** 🡪 Class B extends A and Class C extends A 🡪 when parent has two child class, but two child classes are not related

5. ***Method overriding:***

The method in child class with same as the one in parent class but overrides the internal logic. This will come into picture only when we use inheritance.

**Note**: The output is based on the obj we are using. If it is obj of A, then it will print only parent class else child class. Method definition should be same meaning no parameters or same parameters and data type

For eg.,

Class A{

Void m1(){

Sysout(“Parent class”);

}

}

Class B extends A{

Void m1(){

Sysout(“Child class”);

}

}

B bobj = new B();

Bobj.m1(); 🡪 print “Child class”

**Note**: When we have static method in both parent and child class which are overridden ( actually this is called as overhidden), then the method call is based on class reference and not by object whereas when it is not static, then the method call is based on object type.

Source 🡪 <https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/>

10. ***JDK:*** 🡪 java development kit and will have internally JRE and JVM by default. Used to create / develop java-based applications.

11. ***JRE: 🡪***  java runtime environment. Used to run any java-based application. If you install only JRE then you can’t develop any java application, but you can install any java-based application and run in your system. Internally contains JVM.

12. ***JVM:*** 🡪 Java virtual machine and this where the code gets processed

11. ***Final:*** 🡪 It is a keyword used for variables, methods, and class. When we put the word final before a

a. variable 🡪 final int a=10; then user can’t change the value for the integer

b. methods 🡪 final void m1(){} 🡪 then user can’t override the method in child class

c. Class 🡪 final class Test{} 🡪 then user can’t extend the class

12. ***Interface*** 🡪 Multiple inheritance can be achieved using interface using “Implements” word

a. variables --> static and final by default which means user is not needed to mention it explicitly.

b. Methods --> are abstract (has only definition and allowing others to use the functionality and not to see the internal logic) and public by default.

c. we can create object of an interface but can't instantiate the interface.

d. for instantiation, we use the object reference of the interface and the class name for eg.,

interface Test1{

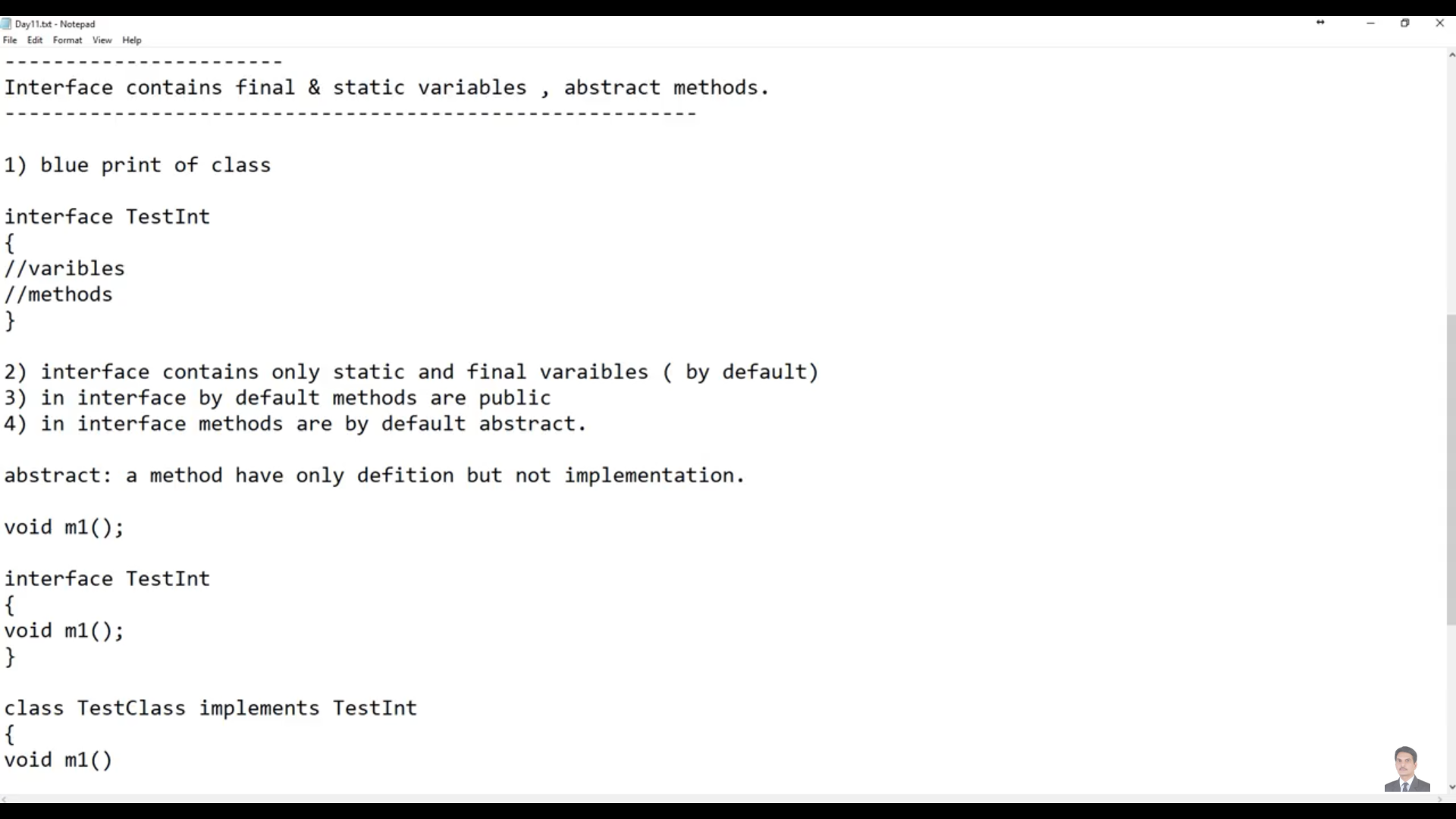
}

Class Test2 implements Test1{

Test1 test1obj = new Test2(); //Valid and **note** object name is not interface but instantiation is class name

Test1 test1obj = new Test1(); // invalid

}



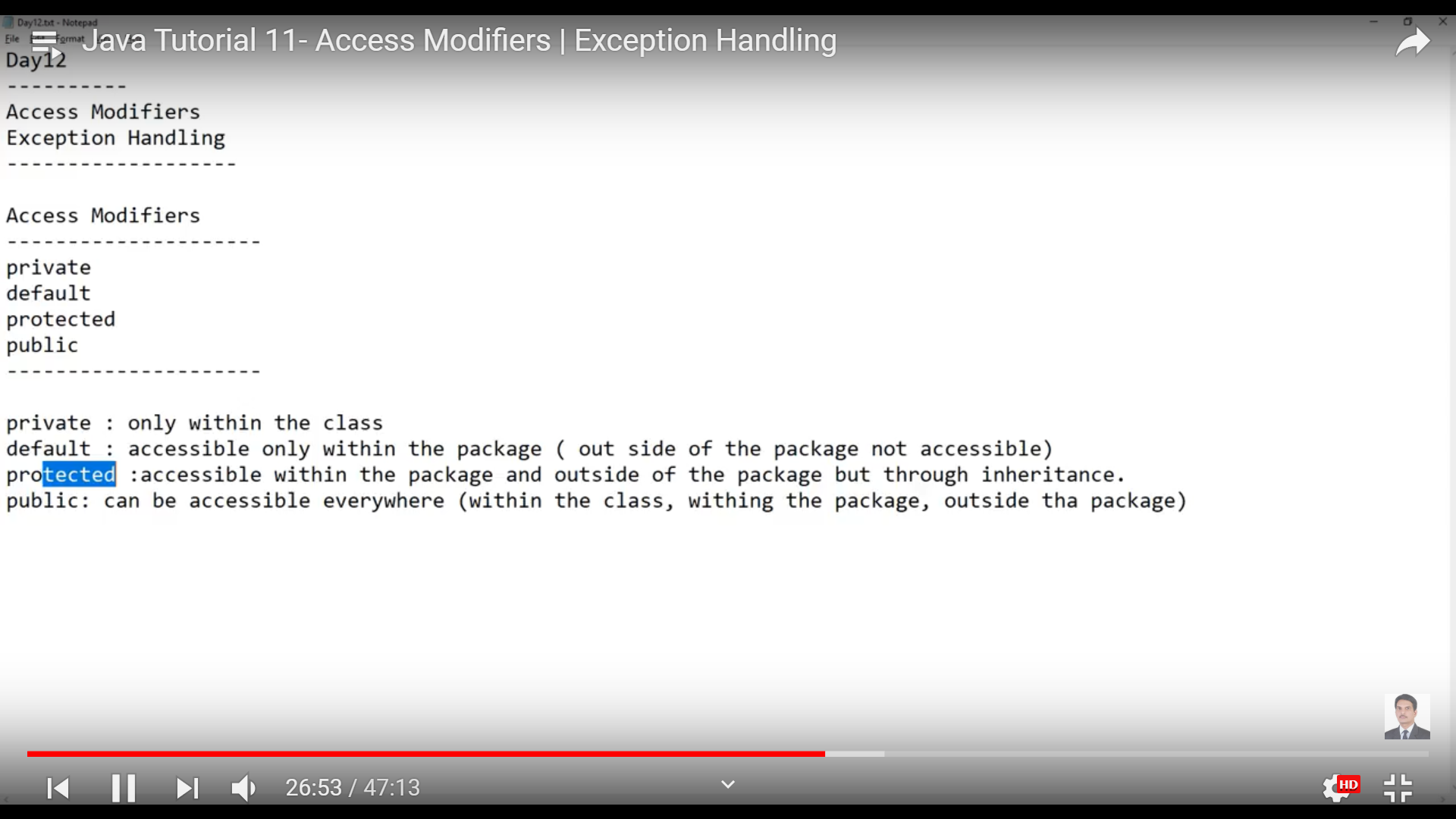
13. ***Access modifiers***

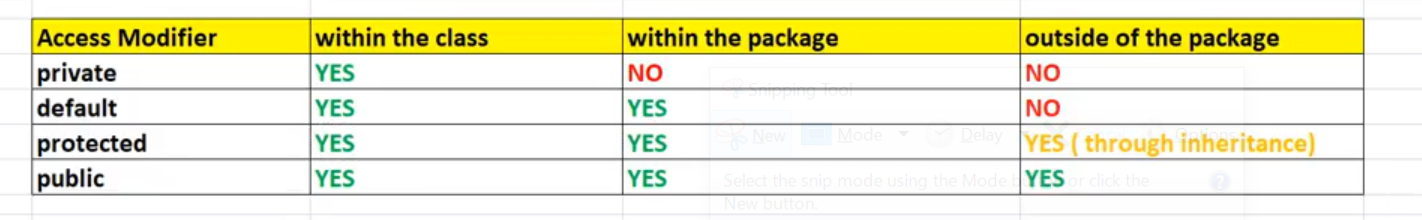
a. Public 🡪 The variables and the methods which are declared as public is accessible across the packages and the class

b. default 🡪 if access modifiers are not mentioned for a variable or to a method, then they are default variable or default method. These variables and methods are accessible inside the package only but across all the classes.

c. protected 🡪 The variable and the methods which are declared as protected then they are accessible inside the package. Also it is accessible outside the package but through inheritance

d. private 🡪 accessible only inside the class.





Super method 🡪 declared as super();

Generally used inside child constructor, to call the parent constructor. All constructor in classes by default have this super() inside constructor implicitly. We can use this explicitly when we want to call a parameterized parent constructor

Class Parent{

Public Parent(){

Sysout(“Parent”);

}

}

Class Child extends Parent{

Public Child(){

Sysout(“Child”);

}

}

When user creates an object for parent class like Parent variable = new Parent(); the output is just “Parent” whereas when user instantiates the child class then the output will be “parent” and “child” because super() will be implicitly mentioned in all constructors and we are not seeing any discrepancies in parent class because it is calling object class which is parent of all classes.

Super:

==========

It is a keyword and used to call a variable or method from parent class. Suppose if we have a variable int I and method public void abc() in parent class, then after inherited the parent class in child, then we can say

Super.i or super.abc() to refer the methods or variables in parent class

**Typecasting:**

Conversion of one primitive data type to another primitive data type is called type casting.

Types of casting:

1. Widening casting / implicit casting 🡪 this happens automatically
2. Narrow casting / explicit casting

**How to remember easily:**

Always check the right-hand side of the syntax

**Widening casting / implicit casting**

Conversion of smaller data type to bigger data type

Long I = short; parent = child

**Narrow casting / explicit**

Conversion of bigger data type to smaller

Short = (short)double;

Explicit casting has 2 types

**Upcasting** 🡪 Parent parentobj = new Child();

**Down casting** 🡪 Child childobj = new Parent(); 🡪 class cast exception and not allowed. Hence try below

Child childobj = parentobj;

Auto Boxing 🡪 <https://www.geeksforgeeks.org/wrapper-classes-java/>

conversion of primitive data type into object of a corresponding wrapper class

class Autoboxing {

public static void main(String[] args)

{

char ch = 'a';

// Autoboxing- primitive to Character object

// conversion

Character a = ch;

ArrayList<Integer> arrayList

= new ArrayList<Integer>();

// Autoboxing because ArrayList stores only objects

arrayList.add(25);

// printing the values from object

System.out.println(arrayList.get(0));

}

}

unboxing 🡪 conversion of wrapper class object into corresponding primitive data type.

class Unboxing {

public static void main(String[] args)

{

Character ch = 'a';

// unboxing - Character object to primitive

// conversion

char a = ch;

ArrayList<Integer> arrayList

= new ArrayList<Integer>();

arrayList.add(24);

// unboxing because get method returns an Integer

// object

int num = arrayList.get(0);

// printing the values from primitive data types

System.out.println(num);

}

}