[Techpils: Java Notes by Durga Sir, Natraj Sir and Naresh Technology. (java4732.blogspot.com)](https://java4732.blogspot.com/2015/08/java-pdf-notes-by-durga-sir-natraj-sir.html) – for all notes of Durga classes

Class Test

{

psvm (String [] args)

{

Int x;

Sop(x); - – CE: variable might not have been initialized

}

}

Tutorial 1 : Language Fundamentals

* Identifiers
* Reserve Words
* Data Types
* Literals
* Arrays
* Types of Variables
* Var-args Methods
* Main Methods
* Command Line Arguments
* Java Coding Standards

**IDENTIFIERS**

A name in java programming is called Identifier, which can be used for identification purposes. It can be method name or variable name, class name or label name

class Test

{

public static void main (String [] args)

{ int x;

// System.out.print( “ Hello ”);

}

}

Following are the identifiers in above program

* Test – class name
* Main - method name
* String - class name
* args – variable name (name of array)
* X - variable name

**\*IMP\***

**Rules for defining java identifiers:**

* **a to z**
* **A to Z**
* **0 to 9**
* **$**
* **\_**

Only allowed characters in java identifiers are above

**Rules**:- for identifiers

* If we are using any other character , we will get compile time error

Total\_number – allowed

Total# - not allowed

* Identifiers can’t starts with numerical values

Total123 – valid

123Total – invalid

* Java identifiers are **case sensitive** of course java languages itself is treated as case sensitive

Programming language

class Test

{

int number = 10;

int Number = 20;

int NUMBER = 30;

}

In this case we can differentiate with respect to case.

* In Java , there is no length limit for java identifiers, but it is not recommended to take too lengthy identifiers

Class Test

{

public static void main (String [] args)

{ int xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx = 10;

// System.out.print( “ Hello ”);

System.out.print(xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx);

}

}

This code is a valid code as java doesn’t have any limit on the length of identifiers,

But it is not a good practice to use a lengthy identifiers.

* We can’t use reserved words as identifiers

Int x = 10 - valid;

Int if = 10 – invalid; (because if is reserved words)

We can’t use reserve words as identifiers

* **We can use already defined java class name or interface name as identifiers.**

class Test{

public static void main (String [] args)

{

int String = 888;

**int Runnable = 999;**

System.out.println(String);

System.out.println(Runnable);

}

}

* **All predefined class name and interface name we can use as a identifiers, but it is not a good practice to use it.**
* Eventhough it is valid but it is not a good programming practice , because it reduces readability and creates confusion

Which of the following are valid java identifiers

* Total\_number - valid
* Total# - not valid
* 123total - not valid
* Total123 - valid
* ca$h - valid
* \_$\_$\_$\_$\_ - valid
* all@hands -not valid
* java2share -valid
* Integer - valid – class name
* Int - valid – not a reserve words
* Int - not valid - reserve words

In java some words are reserved words to represent some meaning or functionality such types of words are called reserved words (53) keywords are in java

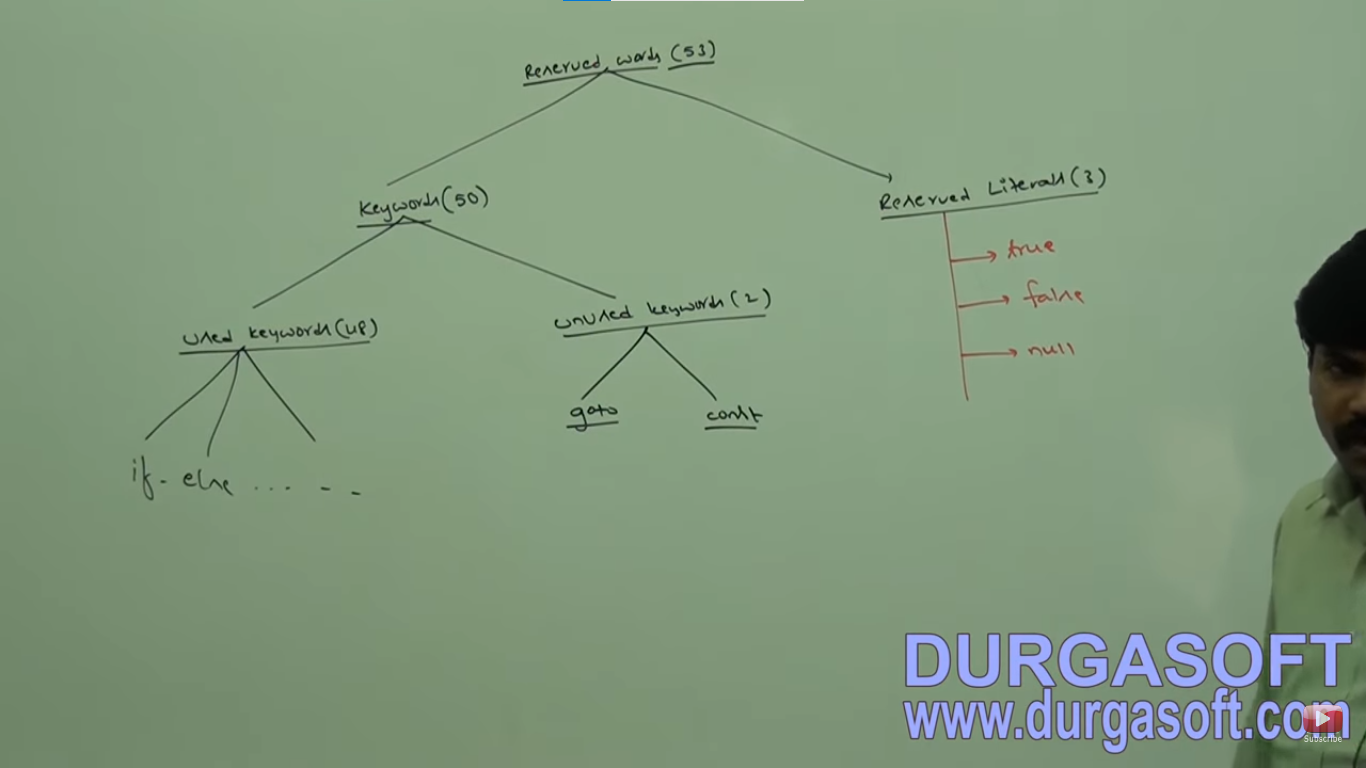
<Goto - 35.12 >

Reserved Words (53) = Keywords (50) + Reserved Literals (3)

Keywords (50) – if the reserved word is assoicated with functionality

Reserved Literals (3) - [true, false, null]

Keyword(50) = used keywords(48) + nounsed keywords(2)



Keywords for dataTypes (8) - bit, short, int, long, float, double, boolean, char

Keywords for flow control (11)- if-else. switch (case, default ), while , do-while, for, break, continue, return

Keywords for modifiers (11) - public, private, protected, static, final, abstract, synchronized, native, strictfp, transient, volatile

keywords for exception handling (6)- try-catch, finally, throw, throws, assert

Keyword related to class (6)- class, interface, extends, implements, package, imports

Object related keywords - new , instanceof, super, this,

in Java return type is mandatory, if method don't written anything we have to return this method with void return type

**Unused keywords:** Goto and const - usage of these reserved words give you compile time error

**Reserved Literals:**  true, false, null

true, false - value for default boolean data types

\*\* null - default value for object reference.

Enum keyword : we can use it to defined a group of named constant

**Common conclusion for Keywords:**

1. all 53 keywords contain lower case alphabet and no numerical

2. in java we have only "new" keyword and there is no delete keyword because destruction of useless object is the responsbility of Garbage Collector

3.The following are new keyword in java

strictfp, enum, assert

4. strictfp is not strictFp ,

instanceof but not instanceOf,

synchronized but not synchronize

extends but not extend,

implements but not implements,

import but not imports,

const but not constant

1. Which of the following list contains only java reserved words.

1. new , delete

2. goto, constant

3. break, continue, return, exit

4. final, finally, finalize

5. throw, thorws, thrown

6. notify, notifyAll

7. implements, extends, imports

8. sizeof, instanceof

9. instanceof, strictFp

10. byte, short, Int

11. None of the above - Ans

2. Which of the following is the reserved words

public

static

void

main - name of method

String - name of class

argse - name of variable

**Tutorial - 2**

In java every variable and every expression has some type

Each and every datatype is clearly defined

Every assignement should be checked by compiler for type compatability

Because of above reasons we can conclude , java language strongly typed programming language

Java is pure object oriented language or not ?

1. No, becuase several OOPs feautre is not suported by Java (Operator Overloading , multiple inheritance)

2. Moreover, we are depending on primitive data types, which are non objects

it support primitive data types.

Primitive DataTypes in Java.

Numberical Data Types: - byte, int, short, long, float, double

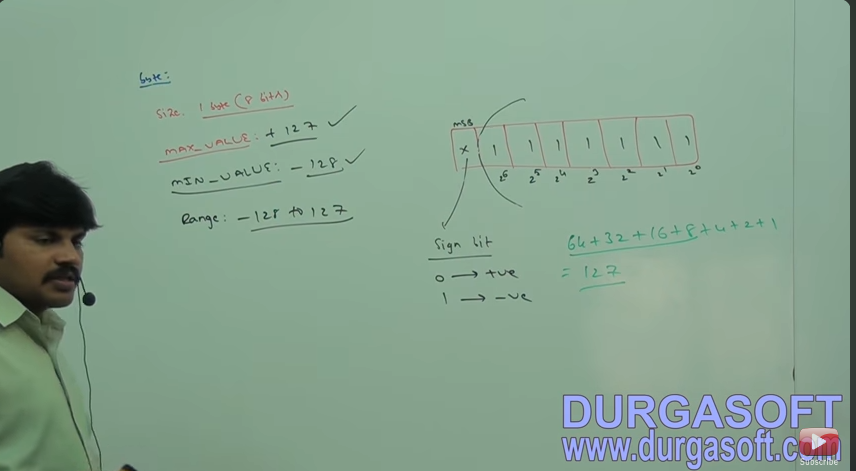
Non Numberical Data Types: boolean



\*\*

Except **boolean and char** remaining data types are considered as - Signed Data Types

int x = -10 or int x = 10; float b = -10.23 or float b = 10.23



the most significate bit is Sign bit .

0 = positive number

1= Negative Number

positive number will be represented directly in memory, negative will be represented in 2' complemented form.

**Tutorial 3:**

[(185) Core Java with OCJP/SCJP: Language Fundamentals Part- 3 || Data Types part-2 - YouTube](https://www.youtube.com/watch?v=HYAosZptldA)

|  |  |
| --- | --- |
| **Float** | **double** |
| Want 4 to 5 decimal of accuracy | Want 14 to 15 decimal of accuracy |
| Single precision | Double precision |
| Size 4 bytes | Size 8 bytes |
|  |  |

Asfj

**\*IMP\***

Boolean: Default values are true and false both in **lower case**

**boolean b = true – correct**

**boolean b = 0 – error: incompatible type required Boolean found int**

**boolean b = “True” – error: incompatible type required Boolean found String**

**boolean b = True – cannot find symbol , Symbol: Variable True, location: class Test**

**Ex: int x = 0;**

**If(x)**

**{**

**System.out.println(“Hello”); Compile Error: incompatible type**

**} found int**

**Required boolean**

**Else**

**{**

**System.out.println(“Hello”);**

**}**

Same with While (1)

{

System.out.println(“Hello”);

}

**Char DataTypes:**

Old languages C or C++ ,

The number of allowed ASCII code characters are <=256

To represent this 256 characters 8 bit are enough hence, the size of char in old languages is 1 byte

But java is Unicode base under the number of different Unicode characters are >256 under <65536

To represent this many characters 8 bit may not enough compulsory we should go for 16 bits

Hence the size of char is 2 bytes, better to take 2 bytes a range 0 - 65535

**\*IMP\***

**Summary: Java Primitive Data Types:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Type | Size | Range | Wrapper | Default Value |
| Byte | 1 Byte |  | Byte | 0 |
| Short | 2 Bytes |  | Short | 0 |
| Int | 4 Bytes |  | Integer | 0 |
| Long | 8 Bytes |  | Long | 0 |
| Float | 4 Bytes |  | Float | 0.00 |
| Double | 8 Bytes |  | Double | 0.00 |
| Boolean | NA |  | Boolean | False |
| Char | 2 Byte |  | Character | 0 (represent space character) |
| String or Object |  |  |  | Null |

\*\* - Null is not applicable for primitive

Tutorial 4:

[(186) Core Java with OCJP/SCJP: Language Fundamentals Part-4 || Literals Part-1 - YouTube](https://www.youtube.com/watch?v=Eng8Oi-p2r4)

**Literals:**

Asdf

Int x = 10

Int - datatype/ keyword

X – name of variable/ identifier

10 – constant value / literal

Integral literals

For integral data types (byte, short, int, long) we can specify literal value in the following base,

decimal literals (base 10 allowed digits are 0-9) ex. Int x = 10

Octal form (base 8 allowed digits are 0-7, literal value should be prefixed with “0”) ex. Int x = 010

HexaDecimal form (base 16 ) allowed digits are 0-9 , a to f ) for extra digits a to f , java is case sensitive language , for extra digit we can use any case for writing

this is one of few areas where java is not case sensitive.

The literal value should be prefixed with 0x or 0X ex. Int x = 0X10;

These are only possible way to specify literal value for integral data types.

Possible Question from Literals

**\*IMP\***

**Which of the following declaration are valid.**

**Int x = 10; - correct**

**Int x = 0786; - wrong one CE: integer number too large**

**Int x = 0777; perfectly valid – correct one**

**Int x = 0XFace; - valid**

**Int x = 0xBeef – valid**

**Int x = 0xBeer – not valid – r is out of range of hexadecimal**

**Programmer has choice to specify the value in any format , but the compiler will provide the output only in decimal format**

By default every integral literal is of int type. But we can specify explicitly as long type by suffixed with small l or L.

Int x = 10; - correct

Long x = 10 L

Int x = 10 L; - incorrect

There is no direct way to specify byte and short literals explicitly

But indirectly we can specify whenever we are assigning integral literal to the byte variable , and the value within the range , then compiler treat its automatically as byte literal and similar for short literals

Byte b = 10;

Byte b = 127 – allowed

Byte b = 128 – possible loss of precision , found int required byte

Short x = 32767 – allowed

Short x = 32768 - possible loss of precision , found int required short

Floating point literals:

Float f = 123.456 f;

By default every floating point literal is of double type and hence, we can’t assign directly to the float variable

But, we can specify floating point literal as float type, by suffixed with small f or F.

double d = 123.456 D; - correct

we can specify explicitly floating by literal

double d = 123.456d; possible loss of precision found double required float

double d = 123.456 – decimal

double d = 0123.456 – octal – allowed but representation is in decimal form

double d = 0x123.456 - hexadecimal – not allowed

CE: malformed floating point literals

\*\*we can specify floating point literals only in decimal form and we can’t specify in ocatal or hexadecimal form

Double d = 0786;

WE can assign integral literal directly to floating point variables, and that integral literal can be specified either in decimal or octal or hexadecimal forms

Double d = 0786 – not allowed

Double d = 0xface - allowed

Double d = 0786.0 –allowed

Double d = 0xface.0 - not allowed

**Tutorial 5:**

As

We can specify char literal as single character within single quotes

Char ch= ‘a’;

Char ch = a; here compiler will treat ‘a’ as a variable

Char ch = “a” – CE: incompatible type found String required char.

Char ch = ‘ab’ - CE: unclosed character literal

Char ch = 97;

Sop(ch); //a

The possible range allowed is the 0 to 65536

We can specify char literal as integral literal which represents Unicode value of the character and that integral can be specified either in decimal or octal or hexadecimal forms but allowed range is 0 to 65535

Char ch = 0xface - allowed

Char ch = 0777; - allowed

Char ch = 655535 – allowed

Char ch = 655536; CE:// possible loss of precision found int required character

We can represent char literal in Unicode representation which is nothing but ‘\uxxx’

Char ch = ‘\u0061’;

Sop(ch);

Ch = a;

Escape character

‘/n’ = new line

‘/t’ = tab

‘/m’ = CE: illegal escape character

Every escape character is a valid char literal

Escape character | description

\n | new line

\t | new tab

` \r carriage return

\b Back space

\f from feed

\’ single quote

\” double quote

\\ Back slash

Which of the following are valid

Char ch = 65536; - invalid – max range is 65535

Char ch = 0xBeer; - invalid - r is out of range

Char ch = \uFace; - invalid single quote missing

Char ch = ‘\ubeef’; - valid

Char ch = ‘\m’; - not valid

Char ch = ‘\iface’; - not valid

**String literal:**

Char ch = ‘\iface’; - not valid

Any sequence of character within double quotes is treated as String literal

String s =

1.7 version enhancement with respect to literals

1. Binary literals – int x = 0B 1111,

2.

For integral datatypes unit 1.6 version we can specify literal value in the following ways

Decimal form , octal form, hexadecimal form ,

From the 1.7 version onwards we can specify literal value even in binary form also.

Allowed digits are 0 and 1;

Literal value should be prefixed with 0b or 0B

Double d = 123456.789

From 1.7 version onward we can use \_ symbols between digits of numeric literal

Double d = 1\_23\_456.7\_8\_9;

as well we can write like

Doubled d = 123\_456.7\_8\_9

The main advantage of this approach is readability of the code will be improved

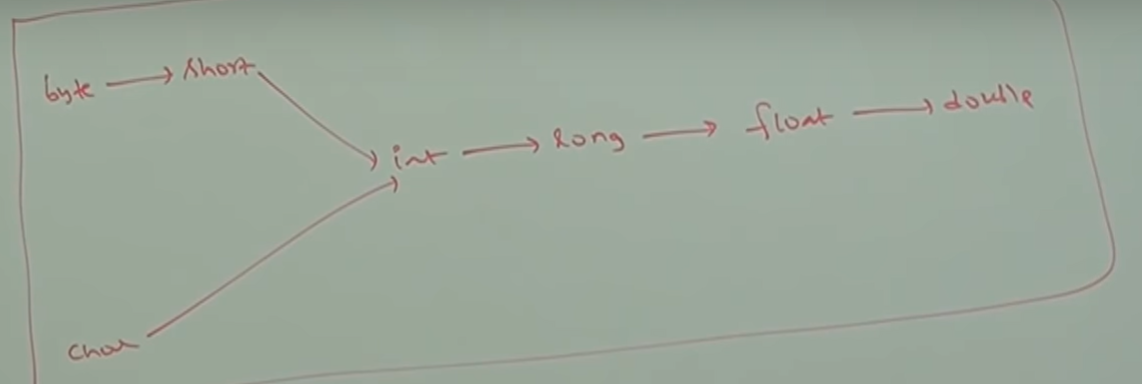
We can use more than one “ \_” symbol also between the digits

Double d = \_1\_23\_456.789; - invalid

Double d = \_1\_ \_23\_ \_ \_456.789\_; - invalid

We can use \_ only between the digits if we are using anywhere else we wil get compile time error

Summary Point:



This type of assignment of variable is always acceptable

8 byte long value we can assign to 4 byte float variables because both are following different memory representation internally

Float f = 10l;

Sop(f); // 10.0

**Tutorial 6:**

Dsf

**Arrays:**

1. Introduction
2. Array Declaration
3. Array Creation
4. Array Initialization
5. Array Declaration, creation and Initialization in a single line
6. Length vs length()
7. Anonymous Array
8. Array element assignment
9. Array Variable Assignment

An array is an index collection of fixed number of homogeneous data elements.

The main advantage of arrays is we can represent huge number of values by using single variable so that, readability of the code will be improved but the main disadvantage of arrays is fixed in size. i.e once we creates an array there is no chance of increasing or decreasing the size of the array as per our requirements .

Hence, to use arrays concept compulsory we should the size in advance, which may not possible always

Array Deceleration

One Dimensional Array Decleration:

Int [] a; - allowed

Int []a; - allowed

Int a []; - allowed

At the time of declaration we can’t specify the size otherwise we will get compile time error

Two Dimensional Array Decleration:

Int [][] x; - valid

Int x[][]; - valid

Int []x[];- valid

In two-dimensional array x

There is declaration

Which of the following are valid

Int [] a,b; a->1, b->1

Int [] a[],b; a->2, b->1

Int [] a[],b[]; a->2, b->2

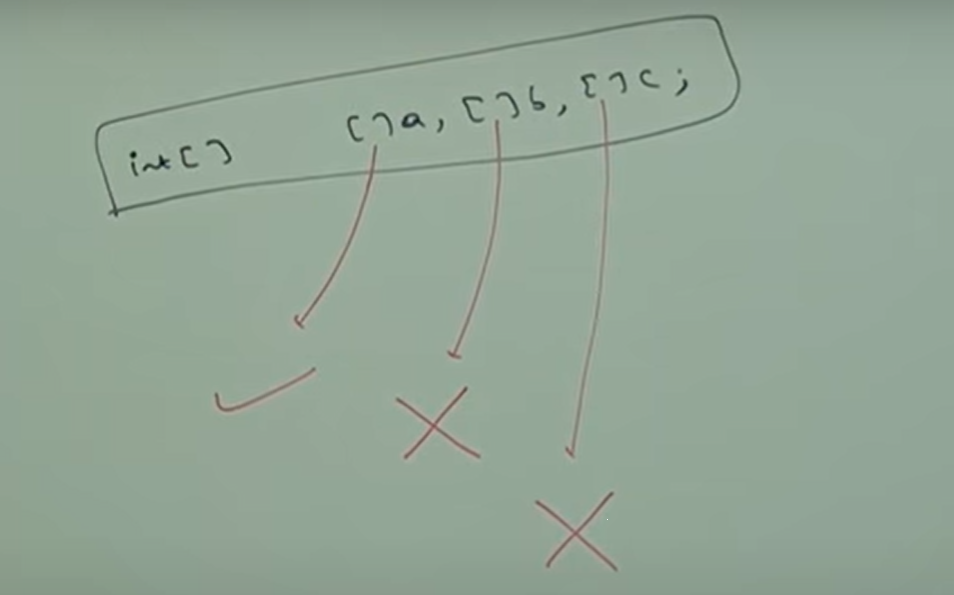
Int [] [] a,b; a->2, b->2 // here Int [] [] is considered as a part of dimension not part of a.

Int [] []a, b[]; a->2, b->3

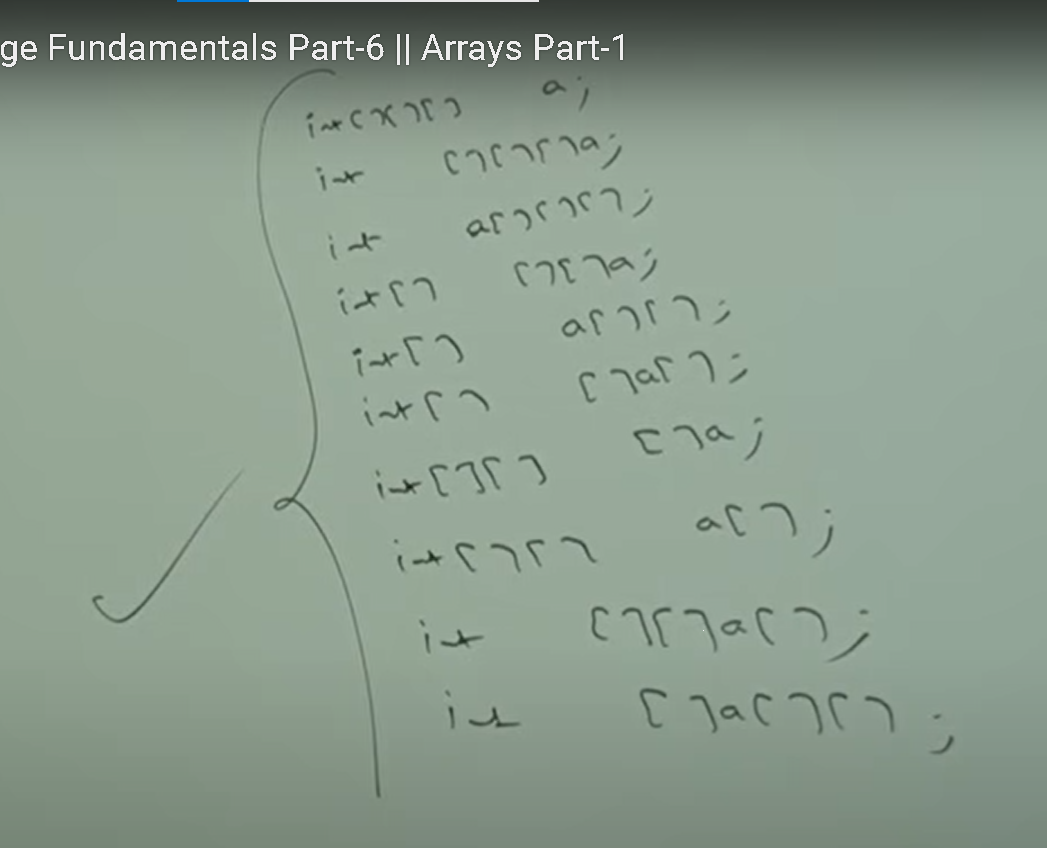
Int [] []a,[]b -->CE:

If want to specify before the variable the facility is applicable only for first variable in our declaration

If we are trying to apply for remaining variable we will get compile time error



Three-Dimensional Array Declaration:



Array Creation:

Int [] a = new int [1];

Every Array in java is an object only hence, we can create arrays by using new operator

Int [] a = new int [3];

For every array type corresponding classes are available and these classes are part of java language and not available to the programmer level.

Int [] a = new int [3];

Sopen(a.getclass().getname());

Array type Corresponding Class Name

Int [] [I

Int [][] [[I

Double[] [D

Short [] [s

Byte[] [B

Boolean [Z

Conclusion:

1. At the time of creation, we must specify the size of array otherwise we will get compile time error.
2. Int [] x = new int[0];
3. It is legal to have an array with size zero in java.
4. Int [] x = new int [-3]; - invalid but no CE: error RE: Negative Array Size Exception
5. If we are trying to specify array size with some negative int value then we will get RE saying Negative Error Size Exception.

Int [] x = new int [10]; - valid

Int [] x = new int [‘a’]; - valid

Byte b = 20;

Int [] x = new int [b]; - valid

Short s = 30;

Int [] x = new int [s]; - valid

Int [] x = new int [10 l]; CE : possible loss of precisions required int provided long.

Byte short char int if we are trying to specify any other type then we will get compile time error.

**Tutorial 7:**

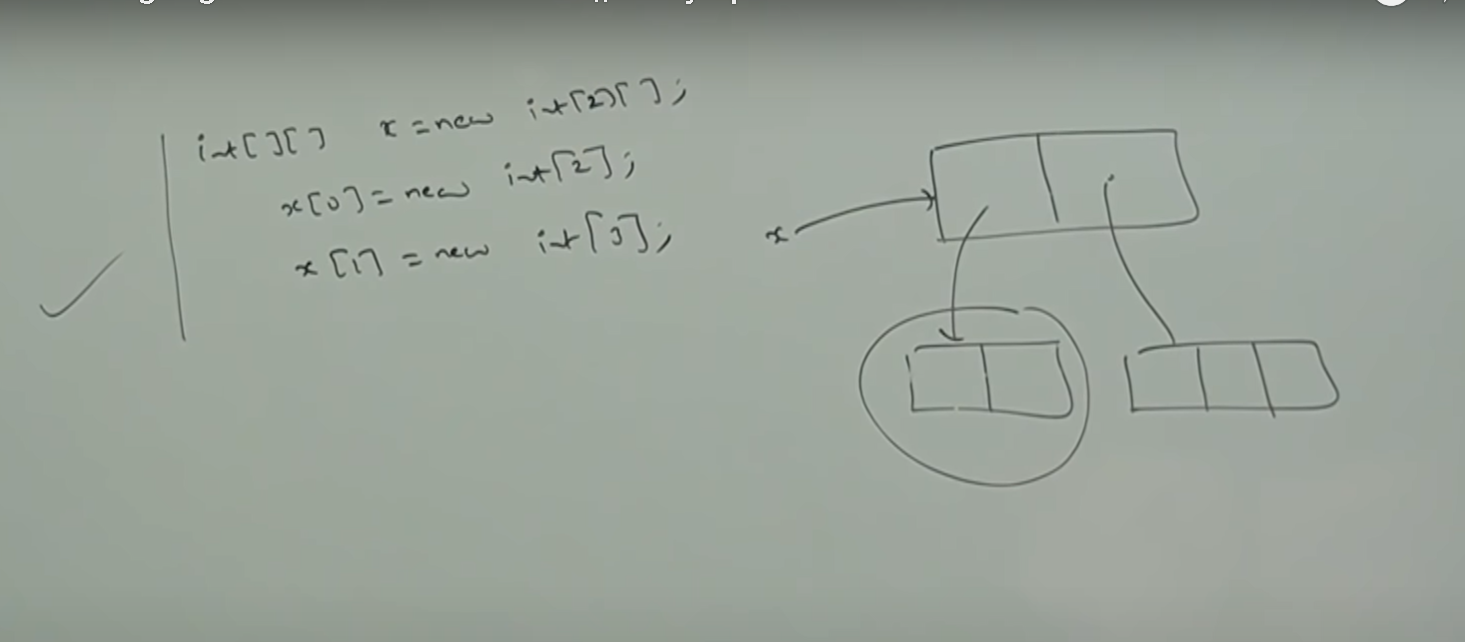
Byte

Two Dimensional Array Creation:

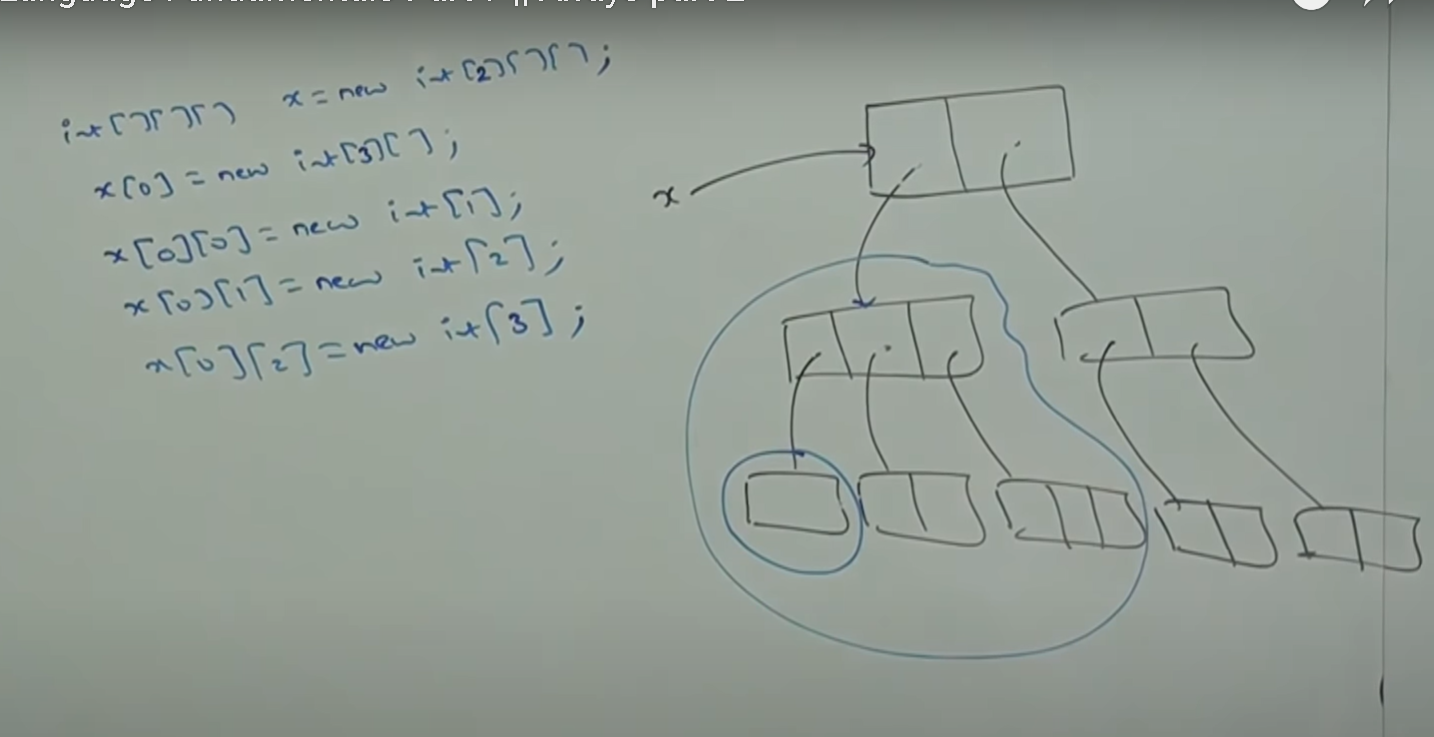
In java two dimensional not implemented by using Matrix style, some people followed Array of Array’s approach for multi dimension array creation.

The main advantage of this approach is Memory Utilization will be improved .

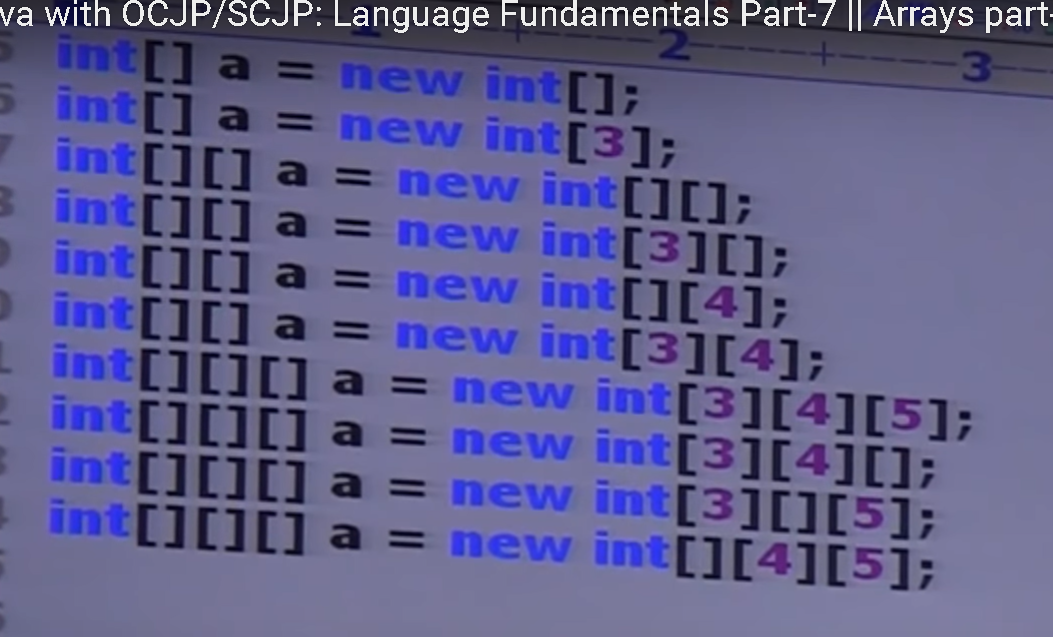
Memory structure and corresponding



Example 2 :



Which of the following Array declaration are valid ?



Red ones are invalid

Once we creates an array every element be default initialize with default values

Int [] x = new int [];

Sop (x); // [I @e3p..

Sop (x[0]); // 0

A whiteboard with writing on it

Description automatically generated

Whenever we are trying to bring any reference variable internally toString method will be called which is implemented by default to written the String in the following form.

**classname@hascode\_in\_hexadecimalform**

**Example 1:**

int [] [] x = new int [2] [3];

sop(x ); // [I @

sop(x [0] ); //3

sop(x [0] [0] ); //0

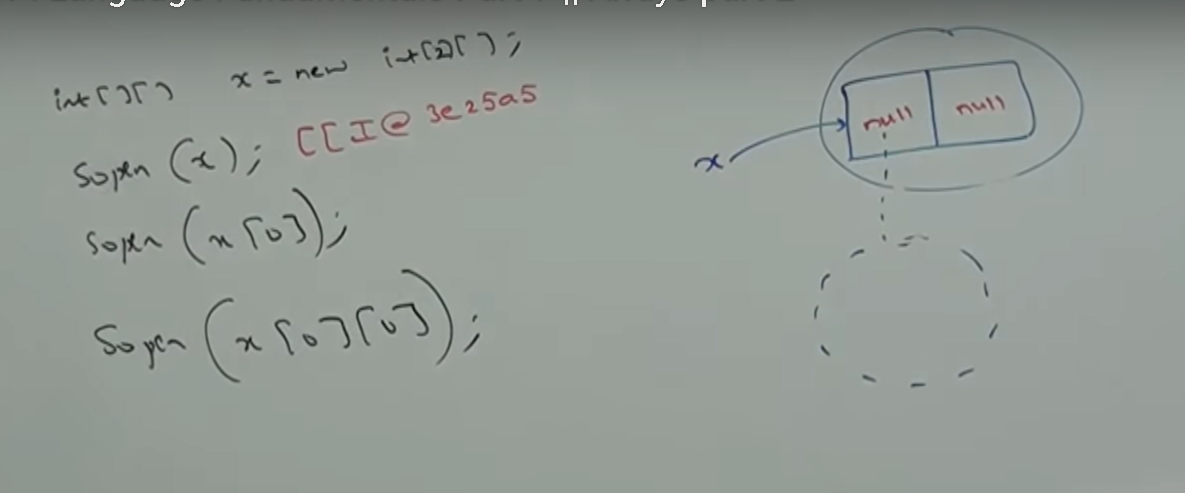
Example 2:

int [] [] x = new int [2] [ ];

sop(x ); // [I @

sop(x [0] ); //null

sop(x [0] [0] ); // null pointer exception – as we don’t have specify the value to initialize it point to object which has default value null



Note:

If we are trying to perform any operation on null then we will get runtime exception saying null pointer exception

Example 3 : int [] x = new int [6];

Once we create an array every array element by default initialize with default values , if we are not satisfied with default values then we can override these values with our customized values.

int [] x = new int [6];

x[0] = 10;

x[1] = 20;

x[2] = 30;

x[3] = 40;

x[4] = 50;

x[5] = 60;

x[6] = 70; RE: Array Out of bound exception

x[-6] = 70; RE: Array Out of bound exception

x[2.5] = 70; CE: Possible loss of exception found double required integer

if we are trying to access array value with out of range index either positive value or negative integer value then we will get run time exception saying array index out of bound exception.

**Tutorial 8:**

if we

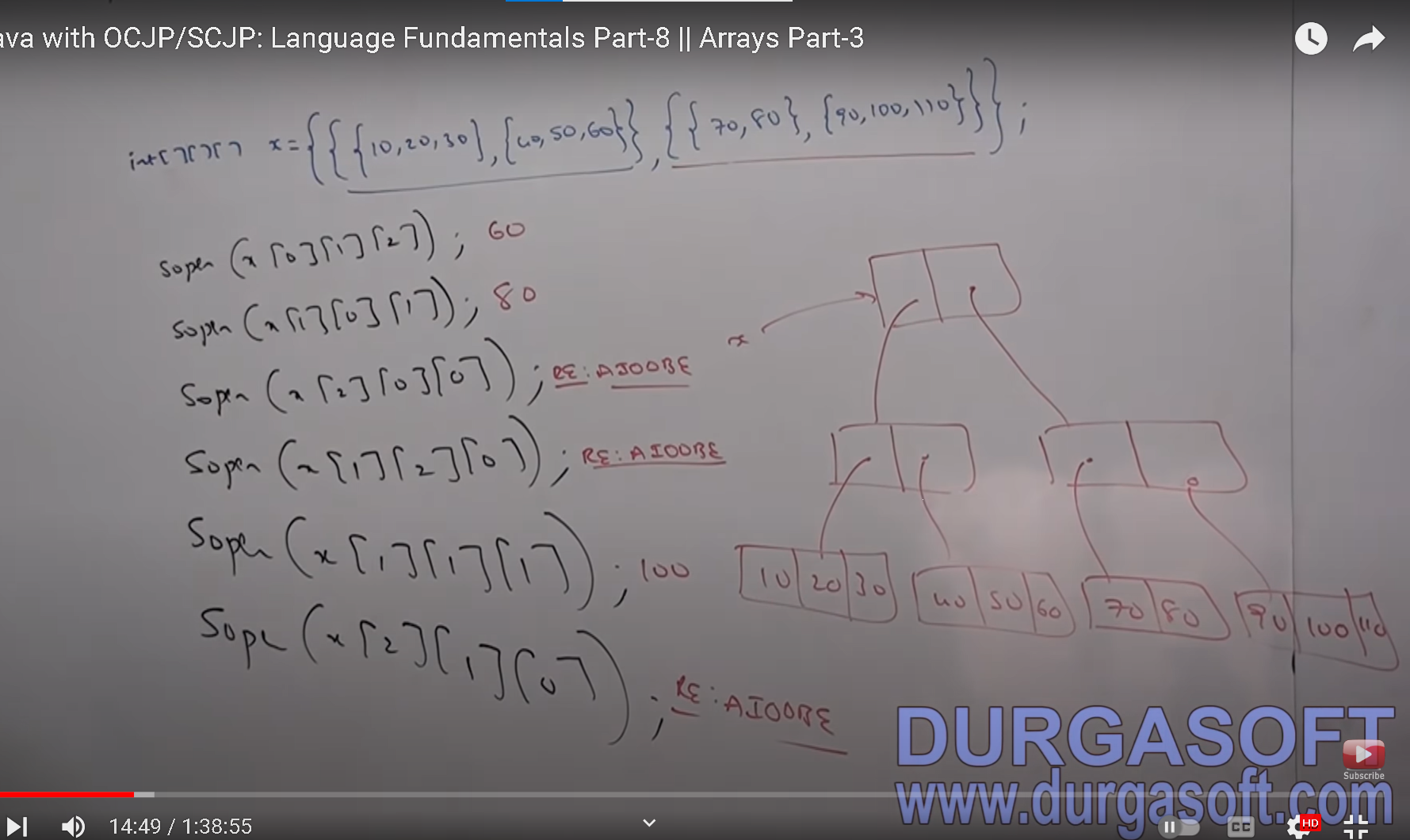
Array Decelaration, Creation and Initialization in a single line

Int [] a = {10,20,30} ;

Char[] a = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’};

String [] A = {‘A’, “AA”, “AAA”};

Int [][] x = {{10,20}, {30,40,50}}



If we went to use this shortcut compulsory we should perform all activities in a single line.

If we are trying to divide into multiple lines then we will get compile error

Int []x;

X = {10,20,30} – CE: illegal start of expression

Length() vs length;

Length:

Int [] x = new int [6];

Sop(x.length()); CE: cannot find symbol

Symbol: method length()

Location: class int[];

Sop(x.length);

Length is a final variable applicable for arrays

Length variable represent the size of the array

Int [] x = new int [6];

Sop (x.length) //6

Length() – is the final method of applicable for String object

Length method returns number of characters present in the string

String s = “durga”

Sop(s.length()); // 5

sop(s.length) // CE:// cannot find symbol

symbol : variable length

location : class is a string

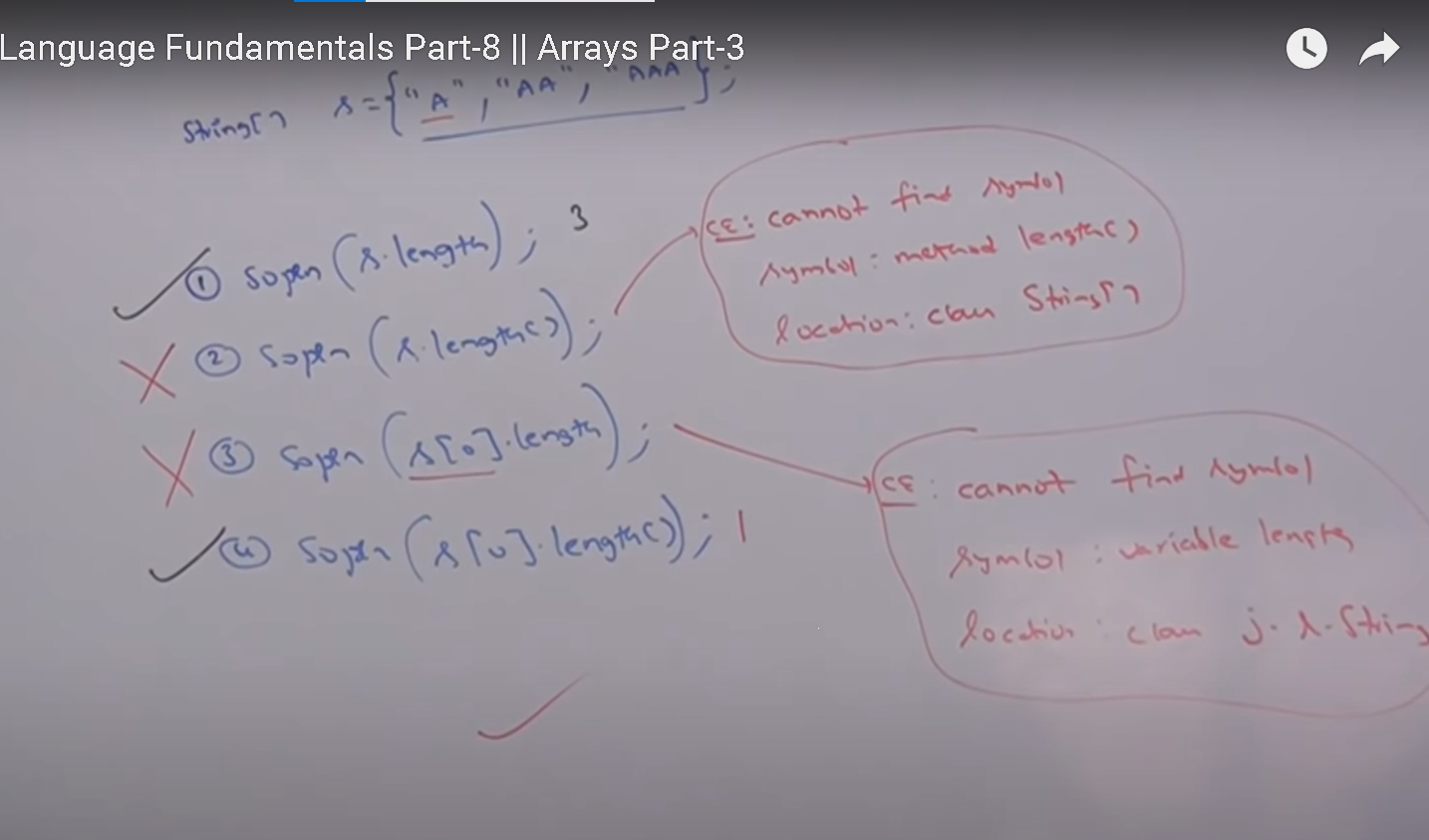
length variable applicable for arrays but not for string objects

whereas length method applicable for string objects but not for arrays

Question:

String [] s = {“A”, “AA”, “AAA”};

1. Sop(s.legnth) //3
2. Sop(s.legnth()) // CE:
3. Sop(s[0].legnth)//CE
4. Sop(s[0].legnth()) //1



Find the length of array

Int [][] x = new int [6][3]

Sop(x.length) //6

In multidimensional arrays length of variable represents only base size but not the total size

There is no direct way to specify length of multidimensional array but indirectly we can find as follows

x[0].length + x[1].length +x[2].length+ …………

**Anonymous Arrays:**

Array

Int [] x = {10, 20, 30};

Sop(x.length);

Sometimes we can declare an array without name, such type of name less arrays are called anonymous arrays

The main purpose of anonymous arrays is just for instance use (one type usage)

Class Test

{

Psvm(String[] args)

{

Sum (new int [] {10,20,30};

}

Psv sum(int[] args)

{

Int total = 0;

For(int x : x

{

Total = total + x;

}

Sopen ( “The sum” + Total)

}

}

We can create anonymous array as follows:

New int[] {}

While creating anonymous array we can’t specify the size , otherwise we will get compile time error

New int[ 3] {10,20,30} – not valid

New int [] {10,20,30}; - valid

We can create two dimensional array as well

New int [][] { {10,20,30}, {40,50,60}, {70,80,90} }

Based on our requirements we can give the name for anonymous array then it is no longer anonymous

Int [] x= new int [10,20,30] ;

In the above example to call some method we required an array but after completing some method call we are not using that array any more. Hence, for this one time requirement anonymous array is best choice.

Array element assignment

**Case 1: in a case of primitive type array as array elements we can provide any type**

Which can be implicitly promoted to declared type

Int [] x =

int [] x = new int [5];

X[0] = 10;

X[1] = ‘a’;

In the case of float type arrays , the allowed datatypes are byte, short char, int , long, float

**Case 2 : in case of object array**

in the case of object type array as array elements we can provide either declared type objects or its child class objects

Object[] a = new object [2];\

a[0] = new object();

a[1] = new String (“durga”);

a[2] = new Integer(10);

Example 2:

Number [] n = new Number [10];

n[0] = new Integer (10);

n[1] = new Double (10.5)

n[2] = new String (“durga”); - not valid CE: incompatible types

found: java.lang.string

required: java.lang.Number;

**Tutorial 9:**

**Case 1**: element level promotion is not applicable at array level

Int [] x = {10, 20, 30}; Case 1:

Char [] ch = {‘a’, ‘b’, ‘c’, ‘d’};

Char element can be promoted to int type where as char array can not be promoted to the int [ ]

Int [] b = x;

int [] c = ch; - char element can be promoted to int but char array can not be promoted to int

C E: incomptable type found char[] required int []

Which of the following promotions will be performed automatically ?

Char -> int - allowed

Char [] -> int[] – not allowed

Int -> double - allowed

Int[] -> double[] – not allowed

Float -> int – allowed

Float[] -> int[] – not allowed

String -> Object – allowed child to parent transition is allowed

But in the case of object type arrays - child class type array can be promoted to parent class type array

String[] -> Object[] – allowed

But in the case of object type arrays - child class type array can be promoted to parent class type array

String [] A = {“A”, “B”, “C”}

Object[] a = s;

**Case 2:**

Int [] a = {10,20,30,40,50,60};

Int [] b = {70,80};

1. a = b;
2. b = a;



Whenever we are assigning one array to another array internal elements won’t be copied just reference variable will be reassigned.

**Case 3:**

Int [][] a = new int [3][];

a[0] = new int [4][3]; CE: incompatabile type

found int [][]

required int []

a[0] = 10; CE: incompatible type found int required int[]

a[0] = new int [2]; - allowed

whenever we are assigning one array to another array the dimension must be matched

for example in the place of one dimensional int array we should provide one dimensional array only if we are trying to provide any other dimension then we will get compile type error.

a[0] = new int [4][3]; CE: incompatabile type

found int [][]

required int []

a[0] = 10; CE: incompatible type found int required int[]

whenever we are assigning one array to another array both dimensiona and types must be matched but sizes are not required to match.

a[0] = new int [2]

Questions:

Class Test

{

Public static void main (String [] args)

{

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

{

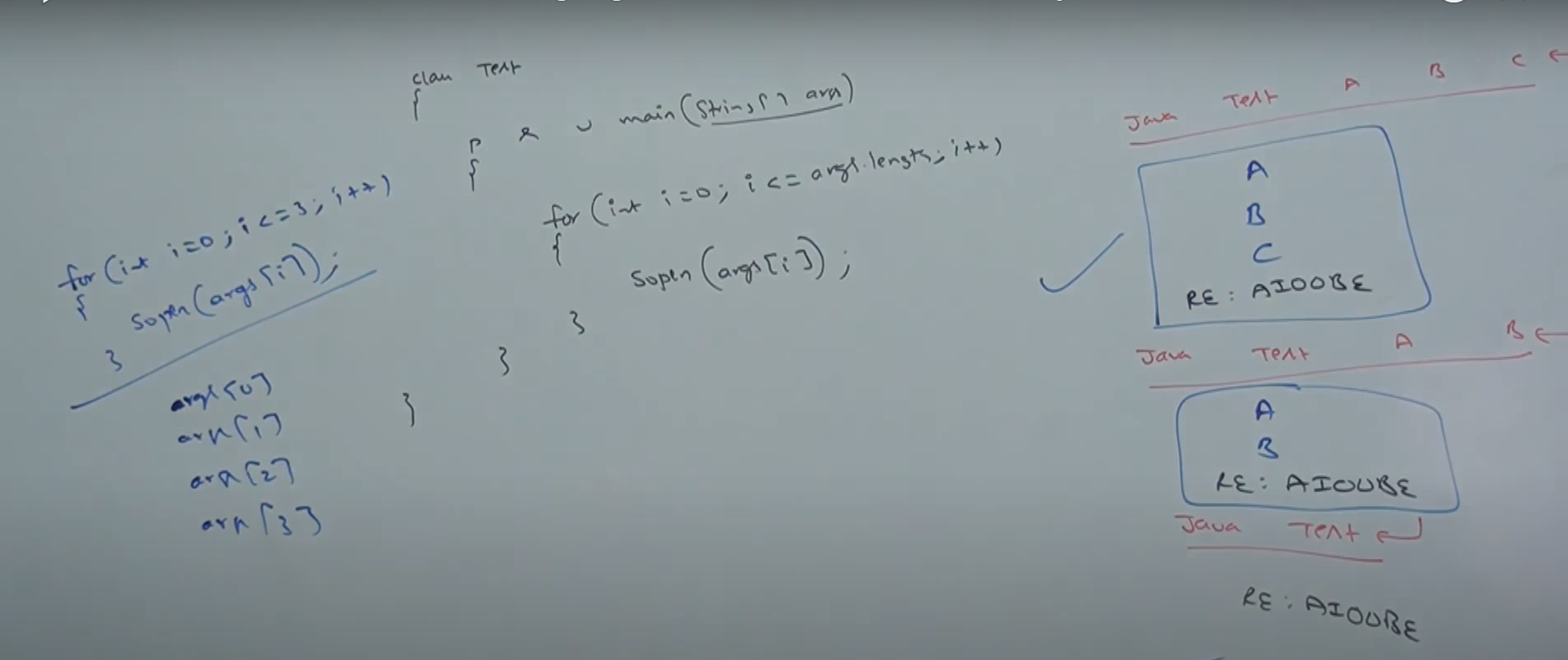
Sop (args[i]);

}

}

}

1. Java Test
2. Java Test A B
3. Java Test A B C



Example 2

A white board with writing on it

Description automatically generated

Example 3

A whiteboard with writing on it

Description automatically generated

**Tutorial 10:**

Types of Variable:

Based on type of value represented by variable all variables are divided into two types

1. Primitive variable : The variable which is used to represent primitive value is called primitive variable.

It can be used to represent primitive values

Int x = 10;

1. Reference variable: The variable which is used to represent refer object is called reference variable

reference variable can be used to refer objects

Student s = new Student();

Based on position of declaration and the behavior all variables are divided into

1. Instance variables
2. Static variables
3. Local variables

Class Student

{

String name;

Int roll no;

}

**Instance variables**

1. Instance variables

If the value of the variable is varied from object to object such type of variable are called instance variables for every object a separate copy of instance variables will be created.

1. instance variables should be decelerated within the class directly but outside of any method, block or constructor.
2. instance variables will be created at the time of object creation and destroyed at the time of object destruction hence, the scope of instance variable is exactly same as scope of object
3. instance variable will be stored in the heap memory as a part of the object.
4. We can’t access instance variable directly from static area but we can access by using object reference.
5. But we can access instance variable directly from instance area.

Class Test

{

Int x = 10;

Psvm (String [] args)

{

Sop (x); - CE: non-static variable x cannot be referenced form static context

Test t = new Test();

Sop(t.x);

}

Public void m1()

{

Sop (x); here instance variable x can be access directly .

}

}

What is the value of x , d, b, s

Class Test

{

Int x;

double d;

boolean b;

String s;

Psvm (String [] args)

{

Test t = new Test();

Sop(t.x); // 0

Sop(t.d); // 0.0

Sop(t.b); // false

Sop(t.s); // null

}

}

**For instance variable JVM will always provide default values and we are not required to perform initialization explicitly.**

Instance variable is also know as **object level variables** or sometimes we can use **attributes**

**Static Variables:**

If the value of a variable is not varied from object to object then it is not recommended to declare variable as instance variable we have to declare such type of variable at class level by using static modifier

In the case of instance variable for every object a separate copy will be created but in the case of static variables a single copy will be created at a class level and shared by every object of the class.

Static variables should be declared within the class directly but outside of any method or block or constructor

Static variables will be created at the time of class loading and destroyed at the time of class unloading hence scope of static variable is exactly same as scope of .class file.

What happens if we run java Test command

1. Start JVM
2. Create and start main thread
3. Locate Test.class file
4. Load Test.class file – load static variable
5. Execute main method
6. Unload Test.class file - unload static variable
7. Terminate main method
8. Shutdown JVM

Static variable will be stored in method area

Ways to access Static Variable

Class Test

{

Int x;

psvm (String [] args)

{

Test t = new Test();

Sop(t.x);

Sop(Test.x);

Sop(x);

}

}

We can access static variables either by object reference or by class name but recommended to use class name.

Within the same class it is not required to use class name and we can access directly.

We can access static variables directly from both instance and static areas.

For static variable

**For static variables JVM will provide default values and we are not required to perform initialization explicitly**

Class Test

{

Static Int x;

Static double y;

Static String z;

psvm (String [] args)

{

Sop(Test.x); //0

Sop(Test.y); //0.0

Sop(Test.z); // null

}

}

For static variables JVM will provide default values and we are not required to perform initialization explicitly

Static variables are also known as class level variables or fields.

Class Test

{

Static Int x=10;

int y = 20;

psvm (String [] args)

{

Test t = new Test();

t.x = 888;

t.y = 999;

Test a= new Test();

Sop(a.x + “ - ” + a.y); // 888 - 20

}

}

**Tutorial 11:**

Local Variables or Temporary Variables or Stack Variables:

Sometimes to meet the temporary requirements of the programmer we can declare variables inside a method or block or constructor .

Such types of variables are called local variables or temporary variables or stack variables or automatic variables.

Local variables will be stored inside stack memory.

Local variables will be created while executing the block in which we declare it.

Once the block execution completes automatically local variable will be destroyed. Hence, the scope of local variable is the block in which we declared it is exactly same as.

Class Test

{

psvm (String [] args)

{

Int i=0;

For(int j=0; i< j<3; j++)

{

I = i+j;

}

sop(I + “ - ” + j); – CE: cannot find symbol variable j location class test

}

}

Class Test

{

psvm (String [] args)

{

Try

{

Int J = Integer.parseInt(“ten”);

}

Catch(NumberFormatException e)

{

j=10;

}

Sopen(j) – CE: cannot find symbol variable j location class test

}

}

For local variables JVM won’t provide default values compulsory we should perform initialization explicitly.

**Case 1:**

Class Test

{

psvm (String [] args)

{

Int x;

Sop(“Hello”);

}

}

Before using that variable , if we are not using than it is not required to perform initialization

**Case 2:**

Class Test

{

psvm (String [] args)

{

Int x;

Sop(x); - – CE: variable might not have been initialized

}

}

**Case 3:**

**Imp: - \*\*\*\***

**Class Test**

**{**

**psvm (String [] args)**

**{**

**Int x;**

**If(args.length > 0)**

**{**

**x = 10;**

**}**

**Sop(x); -**  – CE: variable might not have been initialized

**}**

**}**

**Case 4:**

**Imp: - \*\*\*\***

**Class Test**

**{**

**psvm (String [] args)**

**{**

**Int x;**

**If(args.length > 0)**

**{**

**x = 10;**

**}**

**Else**

**{**

**x = 20;**

**}**

**Sop(x); -**

**}**

**}**

**Java Test A B**

**o/p = 10**

**Java Test**

**o/p = 20**

Note:

1. It is not recommended to perform initialization for local variables inside logical blocks because there is not guaranteed for the execution of these blocks always at a run time.
2. It is highly recommended to perform initialization at the time of declaration at least with default value
3. The only applicable modifier for local variables is final by mistake if we are trying to apply any other modifier then we will get compile time error.

Class Test

{

psvm (String [] args)

{

Private Int x = 10;

public Int x = 10;

protected Int x = 10;

static int x = 10;

transient int x = 10;

volatile int x = 10;

for all above we get CE: illegal start of expression

final int x = 10; - allowed

}

}

By local variable only final modifier is applicable

Class Test

{

Int x = 10;

static int y = 20;

psvm (String [] args)

{

int z = 30;

}

}

If we are not declaring with any modifier then by default it is a default but this rule is applicable only for instance and static and not for local variables

**Conclusions:**

For instance and static variables JVM provide default values and we are not required to perform initialization explicitly but for local variables JVM don’t provide default values compulsory we should initialization explicitly before using that variable.

Instance and static variables can be accessed by multiple threads and hence these are not thread safe. But, in the case of local variables for every thread a separate copy will be created and hence local variables are thread safe.

|  |  |
| --- | --- |
| Type of variable | Thread-Safe ? |
| Instance variable | NO |
| Static variable | NO |
| Local variable | YES |

Class Test

{

Int x = 10;

Static string s = “durga”;

psvm (String [] args)

{

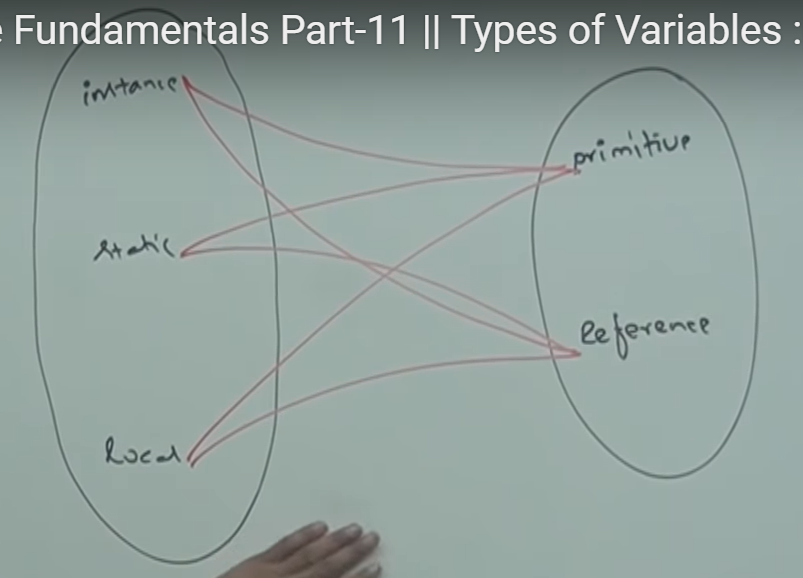
Int [] y = new int [3];

}

}

Every variable in java should be instance or static or local

Every variable in java should be either primitive or referenced hence, various variable possible combinations of variables in java are



Class Test

{

Int [] x = 10;

psvm (String [] args)

{

Test t = new Test();

Sop(t.x) // null

Sop(t.x[0]) RE: NPE (Null pointer exception)

}

}

**Case 1. Instance Level**

1. Int [] x;

1. Int [] x = new int [3];

Sop(obj.x) [I e3e25

Sop(obj.x[0]) // 0 default value for int type is 0

**Case 2. Static Level**

1. Static int [] x;

Sop(obj.x) // null

Sop(obj.x[0]) // RE: null pointer exception

1. Static Int [] x = new int [3];

Sop(obj.x) [I e3e25

Sop(obj.x[0]) // 0 default value for int type is 0

**Case 3. Local Level**

1. int [] x;

Sop(obj.x) // null

Sop(obj.x[0]) // RE: null pointer exception

1. Int [] x = new int [3];

Sop(obj.x) [I e3e25

Sop(obj.x[0]) // 0 default value for int type is 0

Once we creates an array every element is initialized with default value irrespective of whether it is instance or static or local array

\

**Tutorial 12:**

Var-args method

Variable number of argument methods.

Sum(10,20);

Class Test

{

psv sum (int … a)

{

}

}

Until 1.4 version we can’t declare a method with a variable number of arguments, if there is a change I number of arguments compulsory we should go for new method.

It increase a length of the code and reduces readability to overcome this problem some people introduce var args methods in 1.5 version according to this we can declare a method which can take variable number of arguments such type of method are called var args methods.

Int …m

We can declare a var args method as follows int …m

We can call this method by passing any number of int values including zero number.

M();

M(10);

M(10,20);

M(10,20,30);