Identifiers / Reserve Words

Date / / Page

→ int x = 10 ; ✓
int if = 20 ✗

→
```
int string = 888;
s.o.p (string);

Ans = 888
```
✓

→
```
int Runnable = 999;
s.o.p (Runnable);

Ans = 999
```
✓

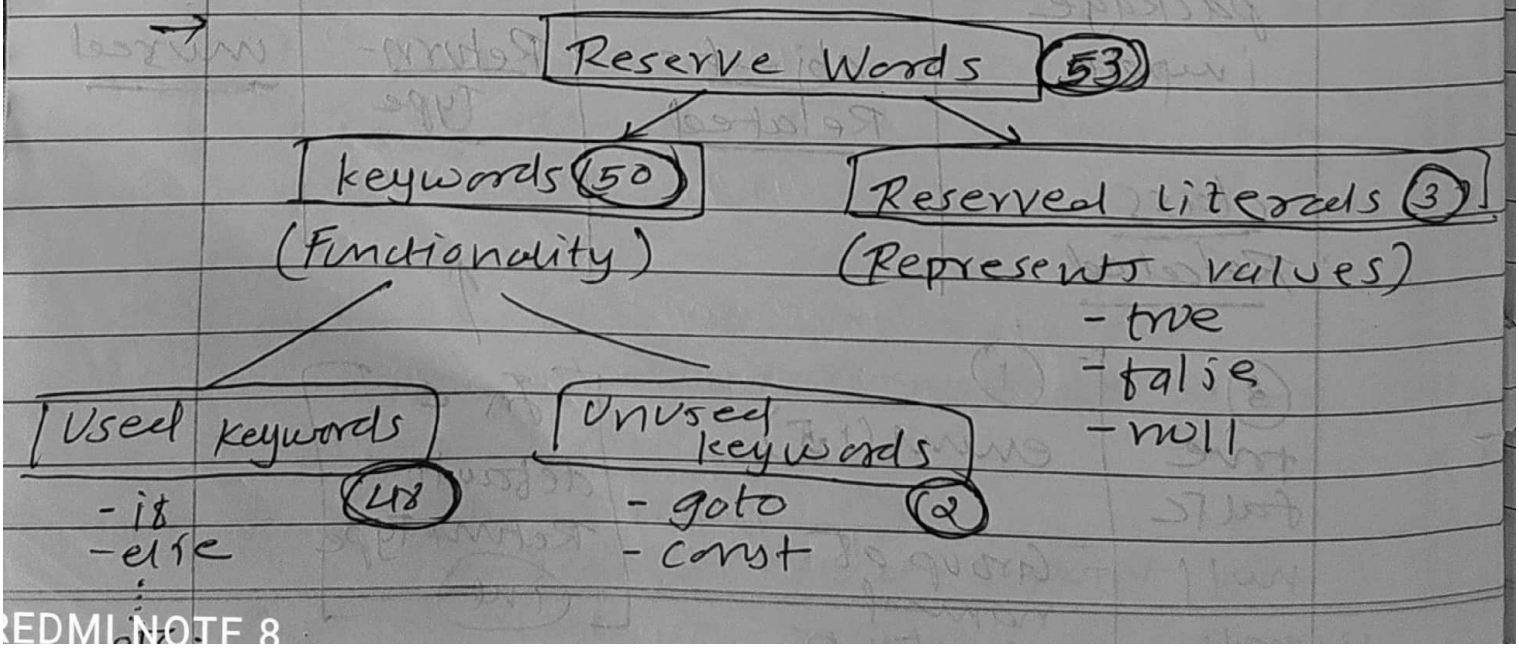
→ All pre-defined java class-name or Interface-name can be used as identifiers, but it is not re-commanded

Valid

→ Reserve words can't be used (if, else, for ... etc.) (53)
(keywords) → 50
Reserve-literal → 3 [true, false, null]

Invalid

→



Date ___/___/_____
Page _____

(8)	(11)	(11)	(6)
byte	if	public	try
short	else	protected	catch
char	switch	private	finally
int	case	static	throw
float	default	final	throws
double	break	abstract	assert (1.4)
long	while	synchronized	
boolean	do	native	
	for	transient	Exception-Handling
Data Types	continue	volatile	
	return	strictfp (1.2)	
	flow-control	Access Modifiers	

(6)	(4)	(1)	(2)
class	new		
interface	instanceof		
extends	super	void	goto
implements	this		const
package			
import			unused
	Object-Related	Return-Type	
class Related			

(3)	(1)
true	enum (1.5)
false	
null	Group of named constants
literals	

In c default Return Type (int)

Data Types

Date __/__/____
Page

→ Java is considered as (PURE OO Lang) when relative comparison is there (like with c++ etc)

But

when we consider Alone JAVA it is (not) (PURE OO Lang) as it does not support OO features like
- Op. overloading
- Mul. Inheritance

&

it also uses primitive data types as well.

Recommended Answer

(2 byte)

→ short data Type is best in earlier days of Java (1995) because of 8085/8086 Micro-processors which are of 16-bit so Read/write operations with short Data Type is well executed but

Today Those processors are out-dated, so short also get affected.

→ int is default data-type for (Integral no. in Java.

double is default for (floating-point no.)

so, they should always be in the range of int, Now,

```
int x = 2147483648l; (X)

Compile Time Error
  possible loss of precisim
    found : long
    required : int
```

```
int x = true; / int x = "abc"; (X)

Incompatible
```

→ long (8 bytes) $(-2^{63}$ to $2^{63}-1)$

file can have too many characters that can be out of range of int

so,

length method of file has Return Type (long) instead of (int). ★

```
long x =10L; (X)
PLP
F: long, R: int
```

→ ~~byte~~ boolean

boolean

```
~~byte~~ b = true; (✓)
```

boolean

```
~~byte~~ b = 0; / ~~byte~~ b = "abc"; (X)
boolean          boolean
Incompatible Types
  f : int / int, String
  r : boolean
```

→ $(-2^7$ to $2^7-1)$

→ (+127 to −128) (byte)

byte b = 10; (✓)

```
byte b = 128; (X)

Compile Time Error
  possible loss of Precision
    found : int
    required : byte
```

```
byte b = true; / byte b = "abc"; (X)

Compile Time Error
  Incompatible Types
    found : boolean / String
    required : byte
```

→ same for (short) $(-2^{15}$ to $2^{15}-1)$ (+32767 to −32768)

→ (int) (+2147483647 to −2147483648)

int x = 2147483647; (✓)

```
int x = 2147483648; (X)

Compile Time Error
  Integer Number Too Large
```

→ we expecting PLP Error found: long, req: int, (But)

Every Integral Number is considered as int by default

* Errors :-

1) Possible Loss of Precision (F:, R:)
2) Incompatible Types (F: R:)
3) Integer number Too Large
4) cannot Find Symbol
(Symbol:, Location:)

Literals

Integral literals

1) Decimal (base-10) (Default) (0 to 9)
2) Octal (base-8) (prefix = 0) (0 to 7)
3) Hexadecimal (base-16) (prefix = 0x) (0 to 9, a to f)
4) Binary (base-2), (prefix = 0b, 0B) (0 & 1)

The only thing where Java is not case-sensitive (0X, 0x, a to f, A to F)

1.7 version

```
int x = 10;   ✓
int x = 0786;   ✗
Integer Number too Large
```

Octal 8 can't take

✓ int x = 0777; (octal)

✓ int x = 0XFace; (Hexa)

✗ int x = 0XBeer;
Integer Number too Large

✗ ✗

```
boolean b = True;   ✗
```

- Here, True is not from any data Types like not int, not string. so compiler will check for variable declared named True.

so, Error will be

cannot Find Symbol

→ char (2 byte) 0 to 65535

old Lang
C/C++
ASCII
256 chars
8-bits
1-byte

Java
256 ASCII chars
+
More chars
= 2-bytes

* default value = 0 (space character)

```
char ch = null;   ✗
Incompatible Types
f: <null-type>
r: char
```

Date / / Page

★ → we can specify Octal, Hexadecimal, binary literals only with Integral numbers.

means,

double d = 0X123.456; ✗

compiler error
(Malformed floating point literal)

double d = 0123.456 ✓
works bez 0 gets ignore here.

→ double d = 10; ✓
S.O.P(d); ⟹ 10.0

double d = 0786; ✗
→ Integral literal
compiler Error
Integer no. Too large
so, 0 stays for octal & in octal 8 is not allowed

double d = 0777; ✓

Here, Integral literal which is octal value converted into decimal (base-10) value & then assigned to double

(10 ↑ int | byte b = 10; ↑ byte)

→ As compiler considers every integral number as (int) by default, there is no Explicit way possible of (byte), (short) literal

But, if we assign int literal to byte & it is in range of byte then compiler automatically takes it as an byte literal:
(like byte b=10;)
Now 10 is byte literal

But if only 10 is there it is int by default.
(same for short)

Floating point literals

→ float f = 123.456; ✗

PLP
F: double
R: float

float f = 123.456f; ✓
float f = 123.456F;

double d = 123.456; ✓
double d = 123.456D; ✓
double d = 123.456d; ✓

Date / /
Page

double d = 1.2e3; ✓
s.o.p(d); ⟶ 1200.0

float f = 1.2e3; ✗

PLP
F: double
R: float

float f = 1.2e3f; ✓

→ char literals

1st way

char ch = 'a'; ✓

char ch = a; ✗
C.E.
cannot find symbol
symbol: variable a
location: class class_name

char ch = "a"; ✗
C.E.
Incompatible types
F: String, R: char

char ch = 'ab'; ✗
C.E.1 unclosed char literal
C.E.2 unclosed char literal
C.E.3 not a java statement

double literal ∴ NO Octal ∴ Both will work

double d = 0786.0; ✓

(0 get ignored)

double d = 0xFace; ✓

↳ Bcz Integral Hex literal

double d = 0xFace.1; ✗

No, Hex floating point literal

Malformed floating point Error

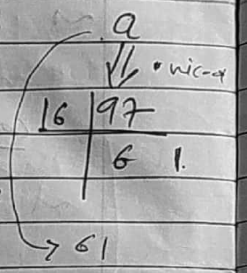
→ int x = 10.0; ✗

Compiler Error
PLP
F: double
R: int

→ Floating-point literals can be specified in exponential form like..

1.2e3
= 1.2 × 10³
= 1200.0 (⇒ double by default)

```
char ch = '\u0061';
S.O.P(ch);
    ⇓
    a
```



4th way

→ Every escape char is a valid char literal

```
char ch = '\n';   ✓
char ch = '\t';
```

```
char ch = 'm';   ✗
C.E:
    Illegal Escape char
```

⇒ Escape character in Java

Escape character	Description
\n	New line
\t	Horizontal tab
\r	carriage return
\b	Back space
\f	Form Feed
\'	Single quote
\"	Double quote
\\	Back slash

2nd way

we can represent char literal as an integral literal, which represent unicode value of char & that integral literal can be specified in either decimal, octal or hexadecimal form but allowed range is 0 to 65,535.

```
int a = 97;
char ch = a;   ✗
PLP, f: int, f: char
```
variable Assignment

```
char ch = 97;
S.O.P(ch);   ⇒ a   ✓   ← literal Assignment
```

```
char ch = 0xface;   ✓
char ch = 0777;   ✓
char ch = 65535;   ✓
```

```
char ch = 65536;   ✗
C.E  PLP
     g: int, f: char
```

3rd way

char literal as in Unicode Representation

```
'\uXXXX'
```
↳ 4-digit hexadecimal number.

Java 1.7 Enhancements of Literal

1) Binary Integral Literals
   (ob & OB) digits = (0, 1)
2) Underscore (_) in between literal values.
   (Improve Readability)

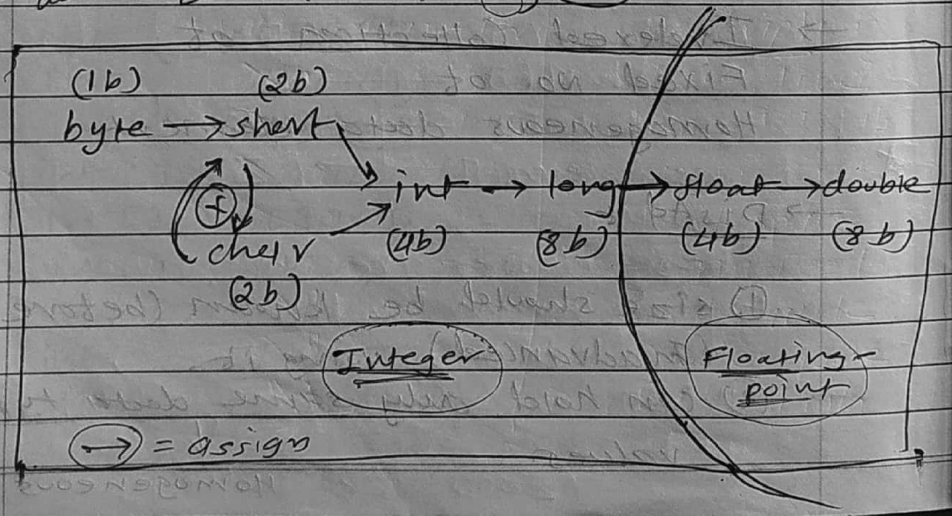
123456789.12 ⇝ 1_23_45_678.12

After compilation there (_) are removed automatically

double d = 1__23__45.7_8;

valid

double d = ⊖123.7_8; ✓
double d = 123(-).7_8; ✗
double d = 123.7_8(-); ✗
→ C.E.



⇝ O/P ⇒ This is ' symbol

code → s.o.p("This is \' symbol");

⇝ In file path

like ("C:\abc\pics"); ✗

("C:\\abc\\pics"); ✓

⇝ \n → next line first character

(four ways: 'a', 97, '\u0061', '\141')

char ch = 65356; ✗
char ch = 0xBeer; ✗ [✓]
char ch = \uface; ✗ ['']
char ch = '\ubeef'; ✓
char ch = '\m'; ✗ [illegal E-code]
char ch = '\itface'; ✗ [i] [u]✗

→ string literal

String s = "abc";

Scanned with CamScanner

Date / /
Page

8 byte long value can be assign to 4 byte float value

Bcz Both are following different Memory Representation Internally

```
float f = 10l;
S.o.p(f);
    ↳ 10.0
```

Even though short & char are of 2 bytes
→ short & char can't be assign to Each other as short has unsigned 32,768 numbers which char can't understand

(Next → L.F.S.6)

Array

→ Indexed Collection of Fixed No. of Homogeneous data element.

→ DisAd :

① size should be known (before)/ (In advanced) using it.
② can hold only same data type values.
↓
Homogeneous

① Declaration ② Creation ③ Initialization

Date __/__/____

Page _____

Declaration

1-D

✓ `int[] x;`
✓ `int []x;` (Here, compiler ignore the space betn int & [])
✓ `int x[];`

2-D

✓ `int[][] x;`
✓ `int [][]x;`
✓ `int x[][];`

✓ `int[] []x;`
✓ `int[] x[];`
✓ `int []x[];`

`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
`int[] []a, b[];` a→2, b→3
`int[] [a]a, []b;` ✗ C-E
`int[] [a]a, [b]b, []c;` ✗ ✗ ✗

3-D

`int[][][] a;`
`int [][][]a;`
`int a[][][];`
`int[] [][]a;`
`int[] a[][];`
`int[] []a[];`
`int[][] [][]a;`
`int[][] a[][];`
`int [][][]a;`
`int [][]a[];`

→ Array size can be (int) only.
(Whether it is long array or anything else) No bigger than (int)

→ int[] a = new int[-3];

Compiles Fine ✓ But
Runtime Exception:
NegetiveArraysizeException

Compiler will only check for valid int value.

→ int[] a = new int[10]; ✓

int[] a = new int['a']; ✓

char can be implicitly converted into int.
∴ int[97]

byte b=10;
int[] a = new int[b]; ✓
short s=10;
int[] a = new int[s]; ✓

int[] a = new int[10d]; ✗

P.I.P : CE
F: long
R: int

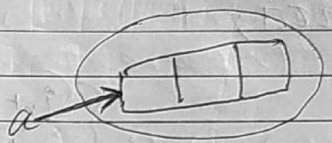
→ 2D Array Creation

In java, 2D Array **not** implemented using Matrix style.
Some people follows Array of Arrays for Multi-dimension Array.

→ Every Array in JAVA is Object.

Creation

int[] a = new int[3];



s.o.p(a.getClass().getName());
↳ [I

int[][] a = new int[3][2];
s.o.p(a.getClass().getName());
↳ [[I

Array Type	Corresponding class Name
int[]	[I
int[][]	[[I
double[]	[D
short[]	[S
byte[]	[B
boolean[]	[Z

★

- These classes are not available at programmer level.

- int[] a = new int[0]; ✓
(Zero size valid)

→ If you want to assign values directly to the array-variable by {} method it should on same line of declaration.

Array Initialization

```
int[] x = new int[3];
S.O.P(x); → [I@3d00c6   (Hex string)
S.O.P(x[0]); → 0 (default)
```

class_name @ hashcode_in_hex_forms

→ Declaration, creation, Initialization in single line

```
int[] a = {10,20,30};   ✓
int[] a;
a = {10,20,30};   ✗  C.E.
```

→ Anonymous Array

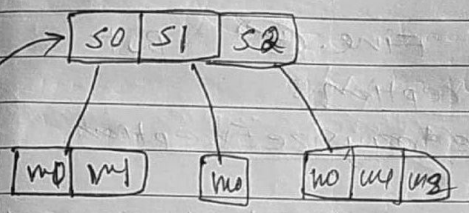
When Array is going use one time only, we should create anonymous array.
(Array without name)

(- Just for Instant use)
(- one time usage)

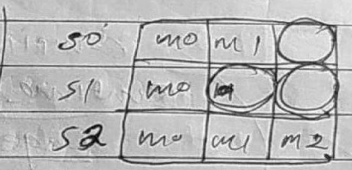
```
new int[]{10,20,30,40}
```

1D Anonym Array.

∴ Memory utilization Improved.

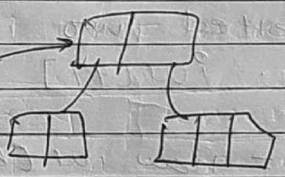


(Java)
(Mem. save)



(C/C++)
(Mem. waste)

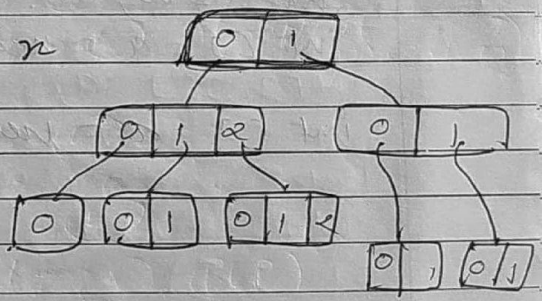
```
int[][] a = new int[2][];
a[0] = new int[2];
a[1] = new int[3];
```



3D-creation

```
int[][][] x =
    new int[2][][];
x[0] = new int[2][];
x[0][0] = new int[1];
x[0][1] = new int[2];
x[0][2] = new int[3];

x[1] = new int[2][2];
```



↳ Bcz its always 2 (given)

Case 2

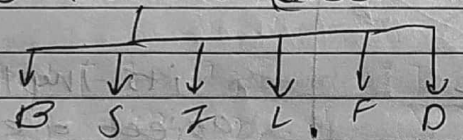
→ Object[] a = new Object[10];
a[0] = new Object(); ✓
a[1] = new String("durga"); ✓
a[2] = new Integer(10); ✓

(Child class obj can be assigned)

Case 3

→ Number[] n = new Number[10];
n[0] = new Integer(10); ✓
n[1] = new Double(10.5); ✓
n[2] = new String("abc"); ✗ (C.E: Incompatible)

java.lang.Number (Abstract class)



Case 4

→ Runnable() r = new Runnable(); ✗ (Bcz Interface)

But

Runnable[] r = new Runnable[10];
r[0] = new Thread(); ✓
r[1] = new String("abc"); ✗ (Incompatible)

Runnable
↑
Thread

new int[3]{10,20,30}; ✗

→ size should not be specified in Anonymo. Array

new int[][]{{10,20},{30,40}}

→ Based on our Requirement, we can give the name to Anonym. Array, but then it is no more Anonymous.

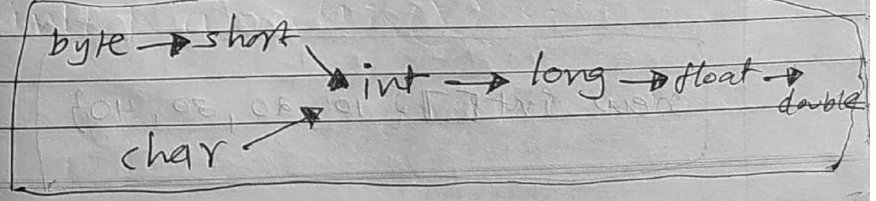
✓ int[] n = new int[]{10,20,30};

Array Element Assignments

Case 1

→ int[] n = new int[5];
n[0] = 10; ✓
n[1] = 'a'; ✓ (Implicit conversion 97)
byte b = 20;
n[2] = b; ✓
short s = 30;
n[3] = s; ✓
n[4] = 10.1; ✗ (PLP)

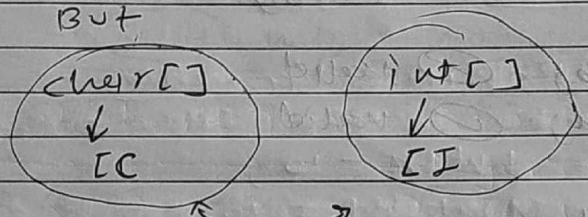
Based on,



Element level conversions are not applicable for Arrays, means

char → int possible ✓

But



No Relation.

~~Automatic conversion~~:

char → int	✓		
char[] → int[]	✗	(No Rel.)	
int → double	✓		
int[] → double	✗	(No Rel.)	
float → int	✗	(P-L-P)	
float[] → int[]	✗	(No Rel.)	
String → object	✓		
String[] → object[]	✓	(Bcz, parent child Rel)	

✓
```
String[] s = {"A", "B"};
Object[] o = s;
```

Array Type	Allowed element Type
primitive Array	Any type which can be implicitly promoted to declared type.
Object Type Array	Either declared type or its child class object
Abstract class Type Array	child class objects.
Interface Type Arrays	its implementation object are allowed.

Array Variable Assignment

Case 1:
```
int[] x = {10, 20, 30};
char[] ch = {'a', 'b', 'c'};
```

int[] b = x; ✓ valid

int[] a = ch; ✗ C.E.

Incompatible types

When Assigning One Array to Another Array

→ Dimension, → Type should match

But

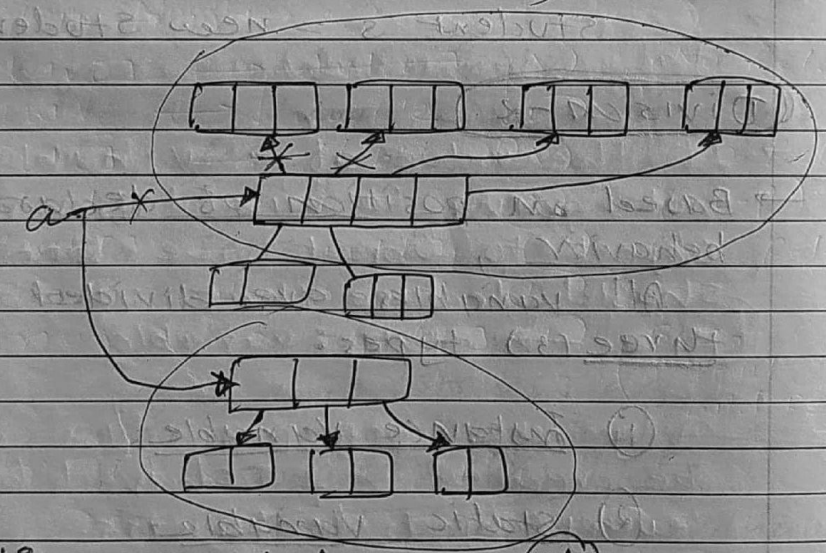
→ Size might not.

→ `int[][] a = new int[4][3];`

`a[0] = new int[2];`

`a[1] = new int[3];`

`a = new int[3][2];`



Object created ? → (11)

Object Eligible for GC ? → (7)

Next session → 10

Case-2

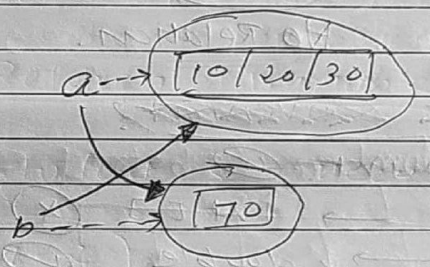
size might not match

`int[] a = {10, 20, 30};`

`int[] b = {70};`

`a = b;` ✓ valid

`b = a;` ✓ valid



Elements not reassigned, Reference will be Re-assigned

Case-3

Dimension must be match

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

`a[0] = new int[4][3];` ✗

C.E Incompatible Type

F: int[][]

R: int[]

`a[0] = 10;` ✗

C.E Incompatible Type

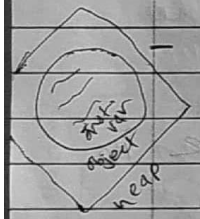
F: int

R: int[]

`a[0] = new int[2];` ✓

Instance Variable

- If value of a variable is varied from object to object, such variable → Instance variable
- For every object separate copy generated
- declared within class directly, outside of any method, block, constructor
- Created when Object created & Destroyed when Object gets destroyed (∴ scope is same as object.)
- Object saved in heap & so the instance variables are (as a part of object)




```
class Test
{
    int x=10;
    p s v m(String[] args)
    {
        S.O.P(x); (X) C.E.
    }
}
```

Non-static variable cannot be referenced from a static content

```
    Test t = new Test();
    S.O.P(t.x);
}
```

Types of Variables

Division-1

→ Based on type of value represented by variable, All variable can be divided into two (2) types:

1. Primitive Variable

```
int x=10;
float f1 = 10.5;
```

2. Reference Variable

```
Student s = new Student();
```

Division-2

→ Based on position of declaration & behavior, All variables are divided into three (3) types:

1. Instance Variable.
2. static variable.
3. local variable.

→ We should create static method when method only depends on its parameters & object state has No effect on it.

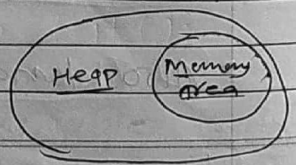
→ Created when class loads & Destroyed when class Unloads. (∵ scope is as same as .class file)

Java Test ⇒ command

1. start JVM. (Memory-area created)
2. Create & start main Thread
3. Locate Test.class file.
4. Load Test.class (static variable created)
5. Execute main() method.
6. unload Test.class (static variable destroyed)
7. Terminate main Thread.
8. shut down JVM.

Scope of static variable: 4 → 6

★ → static variables are stored in method Area.




```
public void m1()
{
    s.o.p(x);  (V) ⟺ sop(this.x);
}
}
```

→ Default values given to instance variable as per data type. (by JVM)

→ Instance variables known as Object level var. or Attributes

Static Variables

→ If the value of a variable is not going to varied from Object to Object; we should declare that variable as a → static variable

→ only one copy per class will be created & shared by every object of the class.

★ → static variable should be declared within the class directly, but outside of any method, constructor or block.

→ stored in Stack Memory.

→ Created while executing the block in which we declare it.
Destroyed automatically when execution of block complete.
scope is as same as block in which we declare it.

→ Default values are not given by JVM.
(∵ Before using we have to initialize)
(If we Not using, it is not Required)

```
class Test
{
    p s v main(String[] args)
    {
        try
        {
            int j = Integer.parseInt("ten");
        }
        catch (NumberFormatException e)
        {
            j=10;
        }
        S.O.p(j);
    }
}
```

C.E.
Cannot Find symbol
S: j
L: Test class

→ It is Never Recommanded to initialize the local variable inside if, else like blocks, bcz they are

→ Static variable can be access
By ① Object Name
② class Name (Recommand)
③ Directly (only within the class)

→ static variable can be accessed within class from both ① Static Area & ② Non-Static Area directly.

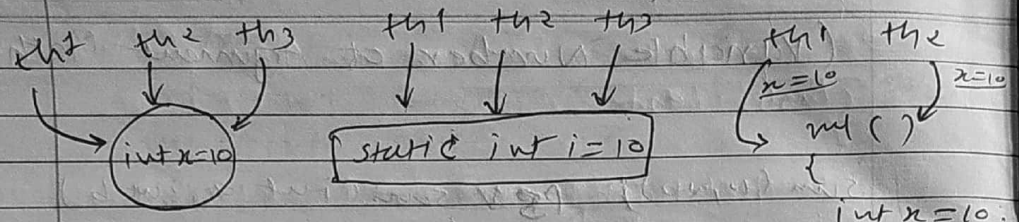
→ Default values are given to static variables also by JVM.

→ static variables also known as
class level var
Fields

Local Variables

→ To meet temporary requirements of programmer, we have to declared inside a
- methods,
- constructors or
- block

→ Also known as - Temporary Variable
- Stack Variable
- Automatic Variable

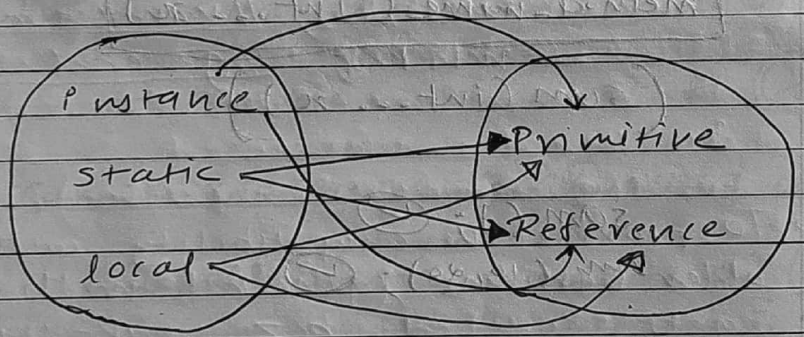


Thread-safe ✗ Thread-safe ✗ Thread-safe ✓

can be accessed by multiple thread simultaneously

separate copy for each Thread

③ Possible combinations of variable ⇒ ⑥



Division-2 Division-1

⇒ Once we create an Array, like Every Array Element gets its default values irrespective of where it declare (means it is instance, static, local)

```
{
    int[] a = new int[3];
    S.O.p(a[0]); ⟶ 0
}
```

not certain of executing at runtime.

→ Highly recommanded to perform initialization with declaration (like int x=10;)

→ The scope of local variable is already defined itself (that is only within the block it is declared) so, Access modifiers like (public, private, protected, default) are no meant to local variable.
∴ compiler give error if are do so.

∴ Only Applicable modifier for local variable is (final)

```
Illegal start of expression
{
    public int x=10;
}
```

⇒ Conclusions

① Default values

② Thread-Safety

Thread-safe if any modification done by one thread Not reflected in other)

Date / / Page

⇒ m1 (int ... x) ✓
m1 (int ... x) ✓ [space ignored]
m1 (int ...x) ✓
m1 (int x ...) ✗
m1 (int . ..x) ✗ } can't separate three dots (...)
m1 (int .x..) ✗

⇒ we can mix var-arg parameter with normal parameter

m1 (int x, int ... y) ✓
m1 (String x, double ... y) ✓

→ But var-arg should be last Parameter

m1 (double ... d, float f) ✗

→ Can't take more than (1) var-arg parameter

m1 (int ... x, double ... d) ✗

⇒ class Test
{
p s v m1 (int ... x)
{
s.o.p ("int ...");
}
p s v m1 (int[] x)
{
s.o.p ("int[]");
}
}

C.E. ✗ Can't declare both m1(int[]) & m1(int ...) simultaneously

★

Var-Arg Methods

(Variable Number of Argument Methods)

sum (10, 20); p s v sum (int a, int b)
{
s.o.p (a+b);
}

sum (10, 20, 30); p s v sum (int a, int b, int c)
{
s.o.p (a+b+c);
}

⇒ method_name (int ... x)

(sum (int ... x))

sum (); ✓
sum (10, 20); ✓

⇒ Internally var-arg parameter will be converted into 1D Array (int[] x)
so, we can differentiate values by using index x[0], x[1], ...

Array stays as Array But
var-arg converted to 1D Array

Date / / Page

⇒ Equivalence b/w var-arg method parameter & 1D Array

void

(m1(int[] x) ⇒ m1(int... x))

m1(int[] x)	m1(int... x)
can be called by m1(new int[]{10});	✓
✓ m1(new int[]{10,20});	✓
✓ m1(new int[]{10,20,30});	✓
	m1(); ✓
	m1(10); ✓
	m1(10,20); ✓

∴ so, we can replace 1D Array with var-arg parameter

Bcz compiler is internally converting var-args into arrays

p s v main(String[] args)

⇓

p s v main(String... args)

But Reverse Not True.

m1(int... x) ⇏ m1(int[] x)

⇒

```
class Test
{
    public static void m1(int... x)
    {
        S.O.P("var-arg method");
    }
    public static void m1(int x)
    {
        S.O.P("general method");
    }
    p s v main(String[] args)
    {
        m1(); // var-arg method
        m1(10,20); // var-arg method
        m1(10); // general method
    }
}
```

→ To provide the compatibility with older versions (general method) gets executed bcz var-args came in java 1.5 version (2014)

★ → so, var-arg method have least priority among all methods if no one executed, it will execute. (like as default in switch case)

Date / /
Page

⇒ can pass

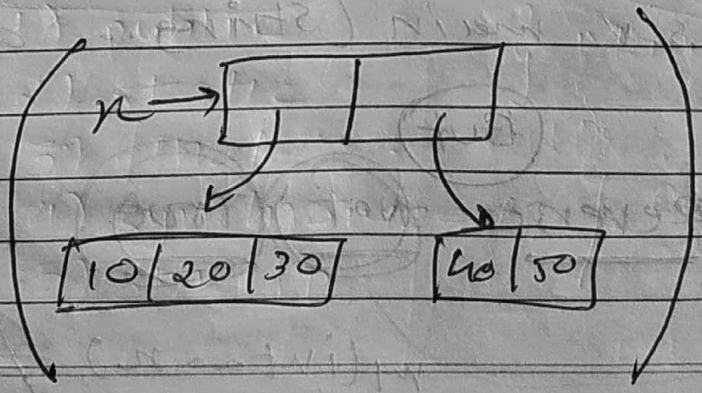
my(int... n) ⇒ int[] n

my(int[]... n) ⇒ int[][] n

```
class Test
{
    p s v main(String[] args)
    {
        int[] a = {10, 20, 30};
        int[] b = {40, 50};

        my(a,b);
    }
    p s v my(int[]... n)
    {
        for(int[] x : n)
        {
            S.o.p(x[0]);
        }
    }
}
```

O/P :- 10, 40



main() Method

Date / /
Page

→
```
class Test
{
}
```

javac Test.java ✓
compiles fine

java Test ✗
JVM gives Error as it is unable to find main() method

Exception occurs → (NoSuchMethodError : main)

→ JVM searches main of following PROTOTYPE

public static void main(String[] args)

- public: To call by JVM from anywhere
- static: without having object also, JVM has to call this method
- void: main() method won't required to return anything to JVM
- main: This is the name which is configured inside JVM-Program
- String[] args: Command-line Arguments

→ Any change in above prototype cause Runtime Exception saying NoSuchMethodError: main

Overloading of main() is possible.

- But JVM will execute only standeird main(),
- so, overloaded main() has to be called explicitly

```
class Test
{
    p s v main(string[] args)
    {
        s.o.p("string[]");
    }
    p s v main(int[] args)
    {
        s.o.p("int[]");
    }
}
```

o/p : string[]

Inheritance concept (Applicable) for main()

save with P.java

```
class P
{
    P S V m(string[] args)
    { s.o.p("parent main"); }
}
class C extends P
{ }
```

Acceptable Changes in main PROTOTYPE

1. Public static ⇒ static public (order of modifier)
2. main(String[] args) (Array declaration variation)
   - ↳ main(String []args)
   - ↳ main(String args[])
3. main(String [] durga) (valid identifier)
4. main(String... args) (var-arg parameter)
5. ★ (Adding new modifiers)

```
static final synchronized
strictfp public void main(String... args)
{
    s.o.p("valid");
}
```

So,

Modifiers acceptable with main()

- 1) public, 2) static } Mandatory
- 3) final, 4) synchronized, 5) strictfp } optional

Enhancements in main()

Date / /
Page

from Java 1.7 version

```
class Test
{
}
```

1.6 V	1.7 V
Javac Test.java ↵	Javac Test.java ↵
Java Test	Java Test
R.E: NSME: main	Error: Main method not found in class Test please define main method as public static void main(String[] args)

```
class Test
{
    static
    {
        s.o.p("static block");
    }
}
```

1.6 V	1.7 V
javac Test.java ↵	javac Test.java ↵
java Test	java Test
o/p: static block R.E: NSME: main	(same as above)

Conclusion: For main(), Overloading & Inheritance are Applicable, But Overriding is Not. Instead of it Method Hiding is there

Compile: javac p.java

p.class c.class

Run: java p java c

o/p: parent main o/p: parent main

New Concept of OOPS

```
class P
{
    p s v m(String[] args)
    { s.o.p("parent main"); }
}
class C extends P
{
    p s v m(String[] args)
    { s.o.p("child main"); }
}
```

It is (Method Hiding) Not (Method Overriding)

compile: javac P.java

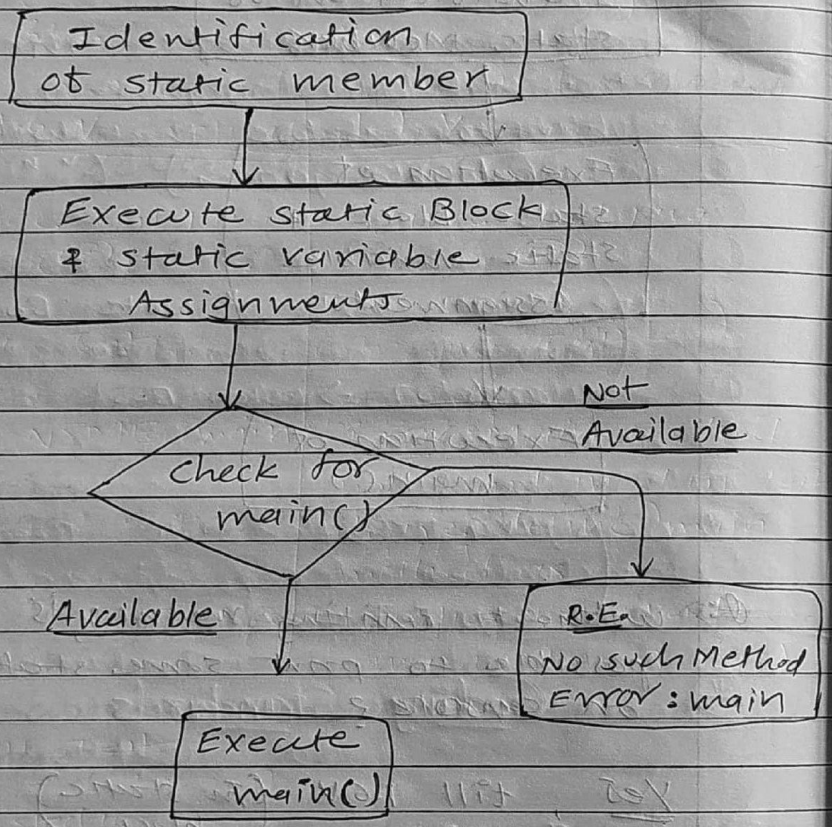
P.class C.class

Run: java P java C

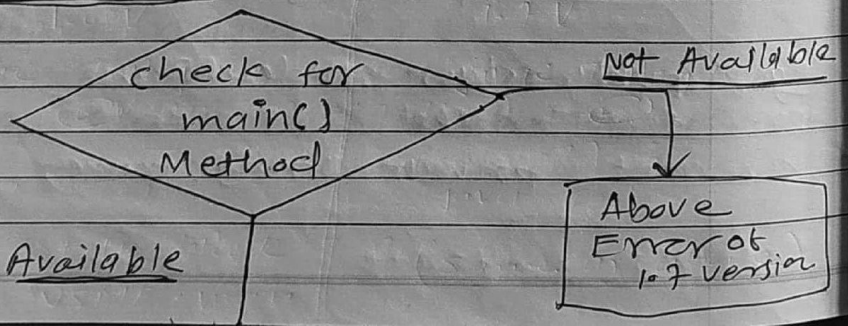
parent main child main

Conclusive Flow diagram

(1) 1.6 version



(2) 1.7 Version

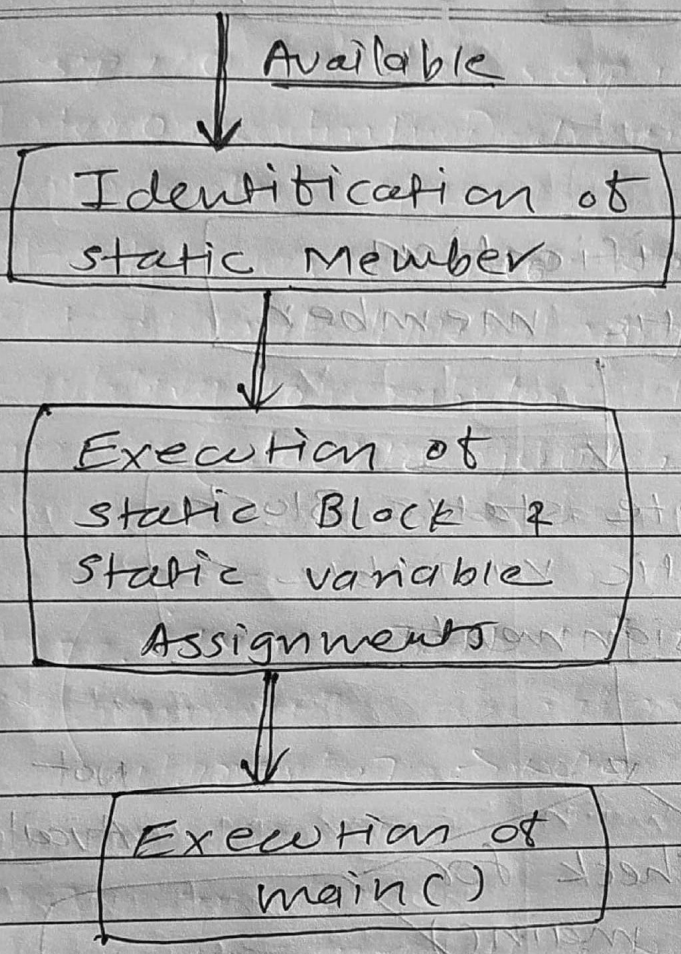



```
class Test
{
    static
    {
        S.O.P ("static block");
        System.exit(0);
    }
}
```

1.6 V	1.7 V
javac Test.java ✓	javac Test.java ✓
java Test O/P : static block	java Test (same as above)

```
class Test
{
    static
    {
        S.O.P ("static Block");
    }
    P S V main(String[] args)
    {
        S.O.P ("main method");
    }
}
```

1.6 V	1.7 V
javac Test.java ✓ java Test O/P : static block main method	javac Test.java ✓ java Test O/P : static block main method



Q: without writing main(), is it possible to print some statement on console?

Yes, till 1.6V (In static block)

No, Impossible from 1.7V

Command Line Arguments

Date __/__/____
Page ____

→ Java Test A B C

command line Argument

JVM creates Array of CLA.

```
p s v m (string [] args)
{
}
```

string[] args = {"A", "B", "C"};

(purpose)/

→ Need of command line Argument

To improve the functionality of main() & customize the behavior of main() method.

→ Why main() argument is decided to keep string type?

Bcz, strings are the most used object in java &

Its easy to convert string into other data type (int, boolean etc.)

```
public String name;

public void setName(String name)
{
    this.name = name;
}
public String getName()
{
    return name;
}
}
```

- Class name ends with 'Bean' (StudentBean here) is not official convension from SUN Microsystems. (But though it is valid)

(End of Language Fundamentals)

(Start Operators & Assignments)

→ space is the separater between two command line Argument

If we want space in one command line Argument, we have to enclose that with (double qoutes) (" ")

(java Test "Note Book"

args[0] ⇒ Note Book)

(java Test Note Book

args[0] ⇒ Note)

⊛ Coding Standards

→ A JavaBean is a simple java class with
- private properties &
- public getters & setters

Eg.
```
public class StudentBean
{
    private String name;
```