Arrays (20)

1. Array declaration
2. Array Creation.
3. Array Initialization.
4. Declaration, Creation, Initialization in a Single Line.
5. length vs length()
6. Annonymous Array
7. Array element assignments
8. Array variable assignments.

Array:-

→ An Array is an Indexed Collection of fixed no. of homogeneous data elements.

→ The main advantage of array is we can represent multiple values under the same name. So, that readability of the code improved.

→ But the main Limitation of array is once we created an array there is no chance of increasing/decreasing size based on our requirement. Hence memory point of view arrays concept is not recommended to use.

→ we can resolve this problem by using Collections.

1) Array declarations:-

(a) Single dimenshional Array declaration:-

✓ 1) int[] a;

✓ 2) int a[];

✓ 3) int []a;

→ 1st one recommanded because Type is clearly seperated from the Name.

→ At the time of declaration we can't specify the size.

Ex:- ✗ int[6] a;

(b) 2D Array declaration:-

✓ 1) int[][] a;

✓ 2) int [][]a;

✓ 3) int a[][];

✓ 4) int[] a[];

✓ 5) int[] []a;

✓ 6) int []a[];

c) 3D - Array declarations:-

1) int[][][] a;
2) int a[][][];
3) int [][][]a;
4) int[] [][]a;
5) int[] a[][];
6) int[] []a[];
7) int[][] []a;
8) int [][] a[];
9) int [][]a[];
10) int []a[][];

Q) Which of the following are valid declarations.

✓ 1) int[] a,b; a→1, b→1

✓ 2) int[] a[],b; a→2, b→1

✓ 3) int[] []a,b; a→2, b→2

✓ 4) int[] []a,b[]; a→2, b→3

✗ 5) int[] []a,[]b; a→2, b→3 C.E:-

→ If we want to Specify the dimension before the variable it is possible only for the first variable.

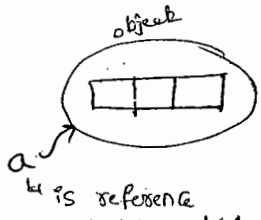
Ex:- int[] []a, []b;

Allowed — not allowed

2) Array Construction:-

→ Every array in Java is an object, hence we can create by using new operator.

ex:- int[] a = new int[3];



→ for every array type corresponding classes are available. but these classes are not applicable for programmer level.

Array type	Corresponding classname
① int[]	[I@----
② int[][]	[[I@----
③ double[]	[D@---
⋮	⋮

→ At the time of construction compulsary we should specify the size otherwise we will get C.E.

ex:- int[] a = new int[]; ✗ C.E:-

int[] a = new int[3]; ✓

→ It is legal to have an array with size 0 in java.

ex:- int[] a = new int[0]; ✓

→ If we are specifying array size as -ve int value, we will get runtime exception saying NegativeArraySizeException.

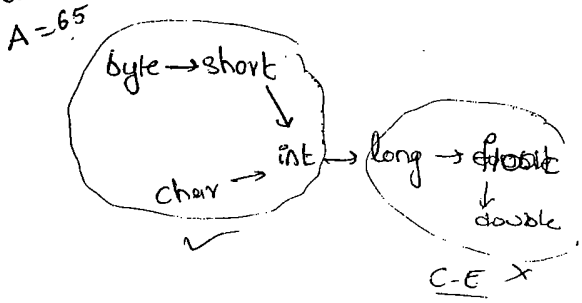
ex:- int[] a = new int[-6]; R.E:- NegativeArraySizeException

→ To Specify array size the allowed data-types are byte, short, int, char. If we are using any other type we will get C.E.

Ex: ① ✓ int[] a = new int['a']; a=97, A=65

② byte b = 10;
✓ int[] a = new int[b];

③ short s = 20;
✓ int[] a = new int[s];
✗ int[] a = new int[10L];
✗ int[] a = new int[10.5];



Note:-

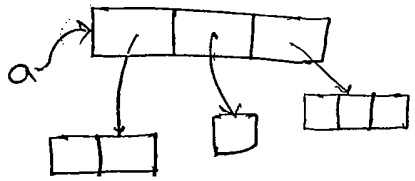
→ The max. allowed arraysize in java is 2147483647 (max. value of int datatype)

Creation of 2D-Arrays:-

→ In java multidimensional arrays are not implemented in matrix form. They implemented by using Array of Array Concept.

→ The main advantage of this approach is memory utilization will be improved.

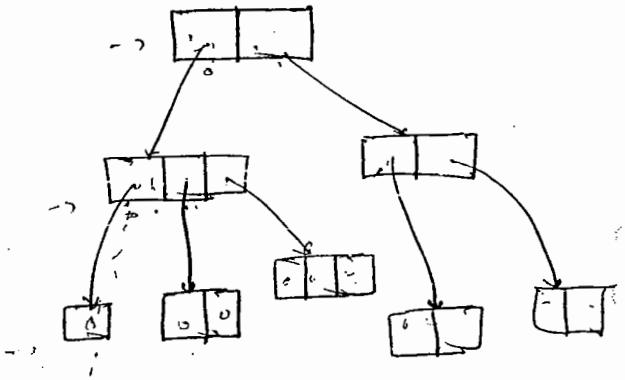
Ex:- int[][] a = new int[3][];
a[0] = new int[2];
a[1] = new int[1];
a[2] = new int[3];



Note:-
In C++, as

Ex 2:

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



Q:- which of the following Array declarations are valid?

✗ ① int[] a = new int[];

✓ ② int[][] a = new int[3][2];

✓ ③ int[][] a = new int[3][];

✗ ④ int[][] a = new int[][2];

✓ ⑤ int[][][] a = new int[3][4][5];

✓ ⑥ int[][][] a = new int[3][4][];

✗ ⑦ int[][][] a = new int[3][][5];

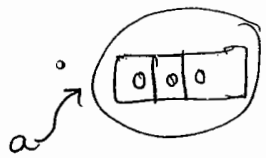
Array Initialization:-

→ Whenever we are creating an array automatically every element is initialized with default values.

ex(1):- int[] a = new int[3];

S.o.pln(a); [I@3e25a5 ↳ hashcode

S.o.pln(a[0]); 0

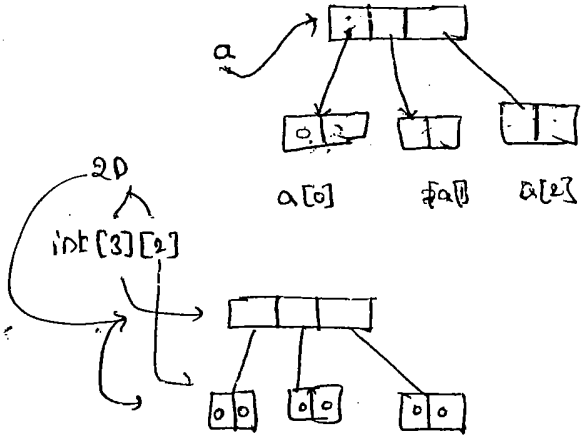


Note:- Whenever we are trying to print any object reference internally toString() will be call which is implemented as follows.

Classname @ hexadecimal_string_of_hashCode.

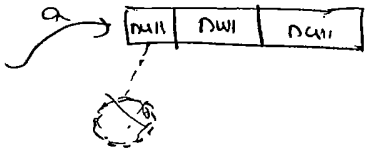
Ex (2):-

```
int[][] a = new int[3][2];
S.o.pln(a);        [[I@-----
S.o.pln(a[0]);     [I@4567
S.o.pln(a[0][0]);  0
```



Ex (3):-

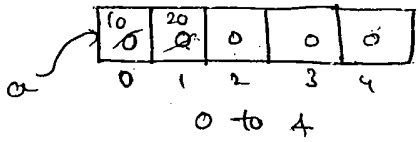
```
int[][] a = new int[3][];
S.o.pln(a);        [[I@-----
S.o.pln(a[0]);     null
S.o.pln(a[0][0]);  R.E:- NPE
```



→ Once we created an array every element by default initialized with default values. If we are not satisfy with those default values then we can override those with our customized values.

Ex:-

```
int[] a = new int[5];
a[0] = 10;
a[1] = 20;
a[3] = 40;
a[50] = 50;   → R.E: AIOBE
a[-50] = 60;  → R.E: AIOBE
a[10.5] = 30; → C.E:- PLP, found = double, required = int.
```



Note:-

→ If we are trying to access an array with out of range index we will get RuntimeException saying "AIOBE".

Array declaration, Construction & Initialization in a Single Line :-

→ We Can declare, Construct & Initialize an array into a Single line.

Ex(1):-

```
int[] a;
a = new int[3];
a[0] = 10;
a[1] = 20;
a[2] = 30;
a[3] = 40;
```

⟹ `int[] a = {10, 20, 30, 40};`

char

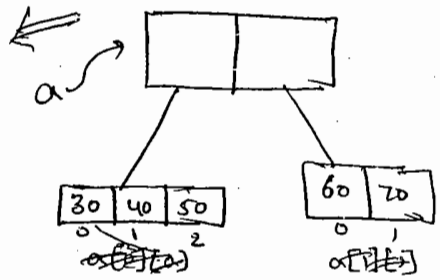
Ex(2):- `char[] ch = {'a', 'e', 'i', 'o', 'u'};`

`String[] s = {"Srinu", "Ravi", "Laxmi", "Sunder"};`

→ We Can Extend this shortcut Even for multidimensional arrays also.

Ex(3) :-

`int[][] a = {{30, 40, 50}, {60, 70}};`

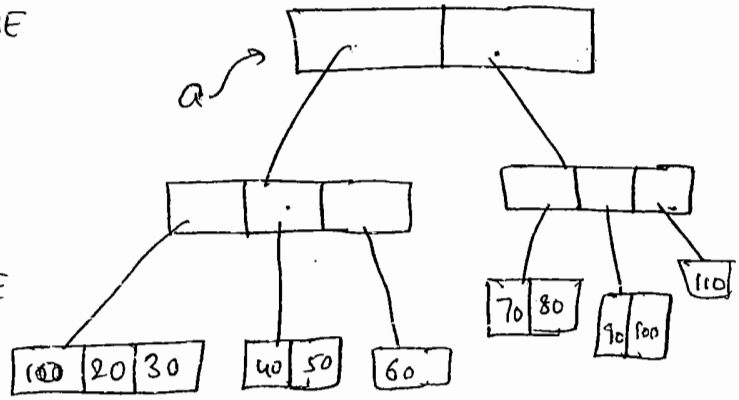


→ We Can Extend this Short Cut Even for 3D array also

Ex:-

`int[][][] a = {{{10, 20, 30}, {40, 50}, {60}}, {{70, 80}, {90, 100}, {110}}};`

Ex: int[][][] a = {{{10,20,30},{40,50},{60}},{{70,80},{90,100},{110}}};



S.o.pln(a[1][2][3]); RE:- AIOBE

S.o.pln(a[0][1][0]); 40

S.o.pln(a[1][1][0]); 90

S.o.pln(a[3][1][2]); RE:- AIOBE

S.o.pln(a[2][2][2]); RE:- AIOBE

S.o.pln(a[1][1][1]); 100

S.o.pln(a[0][0][1]); 20

S.o.pln(a[1][0][2]); R.E:- AIOBE

→ If we want to use Shortcut Compulsary we should perform declaration, Construction & initialization in a Single Line.

→ If we are using multiple lines we will get Compile-time Error.

Ex:-

int x=10; :-	int[] x = {10, 20, 30}; :-
✓ int x;	✓ int[] x;
✓ x=10	✗ x = {10, 20, 30};
	C.E:- Illegal Start of Expression.

length() vs length :-

length :-

→ It is a final variable applicable only for arrays.

→ It represents the size of array

Eg:- int[] a = new int[10];

S.o.pln(a.length); 10

S.o.pln(a.length()); C.E

Cannot find Symbol

Symbol : method length

location : class int[]

length() :-

→ It is a final method applicable only for String objects

→ It represents the no. of characters present in String.

eg:-

String s = "durga";

S.o.pln(s.length()); 5

S.o.pln(s.length);

↳ C.E:- Cannot find Symbol

Symbol : variable length

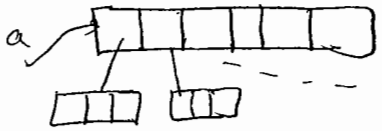
location : java.lang.String.

→ In multidimenshional arrays length variable represents only base size, but not total size.

Eg:- int[][] a = new int[6][3];

S.o.pln(a.length); 6

S.o.pln(a[0].length); 3



Note:-

→ length variable is applicable only for arrays where as length() is applicable for String objects.

Annonymous Array :-

→ Sometimes we can create an array with out name also such type of nameless arrays are called "Annonymus arrays".

→ The main objective of annonymous array is just for instant use. (not future) (only onetime)

→ We can create annonymous array as follows.

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

→ At the time of annonymous array creation we can't specify the size, otherwise we will get compile time error.

Ex:- ✗ new int[4]{10,20,30,40}.

Eg:-

```
Class Test
{
P.S.V.main(String[] args)
{
```

```
Sum(new int[]{10,20,30,40});
}
public static void Sum(int[] x)
{
  int total=0;
  for(int x1 : x)
  {
    total = total+x1;
  }
  S.o.pln("the Sum : "+total);   100
}
```

↳ Based on our requirement we can give the name for Anonymous array, then it is no longer Anonymous.

Eg:-

```
String[] s = new String[]{"A","B"};
S.o.pln(s[0]); A
S.o.pln(s[1]); B
S.o.pln(s.length); 2.
```

Array element assignments :-

Case(1) :-

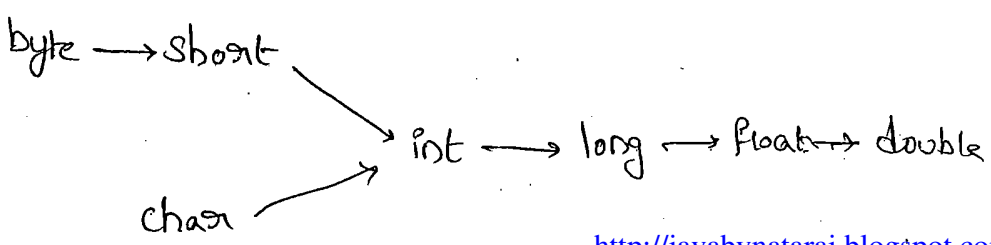
→ for the primitive type arrays as Array elements we can provide any type which can be promoted to declare type.

Eg:- ↳ for the int type arrays, the allowed Element types are byte, short, char, int. if we are providing any other type we will get Compiletime Error.

Eg(1) :-

```
int[] a = new int[10];
✓ a[0] = 10;
✓ a[1] = 'a';
  byte b = 10;
✓ a[2] = b;
  short s = 20;
✓ a[3] = s;
✗ a[4] = 10l;   C.E:- PLP
                       found : long
                       required : int
✗ a[5] = 10.5;  C.E:- PLP, found : double
                       required : int
```

Eg(2) :- for the float type array, the allowed Element types are byte, short, char, int, long, float.



Case(2):-

→ In the case of object type arrays as array elements we can provide either declared type or its child class objects.

eg1:- ① Number[] n = new Number[10];

✓ n[0] = new Integer(10);

✓ n[1] = new Double(10.5);

✗ n[2] = new String("durga"); → C.E:- Incompatible types
found : String
required : Number.

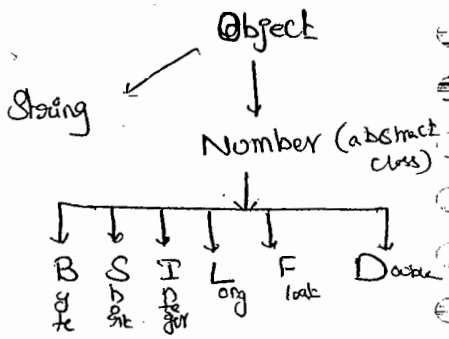
② Object[] a = new Object[10];

✓ a[0] = new Object();

✓ a[1] = new Integer(10);

✓ a[2] = new Double(10.5);

✓ a[3] = new String("durga");



Case(3):-

→ In the case of abstract class type arrays as array elements we can provide its child class objects.

Eg:- ① Number[] n = new Number[10];

✓ n[0] = new Integer(10);

✗ n[1] = new Number();

Case 4:-

→ In the case of Interface type array, as array element we can provide it's implementation class objects

Eg:- Runnable[] r = new Runnable[10];

r[0] = new Thread();

✗ r[1] = new String("durga"); C.E:- Incompatable types
found: String
Required: Runnable

Runnable (I)
↑
Thread (c)

Note:-

Array type	Allowed element type
1. Primitive type arrays	Any type which can be implicitly promoted to declared type.
2. Object type arrays	Either declared type objects or it's child class objects
3. abstract class type arrays	It's child class objects are allowed.
4. Interface type arrays	it's implementation class objects are allowed

Array Variable Assignment :-

Case(1):

→ Element level promotions are not applicable at array level

Eg:- A char value can be promoted to int type. But char array (char[]) can't be promoted to int[] type.

① int[] a = {10, 20, 30, 40};

char[] ch = {'a', 'b', 'c'};

✓ int[] b = a;

✗ int[] c = ch; C.E:- Incompatiable type
found : char[]
required : int[]

Q) Which of the following promotions are valid.

✓ ① char → int

✗ ② char[] → int[]

✓ ③ int → long

✗ ④ int[] → long[]

✗ ⑤ long → int

✗ ⑥ long[] → double[]

✓ ⑦ String (child) → Object (parent)

✓ ⑧ String[] → Object[]

Eg:- Child type array we can assign to the parent type variable

→ child type array we can assign to the parent type variable.

Eg(1) String[] s = {"A", "B", "C"};

✓ Object[] a = s;

Case(2):-

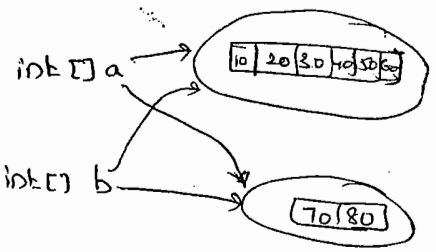
→ When even we are assigning one array to another array only reference variables will be reassign but not underlying elements. Hence types must be matched but not sized.

eg:- ① int[] a = {10, 20, 30, 40, 50, 60};

int[] b = {70, 80};

✓ ① a = b;

✓ ② b = a;



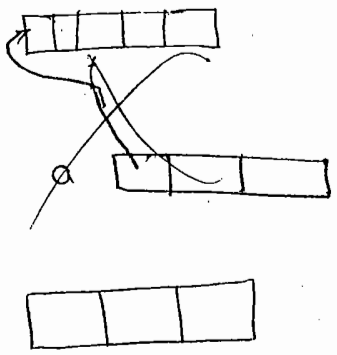
Eg(2):- int[][] a = new int[3][2];

a[0] = new int[5];

a[1] = new int[4];

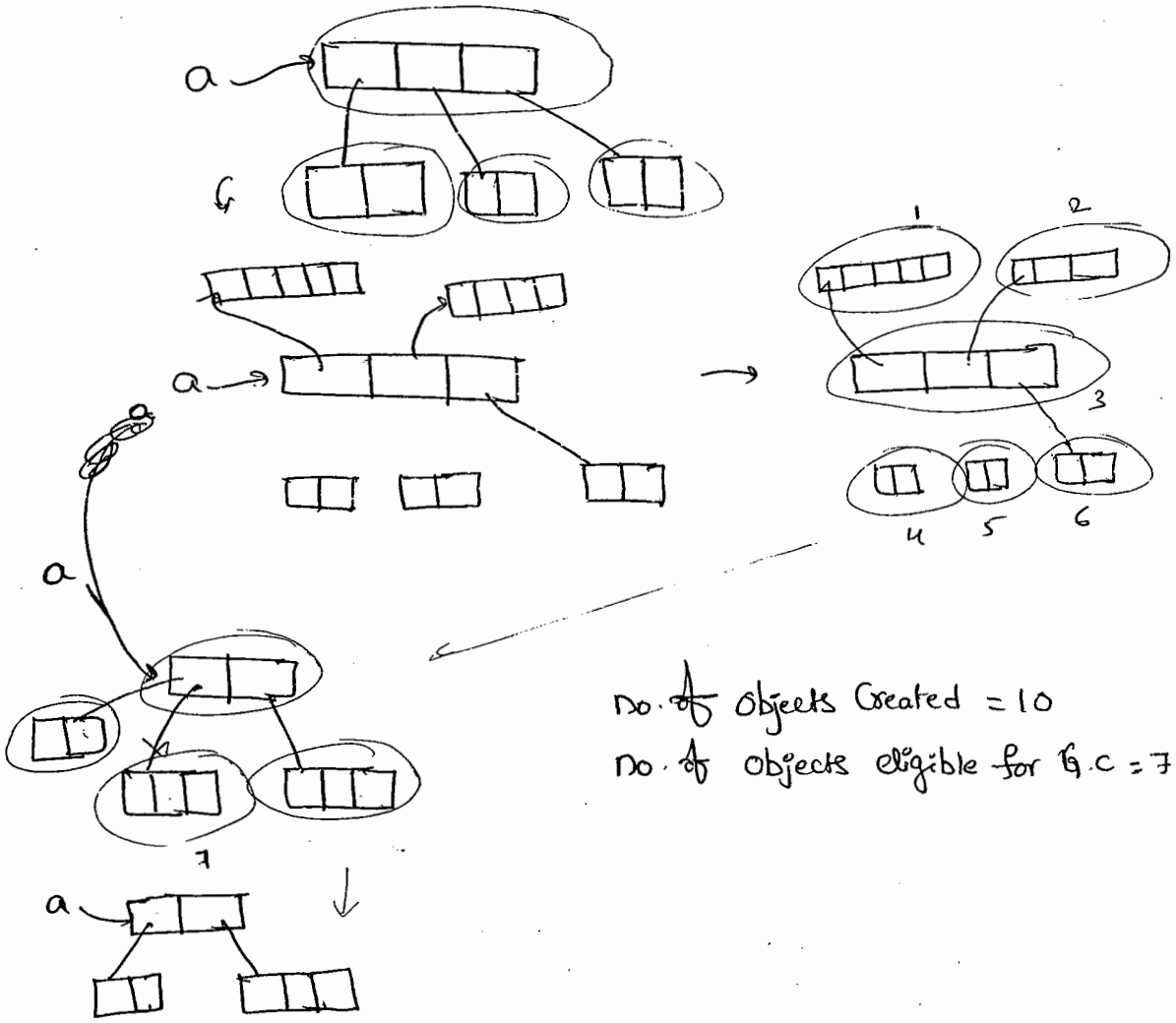
a = new int[2][3];

a[0] = new int[2];



No. of objects created = 10

No. of objects eligible for G.C = 7.



no. of objects Created = 10

no. of objects eligible for G.C = 7

Case 3:-

→ when ever we are performing array assignments dimensions must be matched, i.e in the place of single dimensional int[] array, only we should provide only single dimensional int[]. by mistake we are providing any other dimension we will get Compile time Error

eg:-

```
int[][] a = new int[3][];
a[0] = new int[3];
a[0] = new int[3][2];  ---> C.E: incompatible types
                             found : int[][]
                             required : int[]
a[0] = 0;
```

a[0] = 10; C.E:- incompatible types
found : int
required : int[]

22

Types of Variables

→ Based on the type of value represented by a variable, all variables are divided into 2 types.

(i) primitive variables
(ii) reference variables

(i) Primitive Variables:-

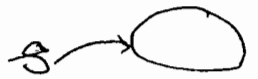
→ Can be used to represent primitive values

ex:- int x = 10;

(ii) reference variables:-

→ Can be used to refer Objects

ex:- Student s = new Student();



s is reference variable

→ Based on the purpose & position of declaration all variables are divided into 3 types.

(i) instance variables
(ii) Static variables
(iii) local variables.

(i) instance variable :-

→ If the value of a variable is varied from Object to object Such type of variables are Called instance variable.

→ for every Object a Seperate Copy of instance variable will be Created.

→ the Scope of instance variables is exactly Same as the Scope of the Objects. because Instance variables will be Created at the time of Objects Creation & destroy at the time of Objects destruction.

→ Instance variables will be Stored as the part of Objects.

→ instance variables should be declare with in the class directly, But outside of any method or Block or Constructor.

→ instance variables Cannot be accessed from ~~insta~~ Static area directly we Can access by using Object reference.

→ But from instance area we Can access instance members directly

Ex!.

```
Class Test
{
  int x=10;
  p.s.v.m(String[] args)
  {
    S.o.p/n(x); → C.E:- non-static variable x Cannot
                      be referenced from Static Context"
```

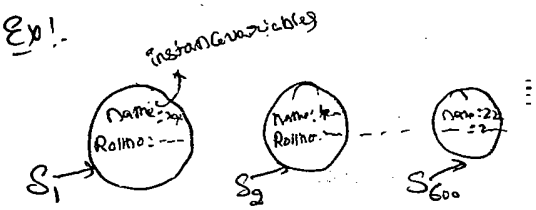
```
        Test t = new Test();
        S.o.pln(t.x); 10 ✓
    }
    public void m1()
    {
        S.o.pln(x); ✓ 10
    }
}
```

→ For the instance variables it is not required to perform initialization explicitly, JVM will provide default values.

eg:-

```
class Test
{
    String s;
    int x;
    boolean b;
p.s.v.m(String[] args)
{
    Test t = new Test();
    S.o.pln(t.s); null
    S.o.pln(t.x); 0
    S.o.pln(t.b); false
}
}
```

Ex!-



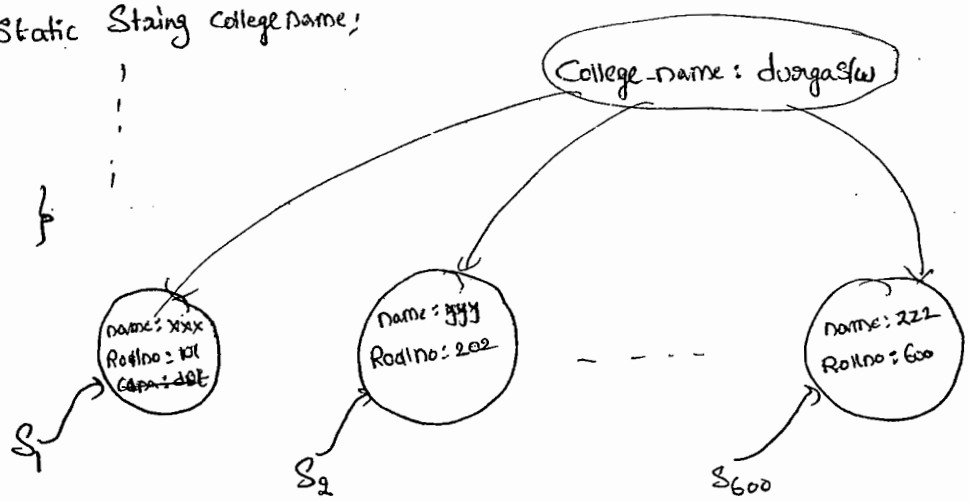
Students objects, In that name, Rollnos are instance variables, Bcz, these values are varied from object to object.

→ Instance variables also known as "Object level variables" or "attributes".

(ii) Static variables :-

ex:-

```
Class Student
{
    String name;
    int rollno;
    static String collegeName;
    :
    :
}
```



↳ If the value of a variable is not varied from Object to Object then it is never recommended to declare that variable at Object Level we have to declare such type of variables at class Level by using Static modifier.

→ In the case of instance variables for every Object a Seperate Copy will be Created, But in the Case of Static Variable Single Copy will be Created at class Level & The Copy will be Shared by all Objects of that class.

→ Static variables will be Created at the time of Class Loading & destory at the time of class unloading. Hence the scope of the static variable is

http://javabymantal.blogspot.com

Exactly Same as the Scope of the class.

Note:- Java Test ↵ execution process is

① Start jvm
② Create main Thread
③ Locate Test.class
④ Load Test.class → Static variables Creation
⑤ Execute main() method of Test.class
⑥ unload Test.class → Static variables destruction
⑦ Destroy main Thread
⑧ ShutDown Jvm

→ Static variables should be declare with in the class directly (but Outside of any method or Block or Constructor), with Static-modifier.

→ Static variables Can be accessed either by using class name or by using Object reference, but recommended to use Class name.

→ With in the Same class even if it's not required to use Class name. also we Can access directly.

Ex:-
```
class Test
{
    Static int x =10;
    p.s.v.main(String[] args)
    {
        S.o.pln(Test.x);  ✓ 10
        S.o.pln(x);  ✓ 10
        Test t = new Test();
        S.o.pln(t.x);  ✓ 10
    }
```

→ Static variables are Created at the time of Class loading. i.e, (at the begining of the program). Hence, we Can access from both instance & Static areas directly.

→ Eg:

```
Class Test
{
  Static int x=10;
  p.s.v.m(String[] args)
  {
    S.o.pln(x);
  }
  Public void m1()
  {
    S.o.pln(x);
  }
}
```

→ For the Static variables it is not required to perform initialization Explicitly, Compulsary JVM will provide default values.

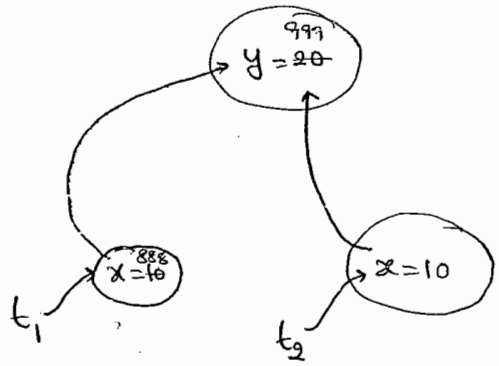
eg:-

```
Class Test
{
  Static int x;
  p.s.v.m(String[] args)
  {
    S.o.pln(x);  o
  }
}
```

→ Static variables will be stored in method-area. Static variables also known as "class-level variables" or "fields".

Ex:

```
Class Test
{
  int x=10;
  Static int y=20;
  P.S.V.M(String[] args)
  {
    Test t1=new Test();
      t1.x=888;
      t1.y=999;
    Test t2=new Test();
    S.o.pln(t2.x + "----" + t2.y);
  }
}
```



$t_1.x = 888$ $t_2.x = 10$

$t_2.y = 999$ $t_2.y = 999$

↳ If we performing any change for instance variables these changes wont be reflected for the remaining objects. because, for every object a seperate copy of instance variables will be their.

→ But, if we are performing any change to the static variable, these changes will be reflected for all objects because we are maintaining a single copy.

(iii) Local variables :-

→ To meet temporary requirements of the programmer sometimes we have to create variables inside method or block or constructor. Such type of variables are called local variables.

→ Local variables also known as stack variables or automatic variables or temporary variables.

→ Local variables will be stored inside a stack.

→ The local variables will be created while executing the block in which we declared it & destoryed once the block completed. Hence, the scope of local variable is exactly same as the block in which we declared it.

Ex :-

```
Class Test
{
  p.s.v.m(String[] args)
  {
    int i=0;
    for(int j=0 ; j<3 ; j++)
    {
      i = i+j;
    }
    S.o.pln (i + "----" + j);
  }
}
```

C.E :-
Can't find Symbol
Symbol : variable j
Location : Class Test

→ For the Local variables JVM won't provide any default values, Compulsory we should perform initialization Explicitly, before using that variable.

Eg:- ①

```
Class Test
{
  p.s.v.m(String[] args)
  {
    int x;
    S.o.pln("Hello");
  }
}
```

O/P:- Hello

```
Class Test
{
  p.s.v.m(String[] args)
  {
    int x;
    S.o.pln(x);
  }
}
```

C.E:- Variable x might not have been initialized.

Eg(2) :-

```
Class Test
{
  p.s.v.m(String[] args)
  {
    int x;
    if (args.length >0)
    {
      x = 10;
    }
    S.o.pln(x);
  }
}
```

C.E:- Variable x might not have been initialized

Eg 3:-

```
Class Test
{
  P.S.V.m(String[] args)
  {
    int x;
    if (args.length >0)
    {
      x=10;
    }
    else
    {
      x=20;
    }
    S.o.pln(x);
  }
}
```

O/P:- Java Test ↵

20

Java Test x y ↵

10

→ Note:-

→ It is not recommended to perform initialization of Local variables inside logical blocks because there is no garantee execution of these blocks at runtime.

→ It is highly recommended to perform initialization for the local variables at the time of declaration, at least with default values.

→ The only applicable modifier for the local variables is "final".

If we are using any other modifier we will get Compile-time Error.

Eg:-

```
Class Test
{
  p.s.v.m(String[] args)
  {
  ✗ private int x=10;
  ✗ public int x=10;
  ✗ protected int x=10;
  ✗ static int x=10;
  ✓ final int x=10;
  }
}
```

C.E:- Illegal Start of Expression.

UN Initialized Arrays:-

```
Class Test
{
int[] a;
p.s.v.m(String[] args)
{
  Test t1 = new Test();
  S.o.pln(t1.a); null
  S.o.pln(t1.a[0]); NullPointerException
}
}
```

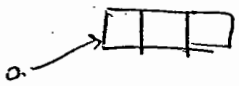
Instance level :-

int[] a ; S.o.p(obj.a) null

i.e a = null S.o.p(obj.a[0]) NullPointerException

int[] a = new int[3]; S.o.p(obj.a) [I@1a2b3

S.o.p (obj.a[0]) 0



Static level :-

Static int[] a; S.o.p(a); null

S.o.p(a[0]); NPE

Static int[] a = new int[3]; S.o.p(a); [I@1234

S.o.p(a[0]); 0

Explanation :-

int[] a; → here the array (i.e object) reference is Create but its not initialized (i.e object is not) Created. So JVM provides null value to the variable a.

int[] a = new int[3]; → here becoz of new operator we are Creating an Object and JVM by default provides '0' value in array

Local Level :-

int[]a ; S.o.p (a) S.o.p(a[0]) } C.E :- variable a might not have been initialized

int[] a = new int[3]; S.o.p(a) [I@1234

S.o.p(a[0]) 0

Note :- Once an Array is Created all its elements are always initialized with default values irrespective weather it is Static or instance or Local array.

8/03/10 (*) **Var-arg methods** (1.5 version)

→ Untill 1.4 version we can't declare a method with variable no. of arguements, if there is any change in no. of arguements compulsary we should declare a new method. This approach increases length of the code & reduces readability.

→ To resolve these problem Sun people introduced var-arg methods in 1.5 version. Hence from 1.5 version onwards we can declare a method with variable no. of arguements such type of methods are called var-arg methods.

→ We can declare var-arg method as follows.

```
m1(int... x)
```

→ We can invoke this method by passing any no. of int values including zero no. also.

Ex:- m1();
m1(10, 20);
m1(10);
m1(10, 20, 30, 40);

Ex(1):-

```
Class Test
{
  p.s.void m1(int... i)
  {
    S.o.pln("var-arg method");
  }
  p.s.v.m(String[] args)
  {
    m1();
    m2(10);
    m3(10, 20);
    m4(10, 20, 30, 40);
  }
}
```

o/p:- var-arg method
var-arg method
" "
" "

→ Internally var-arg method is implemented by using single dimensional arrays concept. Hence with in the var-arg method we can differenciate arrguements by using index.

Ex:-

```
Class Test
{
public static void sum(int... x)
{
 int total = 0;
 for(int y: x)
 {
  total = total+y;
 }
 S.o.pln("the sum: "+total);
}
p.s.v.m(String[] args)
{
 Sum();          0
 Sum(10,20);     30
 Sum(10,20,30)   60
 Sum(10,20,30,40);  100
}
}
```

O/P:

```
the sum : 0
the sum : 30
the sum : 60
the sum : 100
```

Case(i):-

Q) Which of the following var-arg method declarations are Valid.

m1(int... x) ✓

m1(int x...) ✗

m1(int ...x) ✓

m1(int. ..x) ✗

m1(int .x..) ✗

Case 2:-

→ We can mix var-arg parameter with normal parameter also.

ex:- m1(int x, String... y) ✓

Case 3:-

→ If we are mixing var-arg parameter with general parameter then var-arg parameter should be last parameter.

ex:- m1(int... x, String y) ✗

Case 4:-

→ In any var-arg method we can take only one var-arg parameter.

ex:- m1(int... x, String... y) ✗

Case 5:-

```
Class Test
{
  p.s.v.m1(int i)
  {
    S.o.pln("General method");
  }
  p.s.v.m1(int... i)
  {
    S.o.pln("var-arg");
  }
  p.s.v.m(String[] args)
  {
    m1();  var-arg
    m1(10); General (only)
    m1(10, 20); var-arg
  }
}
```

→ In General var-arg method will get Least priority i.e if no other method matched, then only var-arg method will get chance. This is Similar to default case inside Switch.

Case 6:-

Ex:-
```
class Test
{
  p-s-v-m1(int[] x)
  {
    S-o-pln("int[]");
  }
  p-s-v-m1(int... x)
  {
    S-o-pln("int...");
  }
}
```

C.E:- Can't declare Both m1(int[]) and m1(int...) in Test.

Var-arg Vs Single dimenshional arrays:-

Case(1):-

→ Wherever Single dimenshional array present we Can replace with var-arg Parameter.

Ex:- m1(int[] x) ⟹ m1(int... x) ✓

main(String[] args) ⟹ main(String... x) ✓

Case 2:-

→ where ever var-arg parameter present we Can't replace with Single dimenshional array.

✗ m1(int... x) ⟹ m1(int[] x)