JAVA

Note

By Bhushan Laware
bhushanlaware@gmail.com

Contents

1. Language Fundamentals

1.1 Identifier

A name in java program is called identifier. It may be class name, method name, variable name and label name.

Example:

```
public class Main
{
        public static void main(String[] args) {
                int x=10;
                System.out.println("Hello World");
        }
}
```

▸So here, number of identifiers are 5.

Rules to define java identifiers

A. The only allowed characters in java identifiers are
   a. a to z
   b. A to Z
   c. 0 to 9
   d. _
   e. $

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

Example:

1. Total_number ▸valid
2. Total# ▸invalid

B. Identifiers are not allowed to starts with digit.

Example:

1. ABC123 ▸valid
2. 123ABC ▸invalid

C. java identifiers are case sensitive up course java language itself treated as case sensitive language

Example:

```
class Test
{
        int number=10;
        int Number=20;
        int NUMBER=20;        we can differentiate with case.
        int NuMbEr=30;
        Int numBER=40;
}
```

D. There is no length limit for java identifiers but it is not recommended to take more than 15 lengths.

E. We can't use reserved words as identifiers.

Example:

int if = 10 ▸invalid

F. All predefined java class names and interface names we use as identifiers.

Example 1:

```
class Test
{
    public static void main(String[] args){
        int String=10;
        System.out.println(String);
    }
}
```

▸Output: 10

Example 2:

```
class Test
{
    public static void main(String[] args){
        int Runnable=10;
        System.out.println(Runnable);
    }
}
```

▸Output: 10

Even though it is legal to use class names and interface names as identifiers but it is not a good programming practice.

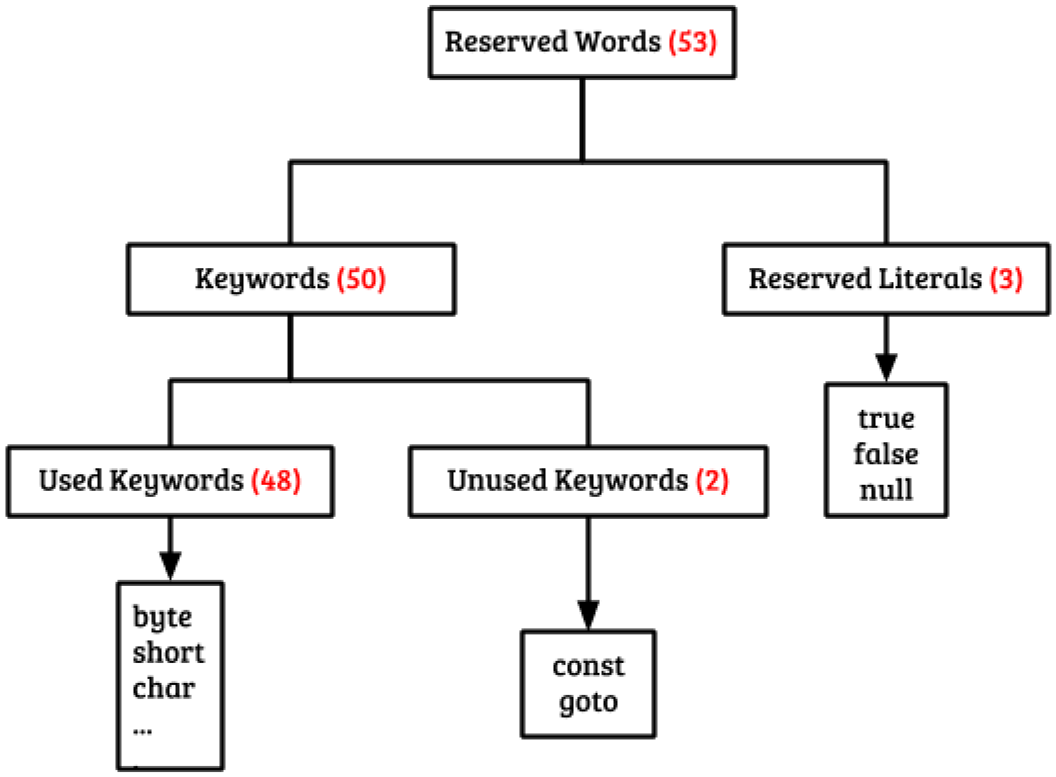
★ Which of the following are valid java identifiers?

1.	_$_	▸valid
2.	Ca$h	▸valid
3.	java2Share	▸valid
4.	all@hands	▸invalid
5.	123abc	▸invalid
6.	Total#	▸invalid
7.	int	▸invalid
8.	Integer	▸valid

1.2 Reserved words

In java some identifiers are reserved to associate some functionality or meaning such type of reserved identifiers are called reserved words.

Diagram:



Reserved words for data types:

1. byte
2. short
3. int
4. long
5. float
6. double
7. char
8. boolean

Reserved words for flow control:

1. if
2. else
3. switch
4. case
5. default
6. for
7. do
8. while
9. break
10. continue
11. return

Keywords for modifiers:

1. public
2. private
3. protected
4. static
5. final
6. abstract
7. synchronized
8. native
9. strictfp (1.2v)
10. transient
11. volatile

Keywords for exception handling:

1. try
2. catch
3. finally
4. throw
5. throws
6. assert (1.4v)

Class related keywords:

1. class
2. package
3. import
4. extends
5. implements
6. interface

Object related keywords:

1. new
2. instanceof
3. super
4. this

void return type keyword:

If a method won't return anything compulsory that method should be declared with the

void return type in java but it is optional in C++.

Unused keywords:

1. goto: Create several problems in old languages and hence it is banned in java.
2. Const: Use final instead of this.

By mistake if we are using these keywords in our program we will get compile time error.

Reserved literals:

1. true ▸values for boolean data type.
2. false ▸values for boolean data type.
3. null ▸default value for object reference.

Enum:

This keyword introduced in 1.5v to define a group of named constants

Example:

```
enum Beer
{
        KF, RC, KO, FO;
}
```

Notes:

★ **All reserved words in java contain only lowercase alphabet symbols.**

★ **New keywords are:**

1. strictfp → 1.2v
2. assert → 1.4v
3. enum → 1.5v

★ Which of the following list contains only java reserved words?

1. final, finally, finalize
   - ➢ Invalid. Here finalize is a method in Object class.
2. throw, throws, thrown
   - ➢ Invalid. thrown is not available in java.
3. break, continue, return, exit
   - ➢ Invalid. exit is not reserved keyword.
4. goto, constant
   - ➢ Invalid. Here constant is not reserved keyword.
5. byte, short, Integer, long
   - ➢ Invalid. Here Integer is a wrapper class.
6. extends, implements, imports
   - ➢ Invalid. imports keyword is not available in java.
7. finalize, synchronized
   - ➢ Invalid. finalize is a method in Object class.
8. instanceof, sizeOf
   - ➢ Invalid. sizeOf is not reserved keyword.
9. new, delete
   - ➢ Invalid. delete is not a keyword.
10. None of the above
    - ➢ Valid.

★ Which of the following are valid java keywords?

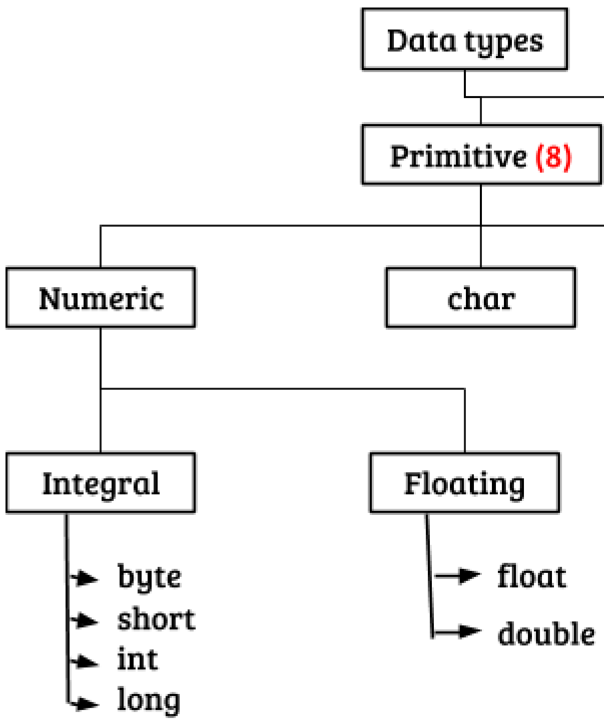
1. public (valid)
2. static(valid)
3. void (valid)
4. main (invalid)
5. String (invalid)
6. args (invalid)

1.3 Data Types

Every variable has a type, every expression has a type and all types are strictly define moreover every assignment should be checked by the compiler by the type compatibility hence java language is considered as strongly typed language.

- **Java is pure object oriented programming or not?**
  - ➢ Java is not considered as pure object oriented programming language because several oops features (like multiple inheritance, operator overloading) are not supported by java moreover we are depending on primitive data types which are non objects.

Diagram:



★ Note:

Except Boolean and char all remaining data types are considered as signed data types because we can represent both “+ve” and”-ve” numbers.

A. byte:

- ❑ Size: 1 Byte (8 bits)
- ❑ Max-value: +127
- ❑ Min-value: -128
- ❑ Range: 128 to 127 [-2^7 to 2^7-1]

Example:

byte b=10;	▸✔
byte b2=130;	▸C.E: possible loss of precision
byte b=10.5	▸C.E: possible loss of precision
byte b=true;	▸C.E: incompatible types
byte b="durga";	▸C.E: incompatible types

- byte data type is best suitable if we are handling data in terms of streams either from the file or from the network.

B. short:

- ❑ Size: 2 Bytes
- ❑ Range: -32768 to 32767 (-2^{15} to $2^{15}-1$)

Example:

short s=130;	▸✔
short s=32768;	▸C.E: possible loss of precision
short s=true;	▸C.E: incompatible types

- The most rarely used data type in java is short.
- Short data type is best suitable for 16 bit processors like 8086 but these processors are
- completely outdated and hence the corresponding short data type is also out data type.

C. Int:

- ❑ Size: 4 bytes
- ❑ Range: -2147483648 to 2147483647 (-2^{31} to $2^{31}-1$)

Example:

int i=130;	▸✔
int i=10.5;	▸C.E: possible loss of precision
int i=true;	▸C.E: incompatible types

- This is most commonly used data type in java.

D. long:

- ❑ Size: 8 bytes
- ❑ Range:-2^{63} to $2^{63}-1$

Example:

long l= 13l;

- To hold the no. Of characters present in a big file int may not enough hence the return type of length() method is long.

  long l=f.length() ; ▸f is a file

- Whenever int is not enough to hold big values then we should go for long data type.

★ **Note:**

All the above data types (byte, short, int and long) can be used to represent whole numbers. If we want to represent real numbers then we should go for floating point data types.

E. Floating Point Data types:

float	**double**
If we want to 5 to 6 decimal places of accuracy then we should go for float.	If we want to 14 to 15 decimal places of accuracy then we should go for double.
Size: 4 bytes	Size: 8 bytes
Range: -3.4e38 to 3.4e38	Range: -1.7e308 to1.7e308.
float follows single precision.	double follows double precision

F. boolean data type:

- ❏ Size: Not applicable (virtual machine dependent)
- ❏ Range: Not applicable but allowed values are true or false.

Example 1:

```
boolean b=true;          ▸✔
boolean b=True;          ▸C.E:cannot find symbol
boolean b="True";            ‣C.E:incompatible types

boolean b=0;                 ‣C.E:incompatible types
```

Example 2:

```
int x=10;
if( x )
        System.out.println("Hello");
else
        System.out.println("HI");
```

‣Compiler error: incompatible types

Found: int

Require: boolean

Example 3:

```
while(1)
        System.out.println("Hello");
```

‣Compiler error: incompatible types

Found: int

Require: boolean

G. Char data type:

- ❏ Size: 2 bytes
- ❏ Range: 0 to 65535

Example:

```
char ch1=97;          ‣✓

char ch2=65536;       ‣C.E:possible loss of precision
```

- In java we are allowed to use any worldwide alphabets character and java is Unicode based to represent all these characters one byte is not enough compulsory we should go for 2 bytes.

Summary of java primitive data type:

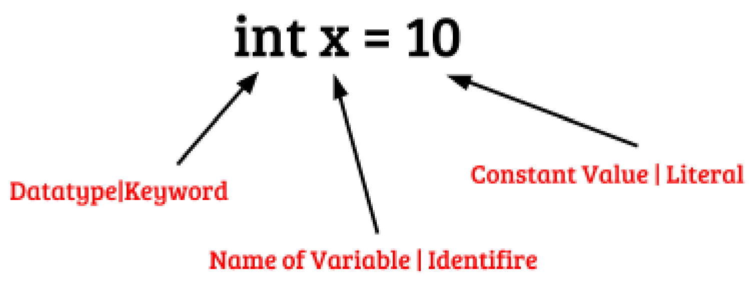
Data Type	Size	Range	Wrapper Class	Default Value
byte	1 byte	-2^{7} to -2^{7} -1 (-128 to 127)	Byte	0
short	2 byte	-2^{15} to -2^{15} -1 (-32768 to 32767)	Short	0
int	4 byte	-2^{31} to -2^{31} -1 (-2147483648 to -2147483647)	Integer	0
long	8 byte	-2^{61} to -2^{61} -1	Long	0
float	4 byte	-3.4e38 to 3.4e38	Float	0.0f
double	8 byte	-1.7e308 to 1.7e308	Double	0.0
boolean	Not applicable	true/false	Boolean	false
char	2 byte	0 to 65535	Character	0 (Blank Space)

★ The default value for the object references is “null”.

1.4 Literals

Any constant value which can be assigned to the variable is called literal.

Example:



A. Integral Literals:

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

1. **Decimal literals:** Allowed digits are 0 to 9.

   Example: int x=10;

2. **Octal literals:** Allowed digits are 0 to 7. Literal value should be prefixed with zero.

   Example: int x=010;

3. **Hexadecimal literals:** The allowed digits are 0 to 9, A to Z. For the extra digits we can use both upper case and lower case characters. This is one of very few areas where java is not case sensitive. Literal value should be prefixed with ox(or)oX.

Example: int x=0x10;

★ **These are the only possible ways to specify integral literal.**

★ **Which of the following are valid declarations?**

a. int x=0786; ‣C.E:integer number too large: 0786 ‣invalid
b. int x=0xFACE; ‣Valid
c. int x=0xbeef; ‣Valid
d. int x=0xBeer; ‣C.E: ';' expected ‣invalid
e. int x=0xBeer; ‣Valid
f. int x=0xabb2cd; ‣Valid

★ **Example 1:**

```
int x=10;

int y=010;

int z=0x10;

System.out.println(x+"----"+y+"----"+z);
```

‣Output: 10----8----16

➢ By default every integral literal is int type but we can specify explicitly as long type by suffixing with small "l" (or) capital "L".

1. int x=10; ‣valid
2. long l=10L; ‣valid
3. long l=10; ‣valid
4. x=10l; ‣invalid ‣C.E:possible loss of precision

➢ There is no direct way to specify byte and short literals explicitly. But whenever we are assigning integral literal to the byte variables and its value within the range of byte compiler automatically treats as byte literal. Similarly short literal also.

1. byte b=10; ‣valid
2. byte b=130; ‣Invalid ‣C.E:possible loss of precision
3. short s=32767; ‣valid
4. short s=32768; ‣Invalid ‣C.E:possible loss of precision

B. **Floating Point Literals:**

- Floating point literal is by default double type but we can specify explicitly as float type by suffixing with f or F.

  Example:

  1. float f=123.456; ‣Invalid ‣C.E:possible loss of precision
  2. float f=123.456f; ‣valid
  3. double d=123.456 ‣valid

- We can specify explicitly floating point literal as double type by suffixing with d or D.

  Example:

  1. double d=123.456D;

- We can specify floating point literal only in decimal form and we can't specify in octal and hexadecimal forms.

  Example:

  1. double d=123.456; ‣valid
  2. double d=0123.456; ‣valid
  3. double d=0x123.456; ‣Invalid ‣C.E:possible loss of precision

★ **Which of the following floating point declarations are valid?**

1. float f=123.456; ▸Invalid ▸C.E:possible loss of precision
2. float f=123.456D; ▸Invalid ▸C.E:possible loss of precision
3. double d=0x123.456; ▸Invalid ▸C.E:malformed floating point literal
4. double d=0xFace; ▸valid
5. double d=0xBeef; ▸valid

➢ We can assign integral literal directly to the floating point data types and that integral literal can be specified in octal and Hexadecimal form also.

Example:

```
double d=0xBeef;
System.out.println(d);
▸Output: 48879.0
```

But we can't assign floating point literal directly to the integral types.

Example:

int x=10.0; ▸Invalid ▸C.E:possible loss of precision

➢ We can specify floating point literal even in exponential form also(significant notation).

Example:

```
double d=10e2;          ▸valid
System.out.println(d);  ▸Output: 1000.0
float f=10e2;           ▸Invalid    ▸C.E:possible loss of precision
float f=10e2F;                      ▸valid
```

C. Boolean literals:

The only allowed values for the boolean type are true (or) false where case is important.

Example:

1.	boolean b=true;	‣valid	
2.	boolean b=0;	‣Invalid	‣C.E:incompatible types
3.	boolean b=True;	‣Invalid	‣C.E:cannot find symbol True
4.	b="true";	‣Invalid	‣C.E:incompatible types

D. Char literals:

➢ A char literal can be represented as single character within single quotes.

Example:

1.	char ch='a';	‣valid	
2.	char ch=a;	‣Invalid	‣C.E:cannot find symbol a
3.	char ch="a";	‣Invalid	‣C.E:incompatible types
4.	char ch='ab';	‣Invalid	‣C.E:unclosed character literal

➢ We can specify a char literal as integral literal which represents Unicode of that character. We can specify that integral literal either in decimal or octal or hexadecimal form but allowed values range is 0 to 65535.

Example:

1.	char ch=97;	‣valid	
2.	char ch=0xFace;	‣valid	
3.	char ch=65536;	‣Invalid	‣C.E: possible loss of precision

➢ We can represent a char literal by Unicode representation which is nothing but '\uxxxx'.

Example:

1. char ch1='\u0061';
   System.out.println(ch1);
   ‣Output: a
2. char ch2=\u0062; ‣Invalid ‣C.E:cannot find symbol
3. char ch3='\iface'; ‣Invalid ‣C.E:illegal escape character

➢ Every escape character in java acts as a char literal.

Example:

1. char ch='\n'; ‣valid
2. char ch='\l'; ‣Invalid ‣C.E:illegal escape character

Escape Character	Description
\n	Newline
\t	Horizontal tab
\r	Carriage return
\f	From feeb
\b	Backspace character
\’	Single Quote
\”	Double Quote
\\	Backspace

❖ **Which of the following char declarations are valid?**

1. char ch=a; ‣Invalid ‣C.E:cannot find symbol
2. char ch='ab'; ‣Invalid ‣C.E:unclosed character literal
3. char ch=65536; ‣Invalid ‣C.E:possible loss of precision

4.	char ch=\uface;	▸Invalid	▸C.E:illegal character: \64206
5.	char ch='/n';	▸Invalid	▸C.E:unclosed character literal
6.	none of the above.	▸Valid	

E. String literals:

- Any sequence of characters with in double quotes is treated as String literal.

  Example:

  String s="bhaskar"; ▸Valid

★ Diagram:

$$\begin{tikzcd}
\text{byte} \arrow[r] & \text{short} \arrow[dr] & & & & \\
& & \text{int} \arrow[r] & \text{long} \arrow[r] & \text{float} \arrow[r] & \text{double} \\
& \text{char} \arrow[ur] & & & &
\end{tikzcd}$$

1.5 Arrays

I. Introduction

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

➢ The main advantage of arrays is we can represent multiple values with the same name so that readability of the code will be improved.

➢ But the main disadvantage of arrays is: Fixed in size that is once we created an array there is no chance of increasing or decreasing the size based on our requirement that is to use arrays concept compulsory we should know the size in advance which may not possible always.

➢ We can resolve this problem by using collections.

II. Array declarations:

A. Single dimensional array declaration:

Example:

1. int[] a;

   ‣recommended to use because name is clearly separated from the type

2. int []a;
3. int a[];

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

Example:

1. int[] a; ‣valid
2. int[5] a; ‣invalid

B. Two dimensional array declaration:

Example:

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

All are valid.

C. Three dimensional array declaration:

Example:

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

All are valid.

★ **Which of the following declarations are valid?**

1. int[] a1,b1; ‣valid ‣a-1,b-1
2. int[] a2[],b2; ‣valid ‣a-2,b-1
3. int[] []a3,b3; ‣valid ‣a-2,b-2
4. int[] a,[]b; ‣Invalid ‣C.E:<identifier> expected

If we want to specify the dimension before the variable that rule is applicable only for the 1st variable. Second variable onwards we can't apply in the same declaration.

Example: int[][] a [] b; ‣Invalid

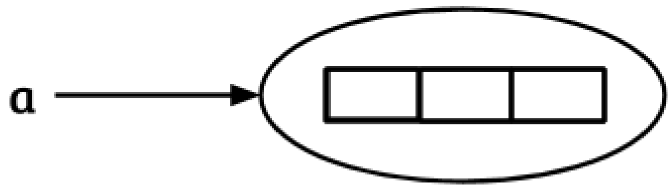
III. Array construction:

- Every array in java is an object hence we can create by using new operator.

Example:

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

Diagram:



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

Array Type	Corresponding Class Name
int[]	[I
int[][]	[[I
double[]	[D
...	...

- **Rules**

  1. At the time of array creation compulsory we should specify the size otherwise we will get compile time error.

     Example:

     ```
     int[] a=new int[3];
     int[] a=new int[];     ▸C.E:array dimension missing
     ```

  2. It is legal to have an array with size zero in java.

     Example:

int[] a=new int[0];

System.out.println(a.length); ▸Output:0

3. If we are taking array size with -ve int value then we will get runtime exception saying NegativeArraySizeException.

   Example:

   int[] a=new int[-3]; ▸R.E:NegativeArraySizeException

4. The allowed data types to specify array size are byte, short, char, int. By mistake if we are using any other type we will get compile time error.

   Example:

   1. int[] a=new int['a']; ▸Valid
      byte b=10;
   2. int[] a=new int[b]; ▸Valid
   3. short s=20;
      int[] a=new int[s]; ▸Valid
   4. int[] a=new int[10l]; ▸Invalid ▸C.E:possible loss of precision
   5. int[] a=new int[10.5]; ▸Invalid ▸C.E:possible loss of precision

5. The maximum allowed array size in java is maximum value of int size [2147483647].

   Example:

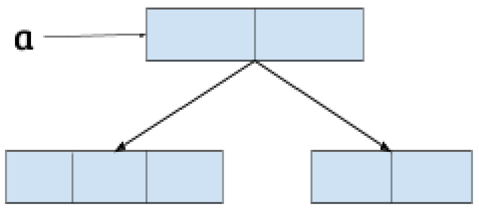
   1. int[] a1=new int[2147483647];
      ▸Valid
   2. int[] a2=new int[2147483648];
      ▸Invalid ▸C.E:integer number too large: 2147483648

Two dimensional array creation:

- In java multidimensional arrays are implemented as array of arrays approach but not matrix form.
- The main advantage of this approach is to improve memory utilization.

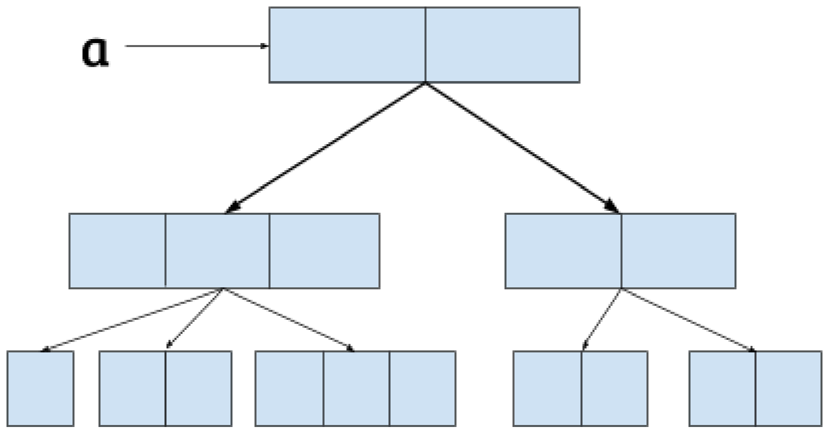
Example 1:

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



Example 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];
```



★ **Which of the following declarations are valid?**

1. int[] a=new int[] ▸Invalid ▸C.E: array dimension missing.
2. int[][] a=new int[3][4]; ▸Valid
3. int[][] a=new int[3][]; ▸Valid
4. int[][] a=new int[][4]; ▸Invalid ▸C.E: ']' expected
5. int[][][] a=new int[3][4][5]; ▸Valid
6. int[][][] a=new int[3][4][]; ▸Valid
7. int[][][] a=new int[3][][5]; ▸Invalid ▸C.E: ']' expected

IV. Array initialization:

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

Example 1:

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

System.out.println(a); ▸[I@3e25a5

System.out.println(a[0]); ▸0

★ **Note:** Whenever we are trying to print any object reference internally toString() method will be executed which is implemented by default to return the following.

classname@hexadecimalstringrepresentationofhashcode.

Example 2:

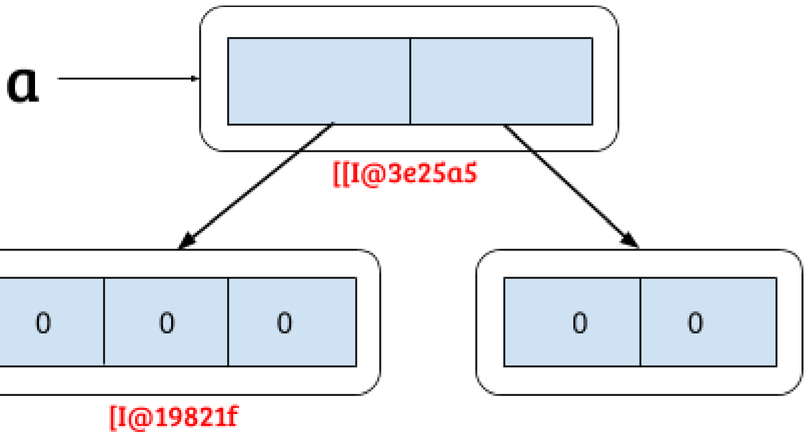
Base Size

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

System.out.println(a); ▸[I@3e25a5

System.out.println(a[0]); ▸[I@19821f

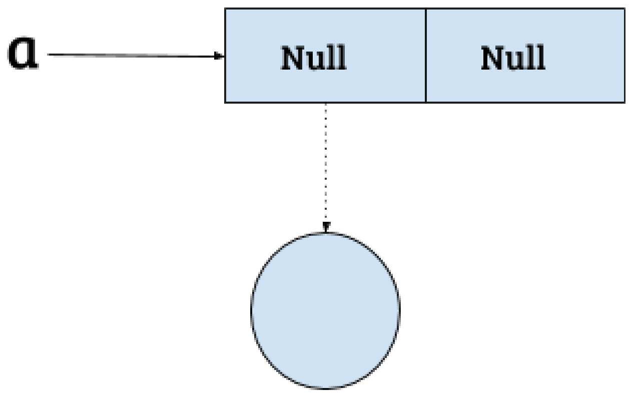
System.out.println(a[0][0]); ▸0



Example 3:

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

```
System.out.println(a);          ‣[[I@3e25a5
System.out.println(a[0]);       ‣null
System.out.println(a[0][0]);    ‣R.E:NullPointerException
```



- Once we created an array all its elements by default initialized with default values. If we are not satisfied with those default values then we can replays with our customized values.

Example:

```
int[] a=new int[4];
a[0]=10;
a[1]=20;
a[2]=30;
a[3]=40;
a[4]=50;    ‣R.E:ArrayIndexOutOfBoundsException: 4
a[-4]=60;   ‣R.E:ArrayIndexOutOfBoundsException: -4
```