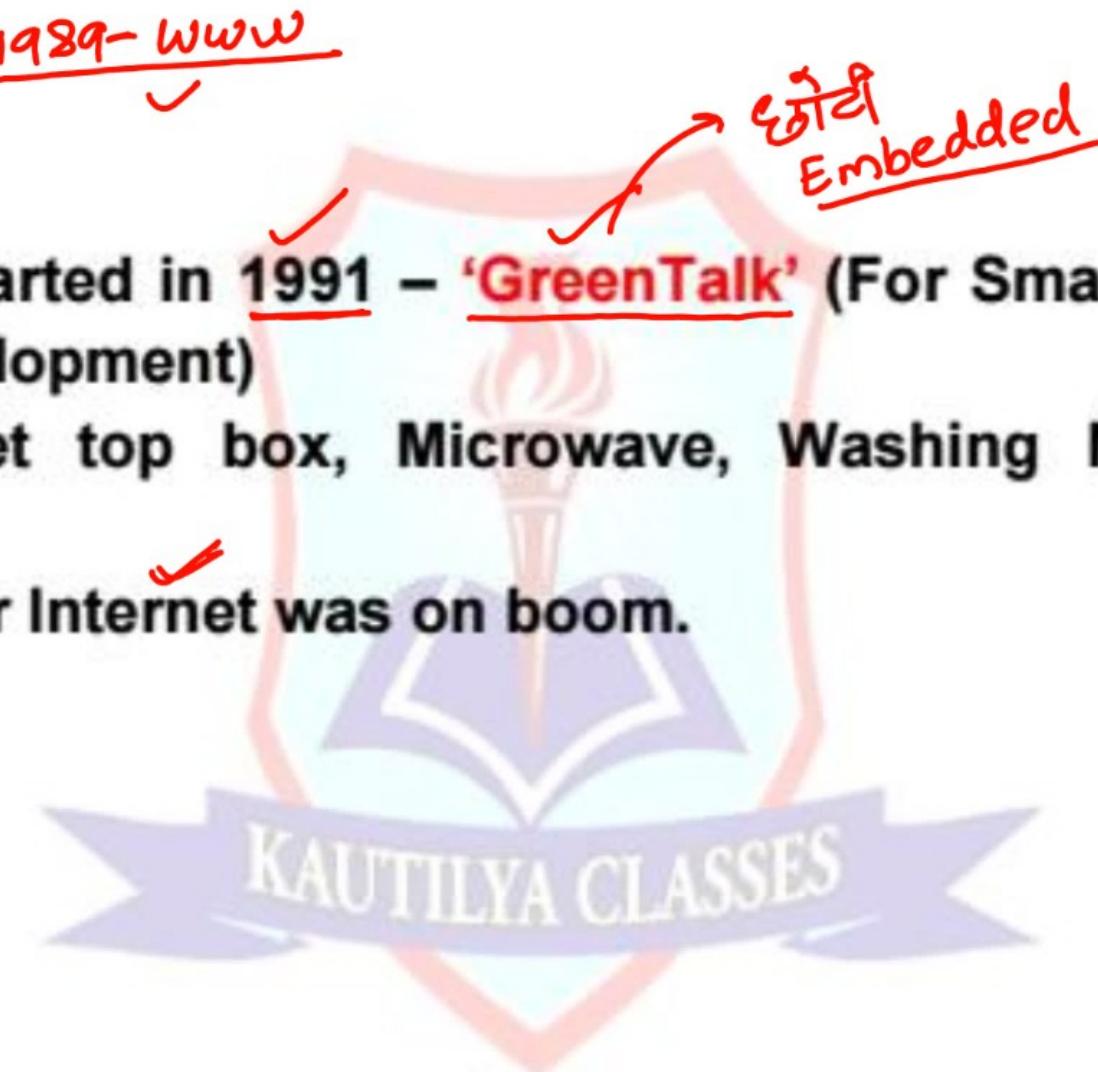


## JAVA

### History -

- ✓ Initially Started in 1991 - 'GreenTalk' (For Small Embedded System Development)
- ✓ Like - Set top box, Microwave, Washing Machine, TV Remote etc.
- ✓ In this year Internet was on boom.



Two requirements -

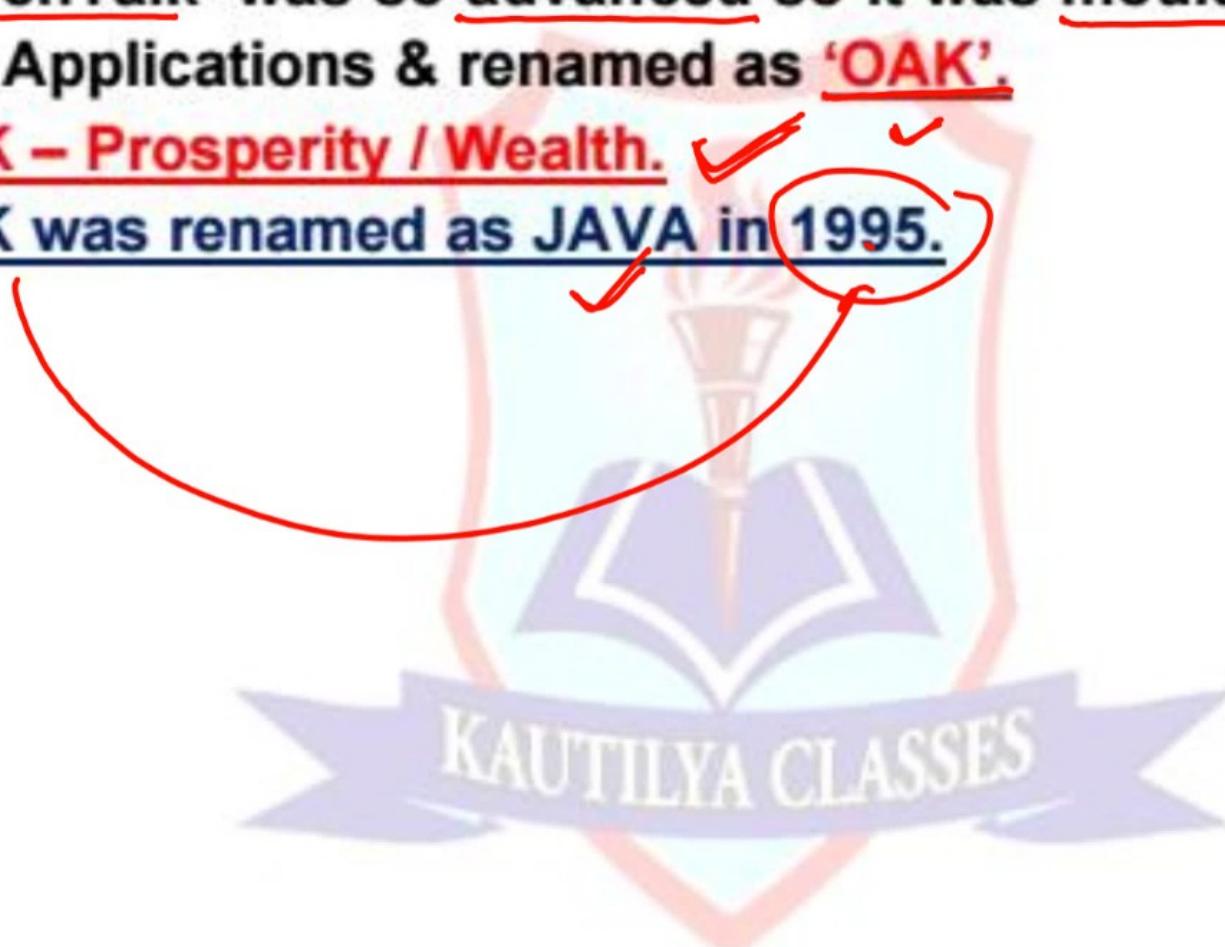
1. Platform Independent
2. Architecture Independent ✓

किसी भी Platform

HLS



- ✓ ‘GreenTalk’ was so advanced so it was moulded for Internet Based Applications & renamed as ‘OAK’.
- ✓ OAK – Prosperity / Wealth.
- ✓ OAK was renamed as JAVA in 1995.



## Green Team – Working for Green Project

- ✓ James Gosling (Team Leader & Father of Java)
- ✓ Mike Sheridan
- ✓ Patrik Naughton
- ✓ Ed Frank

❖ All these people were working for Sun Microsystem.

GreenTalk  
Green Project  
Green Team



- Java name was obtained by Coffee beans of an island located in Indonesia.



- JDK (Java Development Kit) Alpha & Beta versions of Java were launched in 1995.
- JDK 1.0 = 1996
- 
- 
- 
- JAVA SE (Standard Edition) 17 = 2021
- JAVA SE 18 = March 2022



- Sun Microsystem Occupied by - Oracle Corporation  
(January 27, 2010)



## ➤ Features of Java / Java Buzzwords / Java Whitepapers –

1. Simple ✓
2. Secure ✓
3. Portable ✓
4. Architectural Neutral ✓
5. Object Oriented ✓
6. Robust ✓
7. Interpreted ✓
8. High Performance ✓
9. Multithreaded ✓
10. Distributed ✓
11. Dynamic ✓



## **1. Simple –**

- **No Multiple & Hybrid Inheritance.** ✓
- **Virtual Base Class.** ✓
- **No Pointer supports.** ✓
- **No Friend functions.** ✓
- **No structure.** ✓
- **No inline function.** ✓
- **No virtual function.** ✓

**That's why Java supports clean syntax of OOP.**

## 2. Secure –

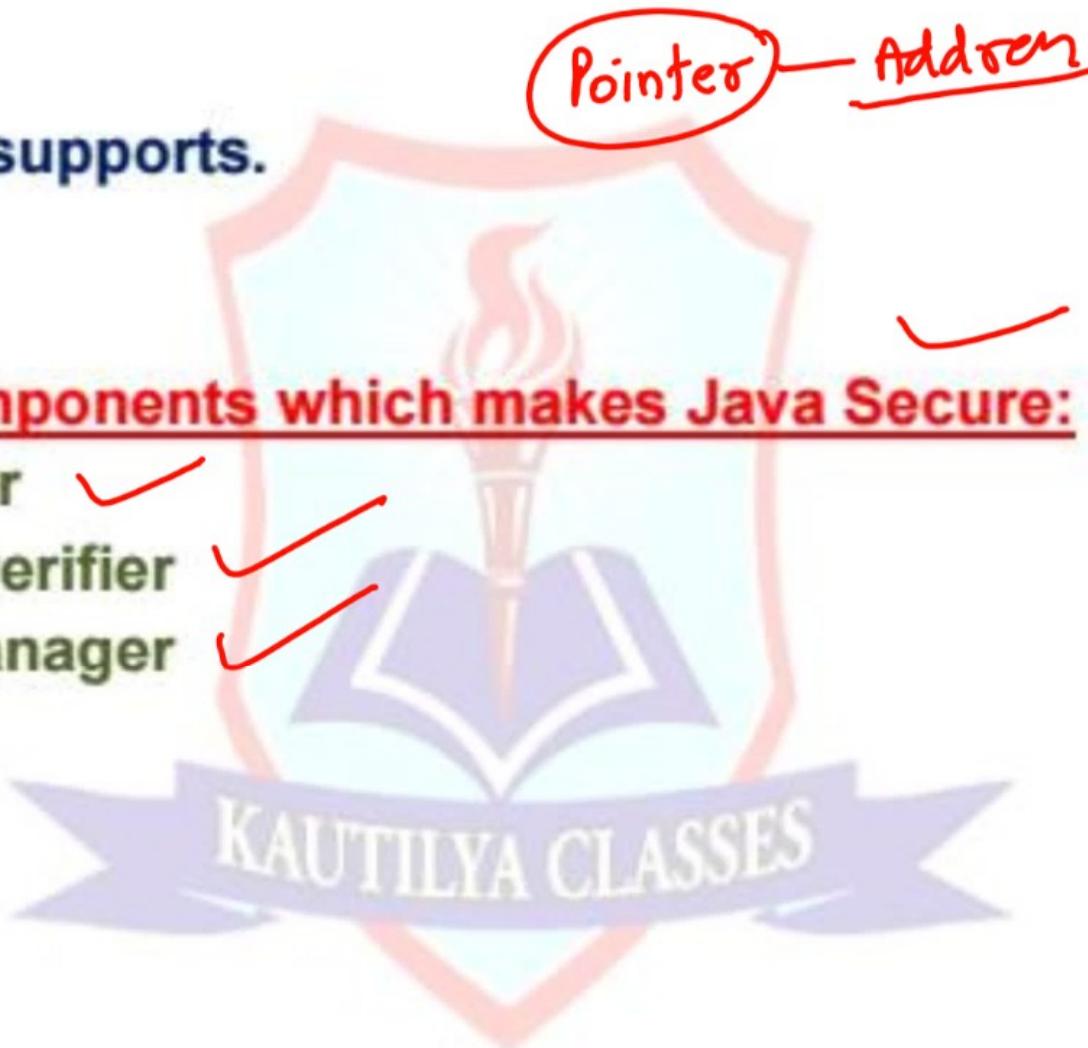
- No Pointer supports.

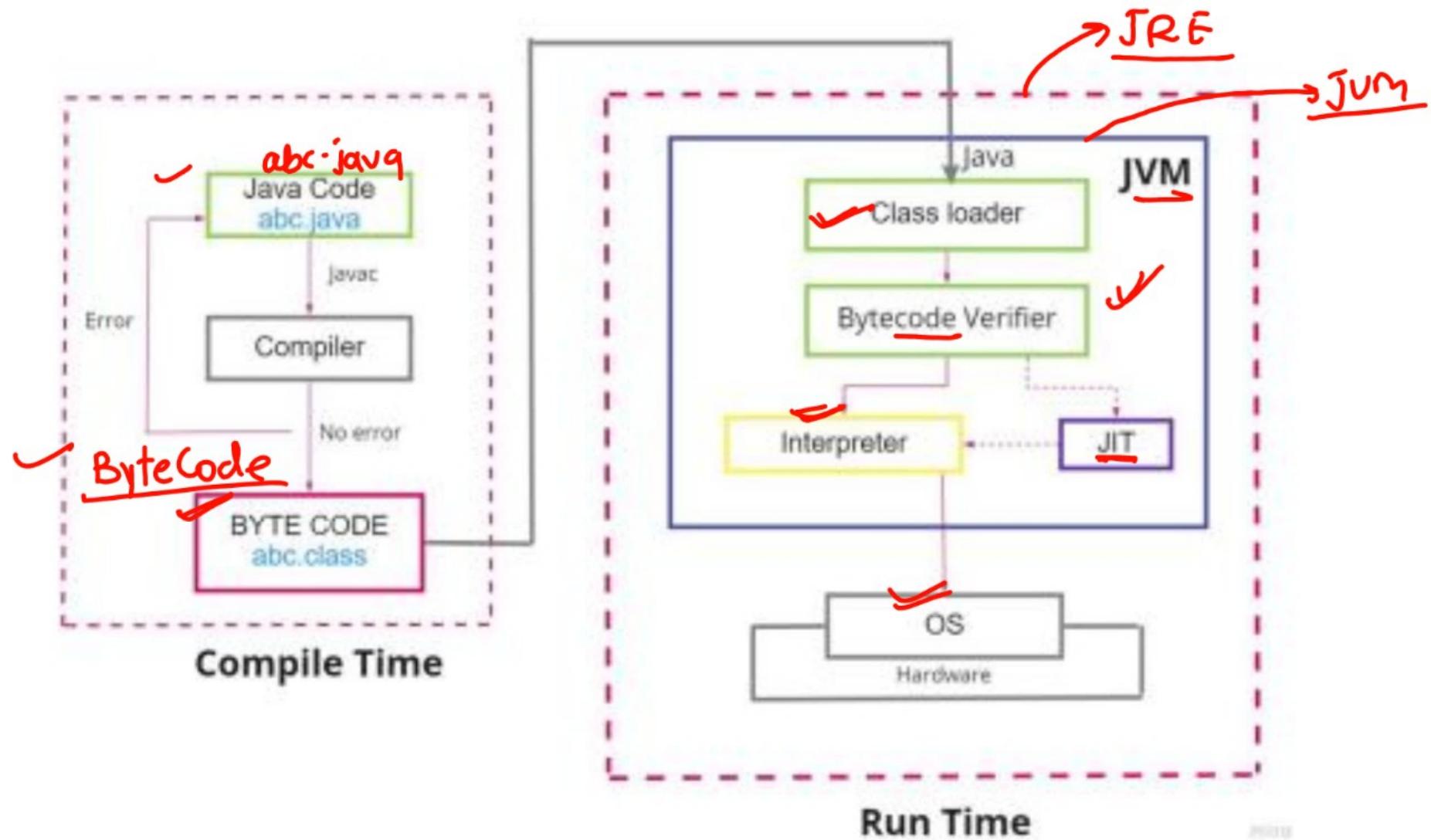
Pointer

Address

## Modules / Components which makes Java Secure:

- A. Class loader
- B. Byte code verifier
- C. Security manager





**A. Class loader - Class Loader JRE (Java Runtime Environment) का भाग है, जो Java की Classes को JVM (Java Virtual Machine) में Dynamically Load करता है।**



**B. Byte Code Verifier** - ये Module Memory में Loaded Java Code को Verify करता है, ताकि उस कोड के साथ कोई अन्य Invalid Code Attach नहीं हो।



**C. Security Manager** – Java Code के Run होने के दौरान **Security Manager** के द्वारा ये सुनिश्चित किया जाता है, कि उस Java Code को कौन **Access** कर रहा है एवं वह Java कोड जिन **Resources** को **Access** कर रहा है वह **Valid** है या नहीं।



3. Portable – Platform – OS ✓

4. Architectural Neutral – Hardware – Processor

★ WOUA - Write Once Use Anywhere



## 6. Robust / Reliable – Java doesn't show warnings.

Errors

- Java एक Robust Language है जिसका मतलब Reliable या Strong होता है।
- Java में Warnings को Error बनाकर पहले ही Solve कर लिया जाता है, इसलिए Java Programs के Run Time पर Fail / Crash होने कि संभावना न के बराबर रह जाती है।
- Java में Strong Memory Management दिया गया है।

- Java में Run Time पर Problems को Handle करने के लिए Exception Handling Mechanism दिया गया है।
- Java में Automatic Garbage Collection दिया गया है।

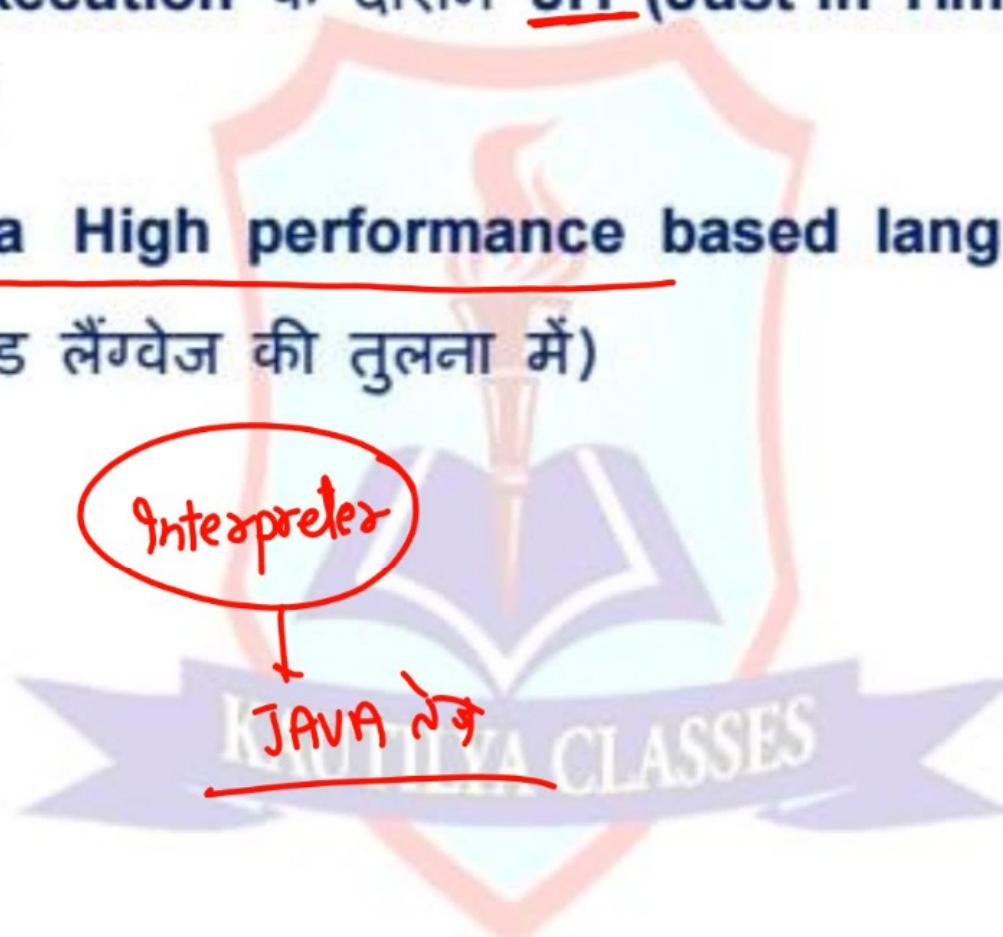


## 7. Interpreted -

javac ~~Byte~~ Compiler

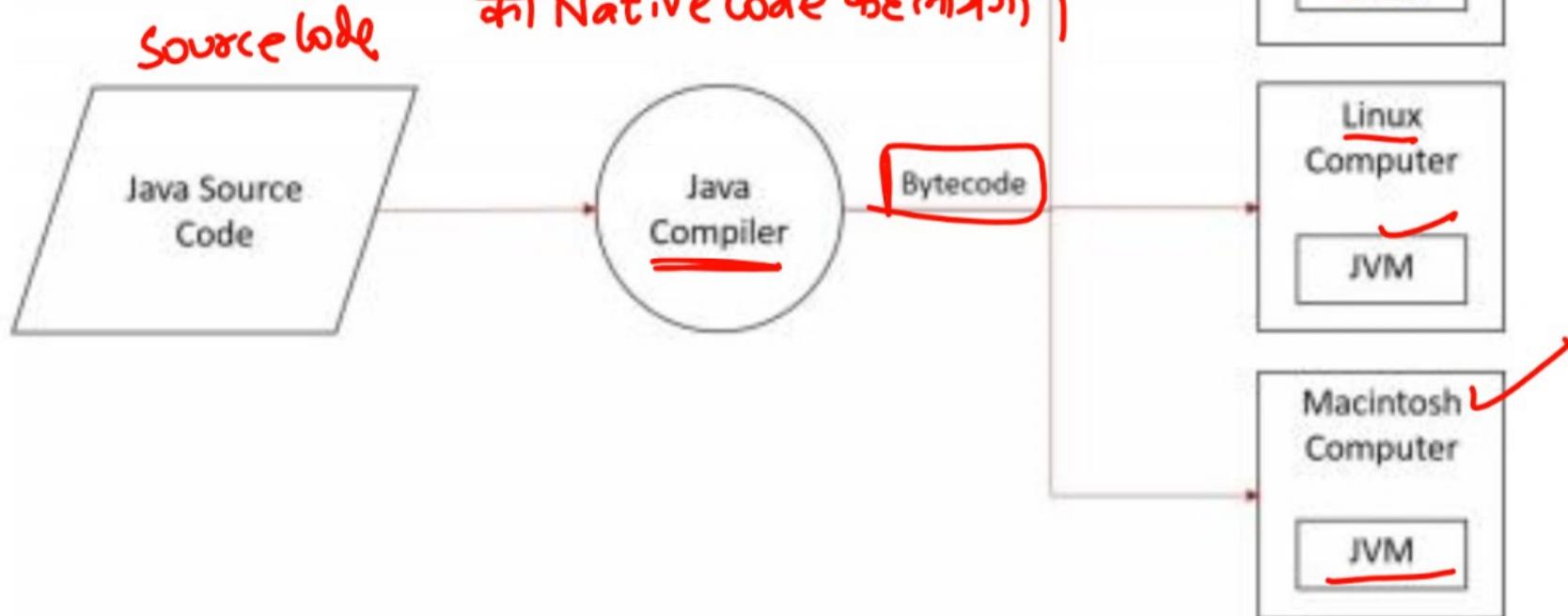
- Java Interpreter Based Language है, ताकि इसमें Portable और Architecture Neutral Feature को जोड़ा जा सके।
- Interpreter Based होने की वजह से Java को Execute करने का समय Compiler Based Languages की तुलना में अधिक लगता है।
- Java के Executive Time को बेहतर करने के लिए Java ने निम्न तरीकों को अपनाया -
- ✓ Intermediate Code से Execute करना जिसे Byte Code कहते हैं।

- ✓ प्रोग्राम के Execution के दौरान JIT (Just In Time Compiler) को इस्तेमाल करना।
- ✓ इसलिए Java High performance based language है। (अन्य Interpreter बेस्ड लैंग्वेज की तुलना में)



## 8. High Performance -

Java Source Code को Byte code में Convert कर दिया जाता है तथा इस Byte Code को जिसी Machine पर run किया जाएगा, ग्राउंटी Machine का Native Code कहलाएगा।

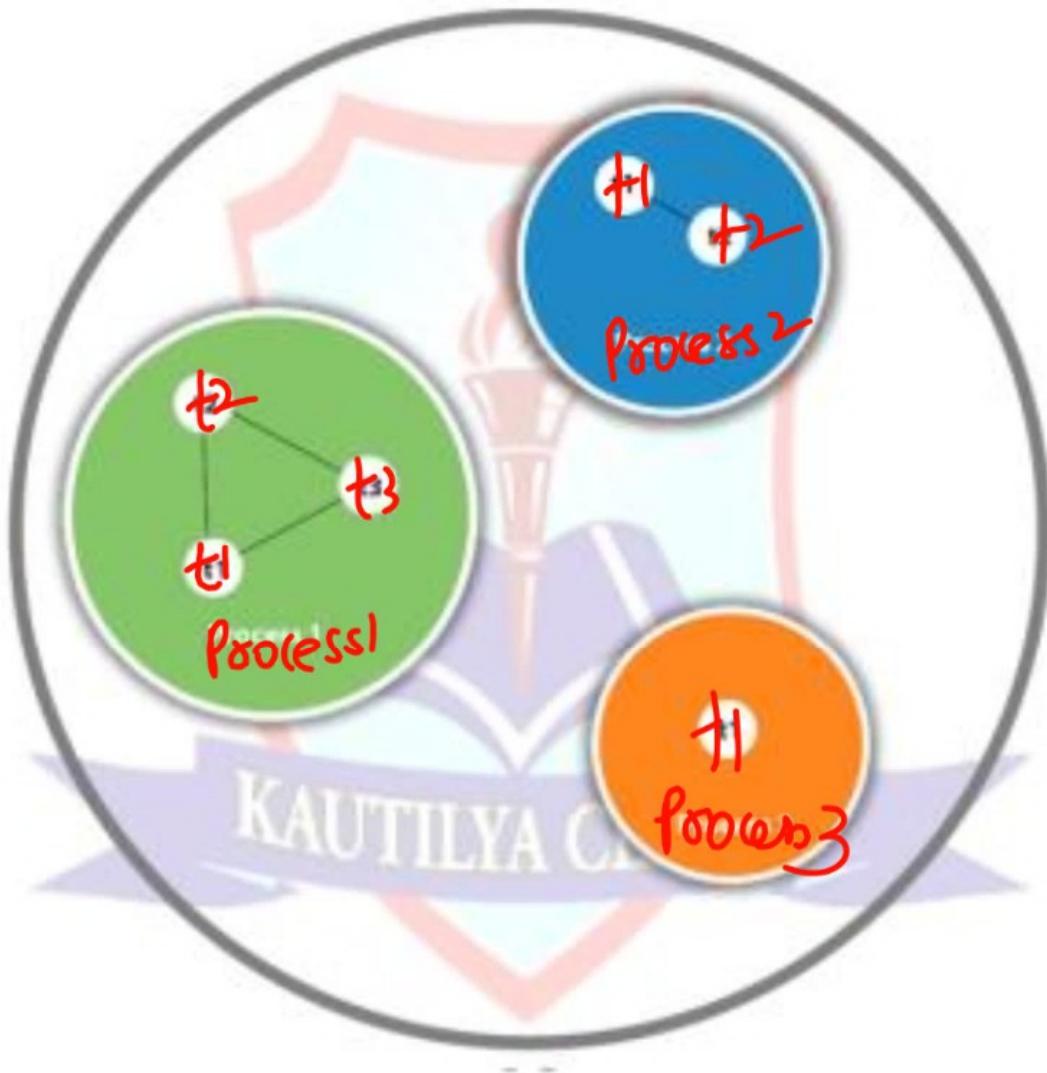


## **9. Multithreaded –**



- JVM (Java Virtual Machine) plays the role of OS in Java.
- Java में **Multiple Threads** बनाने की सुविधा मौजूद है।
- Thread, Process का भाग होता है, जिसके द्वारा एक से अधिक कार्यों को Concurrently Execute किया जा सकता है।
- Thread को Light Weight Process भी कहा जाता है।
- Thread के द्वारा उन Resources को Share किया जाता है, जो उसके Process को Allocate होते हैं।

KAUTILYA CLASSES



## 10. Distributed –

➤ Java में Distributed Applications बनाने के लिए API (Application Program Interface) दिया गया है –

- ✓ RMI - Remote Method Invocation
- ✓ EJB - Enterprise Java Bean



## 11. Dynamic –

- Java एक Dynamic Language है, जिसमें Classes को Run Time पर Load करने की सुविधा मौजूद है।
- Java में Classes को On Demand Memory में Load किया जा सकता है।
- Java में Run Time पर किसी Object के माध्यम से उसकी Class की Details को ज्ञात किया जा सकता है।
- Java Dynamic Compilation Support करता है।

## Java Environment –

- JVM – Java Virtual Machine
- JRE – Java Runtime Environment
- JDK – Java Development Kit

JRE = JVM + Set of Libraries + Other Files

✓ JDK > JRE > JVM



KAUTILYA CLASSES



## Java Development Tools -

- Javac – Java Compiler ✓
- Javadoc – Java Documentation ✓
- Jdb – Java Debugger ✓
- Javah – Java Header File ✓
- Javap – Java Parser (Read from bytecode) ✓
- Appletviewer - Java के Environment में Applets को Run करना ✓
- JVM – Java Interpreter ✓
- Jar – Java archive files ✓

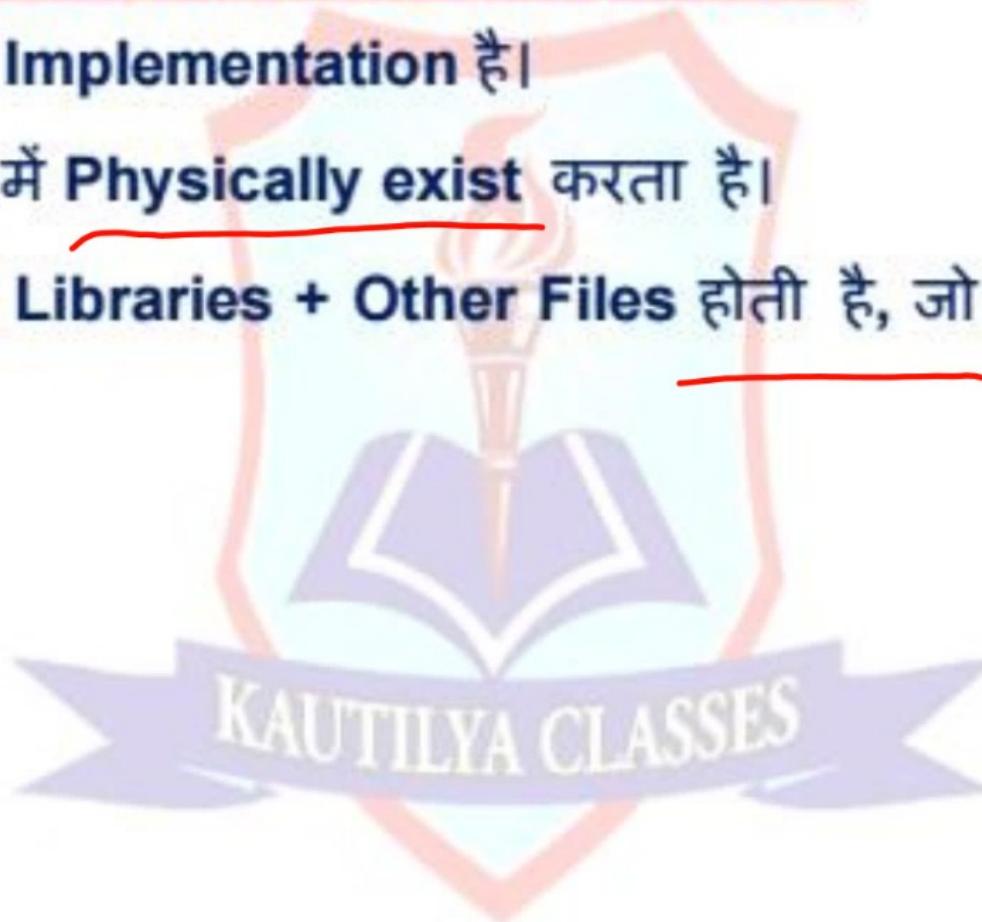
## JVM – Java Virtual Machine –

- Java Programs के लिए एक Virtual Operating System है।
- यह Physically exist नहीं करता है।
- यह Set of instructions है, जो Java के Bytecode को run करने के लिए Runtime Environment प्रदान करता है।



## JRE / JRTE – Java Run Time Environment –

- यह JVM का Implementation है।
- यह System में Physically exist करता है।
- इसमें Set of Libraries + Other Files होती है, जो JVM Use करता है।



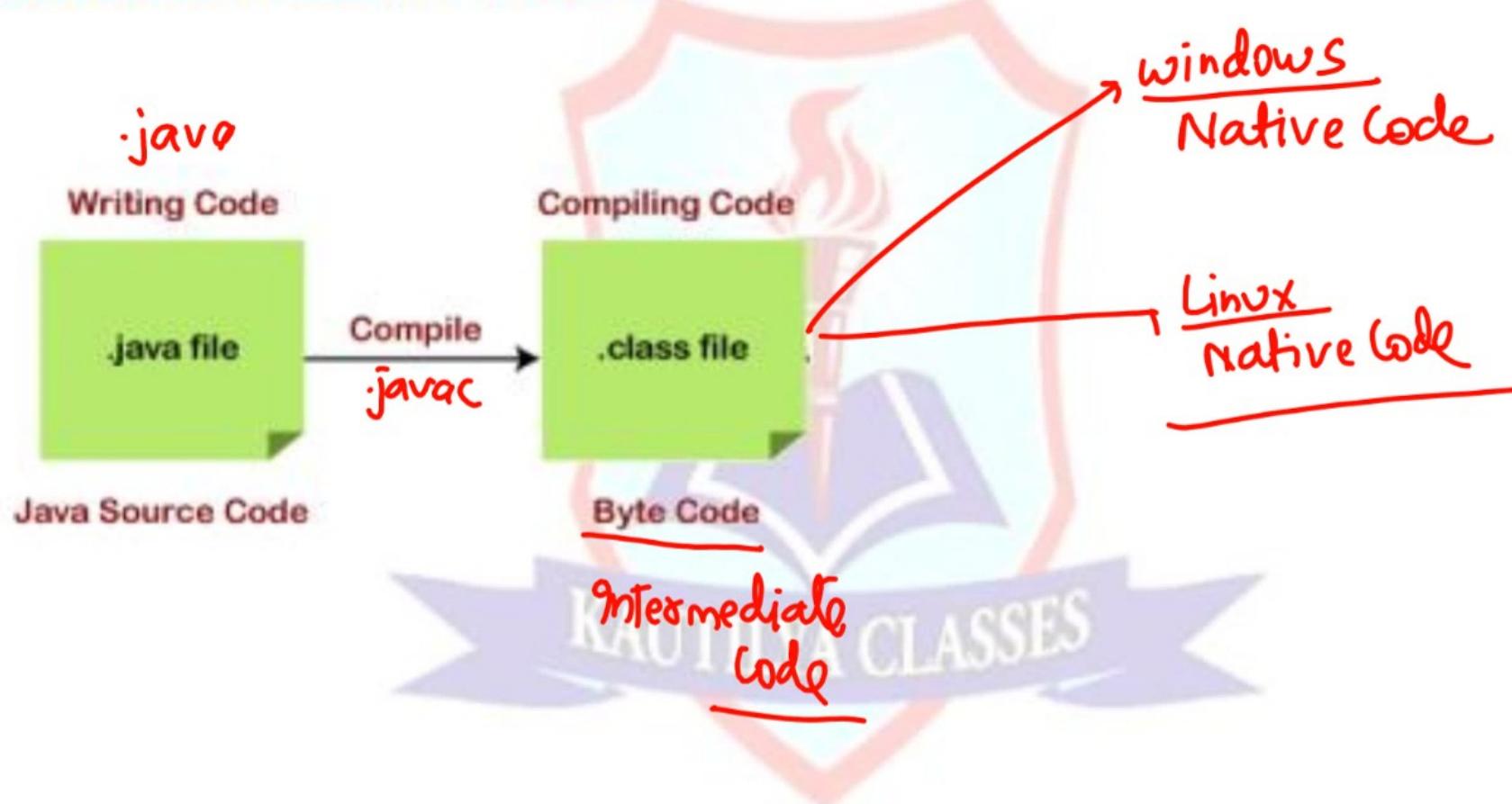
## JDK – Java Development Kit –

- Java Applications को बनाने के लिए Use किया जाता है।
- JDK में JRE + Java applications बनाने के लिए Use किये जाने वाले Tools का Collection होता है।
- Java Application –
  - ✓ J2SE – Java 2 Standard Edition
  - ✓ J2EE – Java 2 Enterprise Edition
  - ✓ J2ME – Java 2 Micro Edition



KAUTILYA CLASSES

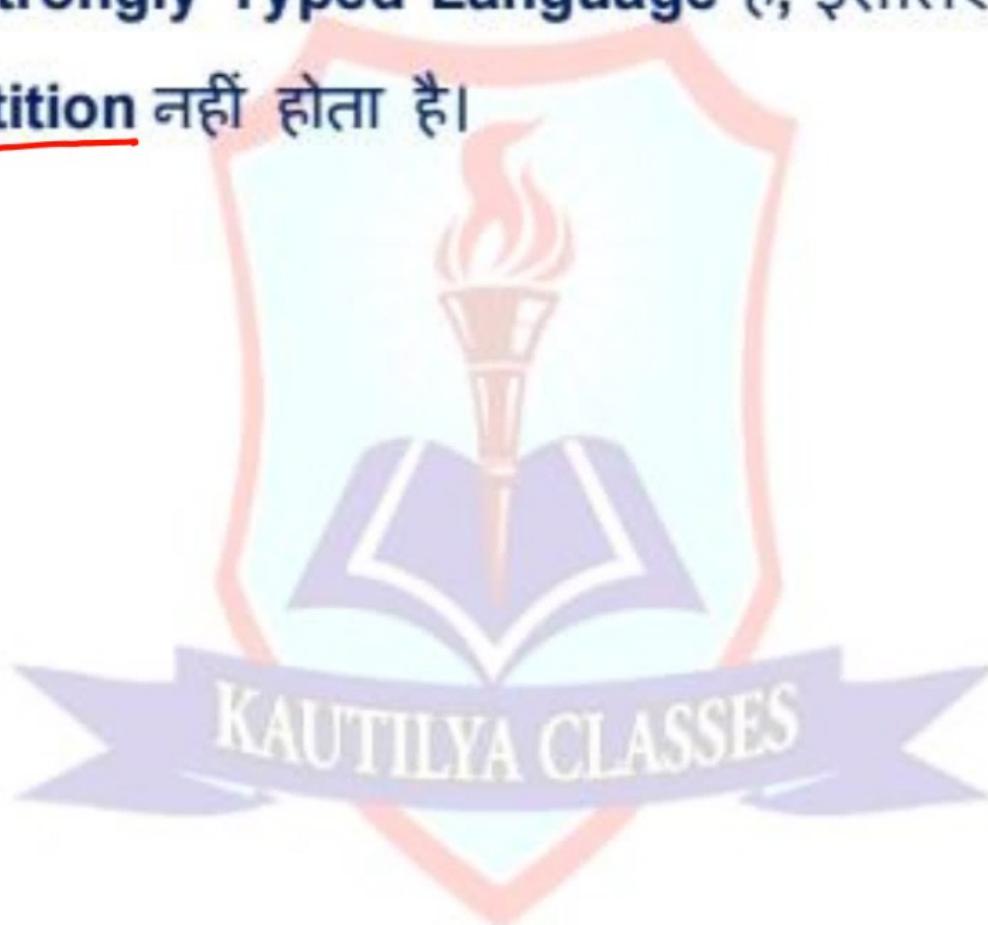
## Java Translation Policy –

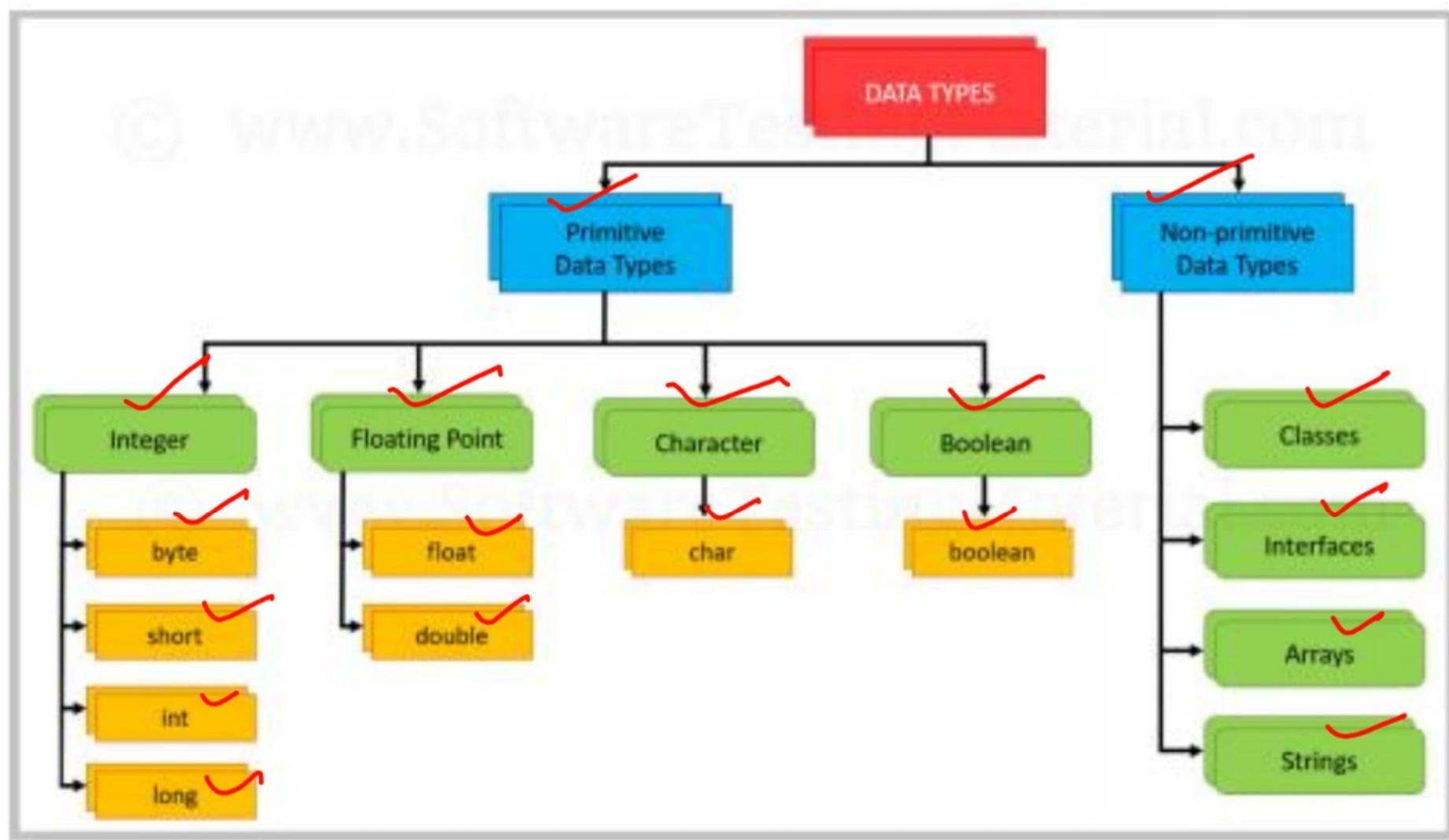


- Byte Code एक Intermediate Code है, जिसे Java Code को Java के Compiler 'javac' के द्वारा परिवर्तित करके प्राप्त किया जाता है।
- Byte Code को JVM के द्वारा interpret करके किसी **Machine** का Native Code प्राप्त किया जाता है।



- Java एक Strongly Typed Language है, इसलिए किसी भी Data Type का Repetition नहीं होता है।





	<b>Size</b>
byte	✓ 1 byte
short	✓ 2 byte
int	✓ 4 byte
long	✓ 8 byte
char	✓ 2 byte
float	✓ 4 byte
double	✓ 8 byte
boolean	✓ 1 bit (till JDK 1.3 it uses 1 byte)



**Q – Java में Char Data Type 2 बाइट का क्यों होता है ?**



- Java में Globalization का Concept support करती है।
- Java में Character Data Type 2 Bytes लेते हैं, क्योंकि Java UNICODE Encoding Technique का use करता है।
- UNICODE Encoding Technique में दुनिया में बोली जाने वाली सभी Human Languages के Character set को Unique Number प्रदान किया गया, जिसे UNICODE (Universal Character Encoding) कहा जाता है।



KAUTILYA CLASSES

**Wrapper class –**



- The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

Primitive Data Types	Wrapper Classes
int ✓	Integer
float	Float
double	Double
boolean	Boolean (Added from 1.5)
short	Short
byte	Byte
char	Character
long	Long

```
Int a, b, c;  
a = 10, b = 5;  
c = a < b;  
boolean
```

```
Int a, b;  
a = 10, b = 5;  
boolean c;  
c = a < b;  
False
```



```
while ()  
{  
...  
...  
}
```

```
while (true)  
{  
...  
...  
}
```

```
while (1)  
{  
...  
...  
}
```



C,C++ = 31

### Variable Rules -

1. Length rule has been ruled out. 
2. Underscore & Letter can be used to start a variable name.



```
class demo
{
    public static void main (string args [ ])
    {
        int i, j;
        for (i=1; i<=5; i++)
        {
            for (j=1; j<=i; j++)
                system.out.print (j + " ");
            system.out.println ( );
        }
    }
}
```



KAUTILYA CLASSES

```
> javac first.java
```

```
demo.class
```

```
➤      java demo
```



Makes it class method so that it can be called using class name without creating an object of the class.

public

static

void

To call by JVM from anywhere

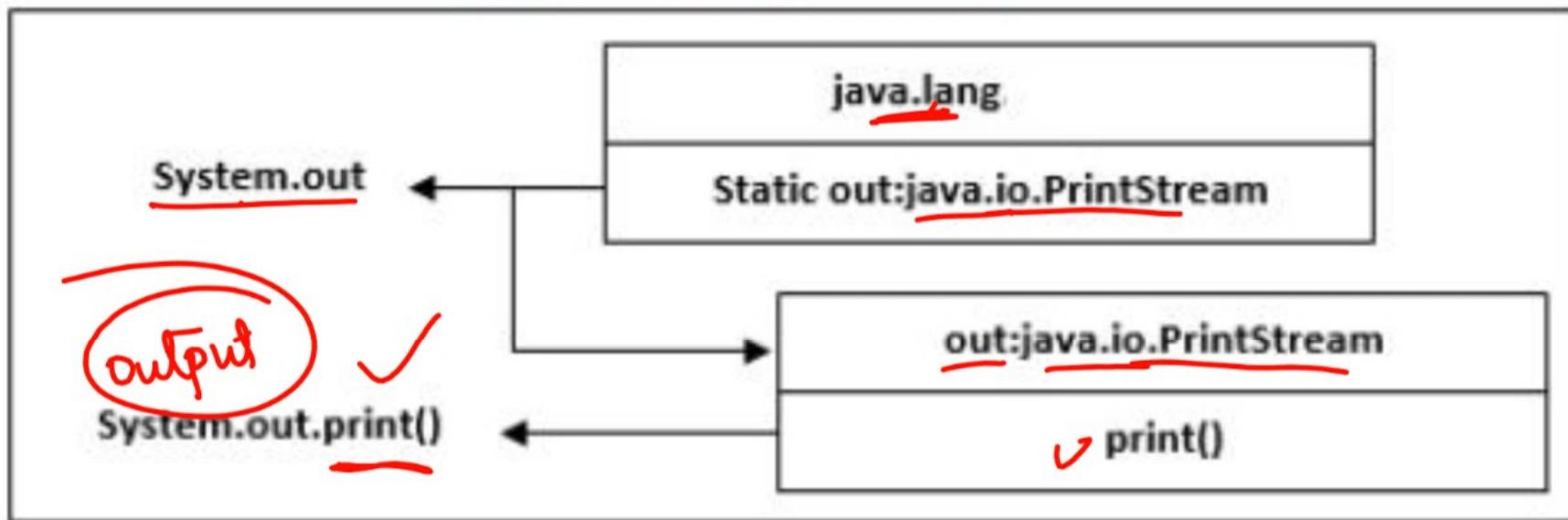
Name of the method which is called by JVM.

main(String[ ] args)

main method does not return value to JVM.

The main() method accepts one argument of type String array.

# system.out.println()



## Operators in Java –

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Conditional Operators
6. Bitwise Operators



## **1. Arithmetic Operators**

- A. sizeof() operator**
- B. addressof (&) and value at (\*) operator**
- C. Increment in capacity of %**
- D. delete**
- E. endl**



### 3. Logical Operators

- A. Logical AND (&)
- B. Logical OR (||)
- C. Logical NOT (!)
- D. Logical Short Circuit AND (&&)
- E. Logical Short Circuit OR (||)



## 7. Bitwise Operators

- A. Bitwise AND (&)
- B. Bitwise OR (|)
- C. Bitwise NOT / Complement (~)
- D. Bitwise ExOR / Exclusive OR (^)
- E. Bitwise Left Shift (>>)
- F. Bitwise Right Shift (>>)
- G. Bitwise Unsigned Right Shift (>>>)



KAUTILYA CLASSES

int a=10; a>>>2

int a = 10; a >>> 2

(10) ~~10~~  
X  
② ✓

Number	128	64	32	16	8	4	2	1
Binary	0	0	0	0	1	0	1	0
a >>> 2	0	0	0	0	1	0	X	X

← 10  
Delete first 2 number

✓

Number	128	64	32	16	8	4	2	1
Binary	0	0	0	0	0	0	1	0

2



## Control Statements

Jump statements enhanced –

break

break <label>

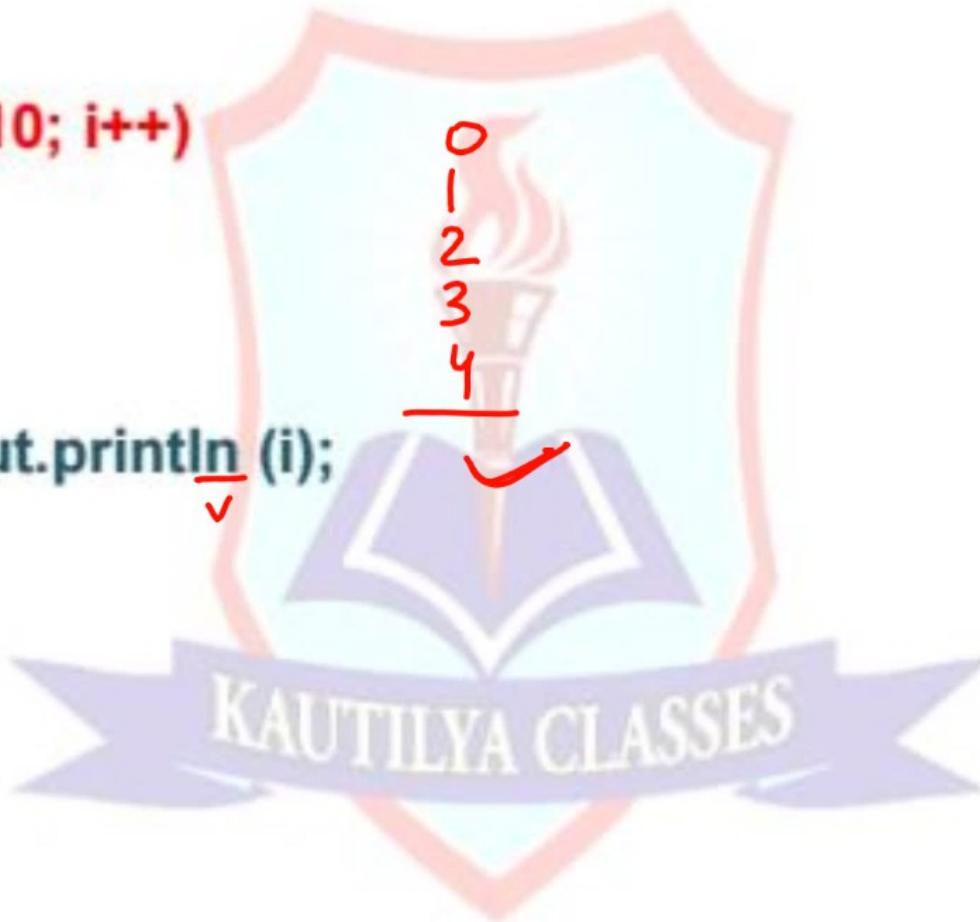
continue

continue <label>



### Ex - 1.

```
int i;  
for (i=0; i<=10; i++)  
{  
    If (i==5)  
        break;  
    system.out.println (i);
```



### Ex - 2.

```
int i, j;  
for (i=1; i<=3; i++)  
{  
    for (j=1; j<=5; j++)  
    {  
        if (i==2)  
            break;  
        System.out.println (j);  
    }  
}
```

$i = 1-3$   
 $j = 1-5$

1 2 3 4 5

KAUTILYA CLASSES

### Ex - 3.

✓ int i, j;

outer:

for ( $i=1$ ;  $i \leq 3$ ;  $i++$ )

{

inner:

for ( $j=1$ ;  $j \leq 5$ ;  $j++$ )

{

if ( $i==2$ )

break outer;

system.out.println (j);

}

$i=1-3$   
 $j=1-5$

1 2 3 4 5  
1 2 3 4 5

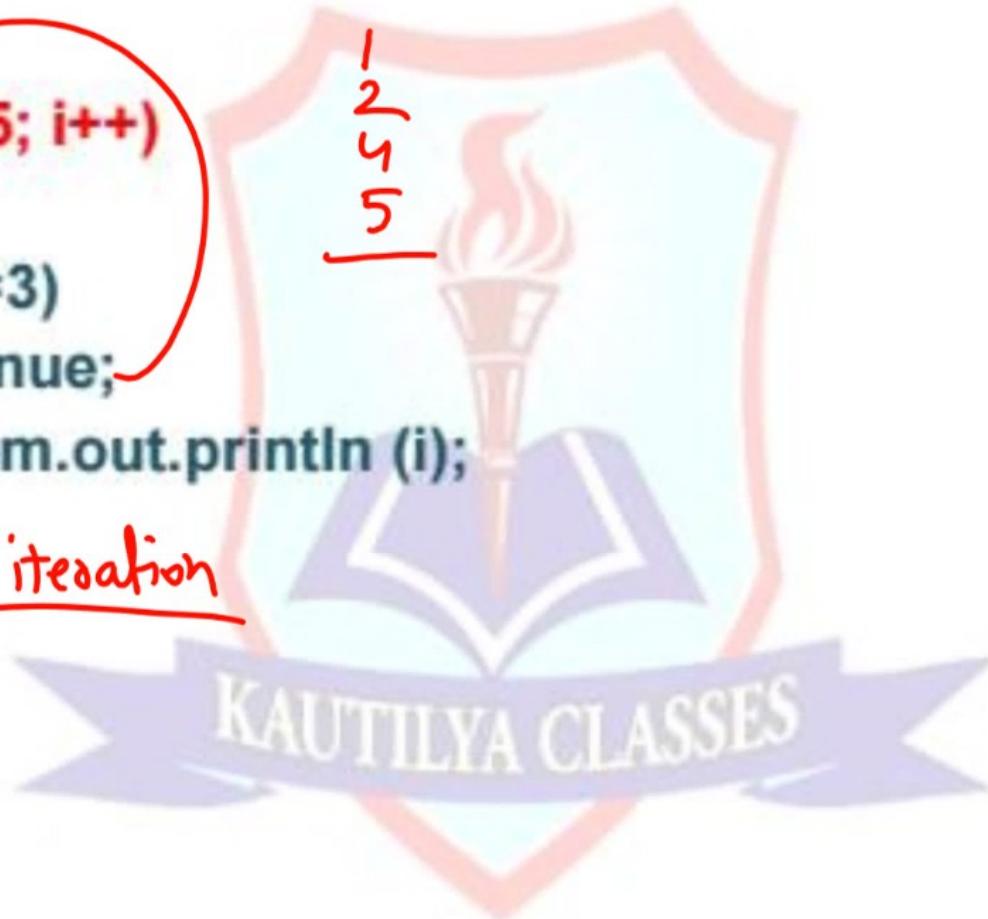
}



#### Ex - 4.

```
i = 1-5  
int i;  
for (i=1; i<=5; i++)  
{  
    if (i==3)  
        continue;  
    system.out.println (i);  
}  
  
next iteration
```

1  
2  
3  
4  
5



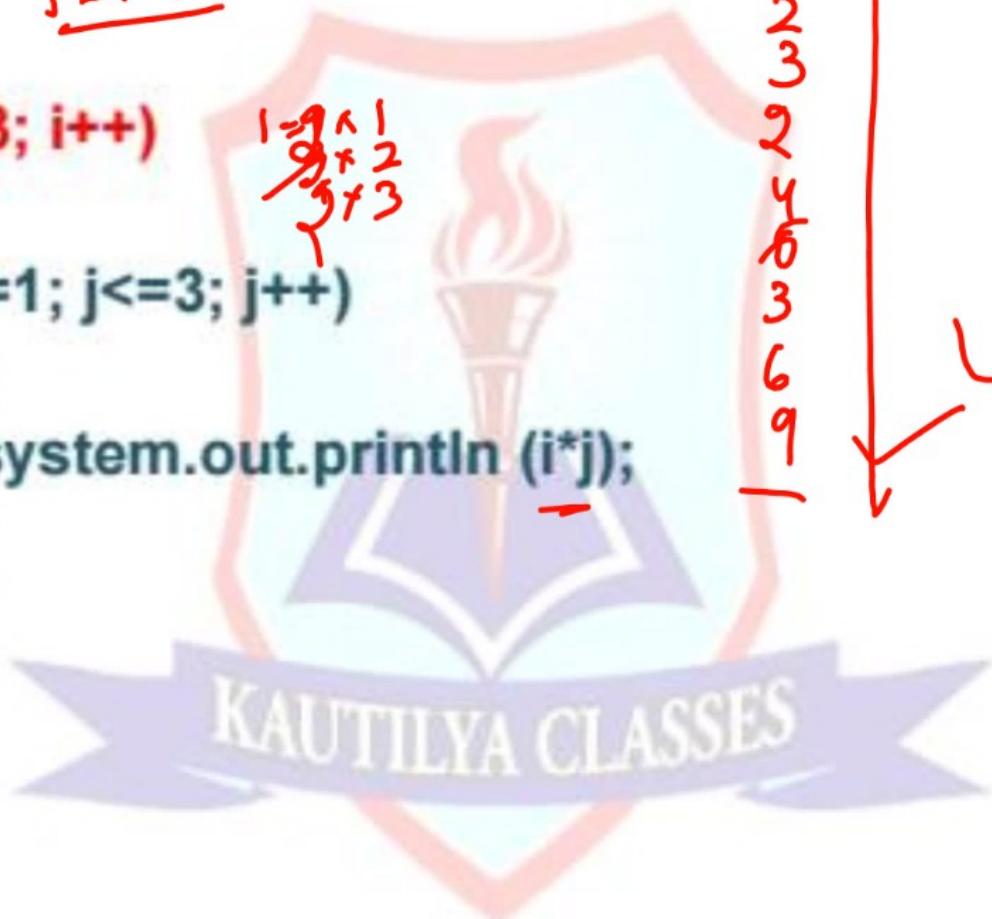
### Ex - 5.

```
int i, j;  
for (i=1; i<=3; i++)  
{  
    for (j=1; j<=3; j++)  
    {  
        system.out.println (i*j);  
    }  
}
```

$$\begin{array}{l} i=1-3 \\ j=1-3 \end{array}$$

$$\begin{array}{l} 1-9 \\ 9+1 \\ 9+2 \\ 9+3 \end{array}$$

1  
2  
3  
2  
4  
5  
3  
6  
9



### Ex - 6.

```
int i, j;  
outer:   ←  
for (i=1; i<=3; i++)
```

```
{     inner:  
      for (j=1; j<=3; j++)  
      {  
        if (i*j == 3)  
          continue outer;  
        system.out.println (i*j);  
      }  
}
```

$$\begin{array}{l} i=1-3 \\ j=1-3 \end{array}$$

$$\begin{array}{l} i=1 \times 1 = 1 \\ 1 \times 2 = 2 \\ 1 \times 3 = 3 \end{array}$$

$$\begin{array}{ll} i=2 \times 1 & 2 \\ \times 2 & 4 \\ \times 3 & 6 \end{array}$$

~~$\begin{array}{ll} 3 \times 1 & 3 \\ 2 & \\ & \end{array}$~~

1  
2  
2  
4  
6

O/P

## Arrays in Java

- Reference type.
- Traditional method is not used.



Dynamic  
Memory  
Allocation

`int arr [ ];  
arr = new int [10];`

OR

`int arr [ ] = new int [10];`

OR

`int arr [ ] = {10, 12, 13, 40, 50};`

~~int arr [5];~~



## **\*\* Implicit Property:**

### **length -**

- जावा में प्रत्येक ऐरे के साथ एक implicit property 'length' लैंथ जोड़ दी जाती है, जो ऐरे में कुल **elements** की संख्या प्रदर्शित करती है।
- यह प्रॉपर्टी automatically मैनेज होती है

s.o.p("Number of element in arr:" + arr.length)



```
int n = 6;  
char [ ] carr;  
caar = new char [n];6
```

OR

```
char [ ] carr = new char [n];
```

OR

```
char [ ] carr = {'K', 'A', 'U', 'T', 'I', 'L', 'Y', 'A'};
```

OR

```
char [ ] carr = {"KAUTILYA"};
```



## ★ Array ⇒

- Reference प्रकार का DataType.
- Traditional method is not used.

int arr[5]; X

- Java में Array को declare करने के बाद new keyword के द्वारा object assign कर execute किया जाता है।

Ex ⇒      int arr[];  
                arr[] = new int[10];      }      arr= 10 Elements

✓ new ⇒ size/memory allocation  
of Array

int arr[] = new int[10];

int arr[] = {5, 6, 4, 2, 1};

## \* Length property of Array =>

→ Java के प्रत्येक Array में एक implicit property "Length" का use किया जाना है, जो Array के Total elements की संख्या को display करता है

Ex :-

```
int arr[] = {5, 4, 3, 9, 15};  
for (int i=0; i<arr.length; i++)  
{  
    System.out.println(arr[i]);  
}
```

O/P: 5 4 3 9 15

Q = length property में i की first value by default क्या होगी ?  
A = 0

System.out.print(arr[i] + " ");

↓

5\_4\_3\_9\_15

1 space will be added

\* foreach loop in Java ⇒

for (data-type variable-name : Array-name)

जह Collection पर्हने हे declare होना जरी है,  
नहीं तो Error आनेगी।

⇒ यह Array का First & Last Element लेता है।

To be Cont.....

\* foreach loop in Java ⇒

for (data-type variable-name : Array-name)  
  ↓

ए Collection पहले से declare होना जरूरी है,  
नहीं तो Error आयेगी।

⇒ यह array का First & Last Element लेता है।

To be Cont.....

Ex⇒

```
int arr[] = {1, 9, 4, 5, 3, 7};
```

```
for (int i : arr)  
    System.out.print(arr[i]);
```

O/P= 194537

## \* 2'D Array in Java ⇒

int arr [ ] [ ] = new int [3] [3];  
                    ✓

[OR]

int arr [ ] [ ];  
arr = new int [3] [3];

[OR]

int arr [ ] [ ] = { {1,2,3}, {4,5,6}, {7,8,9} };

Ex :-

```
int arr[][] = {{1,2,3},{4,5,6},{7,8,9}};
for(int i=0; i< arr.length ; i++)
{
    for(int j=0; j< arr[i].length ; j++)
    {
        System.out.print (arr[i][j] + " ");
    }
}
```

O/P = 1 2 3 4 5 6 7 8 9

## \* Jagged Array ⇒

```
int jagged[][];  
jagged = new int[3][]; // 3 Rows वाली Array
```

```
jagged[0] = new int [4];  
jagged[1] = new int [7];  
jagged[2] = new int [5];
```

	0	1	2	3	4	5	6
0							
1							
2							

- Java में एक Specific Type का Array support किया जाता है, जिसे Jagged Array कहते हैं।
- Jagged Array की प्रत्येक Row में Columns की संख्या छलवा  $\neq$  define की जा सकती है।

## \* Class in Java ⇒

```
class SS
{
    int x; // default
    public int a;
    public void setA(int n)
    {
        x=n;
        a=n;
    }
    public void display()
    {
```

```
    System.out.print("a=" + a);
    System.out.print("x=" + x);
}
class demo
{
    public static void main (String args[])
    {
        SS obj1; // Reference
        obj1 = new SS(); // Object
        obj1.setA(5);
        obj1.display();
    }
}
```

O/P ⇒  
a=5   x=5

## \* Access Modifiers in Java ⇒

- 4 ⇒ A. public  
B. private  
C. protected  
D. default
- Java में class की property by default रूप से "default" होती है।
- Java की class में प्रत्येक variable व function का अलग-<sup>2</sup> Access Modifier बताता पड़ता है, जिसके प्रत्येक variable व function को अलग-<sup>2</sup> प्रकार की functionality से होकर बदलना पड़ता है।
- class का object declare करते के लिए new keyword का प्रयोग किया जाता है।
- महि class के नाम के साथ किसी variable को लिख दिया जाते हो उसे class प्रकार का Reference Variable कहा जाता है।

Ex ⇒

```
SS obj1; // Reference  
obj1 = new SS(); // object
```

Q ⇒ Java में object बनाने के लिए किस keyword का use किया जाता है?

- A. new ✓
  - B. clone
  - C. newInstance
  - D. deserialization
  - E. factory method
- ✓ सभी सही हैं।

## \* Constructor in Java ⇒

```
class SS  
{ int a;  
SS() // Constructor  
{ System.out.print("DAC");  
}  
SS(int n)  
{ a=n;  
System.out.print(" PC=" + a);  
}
```

```
ss(ss obj)  
{ System.out.print("CC");  
}  
class demo  
{ public static void main(String args[]){  
SS obj2,obj3,obj4; // Reference  
obj2 = new SS();  
obj3 = new SS(5);  
obj4 = new SS(obj3);  
}
```

O/P =  
DAC PC=5 CC

\* Java में unused memory की Reclaim करने के लिए **garbage collector** का use किया जाता है।

**garbage collector** का use



एक Utility है, जिसे  
JVM Control करती है।

\* Java में destructor की जगह finalize() method को override करके use किया जाता है।

```
protected void finalize()
{
}
```



## \* Inheritance in Java =>

1. Single level

2. Multi level

3. Hierarchical Level

No Multiple & Hybrid Inheritance

→ Java में inheritance के लिए extends keyword का use किया जाता है।  
→ Inheritance Parent-child Relationship (Is-A) को Represent करती है।

Ex ⇒

```
class SS  
{  
    int a;  
}
```

```
class KC extends SS  
{  
    int b;  
    void setB(int n)  
    {  
        a=n;  
        b=n;  
    }  
}
```

```
void display()  
{  
    System.out.print ("a=" + a);  
    System.out.print ("b=" + b);  
}
```

```
class demo  
{  
    public static void main (String args[])  
    {  
        KC obj1;  
        obj1 = new KC();  
        obj1.setB(10);  
        obj1.display();  
    }  
}
```

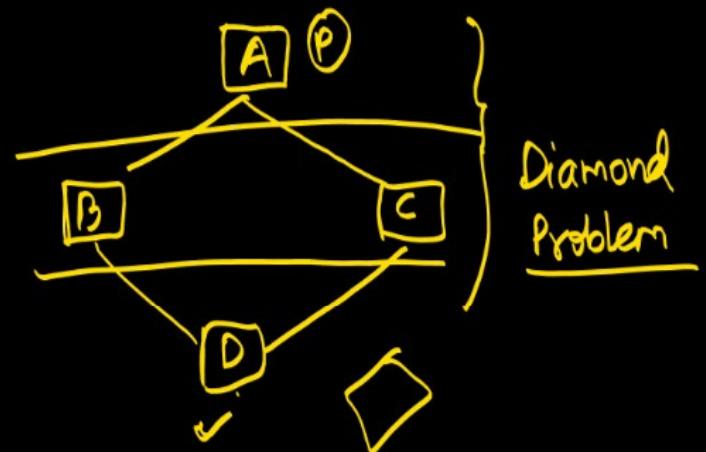
O/P ⇒  
a=10 b=10

\*\* Java में जब कोई derived class multiple base classes से properties को inherit करता है, तो Diamond problem create हो जाती है, इसलिए Java में Multiple inheritance की support नहीं किया जाता है।

\*\*\* As a solution Java में interface का use कर Multiple inheritance का use किया जा सकता है।

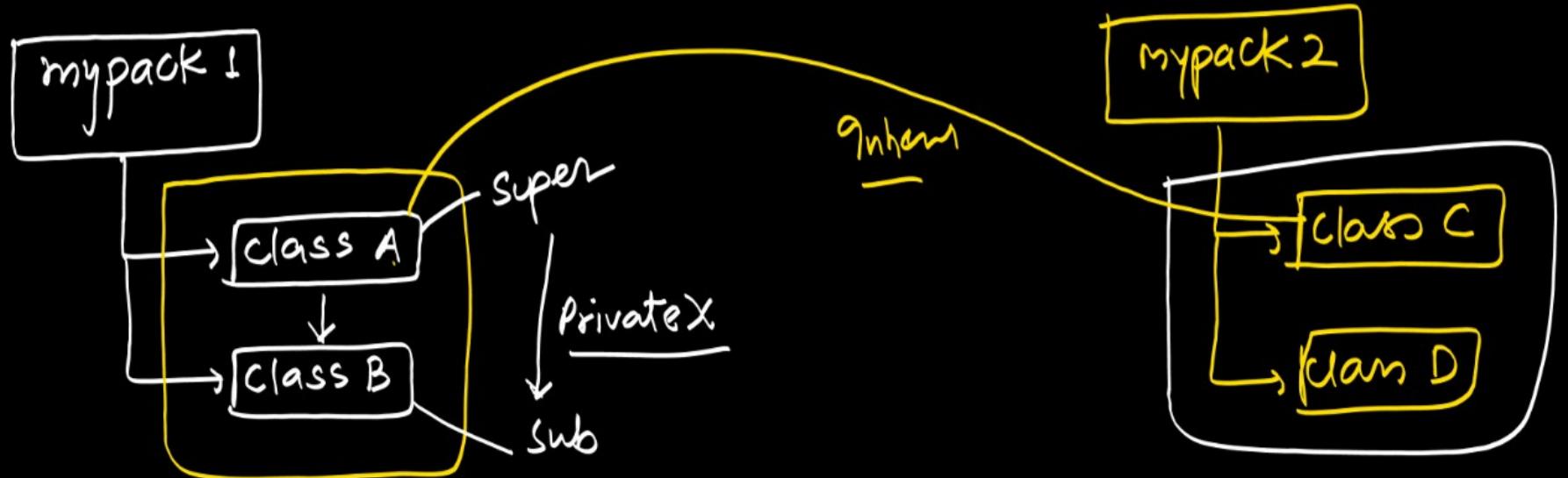
Ex :-

```
interface A { }
interface B { }
class C implements A, B
{ }
```



## \* Java Packages =

- Naming & Access control को handle करने में use.
- Same नाम करने वाली classes की एक ही group/package में स्थित होती है।
- classes की application के type के अनुसार arrange करता।
- दो प्रकार
  - A. Built-in → java.lang, java.io, java.awt (Abstract Window Toolkit),  
java.util
  - B. User defined → java.applet
- userdefined packages बनाने के लिए package keyword का use किया जाता है।



## \* \*

- \* किसी एक package की एक class की सारी properties को उस class में access किया जा सकता है।
- \* किसी एक package की super class की सारी properties को sub class में access किया जा सकता है, केवल private property को छोड़कर।
- \* किसी संक पैकेज की सारी classes के बीच उनकी सारी properties exchange की जा सकती है, केवल private property को छोड़कर।
- \* अलग-2 Packages की अलग-2 classes के बीच केवल public property exchange की जा सकती है।
- \* अलग-2 Packages के बीच inheritance हो तो एक package की super class की केवल public & protected property को ही अन्य package की sub class में access किया जा सकता है।

✓ 7665625262

	private	Protected	Public	Default
1. Same Class	✓		✓	
2. Same Package Subclass	✗	✓	✓	
3. Same Package Other classes	✗	✓	✓	
4. Other Package Sub class	✗	✓	✓	✗
5. Other Packages Other classes	✗	✗	✓	✗

## ★ Exceptions in Java ⇒

- यह एक event है, जो Program के Normal Flow को Disrupt करता है।
- Occur during the execution of the program.
- Due to some issues Like -
  - A. ऐसी file open कर रहे हैं, जो Memory में नहीं है
  - B. JVM के पास Memory का टक्कर ही जाना
  - C. user ने invalid data enter कर दिया है
- 2 प्रकार →
  - A. Unchecked
  - B. checked

#### A. Unchecked $\Rightarrow$

- $\rightarrow$  Runtime Exceptions
- $\rightarrow$  इन्हें Compile Time पर check नहीं किया जाता है।
- $\rightarrow$  Programming में गलती के कारण होता है।
- $\rightarrow$  जैसे -

- a. ArithmeticException = Number को zero से divide करना।
- b. IndexOutOfBoundsException = Array को Out of Bound Access करना।
- c. NullPointerException = Variable को initialize नहीं करना।
- d. IllegalArgumentException = API का सभी से कार्य नहीं करना।

$R =$  यदि किसी Array  
में 5 Elements हैं,  
और 6<sup>th</sup> Element  
को Access किया  
जाते होंगे तो  $O/P =$

(A) A  
(B) B

- B. Error CT
- C. Syntax Error
- D. MTA
- E. NOTA

### B. Checked $\Rightarrow$

- Compile Time पर check
- IOException / SQLException
- डीसे - FileNotFoundException

✓   
 class myexception extends Exception  
 {  
 }

\* Java में userdefined exceptions की allowed हैं।

Unchecked = RuntimeException Class को Extend करना

Checked = Exception Class को Extend करना

## \*Exception Handling =>

- यह सुनिश्चित करता है कि Exception होने के बाद भी program का फ्रिंम चलता रहे।
- इसके द्वारा Error के Type की identify किया जा सकता है।
- Error Handling Code की Normal code से छापाग लिख सकते हैं।
- Keywords => A. Try  
B. Catch  
C. Throw  
D. Throws  
E. Finally

## A. Try ⇒

- प्रद Exception Code के Block की define करता है।
- प्रद कर्नी अंकेला प्रदमें नहीं लिया जाता है, इसलिए इसके ठीक बाद Catch वा finally block होना ही चाहिए।
- इन Try, Catch & Finally को एक साथ भी प्रदमें ले सकते हैं। Sequence - TCF

Ex ⇒ try  
{  
—

} catch { }  
finally { }

गे sequence बटी टोरे पर  
CompileTime Error मार्गी।

### B. Catch ⇒

- Try Block के बाद use.
- यह Block एक Argument लेता है, जिसका Type = Throwable होना चाहिए।

### C. Finally ⇒

- इस Block का use important code की execute करने में किया जाता है। ऐसे - File Resources की close करना, DataBase Connection को Close करना।
- finally block को हमेशा execute किया जाता है, जहाँ Exception को Handle किया गया हो या नहीं।

## \* Final v/s Finally v/s Finalize ⇒

	Final	Finally	Finalize
1. Definition	Used for Constants, Methods & Classes	Used for Exception Handling	Used for Garbage Collection Cleanup
2. Usage	Prevents Modification	Ensures Cleanup Code Runs	Called Before Object is Garbage Collected
3. Applicable for	Variables, Methods & Classes	try-catch block के बाद use	objects
4. Override	✗	✗	✓

YR

#### D. Throw $\Rightarrow$

- Exception को Throw करने में use.
- Checked & Unchecked रोके प्रकार के Exceptions के साथ use.
- इस Keyword का use Generally User-defined Exceptions को throw करने में किया जाता है।

#### E. Throws $\Rightarrow$

- इसका use Exception को declare करने में किया जाता है, Throw करने में नहीं।
- Checked Exceptions को handle.
- पड़ि program में try-catch block का use नहीं करें तो throws का use किया जा सकता है।