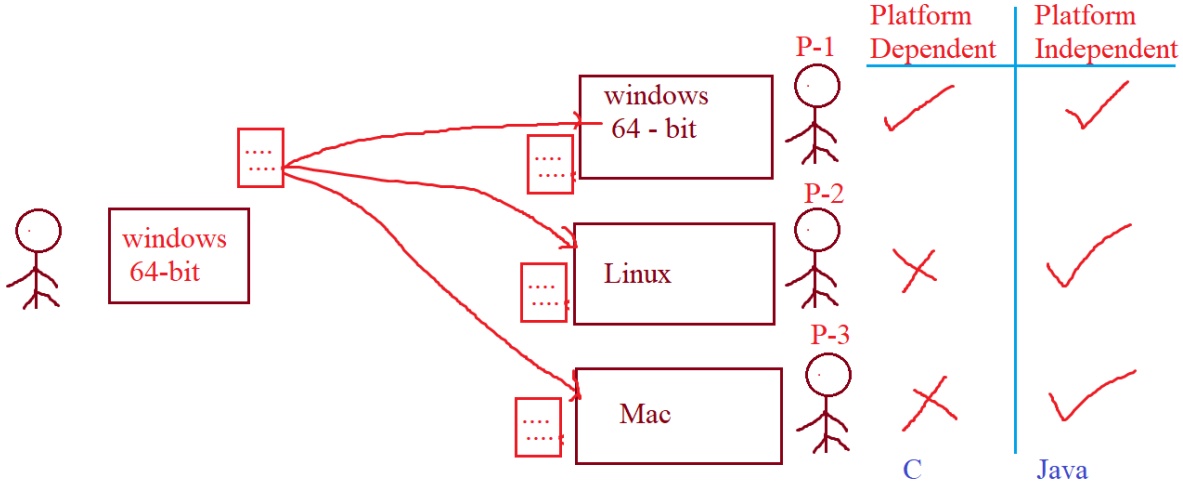
Core Java

--> Java is Platform independent programming Language

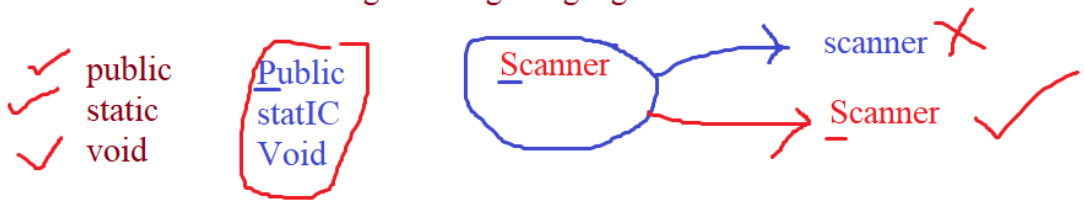
Platform:- Processor + Operating System.



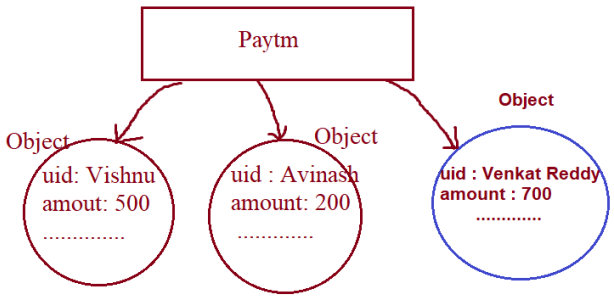
Java is a WORA Language

"Write Once Run Anywhere"

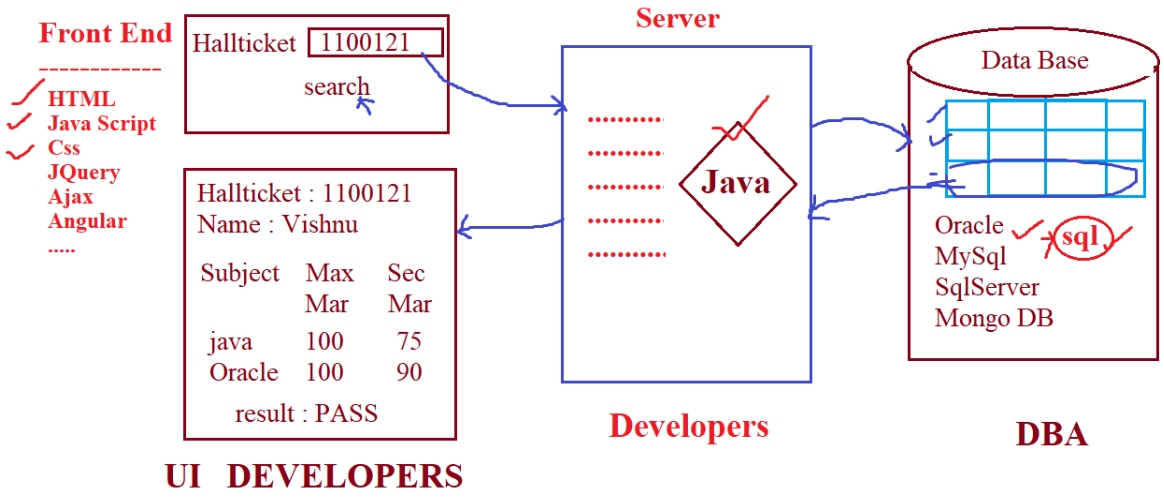
--> Java is Case-sensitive Programming Language



--> Java is a Object-Oriented Programming Language



V E N K A T R E D D Y

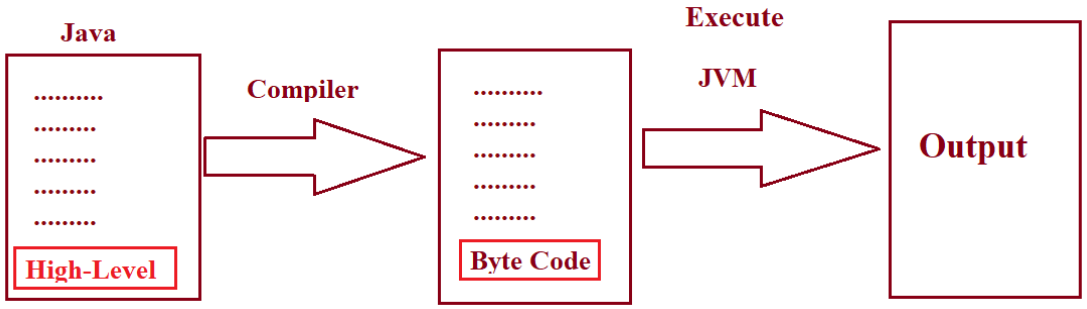


--> To work with java, we need JDK Software ...

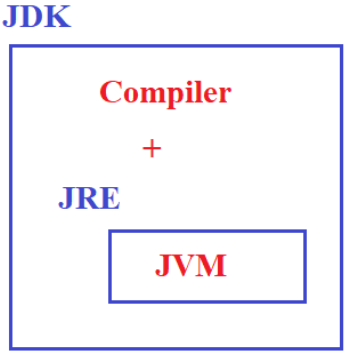
Java Development kit -->1.8

1) Open Any Editor (Notepad/notepad++/edit++,...)
2) Write your Program
3) Save Your Program
4) compile your program

5)Execute your program...



--> First it checks whether you follow all the rules or not ..
if there is no mistakes in our program then only it will convert into Byte code



--> JDK --> Developer

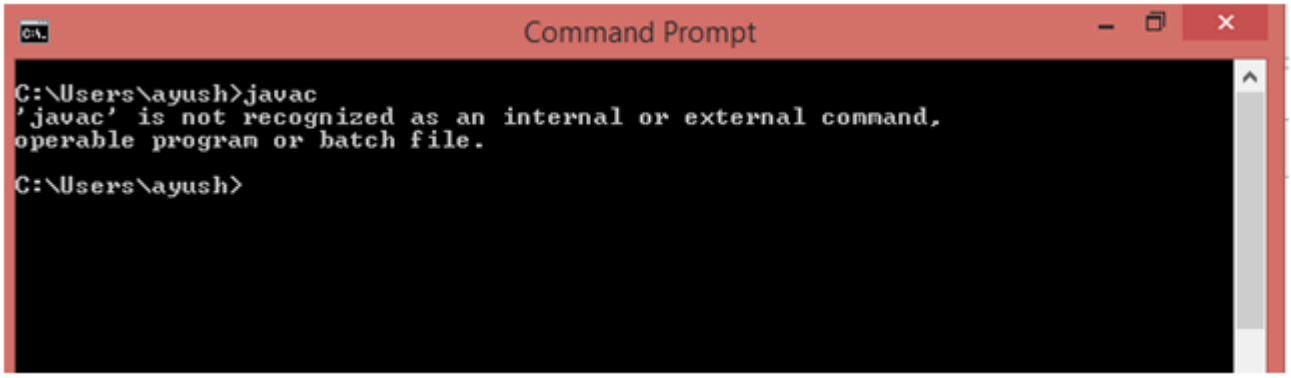
--> JRE --> Consumer

VENKATREDDY

Install Java

Step 1: Verify that it is already installed or not

Check whether Java is already installed on the system or not. In my case, it is not installed therefore I need to install JDK 1.8 on my computer.



Step 2: Download JDK

Click the below link to download jdk 1.8 for you windows 64 bit system.
Download JDK For Windows

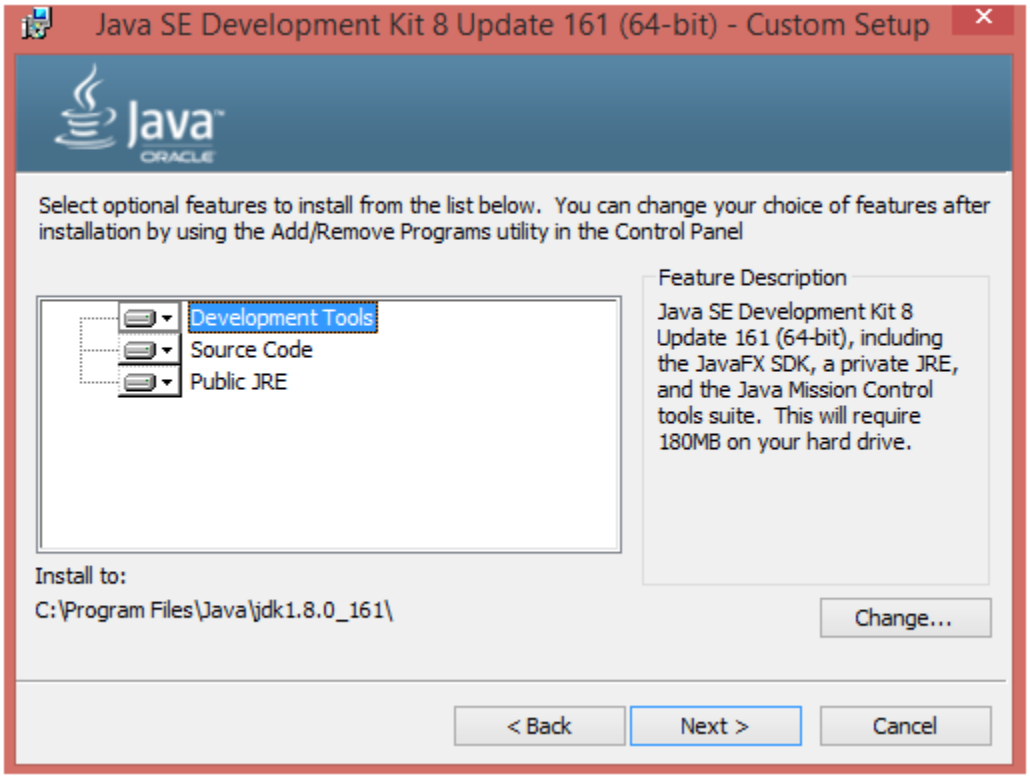
There are available releases for Linux and mac operating systems. You can visit the official link for JDK distributions i.e.
JDK Downloads

Open Google chrome , Just type JDK 1.8v download, then download JDK1.8v

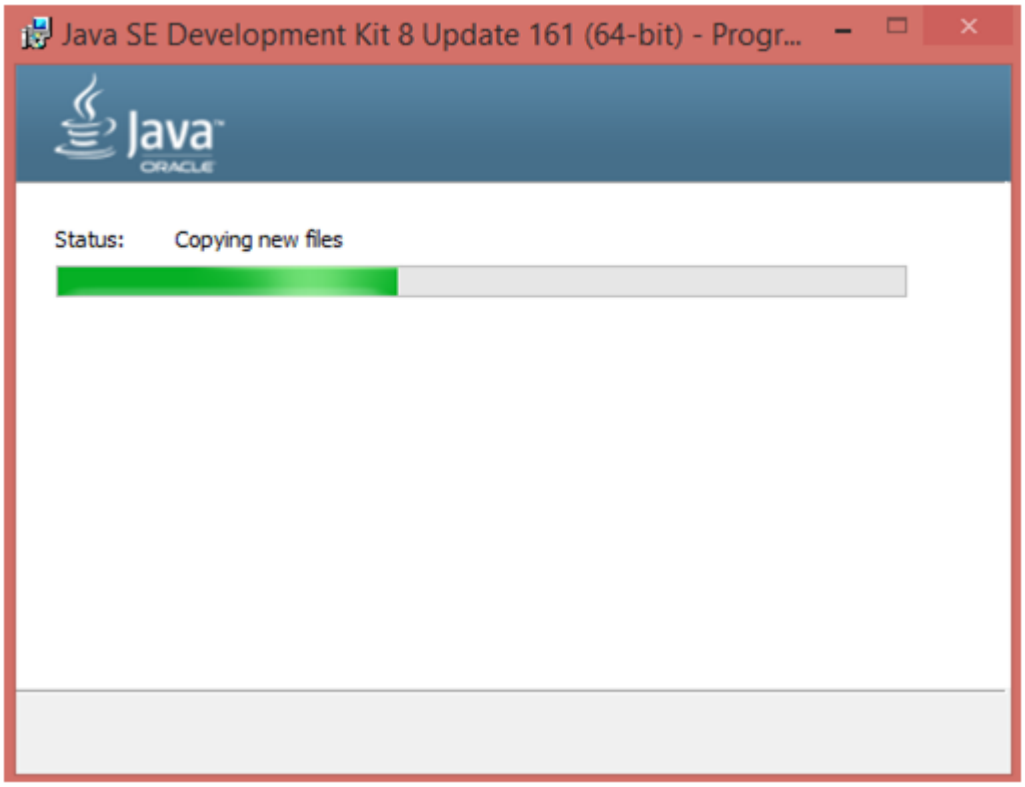
Step 3: Install JDK

Open the executable file which you have just downloaded and follow the steps.

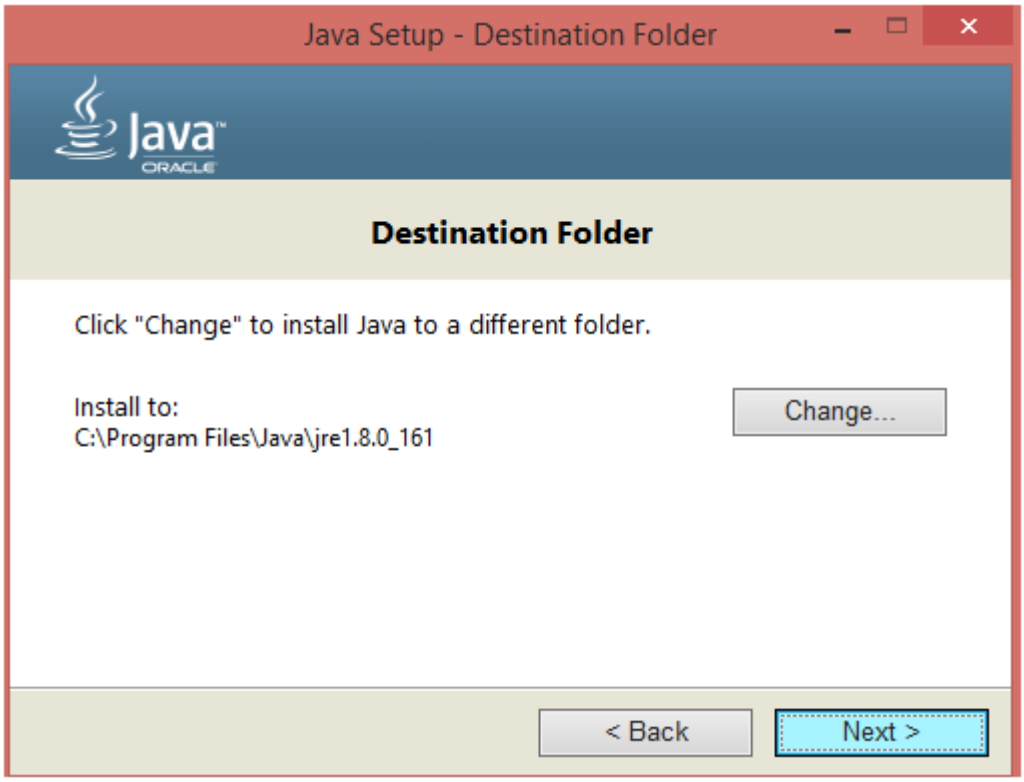
Click **Next** to continue



Just Choose Development Tools and click Next.



Set up is being ready.



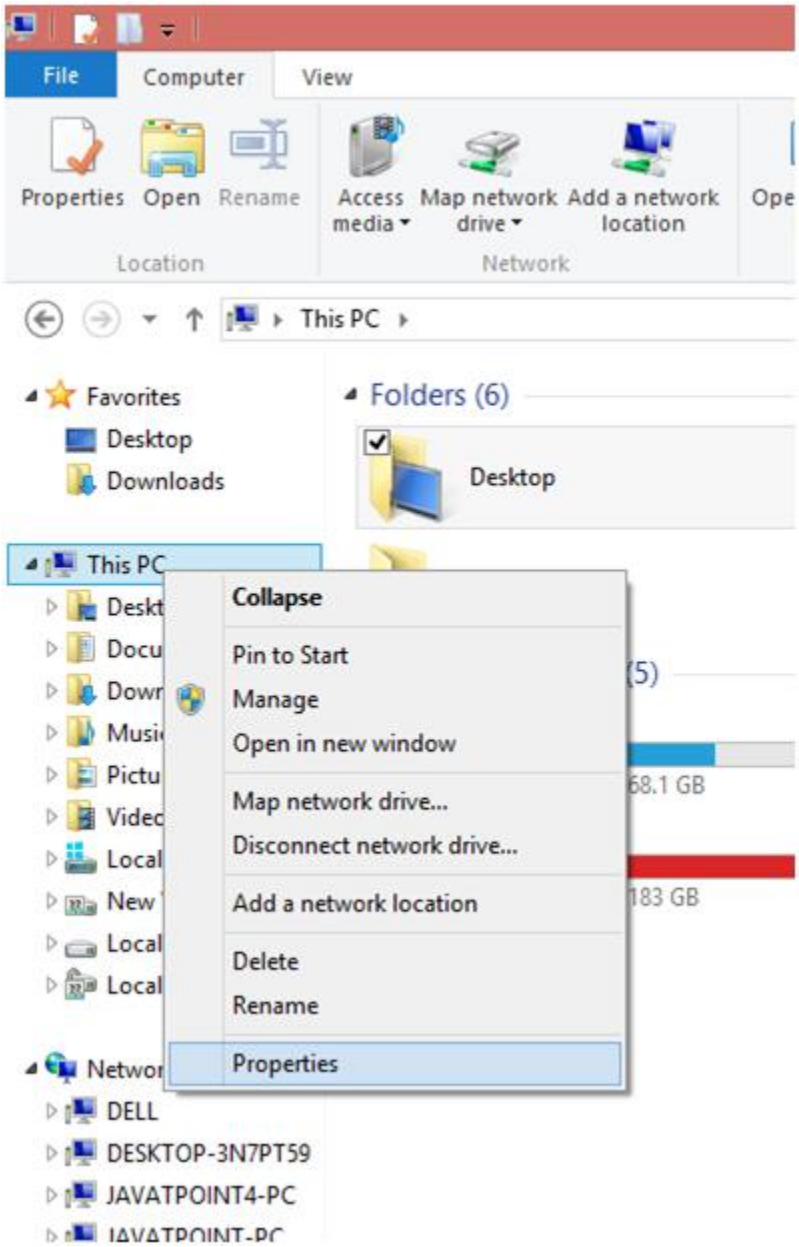
Choose the Destination folder in which you want to install JDK. Click Next to continue with the installation.

Set up is installing Java to the computer.

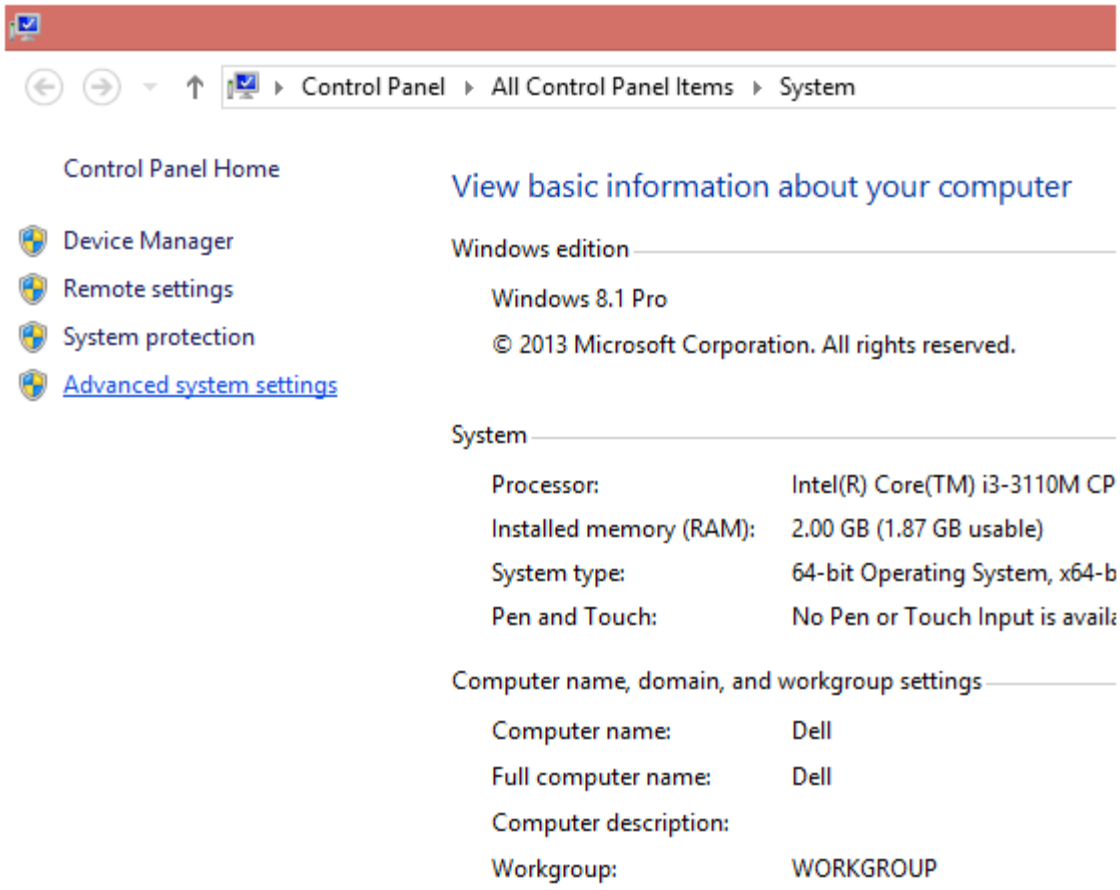
We have successfully installed Java SE development kit 8. Close the installation set up.

Step 4 : Set the Permanent Path

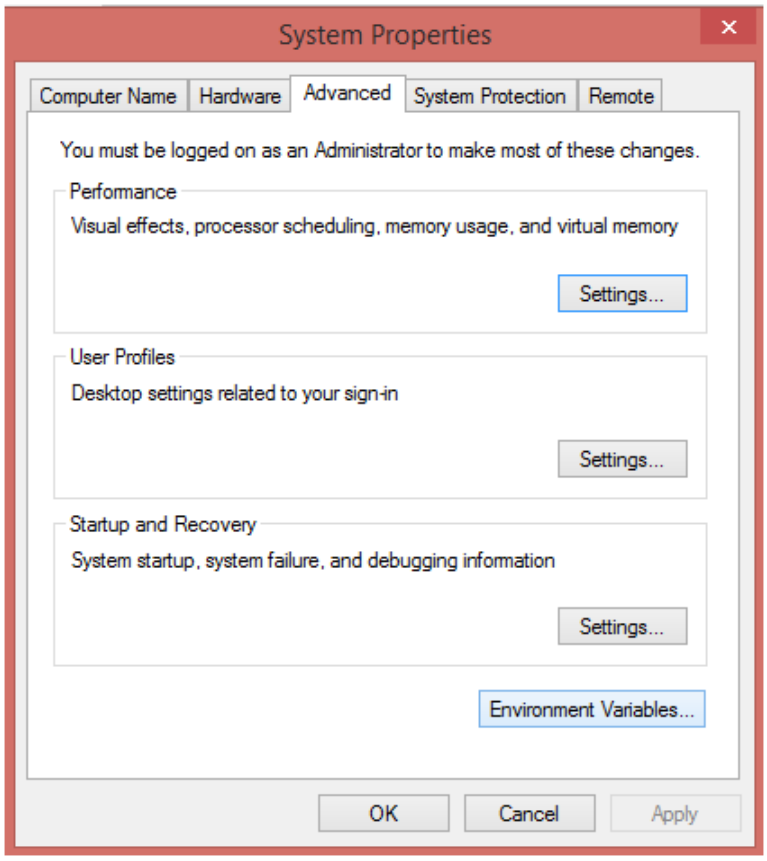
To execute Java applications from command line, we need to set Java Path. To set the path, follow the following steps.



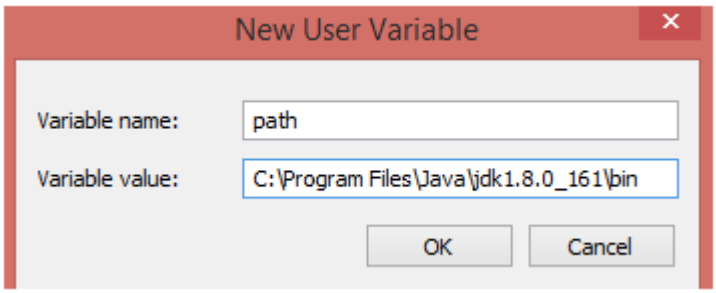
Right click on "this PC". It can be named as "My Computer" in some systems. Choose "properties" from the options.



The screen look alike the above image will open. Click on "Advanced system settings" to continue.



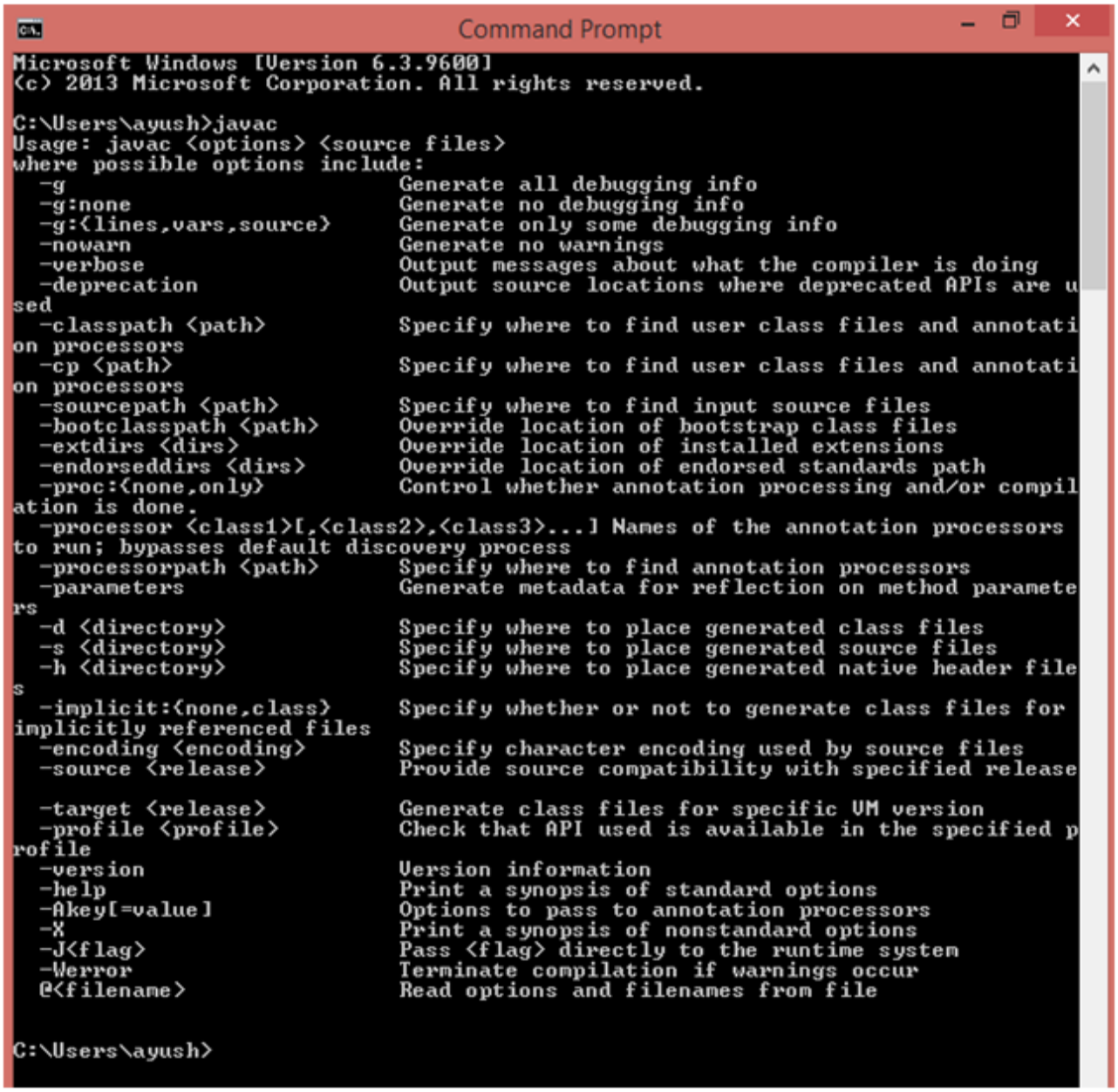
Above window will open. Click on "Environment Variables" to continue.



Enter "path" in variable name and enter the path to the bin folder inside your JDK in the variable value. Click OK.

Now Java Path has been set up. Open the Command prompt and type **"javac"** In case you have already open up the command prompt, I suggest you to close the existing window and reopen it again.

We will get javac executed as shown in the image below.



Tokens

--> Programming Elements which are used to do java Program

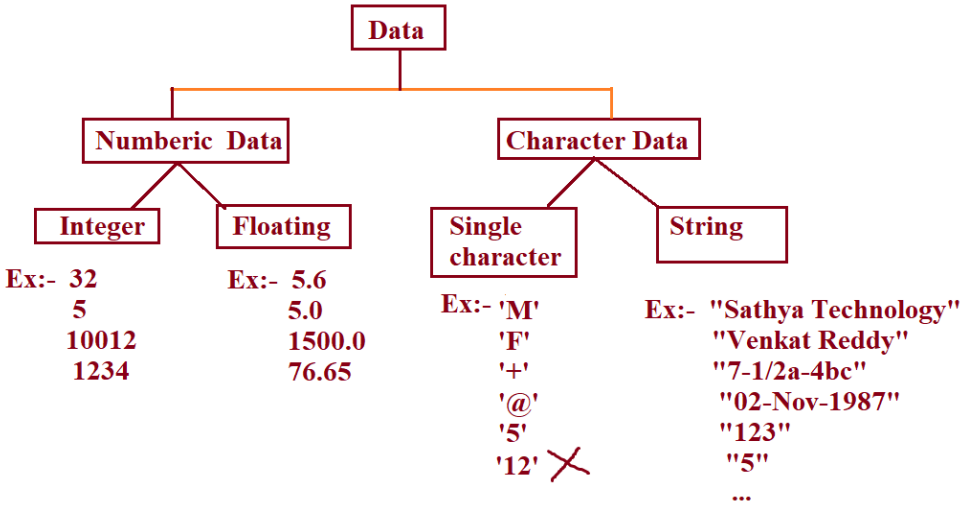
--> Tokens are 5 types

1) Literal
2) Constant
3) Operators
4) Keywords
5) Identifiers

VENKATREDDY

1) Literals

--> The values in our Program is called as Literals
--> Based on Data Literals are classified ..



1) "@" String
2) 5 integer
3) 2.0 floating
4) '12' Error
5) "Venkat" String

--> Based on data, Literals are classified into 5 types, Thye Are

1) Integer Literal --> 5, 120, -2745, ...
2) Floating Literal --> 76.98, 2.36, 1.278465, 5.0 , ...
3) Character Literal --> 'm', 'a' , '5', ...
4) String Literal --> "Venkat Reddy" , ...
5) Boolean Literal --> ture / false

```
int    age  = 29 ;
int    tokenno =  5 ;

double  pcost = 76.82 ;
double  h = 5.62 ;

char   gender = 'f ';
char   grade = 'A';

String  ename = "Venkat Reddy" ;
String  qualification = "MCA, M.Tech" ;

boolean   status = ture;
```


2) Constant:

--> Fixed values are called as constant

PI = 3.14 MAX_MARKS = 100

```
final  double  PI  =  3.14  ;

final  int  MAX_MARKS  =  100  ;
```

3) Operators

--> Any Symbol which performs an operation then it is called as operator

Ex: a + b ('+' is Operator, a and b are operands)

6 - 3 ('-' is Operator, 6 and 3 are operands)

p * q (' * ' is Operator, p and q are Operands)

m $ n (' $ ' is Symbol)

--> +, -, *, /, % (modulo Division)

--> 8 / 2 = 4

--> 5 / 2 = 2

--> 2 / 5 = 0

int / int = int

2) 5 (2 → Quotient
4
---
1 → Remainder

5) 2 (0
0

2

/ --> Quotient
% -> Remainder

8 % 2 = 0

5 % 2 = 1

2 % 5 = 2

Expression:

--> Combination of Operator and operand is called as Expression

Ex:- 5 + 3 125/10%10

--> Keywords

--> Pre-defined Words

--> Pre-defined words also called as Reserved Words

VENKATREDDY

Java Keywords

Package Created and Usage:

1) package
2) import

Class Creation

3) class
4) interface
5) enum (1.5)

Data types and Return-Types (8+1)

6) byte
7) short
8) int
9) long
10) float
11) double
12) char
13) boolean
14) void

Memory Allocation

15) new static

Control flow Statements

Conditional Statements (5)

16) if
17) else
18) switch
19) case
20) default

Looping Statement (3)

21) while
22) do
23) for

Transfer Statements (3)

24) break
25) continue
26) return

Setting Access level Permissions (4)

27) private -- default
28) Protected
29) Public

Execution level Modifiers (8)

30) static
31) final
32) abstract
33) native
34) transient
35) volatile
36) synchronized
37) strictfp

Establishing Inheritance Relationship (2)

38) extends
39) implements

Representing objects and its members (3)

40) this
41) super
42) instanceof

Exception Handling (5 +1)

43) throw
44) throws
45) try
46) catch
47) finally
48) assert (1.4 v)

Unimplemented Keywords (2)

49) goto
50) const
51) _ (underscore) (java 9v)

Default literals (3)

boolean literals
52) true
53) false

null literal
54) null

VENKATREDDY

--> Identifiers

--> Identifiers are user-defined words

--> The names given to identify Variables, methods, classes, interfaces and packages

Rules:

--> No Spaces

account number ✗

--> No Special Characters except $ and _

account-number ✗

account_number ✓

--> No Keywords

import ✗ Import ✓

class ✗

int ✗

--> First character should be starts with either alphabet / _ / $

7thrank ✗

rank7 ✓

_eid ✓

VENKATREDDY

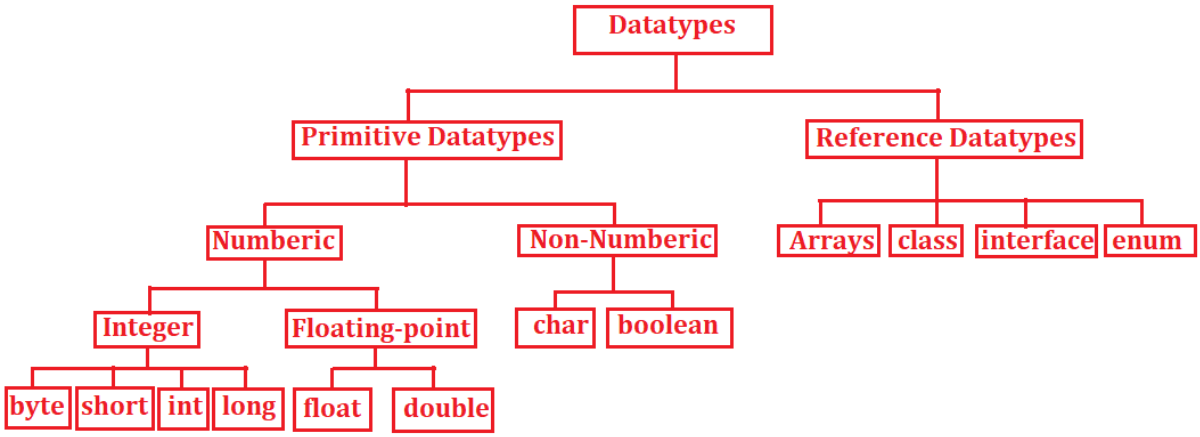
Data types:

--> To doany operation we need data..

--> The type of data is called as data type...

Data Types

Primitive Datatypes	Reference Datatypes
--> byte	--> Arrays
--> short	--> class
--> int	--> interface
--> long	--> enum
--> float	
--> double	
--> char	
--> boolean	



Datatypes	Memory Size	Default Value	Range
byte	1	0	-128 to 127
short	2	0	-32768 to 32767
int	4	0	-2147483648 to 2147483647
long	8	0	-2^{63} to $2^{63}-1$
float	4	0.0	1.4e-45 to 3.4e+38 (7 digits after decimal)
double	8	0.0	4.9e-324 to 4.9e+308 (15 digits after decimal)
char	2	space	0 to 65535
boolean	JVM dependent	false	true / false

Uni-code

A - 65, B - 66, C - 67, ... Z -90
a - 97, b - 98, c - 99, ... z – 122

Variables:

--> Variable is a Named Memory Location
--> It is used to store data
-->While giving names to variables we have to follow 4 rules of identifiers

Declaration:

Syntax:- Datatype variablename ;

Ex:-

```
int   a ;
byte  age ;
int   empid;
double  pcost ;
char  gender ;
String  ename;
```

Ex:- Datatype var1,var2,var3,..... ;

```
int  s1;
int  s2;        (or)   int   s1,s2,s3 ;
 int  s3;

double  p1cost ;
double  p2cost ;      (or)          double   p1cost, p2cost, p3cost ;
double  p3cost ;
```

Initialization:

--> Giving Values to variables for first time is called as Initialization
--> To give values to variables we have to use Assignment Operator(=)
Syntax:

```
Variable name        =       value  ;
 --------------              -----------
   Left                        Right
```

Ex:-

```
int       a ;
byte    age ;
int      empid;
double  pcost ;
char    gender ;
String  ename;
```

```
a = 100 ;
age = 32 ;
x = 25 ;   --> Invalid, Because variable x is not declared
empid = 1001 ;
pcost = 2500.4 ;
Gender = 'M' ;   --> Invalid, Because Gender is not declared
gender = 'M' ;   --> Valid
ename = "Venkat Reddy" ;
```


Note:- We can initialize values at the time of declaration

Syntax: Datatype variablename = value ;

Ex:-
```
int    a = 30;
double   pcost = 100.0;
String   ename = "Venkat Reddy" ;
char   gender = 'F';
int    s1=90, s2 = 100, s3 = 95 ;
double   p1cost = 253.50, p2cost = 186.32, p3cost = 200;
int    p = 5.24 ;  --> Invalid
float   q = 4.23f ;  (or)   float   q = 4.23F;
long   m = 254l;  (or)   long   m = 254L;
```

Assignment:

--> Giving Values to variables from second time onwards is called as Assignment

Ex:-
```
int   a ;                  // Decalration
a  = 100 ;                 // Initialization
a  = 150 ;                 //  Assignment
a  = 200 ;                 // Assignment
```

what is a value :- 200

Ex:-
```
int   p , q;
p = 5 ;
q = 7 ;
p = p + q ;
q = p + q ;
```

what is P value : 12
q value : 19

Example Program :

```
class Prog1
{
    public static void main(String[ ] args)
    {
        System.out.println(" Welcome to Venkat Reddy Sir Classes " ) ;

    }
}
```

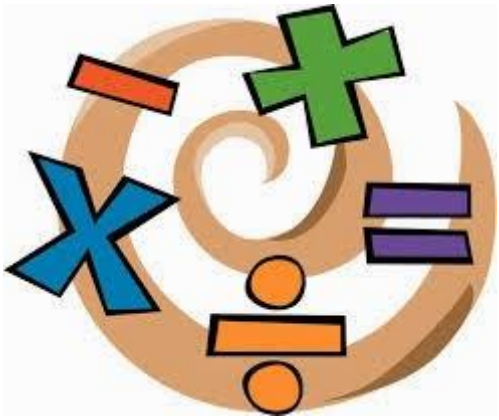
Example2:

```
// Consider p value is 4 and q values is 2.6 now calculate product of 2 numbers
// Output : Product is 10.4

class ProductEx2
{
    public static void main(String[] args)
    {
        int p = 4 ;
        double q = 2.6;
        double r = p * q ;
        System.out.println(" product is " + r );
    }
}
```

Operators

--> Any Symbol, which performs an operation then it is called as Operator

Ex:- a + b (+ is called as Operator, a and b are operands)
5 * 3 (* is Operator , 5 and 3 are operands)



In Java We have different types of operators

1) Arithmetic Operators
2) Relational Operators
3) Logical Operators
4) Assignment Operator
5) Increment/decrement Operator
6) Conditional Operator
7) new operator
8) dot(.) operator

.....

1) Arithmetic Operators

--> + , -, *, / , %

--> These are used to perform any type of calculations

--> The result of these operators are value ...

Ex:- a = 8, b = 4

a + b	--> 12	
a - b	--> 4	
a * b	--> 32	==> Value
a / b	--> 2	
a % b	--> 0	

2) Relational Operators

--> <, <=, >, >=, == , != are called as Realtional operators

--> Relational Operators are used to compare values

--> Relational operators are used to write conditions

--> The result of relational operators are boolean value (either true/false)

Ex:- a = 7 b = 7

a < b	--> 7 < 7	--> false
a <= b	--> 7 <= 7	--> true
a > b	--> 7 > 7	--> false
a >= b	--> 7 >= 7	--> true
a == b	--> 7 == 7	--> true
a != b	--> 7 != 7	--> false

Logical Operators:

--> &&, || , ! are called as Logical Operators

--> If you want to combine more than one condition result then use logical operators

→ Logical Operators always gives Boolean value as a result

➔ && operator returns true only if all the conditions are true

➔ || operator returns true if any one condition is true

4) Assignment Operator

--> ' = ' is called as assignment operator

--> It is also called as Right to Left Operator

--> If You want to initialize values to variables then use Assignment operator

variable = value ;

--- -------

left right

5) Increment / decrement operator

Increment Operator (++)

--> It is used to increase the current value by one.

--> int x = 5 ; int x = 5 ;

x = x + 1 ; x++;

Pre-Increment (++ Variable name)

First value is increased then operation is performed

Post-Increment(Variable name ++)

```
int x = 5, y =5 ;
System.out.println(++x);
System.out.println(x);
System.out.println(y++);
System.out.println(y);
```

Decrement Operator (--)

--> It is used to decrease the current value by 1.

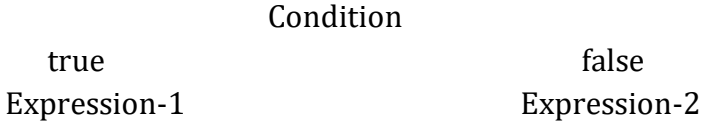
1) Pre Decrement Operator (-- Variable Name)

2) Post Decrement Operator (Variable Name --)

6) Conditional Operator

--> ? : are called as Conditional Operator

--> Based on condition it will execute statements



Syntax:

```
                                        true          false
Variable =    ( Condition ) ?  Expression-1  :  Expression-2 ;
```

Ex: print given number is +ve or not

```
int  n = sc.nextInt( );
String   res = ( n > 0 ) ? " +ve Number "  :  " -ve Number "  ;
System.out.println(res) ;
```

Example1:

```
class  Ex1
{
        public static void main(String[] args)
        {
                int  n = -5 ;
                String   res = ( n > 0 ) ? " +ve Number "  :  " -ve Number "  ;
                System.out.println(res) ;
        }
}
```

7) new Operator:

--> new operator is used to allocate memory

--> To create objects, we have to use new operator

--> java is a object-oriented programming lang...

--> we have to programs by using class and object

8) dot(.) Operator

--> it is used at the time of calling the methods... and calling variables

We are reading values by using Assignment operator

int a = 10 ;

If you want to read values dynamically then Scanner class...

--> Scanner class is available in java.util package


```
// Addition of 2 numbers
import java.util.Scanner;
class AddEx1
{
        public static void main(String... args)
        {
                Scanner sc = new Scanner(System.in);
                System.out.println(" Enter a value : ") ;
                int a = sc.nextInt( );
                System.out.println(" Enter b value : ");
                int b = sc.nextInt( );
                int c = a + b ;
                System.out.println(" Sum is : "+c);
        }
}
```

Scanner Class

byte	--> nextByte( )
short	--> nextShort( )
int	--> nextInt( )
long	--> nextLong( )
float	--> nextFloat( )
double	--> nextDouble( )
char	--> .....
boolean	--> nextBoolean( )
String	--> next( )

If you want to read single character data

```
char gender = sc.next( ).charAt(0) ;
```

Example:

```
------------
class  EmpEx1
{
        public  static  void  main(String...  args)
        {
                Scanner   sc = new  Scanner(System.in);
                System.out.print(" Enter  Emp id : ");
                int  eid = sc.nextInt( );
                System.out.print(" Enter Emp Name : ");
                String  ename = sc.next( );
                System.out.print(" Enter  Emp gender : ");
                char  gender = sc.next( ).charAt(0);
                System.out.print(" Enter Salary : ");
                double  salary = sc.nextDouble( );

                System.out.println(" Emp  id  is : "+eid);
                System.out.println(" Name  is  : "+ename);
                System.out.println(" Gender  is  : "+gender);
                System.out.println(" Salary is   : "+salary);
        }
}
```


Control Flow Statements

--> Control Flow Statements are used to control the flow of execution

--> We have 3 types of control flow statements

* Conditional Statements / Selection Statements / Decision-Making Statements
* Looping Statements / Iterative Statements
* Transfer Statements / Jumping Statements

Control Statements

Selection Statements	Looping Statements	Transfer Statements
* if	* for	* continue
* if-else	* while	* break
* else - if	* do-while	* return
* Nested if	* for-each	
* switch		

1)if

--> If you want to execute set of statements based on condition then we have to use if

```
-->     if( codition )
        {
                S.o.println(" Hello ");
                S.o.println(" Hai " );
        }
```

Syntax:-

```
if(condition)
        Statement - 1 ;
```

Syntax2:-

```
if( condition )
{
        Statement - 1 ;
        Statement - 2 ;
}
```

Ex:-

W. a. p to check given number is +ve or not... incase +ve number then only display number and "HAI" message

```
int   n = sc.nextInt( );
if( n > 0 )
{
        System.out.println(n) ;
        System.out.println("Hai ");
}
```

if-else

--> if you want to execute one set of statements out of two set of statements based on condition then we have to use if-else

if the condition is true then it will execute if-block otherwise it will execute else block

Syntax:

```
if(Condition)
{
        statement - 1 ;   // if - block
        ..............
}
else
 {
        Statement - 2 ;   // else - block
        ..............
  }
  Statement  - 3 ;
```

Condition

True	False
Statement-1	Statement-2
Statement-3	Statement-3

1)Read age , then check whether person is Eligible for Vote or not

```
age = sc.nextInt( );
if( age >= 18 )
        System.out.println(" Eligible for Vote ");
else
        System.out.println(" Not Eligible for Vote ");

System.out.println(" Thank You ");
```

2) Read 3 subject marks then
check Student is pass / Fail , in case Pass then only calculate total and avg marks
Otherwise display "FAIL" Message

```
import java.util.Scanner;
class StudentEx1
{
        public static void main(String[ ] args)
        {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter 3 subject Marks : ");
                int s1 = sc.nextInt( );
                int s2 = sc.nextInt( );
                int s3 = sc.nextInt( );
                if(s1>=35 && s2>=35 && s3>=35)
                {
                        System.out.println(" PASS ");
                        int total = s1+s2+s3;
                        double avg = total/3;
                        System.out.println(" Total Marks : " + total);
                        System.out.println(" Average is : " +avg);
                }
                else
                {
                        System.out.println(" Fail ");
                }

        }
}
```

else-if

--> if you want to execute one set of statements out of more than two set of statements based on condition then we have to use else – If

Syntax:

```
if( condition )
        Statement-1 ;
else if(condition)
        Statement-2 ;
else  if(condition)
        Statement-3 ;
else
        Statement-4 ;
```

Ex:-

```
int   n = sc.nextInt( );
if(n>0)
        System.out.println(" +ve Number ");
else if(n<0)
        System.out.println(" -ve Number ");
else
        System.out.println(" Zero ")
```


Nested If

--> A If inside of another if is called as Nested if

```
if( Condition )        // External If
{
        Statement - 1 ;
        if(Condition)        // Internal If
        {
                Statement - 2 ;
        }
}
Statement - 3 ;
```

Syntax2:

```
if(Condition)
{
        statement - 1 ;
        if(Condition)
                Statement - 2 ;
        else
                Statement - 3;
}
```

Syntax3:-

```
if(Condition)
{
        Statement - 1 ;
        if(Condition)
                Statement - 2;
        else if(Condition)
                Statement - 3 ;
        else
                Statement - 4
}
```

Syntax4:-

```
if(Condition)
{
        Statement - 1;
         if(Condition)
                Statement - 2 ;
         else
                Statement - 3;
}
else
        Statement - 4
```

--> Read number, in case it is not zero then only check given number is +ve or -ve

```
        n!=0
        true
        n>0
    true        false
    +ve         -ve
```

```
import   java.util.Scanner;
class   NestedIfEx1
{
        public  static  void  main(String[ ]   args)
        {
                Scanner   sc = new  Scanner(System.in);
                System.out.print("Enter any Number : ");
                int   n  =  sc.nextInt( );
                if(n!=0)
                {
```

```
                                        if(n>0)
                                                System.out.println(" +ve Number ");
                                        else
                                                System.out.println(" -ve Number ");
                                }
                                System.out.println(" Thank You ");
                        }
                }
```

```
   n = 0
Thank You
```

switch

--> If we have more number of conditions then we have to use Switch case

--> If we have menu type programming then use switch case

Syntax:-

```
switch(variable)
{
        case  constant-1 :     Statement - 1 ;
        case  constant-2:      Statement - 2 ;
                        .....................................................
                        .....................................
        case    constant-n:    Statement - n ;
        default :              statement ;
}
```

Ex:1

```
1       -       ONE
2       -       TWO
3       -       THREE
import   java.util.Scanner;
class SwitchEx1
{
        public  static  void  main(String[]  args)
        {
                Scanner   sc = new  Scanner(System.in);
                System.out.print("Enter any Number : ") ;
                int  n = sc.nextInt( );
```

```
                    switch(n)
                    {
                            case  1:        System.out.println(" Hai ");
                                                    System.out.println(" ONE ");

                            case  2:  System.out.println(" TWO ");

                            case  3 :  System.out.println(" THREE ");

                            default :  System.out.println(" Invalid Number ");
                    }

            }
    }
```

Example2:-

```
import  java.util.Scanner;
class SwitchEx2
{
    public  static void  main(String[ ]  args)
    {
            Scanner  sc = new  Scanner(System.in);
            System.out.println(" Enter any 2 numbers : ");
            int  a = sc.nextInt() ;
            int  b = sc.nextInt( );
            System.out.println("1. Addition ");
            System.out.println("2. Subtraction ");
            System.out.println("3. Multiplication ");
            System.out.println("4. Division ");
            System.out.print(" Enter your choice  : ");
            int  ch = sc.nextInt( );
            int  res ;
            switch(ch)
            {
                    case  1:  res = a + b ;
                              System.out.println(" Sum  is : "+res);
                              break;
                    case  2:  res =  a - b ;
                              System.out.println(" Sub  is : "+res);
                              break;
                    case  3:  res  = a * b;
                              System.out.println(" Mul  is : "+res);
                              break;
                    case  4:  res = a / b ;
```

```
                        System.out.println(" Div  is : "+res);
                        break;
                default :       System.out.println(" Invlaid Choice ");
            }
        }
}
```


Example:

```
r       -       RED
g       -       GREEN
b       -       BLUE

import   java.util.Scanner;
class   SwitchEx3
{
        public  static  void  main(String[ ]   args)
        {
                Scanner   sc = new  Scanner(System.in);
                System.out.print(" Enter your character ");
                char  ch = sc.next().charAt(0);
                switch(ch)
                {
                        case  'r' : System.out.pritnln(" RED ");
                                    break;
                        case  'g' : System.out.println(" GREEN ");
                                    break;
                        case  'b' : System.out.println(" BLUE ");
                                    break;
                }
        }
}
```

Example:

```
import   java.util.Scanner;
class   SwitchEx4
{
        public  static  void  main(String[ ]   args)
        {
                Scanner   sc = new  Scanner(System.in);
                System.out.print(" Enter your character ");
                char  ch = sc.next().charAt(0);
                switch(ch)
               {
                        case  'r' :
                        case  'R' : System.out.println(" RED ");
                                    break;
                        case  'g' :
                        case 'G' : System.out.println(" GREEN ");
                                    break;
                        case  'b' :
                        case  'B': System.out.println(" BLUE ");
                                    break;
               }
        }
}
```

Lopping Statements

--> Looping statements are used to execute certain statements multiple times

statement -1
statement -2

--> Looping means iterations

fixed	unknown
for Loop	while Loop

for Loop

--> for Loop is also called as single line looping statement

Syntax:

=====

```
for(Initialization ;  Condition ;   increment / decrement )
{
        Statement - 1 ;
        Statement - 2 ;
}
```

--> I want to display Venkat Reddy Message 1 time

```
System.out.println(" Venkat Reddy ");
```

--> I want to display Venkat Reddy Message 5 time

```
System.out.println(" Venkat Reddy ");
System.out.println(" Venkat Reddy ");
System.out.println(" Venkat Reddy ");
System.out.println(" Venkat Reddy ");
```

```
1 < 6   T
2 < 6   T
3 < 6   T
4 < 6   T
5 < 6   T
6 < 6    F
```

```
for(int  i = 1;  i < 6 ; i++ )
{
        System.out.println(" Venkat Reddy ");
}
```

W.a.p to display 1 to 5 numbers

```
for(int  i = 1 ; i <= 5 ; i++)
{
        System.out.pritnt(i+"\t");
}
```

```
class  ForEx1
{
        public static void main(String[] args)
        {
                for(int  i = 1 ; i <= 5 ; i++)
                {
                        System.out.print(i+"\t");
                }
        }
}
```

--> W.a. p to display 1 to n numbers

```
import  java.util.Scanner;
class  ForEx2
{
        public static void main(String[] args)
        {
                Scanner   sc = new  Scanner(System.in);
                System.out.print("Enter any Number : ");
                int  n = sc.nextInt( );
                for(int  i = 1 ; i <= n; i++)
                {
                        System.out.print(i+"\t");
                }
        }
}
```

Example2:

```
1       2       3       4
1       2       3       4
1       2       3       4
```

```
for(int  i=1;i<=3 ; i++)          //  rows
{
        for(int   j= 1 ; j<=4; j++)               // Columns
        {
                System.out.print("*  ");
        }
        System.out.println( );
}
```

while Loop

--> While loop is used to iterate statements when we don't know number of iterations
--> While loop is also called as Entry Control Loop

Syntax:

```
while(Condition)
    {
        Statement -1 ;
        Statement -2 ;
        ..................
    }
```

--> First it will check While-Condition, if it is true then it will execute while body other wise it never execute while body

Example:

--> W.a.p to find sum of digits in a given number

253 --> Sum of digits is : 10

```
import java.util.Scanner;
class WhileEx1
{
    public static void main(String[ ] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print(" Enter any number : ");
        int n = sc.nextInt() ;
        int r , sum = 0 ;
        while(n>0)
        {
            r = n % 10 ;
            n = n / 10;
            sum = sum + r;
        }
        System.out.println(" Sum of digits is : "+sum);
    }
}
```

do_while

--> do_while loop is also called as Exit control Loop

-->In do_while Loop first it will execute statements at end it will check condition

Syntax:

```
do
{
        Statement - 1 ;
        Statement - 2;
        .....................
}
while (Condition);
```

Ex:- sum of given numbers

Output:

```
Enter any Number :  5
Do you want to enter any other number : Y
Enter any Number : 3
Do you want to enter any other number : Y
Enter any Number : 9
Do you want to enter any other number : N
Sum of Given Numbers : 17
```

```
import   java.util.Scanner;
class  DoWhileEx1
{
        public static void main(String[ ]   args)
        {
                Scanner   sc =  new  Scanner(System.in);
                int   n ,sum =0;
                char  ch;
                do
                {
                   System.out.print("Enter any Number : ");
                   n = sc.nextInt( );
                   sum = sum + n ;
                   System.out.print(" Do you want to Enter any other Number (y/n) : ");
                  ch = sc.next( ).charAt(0);
                }
                while (ch=='y');
```

```
                System.out.println(" Sum of given numbers : "+sum);

        }
}
```

Transfer Statements

--> Transfer statements are used to pass your control from one place to another place

* continue
    --> only in looping statements
    --> It will stop current iteration and execution starts from next Iteration
* break
    --> switch (or) Looping Statements
    --> To stop all iterations
* return
    --> Methods

Example1:

```
for(int  i = 1 ; i<=5 ; i++)
{
        if(i%2==0)
          {
                continue;
          }
        System.out.print(i+"  ");
}
```

Output:- 1 3 5

Example2:

```
for(int  i = 1 ; i<=5 ; i++)
{
        if(i%2==0)
          {
                break;
          }
        System.out.print(i+"  ");
}
```

Output: 1

Arrays:

Variables : Named Memory Location, Which is used to store data .
In Variables at a time we can store only one value

```
int  a = 5 ;
a = 7 ;
```

Arrays: An Array is a group of same type of data..
I have to store all subject marks
In Arrays index number always starts with 0 and ends with size-1
Arrays are classified into 3 types

1) 1-D
2) Multi-D
3) Jagged Array

1) 1 - D Arrays

--> 1-D Array is used to store row type of data...

Declaration

Syntax:

```
Datatype  Arrayname[ ] = new  Datatype[Size];
```

Ex:-

```
int  x[ ] = new int[5];
double cost[ ] = new  double[3] ;
```

Whenever we declare arrays, it will create array object then by default length variable is created it will store the size of array.

Ex1:-

```
class  ArrayEx1
{
    public static void main(String[ ]  args)
    {
        int  x[] = new  int[5];
        x[0] = 5;
        x[1] = 12 ;
        x[2] = 7;
        x[3] = 9;
        x[4] = 15;
        System.out.prinltn(" Your Array Elements are ");
        System.out.print(x[0]+"\t");
        System.out.print(x[1]+"\t");
        System.out.print(x[2]+"\t");
```

```
                System.out.print(x[3]+"\t");
                System.out.print(x[4]+"\t");
        }
}
```

Ex2:-

```
class  ArrayEx2
{
        public  static  void  main(String[ ]  args)
        {
                int   x[] = new  int[5];
                x[0] = 5;
                x[1] = 12 ;
                x[2] = 7;
                x[3] = 9;
                x[4] = 15;
                System.out.println(" Your Array Elements are ");

                        for(int  i = 0; i < 5 ; i++)
                {
                                System.out.print(x[ i ] +"\t");
                }

        }
}
```

Ex3:-

```
// Read Array Elements dynamically
import  java.util.Scanner;
class  ArrayEx3
{
        public  static  void  main(String[ ]  args)
        {
                Scanner  sc = new  Scanner(System.in);
                int   x[] = new  int[5];

        /*      x[0] = sc.nextInt( );
                x[1] = sc.nextInt( );
                x[2] = sc.nextInt( );
                x[3] = sc.nextInt( );
                x[4] = sc.nextInt( );*/

                        System.out.println(" Enter 5 Elements ");
```

```
                    for(int  i =0; i<5 ; i++)
                    {
                            x[ i ] = sc.nextInt( );
                    }

                System.out.println(" Your Array Elements are ");

                    for(int  i = 0; i < 5 ; i++)
                {
                            System.out.print(x[ i ] +"\t");
                }

        }
}
```


Ex:

```
// Read Array Size and  Elements dynamically
import   java.util.Scanner;
class   ArrayEx4
{
        public  static  void  main(String[ ]   args)
        {
                 Scanner   sc = new   Scanner(System.in);
                System.out.print(" Enter size of Array : ");
                int  size = sc.nextInt( );

                 int   x[ ] = new   int[size];

                    System.out.println(" Enter " + size +" Elements ");
                    for(int  i =0; i<size ; i++)
                    {
                            x[ i ] = sc.nextInt( );
                    }

                 System.out.println(" Your Array Elements are ");

                    for(int  i = 0; i < size ; i++)
                {
                            System.out.print(x[ i ] +"\t");
                }

        }
}
```

Ex:

```
// Using length variables
import   java.util.Scanner;
class   ArrayEx5
{
        public  static  void  main(String[ ]   args)
        {
                Scanner   sc = new   Scanner(System.in);
               System.out.print(" Enter size of Array : ");
               int  size = sc.nextInt( );

                int    x[] = new   int[size];

                       System.out.println(" Enter " + size +" Elements ");
                       for(int  i =0; i<size ; i++)saf
                       {
                               x[ i ] = sc.nextInt( );
                       }

                System.out.println(" Your Array Elements are ");

                       for(int  i = 0; i < x.length ; i++)
                {
                               System.out.print(x[ i ] +"\t");
                }

        }
}
```



```
// Read Array Elements dynamically
import   java.util.Scanner;
class   ArrayEx3
{
        public  static  void  main(String[ ]   args)
        {
                Scanner   sc = new   Scanner(System.in);
                 int    x[] = new   int[5];

        /*       x[0] = sc.nextInt( );
                 x[1] = sc.nextInt( );
                 x[2] = sc.nextInt( );
```

```
                x[3] = sc.nextInt( );
                x[4] = sc.nextInt( );*/

                        System.out.println(" Enter 5 Elements ");
                        for(int  i =0; i<5 ; i++)
                        {
                                x[ i ] = sc.nextInt( );
                        }

                System.out.println(" Your Array Elements are ");

                        for(int  i = 0; i < 5 ; i++)
                {
                                System.out.print(x[ i ] +"\t");
                }

        }
}
// Read Array Size and  Elements dynamically
import   java.util.Scanner;
class   ArrayEx4
{
        public  static  void  main(String[ ]   args)
        {
                 Scanner   sc = new   Scanner(System.in);
                System.out.print(" Enter size of Array : ");
                int  size = sc.nextInt( );

                 int    x[] = new   int[size];

                        System.out.println(" Enter " + size +" Elements ");
                        for(int  i =0; i<size ; i++)
                        {
                                x[ i ] = sc.nextInt( );
                        }

                 System.out.println(" Your Array Elements are ");

                        for(int  i = 0; i < size ; i++)
                {
                                System.out.print(x[ i ] +"\t");
                }
```

```
        }
}

// Using length variables
import   java.util.Scanner;
class   ArrayEx5
{
        public  static  void  main(String[ ]   args)
        {
                 Scanner   sc = new   Scanner(System.in);
                System.out.print(" Enter size of Array : ");
                int  size = sc.nextInt( );

                 int    x[] = new   int[size];

                        System.out.println(" Enter " + size +" Elements ");
                        for(int  i =0; i<size ; i++)
                        {
                                x[ i ] = sc.nextInt( );
                        }

                 System.out.println(" Your Array Elements are ");

                        for(int  i = 0; i < x.length ; i++)
                {
                                System.out.print(x[ i ] +"\t");
                }

        }
}
```


for-each Loop

--> for each loop is used to apply on group of values

Syntax:

```
                for(Datatype  variablename : Arryname)
                {
                        Statement-1 ;
                        Statement-2 ;
                        ..............
                }
```

Ex:-

```
int  a[ ] = new  int[6] ;
a[0] = 6;
a[1] = 9;
a[2] = 3;
a[3] = 5;
a[4] = 1;
a[5] = 2 ;
int   a[ ] = {6, 9, 3, 5, 1, 2 };
for(int  i=0;i<6;i++)
{
        System.out.println(a[i]);
}

int   a[ ] = {6, 9, 3, 5, 1, 2 };
for (int   x : a)
{
        System.out.print(x+"\t");
}
```

Ex1:

```
class  ForEachEx1
{
        public static void main(String[] args)
        {
                int   a[ ] = {6, 9, 3, 5, 1, 2 };
                for (int   x : a)
                {
                        System.out.print(x+"\t");
                }
        }
}

double  a[ ] ={ 3.65,  2.43,  1.71 }
```

--> W.a.p to read n numbers then perform sum of all those numbers

```
import java.util.Scanner;
class ArrayEx6
{
	public static void main(String[ ] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.print(" Enter n value : ");
		int n = sc.nextInt( );
		int a[ ] = new int[n];
		System.out.println("Enter "+ n +" Elements ");
		for(int i = 0; i < a.length ; i++)
		{
			a[i] = sc.nextInt( );
		}
		int sum = 0;
		System.out.println(" Your Array elements are ");
		for(int x : a)
		{
				System.out.print(x+"\t");
				sum = sum + x ;
		}
		System.out.println(" Sum of all Number : "+sum);
	}
}
```


Task:

--> W.a.p to read student marks, then caluclate total and average

Step1: Read no.of subjects
step 2: Create an Array
Step 3: Read subject marks
Step 4: Dispaly marks then calculate total marks then display total marks
Step 5: Cal avg marks then display avg also

--> w.a.p to read group of numbers, then perform sum of even numbers in that group

--> w.a.p to declare passedhallticketnumbers array then check student is pass / fail

--> passedHalltickets[ ] ={ 101, 106, 124, 149, 157, 221, 254]

Enter your hallticket number : 124
You Passed in Exam

--> Take Array --> int passedHalltickets[ ] ={ 101, 106, 124, 149, 157, 221, 254];

step2 : Read Hallticket number from student -->int ht = sc.nextInt( ) ;

step3:- write for Loop

```
for(int a : passedHalltickets)
        {
            if(ht == a)

        }
```


Multi - D Arrays

--> If you have more than one pair of [ ] then it is called as Multi-D Array

--> If you to perform matrix operation

--> Array of an Array is called as Multi - D Array

Syntax:

```
Datatype arrayname[ ][ ] = new Datatype[size1][size2] ;
```

Ex:-

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

Ex:-

```
class ArrayEx7
{
    public static void main(String[ ] args)
    {
        int a[ ][ ] = new int[2][3];
        System.out.println(" Enter 6 Elements : ");
        for (int i=0; i < 2 ; i++ )
        {
            for( int j = 0; j < 3 ; j++)
            {
                a[i][j] = sc.nextInt( );
            }
        }
        System.out.println(" Your Matrix Elemetns are ");
        for (int i=0; i < 2 ; i++ )
        {
            for( int j = 0; j < 3 ; j++)
            {
                System.out.println(a[i][j] + "\t");
```

```
                    }
                }

        }
}
```

Jagged Arrays

--> Array of un-order Array is called as Jagged Arrays

Ex:-

```
import   java.util.Scanner;
class   JaggedArrayEx1
{
        public  static  void   main(String[ ]   args)
        {
                Scanner   sc  =  new   Scanner(System.in);
                int    a[ ][ ] = new   int[3][ ];
                a[0]   =   new   int[4] ;
                a[1]   =   new   int[2] ;
                a[2]   =   new   int[3] ;
                System.out.println(" Enter your Array Elements ");
                for(int  i=0; i < 3 ; i++)
                {
                        for( int   j = 0 ; j < a[i].length ; j++)
                        {
                                a[i][j]  =  sc.nextInt( );
                        }
                }
                System.out.pritnln(" Your Array Elements : ");
                 for(int  i=0; i < a.length ; i++)
                {
                        for( int   j = 0 ; j < a[i].length ; j++)
                        {
                                System.out.print(a[i][j]+"\t");
                        }
                        System.out.println();
                }
        }
}
```

String

--> Any thing which is enclosed in a double quotes is called as string
--> Group of characters is called as String
--> " Java"
--> " Venkat "
--> "a"
--> "A23B"

--> String is a pre-defined class, which is used to store group of characters
String s1 = " Venkat" ; (or) String s1 = new String("Venkat");
--> Strings are immutable objects , that means we can not change the value of String
--> Java Provides pre-defined methods to perform operations on Strings

Methods

1**) int length( ):** It will return the length of a string

```
String s1 = "Java";
int len = s1.length( );
System.out.println(" Length of String is : "+len);
```

2) **char charAt(int index):** It will return a character available at the specified index position

```
String s1 ="Core Java" ;
char ch = s1.charAt(5) ;
System.out.println(" 5th Character is : "+ch);
```

3) **String concat(String s2) :** It is used to join contents of one string to another string

```
String s1 = "Core ";
String s2 = "Java";
String s3 = s1.concat(s2) ;
System.out.println(" New String is : "+s3);
```

Ex:-

```
import java.util.Scanner;
class StringEx3_1
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print(" Enter First Name : ");
        String s1 = sc.next( );
```

```
                System.out.print(" Enter Last Name : ");
                String   s2 = sc.next( );
                 String   s3 = s1.concat(s2) ;
                 System.out.println("Full Name is : "+s3);
        }
}
```


4) boolean equals(String str) : It is used to compare the content of object by considering the case

```
                String   s1 = "Java" ;
                String   s2 = "java" ;
                boolean  b  =  s1.equals(s2) ;
                #if(s1.equals(s2))
                if(b)
                        System.out.println(" Both Strings are Same ");
                else
                        System.out.println(" Both Strings are not Same ");
```

Output :- Both Strings are not Same

5) boolean equalsIgnoreCase(String str) : It is used to compare the content of object by ignoring their case

```
                String   s1 = "Java" ;
                String   s2 = "java" ;
                boolean  b  =  s1.equalsIgnoreCase(s2) ;
        #   if(s1.equalsIgnoreCase(s2))
                if(b)
                        System.out.println(" Both Strings are Same ");
                else
                        System.out.println(" Both Strings are not Same ");
```

Output: - Both String are Same

6) boolean startsWith(String) : It will check whether a string begins with the specified group of characters or not

```
String  ht = "1121A16001" ;
if(ht.startsWith("1121"))
        System.out.println(" Student is belongs to Sathya Technology ");
else
        System.out.println(" Student is not belongs to Sathya Technology ");
```

7) boolean endsWith(string) : It will check whether a string ends with the specified group of characters or not

Ex:-

```
import   java.util.Scanner;
class StringEx5
{
        public static void main(String[] args)
        {
            Scanner   sc = new  Scanner(System.in);
            System.out.print("Enter your Mail id : ");
            String   email = sc.next();
            if(email.endsWith("ibm.com"))
                 System.out.println(" Person is belongs to IBM Company ");
            else
                System.out.println(" Person is not belongs to IBM Company ");
        }
}
```

8) **String toLowerCase( ) :** Used to convert content of a string into a Lowecase

```
String    s1 = "JAVA";
String   s2 = s1.toLowerCase( );
System.out.println(" Your String is : "+s2);
```

9) **String toUpperCase( ):** Used to convert content of string to Upper Case

```
String    s1 = "Java";
String   s2 = s1.toUpperCase( );
System.out.println(" Your String is : "+s2) ;
```

10) **String trim( ) :** This method can be used for removing leading and trailing spaces available in a string

```
String   s1 = "  Core  Java   " ;
String   s2 = s1.trim( );
System.out.println(" New String is : "+s2);
```

11) **String replace(char oldc, char newc):** It will replace all the occurrences of the specified character with another character

```
String    s1 ="Sathya Technology";
String   s2 = s1.replace('a', 'e');
```

```
class   StringEx6
{
        public static void main(String[] args)
        {
          String    s1 ="Sathya Technology";
                String   s2 = s1.replace('a', 'e');
                System.out.println("Your new String is : "+s2);
        }
}
```


12) **String replace(String olds, String news):** It will replace all the occurrences of the specified String with another String

```
        String   s1 = "c++ is imp and c++ is most powerfull language";
        String  s2 = s1.replace("c++","Java");
```

```
class  StringEx7
{
        public static void main(String[] args)
        {
                String   s1 = "c++ is imp and c++ is most powerfull language";
                String  s2 = s1.replace("c++","Java");
                System.out.println(" Your new String is : "+s2);
        }
}
```

13**) String[ ] split(String str) :** It is used to split the given string based on given character

```
                String  s1="Venkat Reddy Java and Python Developer";
                String[ ]   s2 =  s1.split(" ")
```

```
class  StringEx8
{
        public static void main(String[] args)
        {
            String  s1="Venkat Reddy Java and Python Developer";
                String[ ]   s2 =  s1.split(" ");
        //String[ ]  s2 = { "venkat", "Reddy", "Java", "and", "Python", "Developer"} ;
                for(String  x : s2)
                for(int   i=0;i<s2.length;i++)
```

```
                                    System.out.println(x);
                { S.o.p(s2[i]); }
        }
}
```

Task:

1) W.a.p to read string and display in reverse order
2) W.a.p to read Employee email id, then display only Employee name
   Ex:- Enter Email : Venkat@ibm.com
        USer Name is : Venkat
3) W.a.p to display group of related hallticet numbers

Ex:- String hallticket[ ] = ["1123213", "453553", "6633475", "1123563", "3456322", "1123636"];

Enter your college code : 1123
Yorr clg hallticket numbers :1123213
1123563
1123636

OOPS

--> OOPs stands for Object oritnted programming System.

--> As per object oritented analysis and design[OOAD] we have four principles, they are

1)Abstraction

2) Encapsulation

3) Inheritance

4) Polymorphism

--> Any Programming Language, if it follows above 4 principles then it is called OOPL

--> In OOP , we have to develop programs by using classes and objects

1) Abstraction:

--> Abstraction is a process of showing necessary data and hiding unneccessary data

2) Encapsulation:

--> Encapsulation is a process of binding related varialbes and methods in to a single container

--> Encapsulation is possible by using class

--> Class

Class is a collection of varialbes and methods

Where variable are used to store data

Methods are used to perform operations on data

3) Inheritance :

--> Getting properites from one class to another Class is called as inheritance

--> The class which Provides Properites is called as parent class

--> The class which uses properites is called as child class

4) Polymorphism

--> Poly means many and morph means behaviours

--> ploymorphism means many behaviours

Ex:-

"Core"	+	" Java"	==> Core Java
5	+	7	==> 12
"5"	+	"7"	==> 57

--> Overloading

--> Overriding

class

--> Class is a collection of variables and methods

Syntax:

```
class   classname
{
        variables
        methods
}
```

Application --> Modules --> Multiple classes --> multiple methods

Variables:

--> Variable is a named memory location, which is used to store data

--> In Java Variables are classified into 4 types

1) Local Variable
2) Parameters
3) Instance Variables
4) Static Variables

1) Local Variables

--> The Variables which are declared inside of method is called as Local variables

--> Local Variables can access only with in that method

2) Parameters

--> The variables which are declared at the time method declartion is called as Parameters

--> Parameters can access only with in that method

Ex:-

```
class   Transaction
{
        static  String  bname = "Hyd";    // Static variables
        int     acccno;                   // Instance  Variable
        int     amount;                   // Instance  Variable
        public  static  void  withdraw( )
        {
                int   wAmt = 5000 ;    // Local Variabless
                S.o.p(accno);
                S.o.p(Amount) ;   //   7000
                S.o.p(dAmt);              // Error, Because dAmt is local varialbe for deposit() so we can't use in withdraw
```

```
        }
        public  static   void    deposit( )
        {
                int   dAmt        = 2000 ;
                S.o.p(accno);
                S.o.p(amount);
                S.o.p(wAmt);    //Error , because wAmt is local variable for withdraw( )
                S.o.p(newpin) ;  // Error,  Because  newpin is parameter to pinchange( )

        }
        public  static  void  pinchange(int   newpin)   //  newpin is parameter
        {
                int  pinnum = newpin ;
                S.o.p(accno);
        }
        public  static  void  main(String[]   args)
        {
                 int  accno = 1001;     // Local variable
        }
}
```


3**) Instance Variables**

--> The Variables which are declared inside of a class and outside of a method is called as instance variables

--> Instance variables can access all the method of a class

--> If you want to store different data for different objects then use instance variables

4) Static Variables

--> Static variables are declared inside of a class and outside of a method with static keyword

--> Static variables can access all the methods of a class

--> If you want to store common data for all objects then use static variables

Methods:

--> A set of statements Writen inside of a block with name is called as Method

--> Methods are used to perform specific task

--> Methods are 2 types

* Pre-defined Methods --> Already available
* User-defined Methods --> Based on requirement we have to declare methods

Syntax:

```
returntype      Methodname(p1,p2,....)
{
        Statment -1 ;
        Statemnt -2 ;
        ..............
}
```

--> You can take any name as Methodname but you have follow 4 naming rules

1) No Space
2) No Special Character except _ and $
3) No Keywrods
4) First character should be alphabet / _/$

--> Every Method Will return some value, based on type of value we have to decide return type

If it returns integer value then return type is int
If it returns double value then return type is double
If it returns String value then return type is String
If it returns Integer array then return type is int[ ]
If it doesn't return any value then return type is void

Note:- Every Program Execution By default Starts from main( ) Method ,
Except main( ) method it never execute other methods
If you want to execute other methods except main( ) method then we have to call method
Which method you call that method only executed
--> We can call one method any number of times based on requirement

```
int a = sc.nextInt( );
int b = sc.nextInt() ;
int c = sc.nextInt( );
```

Ex:-

```
class   Test
{
        public  static   void   m1( )
        {
                System.out.println(" Test class  m1 method  ");
        }
        public  static void  main(String[ ]  args)
        {
                System.out.println(" Test class Main Method ");
        }
}
```

Ex2:-

```
        class   Test
{
        public  static   void   m1( )
        {
                System.out.println(" Test class  m1 method  ");
        }
        public  static void  main(String[ ]  args)
        {
                m1( );
                System.out.println(" Test class Main Method ");
                m1( );
        }
}
```

Ex3:-

```
        class   Test
        {
        public  static   void   m1( )
        {               System.out.println(" m1 method  ");        }

        public  static   void   m2( )
        {               System.out.println(" m2 method  ");        }

        public  static void  main(String[ ]  args)
        {
                m1( );
                System.out.println(" Main Method ");
                m1( )
        }
}
```

Output:

```
m1 method
main method
m1 method
```

Ex4:-

```
class   Test
{
 public  static   void   m1( )
{                 System.out.println(" m1 method  ");
                  m2( );
 }
 public  static   void   m2( )
{                 System.out.println(" m2 method  ");        }

public  static void  main(String[ ]  args)
{
          m1( );
          System.out.println(" Main Method ");
}
}
```

Output:

```
          m1 method
          m2 method
   main method
```

Ex5:-

```
class   Test
{
 public  static   void   m1( )
{                 System.out.println(" m1 method  ");
                  m2( );
                  System.out.println("Hai");
 }
 public  static   void   m2( )
{                 System.out.println(" m2 method  ");        }

public  static void  main(String[ ]  args)
{
          System.out.println(" Main Method ");
          m1( );
          System.out.println(" Thank you ");
```

```
        }
}
```

Output:-

```
Main Method
m1 method
m2 method
hai
Thank you
```


Ex5:-

```
class   Test
{
        public  static   void   m1( )
        {               System.out.println(" m1 method  ");
                        m2( );
                        System.out.println("Hai");
        }
        public  static   void   m2( )
        {               System.out.println(" m2 method  ");        }

        public  static void  main(String[ ]  args)
        {
                System.out.println(" Main Method ");
                System.out.println(" Thank you ");
        }
}
```

Output:

```
Main Method
Thank you
```

Methods with Parameters

Ex1:-

```
class   Test
{
        public  static   void   m1(int   x)
        {
           System.out.println(x);
        }
        public  static  void  main(String[]  args)
        {
            m1(5);
```

```
        }
}
```

Ex2:-

```
class  Test
{
        public  static  void  m1(int  x)
        {
           System.out.println(x);
        }
        public  static  void  main(String[]  args)
        {
                 int   a = sc.nextInt( );
            m1(a);
        }
}
```

Ex3:-

```
class  Test
{
        public  static  void  m1(int  x, double  y)
        {
           System.out.println(x);
           System.out.println(y);
        }
        public  static  void  main(String[]  args)
        {
                // m1(4, 2.6) ;
                int  a =  sc.nextInt( );
                double  b = sc.nextDouble( );
                m1(a, b) ;
                m1(a) ;   //  Error
        }
}
```

Ex4:-

```
class  Test
{
        public  static  void  m1(String  a,  int  b[ ])
        {
           System.out.println(a);
                for (int  i :b )
                {
```

```
                        System.out.println(i);
                }
        }
        public static void main(String[] args)
        {       String s1 = sc.next( );
                int[ ] x = new int[5];
                for(int i=0;i<5;i++)
                {
                        x[i] = sc.nectInt();
                }
                m1(s1, x) ;
        }
}
```

Methods with return type

Ex1:

```
public static int m1( )
{
        int a = 5, b= 7;
        int c = a + b;
        return c;
}
public static void main(String[] args)
{
        int x = m1( ) ;
        System.out.println(x);
}
```

Ex2:

```
public static double m1( )
{
        int a = 5;
        double b= 7.5;
        double c = a + b;
        return c;
}
public static void main(String[] args)
{
        double d = m1( ) ;
}
```

Ex3:

```
public static boolean login(String uname, String pwd)
{
        boolean b = true ;
         return b;
}
public static void main(String[] args)
{
        String un = sc.next( );
        String pw = sc.next( ) ;
        boolean status = login(un,pw);
}
```

--> Method Declarations are 4 types

--> Based on parameters and return type Methods are classified into 4 types

	Parameter		Return type
Methods --> void sum( )	with no	and	with no
Methods --> void sum(int a, int b)	with	and	no
Methods --> int sum( )	with no	and	with
Methods --> int sum(int a, int b)	with	and	with

--> Methods are 2 types

1) static methods

2) Instance methods (Non-static method)

1)**static Methods**

--> The methods which are declared by using static keyword then it is called as static methods

Ex: static void m1( )
{ ..... }

--> We can call static methods in two ways

1) Directly : If your method is available in same class then call directly

2) By using Class name : If the method is availble in another class then call by using class name

Ex:-

```
class  Test
{
        static  void  m1( )
          {   ..............  }
           void  m2 ( )
          {  m1( ) }
        public  static  void  main(String[]  args)
          {    m1( );                          }
}
```

Ex2:

```
class  Demo
{
        static  void  m3( )
          {    Test.m1( );   }
        void  m4( )
          {     Test.m1( );   }
}
```

2) Instance Methods

--> The methods which are declared without static keyword is called as Instance Methods

Ex:

```
void  m1( )
        {        .....................  }
```

--> We can call instance method by using object reference only

Ex:-

```
class   Test
{
         void  m1( )                  // Instance Method
        { ..................  }
         public  static  void  main(String[]  args)
        {
                    Test   t = new  Test( );
                    t. m1( );
        }
}
```

Ex:-

```
class  Demo
{
        void  m2( )                    # Instance Method
          { ..................... }
}
class    Test
{
          void   m1( )                 // Instance Method
         { ..................... }
         void   m3( )
        {   Test   t1 = new   Test( );
                t1. m1( );
        }
         public  static  void  main(String[]  args)
         {
            Test    t = new   Test( );      // Local  Refrence varriable
                t. m1( );
                t. m3( );
                Demo   d  =  new   Demo( );
                d.m2( );
        }
}
```

Note :- With in a class both are instance mehtods then we can call directly

Ex2:

```
class     Test
{
        void    m1( )
        {   ..................... }
        void   m2( )
        {
                m1( );
        }
        public  static  void  main(String[ ] args)
        {
                Test    t  =  new   Test( );
                t . m1( ) ;
                t . m2( ) ;
        }
}
```

	With in class		Other class	
	static method	instance	Static	Instance
Static -	Directly	Object Ref	classname	Object Ref
Instance -	Directly	Object Ref/ Directly	classname	Object Ref

Constructors

--> Constructor is a block, which is used to assign values to instance variables
--> Constructors are executed automatically, when we create object
--> Constructor name and class name must be same
--> For constructor no return type, even void also not allowed

Ex:-

```
class   Employee
{
          int   x ;                          // Instance  variables
         Employee( ) // Construcors
         {
                  x = 10 ;
         }
         void   show( )                     // Instance method
        {
                  System.out.println(" x value is : "+x);
        }
       public  static  void  main(String[ ]  args)
        {
                        Employee   e1 = new   Employee( );
                        e1.show( );
        }
}
```

--> Constructors are classified into two types, They are
1. default constructor
2. Parameterized constuctor

1. Default constructor

--> Constructor with no parameters are called as default / zero parameter constructor
--> Default constructors are used to initialize default values to instance variables

```
class  Customer
{
        int  bpoints;

        Customer( )
    {   bpoints = 50 ;  }
        void  update( )
    {
            bpoints = bpoints + 20 ;
    }
    void  show( )
        {     System.out.println(" Bouns Points are : "+bpoints); }
    public  static  void  main(String[]  args)
    {
        Customer   c1 = new  Customer( );
        Customer   c2 = new  Customer( );
        c1.update( );
        c1.show( );
        c2.show( );
    }
}
```


2. Parameterized constructor

--> Constructor with parameters is called as parameterized constructor

Ex:-

```
class  Employee
{
    int  eid;
    String  ename;
    double esal ;
    Employee(int  id, String  name,  double  sal)
    {
            eid  = id;
            ename = name ;
            esal = sal ;
    }
    void  show( )
    {
        System.out.println(" Employee  id is : "+eid);
        System.out.println(" Emp name is  : "+ename);
        System.out.println(" Emp Salary is  : "+esal);
    }
```

```
        public static void main(String[ ] args)
        {
                Employee e1 = new Employee(101,"Avinash",20000.0);
                Employee e2 =new Employee(102,"vishnu",300000.0);
                e1.show( );
                e2.show( );
        }
}
```


Ex2:-

```
class Employee
{
        int eid;
        String ename;
        double esal ;
        int deptno;
        Employee( )
        {
                deptno = 10;
        }
        Employee(int id, String name, double sal)
        {
                        this( );
                        eid = id;
                        ename = name ;
                        esal = sal ;
        }
        void show( )
        {
                System.out.println(" Employee id is : "+eid);
                System.out.println(" Emp name is : "+ename);
                System.out.println(" Emp Salary is : "+esal);
                System.out.println(" Dept number : "+deptno);
        }
        public static void main(String[ ] args)
        {
                Employee e1 = new Employee(101,"Avinash",20000.0);
                Employee e2 =new Employee(102,"vishnu",300000.0);
                e1.show( );
                e2.show( );
        }
}
```

Ex2:-

```
class   Employee
{
        int   eid;
        String   ename;
        double esal ;
        int  deptno;
        Employee( )
        {
                this(101,"Vishnu",25000.0);
                 deptno = 10;
        }
        Employee(int  id, String  name,   double   sal)
        {
                        eid  = id;
                        ename = name ;
                        esal = sal ;
        }
        void   show( )
        {
                System.out.println(" Employee  id is : "+eid);
                System.out.println(" Emp name is   : "+ename);
                System.out.println(" Emp Salary is  : "+esal);
                System.out.println(" Dept number   : "+deptno);
        }
        public  static  void  main(String[ ]   args)
        {
                 Employee  e1 = new  Employee( );
                 e1.show( );
         }
}
```


Ex:-

```
class   Test1
{
         int   x;                                        # Instance  Variable
         Test1( )                                        # default Constructor
        {                  x = 25;          }
        void  show( )                      # Instance Method
        {
                int   x = 50 ;             # Local  Variables
                 System.out.println(" In show x value is : "+x);
```

```
        }
        void   display( )
        {        System.out.println(" In display x value is : "+x);    }
        public static  void   main(String[]  args)
        {
                        Test1    t = new   Test1( );
                        t.show();
                        t.display();
        }

}

Ex2:- class    Test1
{
         int   x;                                          //Instance  Variable
         Test1( )                                          // default Constructor
        {                  x = 25;          }
        void  show( )                     // Instance Method
        {
                  int   x = 50 ;              // Local  Variables
                   System.out.println(" In show x value is : "+x);
                   System.out.println(" In show Instance x value is : "+this.x);
        }
        void   display( )
        {        System.out.println(" In display x value is : "+x);

        }
        public static  void   main(String[]  args)
        {
                        Test1    t = new   Test1( );
                        t.show();
                        t.display();
        }

}
```

this:

this keyword is used to access instance members of the current class

this keyword can be specified either in constructors or instance methods of the current class

this keyword cannot be specified in static methods

this( ):

this( ) is used to access Default constructor of the current class

this( ) can be specified in any parameterized constructor

this( ) cannot be specified in methods(either instance or static)

this(...)

this(...) is used to access parameterized constructor of the current class

this(...) can be specified either in default constructor or Parameterized constructor of the current class

this(...) cannot be specified in methods(either instance or static)

Example on static and instance variables

--> static varibles can execute at the time of class loading

--> If we have common data then we have to use static varaibles

Ex:-

```
class   Employee
{
          static   int   ccode = 2255 ;
          static   long  cph =8074864650 ;
          int   empid ;
          String  name;
          Employee(int  empid,  String   name)
         {
                        this.empid = empid ;
                        this.cph = cph;
         }
         void   show( )
         {
                  System.out.println(" Company code is : "+ccode);
                  System.out.println(" Company Ph no   : "+cph);
                  System.out.println(" Emp id  is      : "+empid);
                  System.out.println(" Emp Name : "+name);
         }
}
```

W.a.p to read Emp salary, and no.of leaves, then calculate monthly salary of Employees

--> We need Constructors
--> cal leavesAmt
--> cal Monthlysal
--> display

```
class Employee
{
        double salary,lAmt,msal;
        int leaves;
        Employee(double salary, int leaves)
        {
                this.salary = salary ;
                this.leaves = leaves ;
        }
        void calLeavesAmt()
        {
            lAmt = salary/30 * leaves ;
        }
        void calMonthlySal( )
        {
                 msal = salary - lAmt ;
        }
        void display( )
        {
                System.out.println(" Employee salary is : "+salary);
                System.out.println(" No.of Leaves          : "+leaves);
                System.out.println(" Leaves Amount         : "+lAmt);
                System.out.println(" Monthly Salary   : "+msal);
        }
        public static void main(String[] args)
        {
                Employee e1 = new Employee(42500, 3);
                e1.calLeavesAmt( );
                e1.calMonthlySal( );
                e1.display();
        }
}
```

Inheritance:

--> Getting properties from one class to another class is called as inheritance
--> Inheritance is used for Code reusability
--> The class which provides properties is called as super class / Parent Class
--> The class which uses properties is called as Sub Class / child class
--> To provide link between one class to another class we have to use **extends** keyword
--> Inheritance is also called as IS-A Relationship

Syntax:-

```
class   calssname
{
        variables ;
        methods ;
}
class   childclassname  extends parentclassname
{
        variables ;
        methods ;
}
```

Note:

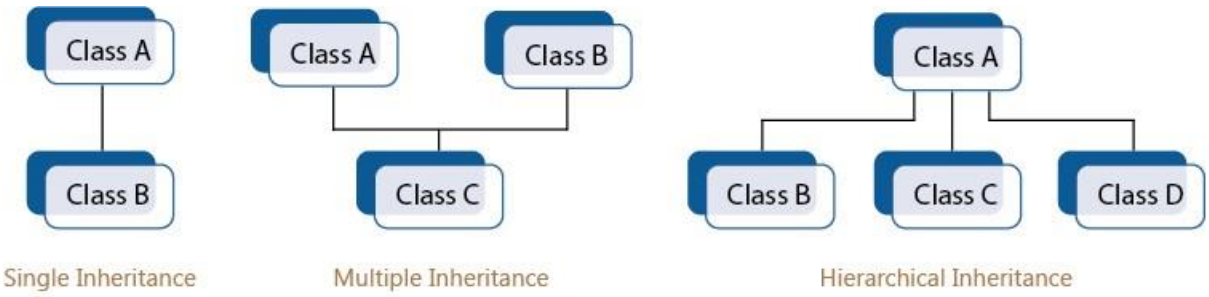
--> If you create object for Parent class we can access only parent class members
--> If you create object for child class we can access parent class and child class members
--> In inheritance always create object for child class

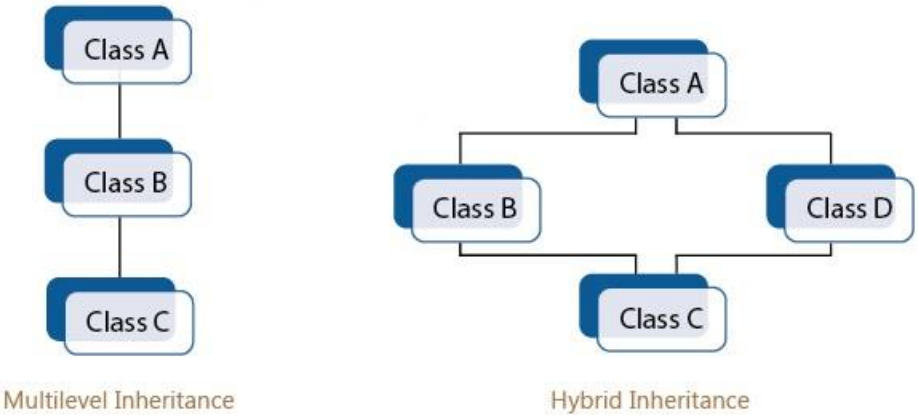
Inheritance are classified into different types based on no.of parent and no.of child classes

1. Single inheritance --> one parent class and one child Class
2. Multi-level inheritance --> one Class will act as parent and child
3. hierarchical inheritance --> one parent Class and more than one child class
4. multiple inheritance --> More than one parent Class and one child class
5. hybrid inheritance --> Combination of more than one inheritance

Note:- In Java Multiple inheritance is not possible at class level
Multiple inheritance is possible by using **interface**

--> Whenever We create object for super class then memory is allocated to super class Members only
--> Whenever we create object for sub Class then memory is allocated for both super class and sub Class





Ex1:

```
 class  Person
 {
            String name ;
            char  gender;
            int   age;
 }
class  Student extends Person
{
            int  sid;
            double fee;
         public static void main(String[ ]  args)
        {
            Student   s = new Student( ) ;
        }
}
```

Ex2:

```
 class  Person
 {
            String  name ;
            char  gender;
            int   age;
 }
class  Student  extends  Person
{
         int  sid;
         double  fee;
}

class   Demo
{
         public  static  void  main(String[]  args)
        {
            Student  s1 =  new  Student( );
            Student  s2 =  new  Student( );
        }
}
```

Single Inheritance

```
 class  Person
 {
        String  name="Vishnu";
        char  gender='M';
        int   age=31;
        void  show( )
         {
            System.out.println(" Person class show method ");
         }
 }
class  Student  extends  Person
{
          int  sid=1001;
         double  fee=20000.0;
         void   display( )
        {
             System.out.println("  Name  is  : " +name);
             System.out.println(" Gender is  : "+gender);
```

```
                System.out.println(" Age  is   : "+ age);
                System.out.println(" Student id   : "+sid);
                System.out.println(" Course  fee  : "+fee);
        }
        public  static  void   main(String[ ]  args)
        {
                Student   s1 =  new   Student( );
                s1.show( );        //  By using child class reference we are calling parent
                                       class method
               s1.display( );      //  By using child class reference we are calling child
                                       class method1
        }
}
```


Ex2:-

```
class   A
{
         A( )
        {                System.out.println(" Parent class Constructor ");          }
        void   show( )
        {                System.out.println(" Parent class show method ");                  }
}
class   B  extends   A
{
        B( )
        {                System.out.println(" Child class Constructor ");           }
        void   display( )
        {                System.out.println(" Child class display method ");                 }
         public  static  void  main(String[ ]  args)
        {
                B   b  =  new   B( );
                b.show( );
                b.display( );
        }
}
```

Output:

```
                Parent class Constructor
                Child class Constructor
                Parent class show method
                Child class display method
```

Ex2:-

```
class   A
{
        A( )
        {                System.out.println(" Parent class Constructor ");          }
   void  show( )
        {                System.out.println(" Parent class show method ");                  }
}
class  B  extends   A
{
        B(int  a )
        {                System.out.println(" Child class Param - Constructor ");          }
   void  display( )
        {                System.out.println(" Child class display method ");                }
        public  static  void  main(String[ ]  args)
        {
                B  b  =  new  B(10);
                b.show( );
                b.display( );
        }
}
```

Output:

Parent class Constructor
Child class Param - Constructor
Parent class show method
Child class display method

Ex3:

```
class   A
{
        A( )
        {                System.out.println(" Parent class Constructor ");          }
        A(int  x)
        {          System.out.println(" Parent class Constructor "); }
        void  show( )
        {                System.out.println(" Parent class show method ");                  }
}
class  B  extends   A
{
        B(int  a )
        {                System.out.println(" Child class Param - Constructor ");          }
        void  display( )
```

```
        {               System.out.println(" Child class display method ");          }
        public static void main(String[ ] args)
        {
                B b = new B(10);
                b.show( );
                b.display( );
        }
}
```

Output:

```
        Parent class Constructor
        Child class Param - Constructor
        Parent class show method
        Child class display method
```

Ex4:

```
class   A
{
        A( )
        {               System.out.println(" Parent class Constructor ");       }
        A(int x)
        {           System.out.println(" Parent class Param-Constructor "); }
 }
class  B extends   A
{
        B(int  a )
        {       super(a) ;
                System.out.println(" Child class Param - Constructor ");        }

        public static void main(String[ ] args)
        {
                B b = new B(10);
        }
}
```

Output:

```
        Parent class param-constructor
        Child class param-constructor
```

Ex5:

```
class   A
{
        A( )
        {                   System.out.println(" Parent class Constructor ");          }
        A(int  x)
        {          this();
                   System.out.println(" Parent class Param-Constructor "); }
  }
class  B  extends   A
{
        B(int   a )
        {         super(a) ;
                  System.out.println(" Child class Param - Constructor ");          }

        public  static  void  main(String[ ]   args)
        {
                   B   b  =  new   B(10);
        }
}
```

Output:

```
        Parent class Constructor
        Parent class Param-Constructor
        Child class Param - Constructor
```

Ex:- Accessing super class instance variable by using super keyword

```
class   Test
{
        int   x = 10 ;
}
class   Demo   extends   Test
{
      int   x = 20;
      void   show( )
      {
         int   x = 30;
         System.out.println("Local variable  x  value : "+x);                  // 30
         System.out.println("instance variable x value : "+ this.x);          // 20
         System.out.println("Super class  x  value  : "+super.x);              //  10
     }
```

```
        public static void main(String[]  args)
        {
                        Demo   d = new  Demo( );
                        d.show() ;
        }
}
```


this

---

this :- It is used to access current class instance variables

this( ) :- It is used to access current class default constructor

this(...) :- It is used to access current class parameterized constructor

super

---

super :- It is used to access super class instacne variables

super( ):- It is used to access super class default constructor

super(...) : It is used to access super class Parameterized constructor

Overloading

--> Writing morethan one method with same name but different parameters is called as Overloading

--> In Overloading based on number of parameters and type of parameters the method is executed

--> We have two types of Overloading

1) Method Overloading

2) Constructor Overloading

--> To Perfrom over loading one class is enough

Method Overloading

Ex1:

```
// W.a. p to perfrom addition of two numbers
class   AddEx1
{
         void   add(int  a,  int  b)
        {
                int   c = a + b ;
                     System.out.println("Integer Sum is : "+c);
```

```
        }
        void  add(double  a,  double  b)
        {
                double  c = a + b ;
                        System.out.println(" Double data type Sum is : "+c);
        }
        void  add(int  a,  int  b,  double  c)
        {
                double   p =  a + b + c ;
                System.out.println(" 3 parameters sum is : "+p);
        }
        public  static  void  main(String[]  args)
        {
                AddEx1   d =  new  AddEx1( );
               d.add(6,9);
               d.add(6.8, 4.2);
               d.add(5, 9 ,2.5);
        }
}

Ex2:
class  AreaEx1
{
        void  area(double  r)
        {
                double  a =  3.14 * r * r ;
                System.out.println(" Area of Circle is : "+a);
        }
        void   area(double  l, double  b)
        {
               double  a = l * b;
               System.out.println(" Area  of  reactangle is : "+a);
        }
        public  static  void  main(String[]  args)
        {
                //create object and call methods
        }
}
```

Ex3:-

```
class   Transaction
{
   void   payAmt(String   upino)
        {
                System.out.println(" Pay Amount through UPI ");
        }
}
class  Order extends  Transaction
{
         void   payAmt(long   phno)
        {
                 System.out.println(" Pay Amount through  Google Pay ");
        }
         void   payAmt(int   cid,  String   pwd)
        {
                System.out.println(" Pay Amount through net banking : ")
        }
        public  static  void  main(String[]  args)
        {
                        Order    d =  new   Order( );
                        d.payAmt(112121,"nikendhku");
                        d.payAmt("sathyatech@upi");

        }
}
```

Method Overriding

--> Writing of more than one method with same name and sam parameters is called as Method overriding

--> To perform Method overriding min 2 classes are required

--> Those two classes should be in IS-A Relation

--> In Method Overriding , when you call method it will execute child class method only

Ex:

```
class  Test
{
        void  show( )
        {
                System. out. println(" Test class show method ");
        }
}
class  Demo  extends  Test
{
        void  show( )
        {
                System. out. println(" Demo class show method ");
        }
        public  static  void  main(String[ ]  args)
        {
                Demo  d = new Demo( );
                d.show( );
        }
}
```

Different ways to create object

```
class  Test
{
        void  show( )
        {
                System.out.prinltn("Test class show ");
        }
}
class  Demo  extends  Test
{
        void  show( )
        {
                System.out.println(" Demo class  show ");
        }
}
```

Polymorphism : - Many Forms

--> One Entity shows behaviors is called as polymorphism

--> Polymorphism are two types, they are

1) Static Polymorphism : Ex:- Overloading
2) Dynamic Polymorphism : Ex:- Overriding

Relationship

We can provide relations in two ways

1) IS- A Relationship :- class to class
2) HAS-A Relationship :- class to Object

Ex:-

```
class A
{
    void show( )
    { ..... }
}
class B extends A
{

}
```

--> Whenever we create object for B Class , then memory is allocated for Class A also

2) HAS-A Relationship

--> Relation between class and object

```
class  Payment
{
        void  payAmt( )
       {  ..............  }
}
class  Transation
{
        void  login( )
       {  ..............  }
        void  search( )
       {  ..............  }
  void  buyProduct( )
       {   // Pay amount
             Payment  p = new  Payment( );
             p. payAmt( );
       }
       void  display( )
       {  ..........  }
}
```


Abstract Class

Concrete method: If a method has both declaration and definition(body) then it is called as concrete method

```
Ex:-   void  show( )
       {
            ..................
       }
```

Concrete class: If a class contains all concrete methods then it is called as concrete class

Abstract method: If a method has only declaration no body then it is called as abstract method

We can declare abstract methods by using abstract keyword

Syntax:

```
abstract  returntype  methodname( p1/p2/...) ;
```

Ex: abstract void show( ) ;

Abstract class:

If a class contains abstract methods then it is called as abstract class

Abstract class can be combination of abstract methods and non-abstract methods

We can create abstract class without abstract method also

If a class contains at least one abstract method then, declaring the class as abstract is mandatory.

Syntax:

```
abstract  class  className
{
        ……….
}
```

Ex:-

```
abstract class  Test
{
        abstract  void  show( );
}
```

Ex2:-

```
abstract class  Test
{
        void  display( )
        {       .....    }
        abstract void  show( ) ;
}
```

Note: For abstract class we **cannot create object**. So if we want to access abstract class members we have to inherit abstract class into another class and override all abstract methods available in abstract class

Syntax:

```
abstract class classname
{
        ……..
}
class  subclassname  extends  abstracclassname
{
        ……..
}
```

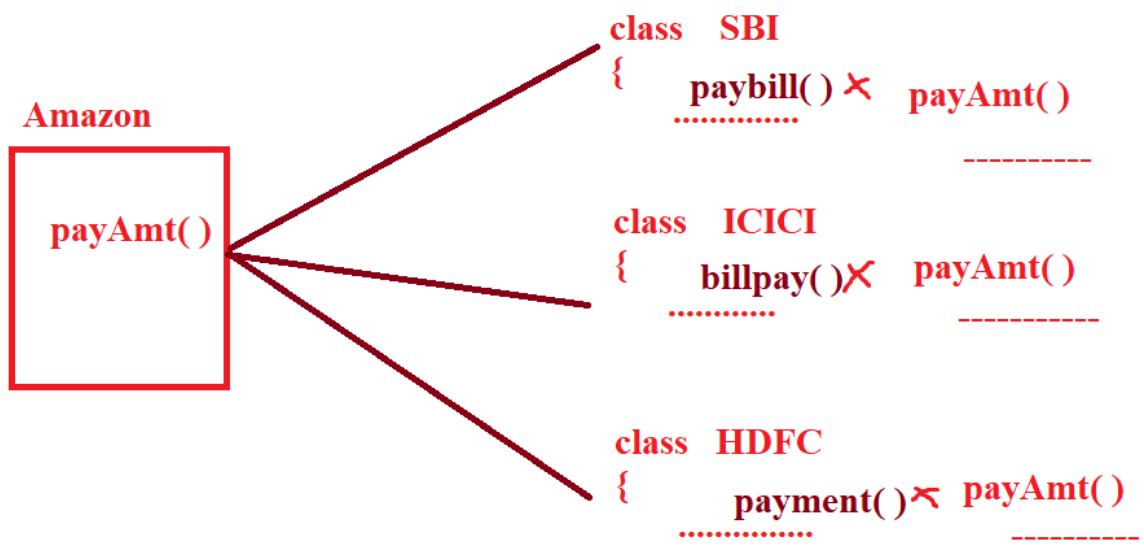
Ex1:-
```
abstract class Test
{
        abstract void show( );
}
class Demo extends Test
{
        void show( )
        { ..... }
}
```

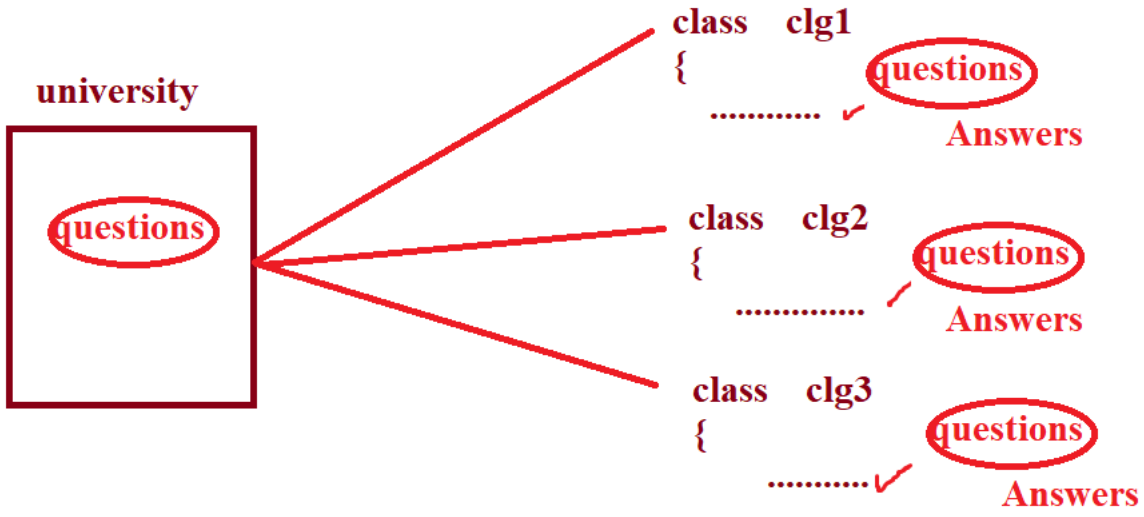
Ex2:-
```
abstract class Test
{
        abstract void show( );
        abstract void display( );
}
abstract class Demo extends Test
{
        void show( )
        { ..... }
}
class Sample extends Demo
{
        void display( )
        { ..... }
}
```

Object Creation:

```
Sample s = new Sample( );
Test t = new Sample( );
s.display( );
t.display( ) ;
s.show( );
t.show( );
```

Sub class should override all abstract methods of abstract class,
otherwise subclass also we have to change as abstract class
We cannot create object for abstract class, but we can create a reference for
abstract class then it will refer to an object of any of its child class which is concrete



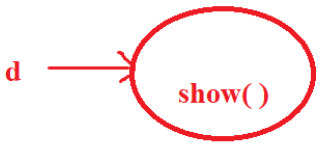


Different ways to create object

```
class      Demo
{
     void   show( )
     {  ...Hai.....  }
}
class   Test   extends   Demo
{
     void   show( )
     {  ...Hello.....  }
}
```

1) Parent class object parent class reference

```
Demo   d  =  new  Demo( );
d.show( );
```



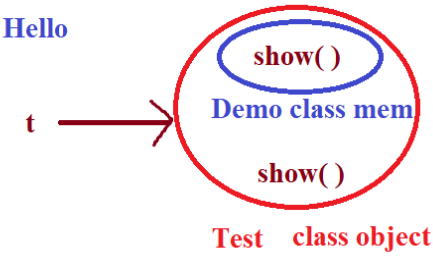
Demo class object

```
class      Demo
{
     void   show( )
     {  ...Hai.....  }
}
class   Test   extends   Demo
{
     void   show( )
     {  ...Hello.....  }
}
```

2) child class object and child class Reference

```
Test   t  =  new   Test ( ) ;
     t .show( ) ;
```

Output: Hello

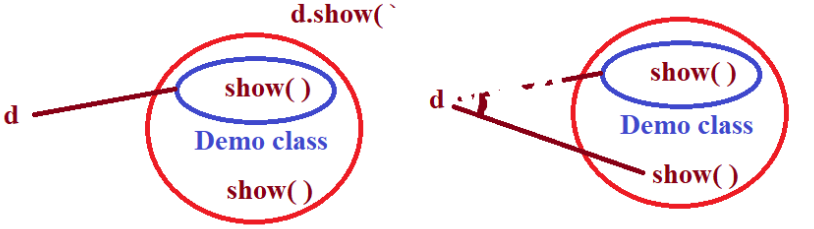


Test class object

```
class      Demo
{
     void   show( )
     {  ...Hai.....  }
}
class   Test   extends   Demo
{
     void   show( )
     {  ...Hello.....  }
}
```

3) Child class object and parent class reference

```
Demo   d  =  new   Test( ) ;
     d.show( `
```



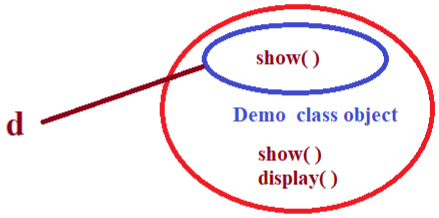
Test class object

Test class object

Note:- At the time of compile time it will point reference type(parenttype), but at the time of execution it will refer child class method....

--> dynamically swictch between parent to child , it is dynamic polymorphism

```
class  Demo
{
     void  show()
     { System.out.println(" Parent Demo class show method ");  }

}
class  Test3  extends  Demo
{
     void  show( )   // Overriding
     { System.out.println(" Child - Test class show method " ); }
     void   display()
     {  System.out.println(" Child - Test class  dislplay method " ); }

     public static void main(String[] args)
     {
     // child class object and parent class reference
          Demo  d = new  Test3( );
          d.display() ;
     }
}
```



Test3 class object

Compile Time Error

--> Parent class doesn't have any display( ) method

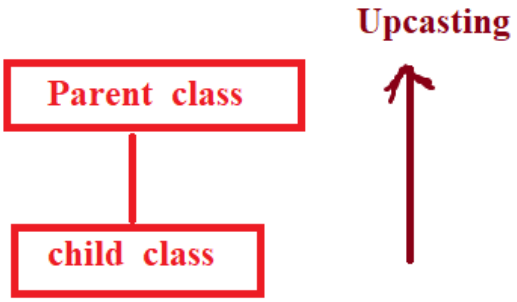
```
abstract  class  Draw
{
    abstract  void  shap( ) ;
}
```

```
class  Circle  extends  Draw
{
    void  shap( )
    {
       ..................
    }
}
```

```
class  Reactangle  extends  Draw
{
    void  shap( )
    {
       ..............
    }
}
```

```
Circle  c = new  Circle( ) ;
  c. shap( ) ;

Reactangle  r = new Reactangle( ) ;
   r.shap( ) ;
```

```
Draw  d = new  ..... ;

      d.shap() ;
```


```
abstract  class  Paymemt
{
    abstract  void  payAmount( ) ;
}
```

```
class  GooglePay  extends  Payment
{
    void  payAmount( )
    {
       ..................
    }
}
```

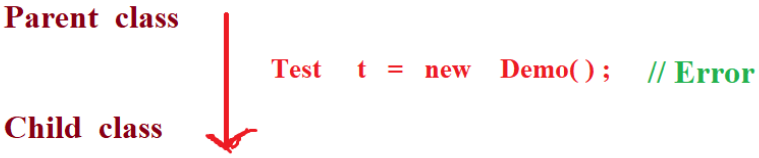
```
class  UPI  extends  Payment
{
    void  payAmount( )
    {
       ..........................
    }
}
```

```
class  Amazon
{
    void  payBill( )
    {
        Payment  p = new ..... ;
         p.payAmount( ) ;
    }
}
```

```
class      Demo
{
     void   show( )
     {  ..Hai.....  }
}
class   Test   extends   Demo
{
     void   show( )
     {  ...Hello.....  }
}
```

4) Parent class Object and child class reference

```
Test   t  =  new   Demo( ) ;    // Error
  t. show( );
```



Note :- **We can not do downcasting directly, first we have to do upcasting then only downcasitng is possible**

VENKATREDDY

Ex1:-

```
abstract class   Test
{
        abstract   void   show( );
}
```

Ex2:

```
abstract  class   Test
{
        abstract   void   show( );
        public   static  void  main( )
       {
              Test   t = new  Test( );
       }
}
```

Error:- error: Test is abstract; cannot be instantiated
Test t = new Test( );

Ex3:-

```
        abstract class  Test
        {
            abstract  void  show( );
         }
        class  Demo  extends  Test
        {
          void  show( )
         {  System.out.println(" Demo class  show  method ") ; }
          public  static  void  main(String[]  args)
        {
                  Demo   d = new  Demo( );
                  d.show( );
        }
}

Ex4:-
abstract  class   Test
{
         abstract   void   show( );

}
class   Demo  extends   Test
{
          void  show( )
         {  System.out.println(" Demo class  show  method ") ; }
          public  static  void  main(String[]  args)
        {
                Test   t  = new  Demo( );   // Upcasting
                 t.show( );
        }
}

Ex5:-
abstract  class   Test
{
         abstract   void   show( );
         abstract   void   display( );

}
abstract class   Demo  extends   Test
{
          void  show( )
         {  System.out.println(" Demo class  show  method ") ; }
```

```
}
class  Sample  extends  Demo
{
        void  display( )
       {                System.out.println(" Display method in Sample class ") ;        }
        public  static  void  main(String[]  args)
       {
                Test   t  = new  Sample( );   // Upcasting
                 t.show( );
                 t.display( );
       }
}
```

final keyword:

---

final	Variables	-->	We can not change values of variables
	Methods	-->	We can not Override
	classes	-->	We can not inherit

Ex1:

```
final   double   PI  = 3.14 ;
PI = 30.21 ;   // Error , because  a is final variable
```

Ex2:-

```
class   Test
{
        final  void  show( )
       {       .............          }
}
class  Demo  extends  Test
{
        void  show( )                 // Override
       {       ...............      }
}
```

Ex3:-

```
final  class   Test
{
          void   show( )
         {        .....          }
}
class   Demo  extends   Test          //  Error   , class is final , so no inheritance
{
          void  show( )                                        //  Override
         {        .....          }
}
```


Access Specifies : - private, default, protected, public

private :- private is keyword, Private members can access only with in that class

Default :- if you are not specifying any keyword , by default the access specified is default with in package

protected : protected is keyword,
protected members can access with in package and other package child class

public : public is keyword,
public members can access in all the packages

interface

Interface is a collection of abstract methods

Syntax:

```
interface   interfaceName
{
        Variables
        Methods
}
```

Ex:

```
interface   Demo
{
        int  a = 20;                    //  public  static  final   int   a = 20 ;
        void  show( );                  //  public   abstract  void  show( );
}
```

- The variables of an interface are declared as public, static, and final whether we specify or not
- The methods of an interface are declared as public and abstract whether we specify or not

Note:

--> For interface we cannot create object, so to access interface we have to inherit into another class by using implements keyword

Ex:1

```
class   Demo
{
}
class  Test   extends   Demo
{
}
```

Ex2:

```
interface   Demo
{
    void  show( );
}
class  Test  implements  Demo
{
        void  show( )
        {  ..............  }
}
```

--> If a class is implementing an interface then that class must provide implementation to all the methods available in that interface

Ex3:-

```
interface   Demo
{
        void  show( );
        void  display( );
}
abstract class  Test  implements  Demo
{
        void  show( )
        {  ..............  }
}
class  Sample  extends  Test
{
        void  display( )
        {  ..................  }
}
```

--> If subclass doesn't provide implementation to at least one abstract method then declare the subclass as abstract

Ex3:-

```
interface   Demo
{
        void  show( );
        void  display( );
}
class  Test  implements  Demo      //  Test class is called as  implementation class
{
        void  show( )
        {  ..............  }
        void  display( )
        {  ..............  }
}
```

--> If sub class is providing implementation to all the abstract methods then, the subclass is called as implementation class

Ex3:-

```
        interface   Demo
        {
                int  a = 30 ;
                void  show( );
        }
  class  Test  implements  Demo      //  Test class is called as  implementation class
 {
        void  show( )
        {  System.out.println(a) ;  }  // Demo.a
}
class   Sample  implements  Demo   //  Sample class is called as  implementation class
{
      void  show( )
      {          .....  }
}
```

Demo

Test Sample

--> An interface can have any number of implementation classes

--> The variables of an interface must be initialized. The variables of an interface can be accessed by using interface name or can be accessed directly by implementing the interface

--> For interface we cannot create an object but we can create a reference variable and it refer to an object of its implementation class

--> By using interface we can support multiple inheritance

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

--> Parent class Reference = child class object

```
class   SBI
{
        void  payAmt( )
          {  ..............  }
}
class   ICICI
{
         void   billPay( )
        { ..................  }
}
class  KVB
{
         void   payment( )
        { ..................  }
}


interface    Bank
{
          void   payAmt( ) ;
}
class   SBI  implements  Bank
{
        void  payAmt()
        { . .........  }
}
class   ICICI   implements  Bank
{
         void   payAmt( )
        { ............  }
}
class   KVB  implements   Bank
{
         void   payAmt( )
        { . ..................  }
}
```

Ex1:

```
interface  Test
{
        void show( );   // --> public  abstract  void  show( ) ;
}
class Demo implements Test
{
        public void  show( )                        // Overriding
        {
                System.out.println(" Demo class show ");
        }
        public static void main(String[] args)
        {
                Test  t = new  Demo( );                 //  Up Casting
                t.show( );
        }
}

//      private         default             protected                   public
```

Ex2:-

```
interface  Test
{
        void show( );   // --> public  abstract  void  show( ) ;
}
class Demo implements Test
{
        public void  show( )                        // Overriding
        {
                System.out.println(" Demo class show ");
        }
        public  void  display( )
        {
                System.out.println(" Demo class display ");
        }
        public static void main(String[] args)
        {
                Test  t = new  Demo( );      //  Up Casting
                t.show( );
                t.display( );                // Error
        }
}
```

Rules for inheriting a class and interface:

==========

--> A class can extend at most one class

--> A class can implement any number of interfaces

--> An interface can extends any number of interfaces

--> A class can extends a class and implement any number of interfaces

Ex1:

```
class   Test
{       }
class  Demo   extends   Test
{
}
```

Ex2:

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

Ex3:-

```
interface   A
{                       }
interface        B
{                       }
interface  C  extends  A,B
{                       }
```

Ex4:-

```
interface   A
{                       }
interface       B
 {                      }
class      Test
 {                      }
class  Demo   extends   Test   implements  A,B
{
}
```

Ex5:-

```
interface   A
{                       }
interface       B
{                       }
class       Test
{                       }
class   Demo  implements  A,B   extends   Test    // Error
{
}
```

Marked Interface:

--> An interface is said to be marked or tagged if it is empty.

```
interface   Demo
{
}
```

-->The purpose of a marked interface is to provide some instructions to the JVM to perform a task in a special way

Predefined Marked interfaces:

Cloneable , serializabel, RandomAcess etc..

Note:

We can create user defined marked interface but they are of no use because the jvm has no meaning for user defined marked interface.

Packages

Package:

- ➔ Package is a collection of related classes
- ➔ Packages are used for code separation
- ➔ Packages are declared by using "**package**" keyword
- ➔ Every package should have at least one public class, and the public class name and file name should be same.

➔ Packages are classified into two types, they are

i) Pre-defined Package

ii) User-defined Package

1) Pre-Defined Package :

➔ The Packages which are provided by your java is called as pre-defined packages

Package Name	Description
java.lang	Contains language support classes (for e.g classes which defines primitive data types, math operations, etc.) . This package is automatically imported.
java.io	Contains classes for supporting input / output operations.
java.util	Contains utility classes which implement data structures like Linked List, Hash Table, Dictionary, etc and support for Date / Time operations.

2) User-defined packages :

Based on Application requirements , programmer is created packages, they are called as user-defined packages

Syntax:-

```
package  packagename;
public class classname
{
        Vaariables;
        methods
}
```

Ex:

```
package  p1;
public class  Emp1
{
    public void show1()
      {
          System.out.println(" Emp1 Show Method ");
      }
}
```

Now Save your program with Emp1.java and then compile your program by using following command

Syntax:

Javac –d DirectoryName filename

Ex:

Javac –d . emp1.java

Here " . " indicates that current Directory

If you want to create package in another directory then use following command

Ex:- javac -d D:\Java_4pm\Interface Index.java

If you want to add one more class in same package, then

```
Package  p1;
public class Emp2
{
        public void show2()
        {
                System.out.println(" Emp2 show method ");
        }
}
```


Accessing packages:

To Access packages we have to import related package in our file by using "**import**" keyword

Syntax:

```
import     packagename.classname;
```

Here classname indicates that importing class

If we want to import all classes of a package then use following syntax

Syntax: **import packagename.*;**

Here " * " indicates that all classes of that package will loaded

Ex:

```
import pack.*;
public class Emp3
{
        public static void main(String[] args)
        {
                pack.Emp1 e1=new pack.Emp1();
                e1.show1();
                System.out.println(" ..... ");
                pack.Emp2 e2=new pack.Emp2();
                e2.show2();
        }
}
```

➔ If we specify * that means all classes of that packages is imported, so here at the time of creating object we have to specify packagename.classname

Ex2:-

```
import  pack.Emp1;
import   pack.Em2;
public class Emp3
{
public static void main(String[] args)
        {
                Emp1 e1=new Emp1();
                e1.show1();
                System.out.println(" ..... ");
                Emp2 e2=new Emp2();
                e2.show2();
        }
```

}

➔ Here are importing specific classes , so we can access classes directly by using class name


```
import  p1.Emp1;
class  Emp3
{
        public  static  void  main(String[ ]  args)
        {
                System.out.println(" Emp3 class main method ");
                Emp1   e1 = new  Emp1( );
                e1.display( );
                System.out.println(" Thank you ");
        }
}
```

Ex2:-

```
import  p1.Emp1;
import  java.util.Scanner;
class  Emp3
{
        public  static  void  main(String[ ]  args)
        {
                System.out.println(" Emp3 class main method ");
                Scanner   sc  = new  Scanner(System.in);
                int  a = sc.nextInt( );
                Emp1   e1 = new  Emp1( );
                e1.display( );
                System.out.println(" Thank you ");
        }
}
```

Access Specifiers

---

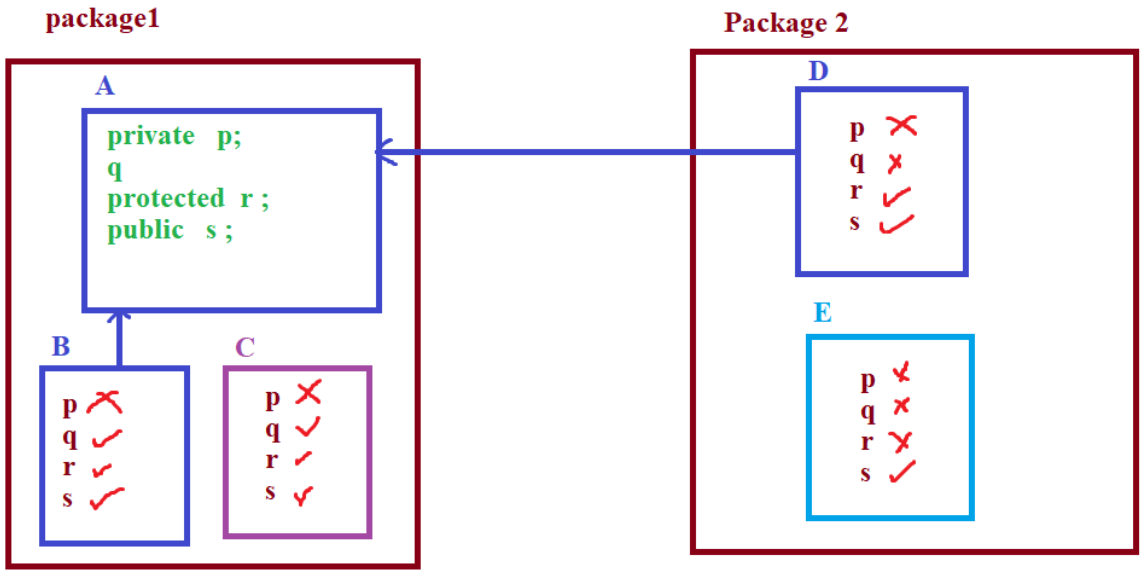
private, default, protected, public

private :- private is keyword, Private members can access only with in that class

default :- if you are not specifying any keyword , by default the access specifier is default with in package

protected : protected is keyword,
protected members can access with in package and other package child class

public : public is keyword,
public members can access in all the packages



	public	private	protected	< unspecified > default
class	allowed	not allowed	not allowed	allowed
constructor	allowed	allowed	allowed	allowed
variable	allowed	allowed	allowed	allowed
method	allowed	allowed	allowed	allowed

	class	subclass	package	outside Sub Class	outside Class
private	allowed	not allowed	not allowed	not allowed	not allowed
< unspecified >	allowed	allowed	allowed	not allowed	not allowed
protected	allowed	allowed	allowed	allowed	not allowed
public	allowed	allowed	allowed	allowed	allowed

Command line arguments

➔ At the time of Program execution we can pass input values in command prompt they are called command line arguments

Ex1:-

```
class   Test
{
        public  static  void   main(String[ ]  args)
        {
                String   fname =  args[0] ;
                String   lname =  args[1] ;
                String   fullname  = fname + lname;
                System.out.println(" Full Name  is  : "+fullname);
        }
}
```

Compile :- javac Test.java
Execute :- java Test "core" " Java"

Ex2:-

```
class   Test
{
        public  static  void  main(String[]  args)
        {
                int   a  =  args[0];
                int   b  =  args[1];
                int   c  =   a + b;
```

```
                System.out.println(" Sum is  : "+c );
        }
}
```

--> Error Occured , Because our input is String[] , but we are trying to convert in int type

--> String is Reference type
--> int is primitive type
--> We have to convert Reference type to primitive type
--> To Do this one java provides a concept called wrapper classes

Wrapper Class

Primitive types	Wrapper classes
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Boxing:

--> It is a process of converting a value from primitive type to Wrapper type
--> This Process is done automatically from java 1.5v onwards so it is called as auto boxing

```
Ex:     int     x  =  25;
        Integer   y  =  x;
```

Unboxing:

--> It is the process of converting a value from wrapper type to primitive type
--> This process is done automatically from java 1.5v onwards so it is called as auto unboxing

```
Ex:     Integer  x  =  new  Integer(20);
            int  y  =  x ;
```

To Convert from String type to primitive type java provides a method called as "parse"

Ex1:-

```
String  x = "10";
int     y  =  Integer.parseInt(x);
```

Ex2:

```
String  a = "3.14";
double  d = Double.parseDouble(a);
```


Ex3:-

```
class  Test
{
        public  static  void  main(String[]  args)
        {
                int   a  =  Integer.parseInt(args[0]);
                int   b  =  Integer.parseInt(args[1]);
                int   c  =   a + b;
                System.out.println(" Sum is  : "+c );
                System.out.println(" Thank you ");
        }
}
```

Generics:

Generics are called as parameterized types. It indicates General types, that means it can handle any type of data

The generics are represented by a pair of angular brackets(< >), called as diamond operator

Generics are designed to provide compile time type safety, which will reduce the need for typecasting

Syntax:

```
<E> returntype   MethodName( E  p1, E  p2  ,....)
   {                        }
```

Here E represents the generic type parameter

Ex:-

```
class Test2
{
        void  display(int[]  a)
        {
                 for( int  x : a)
                 {
                         System.out.print(x+"\t") ;
                 }
        }
  void   display(double[]  a)
        {
                 for( double  x : a)
                 {
                         System.out.print(x+"\t") ;
                 }
        }
        void   display(char[]  a)
        {
                 for(char  x : a)
                 {
                         System.out.print(x+"\t") ;
                 }
        }
        public static void main(String[] args)
        {
                 Test2   t1  =  new  Test2( );
                 int[]  p = {5, 8, 3, 9, 6} ;
                 t1.display(p);
                 System.out.println();
                 double[]  q  = {4.6, 3.5, 3.9,7.2};
                 t1.display(q);
                 char[]   r ={'v','e','n','k','a','t'};
        }
}
```

Ex:

```
class  Test3
{
        public <K> void display(K[ ] a)     // Generic Function
        {
                 for(K  x : a)
                                 System.out.print(x+"\t");
        }
```

```
        public static void main(String[] args)
        {
                Test3   t1 = new  Test3( );
                Integer[ ]  x ={4,7,3,6,3};
                t1.display(x);
                System.out.println();
                String[ ]   s ={"venkat","java","python"};
                t1.display(s);
//      double[ ]   d={2.4,4.5,6.8};  //Error-> Generic concept we have to pass
                //only class type data
                System.out.println();
                Double[]  d = {3.5, 8.2, 5.7, 7.4};
                t1.display(d);
        }
}
```


Errors:-

--> The mistakes in our program is called as Errors

--> Errors are classified into 3 types

1) **Syntax Errors :**

--> Every Progrmaming language provides some rules, as a programmer we have to follow those rules

Ex:- System.out.pritnln(" Sathya Technology) ;

--> When you compile above code, now we are going to get Error, they are called as syntax errors

--> In Your Program if you have syntax Errors then program never executed

2) **Logical Errors**:

--> While writing code , if you Do any mistakes they are called as logical Errors

--> We can not identify logical errors at the time compilation,

--> By checking your program output then only we can identify logical errors

```
Ex:-        amount  =  5000
            wAmt  =  2000
            amount  =  amount + wAmt      //  Logical  Errors
```

3) **Runtime Errors:**

--> The Erros which occur at runtime(at the time of program execution) they are called as runtime errors

--> When ever Runtime Error occured your program is terminated without executing remaining lines of code

--> The Run time Errors are also called as Exceptions

Exceptions

--> Runt time errors are called as Exceptions

--> Whenever Exceptions occured your program is terminated without executing remaining lines of code

Ex1:-

```
// W.a.p to perform division of two numbers
import java.util.Scanner;
class ExceptionEx1
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                System.out.print(" Enter x value : ");
                int x = sc.nextInt( );
                System.out.print(" Enter y value : ");
                int y = sc.nextInt( );
                int z = x / y;
                System.out.println(" Division is : "+z);
                int p = x + y ;
                System.out.println(" Addition is : "+p);
                System.out.println(" Thank you ");
        }
}
```

Output1: -

```
Enter x value : 10
Enter y value : 5
Diviision is : 2
Addition is : 15
Thank you
```

Output2:

```
Enter x value : 25
Enter y value : 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionEx1.main(ExceptionEx1.java:11)
Then program is terminated
```

--> When Ever Exception Occured Jvm is created Exception realated object, then it will check for Exception handling code, if it is not availble then program is terminated

--> To Continue Program Execution we have to handle Exception

--> In Java Exception handling is possible by using following keywords

1) try
2) catch
3) finally
4) throws
5) throw

Note: By using Try and Catch block we can handle exceptions

1) try

--> As a progammer we have to identify problematic code in our program then it will place inside of try block

--> Problemaic code means, the code which gives Exception

--> Try block contains problematic code

2) catch( )

--> Exception handling code we have to place inside of catch block

--> Whenever Exception occurred then only catch block is executed.

Syntax:-

```
try
{
        Problematic code
}
catch (ExceptionClassname   Referencename)
{
        // Print Exception details
}
```

Ex:

```
try
{
        z  = x / y ;
        System.out.println(" Division is : "+z);
}
catch (ArithmeticException   e1)
{
        System.out.println(e1)
}
```

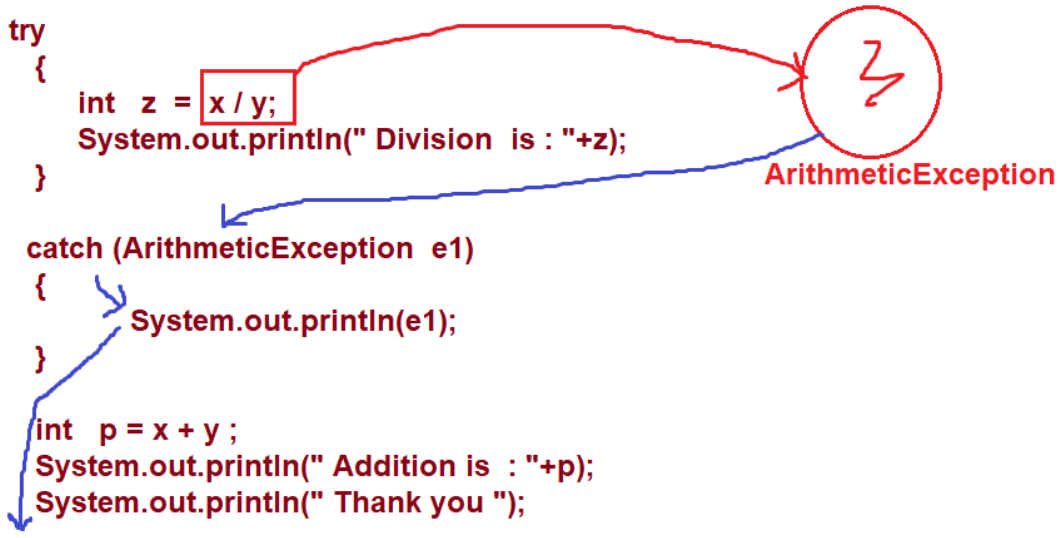
Note :- Handling of Exception means, display Exception details then continue program

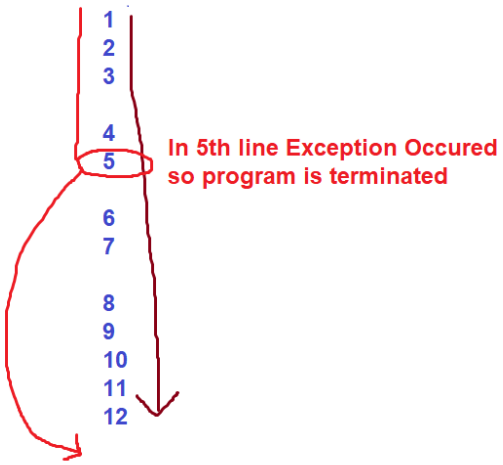
Ex1:-

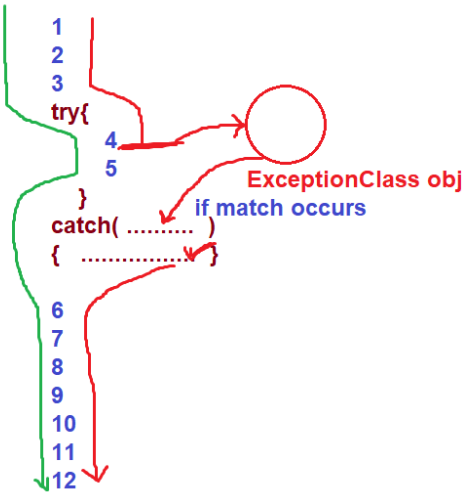
```
import  java.util.Scanner;
class   ExceptionEx2
{
        public  static  void   main(String[]   args)
        {
                Scanner   sc   =  new  Scanner(System.in);
                System.out.print(" Enter  x value : ");
                int   x = sc.nextInt( );
                System.out.print(" Enter  y value : ");
                int   y = sc.nextInt( );
                try
                {
                                int   z  =  x / y;
                        System.out.println(" Division  is : "+z);
                }
                catch (ArithmeticException  e1)   // WhenEver Exception occured then only it will execute catch block
                {
                        System.out.println(e1);
                }

                int   p = x + y ;
                System.out.println(" Addition is  : "+p);
                System.out.println(" Thank you ");
        }
}
```

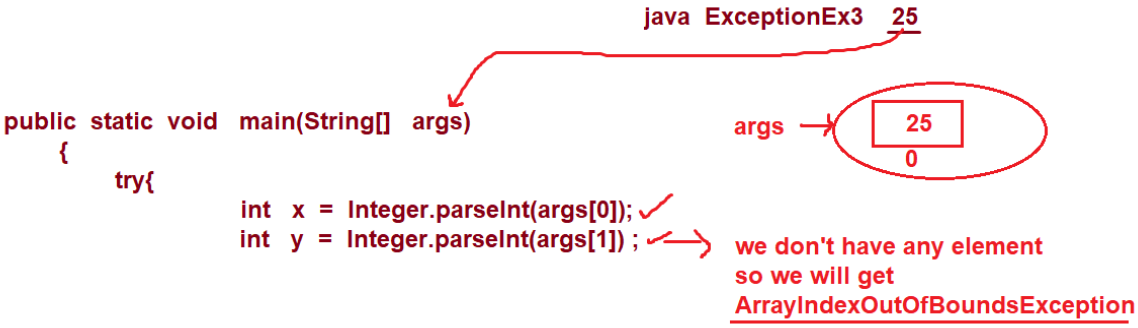
Output:

```
Enter  x value : 25
Enter  y value : 0
java.lang.ArithmeticException: / by zero
Addition is  : 25
Thank you
```









Multiple Catch blocks

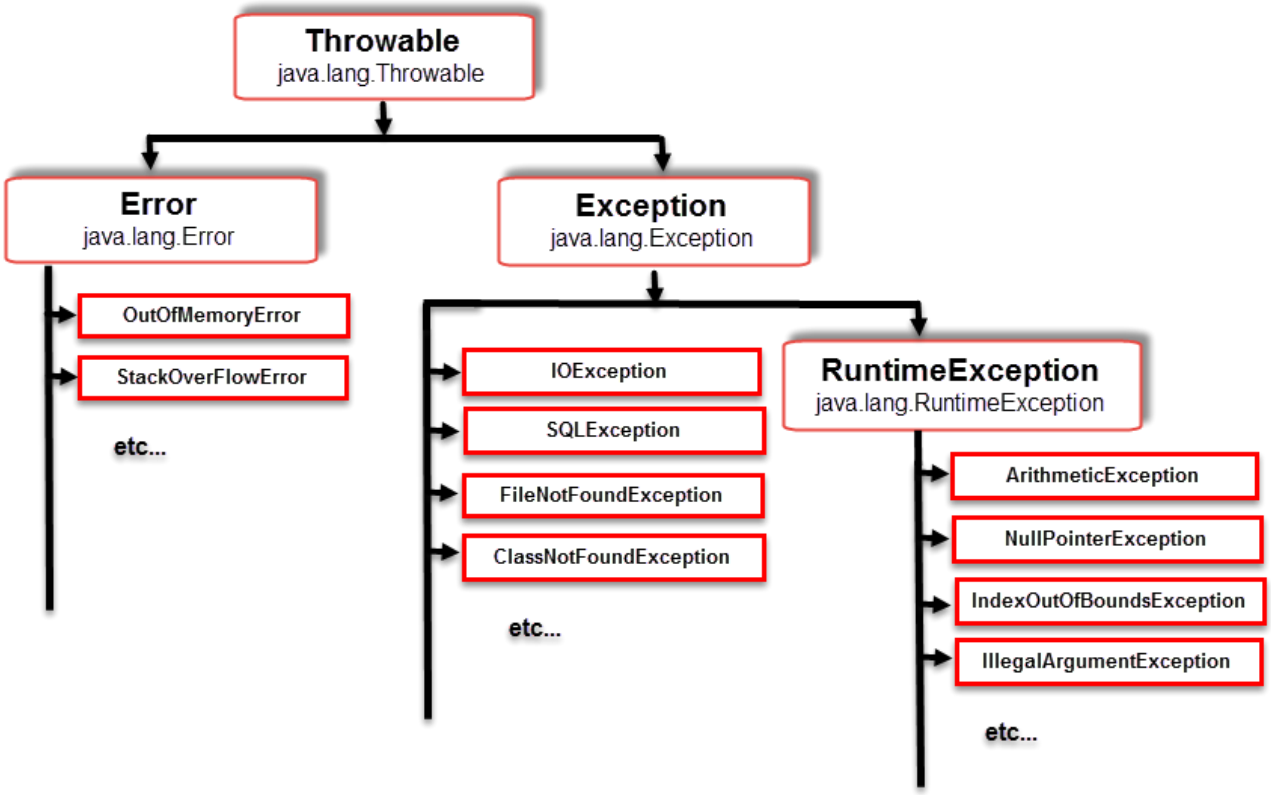
--> For one Try block we can write multiple catch blocks

--> When Ever Exeption occured then exception object is created by jvm, then it check for matching catch block

When matching catch occurs then it will execute that catch block, otherwise program is terminated

```
class   ExceptionEx3
{
  public  static  void   main(String[]   args)
 {
        try{
                int  x  =  Integer.parseInt(args[0]);
                int  y  =  Integer.parseInt(args[1]) ;
                int  z  =  x / y;
               System.out.println(" Division  is : "+z);
          }
        catch (ArithmeticException  e1)   // WhenEver Exception occured then only it will execute catch block
        {
                System.out.println(e1);
        }
        catch(ArrayIndexOutOfBoundsException   e2)
        {
                System.out.println(e2) ;
        }
        System.out.println(" Thank you ");
 }
}
```

Exception Hierarchy



Note: RuntimeExceptions and its subclasses, Errors and its sub classes are unchecked and remaining are checked Exceptions

Whenever exception occurs program is terminated without executing remaining lines of code, To avoid it and execute remaining lines of code we have to use Exception handling Technique.

Exception Handling means it doesn't remove the exception, it will execute remaining lines of code even exception occurs

```
class   ExceptionEx3
{
        public  static  void   main(String[]   args)
        {
                try{
                                int   x  =  Integer.parseInt(args[0]);
                                int   y  =  Integer.parseInt(args[1]) ;
                                int   z  =  x / y;
                               System.out.println(" Division  is : "+z);
                  }
                catch (ArithmeticException  e1)   // WhenEver Exception occured then only it will execute catch block
```

```
                {
                        System.out.println(e1);
                }
                catch(ArrayIndexOutOfBoundsException   e2)
                {
                        System.out.println(e2) ;
                }
                catch(Exception  e3)
                {
                        System.out.println(e3) ;
                }
                System.out.println(" Thank you ");
        }
}
```


Ex:

```
class   ExceptionEx4
{
        public  static  void   main(String[]   args)
        {
                try{
                        int   x  =  Integer.parseInt(args[0]);
                        int   y  =  Integer.parseInt(args[1]) ;
                        int   z  =  x / y;
                       System.out.println(" Division  is : "+z);
                  }
                     catch(Exception  e3)
                {
                        System.out.println(e3) ;
                }
                catch (ArithmeticException  e1)    // WhenEver Exception occured then
only it will execute catch block
                {
                        System.out.println(e1);
                }
                catch(ArrayIndexOutOfBoundsException   e2)
                {
                        System.out.println(e2) ;
                }

                System.out.println(" Thank you ");
        }
}
```

Note:- Whenever we are going for Multiple catch blocks then parent Catch block should write at end only

Different Ways to display Exception

1) **printStackTrace( ):**
   It is used to display detailed information of the exception
   It will display Exception name, reason, line number, method name, class name and program name
2) **getMessage( ):**
   it is used to display only the reason
3) **User defined Message:**
   We can display Our own messages by using S.o.p ( ) statement
4) D**isplay reference name :**
   We can display Exception information by using reference name

Ex:-

```
class   ExceptionEx5
{
        public  static  void   main(String[]   args)
        {
                try{
                                int   x  =  Integer.parseInt(args[0]);
                                int   y  =  Integer.parseInt(args[1]) ;
                                int   z  =  x / y;
                               System.out.println(" Division  is : "+z);
                  }
                catch(ArithmeticException  e1)
                      {                 System.out.println(" Y value should not be Zero ");   }
                catch(ArrayIndexOutOfBoundsException   e2)
                {     System.out.println(" You should pass  2 values  ");    }
                catch(NumberFormatException   e3)
                {   System.out.println("  x and y values should be integer only ");   }
                catch(Exception  e4)
                {
                          //e4.printStackTrace( ) ;
                         //  System.out.println(e4);
                          // System.out.println(e4.getMessage( ));
                           System.out.println(" Exception Occred in Program ");
                }
                System.out.println(" Thank you ");
        }
}
```

Nested Try Block

--> A Try block inside of another try

Syntax:-

```
try      // Outer Try block
{
        Statement - 1
        Statement - 2
        try       // Inner Try Block
        {
                Statement - 3
        }
        catch (Exceptionclassname   ref)   // Inner catch Block
        {  .....                          }
}
catch (Exceptionclassname   ref )     // Outer catch Block
{
                ..............
}
```

Note:- If Exception occur at outer level then outer catch block is executed

If Exception occur at inner level then first it checks Inner catch block If match occurs then it will execute Inner catch block

otherwise it will check outer catch block If match occurs then it will execute Outer catch block, otherwise program is terminated

Ex:

```
class   ExceptionEx6
{
    public  static  void   main(String[]   args)
    {
        try{                                          // Outer Try-Block
            int  x = Integer.parseInt(args[0]);
            int  y = Integer.parseInt(args[1]) ;
             try{                    // Inner Try - Block
                    int  z = x / y;
                     System.out.println(" Division  is : "+z);
                }
              catch(ArithmeticException   e1)
              {   System.out.println(" y value should not be Zero ");  }
        }
       catch(ArrayIndexOutOfBoundsException   e2)
      {    System.out.println(" You should pass  2 values  ");   }
```

```
                catch(NumberFormatException   e3)
                {   System.out.println("  x and y values should be integer only ");   }

                System.out.println(" Thank you ");
        }
}
```


Note:- If Exception occur at outer level then outer catch block is executed
If Exception occur at inner level then first it checks Inner catch block If match occurs then it will
execute Inner catch block otherwise it will check outer catch block If match occurs then it will execute
Outer catch block, otherwise program is terminated

Ex7:-

```
class   ExceptionEx7
{
        public  static  void   main(String[]   args)
        {
                try{                                            // Outer Try-Block
                        int   x  =  Integer.parseInt(args[0]);
                        int   y  =  Integer.parseInt(args[1]) ;
                        try{                                    // Inner Try - Block
                                int   z  =  x / y;
                                System.out.println(" Division  is : "+z);
                            }
                        catch(ArrayIndexOutOfBoundsException   e2)
                        {    System.out.println(" You should pass  2 values  ");   }
                  }
                catch(ArithmeticException   e1)
                        {   System.out.println(" y value should not be Zero ");   }
                catch(NumberFormatException   e3)
                        {   System.out.println("  x and y values should be integer only ");   }

                        System.out.println(" Thank you ");
        }
}
```

finally

--> Finally is a block which will execte all the times, that means whether exception occurs or not occur all the times finally block will be executed

--> finally block contains resource close logic

→ when ever your application is communicate with file / database then we have to provide connections once your program is finished then we have to close that connection , connection close logic we have to place inside of finally block

Syntax:-

```
try{
        Statement-1 ;
        Statement - 2;
 }
catch( Exception name   ref)
        {       .....                   }
finally{
         resource close code
        }
```

Syntax2:-

```
try{
        Statement - 1 ;
        Statement -2 ;
    }
finally
   {
        resource close code
    }
```

throws

--> throws keyword is used to transfer the exception object to its caller.

If you don't want to handle exception, then use throws

If you use throws keyword jvm will handle the Exception in main method

Syntax:

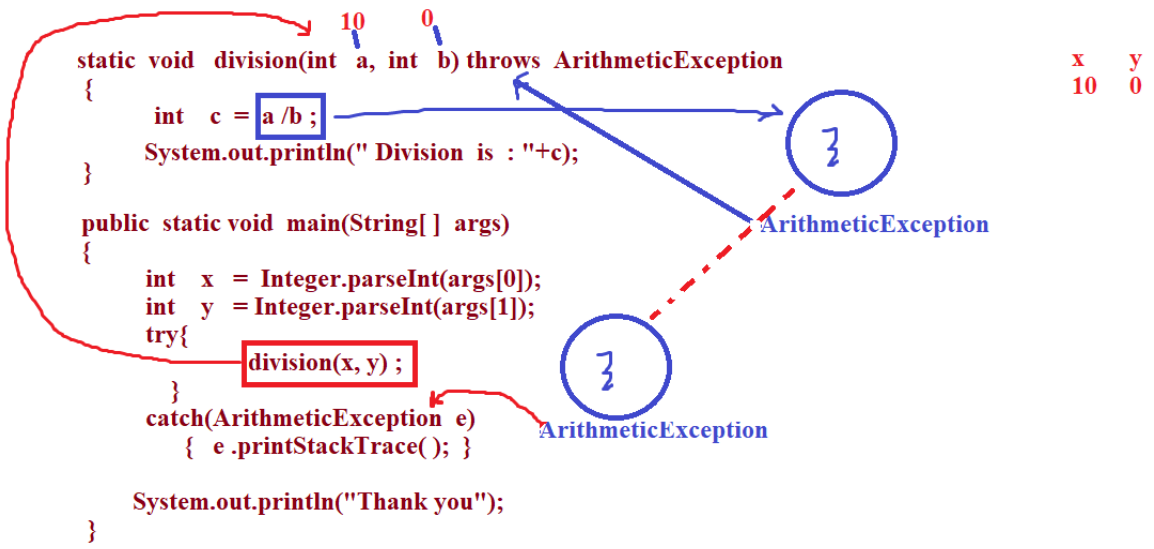
```
returntype   methodname(p1,p2,.....) throws  Exception1,Exception2,...
{
   Statements  ;
}
```

Ex:

```
class   ExceptonEx7
{
        public  static void  main(string[ ]  args) throws Exception
        {
                int   x  =  Integer.parseInt(args[0]);
                int   y  = Integer.parseInt(args[1]);
                int  z  =  x / y
               System.out.println(" Division is : " + z);
        }
}
```

Ex2:-

```
class   ExceptonEx7
{
        static  void   division(int   a,  int   b) throws  ArithmeticException
        {
                 int   c  =  a /b ;
                System.out.println(" Division  is  : "+c);
        }
        public  static void  main(String[ ]  args)
        {
                int   x  =  Integer.parseInt(args[0]);
                int   y  = Integer.parseInt(args[1]);
                try{
                                division(x, y) ;
                   }
                catch(ArithmeticException  e)
                        {   e .printStackTrace( );  }

               System.out.println("Thank you");
        }
}
```



throw

---

throw keyword is used to throw an exception object

Syntax:

throw ExceptionObject;

Ex:-

throw new Exceptionclassname( );

User -defined Exceptions

We can create our own Exception by extends Exception class

➔ All the exceptions are child class of Exception class, so if you want to create user-defiend Exception class then your class also should be child class of Exception class

Every User defined Exception class must have two constructors,

1) Default constructor
2) Parameterized Constructor taking one parameter as String type

super --> call supercalss instance variables
super( ) --> call superclass default constructor
super(...) --> call superclass parameter constructor

Ex:

```
class LessAgeException extends Exception
{
    LessAgeException()
    {}
```

```
        LessAgeException(String s)
        {
                super(s);
        }
}

public class  Test1
{
        public static void main(String[] args)
        {
             int   age = Integer.parseInt(args[0]);
             try
             {
               if(age<22)
                    throw new LessAgeException(" Your Age is Small to Apply this Job ");
             }
            catch (LessAgeException  le)
            {
                 le.printStackTrace();
            }
            System.out.println("Thank you ");
        }
}
```


Exceptions are two types, they are

1)**Checked Exception**

The Exceptions which occurs at run time but we should handle at compile time only is called as checked Exceptions

These are also called as compile time Exceptions

Handling of these exceptions is mediatory

Ex:- FileNotFoundException, ClassNotFoundException, SQLException etc…

2)**Un-Checked Exception**

The Exceptions which occurs at runtime and handling of these exceptions is optional.

These are also called as run-time Exceptions

Ex:- ArithmeticException, ArrayIndexOutOfBoundsException etc…

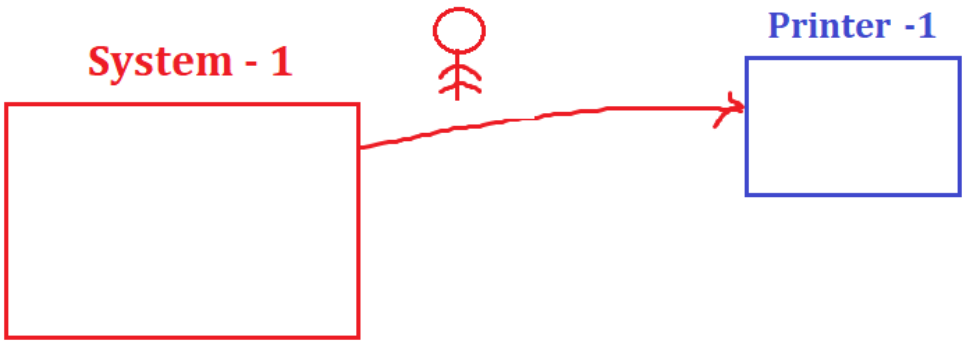
Thread Concept

==========

```
class   Test
{
        public  static  void  main(String[]  args)
        {
                int   x = Integer.parseInt(args[0])
                int   y =Integer.parseInt(args[1])
                int  z = x / y ;
                System.out.println(" z value is : "+z);
        }
}
```

--> A Thread is a separate piece of code which will be executed separately.

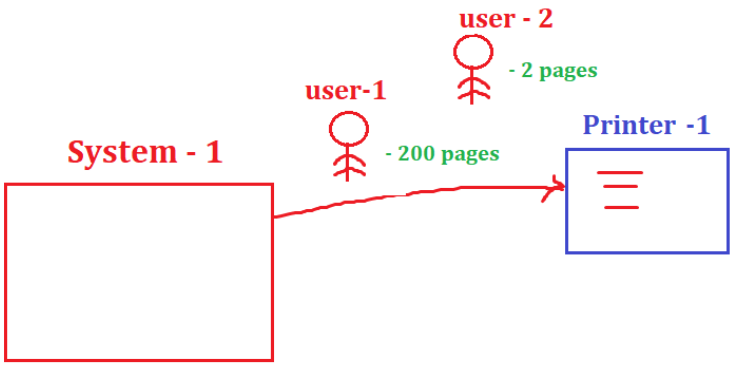
--> Every java Program by default contains one thread called as main thread

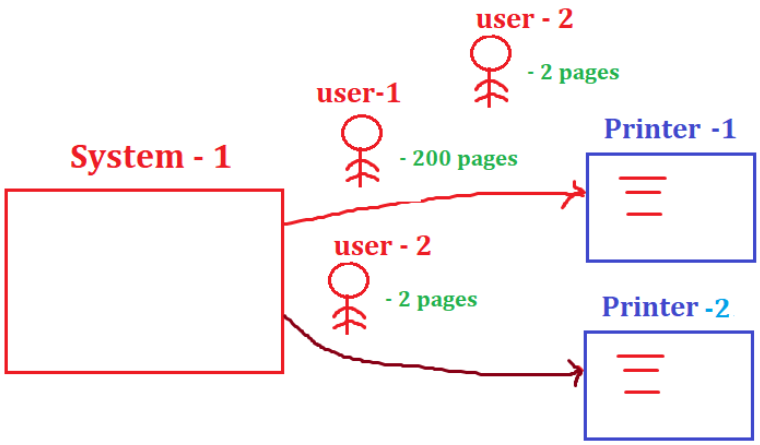



```
class   PrintDocument
{
          void   printPages( )
         {
              for(int  i =1;  i <= 10 ; i++)
                 {   System.out.println(" From  Printer -1  page# ",i);  }
         }
              public  static  void  main(String[]   args)
         {
                      System.out.println(" Main Method ");
                       printDocument   p  = new   printDocument( );
                            p.printPages( );
         }
}
```

Note :- In Our Program one Thread(Main) is created, then it will execute your code

For the seak second user , they arrange second printer

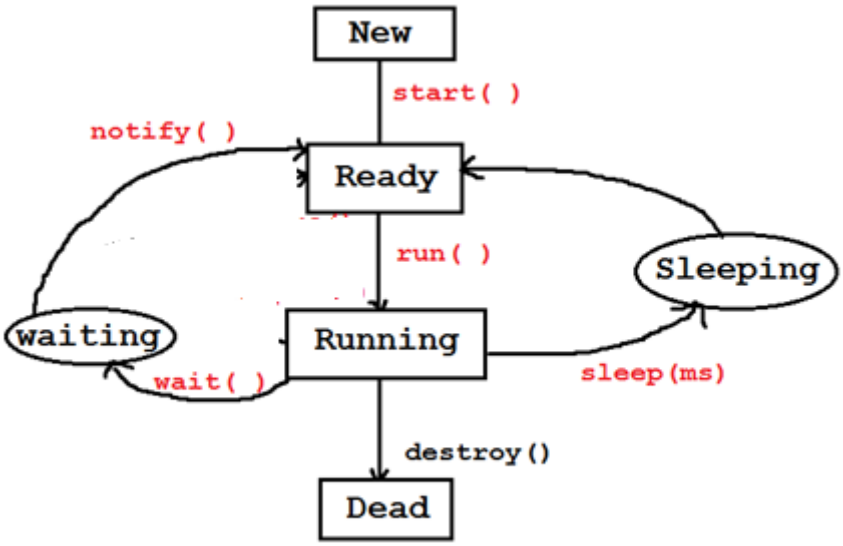





```
class  PrintDocument
{
          void  printPages( )
         {
                    for(int  i =1;  i <= 10 ; i++)
               {   System.out.println(" From  Printer -1  page# "+i);  }
         }
              public  static  void  main(String[]  args)
          {
                     System.out.println(" Main Method ");

                          PrintDocument   p  = new   PrintDocument( );
                          p.printPages( );

                    for(int  i =1;  i <= 10 ; i++)
                          {   System.out.println(" From  Printer -2  page# "+i);  }
          }
}
```

Note:- In Java by default program is executed sequentially
but if you want to execute parallel then we have to use multiple threads

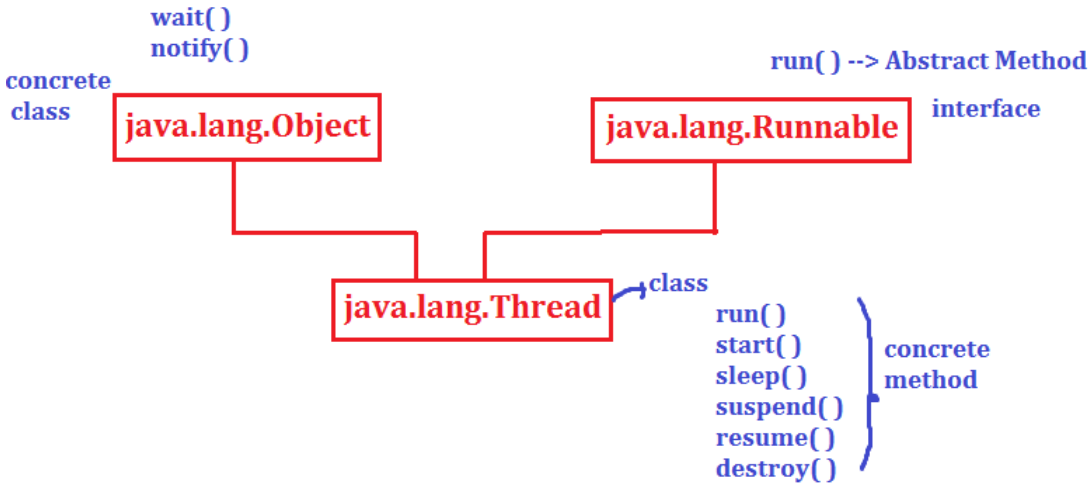
Thread Life cycle:

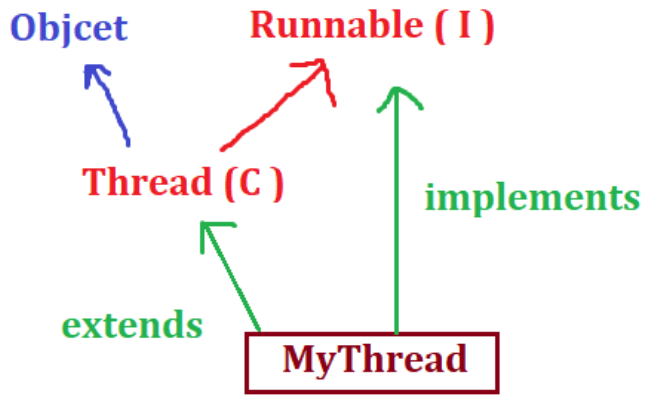


We can creates new threads in two ways

By extending java.lang.Thread class

By implementing java.lang.Runnable interface





--> If you want to create user-defiend Threads then we have two options

1) By extends Thread class
2) By implements Runnable interface

1) By extends Thread class

---

1) Create your class by extends java.lang.Thread class

Ex:- class MyThread extends Thread
{
}

2) Override run( ) Method

Ex- public void run( )
{ ..... }

3) Write main( ) method then Create object

Ex:- public static void main(String[] args)
{
MyThread t1 = new MyThread( );
.....
}

4) call start( ) method

Ex:- t1.start( )

Note:- start( ) method will call run( ) method

Ex:-

```
class PrintDocument extends Thread
{
    public void run( )
    {
        for(int i =1; i <= 10 ; i++)
            { System.out.println(" From Printer -1 page# "+i); }
    }
    public static void main(String[] args)
    {
        System.out.println(" Main Method ");
```

```
                PrintDocument   p  = new   PrintDocument( );
                    p.start( );

                     for(int  i =1;  i <= 10 ; i++)
                          {   System.out.println(" From  Printer -2  page# "+i);  }
            }
}
```


2) By Implements Runnable Interface

Step1:- Create your class by implements runnable interface

```
                     class   MyThread   implements   Runnable
                     {
                     }
```

step2:- Over ride run( )

```
                    public   void   run( )
                    {                 .....           }
```

step3:-write main( ) method, then create object for Your class

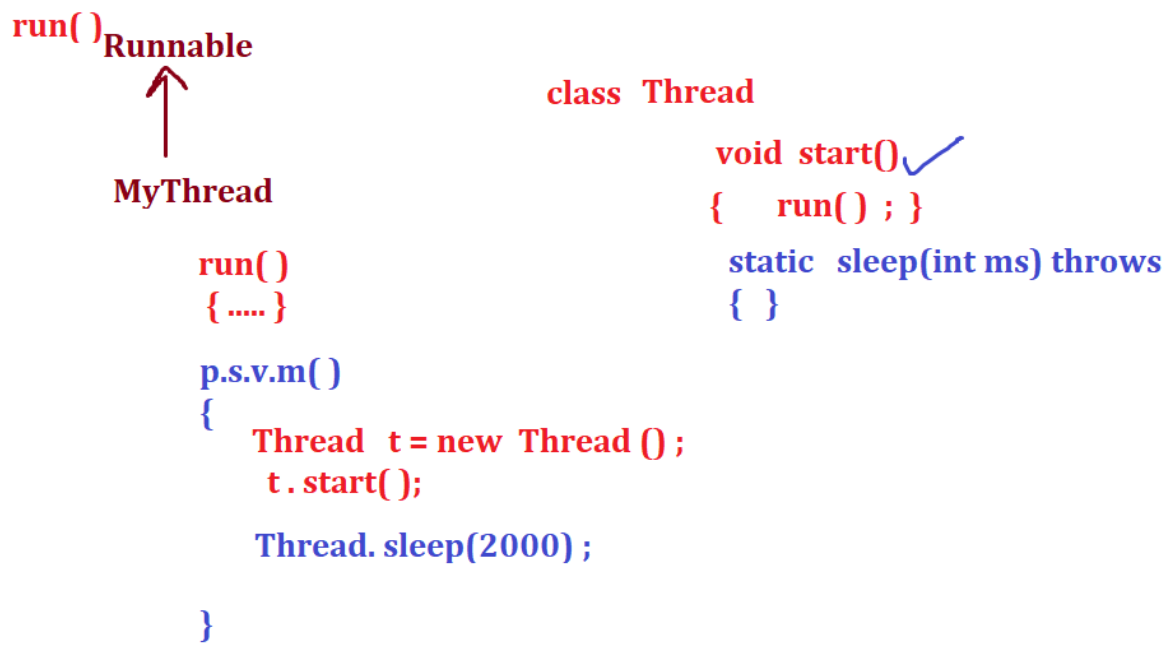
```
                     public  static  void  main(String[]  args)
                    {
                                  MyThread    m =  new  MyThread( );
                    }
```

step4:- Create object for Thread class By attach your class object

```
                    Thread    t  =  new   Thread(m) ;
```

Step5:- call start( ) Method

```
                    t.start( )
```



Ex:-

```
class  PrintDocument  implements  Runnable
{
          public  void   run( )
          {
                      for(int  i =1;  i <= 10 ; i++)
                    {   System.out.println(" From  Printer -1  page# "+i);  }
          }
               public  static  void  main(String[]   args)
          {
                      System.out.println(" Main Method ");

                    PrintDocument    p  = new   PrintDocument( );
                      Thread     t  =  new   Thread(p);
                           t.start( );

                     for(int  i =1;  i <= 10 ; i++)
                           {   System.out.println(" From  Printer -2  page# "+i);  }
          }
}
```

--> If you want to create your own threads , we can create in 2 ways

Ex:-

```
class  Test  extends  Thread
{
}
```

Ex2:

```
class   Test
{
}
class  Demo  extends  Test  implements Runnable
{
}
```

Note:- If your class is normal class then use extends Thread class
If your class is child class then use implements runnable interface

sleep(ms) method

Ex:-

```
class  PrintDocument  extends  Thread
{
    public  void  run( )
    {
        for(int  i =1;  i <= 10 ; i++)
        {
            try{
                sleep(2000);
              }
            catch(InterruptedException  e1)
            {  e1.printStackTrace( ) ; }
            System.out.println(" From  Printer -1  page# "+i);
        }
    }
    public  static  void  main(String[]  args)  throws  InterruptedException
      {
            System.out.println(" Main Method ");
            PrintDocument   p  = new   PrintDocument( );
            p.start( );
            for(int  i =1;  i <= 10 ; i++)
            {
                sleep(2000);
                System.out.println(" From  Printer -2  page# "+i);
```

```
            }
        }
}
```

--> If we have two system with one printer
that meams two threads are trying to access on single resource

VENKATREDDY

```
class  Thread
{
    static  void  sleep(int  ms) throws  InterruptedException
    {
        .....
    }
        .....
}
```

```
class  Test  extends  Thread
{
    p.s.v.m( .....)  (or) throws  InterruptedException
    {
        try{
            sleep(2000) ;
          }
        catch(InterruptedException  e1)
        {   e1.printStackTrace( );  }
}
```

Ex:-

```
class  PrintDocument  implements Runnable
{
        public  void  run( )
        {
            for(int  i =1;  i <= 10 ; i++)
            {
                try{
                    Thread.sleep(2000);
                  }
                catch(InterruptedException e1)
                  {  e1.printStackTrace( );  }
              System.out.println(" From  Printer -1  page# "+i);
            }
        }
      public  static  void  main(String[]  args)
        {
            System.out.println(" Main Method ");
          PrintDocument   p  = new   PrintDocument( );
          Thread   t1   =  new   Thread(p);
                Thread   t2   =  new   Thread(p);
```

```
                    t1.start( );
                    t2.start( );
        }
}
```

--> We can give names to our Thread, and we can get the names of a Thread

VENKATREDDY

Thread class Methods

---

1) setName(String name)

It is Used to set the name for Thread

```
MyThread  t1 = new MyThread( )
t1.setName(" Venkat ");
```

2)getName( )

It is used to get current working Thread

```
//String  name = Thread.currentThread.getName( )
Thread    t = Thread.currentThread( );
String   tname = t.getName( ) ;
```

Ex:-

```
class   PrintDocument  implements Runnable
{
        public  void   run( )
        {
            for(int  i =1;  i <= 10 ; i++)
              {
                  Thread    t = Thread.currentThread( );
                String   tname = t.getName( ) ;
                 try{
                        Thread.sleep(2000);
                    }
                    catch(InterruptedException e1)
                      {  e1.printStackTrace( );  }

                    System.out.println(tname+" page# "+i);
                }
        }
        public  static  void  main(String[]   args)
        {
```

```
                        System.out.println(" Main Method ");
                                PrintDocument   p = new  PrintDocument( );
                                Thread   t1  = new  Thread(p);
                                Thread   t2  = new  Thread(p);
                                t1.setName("Vishnu");
                                t2.setName("Avinash");
                                t1.start( );
                                t2.start( );
          }
}
```

Example Program on Movie Tickets

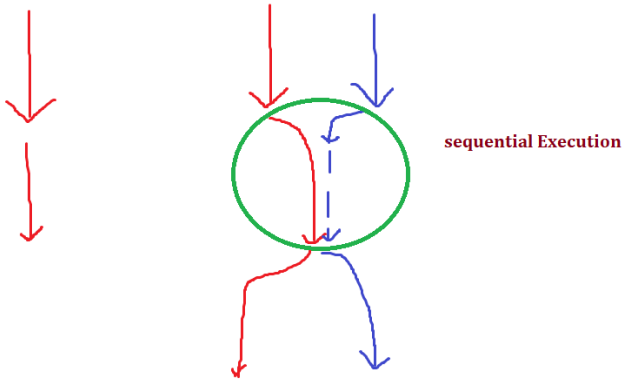
```
class  MovieTicket implements Runnable
{
          int seats ;
          MovieTicket(int seats)
        {
                this.seats = seats ;
        }
        public  void   run( )
        {
                Thread    t = Thread.currentThread( );
                String  name = t.getName( ) ;
                System.out.println(name+" Number of Tickets are : "+seats);
                if(seats > 0)
                {
                        try{
                                Thread.sleep(2000);
                          }
                                catch(InterruptedException e1)
                                                { e1.printStackTrace( ); }
                         System.out.println(name + " Book  Ticket ");
                                seats = seats - 1 ;
                                System.out.println(" Remaing  Tickets  : "+seats);
                }
                else
                                System.out.println(" Sorry! No Tickets ") ;
        }
        public static void  main(String[]  args)
        {
                        MovieTicket   m = new  MovieTicket(1);
                        Thread   t1 = new  Thread(m);
```

```
                Thread   t2 = new  Thread(m);
                t1.setName("Vishnu");
                t2.setName("Avinash");
                t1.start( );
                t2.start( );
        }
}
```


Data In-consistency:

Whenever one of the threads is updating the value and the other thread is trying to read the value at the same time then Problem occurs i.e data inconsistency.

To avoid the data inconsistency problem we have to synchronize the threads that are acting on the same object



Synchronization :

Synchronization is nothing but locking technique.

➔ If multiple threads trying to access same resources at a time then we are going to get "data inconsistency" problem, to avoid that use synchronized keyword.

Thread synchronization is possible in 2 ways

Synchronized block:

It is for specific block of statements mention inside of a block

Syntax: synchronized(this)

{ Statements }

Synchronized method:

To synchronize all the statements in a method then use synchronized method

Syntax: synchronized returntype methodname(list of parameters)

{ statements ; }

Avoid Data inconsistency problem by using Synchronized block

```
class MovieTicket implements Runnable
{
    int seats ;
    MovieTicket(int seats)
    {
        this.seats = seats ;
    }
    public void run( )
    {
        Thread t = Thread.currentThread( );
        String name = t.getName( ) ;
        System.out.println(name+" Number of Tickets are : "+seats);
        synchronized(this)
        {
            if(seats > 0)
            {
                try{
                    Thread.sleep(2000);
                }
                catch(InterruptedException e1)
                    { e1.printStackTrace( ); }
                System.out.println(name + " Book Ticket ");
                seats = seats - 1 ;
                System.out.println(" Remaing Tickets : "+seats);
            }
            else
                System.out.println(" Sorry! No Tickets ") ;
        }
    }
    public static void main(String[] args)
    {
        MovieTicket m = new MovieTicket(1);
        Thread t1 = new Thread(m);
```

```
                Thread   t2 = new  Thread(m);
                t1.setName("Vishnu");
                t2.setName("Avinash");
                t1.start( );
                t2.start( );
        }
}
```


Avoid Data inconsistency problem by using Synchronized Method

```
class  MovieTicket implements Runnable
{
        int seats ;
        MovieTicket(int seats)
        {
                this.seats = seats ;
        }
        synchronized public  void   run( )
        {
                Thread    t = Thread.currentThread( );
                String  name = t.getName( ) ;
                System.out.println(name+" Number of Tickets are : "+seats);
                if(seats > 0)
                {
                        try{
                                Thread.sleep(2000);
                           }
                                catch(InterruptedException e1)
                                {  e1.printStackTrace( );  }
                                System.out.println(name + " Book  Ticket ");
                                seats = seats - 1 ;
                                System.out.println(" Remaing  Tickets  : "+seats);
                }
                else
                                System.out.println(" Sorry! No Tickets ") ;
        }
        public static void  main(String[]  args)
        {
                MovieTicket   m = new  MovieTicket(1);
                Thread   t1 = new  Thread(m);
                Thread   t2 = new  Thread(m);
                t1.setName("Vishnu");
                t2.setName("Avinash");
```

```
                t1.start( );
                t2.start( );
        }
}
```

Different Methods

1) currentThread( ):
   It is used to get current executing Thread information
   Thread t = Thread.currentThread( );
2) start( ):
   It is used to execute a user defined thread, start( ) internally calls run( )
3) sleep(long milliseconds)
   It is used to suspend the execution of a thread for a specified amount of time.
   This method gives InterruptedException, which must be handled

4) getName( )
   It is used to get current working Thread
   //String name = Thread.currentThread.getName( )
   Thread t = Thread.currentThread( );
   String tname = t.getName( ) ;

5) setName(String name)
   It is Used to set the name for Thread
   MyThread t1 = new MyThread( )
   t1.setName(" Venkat ");

6) wait( )
   This method is used to suspend the execution of a thread until it receives a notification
7) notify( )
   It is used to send a notification to one of the waiting threads
8) notifyAll( ):
   It is used to send a notification to all the waiting threads

Note: wiat( ), notify( ), notifyAll( ) are used for inter thread communication, they must be called inside synchronized block otherwise we get a run time error called IllegalMonitorStateException.

Above 3 methods are available in Object class so we can access directly

```
public class MyApp1 extends Thread
{
        static int total=0;
        public void run( )
        {
                 System.out.println(" User Thread started calculation ");
                int   s1 = 50,s2=63,s3=70;
                 total=s1+s2+s3;
                 System.out.println(" User Total is : "+total);
        }
        public static void main(String[] args)
        {
                MyApp1  m1 = new MyApp1();
                System.out.println(" Main Thread calling user Thread ");
                m1.start() ;
                double  avg=total/3;
                System.out.println(" Main Average  is : "+avg);
         }
}
```

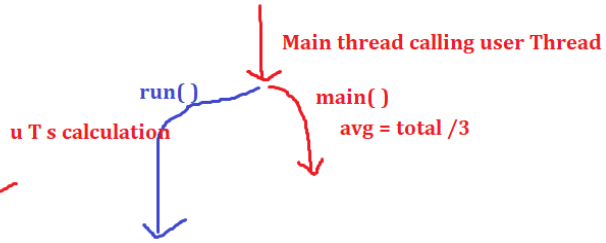


```
 static  int  total = 0 ;
public void run( )
{
       System.out.println(" User Thread started calculation ");
      int   s1 = 50,s2=63,s3=70;
       total=s1+s2+s3;
       System.out.println(" User Total is : "+total);
}

public static void main(String[] args)
{
      MyApp1  m1 = new MyApp1();
       System.out.println(" Main Thread calling user Thread ");
       m1.start() ;
       double  avg=total/3;
       System.out.println(" Main Average  is : "+avg);
}
```

s1 s2 s3 : 50 63 70

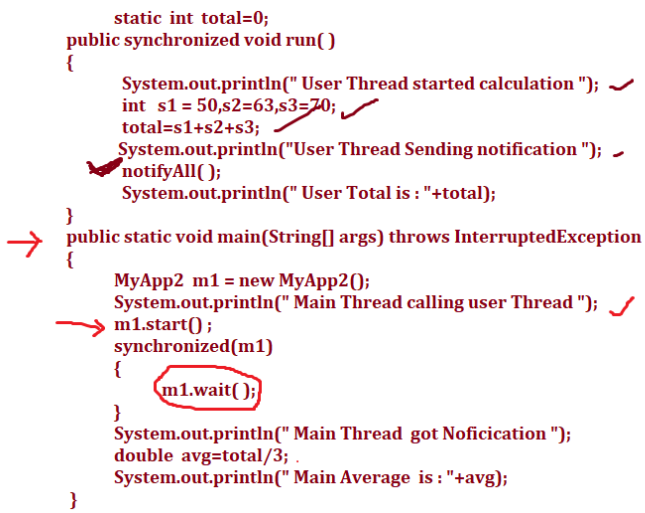
total : 0 → 183

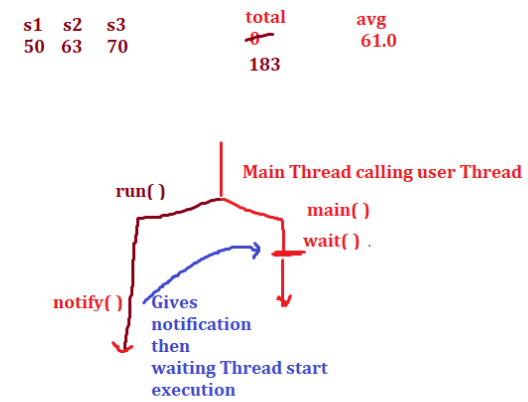
avg : 0



Example for wait and notify

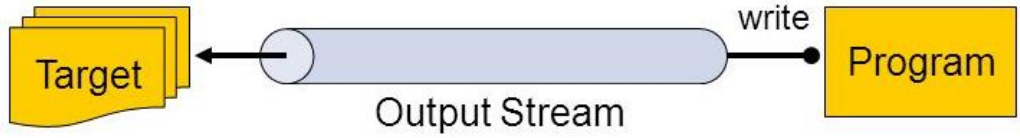
```
public class MyApp2 extends Thread
{
        static  int  total=0;
        public synchronized void run( )
        {
                System.out.println(" User Thread started calculation ");
                int   s1 = 50,s2=63,s3=70;
                total=s1+s2+s3;
               System.out.println("User Thread Sending notification ");
                notifyAll( );
                System.out.println(" User Total is : "+total);
        }
        public static void main(String[] args) throws InterruptedException
        {
                MyApp2  m1 = new MyApp2();
                System.out.println(" Main Thread calling user Thread ");
                m1.start() ;
                synchronized(m1)
                {
                        m1.wait( );
                }
                System.out.println(" Main Thread  got Noficication ");
                double  avg=total/3;
                System.out.println(" Main Average  is : "+avg);
         }
}
```





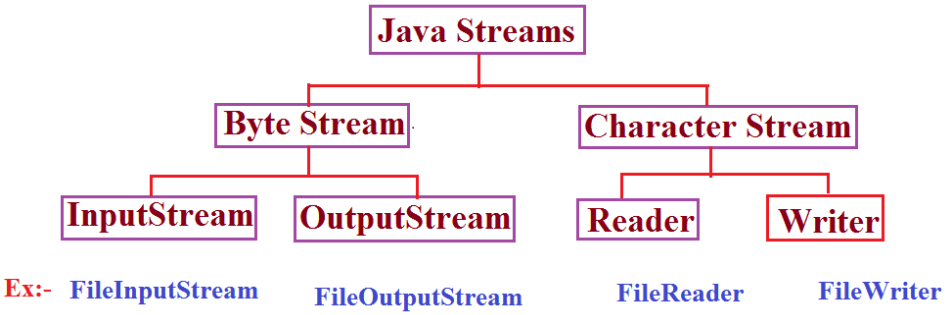
Java Streams

- A Stream is a flow of data from source to destination
- A source can be a keyboard, file, client, server, etc…
- Destination can be monitor, file, client, server, etc…
- The Streams that are used for performing input and output operations are called as IOStreams



Streams are classified into two types they are

1. Byte Stream
2. Character Streams



1. Byte Streams

The streams that are performing operations byte by byte are called as byte streams. These Streams are further classified into two types

InputStream: Using these Streams we can read data from the application or keyboard in the form of byte by byte

Ex: **FileInputStream**

OutputStream: Using these Streams we can write the data in the application or monitor in the form of byte by byte

Ex: FileOutputStream

Character Streams

The Streams which performs operations character by character are called as character streams. These Streams can handle only text.

They are also called as text stream

Character streams are two types:

Reader: It will perform reading operations

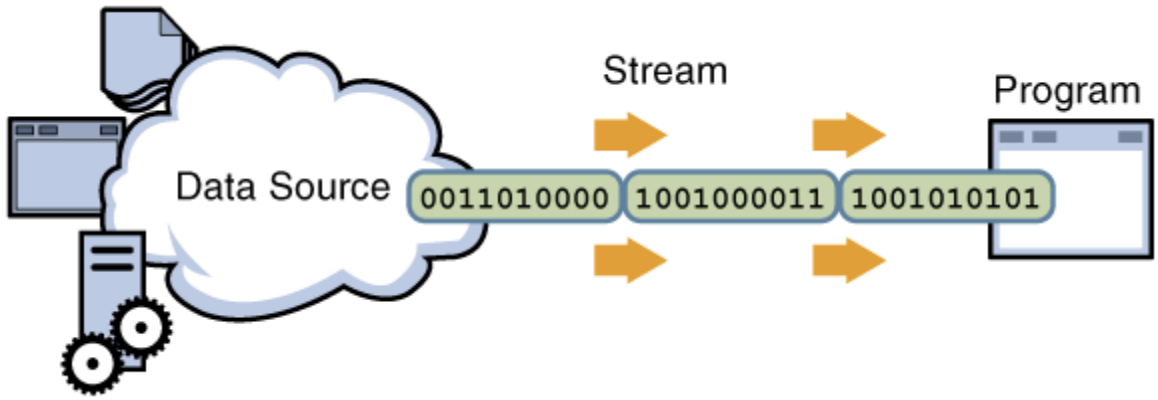
Ex: FileReader

Writer: It will perform writing operation

Ex: FileWriter

Note: All the stream related classes are available in java.io.Package

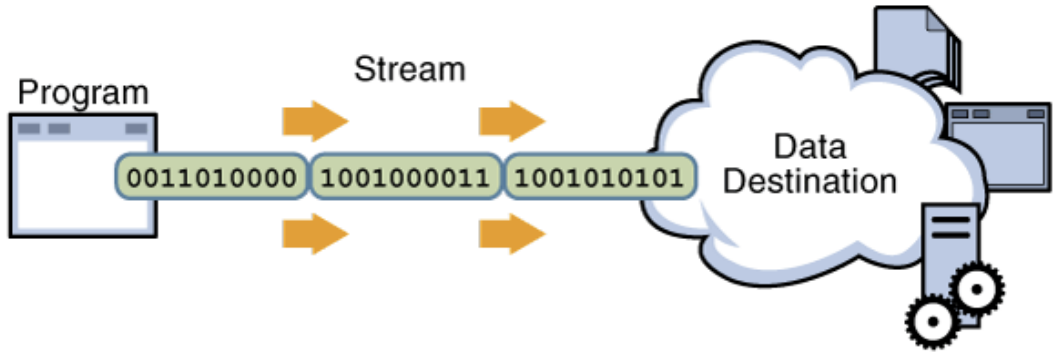
Read Data From File



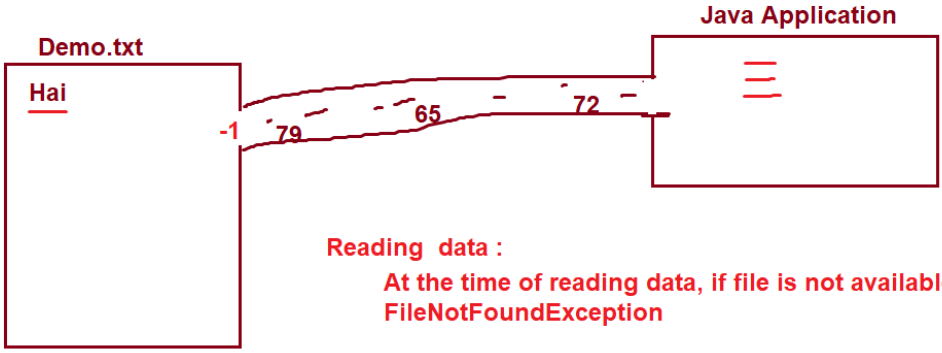
Note: Once data is finish in file, then it returns "-1"

If file is not available then it return Exception "FileNotFoundException"

Write Data into File



Note:- If File is not available, then it will create a new File and store data



Reading data :

At the time of reading data, if file is not available then it returns FileNotFoundException

At the time of Writing data

--> If file is available then, it will store data,
--> if file is not available, then it will create a file then it will store data

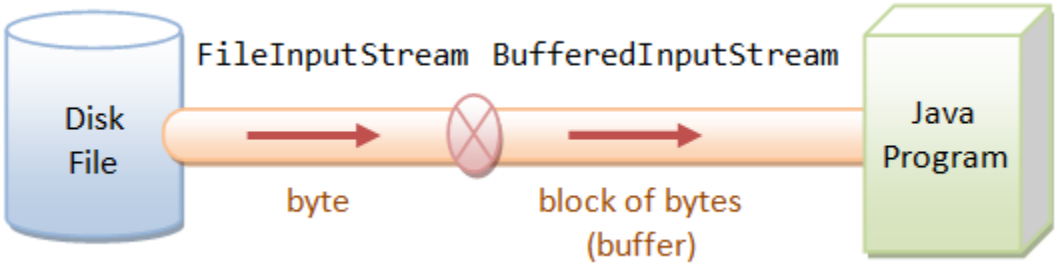
FileInputStream:

FileInputStream is used for reading the contents from file

Syntax:

FileInputStream fis = new FileInputStream(String);

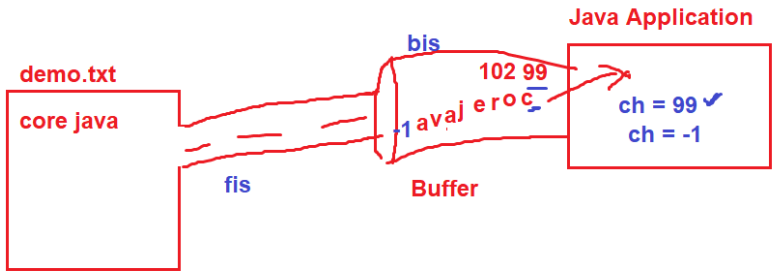
Note: If the file is not available then it will raise "FileNotFoundException"

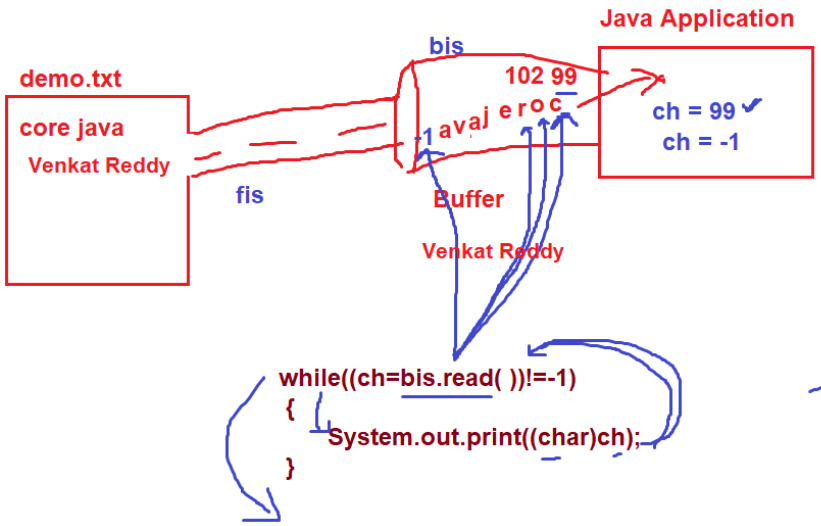


Example Program

```
import java.io.*;
class FileReadEx1
{
	public static void main(String[] args)
	{
		try{
			FileInputStream fis = new FileInputStream("demo.txt");
			BufferedInputStream bis = new BufferedInputStream(fis);
				int ch;
				while((ch=bis.read( ))!=-1)
				{
					System.out.print((char)ch);
				}
				 bis.close( );
		}
		catch(FileNotFoundException e1)
		{ System.out.println(e1); }
		catch(IOException e2)
		{ System.out.println(e2); }

	}
}
```

```
FileInputStream fis   = new  FileInputStream("demo.txt");

BufferedInputStream bis  = new  BufferedInputStream(fis);
 int  ch;

 while((ch=bis.read( ))!=-1)
 {
      System.out.print((char)ch);
 }
  bis.close( );
```





co

V E N K A T R E D D Y

FileOutputStream:

It is used for writing the data into a file

Syntax:

FileOutputStream fos = new FileOutputStream(String);

FileOutputStream is used for writing the data into the specified file.

If the specified file is not available then it will create a new file and stores the data

If the specified file is already available then it will override

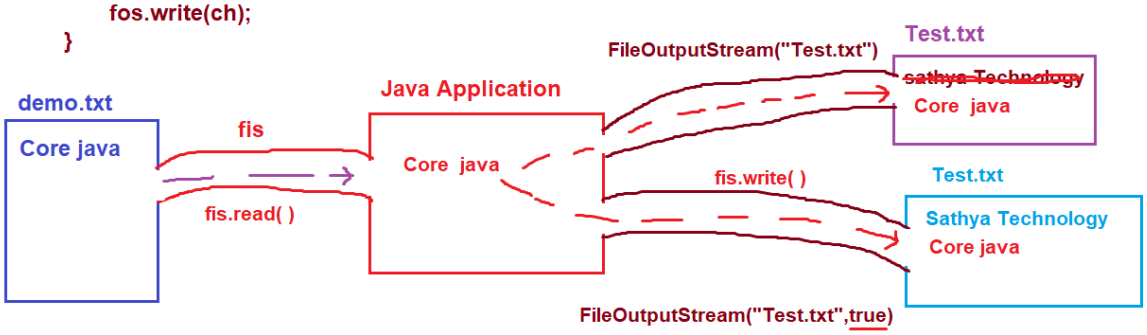
Syntax2:

FileOutputStream fos = new FileOutputStream(String, boolean);

If boolean value is true then data is append otherwise **data is override**

```
import java.io.*;
class FileOutputEx1
{
        public static void main(String[] args)throws
IOException,FileNotFoundException
        {
                FileInputStream fis = new FileInputStream("demo.txt");
                FileOutputStream fos = new FileOutputStream("Test.txt",true);
                int ch;                                 // true means append data
                while((ch=fis.read())!=-1)      // false means override, by default false
                {
                        fos.write(ch);
                }
                System.out.println(" Your Data is stored in Test.txt file ");
                fis.close();
                fos.close( );
        }
}
```

```
FileInputStream  fis = new FileInputStream("demo.txt");
FileOutputStream  fos = new FileOutputStream("Test.txt",true);
int  ch;                                                    // true means append data
while((ch=fis.read())!=-1)                                  // false means override, by default false
{
     fos.write(ch);
}
```



File:

File is a predefined class, by using it we can refer a file or a directory

Syntax:

File f = new File(String path);

It will point to Specified file

FileReader:

It is used to read the content form a file character by character

Syntax: FileReader fr = new FileReader(filename/path);

Ex: File f = new File("demo.txt");

FileReader fr = new FileReader(f);

Note:- If File is not available then we will get FileNotFoundException

Ex:-

```
// FileReader     -- >  Reading Data from File by using   CharacterStream classes
import  java.io.*;
class   FileReaderEx1
{
        public static void main(String[] args) throws IOException,
FileNotFoundException
        {
                File  f =  new  File("D:/CoreJava11am/IO/demo.txt");
                FileReader  fr = new FileReader(f);
                int ch;
                while((ch=fr.read( ))!=-1)
                {
                        System.out.print((char)ch);
                }
                fr.close( );
        }
}
```


FileWriter:

It is used for writing the contents into a file

Syntax:

FileWriter fw = new FileWriter(String);

(or)

FileWriter fw = new FileWriter(String, boolean);

To write the data into file, use write(data) method

Ex:-

```
// Writing data into file

import java.io.*;

class  FileWriterEx1

{

        public static void main(String[] args) throws Exception

        {

                File   f   =  new File("D:/CoreJava11am/IO/demo1.txt");

                FileWriter  fw = new FileWriter(f);

                 fw.write('S.');

                 fw.write(" Venkat Reddy");

                 String  s ="Faculty for Python and Java in Sathya Technology";

                 char[ ]  ch = s.toCharArray();

                 fw.write(ch);

                 fw.close( );

        }

}
```

ObjectInputStream:

It is used to read object

Syntax:

ObjectInputStream ois = new ObjectInputStream(InputStream);

ObjectOutputStream:

It is used to write an object

Syntax:

ObjectOutputStream oos = new ObjectOutputStream(outputStream);

- ➔ If you want to store an object into file, that object should be serialized.
- ➔ An object is said to be serialized when its corresponding class is implementing Serializable interface
- ➔ Serializable interface is called as marked interface or tagged interface.
- ➔ If an object is serializable then only that object broken into pieces

Serialization:

It is a Process of converting an object into a stream of bytes

Deserialization:

It is a process of converting a stream of bytes into an object

We can perform serialization and deserialization only when the object is serialized, otherwise a runtime error called NotSerializableException will occur

Program :

Ex1:

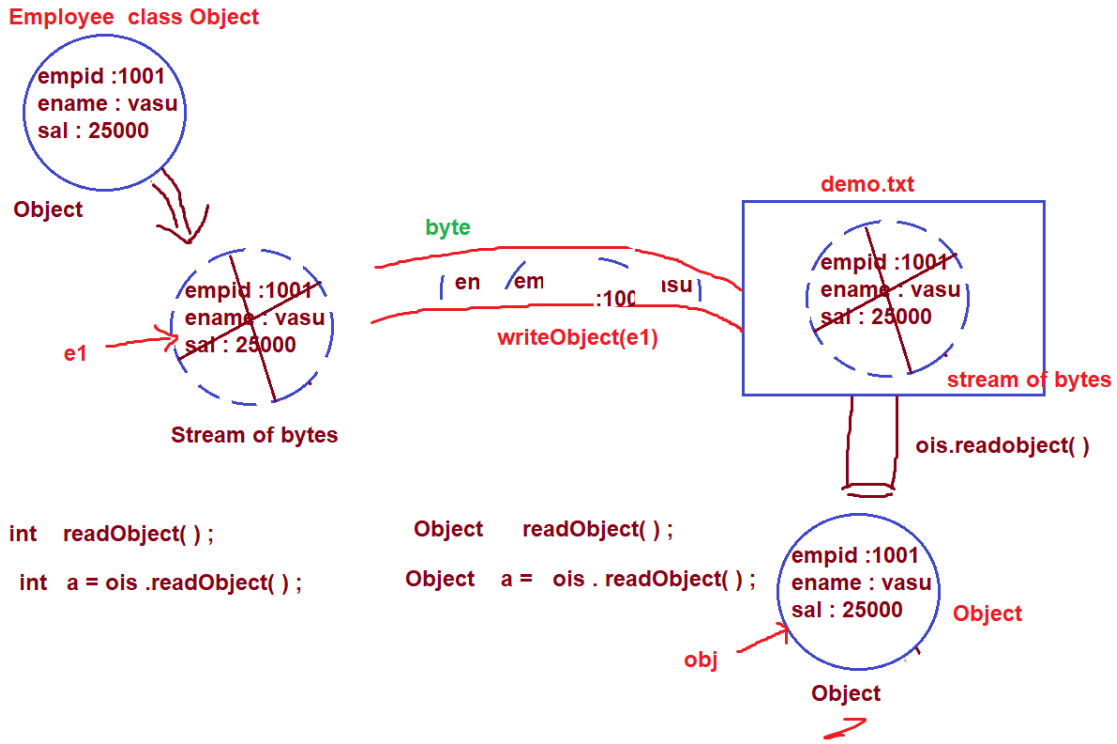
```
// Creating Employee Class

import  java.io.*;
class  Employee implements Serializable
{
        int  eid = 101;
        String name = "Venkat";
        double salary = 15000;
        void show( )
        {
                System.out.println("Employee ID : "+eid);
                System.out.println("Employee Name : "+name);
                System.out.println("Employee Sal : "+salary);
        }
}
```

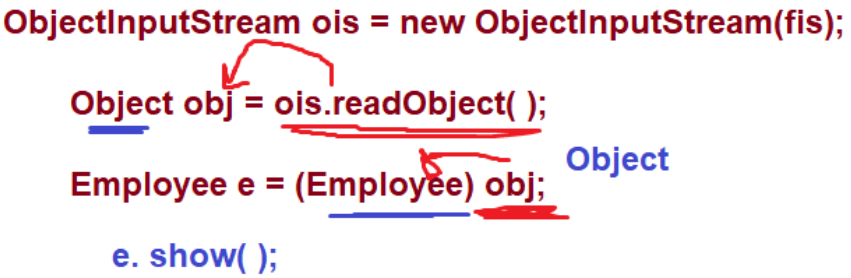
Ex2:

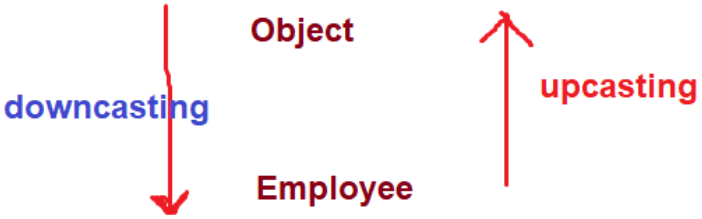
```
//Storing Employee Class Data into file
import java.io.*;
class  StoreObject
{
        public static void main(String[] args) throws IOException
        {
                FileOutputStream  fos = new FileOutputStream("Empdata.txt");
                ObjectOutputStream oos = new ObjectOutputStream(fos);
                Employee e1 = new  Employee();
                oos.writeObject(e1);
                oos.close();
        }
}
```

Ex3:

```
//Reading Employee Object Data from File
        import java.io.*;
        class  ReadObject
        {
                public static void main(String[] args)throws Exception
                {
                        FileInputStream fis = new FileInputStream("Empdata.txt");
                        ObjectInputStream ois = new ObjectInputStream(fis);
                        Object obj = ois.readObject( );
                        Employee e = (Employee) obj;
                        e.show();
                        ois.close();
                }
        }
```







Transient:

If a serialized object is transferred from one location to another location, then all the data available in that object will be transferred.

If you want to restrict any data(variable) from the serialized object, then declare that variable as transient

Ex:- transient String password="12345";

Transient is a modifier which is used to apply only on variables. It will effect only for serialized objects

Array:

Array is used to store group of values in sequential Order(Memory allocation).

Arrays are used to store both primitive types and objects

Limitations of Array:

The size of array is fixed; we cannot increase or decrease the size of the array

The array concept doesn't provide any pre defined methods to perform the basic operations like insertion, deletion, searching or sorting etc…

Collection Frame Work

Collections

It is an object which can store group of other objects

The size of collection is not fixed. It will increase or decrease

Collections provide methods to perform basic operations like insertions, deletion, searching, sorting etc…

Collections are designed to store only objects…

Collection Class :

It is a class whose object can store references of other objects.

The Collection classes are classified into following 3 types

List

Set

Map

List:

List is a collection, which can store group of individual objects

list allows duplicate values

List is an interface.

The Implement classes are

* ArrayList
* LinkedList
* Vector

Stack

Set:

Set is a collection, which is used to store and Perform operations on group of individual objects.

Set doesn't allow duplicates

Set is an interface. The implementation classes are

- HashSet
- LinkedHashSet
- TreeSet

Map:

Map is a collection, which can be used for storing the objects in the form of key-value pairs where the keys cannot be duplicated but the values can be duplicated.

Map is an interface. The implementation classes are

- HashMap
- LinkedHashMap
- TreeMap
- Hashtable

List

ArrayList:

ArrayList is an implementation class of list interface
ArrayList is used to store and Perform operations on group of individual objects
ArrayList allows duplicates
ArrayList allows null value
ArrayList is not synchronized

Creation of ArrayList

Syntax:

```
ArrayList<E> a1 = new ArrayList<E>( );
```

→ The above syntax will create an empty list with default initial capacity as 10.

```
ArrayList<E> a1 = new ArrayList<E>(int initialCapacity );
```

→ The above syntax will create an empty list with default initial capacity as the specified value

Methods:

1) boolean add(elements): it add new element at the end of the list
2) void add(int index,element): add element at the specific index
3) boolean remove(element): it will remove the element at first occurrence
4) boolean remove(int index): It removes elements at specific index
5) void clear( ): Remove all the elements from list
6) int size( ): Returns the size of list
7) boolean contains(elements): it checks whether element is available or not.
8) int indexOf( ) :- It is used to return index number of a specified element

Ex:-

```
import   java.util.ArrayList;
class   ArrayListEx1
{
        public  static  void  main(String[]   args)
        {
                        ArrayList<String>   a1 = new   ArrayList<String>( );
                        a1.add("Vishnu");
                        a1.add("Avinash");
                        a1.add("Sharma");
                        a1.add("Leena");
                        a1.add("JVRAO");
                        System.out.println(a1);
                        // Add  Pavana name in index number 2
                        a1.add(2, "Pavana")  ;
                        System.out.println(" Your New List  is : "+a1);
                        //  remove  Avinash  from list
                        if(a1.contains("Avinash"))
                        {
                                a1.remove("Avinash");
                                System.out.println(" Avinash is deleted ");
                        }
                        else
                                System.out.println(" Avinash is not avaialbe in our list ");

                        System.out.println(" Your List  is : "+a1);
                        // We can remove elements based on index numbers
                        a1.remove(3);
                        System.out.println(" After deleting 3 rd index element : "+a1);
                        int  s = a1.size() ;
                        System.out.println(" Total number of elements are : " +s);
                        System.out.println(" Your List Elements using For loop ");
                        for(String   i : a1)
                                System.out.println(i);

        }
}
```

Ex2:-

```
import   java.util.ArrayList;

import   java.util.Scanner;
class   ArrayListEx1
{
        public  static  void  main(String[]   args)
        {
                        Scanner   sc =  new   Scanner(System.in);
                        ArrayList<String>   a1 = new   ArrayList<String>( );
                        String   s ;
                        char  ch;
                        do{
                                System.out.print("Enter Emp name : ");
                                    s = sc.next( );
                                    a1.add(s);
                                System.out.print("Do you want to another name(y/n) : ");
                                    ch = sc.next( ).charAt(0) ;
                            }while(ch=='y');

                        System.out.println(" Your Employee information is : "+a1);

        }
}
```

```
Enter Your  Order
        Item - 1        :    Pizza
        Item - 2        :    Burger
        Item - 3        :    Chips
Your Order is           :    [ "Pizza", "Burger", "Chips" ]
Do you want to modify order       :     yes
    1 - Add New Item
    2 - Modify Item
    3 - Remove Item
Select Your Option    :     2

Which item you want to modify     :     2
Enter your Item         :     Manchuria
Your New Order is       :     [ "Pizza", "Manchuria", "Chips" ]
```

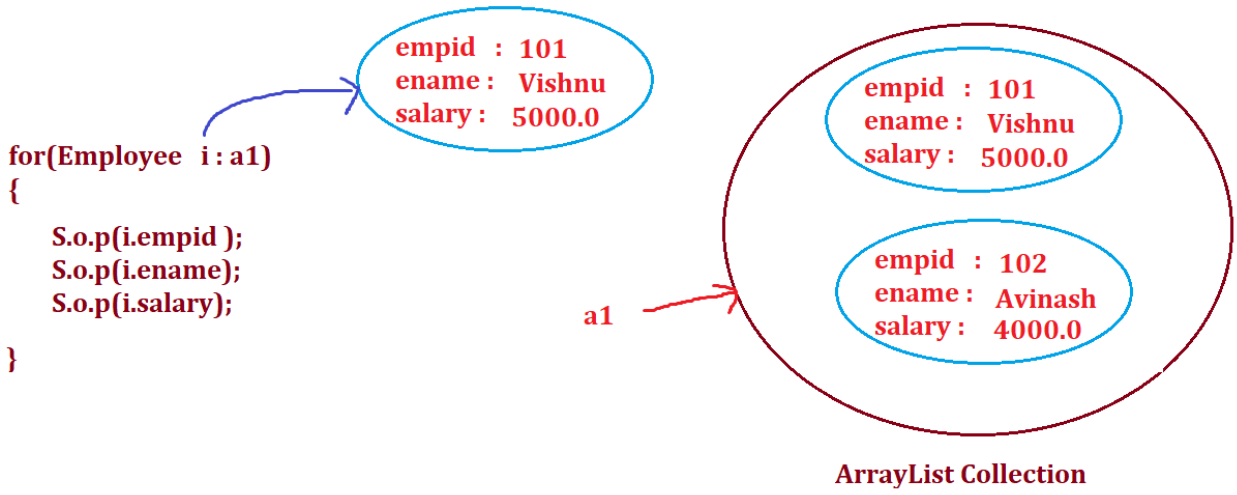
```
// create Employee class objects then store into collections
class Employee
{
        int eid;
        String ename;
        double salary;
        Employee(int eid, String ename, double salary)
        {
                this.eid = eid ;
                this.ename = ename ;
                this.salary = salary ;
        }
}
```

Ex:

```
import java.util.ArrayList;
class ArrayListEx2
{
        public static void main(String[ ] args)
        {
                ArrayList<Employee> a1 =new ArrayList<Employee>() ;
                Employee e1 = new Employee(101,"vishnu",5000.0);
                Employee e2 = new Employee(102,"Avinash",4000.0);
                Employee e3 = new Employee(103,"sharma",7000.0);
                Employee e4 = new Employee(104,"jvrao",3000.0);
                a1.add(e1);
                a1.add(e2);
                a1.add(e3);
                a1.add(e4);
                System.out.println(" Your ArrayList is : "+a1);
        }
}
```

Ex2:

```
import  java.util.ArrayList;
class  ArrayListEx2
{
        public  static  void  main(String[ ]  args)
       {
                ArrayList<Employee>  a1  =new  ArrayList<Employee>() ;
                Employee  e1  =  new  Employee(101,"vishnu",5000.0);
                Employee  e2  =  new  Employee(102,"Avinash",4000.0);
                Employee  e3 =   new  Employee(103,"sharma",7000.0);
                Employee  e4 =   new  Employee(104,"jvrao",3000.0);
                a1.add(e1);
                a1.add(e2);
                a1.add(e3);
                a1.add(e4);
                System.out.println(" Your ArrayList is : "+a1);
                for(Employee  i: a1)
                {
                        System.out.println( i.eid+"\t"+i.ename+"\t"+i.salary) ;
                }
       }
}
```

Iterator:

--> It is used To Display Object Information

--> Iterator is an interface, it contains following methods

1) hasNext( ):

--> It is used to check whether next element is available or not

If available, it returns “true” otherwise it returns “false”

2) next( ):

--> It is used to point next record

Ex:-
```
Iterator<className> i = a1. iterator();
    while(i.hasNext( ))
    {
        classname a=i.next( );
        System.out.println(a.id+" "+a.name+" "+a.sal);
    }
```

Examples :
```
int sum( )
  { ..... }
```
Note : reference is a1
```
int a = a1.sum( );
```

Ex:
```
int[ ] show( )
    { ..... }
// How to call show( ) , Reference is a1
    int[ ] a = a1.show( );
```
Ex:
```
    Customer show( )
  { ..... }
// How to call show( ), Reference is a1
Customer a = a1.show( );
```
Ex:-
```
    Iterator<Element> iterator( );
// How to call iterator( ), Reference is a1
Iterator<Employee> a = a1.iterator( );
```

Ex:-

```
Iterator<Employee>  i  =  a1.iterator( );
while(i.hasNext( ))
  {
        Employee   a = i.next( );
        System.out.println(a.eid+"\t"+a.ename+"\t"+a.salary);
  }
```


Example:

```
import   java.util.ArrayList;
class   ArrayListEx2
{
        public  static  void  main(String[ ]  args)
       {
              ArrayList<Employee>  a1  =new  ArrayList<Employee>() ;
              Employee   e1  =  new  Employee(101,"vishnu",5000.0);
              Employee   e2  =  new  Employee(102,"Avinash",4000.0);
              Employee   e3 =   new  Employee(103,"sharma",7000.0);
              Employee   e4 =   new  Employee(104,"jvrao",3000.0);
              a1.add(e1);
              a1.add(e2);
              a1.add(e3);
              a1.add(e4);
              System.out.println(" Your ArrayList is : "+a1);
            Iterator<Employee>  i  =  a1.iterator( );
            while(i.hasNext( ))
              {
                    Employee   a = i.next( );
                    System.out.println(a.eid+"\t"+a.ename+"\t"+a.salary);
            }

       }
}
```

ListIterator

--> It is used to display object information in both forward and backward directions

Methods:

1)hasNext( )

It is used to check next Element is available or not

2) hasPrevious( )

It is used to check Previous Element is available or not

3)next( )

It is used to move your pointer from current record to next record

4)previous( )

it is used to move your pointer from current record to previous record

Example:

```
import   java.util.ArrayList;
import  java.util.ListIterator;
class   ArrayListEx2
{
          public  static  void  main(String[ ]  args)
         {
                    ArrayList<Employee>   a1  =new   ArrayList<Employee>() ;
                    Employee   e1  =  new  Employee(101,"vishnu",5000.0);
                    Employee   e2  =  new  Employee(102,"Avinash",4000.0);
                    Employee   e3 =   new  Employee(103,"sharma",7000.0);
                    Employee   e4 =   new  Employee(104,"jvrao",3000.0);
                    a1.add(e1);
                    a1.add(e2);
                    a1.add(e3);
                    a1.add(e4);
                    System.out.println(" Your ArrayList is : "+a1);
                          ListIterator<Employee>   i  =  a1.listIterator( );
                          System.out.println(" Your Elements in Forward direction : ");
                          System.out.println(" Eid\tEname\tSalary");
                          System.out.println("---\t-----\t------");
                          while(i.hasNext( ))
                          {
                                    Employee   a = i.next( );
                                    System.out.println(a.eid+"\t"+a.ename+"\t"+a.salary);
                          }
```

```
                    System.out.println(" Your Elements in Backward direction : ");
                    System.out.println(" Eid\tEname\tSalary");
                    System.out.println("---\t-----\t------");
                    while(i.hasPrevious( ))
                      {
                            Employee   a = i.previous( );
                            System.out.println(a.eid+"\t"+a.ename+"\t"+a.salary);
                    }
        }
}
```


LinkedList

LinkedList is an implementation class of list interface

LinkedList is used to store group of individual values

Syntax:

LinkedList<E> l1 = new LinkedList<E> ( );

It will create an Empty list

ArrayList class follows "Resizable array structure", Which is faster in accessing the elements and slower in insertion and deletion

LinkedList follows "DoubleLinked Structure" which is faster in insertions and deletion and slower in accessing the elements.

Ex:-

```
import java.util.LinkedList;
import  java.util.Scanner;
class LinkedListEx1
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                LinkedList<Integer>  l1 = new LinkedList<Integer>();
                l1.add(3512);
                l1.add(3749);
                l1.add(3504);
                l1.add(3508);
                l1.add(3528);
                System.out.println("Bus Service Numbers are : "+l1);
                System.out.println(" Enter Your Service Number : ");
                int  sno=sc.nextInt( );
        if(l1.contains(sno))
                {
                        System.out.println(" Your Service no is Available ");
```

```
                }
                else
                {
                System.out.println(" Sorry..! Serive is not available ");
                }
        }
}
```



```
// For Above Program
// Add 3 more service numbers to existing list
// In place of 3749 place 2240 service
// Read service number dynamically then delete that service ,if Available
otherwise display
// message as Sorry the service is not available
```

Stack:

It is an implementation class of list interface

Stack is a child class of Vector class

The stack class should be use when we want to implement stack data structure

Creation of stack

Stack <E> s = new Stack<E>( );

It will create an empty list with default initial capacity as 10.

Methods:

Element push(Element):

It will push(add) specify element into stack

Element pop( ): It will remove Top most element of stack

Element peek( ) : It will return Top Most element from stack

boolean **empty**( ) : it is used to check whether stack is empty or not

int search(Element) : It is used to check whether specified element is available or not..

If Element is available it returns position of that element otherwise it returns -1

Example:

```
import java.util.Stack;
import java.util.Scanner;
class StackEx1
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
        Stack<Integer>  s1 = new Stack<Integer>( );
                s1.push(10);
                s1.push(20);
```

```
                    s1.push(30);
                    s1.push(40);
                    s1.push(50);
                    System.out.println(" Top Most Element is : "+s1.peek());
                    System.out.println(s1.pop( )+ " Element is deleted ");
                    System.out.print(" Enter any value to search : ");
                    int  x  =  sc.nextInt( );
                    if(s1.search(x)!=-1)
 // If Element is available it returns position of
 //        that  element other wise it returns -1
                       System.out.println(" Your Element is Available ");
                    else
                      Sytem.out.println(" Your Element is not Available ");
          }
}
```


	ArrayList	LinkedList	Stack
Ordered	Order by Insertion	Order by Insertion	Order by Insertion
Duplicates	Allowed	Allowed	Allowed
Null Value	Allowed	Allowed	Allowed
Synchronized	Not Synchronized	Not Synchronized	Synchronized
Data Structure	Resizable Array	Double Linked List	Resizable Array
Initial Capacity	10	.....	10

Set

Set is used to store and perform operations on group of individual objects.
It does not allow duplicates

HashSet:

It Stores elements using Hashing mechanism
It contains unique elements, i.e no duplicates
It uses hashtable for storage
HashSet is not synchronized
No guarantee for order of insertion

Syntax:

HashSet<E> hs = new HashSet<E> ( );

It creates empty set with the default initial capacity as 16.

Methods:

1) boolean add(Element obj): To place specified element in set
2) boolean remove(Element obj): To remove an element
3) boolean contains(element obj): To check whether element is available or not
4) boolean isEmpty( ): To check set is empty or not
5) int size( ): To find size of set
6) void clear( ): Remove all elements from set

Ex:-

W.a.p to store "Vodaphone","Tata docomo","Airtel","IDEA" mobile networks in list

i) Display all the Networks
ii) Add "JIO" network to list
iii) Check Tata docomo is available or not, if available then remove from network
iv) Now display total number of networks available in list and all the networks names

```
import   java.util.HashSet;
import   java.util.Scanner;
class   HashSetEx1
{
          public static void main(String[]   args)
          {
                Scanner   sc  =  new  Scanner(System.in);
                   HashSet<String>   hs  =  new   HashSet<String>( );
                        hs.add("Vodaphone");
                        hs.add("Idea");
                        hs.add("Docomo");
                        hs.add("airtel");
                        hs.add("BSNL");
                        System.out.println("Your Networkd are : "+hs);
                        System.out.print(" Enter network name : ");
                        String   n = sc.next( );
                        hs.add(n);
                        System.out.println("Your Networkd are : "+hs);
                        System.out.print(" Enter network to remove ");
                        String r = sc.next( );
                 if(hs.contains(r))
                        {
                                hs.remove(r);
                                System.out.println(r+" Network is Removed ");
                        }
                   else
                        System.out.println("Sorry..!"+r+" network is not available ");

                System.out.println("Your Networkd are : "+hs);
           }
}
```


Ex1:-

1) Create a hashset called Cart
2) Insert 5 Product ids to cart dynamically
   Ex:- 10012, 10024, 23121, 34231, 32321, …
3) Read one product id from customer
   Now check product is available or not, if available then delete that product
4) Now display number of products available in cart
5) Display all products information

Ex2:-

1) Create a HashSet
2) Store 5 Elements in HashSet
3) Take input from user and check element is available or not , if available then delete that element otherwise display the message that " Element is not available" then give a chance to enter element once again…

Ex3:-

1) Create a HashSet called cart
2) Add products dynamically until user says stop
3) Display all products available in cart
4) Display a menu like

   ADD

   Remove

   Give a chance to select option, based on customer option perform that operation.

```
import   java.util.HashSet;
import   java.util.Scanner;
class   HashSetEx1
{
   public static void main(String[]   args)
   {
        HashSet<String>   cart  =  new   HashSet<String>( );
        char  ch;
        String  network ;
        do{
                System.out.print("Enter Product name : ");
                p = sc.next( );
               cart. add(p);
              System.out.print(" Do you want to add another Product(y/n) : ");
              ch =  sc.next( ).charAt(0);
          }while(ch=='y');

           System.out.println(" Your Products are : " +cart);
           int n =cart.size( );
           System.out.println(" Total number of Products are : "+n);
           System.out.println("1. Add ")
          System.out.println("2. Remove ")
          System.out.print(" Enter your choice : ");
         int   op = sc.nextInt( );
          switch(op)
            {
                    case   1:       Read element to add
                                    Add element
                                     break;
                    case  2:    Read element to delete
                                    delete element
                                    break;
               }
           }
}
```

LinkedHashSet:

It contains only unique elements

It maintains insertion order

It sorts individual objects

Syntax:

LinkedHashSet<E> l1= new LinkedHashSet<E>( );

It creates an empty set with the default initial capacity as 16.

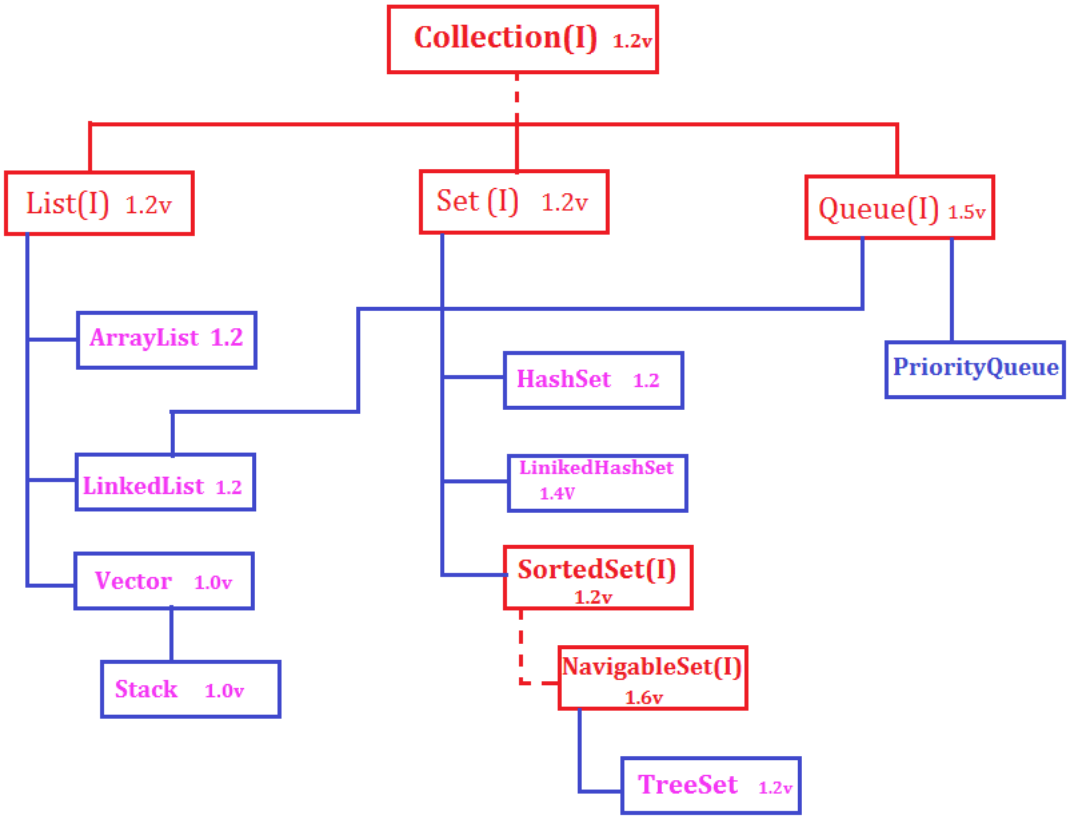
TreeSet:

Treeset sorts the elements in ascending order

Syntax:

TreeSet<E> ts = new TreeSet<E>( );

	HashSet	**LinkedHashSet**	**TreeSet**
Ordered	UnOrder	Ordered by Insertion	Sorted Order
Duplicates	Not Allowed	Not Allowed	Not Allowed
Null Value	Allowed	Allowed	Allowed
Synchronized	Not Synchronized	Not Synchronized	Not Synchronized
Data Structure	Hashtable	Hashtable+ double linked list	Balanced Tree
Initial Capacity	16	16	……..



Priority Queue is introduced from java 1.5v onwards…

Map:

Map is a collection of key-value pairs.

Where Keys never duplicated, Values may be duplicated

Ex:-

Key		**Value**
Code		**Place**
040	–	"HYD"
0866	–	"Vij"
0861	–	"Nellore"

Etc…

HashMap:

HashMap is an implementation class of Map interface

HashMap allows duplicates

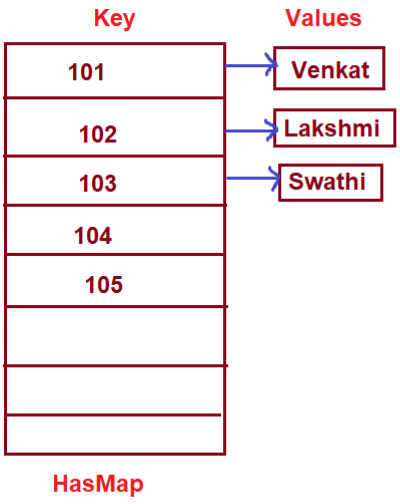
HashMap is not synchronized

No guarantee for order of insertion

Syntax:

HashMap<K,V> hm = new HashMap<K,V>( );

It creates an empty map with the default initial capacity as 16.




```
HashMap<Integer,String>  hm = new HashMap<Integer,String>;
To Add Elements
    hm . put(101,"venkat");
    hm . put(102, "Lakshmi");
To Remove Elements
     hm.remove(103) ;
To get values by using key
      String  v = hm.get(102);
To  get  all the keys
       Set    s  =  hm.keySet( )
To  get  all the values
        Collection   c  = hm.values()
```

Methods:

1) **value put(Object key, Object value)**:It will place a key and a value as a pair into map

2) **value remove(Object key)**: it will remove the specified key and it's corresponding value

3) **value get(Object key)**: it will return the value of the key that is specified

4) **Set keyset( )** : it will returns all the keys available in the map in the form of a set

5) **Collection values( )**: it will returns all the values available in the map in the form of a collection

6) **void clear( ):** it is used to remove all the key-value pairs from the map

7) **int size( ):** it will return the count of the number of key-value pairs available in the map

8) **boolean containsKey(Object key):** it will returns true if the specified key is available in the map

9) **boolean containsValue(Object value):** it will returns true if the specified value is available in the map

10) **boolean isEmpty( )** :it will return true if the map is empty

Example:

```
import   java.util.HashMap;
import   java.util.Collection;
import   java.util.Set;
import   java.util.Scanner;
class  HashMapEx1
{
        public static void main(String[] args)
        {
                Scanner   sc = new  Scanner(System.in);
                HashMap<Integer,String>  h1 = new  HashMap<Integer,String>();
                h1.put(101,"Venkat");
                h1.put(102,"Vishnu");
                h1.put(103,"Avinash");
                h1.put(104,"Lakshmi");
                h1.put(105,"Leena");
                System.out.println("Your Records are : "+h1);
                System.out.print("Enter Student id : ");
                int   sid = sc.nextInt();
                if(h1.containsKey(sid))
                                System.out.println(sid+" Value is : "+h1.get(sid));
                else
                        System.out.println(" Sorry..! No Record Found ");
                System.out.println(" All Student Ids are  ");
                Set   s = h1.keySet();
                System.out.println(s);
                System.out.println(" All Student Names are ");
                Collection  a = h1.values();
                System.out.println(a);
                System.out.println(" No.of Students are : "+h1.size());
        }
}
```

Example2:

```
import   java.util.HashMap;
import   java.util.ArrayList;
import   java.util.Scanner;
class  HashMapEx2
{         public static void main(String[] args)
          {
                    Scanner   sc = new  Scanner(System.in);
                    ArrayList<String>    a1 = new  ArrayList<String>() ;
                              a1.add("Venkat");
                              a1.add("10");
                              a1.add("HR");
                    ArrayList<String>    a2 = new  ArrayList<String>() ;
                              a2.add("Vishnu");
                              a2.add("20");
                              a2.add("HR");
HashMap<Integer,ArrayList>  h1 = new   ashMap<Integer,ArrayList>();
                              h1.put(101,a1);
                              h1.put(102,a2);
                    System.out.println("Your Records are : "+h1);
                    System.out.println(" No.of Students are : "+h1.size());
          }
}
```

Example3:

Ex:-

```
import java.util.HashMap;
import java.util.Scanner;
import java.util.Set;
import java.util.Collection;
class HashMapEx1
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
        HashMap<Integer,String> hm = new HashMap<Integer,String>( );
                hm.put(101,"Venkat");
                hm.put(102,"Vishnu");
                hm.put(103,"Lakshmi");
                hm.put(104,"Tara");
                hm.put(105,"Leena");
                System.out.println(" Your Map is : "+ hm);
                System.out.print(" Enter your key : ");
                int k = sc.nextInt( );
                if(hm.containsKey(k))
                {
                        String s = hm.get(k);
                        System.out.println( k+ " vlaue is : "+s);
                        hm.remove(k);
                       System.out.println(k+ " is Removed from map ");
                }
                else
                {
                        System.out.println(" Sorry..! "+ k +" is Not Available in Map ");
                }

                 Set s1 = hm.keySet( );
                 System.out.println(" All Keys are : "+s1);

                Collection c = hm.values( );
                 System.out.println(" All Values are : "+c);
        }
}
```

V E N K A T

R E D D Y

LinkedHashMap:

It is an implementation class of Map Interface

It will store elements in the form of key-value pairs

LinkedHashMap whatever the order we inserted same order elements are stored.

Syntax:

LinkedHashMap<K,V> hm1=new LikedHashMap<K,V>();

It will create an empty map with the default initial capacity as 16.

TreeMap:

TreeMap is an implementation class of Map interface

TreeMap does not guarantee the order of insertion

TreeMap sorts the keys in ascending order

Syntax:

TreeMap<K,V> tm=new TreeMap<K,V>();

It creates an empty map where the elements will be sorted in natural order

Hashtable:

Hashtable is an implementation class of Map interface

Hashtable doesnot allow null in both keys and values

Hashtable doesnot guarantee the order of insertion

Syntax:

Hashtable <K,V> ht= new Hashtable<K,V>( );

It creates an empty map with default initial capacity as 11.

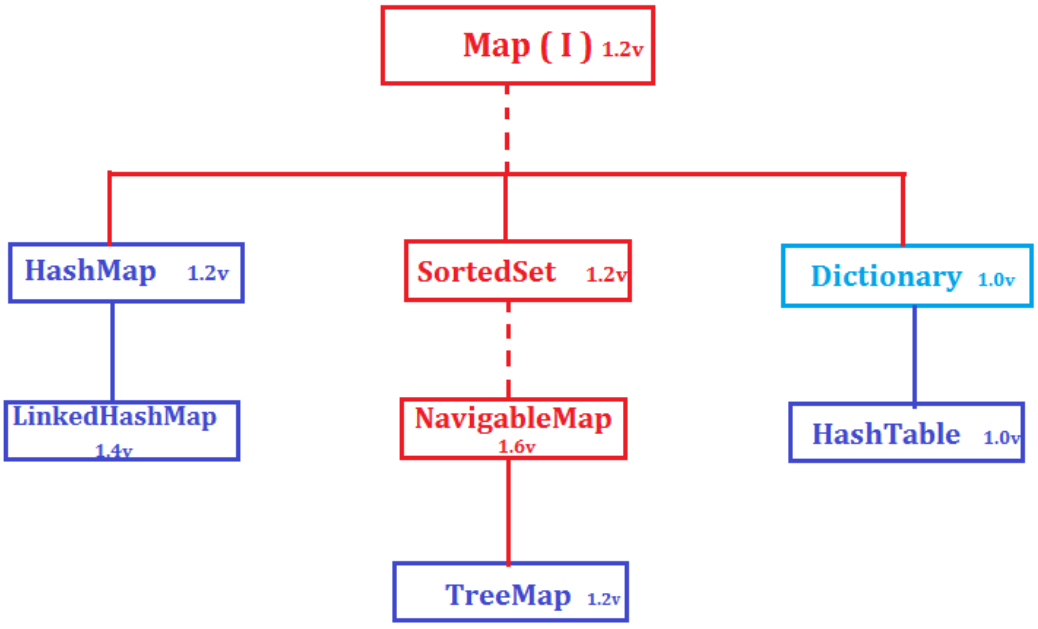
	HashMap	LinkedHashMap	TreeMap	Hashtable
Ordered	Un-ordered	Ordered by Insertion	Sorted Order	Un-ordered
Duplicates (values)	Allowed	Allowed	Allowed	Allowed
Null Value	Allowed	Allowed	Allowed	Not Allowed
Synchronized	Not Synchronized	Not Synchronized	Not Synchronized	Synchronized
Data Structure	Hashtable	Hashtable+ Double Linked list	Red-Black Tree	Hashtable
Initial Capacity	16	16	……..	11

Task

Train Number	Train Name
22204	DURONTO
12740	GARIBRATH
12796	INTERCITY
12728	GODAVARI
12710	SIMHAPURI
12704	FALAKNUMA
12714	SATAVAHANA

--> Read Train Number from customer, based on that display train name

--> Read Train number if available then delete that train, if not available then ask customer once again, enter Valid train number then delete train



Collections:

Collections is a class, which contains pre-defined static methods to perform operations on colletion

Methods:

1) void sort(List): it is used to sort the elements available in the list class
2) int binarySearch(List,Object): it is used to perform binary search operation on the elements of the list
3) void reverse(List): it is used to reverse the elements available in the list
4) void swap(List, int index1,int index2): it is used to swap the elements available in the specified index positions
5) void copy(List, List): it is used to copy the elements of one list to another list
6) Object min(Collection): it will return the smallest element available in the specified collection
7) Object max(Collection): it will return the largest element available in the specified collection

Example:

```
import java.util.ArrayList;
import java.util.Collections;
class CollectionsEx1
{
        public static void main(String[] args)
                        {
                         ArrayList<Integer>    al = new  ArrayList<Integer>() ;
                         al.add(53);
                         al.add(23);
                         al.add(57);
                         al.add(4);
                         al.add(12);
                         al.add(2);
                        System.out.println(" Your Array List  Elements are : "+al);
                System.out.println(" Maximum Value is  : +Collections.max(al));
                System.out.println(" Minimum value is   : "+Collections.min(al));
                        Collections.sort(al);
                        System.out.println(" Your Elements in Order : "+al);
                        Collections.reverse(al);
                        System.out.println(" Your Elements in Reverse Order : "+al);
                        Collections.swap(al,0,2);
                        System.out.println(" Your Elements are : "+al);
                        }
```

}

StringTokenizer:

This class is used to break string into Tokens

It is available in java.util package

Syntax:

StringTokenizer reference = new StringTokenizer(String, delimiter)

Methods:

int countTokens( ) : it counts number of tokens

boolean hasMoreTokens( ) : It checks next Token is available or not

Stirng nextToken( ) : it gives next Token

Ex:-

```
import java.util.StringTokenizer;
class StringTokenizerEx1
{
        public static void main(String[] args)
        {
                String s1 = "Java and Python Classes by Venkat Reddy in Sathya Technology";
                StringTokenizer s2 = new StringTokenizer(s1);
                int n = s2.countTokens();
                System.out.println(" Number of Tokens are : "+n);
                while(s2.hasMoreTokens( ))
                {
                        String s = s2.nextToken( );
                        System.out.println(s);
                }
        }
}
```

Random

Random is a class, which is available in java.util package

Method:

1) int nextInt(int):

It is used to generate random numbers based on given input

Example1:

```
import  java.util.Random;
class RandomEx1
{
        public static void main(String[] args)
        {
             Random   r  = new  Random();
             int  x = r.nextInt(100);
            System.out.println(x);
        }
}
```

Example2:

```
import  java.util.Random;
import  java.util.Scanner;
class RandomEx2
{
      public static void main(String[] args)
      {
         Scanner  sc = new Scanner(System.in);
         Random   r  = new  Random();
         int  x = r.nextInt(100);
         System.out.println("First Number is : "+x);
         int  y = r.nextInt(100);
         System.out.println("Second Number is : "+y);
        System.out.print(" What is Multiplication value : ");
        int  z = sc.nextInt( );
        if(z == (x*y))
                System.out.println(" Congratulation..! You Got 1 Lak ");
        else
               System.out.println(" Better Luck Next Time ");
      }
}
```

Java Practice Programs

S.No	Program
1	How to Print an Integer entered by an user
2	Program to Add Two Integers
3	Find ASCII value of a character
4	Compute Quotient and Remainder
5	Swap two numbers using temporary variable
6	Swap two numbers without using temporary variable
7	Check whether a number is even or odd using if...else statement
8	Check whether an alphabet is vowel or consonant using if..else statement
9	Check whether an alphabet is vowel or consonant using switch statement
10	Find Largest Among three numbers using if..else statement
11	Find the largest number among three using nested if..else statement
12	Find Frequency of Character
13	Program to Remove All Whitespaces
14	Round a Number using format
15	Check if String is Empty or Null
16	Check if String with spaces is Empty or Null
17	Program to Check a Leap Year
18	Check if a Number is Positive or Negative using if else
19	Sum of Natural Numbers using for loop
20	Sum of Natural Numbers using while loop
21	Find Factorial of a number using for loop
22	Find Factorial of a number using while loop
23	Generate Multiplication Table using for loop
24	Generate Multiplication Table using while loop
25	Display Fibonacci series using for loop
26	Display Fibonacci series using while loop
27	Display Uppercased A to Z using for loop
28	Display Lowercased a to z using for loop
29	Reverse a Number using a while loop
30	Program to Check Palindrome using while loop
31	Program to Check Prime Number using a for loop
32	Program to check given number is Amstrong or not
33	Simple Calculator using switch Statement
34	* * * * * *

	* * * * * * * * *
35	1 1 2 1 2 3 1 2 3 4 1 2 3 4 5
36	A B B C C C D D D D E E E E E
37	* * * * * * * * * * * * * * *
38	1 2 3 4 5 1 2 3 4 1 2 3 1 2 1
39	* * * * * * * * * * * * * * * *

	* * * * * * * * *
40	1 2 3 2 3 4 5 4 3 4 5 6 7 6 5 4 5 6 7 8 9 8 7 6 5
41	* *
42	1 1 1 1 2 1 1 3 3 1 1 4 6 4 1 1 5 10 10 5 1
43	1 2 3 4 5 6 7 8 9 10
44	Program to Calculate Average Using Arrays
45	Find largest element in an array
46	Program to Add Two Matrices
47	Program to MultiplyMatrices
48	Check if Int Array contains a given value

49	Convert String to char array
50	Convert char array to String
51	Create a new directory in Java
52	Create a new Directory using the mkdirs() method
53	List all files using list( ) method

1) How to Print an Integer entered by an user

```
import java.util.Scanner;
public class HelloWorld {
   public static void main(String[] args) {
      // Creates a reader instance which takes
      // input from standard input - keyboard
      Scanner reader = new Scanner(System.in);
      System.out.print("Enter a number: ");
      // nextInt() reads the next integer from the keyboard
      int number = reader.nextInt();
      // println() prints the following line to the output screen
      System.out.println("You entered: " + number);
   }
}
```


2) Program to Add Two Integers

```
class AddTwoIntegers {
   public static void main(String[] args) {

      int first = 10;
      int second = 20;
      int sum = first + second;
      System.out.println("The sum is: " + sum);
   }
}
```

3) Find ASCII value of a character

```
class AsciiValue {
   public static void main(String[] args) {
      char ch = 'a';
      int ascii = ch;
      // You can also cast char to int
      int castAscii = (int) ch;
      System.out.println("The ASCII value of " + ch + " is: " + ascii);
      System.out.println("The ASCII value of " + ch + " is: " + castAscii);
   }
}
```

4) Compute Quotient and Remainder

```
class QuotientRemainder {
   public static void main(String[] args) {
      int dividend = 25, divisor = 4;
      int quotient = dividend / divisor;
```

```
        int remainder = dividend % divisor;
        System.out.println("Quotient = " + quotient);
        System.out.println("Remainder = " + remainder);
    }
}
```

5) Swap two numbers using temporary variable

```
class SwapNumbers {
    public static void main(String[] args) {
        float first = 1.20f, second = 2.45f;
        System.out.println("--Before swap--");
        System.out.println("First number = " + first);
        System.out.println("Second number = " + second);
        // Value of first is assigned to temporary
        float temporary = first;
        // Value of second is assigned to first
        first = second;
        // Value of temporary (which contains the initial value of first) is assigned to
second
        second = temporary;
        System.out.println("--After swap--");
        System.out.println("First number = " + first);
        System.out.println("Second number = " + second);
    }
}
```

6) Swap two numbers without using temporary variable

```
class  SwapNumbers
{
    public static void main(String[] args) {
        float first = 12.0f, second = 24.5f;
        System.out.println("--Before swap--");
        System.out.println("First number = " + first);
        System.out.println("Second number = " + second);
        first = first - second;
        second = first + second;
        first = second - first;
        System.out.println("--After swap--");
        System.out.println("First number = " + first);
        System.out.println("Second number = " + second);
    }
}
```

7) Check whether a number is even or odd using if...else statement

```
import java.util.Scanner;
public class EvenOdd {
   public static void main(String[] args) {
      Scanner reader = new Scanner(System.in);
      System.out.print("Enter a number: ");
      int num = reader.nextInt();
      if(num % 2 == 0)
         System.out.println(num + " is even");
      else
         System.out.println(num + " is odd");
   }
}
```

8) Check whether an alphabet is vowel or consonant using if..else statement

```
class VowelConsonant {
   public static void main(String[] args) {
      char ch = 'i';
      if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' )
         System.out.println(ch + " is vowel");
      else
         System.out.println(ch + " is consonant");
   }
}
```

9) Check whether an alphabet is vowel or consonant using switch statement

```
class VowelConsonant {
   public static void main(String[] args) {
      char ch = 'z';
      switch (ch) {
         case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u':
            System.out.println(ch + " is vowel");
            break;
         default:
            System.out.println(ch + " is consonant");
      }
   }
}
```

10) Find Largest Among three numbers using if..else statement

```
class Largest {
   public static void main(String[] args) {
      double n1 = -4.5, n2 = 3.9, n3 = 2.5;
      if( n1 >= n2 && n1 >= n3)
         System.out.println(n1 + " is the largest number.");
      else if (n2 >= n1 && n2 >= n3)
         System.out.println(n2 + " is the largest number.");
      else
         System.out.println(n3 + " is the largest number.");
   }
}
```

11) Find the largest number among three using nested if..else statement

```
class Largest {
   public static void main(String[] args) {
      double n1 = -4.5, n2 = 3.9, n3 = 5.5;
      if(n1 >= n2) {
         if(n1 >= n3)
            System.out.println(n1 + " is the largest number.");
         else
            System.out.println(n3 + " is the largest number.");
      } else {
         if(n2 >= n3)
            System.out.println(n2 + " is the largest number.");
         else
            System.out.println(n3 + " is the largest number.");
      }
   }
}
```

12) Find Frequency of Character

```
class Frequency {
   public static void main(String[] args) {
      String str = "This website is awesome.";
      char ch = 'e';
      int frequency = 0;
      for(int i = 0; i < str.length(); i++) {
         if(ch == str.charAt(i)) {
            ++frequency;
         }
      }
      System.out.println("Frequency of " + ch + " = " + frequency);
   }
}
```

13) Program to Remove All Whitespaces

```
class Whitespaces {
   public static void main(String[] args) {
      String sentence = "T    his is b  ett     er.";
      System.out.println("Original sentence: " + sentence);
      sentence = sentence.replaceAll("\\s", "");
      System.out.println("After replacement: " + sentence);
   }
}
```

14) Round a Number using format

```
class Decimal {
   public static void main(String[] args) {
      double num = 1.34567;
      System.out.format("%.4f", num);
   }
}
```

15) Check if String is Empty or Null

```
class Null {
   public static void main(String[] args) {
      String str1 = null;
      String str2 = "";
      if(isNullOrEmpty(str1))
         System.out.println("First string is null or empty.");
      else
         System.out.println("First string is not null or empty.");
      if(isNullOrEmpty(str2))
         System.out.println("Second string is null or empty.");
      else
         System.out.println("Second string is not null or empty.");
   }
   public static boolean isNullOrEmpty(String str) {
      if(str != null && !str.isEmpty())
         return false;
      return true;
   }
}
```

16) Check if String with spaces is Empty or Null

```
public class Null {
   public static void main(String[] args) {
      String str1 = null;
      String str2 = "   ";
      if(isNullOrEmpty(str1))
         System.out.println("str1 is null or empty.");
      else
         System.out.println("str1 is not null or empty.");
      if(isNullOrEmpty(str2))
         System.out.println("str2 is null or empty.");
      else
         System.out.println("str2 is not null or empty.");
   }
   public static boolean isNullOrEmpty(String str) {
      if(str != null && !str.trim().isEmpty())
         return false;
      return true;
   }
}
```

17) Program to Check a Leap Year

```
public class LeapYear {
   public static void main(String[] args) {
      int year = 1900;
      boolean leap = false;
      if(year % 4 == 0)
      {
         if( year % 100 == 0)
         {
            // year is divisible by 400, hence the year is a leap year
            if ( year % 400 == 0)
               leap = true;
            else
               leap = false;
         }
         else
            leap = true;
      }
      else
         leap = false;
      if(leap)
         System.out.println(year + " is a leap year.");
```

```
      else
         System.out.println(year + " is not a leap year.");
   }
}
```

18) Check if a Number is Positive or Negative using if else

```
public class PositiveNegative {
   public static void main(String[] args) {
      double number = 12.3;
      // true if number is less than 0
      if (number < 0.0)
         System.out.println(number + " is a negative number.");
      // true if number is greater than 0
      else if ( number > 0.0)
         System.out.println(number + " is a positive number.");
      // if both test expression is evaluated to false
      else
         System.out.println(number + " is 0.");
   }
}
```

19) Sum of Natural Numbers using for loop

```
class SumNatural {
   public static void main(String[] args) {
      int num = 100, sum = 0;
      for(int i = 1; i <= num; ++i)
      {
         // sum = sum + i;
         sum += i;
      }
      System.out.println("Sum = " + sum);
   }
}
```

20) Sum of Natural Numbers using while loop

```
class SumNatural {
   public static void main(String[] args) {
      int num = 50, i = 1, sum = 0;
      while(i <= num)
      {
         sum += i;
         i++;
      }
      System.out.println("Sum = " + sum);
   }
}
```

21) Find Factorial of a number using for loop

```
class Factorial {
   public static void main(String[] args) {
      int num = 10;
      long factorial = 1;
      for(int i = 1; i <= num; ++i)
      {
         // factorial = factorial * i;
         factorial *= i;
      }
      System.out.printf("Factorial of %d = %d", num, factorial);
   }
}
```

22) Find Factorial of a number using while loop

```
class Factorial {
   public static void main(String[] args) {
      int num = 5, i = 1;
      long factorial = 1;
      while(i <= num)
      {
         factorial *= i;
         i++;
      }
      System.out.printf("Factorial of %d = %d", num, factorial);
   }
}
```

23) Generate Multiplication Table using for loop

```
public class MultiplicationTable {
   public static void main(String[] args) {
      int num = 5;
      for(int i = 1; i <= 10; ++i)
      {
         System.out.printf("%d * %d = %d \n", num, i, num * i);
      }
   }
}
```

24) Generate Multiplication Table using while loop

```
public class MultiplicationTable {
   public static void main(String[] args) {
      int num = 9, i = 1;
      while(i <= 10)
      {
         System.out.printf("%d * %d = %d \n", num, i, num * i);
```

```
        i++;
      }
    }
}
```

25) Display Fibonacci series using for loop

```
public class Fibonacci {
   public static void main(String[] args) {
      int n = 10, t1 = 0, t2 = 1;
      System.out.print("First " + n + " terms: ");
      for (int i = 1; i <= n; ++i)
      {
         System.out.print(t1 + " + ");
         int sum = t1 + t2;
         t1 = t2;
         t2 = sum;
      }
   }
}
```

26) Display Fibonacci series using while loop

```
public class Fibonacci {
   public static void main(String[] args) {
      int i = 1, n = 10, t1 = 0, t2 = 1;
      System.out.print("First " + n + " terms: ");
      while (i <= n)
      {
         System.out.print(t1 + " + ");
         int sum = t1 + t2;
         t1 = t2;
         t2 = sum;
         i++;
      }
   }
}
```

27) Display Uppercased A to Z using for loop

```
public class Characters {
   public static void main(String[] args) {
      char c;
      for(c = 'A'; c <= 'Z'; ++c)
         System.out.print(c + " ");
   }
}
```

28) Display Lowercased a to z using for loop

```
public class Characters {
   public static void main(String[] args) {
      char c;
      for(c = 'a'; c <= 'z'; c++)
         System.out.print(c + " ");
   }
}
```

29) Reverse a Number using a while loop

```
public class ReverseNumber {
   public static void main(String[] args) {
      int num = 1234, reversed = 0;
      while(num != 0) {
         int digit = num % 10;
         reversed = reversed * 10 + digit;
         num /= 10;
      }
      System.out.println("Reversed Number: " + reversed);
   }
}
```

30) Program to Check Palindrome using while loop

```
public class Palindrome {
   public static void main(String[] args) {
      int num = 121, reversedInteger = 0, remainder, originalInteger;
      originalInteger = num;
      // reversed integer is stored in variable
      while( num != 0 )
      {
         remainder = num % 10;
         reversedInteger = reversedInteger * 10 + remainder;
         num  /= 10;
      }
      // palindrome if orignalInteger and reversedInteger are equal
      if (originalInteger == reversedInteger)
         System.out.println(originalInteger + " is a palindrome.");
      else
         System.out.println(originalInteger + " is not a palindrome.");
   }
}
```

31) Program to Check Prime Number using a for loop

```
class Prime {
   public static void main(String[] args) {
      int num = 29;
      boolean flag = false;
      for(int i = 2; i <= num/2; ++i)
      {
         // condition for nonprime number
         if(num % i == 0)
         {
            flag = true;
            break;
         }
      }
      if (!flag)
         System.out.println(num + " is a prime number.");
      else
         System.out.println(num + " is not a prime number.");
   }
}
```

32) Program to check given number is Amstrong or not

Program:

```
import   java.util.Scanner;
class Armstrong
{
   public static void main(String[ ] args)
           {
         int number , originalNumber, remainder, result = 0;
               Scanner   sc = new  Scanner(System.in);
                     System.out.println(" Enter any Number : ");
                     number = sc.nextInt( );
        originalNumber = number;
        while (originalNumber != 0)
        {
          remainder = originalNumber % 10;
          originalNumber = originalNumber/10;
          result =result+(remainder*remainder*remainder);
        }
          if(result == number)
          System.out.println(number + " is an Armstrong number.");
          else
          System.out.println(number + " is not an Armstrong number.");
   }
```

```
}
```

33) Simple Calculator using switch Statement

Program:

```
import java.util.Scanner;
public class Calculator {
   public static void main(String[] args) {
      Scanner reader = new Scanner(System.in);
      System.out.print("Enter two numbers: ");
      // nextDouble() reads the next double from the keyboard
      double first = reader.nextDouble();
      double second = reader.nextDouble();
      System.out.print("Enter an operator (+, -, *, /): ");
      char operator = reader.next().charAt(0);
      double result;
      switch(operator)
      {
         case '+':
            result = first + second;
            break;
         case '-':
            result = first - second;
            break;
         case '*':
            result = first * second;
            break;
         case '/':
            result = first / second;
            break;
         // operator doesn't match any case constant (+, -, *, /)
         default:
            System.out.printf("Error! operator is not correct");
            return;
      }
      System.out.printf("%.1f %c %.1f = %.1f", first, operator, second, result);
   }
}
```

34) Program to print half pyramid

```
*
* *
* * *
* * * *
* * * * *
```

Program:

```
public class Pattern
{
   public static void main(String[] args)
        {
                        int rows = 5;
             for(int i = 1; i <= rows; ++i)
                        {
                                for(int j = 1; j <= i; ++j)
                                {
                                        System.out.print("*  ");
                                }
                                System.out.println();
                        }
         }
}
```

35) Program to print half pyramid a using numbers

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Program:

```
public class Pattern
{
   public static void main(String[ ] args)
         {
      int rows = 5;
     for(int i = 1; i <= rows; ++i)
                  {
        for(int j = 1; j <= i; ++j)
                        {
            System.out.print(j + "  ");
         }
        System.out.println( );
      }
   }
```

```
}
```

36) Program to print half pyramid using alphabets

```
A
B B
C C C
D D D D
E E E E E
```


Source Code

```
public class Pattern {
   public static void main(String[] args) {
      char last = 'E', alphabet = 'A';
      for(int i = 1; i <= (last-'A'+1); ++i) {
         for(int j = 1; j <= i; ++j) {
            System.out.print(alphabet + " ");
         }
         ++alphabet;
         System.out.println();
      }
   }
}
```

37) Inverted half pyramid using *

```
* * * * *
* * * *
* * *
* *
*
```

Source Code

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 5;
      for(int i = rows; i >= 1; --i) {
         for(int j = 1; j <= i; ++j) {
            System.out.print("* ");
         }
         System.out.println();
      }
   }
}
```

38) Inverted half pyramid using numbers

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Source Code

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 5;
      for(int i = rows; i >= 1; --i) {
         for(int j = 1; j <= i; ++j) {
            System.out.print(j + " ");
         }
         System.out.println();
      }
   }
}
```

39) Program to print full pyramid using *

```
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
```

Source Code

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 5, k = 0;
      for(int i = 1; i <= rows; ++i, k = 0) {
         for(int space = 1; space <= rows - i; ++space) {
            System.out.print("  ");
         }
         while(k != 2 * i - 1) {
            System.out.print("* ");
            ++k;
         }
         System.out.println();
      }
   }
}
```

40) Program to print pyramid using numbers

```
    1
  2 3 2
 3 4 5 4 3
4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
```

Source Code

```
public class Pattern {
  public static void main(String[] args) {
    int rows = 5, k = 0, count = 0, count1 = 0;
    for(int i = 1; i <= rows; ++i) {
      for(int space = 1; space <= rows - i; ++space) {
        System.out.print("  ");
        ++count;
      }
      while(k != 2 * i - 1) {
        if (count <= rows - 1) {
          System.out.print((i + k) + " ");
          ++count;
        }
        else {
          ++count1;
          System.out.print((i + k - 2 * count1) + " ");
        }
        ++k;
      }
      count1 = count = k = 0;
      System.out.println();
    }
  }
}
```

41) Inverted full pyramid using *

```
*********
 *******
  *****
   ***
    *
```

Source Code

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 5;
      for(int i = rows; i >= 1; --i) {
         for(int space = 1; space <= rows - i; ++space) {
            System.out.print("  ");
         }
         for(int j=i; j <= 2 * i - 1; ++j) {
            System.out.print("* ");
         }
         for(int j = 0; j < i - 1; ++j) {
            System.out.print("* ");
         }
         System.out.println();
      }
   }
}
```

42) Print Pascal's triangle

```
      1
     1  1
    1  2  1
   1  3  3  1
  1  4  6  4  1
 1  5  10  10 5  1
```

Source Code

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 6, coef = 1;
      for(int i = 0; i < rows; i++) {
         for(int space = 1; space < rows - i; ++space) {
            System.out.print("  ");
         }
```

```
        for(int j = 0; j <= i; j++) {
          if (j == 0 || i == 0)
            coef = 1;
          else
            coef = coef * (i - j + 1) / j;
          System.out.printf("%4d", coef);
        }
        System.out.println();
      }
    }
}
```


43) Print Floyd's Triangle.

```
1
2 3
4 5 6
7 8 9 10
```

Source Code

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 4, number = 1;
      for(int i = 1; i <= rows; i++) {
         for(int j = 1; j <= i; j++) {
            System.out.print(number + " ");
            ++number;
         }
         System.out.println();
      }
   }
}
```

44) Program to Calculate Average Using Arrays

```
public class Average {
   public static void main(String[] args) {
      double[] numArray = { 45.3, 67.5, -45.6, 20.34, 33.0, 45.6 };
      double sum = 0.0;
      for (double num: numArray) {
        sum += num;
      }
      double average = sum / numArray.length;
      System.out.format("The average is: %.2f", average);
   }
}
```

45) Find largest element in an array

```
public class Largest {
   public static void main(String[] args) {
      double[] numArray = { 23.4, -34.5, 50.0, 33.5, 55.5, 43.7, 5.7, -66.5 };
      double largest = numArray[0];
      for (double num: numArray) {
         if(largest < num)
            largest = num;
      }
      System.out.format("Largest element = %.2f", largest);
   }
}
```

46) Program to Add Two Matrices

```
public class AddMatrices {
   public static void main(String[] args) {
      int rows = 2, columns = 3;
      int[ ][ ] firstMatrix = { {2, 3, 4}, {5, 2, 3} };
      int[ ][ ] secondMatrix = { {-4, 5, 3}, {5, 6, 3} };
      // Adding Two matrices
      int[][] sum = new int[rows][columns];
      for(int i = 0; i < rows; i++) {
         for (int j = 0; j < columns; j++) {
            sum[i][j] = firstMatrix[i][j] + secondMatrix[i][j];
         }
      }
      // Displaying the result
      System.out.println("Sum of two matrices is: ");
      for(int[] row : sum) {
         for (int column : row) {
            System.out.print(column + "    ");
         }
         System.out.println();
      }
   }
}
```

47)

```
public class MultiplyMatrices {
   public static void main(String[] args) {
      int r1 = 2, c1 = 3;
      int r2 = 3, c2 = 2;
      int[][] firstMatrix = { {3, -2, 5}, {3, 0, 4} };
      int[][] secondMatrix = { {2, 3}, {-9, 0}, {0, 4} };
      // Mutliplying Two matrices
      int[][] product = new int[r1][c2];
```

```
      for(int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
          for (int k = 0; k < c1; k++) {
            product[i][j] += firstMatrix[i][k] * secondMatrix[k][j];
          }
        }
      }
      // Displaying the result
      System.out.println("Sum of two matrices is: ");
      for(int[] row : product) {
        for (int column : row) {
          System.out.print(column + "    ");
        }
        System.out.println();
      }
   }
}
```


48) Check if Int Array contains a given value

```
public class Contains {
   public static void main(String[] args) {
      int[ ] num = {1, 2, 3, 4, 5};
      int toFind = 3;
      boolean found = false;
      for (int n : num) {
        if (n == toFind) {
          found = true;
           break;
        }
      }
      if(found)
        System.out.println(toFind + " is found.");
      else
        System.out.println(toFind + " is not found.");
   }
}
```

49) Convert String to char array

```
import java.util.Arrays;
public class StringChar {
   public static void main(String[] args) {
      String st = "This is great";
      char[] chars = st.toCharArray();
      System.out.println(Arrays.toString(chars));
   }
}
```

50) Convert char array to String

```
public class CharString {
   public static void main(String[] args) {
      char[] ch = {'a', 'e', 'i', 'o', 'u'};
      String st = String.valueOf(ch);
      String st2 = new String(ch);
      System.out.println(st);
      System.out.println(st2);
   }
}
```

51) Create a new directory in Java

```
import java.io.File;
class Main {
  public static void main(String[] args) {
    // creates a file object with specified path
    File file = new File("Java Example\\directory");
    // tries to create a new directory
    boolean value = file.mkdir();
    if(value) {
      System.out.println("The new directory is created.");
    }
    else {
      System.out.println("The directory already exists.");
    }
  }
}
```

52) Create a new Directory using the mkdirs() method

```
import java.io.File;
class Main {
  public static void main(String[] args) {
    // creates a file object in current path
    File file = new File("Java Tutorial\\abc");
    // tries to create a new directory
    boolean value = file.mkdirs();
```

```
    if(value) {
      System.out.println("The new directory is created.");
    }
    else {
      System.out.println("The directory already exists.");
    }
  }
}
```


53) List all files using list() method

```
import java.io.File;
class Main {
  public static void main(String[] args) {
    // creates a file object
    File file = new File("C:\\Users\\Guest User\\Desktop\\Java File\\List Method");
    // returns an array of all files
    String[] fileList = file.list();
    for(String str : fileList) {
      System.out.println(str);
    }
  }
}
```