JAVA CORE

INDEX: -

INTRODUCTION: -

Java is a programming language. Java is a high level, robust, object-oriented and secure programming language. Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is intended to let application developers write once, and run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

It is widely used for developing applications for desktop, web, and mobile devices. Java is known for its simplicity, robustness, and security features, making it a popular choice for enterprise-level applications.

HISTORY: - Java was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s. James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991.

The small team of sun engineers called Green Team. Initially it was designed for small, embedded systems in electronic appliances like set-top boxes. Firstly, it was called "Greentalk" by James Gosling, and the file extension was .gt.After that, it was called Oak and was developed as a part of the Green project. Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc. In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies. Java is an island in Indonesia where the first coffee was produced (called Java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having a cup of coffee nearby his office.

Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995. In 1995, Time magazine called Java one of the Ten Best Products of 1995. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds new features in Java.

APPLICATION: - According to Sun, 3 billion devices run Java. There are many devices where Java is currently used.

- Desktop Applications such as acrobat reader, media player, antivirus, etc.
- Web Applications such as irctc.co.in, etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games, etc.

FEATURES:-

1. Platform Independent: Compiler converts source code to bytecode. This bytecode can run on any platform be it Windows, Linux, or macOS which means if we compile a program on Windows, then we can run it on Linux and vice versa. Each operating system has a different JVM that will execute bytecode, but the output produced by all the OS is the same after the execution of the bytecode. That is why we call java a platform-independent language.

2. Object-Oriented Programming Language: Organizing the program in the terms of a collection of objects is a way of object-oriented programming, each of which represents an instance of the class. The four main concepts of Object-Oriented programming are:

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

3. Simple: Java is one of the simple languages as it does not have complex features like pointers, operator overloading, multiple inheritances, and explicit memory allocation.
4. Robust: Java language is robust which means reliable. It is developed in such a way that it puts a lot of effort into checking errors as early as possible that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, Exception Handling, and memory allocation.
5. Secure: java don't have pointers, so it cannot access out-of-bound arrays i.e it shows **ArrayIndexOutOfBound** Exception if we try to do so. That's why several security flaws like stack corruption or buffer overflow are impossible to exploit in Java. Also, java programs run in an environment that is independent of the os(operating system) environment which makes java programs more secure.
6. Distributed: it create distributed applications using the java programming language. Remote Method Invocation and Enterprise Java Beans are used for creating distributed applications in java. The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.
7. Multithreading: Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of the CPU.
8. Portable: As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.
9. High Performance: Java architecture is defined in such a way that it reduces overhead during the runtime and at some times java uses Just In Time (JIT) compiler where the compiler compiles code on-demand basics where it only compiles those methods that are called making applications to execute faster.
10. Dynamic flexibility: Java being completely object-oriented gives us the flexibility to add classes, new methods to existing classes, and even create new classes through sub-classes. Java even supports functions written in other languages such as C, C++ which are referred to as native methods.
11. Write Once Run Anywhere: As discussed above java application generates a '.class' file that corresponds to our applications (program) but contains code in binary format. It provides ease t architecture-neutral ease as bytecode is not dependent on any machine architecture. It is the primary reason java is used in the enterprising IT industry globally worldwide.
12. Power of compilation and interpretation: Most languages are designed with the purpose of either they are compiled language or they are interpreted language. But java integrates arising enormous power as Java compiler compiles the source code to bytecode and JVM executes this bytecode to machine OS-dependent executable code.

FIRST STEP

PROGRAM: - execute the first program in java.

1. First download and install the JDK (Java Development Kit).
2. Write java code using any text editor as :-
   ```
   class hello
   {
      public static void main(String args[])
      {
         System.out.println("hello world");
      }
   }
   ```
3. Save file using .java extension.
4. Open cmd (command prompt), type “cd” and path to folder that file is saved
   Example: - cd java1\folder2
   And press enter. Then in cmd it will show entered folder
   Example: - c:\java1\folder2>
5. Then type “javac” after it filename with .java extension and press enter.
   Example: - javac firstprog.java
6. It will compile java source file and generate class file.
7. Then type “java” and class name that contain main function and press enter.
   Example: - java hello
8. Then it will show output of code.

Basic information about java program code:-

- **Class:** class keyword is used to declare classes in Java
- **Public:** It is an access specifier. Public means this function is visible to all.
- **Static:** static is again a keyword used to make a function static. To execute a static function you do not have to create an Object of the class. The main() method here is called by JVM, without creating any object for class.
- **Void:** It is the return type, meaning this function will not return anything.
- **Main:** main() method is the most important method in a Java program. This is the method which is executed, hence all the logic must be inside the main() method. If a java class is not having a main() method, it causes compilation error.
- **String args[] :** This is used to signify that the user may option to enter parameters to the Java Program at command line. Use of both String[] args or String args[] is acceptable. Java compiler would accept both forms.
- **System.out.println:** This is used to print anything on the console.
- **"Hello world":** it is string that passed to method.
- **; -** it is used for terminate the statement.

COMMENTS: - these are used for write information about program or block of code. When Want to display output without some part of code then comments can be used.

Single-line comment: - it is used for comment single line using double forward slash “//”.
Example: - //this is single-line comment

Multi-line comment: - it is used for comment in multiple lines using forward slash asterisk and asterisk forward slash “/* … */”.
Example: -
```
/*
        This is multi-line comment
*/
```

VARIABLES: - these are containers that used for store values of several types. It is declare with data-type that will store value.
The general rules for naming variables are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter and it cannot contain whitespace.
- Names can also begin with $ and _.
- Names are case sensitive, uppercase and lowercase letters consider differently.
- Reserved words (keywords) cannot be used as names.

Example: - name, $good, $Good, num1, etc…

KEYWORDS: - there are some reserved words in java with special purpose.

abstract	default	Goto	package	strictfp	Float	volatile
Assert	do	if	private	Super	native	While
boolean	double	implements	Protected	Switch	Short	Const
Break	else	Import	Public	synchronized	Var	For
Byte	Enum	Instanceof	requires	This	Class	New
Case	Exports	Int	return	Throw	Finally	Static
Catch	Extends	Interface	short	throws	module	Void
char	Final	long	Static	transient	Return	try

DATA TYPES: - these are used to declare which type of data variable will stores. There are two types of data types are primitive (built-in) and non-primitive (user-defined) data types.
Primitive data types: - these are built-in data types.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Example:-

```
class j2datatypes
{
   public static void main(String args[])
   {
          byte $byte=45;
         System.out.println("Byte = "+$byte);
         short $short=567;
         System.out.println("Short = "+$short);
         int $int=4556;
         System.out.println("Int = "+$int);
         long $long=65279;
         System.out.println("Long = "+$long);
         float $float=45.78f;
         System.out.println("Float = "+$float);
         double $double=45.7834;
         System.out.println("Double = "+$double);
         boolean $boolean=true;
         System.out.println("boolean = "+$boolean);
         char $char='D';
         System.out.println("Char = "+$char);
   }
}
```

Non-primitive data types: - these are user-defined data types. String, Array, Class are non-primitive data types.

String: - it is used for print character of array.

Example:-

```
class j2datatypes
{
   public static void main(String args[])
   {
          String $string="hello";
          System.out.println("String = "+$string);
   }
}
```

CONSTANT: - these are values that do not change during program execution. "final" keyword is used for declare constant in java. It is called final variable also.

Syntax:-

Final datatype constant_name =value;

Example:-

final float PI=3.14f;

INPUT AND OUTPUT

OUTPUT: - for display output on console, it will need “out” stream from system class.

Streams: - these are used for flow data from program to console and console to program.

Print: - it is method used for display output on console.

Example:-

```
class j4output
{
	public static void main(String args[])
	{
		System.out.print("hello here");
	}
}
```

System: - it is class that contains methods and streams.

Out: - it is standard output stream for display output that flow data from program to console.

Print: - it is method that will help print output on console.

Println: - it is method used for display output with adding new line at end.

Example:-

```
class j4output
{
	public static void main(String args[])
	{
		String name="ram";
		System.out.println("hello "+name);
	}
}
```

+: - plus sign is used for show variable value with constant string value.

Escape sequences: - there are some important escape sequences that can be used in program. It is also called special characters. Backslash (/) is used for special character.

Name	**Escape sequence**	**Description**
New line	\n	It is used for take new line on console.
Tab	\t	It is used for more space between characters.
Backspace	\b	It is used for remove single character from output.
Backslash	\\	It is used for print backslash.
Double quotes	\"	It is used for print double quotes in output.

INPUT: - for take input from user it will need to import “util” package from java.

Package: - it is collection of classes, methods.

Scanner class:-

Example:-

```
import java.util.Scanner;
class j3userinput
```

```
{
        public static void main(String args[])
        {
                Scanner input= new Scanner(System.in);
                System.out.print("Enter name ");
                String name = input.next();
                System.out.println("your name is "+name);
        }
}
```

Import: - it is keyword for used import or includes packages and classes in program.
Java.util: - it is built-in utility package for used classes from that package.
Scanner: - it is built-in class for take user input.
New: - it is keyword for used take new instance of class.
Scanner input = new Scanner(System.in); :- making new object of class Scanner.
System.in: - it is standard input stream that flow data console to program.
String name = input.next(); :- it takes input from user and assign it to variable.
Input.next():- it is method from Scanner class, that access using input object.

Input methods: - there are several methods of Scanner for take data from user in different types.

Methods	**Description**
next()	It is used for take string input.
nextByte()	It is used for take byte input.
nextShort()	It is used for take short input.
nextInt()	It is used for take int input.
nextLong()	It is used for take long input.
nextFloat()	It is used for take float input.
nextDouble()	It is used for take double input.
nextBoolean()	It is used for take boolean input.
next().charAt(0)	It is used for take char input, with charAt(0) it take first character from input.
nextLine()	It is used for take string input with space. Eg. Enter name: - abc xyz

Example:-

```
import java.util.Scanner;
class j3userinput
{
        public static void main(String args[])
        {
                Scanner input= new Scanner(System.in);

                System.out.print("Enter string ");
                String $string = input.next();   //string input
                System.out.println("String = "+$string);

                System.out.print("Enter byte ");
                byte $byte = input.nextByte();          //byte input
                System.out.println("Byte = "+$byte);

                System.out.print("Enter short ");
                short $short = input.nextShort();       //short input
                System.out.println("Short = "+$short);

                System.out.print("Enter int ");
                int $int = input.nextInt();             //int input
                System.out.println("Int = "+$int);

                System.out.print("Enter long ");
                long $long = input.nextLong();          //long input
                System.out.println("Long = "+$long);

                System.out.print("Enter float ");
                float $float = input.nextFloat();       //float input
                System.out.println("Float = "+$float);

                System.out.print("Enter double ");
                double $double = input.nextDouble();            //double input
                System.out.println("Double = "+$double);

                System.out.print("Enter boolean ");
                boolean $bool = input.nextBoolean();            //boolean input
                System.out.println("Boolean = "+$bool);

                System.out.print("Enter char ");
                char $char = input.next().charAt(0);    //char input
                System.out.println("Char = "+$char);
        }
}
```

OPERATORS

ARITHMATIC OPERATORS: - these are used for perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x – y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y

Example:-

```
import java.util.Scanner;
class j5arithops
{
        public static void main(String args[])
        {
                Scanner in = new Scanner(System.in);
                int a,b,c;
                System.out.print("Enter num1 ");
                a=in.nextInt();
                System.out.print("Enter num2 ");
                b=in.nextInt();
                System.out.println("Num1 = "+a);
                System.out.println("Num2 = "+b);
                c=a+b;
                System.out.println("Addition = "+c);
                c=a-b;
                System.out.println("Subtraction = "+c);
                c=a*b;
                System.out.println("Multiplication = "+c);
                c=a/b;
                System.out.println("Division = "+c);
                c=a%b;
                System.out.println("Modulus = "+c);
        }
}
```

RELATIONAL OPERATORS: - these are used for comparison between two values. Result of them is true or false. These operators are also known as comparison operators.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y

>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Example:-

```
class j6relops
{
	public static void main(String args[])
	{
		int a=7,b=8;
		boolean c;
		c=a>b;
		System.out.println(a+" > "+b+" = "+c);
		c=a>=b;
		System.out.println(a+" >= "+b+" = "+c);
		c=a<b;
		System.out.println(a+" < "+b+" = "+c);
		c=a<=b;
		System.out.println(a+" <= "+b+" = "+c);
		c=a==b;
		System.out.println(a+" == "+b+" = "+c);
		c=a!=b;
		System.out.println(a+" != "+b+" = "+c);
	}
}
```

LOGICAL OPERATORS: - these operators are used for determine logic between two or multiple conditions.

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result returns false if the result is true	!(x < 5 && x < 10)

Example:-

```
class j7logicalops
{
	public static void main(String args[])
	{
		int a=3,b=7;
		boolean c;
		c=a<5&&a<7;
		System.out.println(a+" < 5 && "+a+"< 7 is "+c);
		c=a<5||b<5;
		System.out.println(a+" < 5 || "+b+"< 5 is "+c);
		c=!(a < 5 && b < 10);
		System.out.println("!("+a+" < 5 && "+b+" < 10) is "+c);
	}
}
```

BITWISE OPERATORS: - these operators convert decimal values in binary form and perform logical operations on them and return result.

operator	name	Description	Example
&	Bitwise and	Perform and operation bitwise on values	X&Y
|	Bitwise or	Perform or operation bitwise on values	X|Y
^	Bitwise xor	Perform xor operation bitwise on values	X^Y
~	Bitwise negation	Perform negation operation bitwise on values	~X
<<	Bitwise left shift	Perform left shift operation bitwise on values	X<<Y
>>	Bitwise right shift	Perform right shift operation bitwise on values	X>>Y

Example:-

```
class j8bitwiseops
{
	public static void main(String args[])
	{
		int a=5,b=7;
		int c;
		c=a&b;
		System.out.println(a+" & "+b+" is "+c);
		c=a|b;
		System.out.println(a+" | "+b+" is "+c);
		c=a^b;
		System.out.println(a+" ^ "+b+" is "+c);
		c=a<<b;
		System.out.println(a+" << "+b+" is "+c);
		c=a>>b;
		System.out.println(a+" >> "+b+" is "+c);
		c=~a;
		System.out.println("~ "+a+" is "+c);
	}
}
```

ASSIGNMENT OPERATORS: - these operators assign value to variables with arithmetic and bitwise operations.

Operator	Name	Example	Same As
=	Assignment	x = 5	x = 5

+=	add and assign	x += 3	x = x + 3
-=	subtract and assign	x -= 3	x = x – 3
*=	multiply and assign	x *= 3	x = x * 3
/=	divide and assign	x /= 3	x = x / 3
%=	modulus and assign	x %= 3	x = x % 3

Example:-

```
class j9assignops
{
        public static void main(String args[])
        {
                int a=8,b=5;

                System.out.println("a= "+a+"\nb= "+b);
                a+=b;
                System.out.println("a+=b = "+a);
                a*=b;
                System.out.println("a*=b = "+a);
                a-=b;
                System.out.println("a-=b = "+a);
                a/=b;
                System.out.println("a*=b = "+a);
        }
}
```

INCREMENT OPERATORS: - it is unary operator. It increases value by 1.

Pre-increment operator: - it first increases value by 1 then reference the variable.
Example: - ++a;
Post-increment operator: - it first references the variable then increase value by 1.
Example: - a++;

Example:-

```
class j10increops
{
        public static void main(String args[])
        {
                int a=56;

                System.out.println("a = "+a);
                ++a;
                System.out.println("++a = "+a);
                a++;
                System.out.println("a++ = "+a);
        }
}
```

DECREMENT OPERATORS: - it is unary operator. It decreases value by 1.

Pre-decrement operator: - it first decreases value by 1 then reference the variable.
Example: - --a;
Post-decrement operator: - it first references the variable then decrease value by 1.
Example: - a--;

Example:-
```
class j10decreops
{
        public static void main(String args[])
        {
                int a=45;

                System.out.println("a = "+a);
                --a;
                System.out.println("--a = "+a);
                a--;
                System.out.println("a-- = "+a);
        }
}
```

CONTROL STATEMENTS

DECISION MAKING STATEMENT: - these are used for execute block of code, when particular condition is true.

If statement: - it will execute the block of code when condition is true.

Syntax:-

```
If(condition)
{
        //code of statement
}
```

Example:-

```
class j11if
{
        public static void main(String args[])
        {
                int a=30;
                if(a>10)
                {
                        System.out.println("a is greater than 10");
                }
        }
}
```

If-else statement: - it execute block of code if condition is true. If condition is not true then it execute else block of code.

Syntax:-

```
If(condition)
{
        //block of code
}
Else
{
        //block of code
}
```

Example:-

```
class j11if
```

```
{
        public static void main(String args[])
        {
                int age=34;
                if(age>18)
                {
                        System.out.println("eligible for voting");
                }
                else
                {
                        System.out.println("not eligible for voting");
                }
        }
}
```

Else-if ladder: - it is used for checking multiple conditions. it will execute new condition if previous is false.

Syntax:-

```
If(condition1)
{
        //block of code
}
Else if(condition2)
{
        //block of code
}
Else
{
        //block of code
}
```

Example:-

```
class j12elseifladder
{
        public static void main(String args[])
        {
                int percentage=75;
                if(percentage>=90)
                {
                        System.out.println("A+ class");
                }
                else if(percentage>=80)
                {
                        System.out.println("B+ class");
                }
                else if(percentage>=70)
                {
                        System.out.println("c+ class");
                }
                else
                {
                        System.out.println("good work");
                }
        }
}
```

Switch statement: - this statement selects one of many code blocks to be executed. Use case and value with (:) colon then write code after colon. In switch statement expression value will compare with case, if there is match then associated case block will be executed. Break used for exit from switch statement. Default will execute if none case match to expression. In switch statement numerous cases can write. Switch, case, default and break are key

Syntax:-

```
Switch(expression)
{
        Case 1: block of code
                Break;
        Case 2: block of code
                Break;
        …
        …
        Case n: block of code
                Break;
        Default: block of code
}
```

Example:-

```
class j13switch
{
        public static void main(String args[])
        {
                int a=0;

                switch(a)
                {
                        case 1: System.out.println("Rank 1st");
                        break;
                        case 2: System.out.println("Rank 2nd");
                        break;
                        case 3: System.out.println("Rank 3rd");
                        break;
                        default: System.out.println("No rank");
                }
        }
}
```

LOOPING STATEMENT: - these are used for execute block of code for multiple time.

While: - it is also called entry controlled loop, because it first checks condition, then execute block of code.

Syntax:-

```
While(condition)
{
        //block of code.
}
```

Example:-

```
class j14while
{
        public static void main(String args[])
        {
                int a=0;
                while(a<10)
                {
                        System.out.println("looping");
                        a++;
                }
        }
}
```

Do-while: - it is also called exit control loop, because it first execute block of code and then check condition.

Syntax:-

```
Do
{
        //block of code
}while(condition);
```

Example:-

```
class j15dowhile
{
        public static void main(String args[])
        {
                int a=0;
                do{
                        System.out.println("looping");
                        a++;
        }while(a<10);
    }
}
```

For loop: - it is used when know how many times block of code needs to execute. It takes initialization, looping condition and increment or decrement each one is separated with semicolon (;).

Syntax:-

```
For (initialization; condition; increment/decrement)
{
    //Block of code
}
```

Example:-

```
class j16for
{
    public static void main(String args[])
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("looping");
        }
    }
}
```

- **Break: -** it is used for stop loop execution when certain condition is true.

Example:-

```
class j16for
{
    public static void main(String args[])
    {
        for(int i=0;i<10;i++)
        {
            if(i==7)
            {
                break;
            }
            System.out.println("looping");
        }
    }
}
```

- **Continue: -** it skip current looping block when condition is true.

Example:-

```
class j16for
{
      public static void main(String args[])
      {
              for(int i=0;i<10;i++)
              {
                      if(i<=5)
                      {
                              continue;
                      }
                      System.out.println("looping");
              }
      }
}
```

ARRAYS

ONE DIMENSIONAL ARRAY: - it is collection of data elements of same data types.

Declare array: - it is declare with square brackets.

1) Datatype[] arrayname = new datatype[size];
2) Datatype arrayname[] = new datatype[size];

Assign values to array: - assign values to array in curly brackets, separated by comma.

- Datatype[] arrayname = {value1, value2,…, valueN};
- Datatype arrayname[] = {value1, value2,…, valueN};

Example: - String car[] = {"volvo","ford","bmw","ferrari"};

Access values of array: - access value of array using arrayname and index of particular element. Index of array is from 0 to sizeofarray -1.

- Arrayname[index];

Example: - System.out.println(car[2]);

For loop in array: - assign or access values of array using for loop with arrayname.length property of array.

Example: -

```
class j17array
{
        public static void main(String args[])
        {
                int num[] = new int[10];
                for(int i=0;i<num.length;i++)
                {
                        num[i]= i+1;
                        System.out.println(num[i]);
                }
        }
}
```

For-each loop: - it is exclusive loop for loop through elements in array.

Syntax: -

```
For(datatype variable : arrayname)
{
        //Block of code
}
```

Example: -

```
class j17array
{
	public static void main(String args[])
	{
		String car[] = {"volvo","ford","bmw","ferrari"};
		for(String company: car)
		{
			System.out.println(company);
		}
	}
}
```

TWO DIMENSIONAL ARRAYS: - it is also called matrix. It is array of array.

Declare array: - it declare with two square brackets. First is for row and second is for colomn.

1) Datatype array_name[] [] = new datatype[row_size] [colomn_size];
2) Datatype[] [] array_name = new datatype[row_size] [colomn_size];

Assign values: - assign values with brackets or using for loop.

Example: - Assign values using brackets.

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

Example: - assign values using for loop.

```
int[][] matrix= new int[3][4];
for(int i=0;i< matrix.length;i++)
{
	for(int j=0;j< matrix[i].length;j++)
	{
		matrix[i][j]= 1;
	}
}
```

Accessing values: - access 2d arrays value using row and column index.

Syntax:-

```
Array_name[ row_index] [column_index];
```

Example:-

```
System.out.println(arr[1][1]);
```

CLASSES AND OBJECTS

CLASSES: - these are combination of properties, fields and methods. These are real-time entity and user-defined data type. Classes are used for object oriented programming. As java is fully object oriented language hence the main method or entry point program is also contain in class.

Declare class: - declare class using class keyword and write all fields and methods inside class.

Example:-

```
class abc
{

}
```

Properties: - these are variables of class that can only accessible by object of that class.

Example:-

```
class abc
{
        int a,b;
}
```

Methods: - these are functions of classes used for processing. These methods are only accessible with object of that particular class.

Example:-

```
class abc
{
        void get(int a1,int b1)
        {
                a=a1;
                b=b1;
        }
        void show()
        {
                System.out.println("a= "+a);
                System.out.println("b= "+b);
        }
}
```

OBJECTS: - objects are instance of classes. There can be multiple objects of single class. Creating object allocate memory to particular class, with help of new keyword object get created.

Syntax:-

Class_name object_name = new class_name();

Example:-
abc obj= new abc();

Accessing class members: - access class member using (.) member accessing operator.

Example:-
obj.get(4,5);
obj.show();

Example: - assign values to class variables.

```
class j19class
{
        int a;
        int b;
        public static void main(String artgs[])
        {
                abc obj= new abc();
                obj.get(4,5);
                obj.show();
        }
}

class abc
{
        int a,b;
        void get(int a1,int b1)
        {
                a=a1;
                b=b1;
        }
        void show()
        {
                System.out.println("a= "+a);
                System.out.println("b= "+b);
        }
}
```

TYPES OF METHODS: - there are four types of methods.

1) **Without argument, Without return value**

Example:-

```
void good()
{
    System.out.println("Good day");
}
```

2) With argument, Without return value

Example:-

```
void add(int a, int b)
{
    int c=a+b;
    System.out.println("addition "+c);
}
```

3) Without argument, with return value

Example: - use return key word for return value and define return type also.

```
int subtract()
{
    int a=7;
    int b=4;
    int c=a-b;
    return c;
}
```

4) With argument, with return value

Example:-

```
int multi(int a,int b)
{
    int c=a*b;
    return c;
}
```

CONSTRUCTOR: - it is special method whose name is same as class. It automatically gets called when object is created. Constructor is used to allocate memory to fields or properties and methods of the class. Constructor is real-time entity of class. it does not have any return type.

Types of constructors: -

Default constructors: - it does not take any parameters. Java provide default constructor if user did not create in class.

Syntax:-

```
Class class_name
{
        Class_name()
        {
                //codes of constructors
        }
}
```

Example:-

```
class construct
{
        construct()     //default constructor
        {
                System.out.println("constructor is called ");
        }
}
class j22defaultconstructors
{
        public static void main(String args[])
        {
                construct c = new construct();
        }
}
```

Parameterized constructors: - it is takes parameters when object is getting created, it is mostly used for initialize values of properties for objects.

Syntax:-

```
class add
{
        int num1,num2;
        add(int a,int b)          //parameterised constructor
        {
                num1=a;
                num2=b;
        }
        void show()
        {
                System.out.println("Addition of "+num1+" and "+num2+" is "+(num1+num2));
        }
}

class j23parameterisedconstruct
{
        public static void main(String args[])
        {
                add a= new add(5,6);
                a.show();
        }
}
```

OVERLOADING: - It applies to methods when both methods are in same class

Method overloading: - when two or more methods having same name but different parameter list, that called as method overloading.

Syntax:-

```
Return_type method_name1 (parameter_list1)
{
…
}

Return_type method_name1 (parameter_list2)
{
…
}
```

Example:-

```
class sum
{
        int ans;
        void add(int a,int b)
        {
                ans=a+b;
                System.out.println("addition of "+a+" and "+b+" is "+ans);
        }

        void add(int a,int b,int c)
        {
                ans=a+b+c;
                System.out.println("addition of "+a+" and "+b+" and "+c+" is "+ans);
        }
}

class j24methodoverloading
{
        public static void main(String args[])
        {
                sum s1= new sum();
                s1.add(4,5);
                s1.add(4,5,6);
        }
}
```

Constructor overloading: - when class has more constructors than one, with different parameters list then it is called constructor overloading.

Syntax:-

```
Class class_name
{
        Class_name(parameter_list1)
        {
```

```
        Block of code
        }

        Class_name(parameter_list2)
        {
        Block of code
        }
}
```

Example:-

```
class multiply
{
        int ans;
        multiply(int a,int b)
        {
                ans=a*b;
                System.out.println("multiplication of "+a+" and "+b+" is "+ans);
        }

        multiply(int a,int b,int c)
        {
                ans=a*b*c;
                System.out.println("multiplication of "+a+" and "+b+" and "+c+" is "+ans);
        }
}

class j25constructoroverloading
{
        public static void main(String args[])
        {
                multiply m1= new multiply(4,5);
                multiply m2= new multiply(4,5,6);
        }
}
```

INHERITANCE: - the process of creating new class from existing class is called inheritance. In inheritance base class's methods and properties inherits in derived class. for inherit base class in new class "extends" keyword is used.

Base class is also known as parent or super class.
Derived class is also known as child or sub class.

Syntax:-

```
Class parent
{
        Parent class code
}
```

```
Class child extends parent
{
        Child class code
}
```

Example:-

```
class person
{
        String name;
        int age;
        void show()
        {
                System.out.println("Name:- "+name);
                System.out.println("age:- "+age);
        }
}

class student extends person
{
        int roll;
        float percent;
        void get(String n,int a,int r,float p)
        {
                name=n;
                age=a;
                roll=r;
                percent=p;
        }
        void display()
        {
                show();
                System.out.println("roll no:- "+roll);
                System.out.println("percentage:- "+percent);
        }
}
class j26inheritance
{
        public static void main(String args[])
        {
                student s1= new student();
                s1.get("ram",20,1,97.67f);
                s1.display();
        }
}
```

INHERITANCE USING CONSTRUCTORS:-

Default constructor:-

```
class num1
{
        int n1;
        num1()
        {
                n1=10;
        }
        void show1()
        {
                System.out.println("N1:-"+n1);
        }
}

class num2 extends num1
{
        int n2;
        num2()
        {
                n2=20;
        }
        void show2()
        {
                System.out.println("N2:-"+n2);
        }
}

class j27inheritanceconstruct
{
        public static void main(String args[])
        {
                num2 o= new num2();
                o.show1();
                o.show2();
        }
}
```

SUPER FUNCTION: - it is used for passed the value to the constructor of the parent class.

Example:-

```
class first
{
        int n1;
        first(int a1)
        {
                n1=a1;
        }
        void showfirst()
        {
                System.out.println("N1= "+n1);
        }
}

class second extends first
{
        int n2;
        second(int a1,int a2)
        {
                super(a1);
                n2=a2;
        }
        void showsecond()
        {
                System.out.println("N2= "+n2);
        }
}

class j29superfunction
{
        public static void main(String args[])
        {
                second o= new second(23,34);
                o.showfirst();
                o.showsecond();
        }
}
```

MULTILEVEL INHERITANCE: - it is multiple classes inherit in one another as grandparent class, parent class child class.

Syntax:-

```
Class grandparent
{
        Block of code
}

Class parent extends grandparent
{
        Block of code
}

Class child extends parent
{
        Block of code
}
```

Example:-

```
class first
{
        int n1;
        first(int a1)
        {
                n1=a1;
        }
        void showfirst()
        {
                System.out.println("N1= "+n1);
        }
}

class second extends first
{
        int n2;
        second(int a1,int a2)
        {
                super(a1);
                n2=a2;
        }
        void showsecond()
        {
                System.out.println("N2= "+n2);
        }
}

class third extends second
{
        int n3;
        third(int a1,int a2,int a3)
        {
                super(a1,a2);
                n3=a3;
        }
        void showthird()
        {
                System.out.println("N3= "+n3);
        }
}

class j30multilevelinherit
{
	public static void main(String args[])
	{
		third o= new third(23,34,45);
		o.showfirst();
		o.showsecond();
		o.showthird();
	}
}
```

OVERRIDING: - it performs with inherited classes those have same methods with same name. The child class overrides parent class method.

Example:-

```
class first
{
	int n1;
	void show(int x1)
	{
		n1=x1;
		System.out.println("N1= "+n1);
	}
}

class second extends first
{
	int n2;
	void show(int x2)
	{
		n2=x2;
		System.out.println("N2= "+n2);
	}
}

class j31overriding
{
	public static void main(String args[])
	{
		second o=new second();
		o.show(34);
	}
}
```

SUPER KEYWORD: - it is used for called override methods or properties from parent class into child class.

Example:-

```
class first
{
        int n1;
        void get(int x1)
        {
                n1=x1;
        }
        void show()
        {
                System.out.println("N1= "+n1);
        }
}

class second extends first
{
        int n2;
        void get(int x1,int x2)
        {
                super.get(x1);
                n2=x2;
        }
        void show()
        {
                super.show();
                System.out.println("N2= "+n2);
        }
}

class j32superkeyword
{
        public static void main(String args[])
        {
                second o= new second();
                o.get(23,34);
                o.show();
        }
}
```

ACCESS MODIFIERS: - there are four types of access modifiers/specifiers in java.

Private access modifiers: - the access level of private modifier is only within the class. It cannot access outside class.

Default access modifiers: - the access level of default modifier is only within the package it cannot be access from outside package, if any other access is not specify then it is default access modifiers.

Protected access modifiers: - the access level of protected modifiers within the package and outside the package through the child class. Without child class it cannot be accessed outside package but can be access within the package.

Public access modifiers: - the access level of public modifier is everywhere. It can access within package and outside package also within the class and outside the class.

Example:-

```
class access
{
        private int a;
        private int b;

        public void set(int x,int y)      //public access modifier
        {
                a=x;
                b=y;
        }

        private void show()               //private access modifier
        {
                System.out.println("a="+a);
                System.out.println("b="+b);
        }
        void display()              //default access modifier
        {
                show();
        }
}
class j33accessmodifiers
{
        public static void main(String args[])
        {
                access a1= new access();
                a1.set(3,4);
                a1.display();
        }
}
```

THIS KEYWORD: - this keyword can be used to refer current class object/instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Example:-

```
class first
{
        protected int a;
        first(int a)
        {
                this.a=a;       //using this keyword
        }
        void show()
        {
                System.out.println("a="+a);
        }
}
class second extends first
{
        protected int b;
        second(int a,int b)
        {
                super(a);
                this.b=b;       //using this keyword
        }
        void show()
        {
                super.show();
                System.out.println("b="+b);
        }
        void add()
        {
                int c=a+b;
                System.out.println("Adition of "+a+" and "+b+" is "+c);
        }
}
class j34thiskeyword
{
        public static void main(String args[])
        {
                second o = new second(5,3);
                o.show();
                o.add();
        }
}
```

ABSTRACT CLASS AND METHODS:-

Abstract class:-

- Abstract class is a class which must have to inherit in other class and must have to create child class of abstract class.
- Creating object of abstract class is not allowed.

- Abstract class may have one or more abstract methods.
- Abstract class is declares using abstract keyword.

Abstract method:-

- The method which declared using abstract keyword is called as abstract method.
- Abstract method must be override.
- It does not have any-body.
- It is must declared inside abstract class.

Example:-

```
abstract class square
{
        int l;
        void get(int l)
        {
                this.l=l;
        }
        void sqarea()
        {
                int a=l*l;
                System.out.println("Area of sqarea "+a);
        }
        abstract void findmax();
}
class rectangle extends square
{
        int b;
        void accept(int b)
        {
        this.b=b;
        }
        void rearea()
        {
                int a=l*b;
                System.out.println("Area of rectangle "+a);
        }
        void findmax()
        {
                int m;
                if(l>b)
                {
                        m=l;
                }
                else
                {
                        m=b;
                }
                System.out.println("max is "+m);
        }
}
class j35abstractclass
{
        public static void main(String args[])
        {
                rectangle o = new rectangle();
                o.get(5);
                o.accept(7);
                o.sqarea();
                o.rearea();
                o.findmax();
        }
}
```

INTERFACE: - An interface is a blueprint of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Syntax: -

```
Interface interface_name
{
        Declare constants field
        Declare abstract methods
}
```

Example: -

```
interface first
{
        int a=34;
        public void show1();
}

interface second
{
        int a=43;
        public void show2();
}

class internal implements first, second
{
        public void show1()
        {
                System.out.println("First show method "+first.a);
        }
        public void show2()
        {
                System.out.println("Second show method "+second.a);
        }
}

class j36interface
{
        public static void main(String args[])
        {
                internal o= new internal();
                o.show1();
                o.show2();
        }
}
```

EXCEPTION

WHAT IS: - An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occur the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

WHY: -

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

EXCEPTION TYPE: -

- Checked exceptions
- Unchecked exceptions
- Errors

Checked exceptions: -A checked exception is an exception that is checked (notified) by the compiler at compilation-time; these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

Unchecked exceptions: - An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

Errors: - These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in code because rarely it can be solve. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Stack overflow: - A stack overflow is a type of buffer overflow error that occurs when a computer program tries to use more memory space in the call stack than has been allocated to that stack.

TRY CATCH: - A method catches an exception using a combination of the try and catch keywords. A try and catch block is placed around the code that might generate an exception.

Try block: - The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it.

Catch block: - A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Syntax: -

```
try{
        //block that can occurs exception
}
catch()
{
        //block that handle exception
}
```

Example: -

```
class j39trycatch
{
        public static void main(String args[])
        {
                try{
                        int a=34;
                        int b=0;
                        int c=a/b;
                        System.out.println("a/b = "+c);
                }
                catch(Exception e)
                {
                        System.out.println(e.);
                }
        }
}
```

THROWS: - If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.

Reading data from bufferedReader: -

```
import java.io.*;

class j40throws
{
        public static void main(String args[])
        {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

                System.out.println("Enter Name");

                String str=br.readLine();
                System.out.println("Name:- "+str);
        }
}
```

import java.io.*; : - it import io package in program.
BufferedReader br: - creating buffered reader object.
new InputStreamReader(System.in): - passing object of InputStreamReader class to BufferedReader constructor.
System.in: - passing input stream to InputStreamReader constructor.

This program will throw exception on compiling as: -

```
j40throws.java:11: error: unreported exception IOException; must be caught or de
clared to be thrown
            String str=br.readLine();
                                  ^
1 error
```

Using throws in program: -

```
import java.io.*;

class j40throws
{
        public static void main(String args[]) throws IOException
        {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

                System.out.println("Enter Name");

                String str=br.readLine();

                System.out.println("Name:- "+str);
        }
}
```

public static void main(String args[]) throws IOException: - throws IOException at main method's header's end.

Throws in other methods and constructor: -

```
import java.io.*;

class div
{
        div() throws IOException
        {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                try
                {
                System.out.print("Enter Number1 = ");
                int a=Integer.parseInt(br.readLine());

                System.out.print("Enter Number2 = ");
                int b=Integer.parseInt(br.readLine());

                int c=a/b;
                System.out.println("Division of "+a+" and "+b+" is "+c);
                }
                catch(Exception e)
                {
                        System.out.println("Something went wrong");
                }
        }
}

class j40throws
{
        public static void main(String args[]) throws IOException
        {
                div d = new div();
        }
}
```

- Declaring throws in main static method along with other class methods or constructor is reqired.

TYPE CASTING: -

Implicit type casting: - a smaller type is converted into a larger type, it is done by the compiler automatically.
byte > short > char > int > long > float > double

Example: -

```
class j41typecasting
{
        public static void main(String args[])
        {
                int a=45;
                double d=a;
                System.out.println(d);
        }
}
```

Explicit type casting: - is done by the programmer manually. In the narrowing type casting a larger type can be converted into a smaller type.

Example: -

```
class j41typecasting
{
        public static void main(String args[])
        {
                double dd=45.6786;
                int aa= (int)dd;
                System.out.println(aa);
        }
}
```

Type casting with parse and valueof methods: -

```
class j41typecasting
{
        public static void main(String args[])
        {
                String n="34";
                int num1=Integer.parseInt(n);
                System.out.println(num1);

                String f="34.565";
                float f1=Float.parseFloat(f);
                System.out.println(f1);

                String d="34.5678989";
                double d1=Double.parseDouble(d);
                System.out.println(d1);

                int a=45;
                String s=String.valueOf(a);
                System.out.println(s);

                float f2=45.56f;
                String s2=String.valueOf(f2);
                System.out.println(s2);

                double d2=45.45767;
                String s3=String.valueOf(d2);
                System.out.println(s3);

                char c1='a';
                String s4=String.valueOf(c1);
                System.out.println(s4);
```

```
          }
}
```

MULTIPLE CATCH BLOCKS: - After try blocks there can be take multiple catch blocks.

Example: -

```
import java.io.*;
class j42multicatch
{
          public static void main(String args[])
          {
                    try
                    {
                              BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
                              System.out.print("Enter S1= ");
                              String s1 = br.readLine();
                              System.out.print("Enter N1= ");
                              int n1=Integer.parseInt(br.readLine());
                              int n2=Integer.parseInt(s1);
                              int c=n1/n2;

                              System.out.println("C = "+c);
                    }
                    catch(ArithmeticException ae)
                    {
                              System.out.println(ae);
                    }
                    catch(NullPointerException np)
                    {
                              System.out.println(np);
                    }
                    catch(NumberFormatException nf)
                    {
                              System.out.println(nf);
                    }
                    catch(IOException io)
                    {
                              System.out.println(io);
                    }
          }
}
```

Run program: -
Input1:-

```
Enter S1= d
Enter N1= 4
java.lang.NumberFormatException: For input string: "d"
```
Input2:-
```
Enter S1= 0
Enter N1= 4
java.lang.ArithmeticException: / by zero
```
Input3:-
```
Enter S1=
Enter N1=
java.lang.NumberFormatException: For input string: ""
```
Input4:-
```
Enter S1= 4
Enter N1=
java.lang.NumberFormatException: For input string: ""
```
Input5:-
```
Enter S1= 4
Enter N1= 8
C = 2
```

FINALLY BLOCK: - The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception.
Using a finally block allows to run any cleanup-type statements that want to execute, no matter what happens in the protected code.

Why: -

- Java finally block can be used for clean-up (closing) the connections, files opened, streams, etc. those must be closed before exiting the program.
- It can also be used to print some final information.

Example: - without finally block
```
class j43finally
{
	public static void main(String args[])
	{
		int a[] = {1,2};
		try
		{
			System.out.println("Access element three :" + a[3]);
		}
		catch(ArithmeticException e)
		{
			System.out.println("Exception thrown  :" + e);
		}
		a[0] = 6;
		System.out.println("First element value: " + a[0]);
```

```
                System.out.println("The finally statement is executed");
        }
}
```

Output: -

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out
 of bounds for length 2
        at j43finally.main(j43finally.java:8)
```

Example: - with finally block

```
class j43finally
{
        public static void main(String args[])
        {
                int a[] = {1,2};
                try
                {
                        System.out.println("Access element three :" + a[3]);
                }
                catch(ArithmeticException e)
                {
                        System.out.println("Exception thrown  :" + e);
                }
                finally
                {
                        a[0] = 6;
                        System.out.println("First element value: " + a[0]);
                        System.out.println("The finally statement is executed");
                }
        }
}
```

Output: -

```
First element value: 6
The finally statement is executed
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out
 of bounds for length 2
        at j43finally.main(j43finally.java:8)
```

Finally with function: - it execute also after returning the value

Example: -

```
class excep
{
        int div(int a, int b)
        {
                boolean f=true;
                try
                {
                        return a/b;
                }
                catch(Exception e)
                {
                        System.out.println(e);
                        f=false;
                        return 0;
                }
                finally
                {
                        String m=f?"program executed":"Error occured";
                        System.out.println(m);
                }
        }
}

class j43finally
{
        public static void main(String args[])
        {
                excep e= new excep();
                int ans=e.div(4,0);
                System.out.println("division = "+ans);
        }
}
```

THROW: - it used for create user defined exceptions using throw keyword and user defined exception class.

Example: -

```
import java.io.*;

class myexception extends Exception
{
        public myexception(String str)
        {
                super(str);
        }
}

class j44throwerror
{
        public static void main(String args[]) throws IOException
        {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Enter Number = ");
                int a=Integer.parseInt(br.readLine());
                try
                {
                    if(a>100)
                    {
                        throw new myexception("Number is greater");
                    }
                    else
                    {
                        System.out.print("Number = "+a);
                    }
                }
                catch(myexception e)
                {
                    System.out.println(e.getMessage());
                }
        }
}
```

FILE HANDLING

FILE: - it is referred as abstract datatype a named location used to store related information is known as the file.
File is a named location used to store related information on disk storage.
There are several operations which perform on file: -

- Creating new file
- Getting information about file
- Writing into file
- Reading from file
- Deleting file

CREATING FILE: -

Example: -

```
import java.io.File;
import java.io.IOException;

 class j46createfile
 {
        public static void main(String args[])
        {
                try
                {
                        File f = new File("D: /javacore/files/file1.txt");
                        if(f.createNewFile())
                        {
                                System.out.println("File "+f.getName()+" is created Successfully");
                        }
                        else
                        {
                                System.out.println("File is already Exists");
                        }

                        if(f.exists())
                        {
                                System.out.println("File Path:- "+f.getAbsolutePath());
                                System.out.println("File Writable:- "+f.canWrite());
                                System.out.println("File Readable:- "+f.canRead());
                                System.out.println("File Length:- "+f.length());
                        }
                }
                catch(IOException e)
                {
                        System.out.println("Exception occured");
                }
            }
}
```

- **Import java.io.File; -** importing file class for file handling operations.
- **import java.io.IOException; -** importing ioexception class for handling exceptions.
- **File f: -** creating file class object in try block and passing path with file name and extension to file class constructor.
- **f.createNewFile(): -** it is method from file class to create new file with provided constructor argument. If file is created it return true.
- **f.getName(): -** it is method for get file name.
- **f.exists(): -** it is method that determine file is exists or not that create with current object. It returns true if file is exists.
- **f.getAbsolutePath(): -** return file path that created with current object.
- **f.canWrite(): -** it return true if file is writable.
- **f.canRead(): -** it return true if file is readable.
- **F.length(): -** it return length of file that how many character it contain.
- **IOException: -** it is catching any exception occurred.

WRITE FILE: -

Example: -

```
import java.io.*;

class j47writefile
{
        public static void main(String args[])
        {
                try{
                        FileWriter writer= new FileWriter("D:/javacore/files/good.txt");
                        writer.write("I love Java");
                        writer.close();
                        System.out.println("Data Stored");
                }catch(IOException e)
                {
                        System.out.println(e);
                        System.out.println("Error occured");
                }
        }
}
```

- **FileWriter writer**: - it is class for write file in constructor it take file name with path as string.
- **Write(): -** it is method from FileWriter class to writer data in file, it take data as string.
- **Close(): -** it is method from FileWriter class to close file after writer.

Write multiple lines in file: -

```
import java.io.*;
import java.util.Scanner;

class j47writefile
{
        public static void main(String args[])
        {
                try{
                        FileWriter writer= new FileWriter("D:/javacore/files/good.txt");
                        Scanner sc=new Scanner(System.in);
                        System.out.println("Enter data (Type 'EXITFILE' for stop writing)");
                        while(true)
                        {
                                String s=sc.nextLine();
                                if(s.equals("EXITFILE"))
                                {
                                        break;
                                }
                                writer.write(s+"\n");
                        }
                        writer.close();
                        System.out.println("Data Stored");
                }catch(IOException e)
                {
                        System.out.println(e);
                        System.out.println("Error occured");
                }
        }
}
```

- **sc.nextLine(): -** it is method from Scanner class to read line.
- **S.equals("…"): -** it is method from String class to check String is equals to given value string.

READ FILE: -

Example: -

```
import java.io.*;
import java.util.Scanner;

class j48readfile{

        public static void main(String args[])
        {
                try{
                        File f= new File("D:/javacore/files/good.txt");
                        Scanner reader= new Scanner(f);
                        while(reader.hasNextLine())
```

```
                    {
                        String data=reader.nextLine();
                        System.out.println(data);
                    }
                    reader.close();
                }catch(IOException e)
                {
                    System.out.println(e);
                    System.out.println("Error occured");
                }
        }
}
```

- **File f: -** create and initialize object of file class pass file name and path to its constructor.
- **Scanner reader: -** create object of Scanner class and pass file object to its constructor.
- **Reader.hasNextLine(): -** it is method from Scanner class to check to another line in input of scanner.
- **Reader.close(): -** it is method from Scanner class to close reader.

DELETE FILE: -

Example: -

```
import java.io.*;

class j49deletefile{

        public static void main(String args[])
        {
                try{
                        File f= new File("D:/atulmore/javacore/files/good.txt");
                        String name=f.getName();
                        f.delete();
                        System.out.println(name+" deleted successfully");

                }catch(Exception e)
                {
                        System.out.println(e);
                        System.out.println("Error ocuured");
                }
        }
}
```

- **F.delete(): -** it is method from file class for delete file.