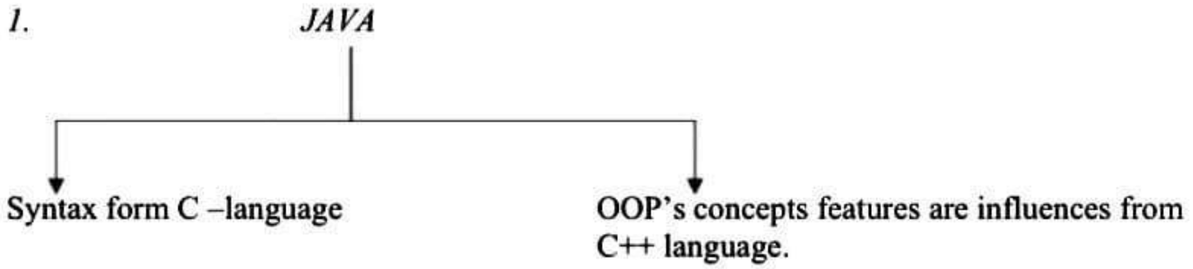
EDUCATALYSTS

JAVA PROGRAMMING

Java notes

Genesis of JAVA

1.



2. C, Pascal , Basic, Fortran are the based on either compiler or interpreter. But Java programs are based on compiler and interpreter both.

$$\begin{tikzcd}\boxed{\text{Source code}} \arrow[r, "\text{Java Compiler}"] & \boxed{\text{Byte Code}} \arrow[r, "\text{Java Interpreter}"] & \boxed{\text{Machine Code}}\end{tikzcd}$$

3. Java program is platform independent to achieve the independency the Byte codes area used.
4. Java Compiler is generally known as JVM(Java Virtual Machine).

Importance of java in Internet

One of the java advantages for use over the Internet is that platform independent, meaning that programs created with the java can run on any machine (computers).Java achieves its independence by creating programs designed to run on the JVM rather than any specific computer system.
These java programs once compiled and stored in a compact machine independent format known as Byte codes. Byte code files area easily transferred across the Internet. The preloaded interpreter run the byte code in the local machine then plays them.

Java Features

- **Simple** - Java's developers deliberately left out many of the unnecessary features of other high-level programming languages. For example, Java does not support pointer math, implicit type casting, structures or unions, operator overloading, templates, header files, or multiple inheritance.

- **Object-oriented.** Just like C++, Java uses classes to organize code into logical modules. At runtime, a program creates objects from the classes. Java classes can inherit from other classes, but multiple inheritance, wherein a class inherits methods and fields from more than one class, is not allowed.

- **Statically typed** - All objects used in a program must be declared before they are used. This enables the Java compiler to locate and report type conflicts.

- **Compiled** - Before you can run a program written in the Java language, the program must be compiled by the Java compiler. The compilation results in a "byte-code" file that, while similar to a machine-code file, can be executed under any operating system that has a Java interpreter. This interpreter reads in the byte-code file and translates the byte-code commands into machine-language commands that can be directly executed by the machine that's running the Java program. You could say, then, that Java is both a compiled and interpreted language.

- **Multi-threaded** - Java programs can contain multiple threads of execution, which enables programs to handle several tasks concurrently. For example, a multi-threaded program can render an image on the screen in one thread while continuing to accept keyboard input from the user in the main thread. All applications have at least one thread, which represents the program's main path of execution.

- **Garbage collector** - Java programs do their own garbage collection, which means that programs are not required to delete objects that they allocate in memory. This relieves programmers of virtually all memory-management problems.

- **Robust** - Because the Java interpreter checks all system access performed within a program, Java programs cannot crash the system. Instead, when a serious error is discovered, Java programs create an exception. This exception can be captured and managed by the program without any risk of bringing down the system.

- **Secure** - The Java system not only verifies all memory access but also ensures that no viruses are hitching a ride with a running applet. Because pointers are not supported by the Java language, programs cannot gain access to areas of the system for which they have no authorization.

- **Extensible** - Java programs support native methods, which are functions written in another language, usually C++. Support for native methods enables programmers to write functions that may execute faster than the equivalent functions written in Java. Native methods are dynamically linked to the Java program; that is, they are associated with the program at runtime. As the Java language is further refined for speed, native methods will probably be unnecessary.

Save this program named as ***firstdemo.java*** and complied as ***c:\>javac firstdemo.java*** and run above program as ***c:\>java firstdemo*** **{where object call the data with the help of objects}**

What is the Polymorphism?

Polymorphism build in two words =poly+morphism here poly means many and morphism means shapes. And merely means using the same one name to refer to different methods. "Name Reuse" would be better terms to use for polymorphism.
There are two types of polymorphism in java.

1. **Overloading:(Same name but different parameter)**

In java, it is possible to create methods that have same name but different definition. That is called method overloading. Method overloading is used where objects area required performing similar task but using different input parameters. When we call a method in an object, JVM matches up the method name first and then the number and type of parameters to decide which one of the definition to execute.

```
//Example of overloading.

class student {
        int idno, mark;
        String name;
        void setData() {
                idno=90;mark=89;name="raja";
        }
        void setData(int id,int m,String n) {
                idno=id;mark=m;name=n;
        }
        void show() {
                System.out.println("\n\nStudent RollNo.="+idno);
                System.out.println("Student Name.="+name);
                System.out.println("Student RollNo.="+mark);
        }
}//end of student class

class demoOverload {
        public static void main(String args[]){
                student obj=new student();
                obj.setData();
                obj.show();
                student obj1=new student();
                obj1.setData(1,67,"vinu");
                obj1.show();
        }
}
```

```
C:\corejava\oops>javac demoOverload.java
C:\corejava\oops>java  demoOverload
Student RollNo.=90
Student Name.=raja
Student RollNo.=89

Student RollNo.=1
Student Name.=vinu
Student RollNo.=67
```

```
//Example of overloading.
class Room {
        int length;
        int width;
        int area;
        void setData(int l,int w) {
                length=l;
                width=w;
        }
        void setData(int x) {
        length=width=x;}
        void show() {
                area=length*width;
                System.out.println("\n\nArea of rectangle="+area);
        }
}//end of Room class

class RoomArea {
        public static void main(String args[]) {
                Room obj=new Room();
                obj.setData(25,15);
                obj.show();
                Room obj1=new Room();
                obj1.setData(20);
                obj1.show();
        }
}
```

```
C:\corejava\oops>javac RoomArea.java

C:\corejava\oops>java  RoomArea

Area of rectangle=375

Area of rectangle=400
```

Difference between method overloading and method overriding

Method Overriding	**Method Overloading**
1. In method overriding methods have same name and have the same parameter list as a super class method.	1. In method overloading methods have same name but have different parameter lists.
2. Appear in subclass only.	2. Appear in the same class or a subclass.
3. Inheritance concept involves.	3. Inheritance concept is not compulsory.
4. have the same return type as a super class method.	4. Can have different return types.
5. Late binding or redefine a method is possible.	5. Not possible.

Abstract class

An abstract class is a class that has at least one abstract method, along with concrete methods. An abstract method is a method that is declared with only its signature, it has no implementation. That means method without functionality.
Since abstract class has at least one abstract class has at least one abstract method, an abstract class cannot be instantiated.

Characteristics of Abstract class

1. There can be no objects of an abstract class.
2. An abstract class cannot be directly instantiated with the '**new**' operator.
3. **abstract** keyword use as preface in class declaration to create a abstract class.
4. **abstract** keyword use as preface in method declaration to create a abstract method.
5. An abstract class is not fully defined.
6. An abstract class provide the frame for subclasses and subclass which extends super abstract class have must functionality for abstract method of super abstract class.

Example # 1

```
abstract class A {
        abstract void callme(); // this is abstract method
        void callmetoo() // this is concrete method
        {
        System.out.println("this is concrete method of abstract class");
        }
}
class B extends A  {
        void callme() {
                System.out.println("give the functionality of super class abstract method");
        }
}
class abstractDemo {
        public static void main(String args[]) {
                B obj=new B();
                obj.callme();
                obj.callmetoo();
        }
}
```

Example # 2

```
abstract class shape {
        int a,b;
        shape(int x,int y) {
                a=x;
                b=y;
        }
        abstract int area();
}

class Triangle extends shape {
        Triangle(int x,int y) {
                super(x,y);
        }
        int area() {
                return (a*b)/2;
        }
}
class Rectangle extends shape {
        Rectangle(int x,int y) {
                super(x,y);
        }
        int area() {
                return a*b;
        }
}

class abstractDemo2 {
        public static void main(String args[]) {
        // shape obj=new shape(10,29);  it is not possible bez
        Rectangle r1=new Rectangle(12,3);

        /*System.out.println("Area of Rectangle is="+r1.area());*/
        /* suppose we want to call area method of super abstract class.
        so call it with the help of reference variable of superclass which
        hold the object of subclass */
```

```
class override {
        public static void main(String args[]) {
                Sub obj=new Sub(100,50);
                obj.show();
        }
}
```

Example # 2

```
// this is the prg for invoking superclass variable in subclass by using  super keyword.

class A {
        int a=100;
         void show() {
                System.out.println("Method in super class");
                System.out.println("value of a in super class="+a);
        }
} // end of super class
class B extends A {
        int a=200;
        void show() {
                System.out.println("Method in sub class");
                System.out.println("value of a in SUB class="+a);
                System.out.println("value of a in super class(invoke by super)="+super.a);
        }
}
class SuperDemo {
        public static void main(String args[]) {
                B obj=new B();
                obj.show();
        }
}
```

Example # 3

```
// this is the prg for invoking superclass constructor in subclass by using super keyword.
class Room {
        int length;
        int breath;
        Room(int x,int y) {
                length=x;
                breath=y;
        }
        int area() {
                return length*breath;
        }
}// end of superclass
class vol extends Room {
        int height;
        vol(int x,int y,int z) {
                super(x,y); // invoke superclass constructor.
                height=z;
        }
        int volume() {
                return length*breath*height;
        }
}//end of subclas.
class SuperDemo {
        public static void main(String args[]) {
                vol r1=new vol(2,2,3);
                System.out.println("Area="+r1.area());
                System.out.println("Volume="+r1.volume());
        }
}
```

this keyword:-

All instance methods received an implicit agument called **this**, which may be used inside any method to refer to the current object, i.e. the object on which method was called. Inside an instance of method, any unqualified reference is implicitly associated with the this reference.

Characteristics of this keyword.

1. when you need to pass a reference to the current object as an argument to another method.
2. when the name of the instance variable and parameter is the same, parameter hides the instance variable.

this and super keyword

1. Both are used for object reference.
2. "this" reference is passed as an implicit parameter when an instance method is invoked. It denotes the object on which the method is invoked. "super" can be used in the body of an instance method in a subclass to access variables/methods of superclass.
3. "this" is used in the constructor then it should be the first statement in the constructor, it is used to invoked the overloaded constructor of the same class. "super" is used in the constructor then it should be the first statement in that constructor, it is used to invoke the immediate super class constructor.
4. Subclass without any declared constructor would fail it the super class doesn't have default constructor.
5. "this" and "super" statements cannot be combined.

Final Keyword

Final is a keyword which can be used to create constant value.

Basically final keyword used for 3 purpose.

What is an Arrays?

An array is a finite set of elements of homogeneous data types. The elements of array are accessed by index.
Arrays index starts with 0 in java. In java arrays are treated as objects.

Creating an Array:->
Creation of array involves the three steps.

1. **Declare the array.**

   There are two ways to declare an arrays

   - *<data type>*[]   arrayvariablename;

   example:-int[] a;

   - *<data type>*   arrayvariablename[];

   example:-int a[];

Remember: we do not enter the size of array in the declaration. you cannot assign the values to the array unless the array is created, because array is an object in java it should be create using new operator.

2. **create location into the memory.**

After decalartion an arrays we needed to create it in the memory. Java allows as to create arrays using new operator.

Arrayname=new <data type>[size];

Example:- a =new int[5];
It is array holds the 5 integer values.

➢ **It is possible to combine the two steps(declaration and creation) into one as follows:-**

int a[]=new int[5]; OR int[] a =new int[5];

3. **put values into the memory location.(Initialization of Arrays)**

The final steps is to put values into the array created. This process in known as initialization. This is done using the array subscripts as shown below.

```
    Arrayname[subscript]=values;
Example:--  number[0]=67;
            number[1]=7;
            number[2]=6;
            number[3]=37;
```

Note:-we can also initialize arrays automatically in the same way as the ordinary variables when they are declare.

Data Type   ArrayName[]={list of values};

Example::-int numbers[]={2,4,5,6,7};

Array length:-

In java all arrays store the allocate size in a variable named length. We can access the size of array using following syntax.

```
        int     variableName=array.length;
```

example :- int n=number.length;
here n assign value 5;

Some key feature of Array in JAVA:-

- *Once array is created its size cannot be changed.*
- *The size of an array give by array property called length.*
- *Array has index from 0 to length-1.*
- *By default array elements are initialized with zero or NULL.*
- *When user access the any element other then its range, java compiler give ArrayOutOfBoundsException.*

Example # 1

```
    //EXAMPLE OF SINGLE DIMENSIONAL ARRAY FIND THE AVG OF ARRAY
    ELEMENTS.
    class Avgarraydemo {
        public static void main(String args[]) {
                int a[]={12,12,12,12,12};
                double avg=0;
                int i;
            for(i=0;i<a.length;i++) {
                avg=avg+a[i];
            }
        System.out.println(i);
        avg=avg/i;
        System.out.print("Average="+avg);
        }
    }
```

Example # 2

```
    //Example for set elements of array in ascending order and descending order.
    class AscDsc {
        int i,j,a[]={22,33,21,5,26};
        int l;
        void a() {
                l=a.length;
                for(i=0;i<l;i++) {
                        System.out.println("Before the arrange array is a["+i+"]="+a[i]);
                }
                System.out.println("After sorting array in Ascending order::");
                for(i=0;i<l;i++) {
                        for(j=0;j<l-1;j++)
```

TWO DIMENSIONAL

Example # 3

```
//Example to find the order of matrix.
class Text {
        public static void main(String args[]) {
                int[][] a=new int[7][9];
                System.out.println("a.lenght ="+a.length);
                System.out.println("a[0].lenght ="+a[0].length);
        }
}
//Example to initialize the value of matrix and print the matrix at output.
class Text1 {
        public static void main(String args[]) {
                int[][] a={{77,33,88},
                          {11,55,22,99},
                          {66,44}};
                for(int i=0;i<a.length;i++) {
                        System.out.println("\t"+a.length+"hai==i="+i);
                        for(int j=0;j<a[i].length;j++)
                                System.out.print("\t"+a[i][j]);
                        System.out.println();
                }

        }
}
```

Example # 4

```
//Example of two dimensional array MATRIX ADDITION, when values of matrix are
initialized.
class MatrixM {
        public static void main(String args[]) {
                int a[][]={{2,2},{2,2}};
                int b[][]={{1,1},{1,1}};
                int c[][]=new int[2][2];
                int n=a.length;
                int m=a[0].length;
                if(n==m){
                        try{
                                for(int i=0;i<n;i++) {
                                        for(int j=0;j<m;j++) {
                                                c[i][j]=a[i][j]+b[i][j];
                                                System.out.print("\t"+c[i][j]);
                                        }
                                System.out.println();
                        }
                }
                catch(Exception e){System.out.print("Some Error"+e);}
        }
        else{
                System.out.print("\n Addition of matrix is not possible");
        }
    }
}
//this is the program for matrix addition.
```

Example # 5

//Example of two dimensional array MATRIX ADDITION When value of matrixs are given by the user.

```
import java.io.*;
class M {
        int a[][]=new int[3][3];
        int b[][]=new int[3][3];
        int c[][]=new int[3][3];
        int i,j,k;
        void getdata() {
                DataInputStream in=new DataInputStream(System.in);
                System.out.println("Enter the matrix value of a:");
                try{
                        for(i=0;i<3;i++)
                                for(j=0;j<3;j++)
                                        a[i][j]=Integer.parseInt(in.readLine());
                }
                catch(Exception e){}
                System.out.println("Enter the matrix value of b:");
                try{
                        for(i=0;i<3;i++)
                                for(j=0;j<3;j++)
                                        b[i][j]=Integer.parseInt(in.readLine());
                }
                catch(Exception e){}
        }

        void cal() {
                for(i=0;i<3;i++)
                        for(j=0;j<3;j++)
                                c[i][j]=a[i][j]+b[i][j];
        }
        void show() {
                System.out.println("addition of matrix is::");
```

```
//Example of Vector class using the print of list of element of vectors.
import java.util.*;
class vdemo {
        public static void main(String args[]) {
                Vector list=new Vector();
                list.addElement("One");
                list.addElement("Two");
                list.addElement("Four");
                list.addElement("Three");
                list.addElement("Five");
                System.out.println(list);
                int n=list.indexOf("Three");
                list.insertElementAt(list.elementAt(n),2);
                System.out.println(list);
                int l=list.size();
                list.removeElementAt(l-2);
                System.out.println(list);
                int m=list.size();
                String str[]=new String[m];
                list.copyInto(str);
                for(int i=0;i<m;i++) {
                        System.out.println(str[i]);
                }
        }
}
```

output

```
C:\>java vdemo
[One, Two, Four, Three, Five]
[One, Two, Three, Four, Three, Five]
[One, Two, Three, Four, Five]
One
Two
Three
Four
Five
```

Example # 2

//Example of vector when value gives through args[];

```
import java.util.*;
class Vectordemo {
        public static void main(String args[]) {
                Vector list=new Vector();
                int n=args.length;
                for(int i=0;i<n;i++) {
                        list.addElement(args[i]);
                }
                list.insertElementAt("COBOL",2);
                int size=list.size();
                String a[]=new String[size];
                list.copyInto(a);
                System.out.println("List of languges are::");
                for(int i=0;i<size;i++) {
                        System.out.println(a[i]);
                }
        }
}
```

```
/*run
c:\corejava\class>java Vectordemo Ada BASIC FORTRAN JAVA C++
```

<u>OUTPUT</u>

```
List of languges are::
Ada
BASIC
COBOL
FORTRAN
JAVA
C++
*/
```

<u>Using objects as parameter of methods.</u>

So far we have only been consider the simply types of method where we give some parameter for the iteration. But here we consider passes objects as parameter to the methods .