Java Class

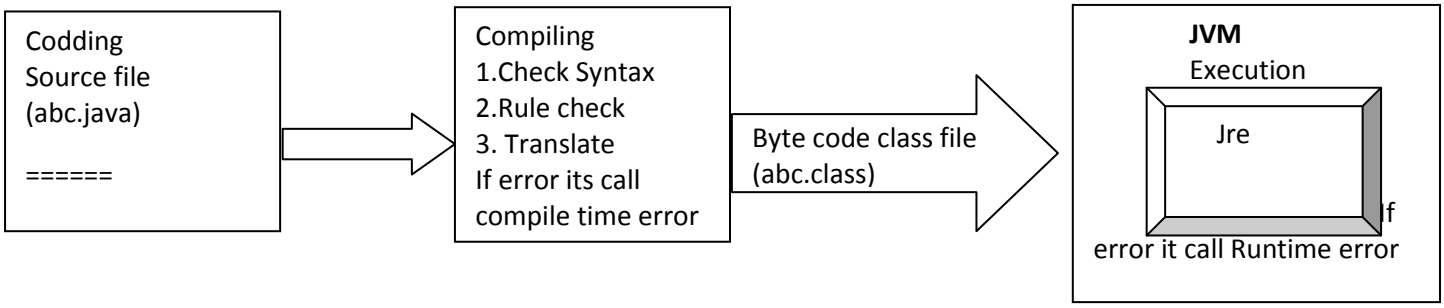
(Keshav Sir)

Done By Azam Amir Reza(azam20104u@gmail.com)

10-Aug-15:

Basic Knowledge of programming.

11-Aug-15:



- A.java creating with any OS then compiles it and create byte code .class file and then copy it to any other device or OS like mac, Unix, solarize to run it but one thing remember that every os must have jre ….
- Java program should be return the syntax java statements and it should be save as ".java".
- The ".java" file is known as source file.
- The java compiler is used to compile the source code to the byte code. Before compilation compiler check the syntax and the rules of java language if the source code is not following the syntax and rules then compiler those error which is known as compile time error (CTE).
- The compiler generate ".class" file which content the byte code format. ".class" file is executable file in java language
- Jvm is responsible for executing the ".class file", it is an interpreter which introvert the byte codes and execute it to provide the result.
- Jre stands for java ran time environment provides an environment for jvm to execute the ".class file" .Jre is system library which interacts with os but executing the program.
- The ".class" file can be executed on any os provided in that os jre is installed. Hence java is known as platform independent.
- ".class" file is jre dependent but independent of os.
- Jre is develop separately for each os

You can write java source file with Text Editor-notepad, edit+, notepad ++.
Or IDE (Integrated Development Environment) –eclipse, net beans.

Java edition:

1) Java SE- standard edition-(desktop edition)
2) Java EE-enterprise edition –(web app)
3) Java ME- Micro edition-(embedded application)

12-Aug-15

public class Program1 {----→(class declaration)

public static void main(String[] args) {-------->(main method)

System.out.println("Hi Three tech");----→print something.

}

}

Compilation (command prompt)

C:\Users\Programmer>D:

D:\>cd D:\JECM3_1\BASICS\INTRODUCTION

D:\JECM3_1\BASICS\INTRODUCTION>javac Program1.java

D:\JECM3_1\BASICS\INTRODUCTION>java Program1

Hi Three tech

- **OCJP**---→ORACLE CERTIFICATE JAVA PROGRAMMER (YOU MUST CLEAR THAT EXAM)

Ex,

```
class Program2 {
	public static void main(String[] args) {
	System.out.println("Jspider");//character
	System.out.println(12345);//intiger
	System.out.println(123.34);//floating point
	System.out.println(true);Boolean
	System.out.println('j');//character

	}
	}
```

Output is:

```
Jspider
	12345
	123.34
	true
	j
```

Ex,

+→numaric→addition

+→string→concatination

```
class Program3 {

	public static void main(String[] args) {

		System.out.println(20+20);//addition

System.out.println("java"+"program");//string concatination
```

```
	}
}
```

Output:

40

Javaprogram

Ex, of operator association

```
class Program4 {
	public static void main(String[] args) {
		System.out.println("Hello World!" +20);
		System.out.println("Hello World!" +20+20);
		System.out.println("Hello World!" +(20+20));//after string it will concat,it is based on the operator
		System.out.println(20+"Hello World!");
		System.out.println(20+20+"Hello World!");
	}
}
```

Output is:

Hello World!20

Hello World!2020

Hello World!40

20Hello World!

40Hello World!

Note:

Values:

1) Numeric:

1) Integer
2) Float point

2) Character: ‘ ‘

3) Boolean: true or false

4) String: " "

Keywords:

- Predefined words
- Has its own meaning
- Reserved words
- Lower case

E.g. public, void

Identifier:

- Name given by programmer
- Alpha numeric, should begin with alphabet
- Special underscore is allowed

Variable name/ method name:

- First later should be in lower case
- Subsequent words should begin with uppercase

Exam, empId; createSavingPoint

Class name:

- First later of each word should be in uppercase

Exam, AddEmploy, DeleteEmploy

13-08-15

Variable declaration

Data types	**Variable Name**
Byte	
Int	Integer number
Long	
Short	
Float number	Floating point
Double	
Char	Character
Boolean	Boolean values

Note , String is not a data type

Variable initialization:

Variable name= value;

Ex,

emId = 1856

empSalary = 12500.50

empGrade = ‘A’

Memory

Location Id	Memory Location
emId:	[1856]
empSalary:	[12500.50]
empGrade:	[‘A’]

Declaration, Initialization:

Ex,

```
class pro1 {

public static void main(String[] args) {
        //variable declaration
        int empId;
        double empSalary;
        char empGrade;

//variable initialization
        empId=12649;
        empSalary=1234.500;
        empGrade='A';

        System.out.println("empId "+empId);
        System.out.println("empSalary "+empSalary);
        System.out.println("empGrade "+empGrade);

        }
        }
```

Output is:

```
empId 12649
        empSalary 1234.5
        empGrade A
```

Ex,

```
class pro1 {
	public static void main(String[] args) {

		//variable declaration

		int empId;

		double empSalary=1234.500;

		char empGrade='A';

		System.out.println("empId "+empId); //use the it but not initialize it so it gets error

		System.out.println("empSalary "+empSalary);

		System.out.println("empGrade "+empGrade);

	}

}
```

Output:

Error

Note,

In java language all local variables should be initialized before using in any of the operation. Other ways compiler goes error.

Ex, of re-assignment

```
class pro2 {
	public static void main(String[] args) {

		int n1=12;
		System.out.println("n1 " +n1);

		n1=23;//re-assign
		System.out.println("n1 " +n1);

		n1=45;//re-assign
		System.out.println("n1 " +n1);
	}
	}
```

Output:

```
n1 12
	n1 23
	n1 45
```

Ex, Final value:

```
class  pro3{
	public static void main(String[] args) {

		final int n1=34;//value assigned will be final
		System.out.println("n1 value: " +n1);

		n1=45;//error, cannot re-assign value to final variable
		System.out.println("n1 value: " +n1);
		}
	}
```

Note, in java language constant can be achieved declaring as final.

Whenever variable declaring as final we can initialized only once but we cannot re-initialized in the code again.

Ex. of Normal value which we can reassign

```
class pro4 {
	public static void main(String[] args) {
		int n1=25;
		int n2=45;
		int n3=0;

		System.out.println("n1 value !"+n1);
		System.out.println("n1 value !"+n2);
		System.out.println("n1 value !"+n3);

		n2=n1;//copy value of n1 to n2
		n3=n2;//copy value of n2 to n3

		System.out.println("n1 value !"+n1);
		System.out.println("n1 value !"+n2);
	}
	}
```

Output is:

```
n1 value !25
	n1 value !45
	n1 value !0

n1 value !25
	n1 value !25
	n1 value !25
```

Ex, Use of final value or where we can use the final value:

```
class  pro6{
	public static void main(String[] args) {

		double r1=2.1;
		final double pi=3.14;
		double a1,c1;

		a1=pi*r1*r1;
		c1=2*pi*r1;
```

```
                System.out.println("area : "+a1);
                System.out.println("cercun : "+c1);

        }
}
```

Output:

area : 13.8474
cercun : 13.188

Ex, long and float value

```
class pro5 {
        public static void main(String[] args) {


        long empPhNum=999999999999999l;
        float empsal=12334335.34f;

        System.out.println("emp ph no "+empPhNum);
        System.out.println("emp sal no "+empsal);

        }
        }
```

Note, if u not use "l" in the last in long value then it getting error because of default value is integer in java

And also float value "f" in default in java is double value

14-Aug-15:

Use of Unary Operator: ++ or –

Int K=0;

K++=post increment

++k=pre increment

Ex,

```
class Pro7 {
        public static void main(String[] args) {

                int k=0;
                k++;
                System.out.println("after incr: "+k);
                k--;
                System.out.println("after dec: "+k);
```

```
	}
	}
```

```
class Pro8 {
	public static void main(String[] args) {

		int i=0;
		int j=0;

		j=i++;
		System.out.println("i value "+i);
		System.out.println("j value "+j);

	}
	}
```

Output is:

```
i value 1
	j value 0
```

Ex,

```
	class Pro8 {
		public static void main(String[] args) {

		int i=0;
		int j=0;

		j=i +i++;
	//0+0
	// use current value of i and than increment value of i by 1

		System.out.println("i value "+i);
		System.out.println("j value "+j);

	}
	}
```

Output is:

```
i value 1
	j value 0
```

Ex,

```
	class Pro8 {
		public static void main(String[] args) {

		int i=0;
		int j=0;
```

```
		j=I +i++ + i;
		//0+0+1
		// use current value of i and then increment value of i by 1

		System.out.println("i value "+i);
		System.out.println("j value "+j);

	}
	}
```

Output is:

```
1
	1
```

Ex,

```
class Pro9 {
	public static void main(String[] args) {

		int i=0;
		int j=0;

		j=++I;
		//1
		// first increment value of i by 1 and use the increment value

		System.out.println("i value "+i);
		System.out.println("j value "+j);

	}
	}
```

Output is:

```
1
	1
```

Ex,

```
class Pro9 {

	public static void main(String[] args) {

		int i=0;
		int j=0;

		j=++i + ++i;
		//1+2 first increment value of i by 1 and use the increment value

		System.out.println("i value "+i);
		System.out.println("j value "+j);
	}
	}
```

Output

```
	2
	3
```

Ex,

```
class pro9 {

	public static void main(String[] args) {

		int i=0;
		int j=0;

		j=i + ++i +i++ + i;
	//0+1+1+2

		System.out.println("i value "+i);
		System.out.println("j value "+j);
	}
	}
```

Output is:

2
4

Ex,

```
	class pro9 {
		public static void main(String[] args) {

		int i=0;
		int j=0;

		j=++i + ++i + i + i++ + ++i;
		//1+2+2+2+4

		System.out.println("i value "+i);
		System.out.println("j value "+j);

	}

}
```

Output is:

4
11

Methods:

- To implement a task /operation
- Reusability--- write once /run many times

Syntax<Modifier> retuntype methodname (<argument>)

```
	{
	Function
	Return value;
	}
```

Discus more in OPPS concept continue …….

Tuesday August 18, 2015:

Control Statements:

Switch Case:

Ex,

```
class Pro5 {

	publicstaticvoid main(String[] args) {
		char grade='A';
		switch(grade){
		case'A': System.out.println("first class with distinction");
		break;
		case'B': System.out.println("first class");
		break;
		case'C': System.out.println("Second class");
		break;
		case'D': System.out.println("Just pass");
		break;
		case'E': System.out.println("Get Lost");
		break;
		default: System.out.println("Invalid grade");
		}

	}

}
```

Out put:

```
first class with distiction
```

Note: switch case jump to the correct statement. But if is test one by one check condition statement.

For Loop:

```
class Pro6 {
	publicstaticvoid main(String[] args) {
	for (int i=1; i<=5; i++){
		System.out.println("I Love Programming");
		}
	}
}
```

o/p:

```
I Love Programming
I Love Programming
I Love Programming
I Love Programming
I Love Programming
```

Ex, of Increment.

```
class Pro6 {
	publicstaticvoid main(String[] args) {
```

```
	for (inti=1; i<=5; i++){
	System.out.println(i);
	}
}
}
```

o/p:

```
1
2
3
4
5
```

Ex, of decrement:

```
class Pro6 {
	publicstaticvoid main(String[] args) {
	for (int i=5; i>=1; i--){
		System.out.println(i);
	}
}
}
```

o/p:

```
5
4
3
2
1
```

Ex, Use of print statements:

```
class Pro6 {
	publicstaticvoid main(String[] args) {
	for (int i=5; i>=1; i--){
		System.out.print(i);
	}
	System.out.println();
	}
}
```

o/p:

```
54321
Program ended
```

Ex6, Use of Inner for loop:

```
class Pro6 {
	publicstaticvoid main(String[] args) {
		for(int k=1; k<=5; k++){ //rows
			for (int i=5; i>=1; i--){ //column
				System.out.print("*");
			}
	System.out.println();
		}
	System.out.println("Program ended");
	}
}
```

o/p:

```
*****
*****
*****
*****
*****
```

Ex,

```
class Pro6 {
	publicstaticvoid main(String[] args) {
		for(int k=1; k<=5; k++)//rows
		{
			for (int i=5; i>=1; i--)//column
			{
				System.out.print(i+" ");
			}
	System.out.println();
		}
	System.out.println("Program ended");

	}

}
```

o/p:

```
	5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
5 4 3 2 1
Program ended
```

Ex,

```
publicclass Pro6 {
	publicstaticvoid main(String[] args) {
		for(int k=1; k<=5; k++)//rows
		{
			for (int i=5; i>=1; i++)//column
			{
				System.out.print(k+" ");
			}
	System.out.println();
		}
	System.out.println("Program ended");

	}

}
```

o/p:

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Ex,

```
public class Pro6 {
	public static void main(String[] args) {
	for(int k=1; k<=5; k++){ //rows
		for (int i=1; i<=k; i++){ //column
		System.out.print(i+" ");
		}
		System.out.println();
	}
	System.out.println("Program ended");
	}
	}
```

o/p:

```
	1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
Program ended
```

Ex,

```
public class Pro6 {
	public static void main(String[] args) {

		for(int k=5; k>=1; k--)//rows
		{
		for (int i=1; i<=k; i++)//column
		{
		System.out.print(i+" ");
	}
	System.out.println();
	}
	System.out.println("Program ended");                                                    }
                                                                                        }
```

o/p:

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

Program ended

19-Aug-2015:

While loop:

Ex,

```
class Pro7 {
	public static void main(String[] args) {
	int i=1;
	while (i<=5)
	{
		System.out.print(i);
		i++;

		}
	}
	}
```

o/p:

```
12345
```

If Statements:

Ex,

```
class Pro7 {
	public static void main(String[] args) {
	int i=5;
	if (i>7)   //check the condition is true or not
	System.out.print("number above 7");
	System.out.print("number below 7");
	//if multi statements then u should use curly braces otherwise you will get both statements

	}

}
```

```
class Pro7 {
	public static void main(String[] args) {
	int i=5;
	if (i<7)   //check the condition is true or not
	{
	System.out.print("number above 7");
	}
	System.out.print("number below 7");
	//if multi statements then u should use curly braces otherwise you will get both statements

	}

}
```

o/p:

```
number below 7
```

- We can write for loop below

```
class Pro9 {
        public static void main(String[] args) {

        int i=1;
        for (; i<=5; i++ ){
        System.out.print(i);
        }
        System.out.println();
        }
        }
```

Ex,

```
class Pro9 {
        public static void main(String[] args) {
        int i=1;
        for (; i<=5;){
        System.out.print(i);
        i++;
        }
        System.out.println();

        }
        }
```

```
class Pro10{
        static void square(int n){ //int arg method
        System.out.println("Square of n "+n);

        int res=n*n;
        System.out.println("Square is "+res);

        if(n<=9){
        square(n+1);//recursive call,
        }
        }

public static void main(String[] args) {
        square(5); //passing int value to the method
        }
        }
```

o/p:

```
Square is 25
Square of n 6
Square is 36
Square of n 7
Square is 49
Square of n 8
Square is 64
```

Square of n 9

Square is 81

Square of n 10

Square is 100

- ❖ It runs perfectly but if the condition change like if(n>=9) and pass the value like square (10) then error is occurred. called stackoverflow .

Steps to Read Input From User:

Step1: import java.util.Scanner;

Step2: Scanner sc1=new Scanner (System.in);

System.in=represent standard input device

Step 3; use methods to read inputs

1) nextInt()➔used to read integer value from keyboard return type of this method is integer.
2) nextDouble()➔used to read double value from keyboard ,return of this method is double
3) next()➔ used to read string value from keyboard ,return of this method is string .

Ex,

```
import java.util.Scanner;

class Pro11 {
	public static void main(String[] args) {

		Scanner sc=new Scanner(System.in);
		String stName;
		int stAge;

		System.out.println("Enter your Name: ");
		stName=sc.next();

		System.out.println("Enter Your age: ");
		stAge=sc.nextInt();

		if (stAge>18){
		System.out.println("eligable for vote");

		}else{
		System.out.println("not eligable!");
		}
	}
	}
```

o/p:

```
Enter your Name: Azam
Enter Your age: 22
eligable for vote
```

Ex,

```
import java.util.Scanner;

class Pro12 {
	static void square(int n){
	System.out.println("Square of n "+n);

	int res=n*n;
	System.out.println("Square is "+res);

	if(n>1){
	square(n-1);//recursive call
			//when error is called stackoverflow .
		}

	}

public static void main(String[] args) {
	Scanner sc=new Scanner(System.in);
	System.out.println("Enter any number");
	int num=sc.nextInt();
	square(num);
	}
	}
```

20-Aug-15:

Array:

int x1=4, int x2=5,int x3=6,int x4=7,

int n=n; In above we declare variable and value like this , but if we need same data type value at that time we need create array .Array is nothing but a collection of same Data type.

Array declaration:

Datatype [] array name;

Array initialization:

Arrayname= new datatype[size];

Declaration and initialization:

Int[] arr1= new int[5];

Default value

0	0	0	0	0

Index 0 1 2 3 4

Ex,

```
class Pro1 {
	public static void main(String[] args) {
	System.out.println("Program started");

		int[] arr1=new int[5];

		//array of int type , we store 5 int values

		// every array has a property name length, which contains size of the array

		System.out.println("array size"+arr1.length);

		//store values separately

		arr1[0]=25;

		arr1[1]=26;

		arr1[2]=27;

		arr1[3]=24;

		arr1[4]=29;

		System.out.println("4th element"+arr1[3]);

		for(int i=0; i<arr1.length; i++){

		System.out.println(arr1[i]);

		}

		System.out.println("Program ended");

	}

}
```

o/p:

```
Program Start
24

25
26
27
24
29
Program ended
```

Ex,

```
class Pro1 {
	public static void main(String[] args) {
	System.out.println("Program started");

		int[] arr1=new int[5];
```

```
		//array of int type , we store 5 int values
		// every array has a property name length, which contains size of the array
		System.out.println("array size"+arr1.length);
		//store values
		arr1[0]=25;
		arr1[1]=26;
		arr1[2]=27;
		arr1[3]=24;
		arr1[4]=29;
		System.out.println("Array element in reverse order");
		for(int i=arr1.length-1; i>=0; i--){
		System.out.println(arr1[i]);
		}
		System.out.println("Program ended");
	}
}
```

o/p:

reverse output is happen.

Ex,

```
import java.util.Scanner;
class  Pro2{
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.print("Enter the array size: ");
		int size=sc.nextInt();
		int [] arr=new int[size];//read values from keyboard
		System.out.print("Now Enter the array elements: ");
		for (int i=0; i<arr.length ; i++ ){
			System.out.println("<"+i+">");
			arr[i]=sc.nextInt();
			}
		System.out.println("============================");
		System.out.println("Arry elements");
		for (int i=0; i<arr.length ; i++ ){
			System.out.println(arr[i]);
```

```
                    }
                }
        }
```

o/p:

```
Enter the array size: 5
Now Enter the array elements
<0>1
<1>2
<2>3
<3>4
<4>5
============================
Arry elements are:
1
2
3
4
5
```

Ex,

```
import java.util.Scanner;

class  Pro4{

        //method arg is arry type
        // print the element of given arry

        static void printArray(int[] arg){
        System.out.println("Arry elements");
                for (int i=0; i<arg.length ; i++ ){
                System.out.print(arg[i]+" ");
                }
        }

        /* methods takes an argument of in array type.
        Fills the array with values and returns
        Filled array*/

        static int[] fillArray(int[] arg){

        System.out.println("filling an  Array ");

        Scanner sc=new Scanner(System.in);

        //read values from keyboard

                for (int i=0; i<arg.length ; i++ ){

                System.out.print("<"+i+">");

                arg[i]=sc.nextInt();//passing an arry

                }

                return arg;

        }
```

```
	public static void main(String[] args) {

		Scanner sc=new Scanner(System.in);

System.out.println("Enter the array size: ");

	int size=sc.nextInt();

	int [] arr=new int[size];

	int [] arr2=fillArray(arr);//passing an empty array

		printArray(arr2);
	}
	}
```

21-Aug-15:

String Function:

String str; →declaration.

Str=”jspiders”; →initialization.

How to store

Str variables Names

J	S	p	i	d	e	r	s
0	1	2	3	4	5	6	7

System.out.println(str);→jspiders

- **length()→**returns total no of chars in a given string
- **charAt**→returns char present at specified index
- **indexOf(char**)→returns first occurrence position of given char
- **contains(charseq**)→returns true if given sequence of char is present in the string
- **startsWith(charseq)--->**returns true if string begins with specified sequence of char
- **endsWith(charseq)→**returns true string ends with specified sequence of char
- **substring (startindex**)→returns string from the specified start index till end of the string
- **substring(startindex, endindex**)→returns string between start index and end index of the string, end index position is excluded
- **equals(string**)→returns true if the current string is equal to given string
- **toUpperCase(**)→returns string which is converted to upper case
- **toCharArray()→**converts the string to char array and returns the array
- **split(value**)→split the given string based on specified value and return string array

Pr1:

```
class Pro1 {
	public static void main(String[] args) {

	String str= "javadeveloper";
	System.out.println("given string"+ str);
```

```
int tot=str.length();
        System.out.println("total char"+tot);//o/p is:total char13

char c1=str.charAt(4);
        System.out.println("chaar @ 4 pos: "+c1);// o/p is: char @ 4 pos: d

int i1=str.indexOf('e');
        System.out.println("first occ of e : "+i1);//o/p : first occ of e : 5

int i2=str.indexOf('e',6);
        System.out.println("2nd occ of e : "+i2);//o/p: 2nd occ of e : 7

int i3=str.lastIndexOf('e');
System.out.println("Last occ of e: "+i3);//o/p: last occ of e: 11

                }
        }
```

Pro2:

```
class Pro2 {
        public static void main(String[] args) {

        String str= "javadeveloper";

System.out.println("given string: "+ str);// o/p: javadeveloper

boolean b1=str.contains("dev");
        System.out.println("contain 'dev' : "+ b1);//op: true

boolean b2=str.startsWith("java");
        System.out.println("contain 'java' : "+ b2);//op: true

boolean b2=str.endsWith("per");
        System.out.println("contain 'per' : "+ b2);//op: true

String s1=str.substring(8);
        System.out.println(s1);// op: loper

String s2=str.substring(4,11);
        System.out.println(s2);// op: develop

        }
        }
```

```
class Pro3 {
        public static void main(String[] args) {

                String str1= "javadeveloper";
                String str2= "javadeveloper";

                if (str1.equals(str2)){
                System.out.println("String are  same");
```

```
                }else{

                System.out.println("String are not same");

                }

        }

}
```

Pro4:

```
class Pro4 {
        public static void main(String[] args) {
        System.out.println("Program started");

        String[] str ={"Ramesh","Samesh","Tamesh"};
        System.out.println("array size: "+str.length);

        for (int i=0; i<str.length;i++ ){
        System.out.println(str[i].toUpperCase());
                }

        System.out.println("Program ended");
        }
        }
```

o/p:

```
        Program started
array size: 3
RAMESH
SAMESH
TAMESH
Program ended
```

Pro5:

```
class Pro5{

        public static void main(String[] args) {
        System.out.println("Program started");

        String str ="Ramesh Samesh Tamesh";
        System.out.println("given string: "+ str);

        String[] strArr=str.split(" ");

        for (int i=0; i<strArr.length;i++ ){
        System.out.println(strArr[i].toUpperCase());
        }
        System.out.println("Program ended");
        }
        }
```

Note,

- *Main method arguments should be string because is use*
- *Whenever we execute program JVMcreates an array of strings file the values of command line and passé to main function of the class which is be executed.*

Tomorrow pattern

Pattern is done by Teja Mem.

24-Aug-15:

Object Oriented Program (OOPS):

(whatever we see it's an object)

Ex, 1:

```
class Demo1 {//functional class
        static void test() {
        System.out.println("Running user define method");
        }
        }

class Demo2{//functional class
        static void disp() {
        System.out.println("running user define method");
        }
        }

class Demo3 {//Main class
        public static void main(String[] args) {
        System.out.println("running demo3");
        }
        }
```

Note ,

When it compile its compile three class but run only main method class.

In java src file any no of class body unique the function the compiler generates separate class file for each class body, we can or jvm can execute only those class file which as main method.

class class_name-→class declaration

{

Member of class:

1) Variable←data member
2) Methods←function member

Member type:

1) Static member/ class member
2) Non-static member/instance member.

Ex,

```
Clss demo1{

        static int k=12;//static data member
        Int j=34; //data member

        static void test(){ →static function member
        Int i=o;→local variable
        }

        Void disp(){→non static fun
        }

Class demo 2{

        static int k=12;
        static void test(){
                // static mem

}
```

Access spacifier:(restrict access to members)

1) Private
2) Package level
3) Protected
4) public

How to access static members of class?

Syntax,

classname.StaticMemberName;←is called reference.

Ex,

```
class Demo1{  //functional class
        static int k=12;
        static void test() {
        System.out.println("Running user define method");
        }
        }

class Demo2{//functional class
        static void disp() {
        System.out.println("running user defuine method");
        }
        }

class Demo3{  //Main class
        public static void main(String[] args) {
        System.out.println("k value: "+Demo1.k);
        Demo1.test();
        }
        }
```

o/p:
k value: 12
Running user define method

Ex,

```
class D2{

        static int i=12;
        static int j=13;
        }

class MainClass2 {
        public static void main(String[] args) {

        System.out.println("i value: "+D2.i);
        System.out.println("j value: "+D2.j);
        System.out.println("----------------------- ");

        D2.i=67;//reassignment
        D2.j=23;//reassignment

        System.out.println("i value: "+D2.i);
        System.out.println("j value: "+D2.j);
        }
        }
```

- Class definition block is used to define the members of class ,
- The member can be data member or function member
- The data members are used to store data required for the program when as the function members are used to operate on the data members
- The members are classified into tow type static and non-static,
  a) The static members are declared using static keyword and is also known as class members
  b) The non-static member declared without static keyword it's also known as instance members of the class.
- Static data members of a class can be rename by any class
- The members of a class can be access from any class based on the access spicier
- The static members of a class are refer by using class name

Ex,

```
class D3{
        static int x;//data member
        static void test(){
        int i=56;//local variable
        System.out.println("i valu: "+i);
        System.out.println("x valu: "+x);
        //you can access global variable to anywhere e.g. "x" but local variable can't use outside the block e.g. "I"
        }
        }

class MainClass3 {
        public static void main(String[] args) {
        System.out.println("x valu: "+D3.x);
        D3.test();
        }
        }
```

o/p: x value 0

I value 56

X value 0

25-Aug-15:

How to access non-static member:

```
class Demo1{
        Int k=20;//non-static member
        Void test(){// non-static function
        }
        }
```

Instance creation or object Creation:

1) New Operator
2) Constructor

new Constructor();

constructorname==class name

A copy of all non-static members of the class are loaded in the memory.

Ex,

Demo1 obj1=new Demo1();
K[20]
Test(){
==
==
}
Memory location
	Demo1 obj2=new demo1();
	K[20]
	Test(){
	==
	==
	}
	Memory location
		Demo1 obj3=new demo1();
		K[20]
		Test(){
		==
		==
		}
		Memory location

Reference variable:

- Used to refer instance of class
- Are declared using class name

Syntax className reference_variable_name;
Demo1 obj1;

1) Obj1-null
2) Obj1=new Demo1();

Ex,

```
class Demo1{
        int k=20;
        void test() {
        System.out.println("Runnig non-static method");
```

```
	}
	}

class MC4 {
	public static void main(String[] args) {

		Demo1 obj1=new Demo1();
		System.out.println("k value is : "+ obj1.k);
		obj1.test();
		System.out.println("obj1 location value is : "+ obj1);
		}
	}
```

****

```
Class Demo2{

Int x=20;

Int y=40

}

//instance

Demo2 obj1=new Demo2();

Demo2 obj2=new Demo2();

Syso(obj1.x)→20;

Syso(obj1.y)→40;

Obj1.x=50;//re assignment of obj1 object, it will not effect to any other object like obj2 even main class Demo1

Obj1.y=60;//re assignment



Sop(obj2.x)→20

Sop(obj2.y)→40

Sop(obj1.x)→50

Sop(obj1.x)→60
```

****

```
Ex,

class Demo2 {

	int x1=20;
	int x2=40;
	}
```

```
class MC5{
	public static void main(String[] args) {

		Demo2 obj1=new Demo2();
		Demo2 obj2=new Demo2();

		System.out.println("1st insatance x1 valu: "+obj1.x1);
		System.out.println("1st instance x2 valu: "+obj1.x2);

		System.out.println("2nd instance x1 valu: "+obj2.x1);
		System.out.println("2nd insatnx1 valu: "+obj2.x2);
		System.out.println("-------------------------- ");

		obj1.x1=50;
		obj1.x2=60;

		System.out.println("1st insatance x1 valu: "+obj1.x1);
		System.out.println("1st instance x2 valu: "+obj1.x2);

		System.out.println("2nd instance x1 valu: "+obj2.x1);
		System.out.println("2nd insatnx1 valu: "+obj2.x2);
		}
		}
```

****

Ex,

```
class Demo3{
	double y1=56.23;
	boolean b1=true;
	{
	System.out.println("Hello World!");
	}
	}

class MC6 {
	public static void main(String[] args){
	Demo3 obj1=new Demo3();

	Demo3 obj2;
	obj2=obj1;//copy value obj1 to obj2

	System.out.println("obj1 value: "+obj1);
	System.out.println("obj2 value: "+obj2);

	obj1.y1=98.23;  //now it also reassign to obj2
	obj1.b1=false;

	System.out.println("y1 value: "+obj2.y1);
	System.out.println("b1 value: "+obj2.b1);

	}

}
```

****

26-Aug-15:

Class and Objects:

Any entity digit with own class, state and behavior is known as object,the staterepresent the character of the object whereas the behavior represents functionality of the objects

- A class is design of objects which defines the properties of the objects
- A class is object which defines states and behaviorof an object.
- States of the objects are defined by data members whereas the behaviors of the objects are define by non-static function member of the class.

Reference variables:

- Any references the instance or of the class can be created by using new operator and constructor of the class.
- The new operator is used to instance of class whereas the constructors are used to initialize the instance.
- Whenever we create instance of the3 class copy of non-static member are loaded in the memory, to refer each instance we shouldcreatereference variable or object references.
- Reference variables are special variable which is used to refer an instance of the classes.
- The reference variable should be declared by using class name.
- For a reference variable we can assign either null value or an instance of class.
- An instance can by referred by any no of reference variables. In such case modifying the state of the instance from 1 reference variable will reflect in another reference variable also
- The instance created will always have non-static data member and non-static function member
- The modifying the states of one statements will not update the entire instance.

The variables declaration using data type are known as positive variable, is used to store primitive values.

****

Ex, of create objects:

```
class  Circle{

        double rad;
        final static double pi=3.14;

         void area() {
         double a1=pi*rad*rad;
        System.out.println("area is: "+a1);
        }

        void circum() {
        double c1=2*pi*pi;
        System.out.println("circum is: "+c1);
        }
        }

class MC7{
        public static void main(String[] args) {

        Circle cir1=new Circle();
        cir1.rad=2.1;
        cir1.area();
        cir1.circum();
```

```
        Circle cir2=new Circle();
        cir2.rad=3.1;
        cir2.area();
        cir2.circum();

        }

}

**********************************************

class NoteBook{

        //data members
        int pages=50;
        double price=25.50;

        //function member
        void open() {
        System.out.println("oppening");
        }

        void close() {
        System.out.println("closing");
        }

        void turnpages() {
        System.out.println("turn pages");
        }

}

class MC9{
        public static void main(String[] args) {

                NoteBook b1=new NoteBook();
                b1.open();
                b1.close();
                b1.turnpages();
        }
        }
```

27-Aug-15:

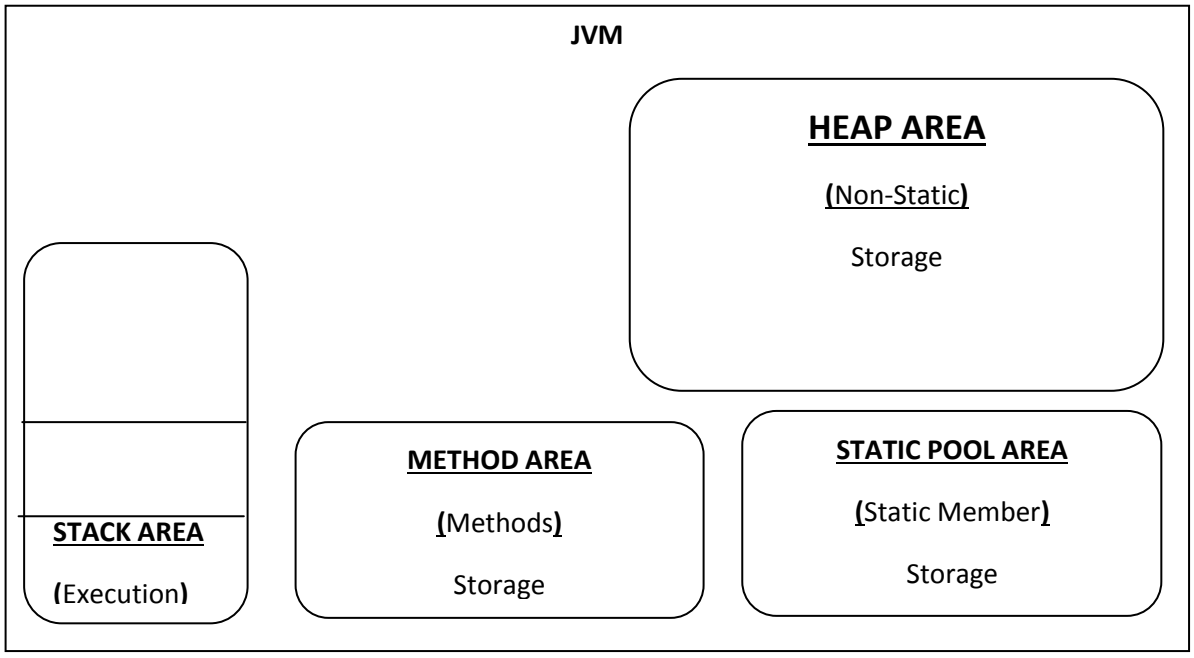
MEMORY ALLOCATION:

To execute any program the JVM makes use of following memory area:-

1) **Heap Area:**
   The Heap Area is used to store the instance in the program. In the heap area the memory allocated is random. The new operator loads the non-static member into heap area.
2) **Static Pool Area:**

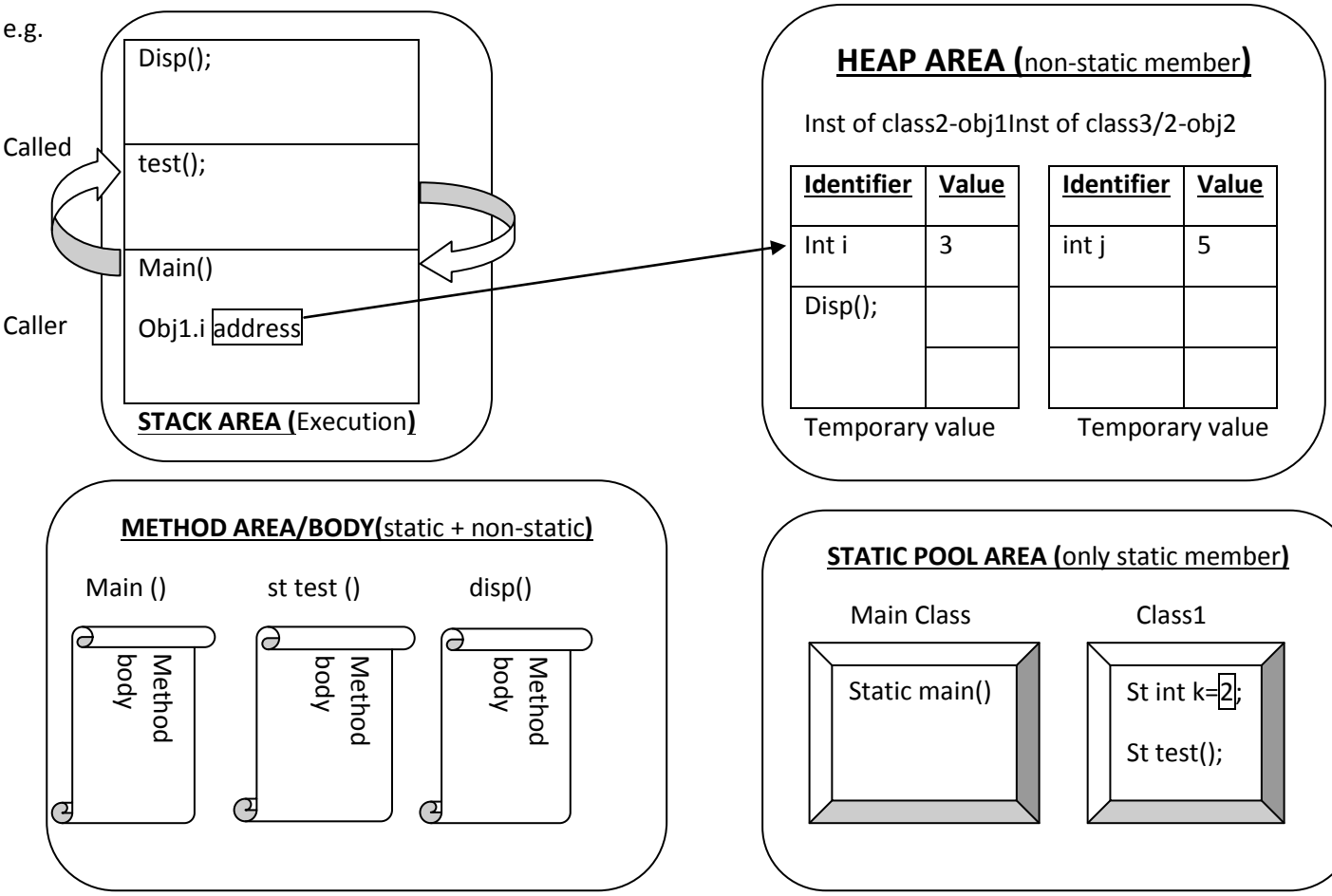
This area is used for storing the static member of the class .The pool will be created for each class, the class loader program of JVM is responsible to load the static member of the class to the static position.

3) **Method Area:**
   The method area is used to store the definition statements of methods.
4) **Stack Area:** The stack area is used for execution purpose and normal statements to which has to be executed in JVM should be come to stack area.
   The local variable components always store in stack memory area.



Garbage collector

e.g.



Ex,

```
class Demo1 {
static int k=25;
int j=5;
}

class MainClass1 {
                                                        public static void main(String[] args) {

        System.out.println(" static k value: "+Demo1.k);//call static member from static pool area

     Demo1 obj1=new Demo1();// obj1 object create to  Heap Area and address the member of class Demo1

System.out.println(" non-st j value: "+obj1.j);//calling from heap area

}

}
```

Ex,

```
class Demo2{
        static void test() {
        System.out.println("Running static test() method");
        return;
        }

        void disp(){
        System.out.println("Running non-static disp() method");
        }
        }

class MainClass2 {
        public static void main(String[] args) {

                Demo2.test();    //calling static method from static pool area where store Demo2 class static member
                Demo2 obj1=new Demo2();
                 //obj1 object create to  Heap Area and address the member of class Demo1
                obj1.disp();//address disp method to heap area where created obj1 object.
                return;
        }
        }
```

Ex,

```
class Demo3{
        static void test() {
        System.out.println("Running static test() method");
        return;
        }

        void disp(){
        System.out.println("Running disp() method");
```

```
        Demo4 obj2=new Demo4();
        System.out.println("x value: "+obj2.x);
        }
}

class Demo4{
        int x=34;
}

class MainClass3 {
        public static void main(String[] args) {
        Demo3.test();
        Demo3 obj1=new Demo3();
        obj1.disp();
        return;
        }
}
```

28-Aug-15:

Blocks (static & non-static**):**

Ex,

```
        class MainClass1 {
                static{
                System.out.println("Running 1st st block");
                 }

        public static void main(String[] args) {
        System.out.println("Running main");
        }

        static{
        System.out.println("Running 2nd st block");
        }
        }
```

o/p:

```
Running 1st st block
Running 2nd st block
Running main

*****************
```

Ex,

```
class D1{
        int k=20;
        static{
        System.out.println("Running D1 class st block");
        }
        }

class MC2{
        static{
```

```
        System.out.println("Running main class st block");
        }

        public static void main(String[] args) {

                System.out.println("Running main");

                D1 obj=new D1();

System.out.println("k value "+obj.k);

        System.out.println("End main");

        }

}
```

```
o/p:Running main class st block
Running main
Running D1 class st block
k value 20
End main
```

Ex,

```
class D3{
        static int k=12;
        static{
        System.out.println("k value "+k);//12
        k=34;
        }

        static{
        System.out.println("k value "+k);//34

        Int k=45;
        System.out.println("k value "+k);//45
        }
        }

class MC3{
        public static void main(String[] args) {
        System.out.println("Running main");

        System.out.println("k value "+D3.k);//34

        System.out.println("End main");
        }
        }
```

```
         o/p:
        Running main
        k value 12
        k value 34
        k value 45
        k value 34
        End main
```

Ex,

```
class S1{
        int j=34;
        {
        System.out.println("Running non st block");
        }
        }

class MC4 {
        public static void main(String[] args) {
        System.out.println("p str");

        S1 obj1=new S1();
        System.out.println("j value "+obj1.j);


        S1 obj2=new S1();
        System.out.println("j value "+obj2.j);

        System.out.println("p end");



        }

}
```

o/p:

```
p str
Running non st block
j value 34
Running non st block
j value 34
p end
```

Java language provides initialize block to initialize the data member of the class

There are two types of blocks

1) **Static initialize block :**
- Static initialize block is used to initialize the static members of the class. The static block are executed at the time of class loading by class loader , we can define multiple static blocks ,JVMexecuted sequentially
- If main class run contains main method and static blocks then JVM first run static block and then main method

2) **Non-Static initialize:**
- The non-static initialize blocks is used to initialize the non-static data member of the class
- The non-static blocks is executed whenever instance class is to be created
- In a class we can define multiple non-st blocks , which will be executed sequentially

31-Aug-15:

Constructor:

Object creation:

1) New operator
2) Constructor of class

New constructor ()://initialize non-static data member

1) Allocate memory
2) Load non-static member
3) Calls constructor of class

EVERY CLASS MUST HAVE CONSTRUCTOR

1) Compiler defined constructor(default constructor)
2) User defined constructor.

Syntax, Constructor_name(arg)→constructor declaration

{

Write code to do initialize

}

1) Cons name must be same as class name
2) No return type.

Ex,

```
class Sample1 {
int k=34;

 Sample1() {
System.out.println("running user define constrructor");
}

 void test() {
System.out.println("running test method");
}
}

class MainClass1 {
public static void main(String[] args) {

Sample1 obj1=new Sample1();
        Obj1.test();
System.out.println("k value: "+obj1.k);

}

}
```

o/p:

```
running user define constructor
```

```
running test method
k value: 34
```

Ex,

```
class Sample1 {
int k;
 Sample1() //user define constructor{
System.out.println("running user define constrructor");
k=96;//initialize the k value in the constructor
}

 void test() {
System.out.println("running test method");
}
}

class MainClass2{
public static void main(String[] args) {
Sample1 obj1=new Sample1();
        obj1.test();
System.out.println("k value: "+obj1.k);

}

}
```

o/p:

```
running user define constrructor
running test method
k value: 96
```

Ex,

```
class S2{
        int k=21;

//user define arg constructor
//parameterized constructor
S2(int arg1){
System.out.println("Running user define parameterized constructor");

k=arg1;

}

}

class MC3 {
public static void main(String[] args) {

        S2 obj1=new S2(67);
        System.out.println("k value: "+obj1.k);
        System.out.println("-------------------------------------");
```

```
		S2 obj2=new S2(37);
		System.out.println("k value: "+obj2.k);

	}

}
```

o/p:

```
Running user define parameterized constructor
k value: 67
-----------------------------------
Running user define parameterized constructor
k value: 37
```

Constructor is continue…….

01-Sep-15:

Tejja mem Array String:

```
package tejja;

publicclass Mango {

	publicstaticvoid main(String[] args) {
		String str = "Mango";
		char[] ch=str.toCharArray();

		for(int i=ch.length-1;i>=0;i--){
			System.out.print(ch[i]);
		}
	}

}
```

```
package tejja;

publicclass Sun {

	publicstaticvoid main(String[] args) {
	 String str ="Sun rises in the east";
	 String[] str1=str.split(" ");

	//opsud be like "east rises in the Sun"

	 String temp;
	 temp=str1[0];
	 str1[0]=str1[str1.length-1];
	 str1[str1.length-1]=temp;
	for(int i=0;i<=str1.length-1;i++){
		System.out.print(str1[i]+" ");
	 }

	}

}
```

Assignment from tejja mem

String= “Sun rises in the east”

o/p=Sun sesir in eht east;

03-Sep-15:

Constructor:

Ex,

```
class S3{
        int k;
        double d;

        //user define argument constructor or parameterized constructor
        S3(int arg1,double arg2){
        System.out.println("Running int, double constructor");
        k=arg1;
        d=arg2;
        }

        void test(){
        System.out.println("k value: "+k);
        System.out.println("d value: "+d);
        }
        }

class MainClass4 {
        public static void main(String[] args) {

        S3 obj1=new S3(25,7.5);
        S3 obj2=new S3(95,47.5);
        obj1.test();
        obj2.test();
        }
        }
```

o/p:

```
Running int, double constructor
Running int, double constructor
k value: 25
d value: 7.5
k value: 95
d value: 47.5
```

Constructor Overloading:

Ex,

```
class S4{
        int k;
        double d;
```

```
	//constructor overloading
	S4() {
	System.out.println("running no arg cons");
	}

	S4(int arg1) {
	System.out.println("running int arg cons");
	k=arg1;
	}

	S4(double arg1) {
	System.out.println("running double arg cons");
	d=arg1;
	}

	S4(int arg1,double arg2) {
	System.out.println("running int, double arg cons");
	k=arg1;
	d=arg2;
	}

	void test(){
	System.out.println("k value: "+k);
	System.out.println("d value: "+d);
	}
	}

class MainClass5 {

	public static void main(String[] args) {

	S4 obj1=new S4();
	obj1.test();

	S4 obj2=new S4(23);
	obj2.test();

	S4 obj3=new S4(7.5);
	obj3.test();

	S4 obj4=new S4(45,95.4);
	obj4.test();
	System.out.println("Program Ended");
	}
	}
```

o/p:

```
running no arg cons
k value: 0
d value: 0.0
running int arg cons
k value: 23
d value: 0.0
running double arg cons
```

```
k value: 0
d value: 7.5
running int, double arg cons
k value: 45
d value: 95.4
Program Ended
```

Note,

- Constructors are special member of the class which is used to initialize the data members of the class.
- The constructors are executed whenever the instances of class are created.
- Every class must have constructor in order to create an instance of the class.
- The constructor can be created either by compiler or by user.
- The constructor define by compiler is known as default constructor
- Compiler defines a constructor if the class doesnot have any user defineconstructor. If in case the class is having user define constructor then the compiler not define default constructor.
- The constructor define by programmer is known as user define constructor .the user can define either 0 orwith argument constructor.
- The constructor defines with arguments constructor are known as parameter constructor.
- Whenever as object created using parameter constructor we need to pass value to the constructor
- The constructorcannot be declare as static

Constructor Overloading:

In a class defining multiple constructor with deferent parameters is known as constructor overloading

- When definingoverloading constructor the constructor should deferent in terms of parameter type or parameter length.
- Constructor overloading help us to create an instance of class to the different initialization.
- While defining a constructor the constructor name should be same as class name, and constructor should not have return type.

04-Sep-15:

Program define total object of a class:

Ex,

```
class Car {
        static int count;
        Car(){
         System.out.println("creating car instance");
         count++;
        }

        static void total(){
        System.out.println("total car manufacture: "+count);
        }
        }

class MC6{
        public static void main(String[] args) {
        for (int i=1;i<=15 ;i++ ){
        new Car();
        }
```

```
        Car.total();
        }
        }
```

o/p:

```
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
total car manufacture:  15
```

Ex,

```
class Car {
        String brand;
        static int count;
        Car(){
        System.out.println("creating car instance");
        count++;
        }

          Car(String arg1){
         System.out.println("creating car instance of brand: "+arg1);

         brand=arg1;
        count++;
        }
        static void total(){
         System.out.println("total car manu: "+count);
         }
        }

class MC7{
        public static void main(String[] args) {
        for (int i=1;i<=10 ;i++ ){
        new Car();
        }
        for (int i=1;i<=10 ;i++ ){
        new Car("Audi");
        }
        Car.total();

        }

}
```

o/p:

```
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
total car manu: 25
```

Ex,

```
class S1 {
        int k=12;
        {
        System.out.println("runing non static block");
        k=56;
        }

        S1(){
        System.out.println("runing S1 constructor");
        k=34;
        }
        }

class MC8 {
        public static void main(String[] args) {
        S1 s1=new S1();
        System.out.println("k value is: "+s1.k);
        }
        }
```

o/p

```
runing non static block
runing S1 constructor
k value is: 34
```

Use non-static block only so that you can easily increment overloaded constructor:

Ex,

```
class Car {
        String brand;
        static int count;


        {
        count++;
        }

         Car(){
        System.out.println("creating car instance");
         }

        Car(String arg1){
        System.out.println("creating car instance of brand: "+arg1);
        brand=arg1;
        }

        static void total(){
        System.out.println("total car manu: "+count);
        }
        }

class MC7{
        public static void main(String[] args) {

        for (int i=1;i<=10 ;i++ ){
        new Car();
        }
        for (int i=1;i<=10 ;i++ ){
        new Car("Audi");
        }
        Car.total();
        }
        }
```

o/p:

```
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
```

```
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
creating car instance of brand: Audi
total car manu: 20
```

Ex,

```
class Student{
        final     int id;
        String name;
        Student(int arg1,String arg2){
        System.out.println("creating instance!");

        id=arg1;
        name=arg2;
        }
        }

class MC9 {
        public static void main(String[] args) {

        Student st1=new Student(21458,"Ramest");
        Student st2=new Student(31458,"Suresh");

        System.out.println("id of first student: "+st1.id);
        System.out.println("Name of first student: "+st1.name);

        System.out.println("id of 2nd student: "+st2.id);
        System.out.println("name of first student: "+st1.name);
        }
        }
```

o/p:

```
creating instance!
creating instance!
id of first student: 21458
Name of first student: Ramest
id of 2nd student: 31458
name of first student: Ramest
```

In a class if a data member as declare as final then it mustbe initialize in any of the below braces

- At the time of declaration
- Using respective blocks
- Using constructor

For each instance if we need it very separate constant values then we should use constructor to initialize the final data members.

Location identify using "this":

Ex,

```
class Student{
	final	int id;
	String name;

	Student(int arg1,String arg2){
	System.out.println("creating instance!");
	id=arg1;
	name=arg2;
	}

	void dispinfo(){

	System.out.println("object: "+this);
	System.out.println("student id: "+this.id);
	System.out.println("student name: "+this.name);
	}
	}

class MC10{
	public static void main(String[] args){

	Student st1=new Student(21458,"Ramest");
	Student st2=new Student(31458,"Suresh");

	System.out.println("st1: "+st1);
	System.out.println("st2: "+st2);

	st1.dispinfo();
	st2.dispinfo();
	}
	}
```

o/p:

```
creating instance!
creating instance!
st1: Student@659e0bfd
st2: Student@2a139a55
object: Student@659e0bfd
student id: 21458
student name: Ramest
object: Student@2a139a55
student id: 31458
student name: Suresh
```

05-Sep-15:

For the local and global variable declaration time used to "this" key word.

Ex,

```
class Student{
	final	int id;//global variable
	String name;//global variable

	Student(int arg1,String arg2){
	System.out.println("creating instance!");
	id=arg1;
```

```
        name=arg2;
        }

        void dispinfo(){
        int id=1234;//local variable
        String name="Ashok";//local variable

        System.out.println("object: "+this);
        System.out.println("student id: "+this.id);
        System.out.println("student name: "+this.name);
        }
        }

class MC9{
        public static void main(String[] args) {

        Student st1=new Student(21458,"Ramest");
        Student st2=new Student(31458,"Suresh");

        System.out.println("st1: "+st1);
        System.out.println("st2: "+st2);

        st1.dispinfo();
        st2.dispinfo();
        }
        }
```

o/p:

```
creating instance!
creating instance!
st1:Student@659e0bfd
st2:Student@2a139a55
object:Student@659e0bfd
student id: 21458
student name: Ramest
object:Student@2a139a55
student id: 31458
student name: Suresh
```

If the global and local variable data member same at that time u should use"this" keyword.Otherwise its id=id is declare as local variable.

Ex,

```
class Student{
        int id;//global variable
        Student(int id){ //constructor variable is local variable but same name as global variable
        this.id=id; //initialize the constructor local variable to the global variable using this keyword
                }
        }

class MC11 {
        public static void main(String[] args){
        Student st1=new Student(1234);//pass the value to the constructor int id local variable
        System.out.println("student id: "+st1.id);//call to the global variable id
```

```
    }
}
```

o/p:

```
student id: 1234
```

- The java language provides special keyword by name “this” which is used to refer current object member
- “This” key word always points to the current object that is it refers to the current object
- “This” key word should be used either in a non-static method content or in the constructor body
- “This” keyword can’t be used in the static methods
- Whenever the local variable name and the data member name are same, in such case the data members name can be differentiate local variable name by using “this” keyword.

The constructor are used to initialize the object data members, after initialize the data member the constructor returns the object of the constructor

The return type of the constructor will be the class type

1) Passing the object
2) Returning object

07-Sep-15:

- ***Pass the object address to reference variable IF Method Argument is Class Type:***

Ex,

```
class Sample1{

int i=34;
double j=4.5;

 void disp() {
System.out.println("Runnigg disp of S1  ");
}
}

class D1 {
void test(Sample1 arg1) //method argument is class type
{
System.out.println("running test() of D1");
System.out.println("i value: "+arg1.i);
System.out.println("j value: "+arg1.j);
arg1.disp();
}
}

class MC1 {
public static void main(String[] args) {

D1 obj=new D1();
obj.test(new Sample1());//passing an instance of S1 Class
}
}
```

o/p:

```
running test() of D1
i value: 34
j value: 4.5
Runnigg disp of S1
program ended!
```

Passing a ref1 value of S1 class:

```
class MC1 {

        public static void main(String[] args) {

                D1 obj=new D1();

                S1 ref1=new S1();

                obj.test(ref1);

                //value of ref1 is copied to argument

                //passing areference of S1 Class

                System.out.println("program ended!");

        }

}
```

- U can change data member to the another class and it will copied to that class which is references and u will get change value in the main class

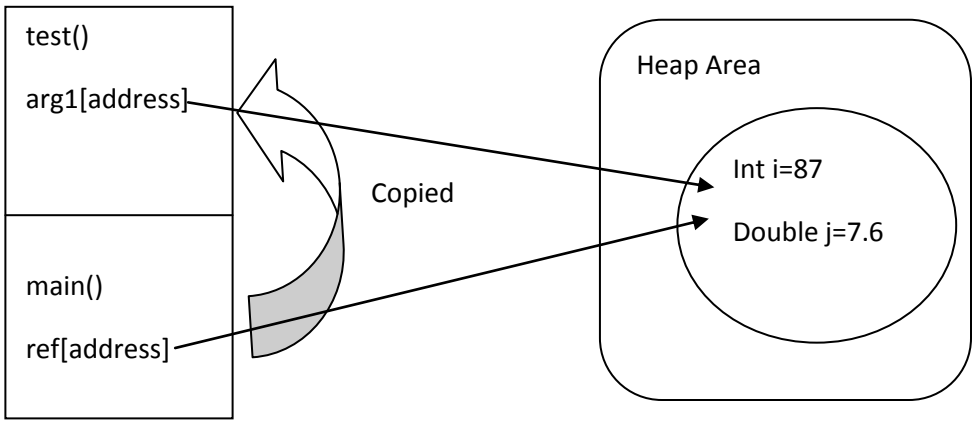
Ex,

```
class Sample1{

        int i=34;
        double j=4.5;

        void disp() {
        System.out.println("Runnigg disp of Sample1  ");
        }
        }

class Demo1 {

        void test(Sample1 obj1) //method argument is class type
        {
        System.out.println("running test() of Demo1 class");
        System.out.println("i value: "+obj1.i);
        System.out.println("j value: "+obj1.j);
        obj1.disp();

        obj1.i=87;//reassign and it will change the Sample1 class variable also
        obj1.j=7.6;//reassign
        }
        }
```

```
o/p:
running test() of Demo1 class
i value: 34
j value: 4.5
Runnigg disp of Sample1
i value: 87
j value: 7.6
program ended!
```

```
class MainClass1 {
        public static void main(String[] args) {

        Demo1 d1=new Demo1();
        Sample1 ref1=new Sample1();
        d1.test(ref1);//value of ref1 is copied to obj1
                //passing an reference of Sample1 Class

        System.out.println("i value: "+ref1.i);
        System.out.println("j value: "+ref1.j);
        System.out.println("program ended!");
        }
        }
```

- If the method argument is a class type then while invoking such method we have to pass the instance of the class mentioned in the method argument.
- The instance can be passed directly or we can pass the reference of instance also
- If we passing the ref of instance then any modification to that instance will reflect in the other references



***** Java does not support pass by ref since java language is not be using pointer concept ,however we can pass the object address to reference variable not true pointers.***

If Method return type is class type:

Ex 1,

```
class Sample1{
                int i=34;
                double j=4.5;
                 void disp() {
                System.out.println("Runnigg disp() of Sample1 class ");
                }
                }
```

```
class Demo1 {

        Sample1 test(){//method return type  is class type
        System.out.println("running test() of Demo1 class");

        retrun new Sample1();//return new Sample1()

            //returning instanceof Sample1 class

        }

}

class MainClass2 {

        public static void main(String[] args) {

            Demo1 d1=new Demo1();

            Sample1 ref1=d1.test();

            System.out.println("i value: "+ref1.i);

            System.out.println("j value: "+ref1.j);

            ref1.disp();

            System.out.println("Hello World!");

        }

}
```

Ex 2,

```
class Sample1{

        int i=34;
        double j=4.5;
        void disp(){
        System.out.println("Running disp() of Sample1 class");
        }
        }

class Demo1 {

        Sample1 test() {//method return type is class type
        System.out.println("Running test() of Demo1 class");

        Sample1 obj1= new Sample1();

        System.out.println("i value: "+obj1.i);
        System.out.println("j value: "+obj1.j);

        obj1.i=76;//reassign
        obj1.j=7.6;//reassign
```

```
        return obj1;
        }
        }

class MainClass3{
        public static void main(String[] args) {

        Demo1 d1=new Demo1();
        Sample1 ref1=d1.test();

        System.out.println("i value: "+ref1.i);
        System.out.println("j value: "+ref1.j);
        ref1.disp();
        }
        }
```

o/p:

```
Running test() of Demo1 class
i value: 34
j value: 4.5
i value: 76
j value: 7.6
Running disp() of Sample1 class
```

- If the method return type is class type then such method should return the instance.
- Whenever the call an method which has class type as a return type, we should define a reference variable to store the return object.

A java method can return only one value if we have to return multiple values of same type then go for arrays

If we have to return multiple values of diff type then go for objects

08-Sep-15:

A ref variable declare in the method context is known as local ref variable. The scope of local ref variable is limited to only that method context it can't be access from other method context until unless the method returns the reference.

```
void test(){

Int x=10;

Sample1 ref1=new sample1();//local object

}
```

Static and non-static reference variable:

Ex,

```
class Sample1{
        ====
}
```

```
class Demo1{

static Sample1 ref1=new sample1();//static ref variable
        Sample 2 ref2=new sample2();//non-static ref variable
            }
```

How to call in the Main class,
Demo1.ref1.member// for static ref variable

Demo1 d1=new Demo1();
D1.ref2.member;// for non-static ref variable

- A ref variable can be declare as the data member of the class
- It can be static data member or non-static data member
- If a ref variable static then we can only one copy of the ref variable in memory, this can also be call as static object
- If the ref variable is non-static then for each instance variable a copy of ref variable will be create this can also be called as non-static object.
- Whenever a ref variable is data member of the class then the memory allocation access either in the heap or in the static pool

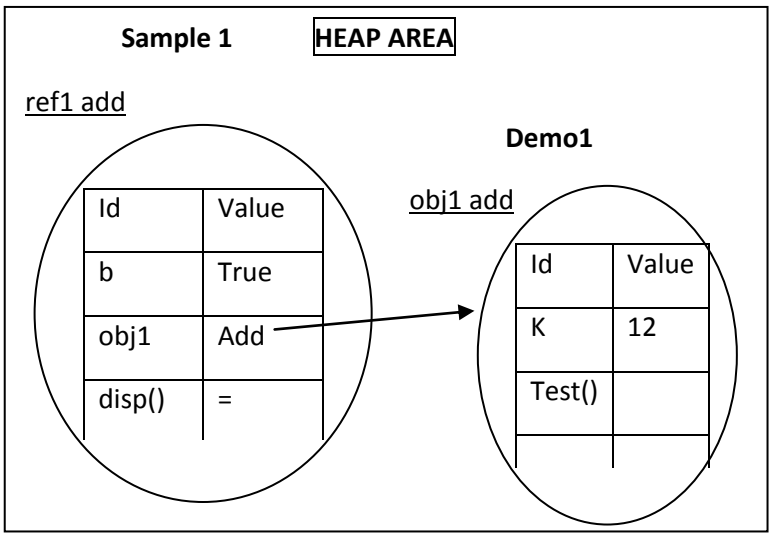
1) If a ref variable is declare as static and final then we get a single copy of ref variable which cannot be reassign
2) If a non-static ref variable is final we get multiple copy of ref variable for each instance which cannot be re assign

09-Sep-15:

Composition

Non-static reference variable or non-static object example:-

```
class Demo1 {
        int k=12;
        void test() {
        System.out.println("running test()");
        }
        }
        class Sample1 {
        boolean b=true;
        Demo1 obj1=new Demo1();//non-st ref variable of type Demo1

        void disp() {
        System.out.println("running disp()");
        }
        }

class MainClass1 {
        public static void main(String[] args) {

        Sample1 ref1=new Sample1();
        System.out.println("b value: "+ref1.b);
        ref1.disp();
        System.out.println("k value: "+ref1.obj1.k);
        ref1.obj1.test();
        }
        }
```



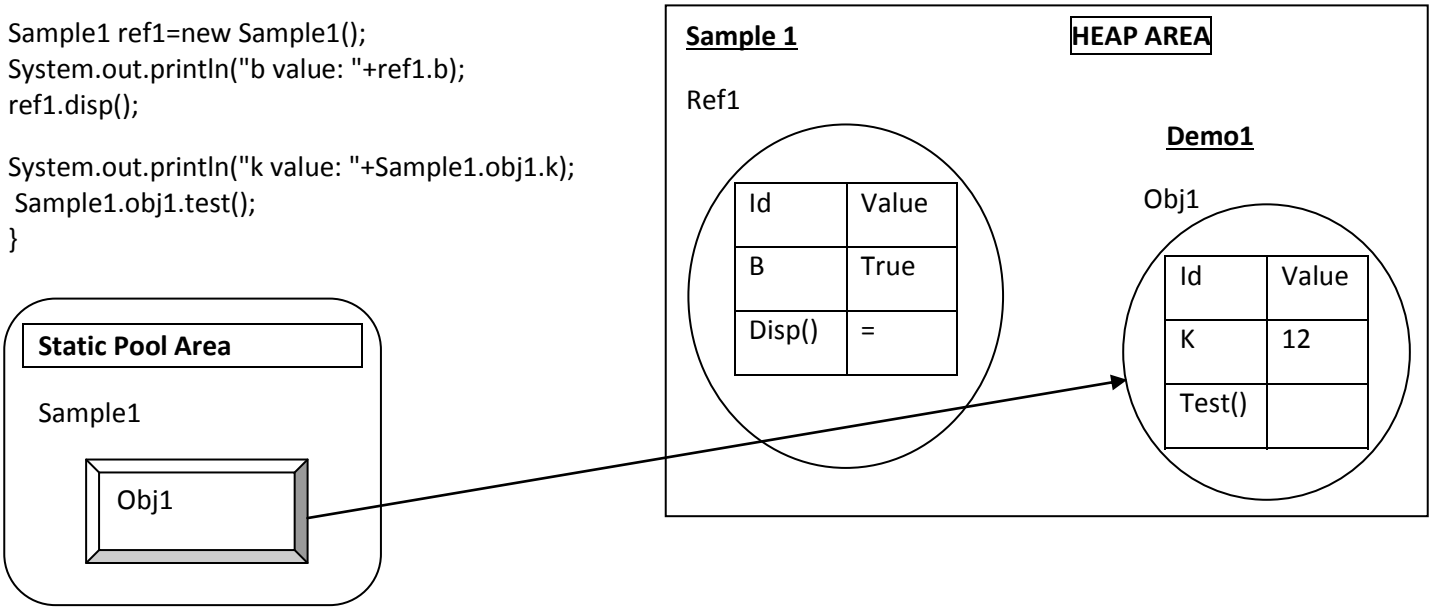
For output command is ” javac *.java”
then “java MainClass1”

o/p:

b value: true
running disp()
k value: 12
running test()

Static reference variable or static object:

```
class Demo1 {
        int k=12;
        void test() {
        System.out.println("running test()");
        }
}
class Sample1 {
        boolean b=true;
        static Demo1 obj1=new Demo1();//static ref variable of type Demo1
        void disp() {
        System.out.println("running disp()");
        }
}

class MainClass1 {
        public static void main(String[] args) {

        Sample1 ref1=new Sample1();
        System.out.println("b value: "+ref1.b);
        ref1.disp();

        System.out.println("k value: "+Sample1.obj1.k);
         Sample1.obj1.test();
        }

}
```



o/p:

b value: true
running disp()
k value: 12
running test()

INHERITANCE

- A class acquiring/gathering properties of another class is called as inheritance.
- The classes from where the properties are inherited are known as base class or Super class.
- The class through which properties are inherited is known as derived class or Sub class.
- Always subclass inherits the non-static properties from the super class.
- The static properties will never be inherited to the sub class.
- Whenever we create the instance of sub class that instance will always have the non-static properties of sub class and its super class

There are 4 types of inheritance:-

1) Single Inheritance: In this type of inheritance a sub class inherits properties from only one super class.
2) Multi-level inheritance: In this case the sub class inherits the properties of super class which inherits the properties of another super class. We can define any level of inheritance
3) Multiple Inheritance: Sub class inheriting from more than one super class is known as multiple inheritance java does not support multiple inheritance through classes.
4) Hierarchical Inheritance: In this type of inheritance more than one subclass inherits the properties from one super class, In other words the sub classes having common super class. This type of inheritance is used to achieve generalization.

Note,

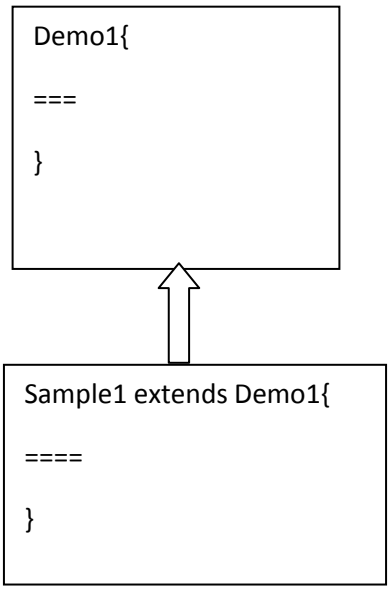
- If a class is declare as final, such classes can't have subclasses in other words we can't inherit the properties of final class. We can create an instance of final class and we can access it as non-static properties.
- A constructor of a super class will not be inherited to subclass, but constructor plays major rule in inheritance
- Whenever we create an object of sub class it will always have the non-static properties of any level super class
- Object is a super class for every java class
- Every java class should define a super class, if user is not define a super class then compiler define default super class by name "object"
- Each and every class created in java language is always having properties of object.

10-Sep-15:

Example of Single Inheritance when we create an instance of sample1 class then instance should be work on D1 class .

```
class Demo1 {

        int k=12;
        void test() {
        System.out.println("runing test() of Demo1 ");
        }
        }

class Sample1  extends D1{

        double d=45.67;
        void disp() {
        System.out.println("running disp() of Sample1");
        }
        }
```

```
class MainClass1 {
	public static void main(String[] args) {

		Sample1 ref1=new Sample1();
		System.out.println("d value: "+ref1.d);
		ref1.disp();

		System.out.println("K value: "+ref1.k);
		ref1.test();
		}
		}
```



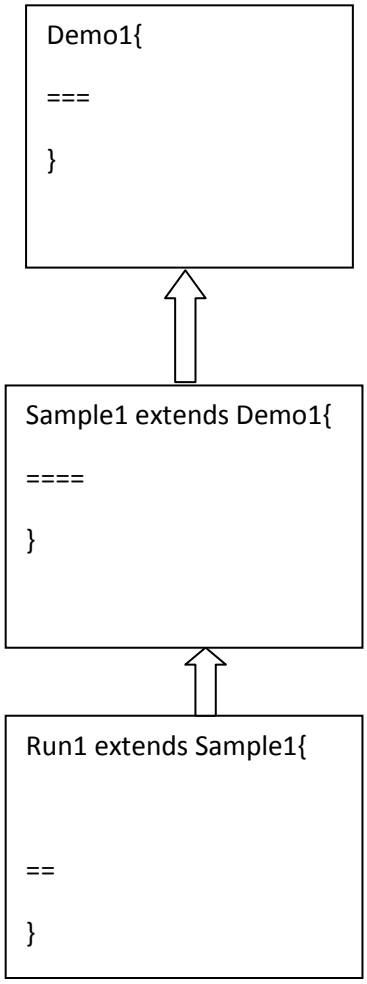
o/p:

d value: 45.67
running disp() of Sample1

K value: 12
	runing test() of Demo1

Example of multi-level Inheritance:

```
class D1{ //super class

	int k=12;
	void test() {
	System.out.println("runing test() of D1 ");
	}
	}

class S1  extends D1{  //sub class

	double d=45.67;
	void disp() {
	System.out.println("running disp() of S1");
	}
}

class R1  extends S1{

	boolean b=true;
	void view() {
	System.out.println("running view() of R1");
	}
	}

class MC2 {
	public static void main(String[] args) {

	R1 ref1=new R1();

	System.out.println("K value: "+ref1.k);
	ref1.test();
	System.out.println("d value: "+ref1.d);
	ref1.disp();
	System.out.println("b value: "+ref1.b);
	ref1.view();
```



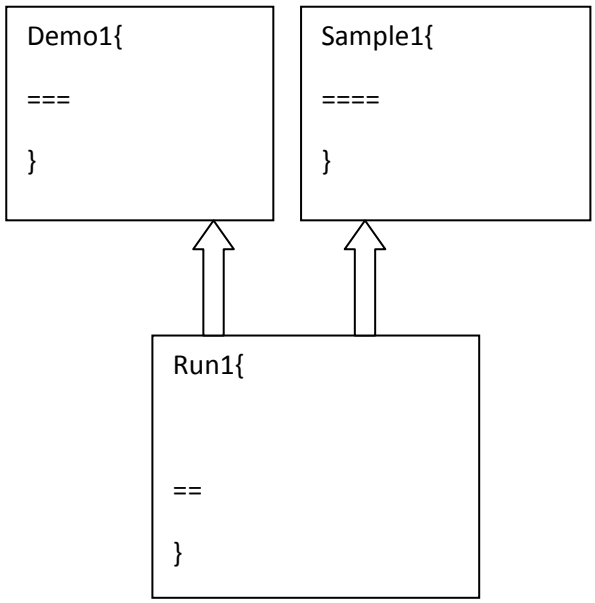
```
}
}
```

o/p:

```
K value: 12
        runing test() of D1
        d value: 45.67
        running disp() of S1
        b value: true
        running view() of R1
```

Exam of multiple Inheritance:

```
Class Run1 extends Demo1,Sample1{   //error

====

}
```



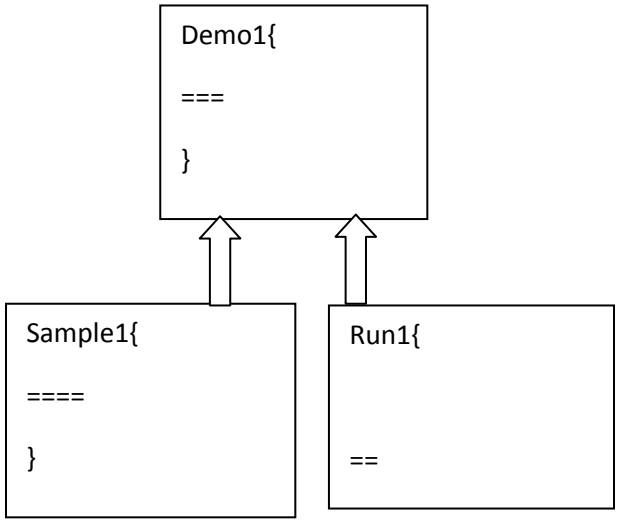
Why java doesn't support multiple Inheritances

Answer is below after super () statements

Exam of Hierarchical Inheritance:

```
class Demo1{  //super class

        int k=12;
        void test() {
        System.out.println("runing test() of Demo1 ");
        }
        }

class Sample1  extends D1{  //sub class

        double d=45.67;
        void disp() {
        System.out.println("running disp() of Sample1");
        }
        }

class Run1  extends Demo1{

        boolean b=true;
        void view() {
        System.out.println("running view() of Run1");
```

```
    }
    }

class MainClass4 {
    public static void main(String[] args) {

        Run1 ref1=new Run1();
        System.out.println("b value: "+ref1.b);
        ref1.view();
        System.out.println("K value: "+ref1.k);
        ref1.test();

        Sample1 ref2=new Sample1();
        System.out.println("d value: "+ref2.d);
        ref2.disp();
        System.out.println("K value: "+ref2.k);
        ref2.test();
    }
    }
```



o/p:

```
b value: true
    running view() of R1
     K value: 12
    runing test() of D1


d value: 45.67
    running disp() of S1
    K value: 12
    runing test() of D1
```

Constructor:

Class Diagram:

Sample1	Class name
Name : type	Data member
Name(); return type	Function member & constructor

Constructor Calling System in Inheritance:

- In an inheritance program the sub class constructor should make a call to the constructor of superclass.
- The call can be made either implicitly or explicitly.
- Sub class can call super class in the using “super () “statements.
- If compiler makes a call to super class is called as implicitly, implicitly call zero argument constructor ,in other words Compiler can call only the 0 arguments of the super class constructor ,if super class constructor designing parameter then compiler not make a implicitly call , we should go for explicitly call.

Use of “super ()” statement and Rules:

- "super ()" statements used to make a call to the super class constructor from sub class constructor.
- "super ()" statements should be used only inside the constructor body.
- "super ()" statements should be the first statements of constructor body; it can't be used anywhere else.
- Multiple super statements is not allowed in the constructor.

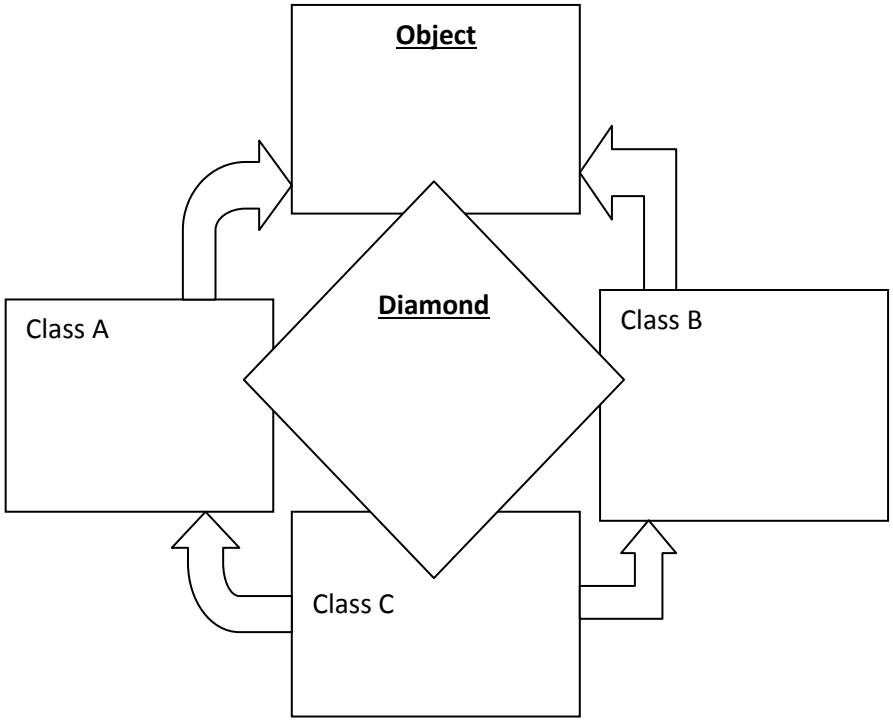
Constructor Chaining:

- Constructor chaining is a phenomenon where subclass constructor makes a call the super class constructor; the super class constructor makes a call to its super class constructor. Constructor chaining can be done either implicitly or explicitly.

Q. Why java doesn't support multiple Inheritances?

- Sub class constructor can't call more than one super class because multiple "super ()" statements are not allowed.
- The multiple inheritances lead to the ambiguity of diamond problem because super class properties can't be inherited to some sub class in two different classes.

super();



Example of Implicitly Constructor Chaining:

```
class D1 {
        D1() {    // zero argument parameterize constructor
        System.out.println("runing const of D1 ");
        }
        }
        class S1  extends D1{
        S1() {
        System.out.println("running const of S1");
        }
}
class MC5 {
        public static void main(String[] args) {
        S1 ref1=new S1();
```

```
}
}
```

o/p: here no need to call super class constructor its automatically call whenever sub class create instance or call

```
runing const of D1
running const of S1
```

Exam of Explicitly constructor chaining:

```
class D1{
        D1(int arg1) {
        System.out.println("runing const of D1 ");
        System.out.println("arg1 value: "+arg1);
        }
        }

class S1  extends D1{
        S1() {
        super(25);//calling to the super class constructor
        System.out.println("running const of S1");
        }
        }

class MC5 {
        public static void main(String[] args) {

        S1 ref1=new S1();
        }
        }
```

o/p:

```
runing const of D1
arg1 value: 25
running const of S1
```

11-Sep-15:

Use of "super ()" statement and Rules:

Constructor of a class can call another constructor of same class using "this()" statements

- "this()" statements can be used call constructor of the current class, it can't call the constructor of super class
- "this()"statements can be used to call method 0 argument or parameterize const.
- "this()"statements should be used only In the constructor body and it must be the first statements of the constructor. Multiple this statements are not allowed
- Recursive constructor calls are not allowed.

Ex,

```
class S1 {
        S1() {//zero argument constructor
        this(10);
        System.out.println("running zero argument constructor ");
        }
```

```
        S1(int arg) {
        this(2.5);
        System.out.println("running int argument constructor ");

        }

        S1(double arg) {
        System.out.println("running double argument constructor ");
        }
        }

class MC6 {
        public static void main(String[] args) {
        S1 obj=new S1();
        System.out.println("Program ended");
        }
        }
```

o/p:

```
running double argument constructor
running int argument constructor
running zero argument constructor
Program ended
```

Example Use of "this" and "super" keyword both and their differences:

```
class D1 {
        int k=34;
        }

class S1 extends D1{

        int k=56;
        void disp(){
        System.out.println("k value of s1: "+this.k);//k value is 56
        System.out.println("k value of s1: "+super.k);//k value is 34
        }
        }

class MC7 {
        public static void main(String[] args) {

        S1 obj=new S1();
        obj.disp();

        System.out.println("Ended Program");
        }
        }
```

o/p:

```
k value of s1: 56
k value of s1: 34
Ended Program
```

1) Java provides the special keywords by “super” which is used to refer super class properties in subclass calling or from subclass.
   Java also provides the special keywords by “this” which is used to identify or refer to that current class global properties and differentiate to local variable as well.
2) The super keywords should be used either in non-static method context or constructorcontext.
   “this” keyword should be used in the method context.
3) It can’t be used in static context.
   But “this” keyword can be used in static context.

Method Overloading:

Defining multiple methods in a class with same name and different arguments is known as method overloading.
This should work either in an argument type or a length of the argument

- In class we can overload both static method and non-static methods
- The overloaded methods are executed based on the method arguments
- Method overloading is used to achieved compile time polymorphism

When we go for Method Overloading?

While develop an application or function if the come cross an operation to be perform with different parameters then we go for method overloading.

Ex 1,

```
class D1 {
        void test() {
        System.out.println("running no arg test()");
        }

        void test(int arg) {
        System.out.println("running int arg test()");
        System.out.println(" arg value: "+arg);
        }

        void test(double arg) {
        System.out.println("running double arg test()");
        System.out.println(" arg value: "+arg);
        }
}

class MC1 {
        public static void main(String[] args) {
        D1 obj= new D1();

        obj.test();
        obj.test(25);
        obj.test(2.5);

        System.out.println("End Program");
        }
        }
```

o/p:

```
running no arg test()
running int arg test()
```

```
 arg value: 25
running double arg test()
 arg value: 2.5
End Program
```

Ex 2,

```
class S1 {
        void disp() {
        System.out.println("Runnig disp ()");
        }
        }

class D1 extends S1{
        void disp(int arg) {
        System.out.println("Runnig disp (int)");
        }
        }

class MC2 {
        public static void main(String[] args) {
        D1 obj1=new D1();

        obj1.disp();
        obj1.disp(235);
        System.out.println("Hello World!");
        }
        }
```

o/p:

```
Runnig disp ()
Runnig disp (int)
Hello World!
```

12-Sep-15:

Method Overriding:

Inheriting in a method on super class changing in the implementation in the sub class is known as method over ridding

1) To override a method inheritance is must.
2) When sub class overrides the method of super class sub class should rewrite the same method signature of the super class, it should change the implementation.
3) The sub class can't override bellow method of super class:-
    1) Static method-because the static methods can't be inheriting to subclass
    2) Final non-static method: because the final key word doesn't allow to change the method implementation.it can be inheriting to the sub class
    3) Private non-static methods: because the private access is restricted only to the class where it is declare
4) Using method overriding we can achieved runtime polymorphism

When we go for Method Overriding?

When developing an apps if we come cross the situation where the functionality should be written with different implementation then we go for method overriding.

Ex 1,

```
class S1 {
		void disp() {
		System.out.println("Runnig disp ()");
		}
		}

class D1 extends S1{
		void disp() {//override method disp() of S1
		System.out.println("Runnig disp() of D1");
		}
		}

class MC2 {
		public static void main(String[] args) {

		D1 obj1=new D1();
		obj1.disp();
		}
		}
```

o/p:

Runnig disp() of D1

********** **********

Ex 2, Very Important for Interview

```
class MN {
		void disp(int x,int y) {
		System.out.println("before swap");
		System.out.println("x value: "+x);
		System.out.println("y value: "+y);

		System.out.println("swapping using temp");
		int temp=0;
		temp=x;
		x=y;
		y=temp;

		System.out.println("x value: "+x);
		System.out.println("y value: "+y);
		}
		}

class NN extends MN{
		void disp(int x,int y) {

		System.out.println("before swap");
		System.out.println("x value: "+x);
		System.out.println("y value: "+y);
```

```
        System.out.println("swapping without using temp");
        x=x+y;//30

        y=x-y;
        x=x-y;

        System.out.println("x value: "+x);
        System.out.println("y value: "+y);
        }
        }

class MC3 {
        public static void main(String[] args){

        NN obj1=new NN();
        obj1.disp();

        int a=10;
        int b=20;
        obj1.disp(a,b);

        System.out.println("End Programm");
        }
        }
```

o/p:

```
before swap
x value: 10
y value: 20
swapping without using temp
x value: 20
y value: 10
End Programm
```

12-Sep-15:

Type casting:

Casting one type of function to another type is known as type casting.

There are two type of casting:-

1) **Primitive Data type casting** (Simple): Datatype----casted--->datatype .
2) **Class type casting** (V.V.Imp): classtype--- casted -- >classtype.

1) **Primitive Data type casting:**
   In the Primitive data type casting, a data type is casted to another data type. There are two type of Primitive data type casting:-
   a) **Widening:** Casting lower data type to any of the higher data type is known as Widening.
      The widening can be done implicitly by compiler hence it is also known as implicit widening or auto-widening.
   b) **Narrowing:**Casting higher data type to any of the lower data type is known as Narrowing.
      The Narrowing should be explicitly performed in the code. Compiler can't perform narrowing because whenever narrowing is performed data loss happens.

Syntax, datatype variable= (data type) value;

int x= (int)10.2;→double is casted to int.//Narrowing, must explicit
double y= (double)10;→int is casted to double. //Widening
=10; //implicit

Prog1, **********

```
class MC1 {
        public static void main(String[] args) {
        int x1=(int)3.2; //double is casted to int.//Narrowing, must explicit
        double y1=(int)3;//int is casted to double. Widening
        System.out.println("x value: "+x1);
        System.out.println("y value: "+y1);

            int a=23;
            double b=59.99;

            int x2=(int)b;
            double y2=(double)a;

            System.out.println("x2 value: "+x2);
            System.out.println("y2 value: "+y2);
        }                                                                    }
```

o/p:

```
x value: 3
        y value: 3.0
            x2 value: 59
                    y2 value: 23.0
```

****

int x1=(int)3.2;//explicit narrowing
double y1=3;//implicit widding

int x2=(int)b;//explicit narrowing
double y2=a;//implicit widding

****

Pro2,

```
class NM{
        int sqr(int num) {
        System.out.println("cal sqr of : "+num);
        int res=num*num;
        return res;
        }                                                                    }

class MC2 {
        public static void main(String[] args) {
        double x1=6.34;
        NM no=new NM();
        int x2=no.sqr((int)x1); //cast x1 value to int and then pass to method argument
```

```
        System.out.println("Result is: "+x2);
        }                                                                                    }
```

o/p:

Result is: 36

****

** While passing value to the method argument or constructor argument we can perform type casting .when returning a value of a method type casting can be perform.*

```
class MC3 {
        public static void main(String[] args) {
        char x='A';
        int i=x;//implicit

        System.out.println("x value: "+x);
        System.out.println("i value: "+i);

            int j=78;
            char y=(char)j;//explicit

            System.out.println("j value: "+j);
            System.out.println("y value: "+y);
        }                                                                                    }
```

o/p:

```
x value: A
i value: 65
j value: 78
y value: N
```

********** **********

***without using java inbuilt keyword , convert "jspider" to "JSPIDER":(MOST OF INTERVIEW QUESTION)**

```
class MC4 {
        static String upperCase(String arg){
        char[] arr=arg.toCharArray();//converting string to char array
        char[] temp=new char[arr.length];

        for (int i=0;i<arr.length ;i++ ){
        int x=arr[i];//getting asccii valur of char
        temp[i]=(char)(x-32);//convert to upper case
        }
        String uprStr=new String(temp);//converting char aray to string
        return uprStr;
        }

        public static void main(String[] args) {
        String str="jspider";
        System.out.println("Giving str: "+str);
        String s1=upperCase(str);
        System.out.println("Giving str in upper Case: "+s1);

        }
```

}

o/p:

```
Giving str: jspider
Giving str in upper Case: JSPIDER
```

14-Sep-15:

---

2) <u>Class type casting:</u>

Casting one class type to another class type is known as Class type Casting.

```
Prog1,                                                                                    Class
S1{
        int k=23;
        char x='j';
        void test() {
        System.out.println("running test() of S1");
        }                                                                                 }

                                                                                    class D1
extends S1{                                                                  boolean b=true;
                                                                    float f=21.34f;
                                                            void disp() {
                        System.out.println("running disp() of D1");

        }

}

class MC5 {
        public static void main(String[] args){

            S1 ref1; //ref variable of type of S1
            ref1=(S1) new D1(); //up casting,cast D1 to S1

            D1 ref2;//rerf variable of D1 type
            ref2=(D1) new S1(); //down casting, this line goes error like Exception in thread
                    }

}

o/p                                                                                       Exception in
thread -------- S1 can't casted to D1…..
```

To perform to class type casting we have to fulfill provide rules:-

a) The classes should have is a relationship(inheritance)
b) The object should have the properties of the class to which we have to casted
c) Class type casting can be done in two ways
    1) Up casting
    2) Down casting

- Casting subclass type to super class type is known as up casting. Up casting can be done either implicit or explicit. The implicit casting done by compiler
- Casting super class type to sub class type is known as down casting. The down casting should be done explicit in the code.
- Down casting should be perform only on the object which is already up casting

d) During compilation the compiler check the class type casting statements, if classes having is a relationship then compiler compiles statementsotherwise compiler goes an error in compatible type.

*classcastEx is a type of RuntimeExp which through by jvm and runtime.

Why this Exception is occurred?

Whenever an object of a class type is casted to another class type which is not having the properties in the object then jvm throughs classcastExp

Why compiler doesn't detect at compile time?

Because os the compiler only check the is a relationship not the properties

How to avoid ClassCastExp?

The ClassCastExp can be avoided by using "instanceof" operator.

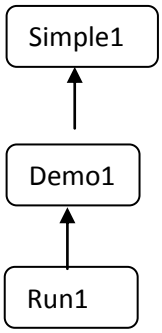
*We can access only the member of which class is casting another class.

Ex, of above class.

```
System.out.println("k value: "+ref1.k);

        ref1.test();
```

- When an instance of a class is casted to another class then we can access only those properties the class type which is casted.
- If a ref variable type of super class, to that ref variable we can assign any of its sub class object because of implicit upcasting.
  In other word, a super class ref variable can point any of the sub class objects

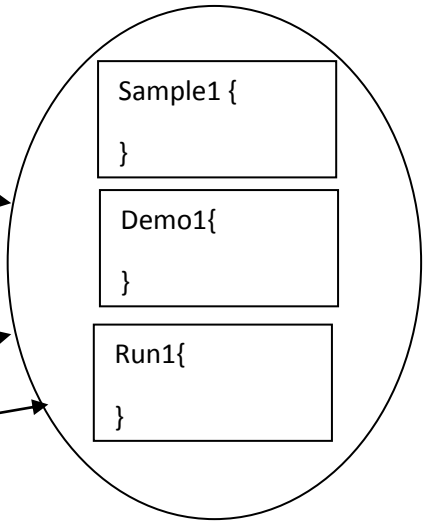
15-Sep-15:




```
Sample1 ref1;
        ref1=new Sample1 ();
        ref1=new Demp1 (); //implicit
        ref1=new Run1 (); //implicit

Run1 r1;
        r1=new run1;
        r1= new (run1) D1 // not possible because D1 have not property of R1
        r1= new (run1) S1 // not possible because S1 have not property of R1
```

Ex,

```
class Sample1 {
        int k=23;
        char x='j';
        void test() {
        System.out.println("running test() of Sample1");
        }
        }

class Demo1 extends Sample1{
        boolean b=true;
        float f=21.34f;
        void disp() {
        System.out.println("running disp() of Demo1");
        }
        }

class Run1 extends Demo1{
        boolean c=true;
        double d=21.34f;
        void view() {
        System.out.println("running view() of Run1");
        }
        }

class MC6 {
        public static void main(String[] args) {


        Run1 r1; //ref variable of type of Run1
        r1=new Run1();//using r1 ref we can access run1,demo1,simple1
        r1.test();
        r1.disp();
        r1.view();

        Demo1 d1;
        d1=r1;//using d1 ref we can access ,demo1,simple1
        d1.test();
        d1.disp();

        Sample1 s1;
        s1=r1;////using s1 ref we can access simple1
        s1.test()
        }
        }
```



- If a method argument is class type, then during method invocation we can pass any of the sub class instances, if you pass super class instance then we get ClassCastException.

Ex,

```
class Sample1 {
		int k=23;
		char x='j';
		void test() {
		System.out.println("running test() of Sample1");
		}
		}

class Demo1 extends Sample1{
		boolean b=true;
		float f=21.34f;
		void disp() {
		System.out.println("running disp() of Demo1");
		}
		}

class Run1 extends Demo1{
		boolean c=true;
		double d=21.34f;
		void view() {
		System.out.println("running view() of Run1");
		}
}

class Start {
		void execut(Sample1 arg){
		System.out.println("running execut() of Stratt");
		System.out.println("k value: "+arg.k);
		System.out.println("k value: "+arg.x);
		arg.test();

		}
}

class MC7 {
		public static void main(String[] args){
		Start srt=new Start();
		srt.execut(new Sample1());
		System.out.println("----------------------");
		srt.execut(new Demo1());
		System.out.println("----------------------");
		srt.execut(new Run1());
		}
		}
```

18-Sep-15:

Down Casting:

Ex,

```
class Sample1 {
        int k=23;
        char x='j';
        void test() {
    System.out.println("running test() of Sample1");
}
}

class Demo1 extends Sample1{
boolean b=true;
float f=21.34f;
void disp() {
System.out.println("running disp() of Demo1");
}
}

class Run1 extends Demo1{
boolean c=true;
double d=21.34f;
void view() {
System.out.println("running view() of Run1");
}
}



class Start {
void execut(Sample1 arg) {
System.out.println("running execut() of Start");
System.out.println("k value: "+arg.k);
System.out.println("x value: "+arg.x);
arg.test();

//If you want to use the Demo1 Properties then you should first up casting then go to down casting
Demo1 ref1= (Demo1)arg;//explicit down casting
System.out.println("b value: "+ref1.b);
System.out.println("f value: "+ref1.f);
ref1.disp();
}
}

class MC8{
public static void main(String[] args) {
Start srt=new Start();
srt.execut(new Demo1());   //up casting
    }
}
```

o/p:

```
running execut method
        k value: 23
            x value: j
                running test() method
```

```
b value true
f value 21.34
        running disp() method
```

Instance of operator Demo:

Ex,

```
class Sample1 {
        int k=23;
        char x='j';
        void test () {
        System.out.println("running test() of Sample1");
        }
        }

class Demo1 extends Sample1{
        boolean b=true;
        float f=21.34f;
        void disp() {
        System.out.println("running disp() of Demo1");
        }
        }

class Run1 extends Demo1{
        boolean c=true;
        double d=21.34f;
        void view() {
        System.out.println("running view() of Run1");
        }
        }



class Start {
        void execut(Sample1 arg) {
        System.out.println("running execut() of Stratt");
        System.out.println("k value: "+arg.k);
        System.out.println("k value: "+arg.x);
        arg.test();

        if(arg instanceof Demo1){
        Demo1 ref1=(Demo1)arg;//explicit Down casting

        System.out.println("running execut() of Stratt");
        System.out.println("b value: "+ref1.b);
        System.out.println("k value: "+ref1.f);
        ref1.disp();
        }else{
        System.out.println("instance doesn't have properties of Demo1");
            }
        }
        }

class MC8{
        public static void main(String[] args) {
```

```
        Start srt=new Start();
        srt.execut(new Sample1()); //not done up casting
                }
            }
```

o/p:

```
running execut() of Stratt
k value: 23
k value: j
running test() of Sample1
instance doesn't have properties of Demo1
```

Polymorphism

An object showing different behavior and different status of a its life cycle is known as polymorphism,

There are two type of polymorphism:

1) Compile time polymorphism
2) Runtime polymorphism

1. Compile Time Polymorphism:

In compile time polymorphism the method declaration is binded to the method definition at compile time by compiler.

Since the binding is happening at operation time it is known as compile time binding or static binding or early binding, for overloaded methods ,static methods and final methods compile time binding happens. Hence this methods are example for compile time polymorphism.

2. Runtime Polymorphism:

In run time polymorphism the method declaration is binded to the method definition by the JVM during execution. Since binding is happening at runtime it is known as runtime binding or dynamic binding or late binding. The method overriding is an example for runtime polymorphism since binding happens at execution time.to achieved runtime polymorphism we have to fulfill the following concept.

a) Inheritance
b) Method overriding
c) Upcasting

Whenever we create an instance of a subclass which is having overridemethods, if we refer that instance either by subclass ref or by super class methods we get overrideimplementation only.

Ex,

```
class Sample1 {

        int k=23;
        char x='j';
        void test() {
        System.out.println("running test() of Sample1");
        }
        }

class Demo1 extends Sample1{
```

```
        boolean b=true;
        float f=21.34f;
        void test(){
        System.out.println("running disp() of Demo1");
        }
        }

class MC1 {
        public static void main(String[] args) {

        Sample1 srt=new Demo1();//implicit up casting
        srt.test
        }
        }
```

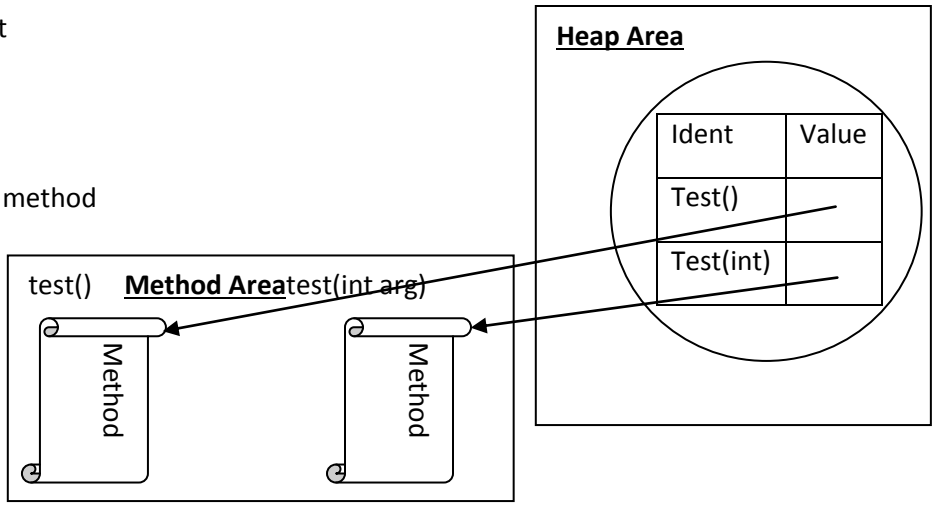
o/p:

```
running disp() of Demo1
```

19-Sep-15:

Example of compile time Polymorphism:

```
class A {
        void test ()//declaration object
        {
        ===          //method body
        }

        void test(int arg)//overloaded method
        {
        ===
        }
        }
```



We can call it like

```
A a1=new A();
a1.test();
a1.test(10);
```

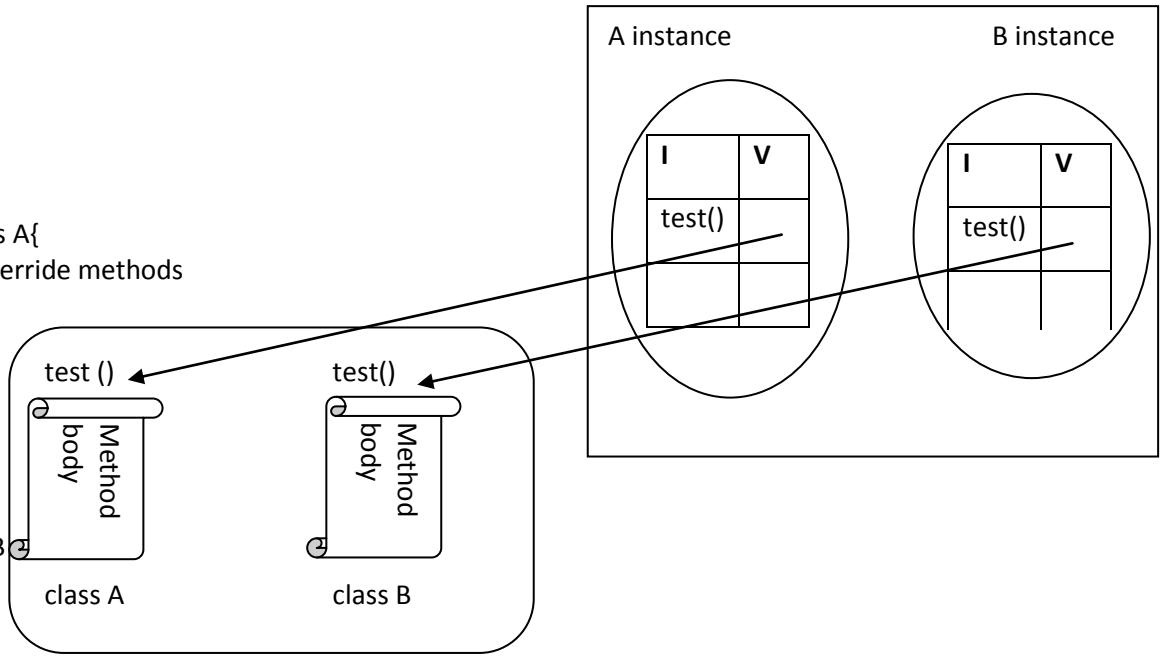
Methods are binding based on the argument.

Example of Run time Polymorphism:

```
Class A{
        Void test()
        {
        ====
        }
        }
        Class B extends A{
        Void test()//override methods
        {
        ====
        }
        }
```

Calling like

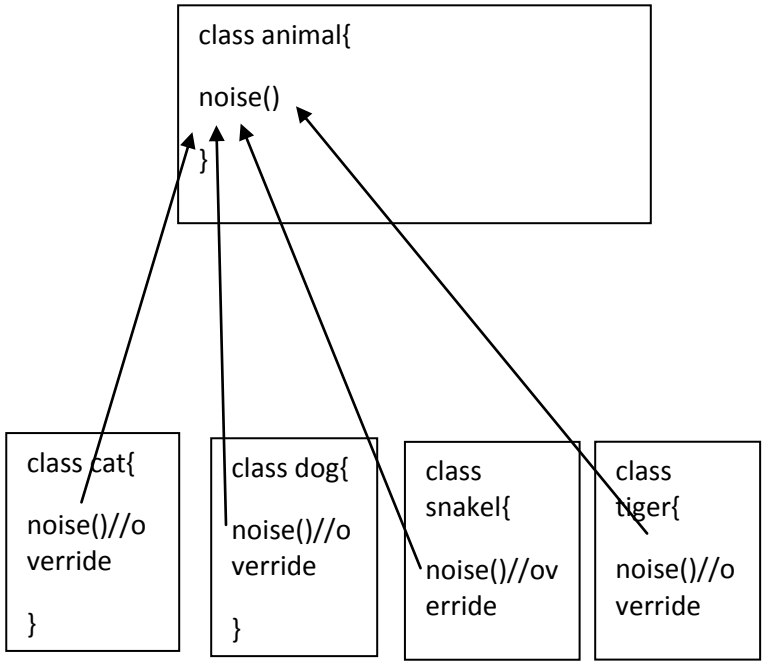
```
A a1=new A ();
    B b1=new B ();
```



Methods are binding based on the instance.

Ex,

```
class Animal {
        void noise(){
        System.out.println("all animal makes noise");
        }
}
class Cat extends Animal {
        void noise(){//override noise() of Animal class
        System.out.println("mew mew");
        }
}
class Dog extends Animal {
        void noise(){//override noise() of Animal class
        System.out.println("bow bow");
        }
}
class Snake extends Animal {
        void noise(){//override noise() of Animal class
        System.out.println("Hiss Hiss");
        }
        }
```



```
class Similar{
        void makenoise(Animal arg){
        arg.noise();
        }
        }

class TestAnimal{
        public static void main(String[] args){

        Cat c1=new Cat();
        Dog d1=new Dog();
        Snake s1=new Snake();

        Similar sim=new Similar();
        sim.makenoise(s1);

        }

}
```

o/p:

Hiss Hiss
//sim.makenoise(s1); at this line whatever you pass like s1,d1,c1 the makenoise(Animal arg) call that object method.
E.g. if you pass s1 then output will be “Hiss Hiss”, if you pass d1 then output will be “vow vow”.

Tomorrow abstraction