Unit 2 (Java Programming)

Date : / /
Page No.

✓ The wrapper class in Java provide the mechanism to convert primitive into object and objects into primitives ~~automatically~~.

[The automatic conversion]

✓ Since J2SE 5.0, autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically. Automatic conversion of primitives into objects known as autoboxing and vice-versa unboxing.

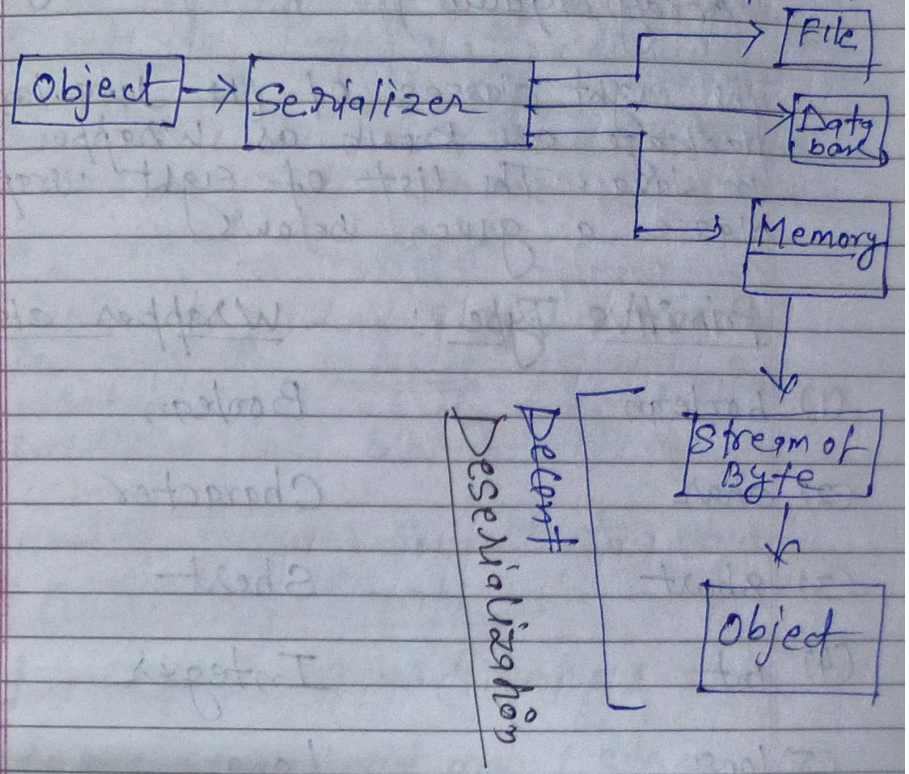
✓ Java is an object oriented programming language, so we need to deal with objects many time like in collections Serialization, synchronisation, etc

Let see different scenarios, where we need to use the wrapper class.

(1) Changed the value in Method: Java supports only call by value. so we pass a primitive value, it will not chang the orginal value. But if we convert the primitive value in an object, it will change the original value

Date : / /
Page No.

(2) Serialization: We need to convert the object into streams to perform the serialization.
Serialization is the process of converting the state of an object into a form that can be transported.



(3) Synchronisation: Java Synchronisation works with objects into Multithread

(4) Java.util package: The Java Util package provides the utility classes to deal with objects.

Date : / /
Page No.

(5) Collection Frame work : Java collection Frame works with object only. All classes of the collection frame works (ArrayList, Linked List, Vector, Hash set Linked Hash Set, Tree set, Priority Queue, Array Dqueue)

The eight classes of the Java.lang package are known as Wrapper classes in Java. The list of eight wrapper classes a given below

Primitive Type :	Wrapper class
(1) boolean	Boolean
(2) char	Character
(3) short	Short
(4) int	Integer
(5) long	Long
(6) float	Float
(7) double	Double
(8) byte	Byte

Date : / /
Page No.

Autoboxing : The automatic conversions of primitive data type into its' corresponding class is known as Autoboxing. For Example:

(1) byte to Byte (2) char to Character
(3) int to Integer (4) long to Long
(5) float to Float (6) boolean to Boolean. (7) double to Double
(8) short to Short

Since Java 5, we do not need to use the valueOf method of wrapper classes to Convert the primitive into objects

```
// Java program Convert to objects
// AutoBoxing Example.
class WrapperExample1
{
public static void main(String args[])
{
int a=20; // Converting int into Integer
Integer i = Integer.valueOf(a)
Integer j = a; // autoboxing
// autoboxing, now Compiler will write Integer.valueOf( )
```

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

output 20 20 20

Unboxing: The automatic Conversion of wrapper type into its Corresponding primitive type is known as unboxing It is reverse process of auto boxing. Since Java 5, we do not need to use the intValue() method of wrapper classes to Convert the wrapper types into primitive

```
Class Wrapper2
{
public static void main(String args[])
{
// Converting Integer into int
int b=3;
Integer a= new Integer(b);
int i = a.intValue();
                // Converting Integer to
                   int explicitly
int j = a;  // unboxing, now
              Compiler will
              write a.intValue()
              Internally
```



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

output 3 3 3

```
class Wrapper2
{
public static void main(String args[])
{
 byte b=10;
 short s=20;
 int i=30;
 long l=40;
 float f=40.0F;
 double d=60.0D;
 char c='a';
 boolean b=true;
// Autoboxing
 Byte b1=b;
 Short s1=s;
 Integer i1=i;
 Float f1=f;
 Double d1=d;
 Character c1=c;
 Boolean b1=b;
```

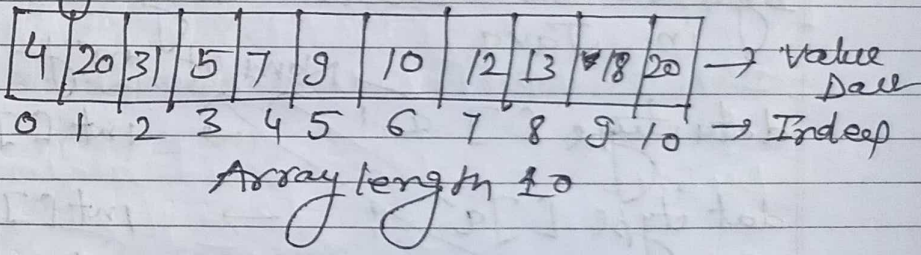
```
// Printing object
Sop(b1);
Sop(s1);
Sop(i1);
Sop(f1);
Sop(d1);
Sop(c1);
Sop(b1);
// Unboxing
byte b2 = b1;
short s2 = s1;
int i2 = i1;
float f2 = f1;
double d2 = d1;
char c2 = c1;
boolean b2 = b1;
Sop(b2); Sop(s2); Sop(i2);
Sop(f2); Sop(d2); Sop(c2);
Sop(b2); } }
```


[Java Arrays] ① Normally, an array is a collection of similar type of elements which has continuous memory location.

② Java array is an object which contains elements of similar data type.

③ Array in Java is index based, the first element stored in 0th index.

④ In Java, array is an object of a dynamically generated class. Java array inherit the object class



Advantage: ① Code optimisation
② Random access.

(1) Code Optimization :→ It make code optimised, we can retrieve or sort the data efficiently.

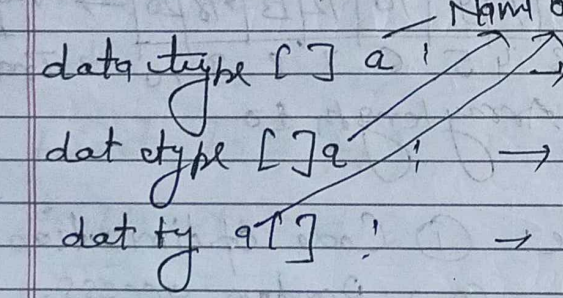
(2) Random access: We can get any data located at an index position

Disadvantage: Size Limit: We can store only the fixed size of elements in the array. It does not grow its size at runtime. To solve this problem, collection framework is used in Java

[Type of Array in Java]

(1) Single Dimensional Array
(2) Double Dimensional Array
(3) Multi Dimensional Array

Syntax to Declare an Array in Java



Instantiation of an array in Java

int a[] = new int[5]

// declaration and Instantiation

(1) // Java program to illustrate how to declare (1), instantiate (2), initialized (3)

Traversing array (4)

```
class Testarray
{
public static void main (String args[])
{
int a[] = new int[5]; // declare & Instantiate
a[0] = 10;
a[1] = 20;
a[2] = 70;
a[3] = 40;
a[4] = 50;
}                                   // initialization
// traversing
for(int i=0; i< a.length; i++)
{
  sop(a[i]);
}}
```

Output

```
10
20
70
40
50
```

```
class Array2
 {
public static void main(String args[])
 {
int a[] = { 20, 30, 40 };
for(int i=0 ; i<a.length ; i++)
   {
   sop(a[i]);
   }}
```

[Multi Dimensional Java Array]

```
class MultiDimensional
 {
public static void main
                (String args[])
{
int a[][] = { {1,2,3}, {2,4,5}
              {4,4,5} };
// Printing 2D array
for(int i=0 ; i<3 ; i++)
    {
for (int j=0 ;j<3 ; j++ {
```


Print

```
sop( a[i][j] + " ");
   }
sop( );
   }}}
```

println

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

output

[Jagged Array in Java]

	J=0	J=1	J=2	J=3
i=0	0	1	2	
i=1	3	4	5	6
i=2	7	8		

```
class TestJagged
 {
int a[][]= new int a[3][];
   a[0]= new int[3]
   a[1] = new int[4]
   a[2]= new int[2];
 int Count=0
 for(int i=0 ; i<a.length ; i++)
   {
```

```
for (int J=0 ; J < a[i].length ; J++)
 {
a[i][J] = count++;
   }
   }
// printing
for(int i=0 ; i < a.length ; i++){
for(int J=0 ; J < a[i].length ; J++)
  {
  System.out.print(a[i][J] + " ");
   }
  Sop(   ); //new line
  }
  }
  }
```


[Program 3 diamensional Array']

```
Class 3D_Array
{
public static void main(String args
 {
int i, j k ;
int a[][][] = new int[3][4][5]
for(int i=0 ; i<3 ; i++)
  {
 for(int J=0 ; J<4)
 for(int K=0 ; K<5)
  a[i][J][K] = i*j*k;
// for print
 for(int i=0 ; i<3 ; i++)
    {
  for (int J=0 ; J<4; J++)
    {
 for (K=0; k<5; k++)
   {
  Sop( a[i][J][k] + " ")
   }
  Sop( ); } }}.
```

page
row
column

output

	C_1	C_2	C_3	C_4	C_5
r_1	0	0	0	0	0
r_2	0	0	0	0	0
r_3	0	0	0	0	0
r_4	0	0	0	0	0

0	0	0	0	0
0	1	2	3	4
0	2	4	6	8
0	3	6	9	12

0	0	0	0	0
0	2	4	6	8
0	4	8	12	16
0	6	12	18	24

[Operators in Java] : operator in Java is symbol that is used to perform operations.
Examples: +, -, *, /, % etc

There are many types of operators in Java which are given below

(1) Unary operator (++A, A++, --A, A--)
(2) Arithmetic operator
(3) Shift operator
(4) Relational operator
(5) Bitwise operator
(6) Logical operator
(7) Ternary operator
(8) Assignment operator.

(1) Unary operator : There is only one operands

	operator	Use	Description
(1)	Post Increement	A++	A = 1 B = A++ then B will hold = 1 and A will hold = 2
(2)	Post decrement	A--	A = 1 B = A-- B will hold A then A will hold Zero

operator	Use	Description
(3) Pre-increment	++A	A = 1 B = ++A B will hold 2 and A will hold = 2
(4) Pre-decrement	--A	A = 1 B = --A B will hold 0 A will hold 0

[Program Unary operator]

```
class Unaryoperator1
{
public static void main (String args)
  {
  int x = 5;
  System.out.println(x++);   5, 6
  System.out.println(++x);   7
  System.out.println(x--);   7, 6
  System.out.println(--x);   5
  }}
```


Output
5
7
7
5

```
class operatorUnary2
{
public static void main (String args)
  {
  int a = 10;
  int b = 20;
  System.out.println(a++ + ++a);
  System.out.println(b++ + b++);
  }}
```

output 10 + 12 = 22
10 + 11 = 21

```
class operatorUnary3
{
public static void main (String args
  {
  int a = 10;
  int b = -10;
  boolean c = true;
  boolean d = false;
  System.out.println(!a);
  System.out.println(!b);
```

```
System.out.println(!c); // false
System.out.println(!d); // true
}}
```

output → -11
9
false
true

[Bitwise operator]

(1) Bitwise AND → &
(2) Bitwise OR → |
(3) Bitwise XOR (Exclusive OR) ^
(4) Bitwise Complement ~ (tilde)
(5) Bitwise (Shift operator)
(1) Right Shift >>
(2) Left Shift <<

	operator	symbol	use
(1)	Bitwise AND	&	A & B
(2)	Bitwise OR	|	A | B
(3)	Bitwise XOR	^	A ^ B
(4)	Bitwise Complement	~	~A

```
class ANDOperator
{
public static void main (String args)
{
int x = 9, int y = 8;
System.out.println("x & y" +
(x & y));
}
}
```

output 8

x & y

8 =	1	0	0	0
9 =	1	0	0	1
	1	0	0	0

↓
(8)

Bitwise XOR

x=9	1	0	0	1
x=8	1	0	0	0
XOR	0	0	0	1

Same Digit return = 0
Different Digit return 1

sop(x^y);

output = 1

sop(x|y);

output = 9

→ Shift operator → (1) Right → Bit Shifted Right side
(2) Left → Bit Shifted Left side.

Example

Right Shift

a = 10 = 1 0 1 0 (10)

a>>1 = (5) 1 0 1 (5)

1 0 1 0 0

$2^4 + 2^2 = 16 + 4 = 20$ ← (20)


```
class operatorRightShifted
{
public static void main(String args
{
int a = 10; // 10÷2¹ = 5
int b = 20; //
int c = 30; //
System.out.println(a>>1); 5
System.out.println(9>>2); 2
System.out.println(b>>1); 10
System.out.println(b>>2); 5
System.out.println(c>>1); 15
System.out.println(c>>2); 7
}
}
class operatorLeftshift
{
public static void main(String args
{
int a = 5;
sop(a<<1); // 5×2¹ = 10
sop(a<<2); // 5×2² = 20
sop(a<<3); // 5×2³ = 40
}
```

Logical operator → (1) Logical and &&
(2) Logical OR ||

Program (Logical && and Bitwise &)

&& → check only First Condition
& → check both condition

```
class operatorAnd
{
public static void main(String args[])
{
int a=10;
int b=5;
int c=20;
Sop( a>b && b<c); True
          (T)  (T)
Sop( a<b && b<c); False
       F    T
Sop( a<b & b<c ); False;
}
}
```


check only one condition
check both condition

Java Logical OR (||) vs Bitwise |

[Control statement in Java]
Control flow in Java

Java Compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provide statements that can be used to control the flow of Java code.

(1) Decision Making Statement
Branching / Selection

(a) [If statements
Else

(b) [Switch statement

(2) Iterations / Looping

(a) for loop
(b) while loop
(c) do while loop
(d) for each loop

(3) Jump Statements

(a) break statement
(b) Continue statement
(c) Label : return

(1) Decision Making / Selection

(a) Simple if
(b) if - else
(c) if else - if Ladder
(d) Nested if - statement

(a) if (test expression)

(1) if (bank bitance is zero)
{
System.out.println ("Borrow");
}

(2) if (age is more than 55)
{
System.out.println ("Person is Retired");
}

(3)
```
class Student
{
int x = 10;
int y = 12;
if(x+y > 20)
{
SOP(" x+y is greater than 20");
}}}
```

[Program if statement]

```
class IfTest
{
public static void main(String
    args[])
{
int i, Count, Count1, Count2;
double weight[] = {45.0, 70.50, 30.70};
double height[] = {176.50, 170.0, 150.04};
Count=0;
Count1=0;
Count2=0;
for(i=0; i<3; i++)
  {
  if(weight[i]<50.0 && height[i]>170.0)
    {
    Count1 = Count1+1;
    }
    Count = Count+1    // Total
                          Person
  }
  Count2 = Count - Count1
  sop("Total Person =" + Count);
  sop("Full Fill Condition =" + Count1);
```



```
sop("other person" + Count2);
  }
}
```

(b) If-else element

```
if (test expression)        True
    Condition               False
  {
    sop(          )<
  }
else {
      sop(          );
      }
```

Explaination:

```
if (code == 1)
{
  boy = boy+1;
}
if (code == 2)
{   girl = girl+1;
sop(
}
  =
  =
```

```
if (code == 1)
 {
 boy = boy+1 )
 }
else
   {
 girl = girl+1;
   }
```

✓ class If-else-test
 {
public static void main(String args[])
   {
int number[] = { 50, 70, 80, 90 91 73 };
 int even=0, odd = 0
for(int i=0 ; i< number.length ; i++)
   {
 if((number[i]% 2)==0)
   {
 even = even+1 ;
   }
else
odd = odd+1;
}} Sop(even);
 Sop(odd); }}

Nested (If-Else)

```
if (  )
{
  if( )
    { statement1
    }
  else( )
    { statement2
    }
}
else {
     statement3
     }
```

Class Nested If_else
 {
public static void main (String
 args[])
 {
int a= 325;
int b= 425;
int c= 405;
if(a>b)
 {
 if(a>c)
 {sop("Largest number a="+a)

```
if ( a>b && a>c)
  {
  sop(a)
  }
else {
   if ( b>a && b>c)
     {
     sop(b)
     }
   else
      sop(c) }}}}
```

The Else If Ladder
Multipath decisions

```
if ( Condition1)
  {
  statement 1
  else if ( Condition2)
    statement 2
  else if ( Condition3)
  else default statement
  statement
```



```
else
  {
sop("Largest number is =" + c);
  }
else {
   if (c>b)
     {
     sop("Largest" + c)
     }
   else
     {
     sop(b);
     }
   }
  }
 }
```

(IInd) Program

```
class Nested AND
  {
public static void main(String args[])
    {
   int a, b c;
   a= 5;
   b= 10;
```

[Else If Ladder]

```
if(marks > 79)
    grade = "Honors";
else if(marks > 59)
    grade = "First Division";
else if(marks > 49)
    grade = "Second";
else grade = "fail");
```

[Program]

```
class ElseLadder
{
public static void main(String args[])
{
int rollNumber[] = {11, 12, 13, 14, 15};
int marks[] = {50, 60, 49, 75, 80};
for(int i=0; i< rollNumber.length; i++)
{
if(marks[i] > 79
{
Sop(rollnumber[i] + " ~~I'S Division~~ Honor");
```



```
else if(marks[i] > 59)+ " I st Division");
Sop(rollNumber[i] + " I st Division");
else if(marks[i] > 49)
Sop(rollNumber[i] + " II nd Division");
else if(marks[i] > 33)
Sop(rollNumber[i] + " III rd Division");
else
Sop(rollNumber[i] + "fail");
}}}
```

[Java Switch statement]

The Java Switch Statement execute one statement from multiple Condition. It is like if-else-if ladder statement. The Switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Integer, Long. Since Java 7, you can use String in Switch Statement

Syntax

```
switch(expression)
  {
case value1:
  Code
  break;
case value2:
  code
  break;
- - -
-
default:
code
}
```

```
class Day_Test
{
public static void
 main(String args[])
 {
 int day = 5;
 switch(day)
  {
case 1:
  Sop("MON");
   break;
case 2:
  Sop("TUE");
   break;
case 3:
  Sop("WED");
   break;
case 4:
  Sop("Thu");
   break;
case 5;
```



```
Sop("Friday");
   break;
case 6:
 Sop("sat");
  break;
default:
Sop("Not valid");
 }
 }}
```

output Friday

[Iteration] / looping

① **For loop**: Java for loop is used to iterate a part of Program Several Times if number of Iteration is fixed, It is recommended to use for loop.

② **while loop**: Java while loop is used to iterate a part of Code Several times. If number of Iteration is not fixed

③ **do while**: Same while loop But minimum or at least loop occur once (if Condition is false)

✓ (while loop)

(1) Initialization
(2) Condition
(3) Updation
(4) Statement

```
Initialization ①
while (Test Condition) ②
{
  body ③
  updation ④
}
```



```
class while loop
{
public static void main (String args[])
{
int i=0;
int Sum=0;
while (i<5)
{
Sum = Sum+i;
System.out.println(Sum);
i++
}
sop(Sum);
}
}
```

Output
0 1 2 5 9
9

✓ (do while)

```
initialization
do
{
  body updation
}
while (Condition);
```

```
class Table
{ public state void main( shing args)
int i=1;
do
  {
 Sop( 2*i)
   i++
  }
while (i<=10)
  }
  }
```

2
4
6
8
10
12
14
18
18
20

```
class Table
  {
public static void main( shng args
int i=1;
while(i<11)
  {
Sop( 2*i);
    i++;
  }
  }
```

[Table 2]

[Simple for loop]


```
class Table_for
{
public static void main( String
                         args[] )
{
for(int i=1 ; i<=10 ; i++)
  {
Sop ( 2*i);
  } } }
```

[Java For loop]

Simple Java for loop :

1. Initialization
2. Condition
3. [Increment / Decrement] updation
4. statement

[Java Nested for loop]

```
class Nested for loop
{
public static void main( String args)
  {
for( int i=1 ; i<=3 ;i++)
    {
```

```
for(int J=1; J<=3; J++)
  {
System.out.println(i+" "+J);
  } end of
 }
}
}
```

output

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

Example [Pyramid]

```
class Pyramid
{
public static void main(String args[])
 {
for(int i=1; i<=5; i++)
  {
for(int J=1; J<=i; J++)
  {
 sop("*");
  }
  }
```


① ②

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

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

```
int J = 5
J = < i J--
```

* [Java for-each Loop]

✓ The for-each loop is used to traverse array or collection in Java

✓ It is simple use of simple for loop there is no need incremental

Syntax: for(data type variable : array name)
{
code
}

Class ForEachLoop [J2SE 5.0]

```
{
public static void main(String args[])
 {
int a[] = {1, 2, 3, 4, 5, 6};
 for(int i : a)
  {
  sop(i)
  }}}
```

out put
1, 2, 3, 4, 5, 6

for each loop

```
class CAR
{
public static void main(String args[])
  {
String [] Cars = {"Volvo", "Bmw",
                  "Ford", "I10", };
for(String i : Cars)
  {
  Sop(i);
  }}}
```

Simple for loop

```
for(int i=0; i<= Cars.length; i++)
  {
Sop( Cars[i] );
  }
```


(Jump Statement in Java)
Break / Continue

(1) Break : statement can also be used to Jump out of loop.

(2) Continue : The Continue statement break one iteration (in the loop) if specified condition occured. and Continue;

```
class Break_Test
{
public static void main(String args[])
{
for(int i=0 ; i<10 ; i++)
  {
  if (i==4) {
    break;
    }
  Sop(i) } } }
```

output 0 1, 2, 3

we use continue place of break

output 0 1, 2, 3 5 6 7 8 9

```
class TestContinue
{
public static void main (String args[])
{
for(int i=1 ; i<10 ; i++)
{
if(i==4)
continue;
SOP(i);
}
}
}
```

Output 1, 2, 3 5 6 7 8 9 10

[Classes, Objects and Methods]

Class :→ ① Class is group of similar type objects

② It is is template or blueprint from which object are created

③ It is logically entity

Class Contain

① Fields ② Methods ③ Constructor ④ Blocks ⑤ Nested classes and Interfaces

```
Class <class Name>
{
<type> <variable>;
<type> <variable>;
<type> <method>(<parameter list>)
// body of method
}
```

Instance Variable

Example:

```
class circle {
double r;
void getData ( double x )
{
r=x;
}
```

```
double circumference()
void {
    return 2*π*r
    }

double, Area()
void {

Sop  return (22/7)*r*r;
    }

Class Test
  {
public static void main(String
      args[])
  {
Circle c1 = new circle();
      c1.getData(7.5);
      c1.circumference();
      c1.Area();
  }
}
```


[CREATING OBJECT]

(1) Declare object using reference variable

(2) Allocate memory → (Instantiate) [Instantiation]

Rectangle rect1; declare object by reference variable

rect1 = new Rectangle(); (Instantiate)

Action	Statement	Result
Declare	Rectangle rect1;	null
Instantiate	rect1 = new Rectangle;	rect1 → Rectangle object

Combined Statement

Rectangle rect1 = new Rectangle();

Default constructor

rect1.length
rect1.width

15
30

Now Two object rect1 and rect2.

20	← rect2.length
40	← rect2.length

↳ We can create number of object of Rectangle class

Rectangle R1 = new Rectangle()
Rectangle R2 = new Rectangle()
OR
Rectangle R2 = R1

[Accessing class members]
by using dot operator (.)

The instance variable of Rectangle class may be accessed or assigned

rect1.length = 15;
rect1.length = 20;
rect1.width = 30;
rect2.width = 40;

Application of classes and Object

```
Class Rectangle
  {
int length, width ; // Instance variable.
void getData( int x, int y)
  {
  length = x;
  width = y;
  }
int rectArea (    )
  {
  int area = length * width;
  return (area);
  }
int circumference (    )
  {
  int cir =  2(length + width);
  return (cir);
  }
```



```
class RectArea
  {
public static void main (String
  args[])
{
int area1, int area2 ;
Rectangle rect1 = new Rectangle();
Rectangle rect2 = new Rectangle();
  rect1.length = 10;
  rect1. width = 20;
  area1 =  rect1.length *
           rect1. width
  Sop(area1);
  rect2. getData (10, 15);
  area2 =  rect2 . rectArea();
    Sop( area2)
```

Output : 200
150

[Access Variable by object]

```
class TestObject
 {
 int x = 5;
public static void main (
        String args[])
 {
 int z = 10;
TestObject ob = new TestObject();
                    // create
                    object
  System.out.println(z);
  System.out.println (ob.x);
    }}
```

(Multiple) objects

```
public class Multiple_object
  {
  int x = 5;
  public static void main(String args[])
  {
  Multiple_object o1 = new Multiple_object();
  Multiple_object o2 = new Multiple_object();
```



```
sop(o1.x);      output 5, 5
sop(o2.x);
  }}
```

static method vs Normal method

```
class Test_static
  {
static void mystatic ( )
    {
    S.O.P ("static method is called
            with out create object");
    }
void my_method ( )
  {
  So.p("my method is called
        creating object");
  }
public class static void
  main ( String args[])
  {
    mystatic ( );     call static
                      method
  Test_static s1 = new
  Test_static ( );
```

```
s1.mymethod();
}}
```

output:

[Java Constructor]

(1) A Constructor in Java is special method that is used to initialized objects. The Constructor is called when object of a class is created. It is used to initial value for object Attribute.

(2) They not Speify a return type not even void implicit return type class object. This because they instance of class itself.

(3) Class Name and method Name same.

[Types of Java Constructors]

There are two types of Constructor in Java

(1) Default Constructor (No argum
(2) parameterised Constructor

(4) A Java Constructor can not be abstract, static, final, Synchronized.

[Default Constructor] A Constructor is called Default Constructor " when it does not have any parameter.

(A)
```
class Bike1
{
Bike1 ( )          // Default constructor
{
System.out.println("Bike is created");
}
public static void main(String args[])
{
Bike1 b1 = new Bike( ); // Calling Constructor.
}}
```
output Bike is created

(B)
```
class Point
{
int x;
int y;
Point ( )          // Default Constructor
{ x = 2;
  y = 3; }
```



```
void display_point ( )  // method
{
System.out.println(x + ", " + y);
}
class Point_Test
{
public static void main(String args[])
{
Point P1 = new Point( );
Point P2 = new Point( );
P1.Display_Point( ); // Calling method
P2.Display_Point( ); // Calling method
}
}
```
(2, 4)
(2, 4)

[Default Constructor]

(1) Default Constructor does not have any parameter list

Example :

```
(1) class Bike
    {
    Bike()  // Default Constructor
    {
    System.out.println("BIK is created");
    }
    public static void main(String
      args[])
    {
    Bike b1 = new Bike(); //Calling
    }}
```

default Constructor

→ Here no Constructor But automatically one add by JVM

```
(2) class Bike
    {
    public static void main(String
                  args[]
    {
    System.out.println("Bike is created");
    Bike b1 = new Bike();
    }}
```

output : "Bike is created"

(3)

```
class Bike
  { }
```

⇒ Compiler

```
class Bike
  {
  Bike() { }
  }
```

Here add default Constructor

```
class Bike
  {
  Bike()
  {
  }
  public static void main(String args)
```

```
{
System.out.println(" Bike is created");
}
Bike b1 = new Bike();
}
}
```

output Bike is created.

Q: What is purpose of default Constructor ?

The default Constructor is used to provide the default values to the object like zero, null, depending on the type.

Example of default Constructor that display the default values.


```
class student
{
int id;
String name;
// method to display the value of
   id and name.
void display()
{
System.out.println(id+" "+name);
}
public static void main(String args
{
Student s1 = new Student();
Student s2 = new student();
s1.display();
s2.display();
}
}
```

automatic create default Construc

output 0 null
0 null

How default Constructor add by (Compiler)

```
class Student
{
int id;
String name;
Student ()
 {


 }
void display ()
  {
sop(id + "  " + name);
  }
public static void main(String
 args[]
  {
Student S1= new Student();
Student S2= new Student();
 }
```

output
0 null
0 null

Final Example (default Constructor)

```
class Student
  {
  int id;
  String name;
  Student ()
   {
  id= 10;
  name= "ved";
  sop(id); sop(name);
  }
  void display()
   {
  sop(id + " " + name);
   }
public static void main(String args
   {
  Student S1= new Student()
    S1.display()
   }
  }
```

10
ved
10 ved

Q: Why use the parameterised Constructor ?

The parameterised Constructor is used to provide different value to object.

Example:

```
class student
int id;
String name
Student (int x, String s)
    {
id = x;
name s;
    }
void display( )
   {
Sop( id + " " + name);
   }
public static void main (String args)
   {
Student S1 = new Student (20, "ved");
student S2 = new Student (21, "Ram");
   S1. display();  }}
   S2. disply();
```


Constructor overloading / method overloading

[Method overloading]

* Multiple method same name but different parameter list

* More than one methods in a class with same name but different parameter List.

Example

1. int mymethod (int x)
2. float mymethod (float x)
3. double my method (double x)
4. int my method (int x, int y)

Example

① class Methodoverloading
{
static int plusmethod (int x, int y)
{
sop("Sum is is =" + (x+y));
}
static double plusmethod (double x, double y)
{
sop("Sum is =" + (x+y));
}
static float plusmethod (float x, float y)
{
sop("Sum is =" + (x+y));
}
public static void main (String args[])
{
plusmethod (8,5);
plusmethod (4.3f, 5.0f);
plusmethod (4.0, 7.0);
}}

Example2

class TestOverLoading
{
int Areq (int i)
return i*i;
}
int Areq (int a, int b)
{
return a*b;
}}
Class Areq_overloaded
{
public static void main (String args[])
{
TestOverLoading t1 = new TestOverLoading();
int areq;
areq = t1.Areq (5);
System.out.println("Areq of Square is =" + areq);
int areq1;
areq1 = t1.Areq (4, 5);
System.out.println("Areq of Rectangle =" + areq1);

[Benifits of Using Method overloading]

1. Method overloading increase the readability
2. This Provides flexibility to programmer so they can call the same method for different data type
3. It reduces the execution time binding is done Compilation time
4. minimised Complexity
5. Save memory.

[We Take another Example]

```
class Test byteshortOverloading
{
void show (int x)
  {
    s.o.p("Integer=" + x);
  }
void show (String s)
  {
    s.o.p("String=" + s);
```



```
void show (byte b)
  {
s.o.p("byte=" + b);
  }
void show (short s1)
  {
s.o.p("short=" + s1);
  }
class Test {
public static void main(String args)
    {
byteshortOverloading b1= new
    byteshortOverloading ();
  b1.show(25);
  byte a= 25;
  b1.show(a);
  short b2= 40;
  b1.show(b2);
  b1.show("red");
}}      output:
```

Integer = 25

String = Ved

byte = 35

short = 40

[Constructor overloading]

(1) In a class more than one Constructor is called Constructor overloading.

(2) Different type of argument // parameter list

(3) Different ~~type~~ number of arguments passed to it.

Example Student (int i, String s)

Student (int i, String s, int a)

Constructor Overloading Example

```
class student
 {
int id ;
string name ;
int age ;
Student (int x, string s)
    {
   id = x;
   name = s;
    }
Student (int x, string s, int a)
    {
   id = x;
   name = s;
   age = a;
    }
void display ( )
    {
System.out.println (id + " "
+ name + " " + age)
    }
```

```
Class Student
  {
public static void main(String args[])
  {
Student S1 = new Student(4, "Gyanjan");
Student S2 = new Student(5, "veena", 10);
    } S1.display();
    } S2.display();
        }
      }
```

output 4 Gyanjan

5, veena . 10

Difference B/W Constructor and method

	Constructor	Method
(1)	A Constructor is used to initialize the state of object	A method is used to expose the behavior of an object
(2)	Constructor must not have any return type	Method have return type
(3)	Constructor invoked implicitly	method invoke explicitly
(4)	The Java provide default Constructor if you do not have any Constructor in a class	The method is not provided by Java compiler any case.
(5)	Constructor name must be same as the class Name	method name may or may not be same.

[Java Copy Constructor] X

There is no Copy Constructor in Java

There are many way to copy the values of one object to another object.

(1) By Constructor

(2) By assigning the values of one object into another

(3) By clone() method of Object class (Using Constructor)

Program to initialise the values from one object to another object

```
class student
  {
int id;
String name;
double salary;
Student(int a, String b, double c)
  {
name=a;
salary=c;
id=a;
}
```



```
student(student s)
  {
  id = s.id;
  name = s.name;
  salary = s.salary;
  }
void display()
  {
system.out.println(id+" "+name+" "
                   +salary);
  }
public static void main(String
                        args[])
{
Student s1 = new Student(
     (10 "ved", 40000.50)
Student s2 = new student(s1)
  s1.display(); } }
  s2.display();
      OR
  s2.display();
  s2.display()
```

output 10 ved 40000.50
10 ved 40000.5

*75] Remaing part

Date : / /

[Copying values without Constructor]

We can copy the values of one object into another objects values.

There is no need to create Constructor.

```
class Student
  {
int id
String name;
Student (int x, int y)
                String
  {
  id = x;
  name = y;
  }
Student ( ) { }
void display ( ) { sop (id + "" name);
public static void main (String[] args
Student s1 = new Student (10, "ved
Student s2 = new student ( ) ;
    s2.id = s1.id ;
    s2.name = s1.name ;
    s1.display ( ) ;
    s2.display ( ) ;
  }
```

output 10, ved

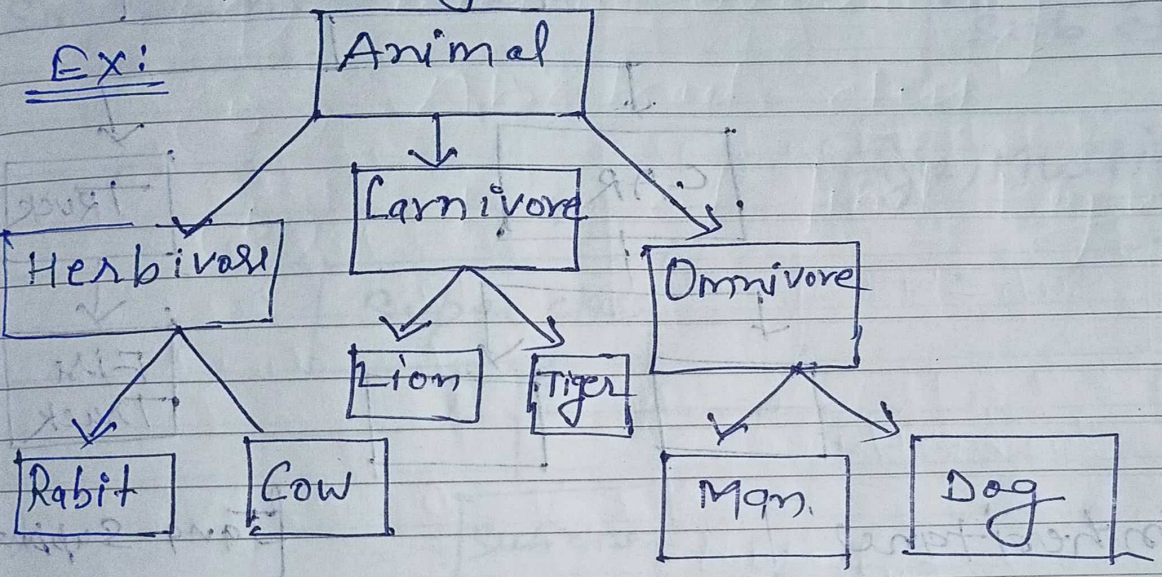
10 ved

[Concept of Inheritance]

[Concept of Inheritance]

Inheritance is biologically term. In which there are two or more classes, one super class and other subclass. Subclass acquire the property of super class.

Ex:

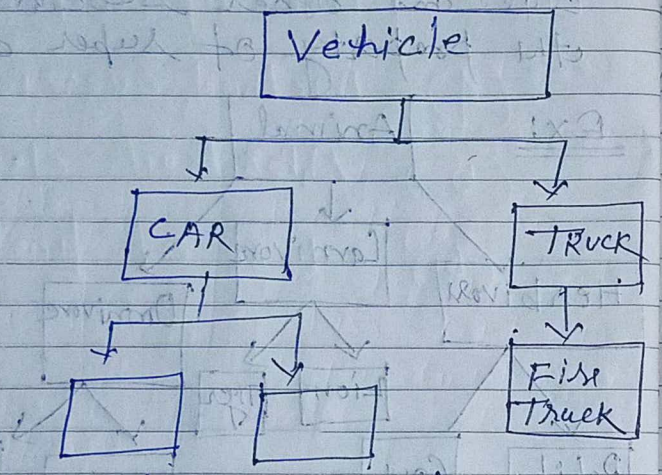


[Single Multi-level inheritance]

(1) Subper class: A class that is inherited is called a super class

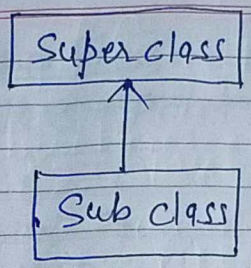
(2) Sub class: The class that does inheriting or acquiring property is called Subclass.

Reusability: It is a Mechanism which facilitates you to reuse the data and method of the existing class when one create a new class.

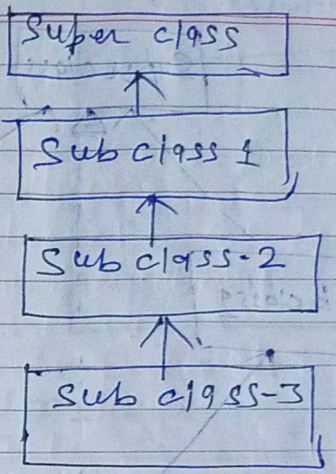


[Type of Inheritance] [Java Support]

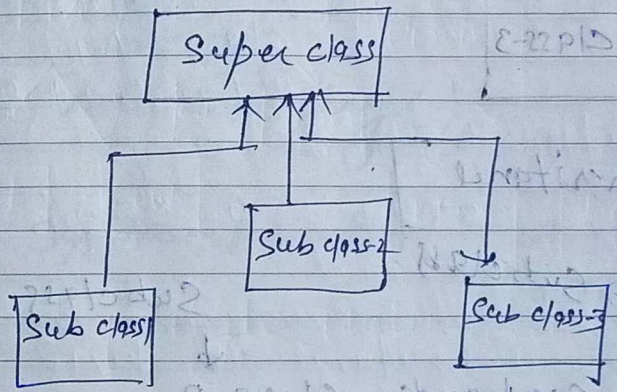
(1) Single Inheritance ✓ (yes)
(2) Multi Level Inheritance ✓ (yes)
(3) Multiple Single Inheritance ✓ (yes)
(4) Multiple Inheritance (X) No
(5) Hybrid Inheritance (X) No



Fig(1)
Single Inheritance



Fig(2) Multi Level inheritance



Fig(3) → Multiple single Inheritance (Hierarchy) Inheritance

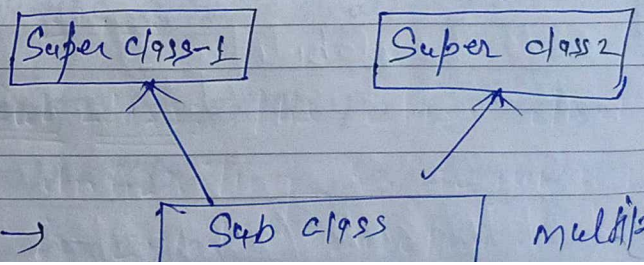


Fig (4) → Multiple Inheritance

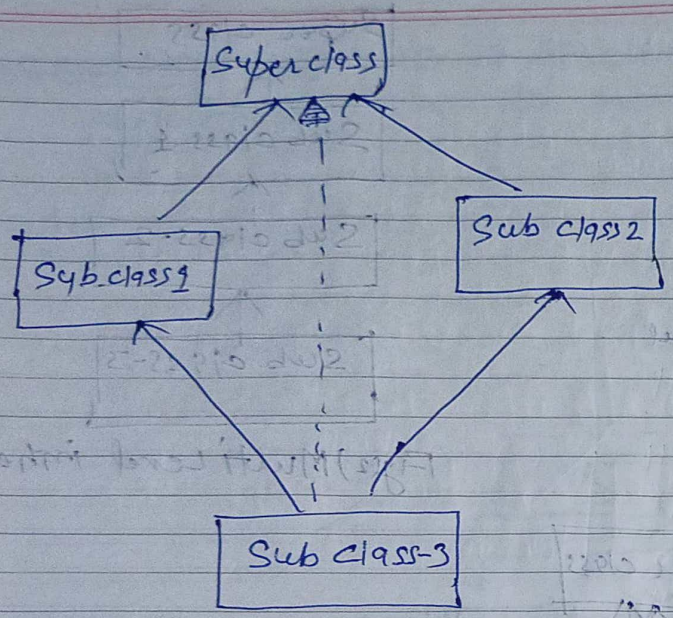
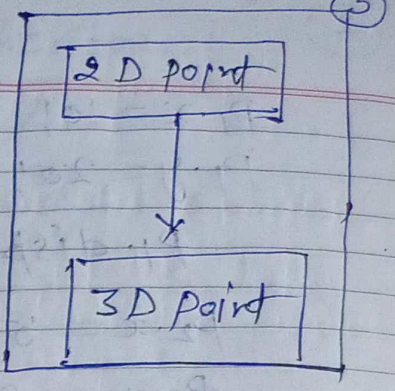


Fig: Hybrid Inheritance

Syntaxe → Subclass

```
class A extends class B
{
}
```

Subclass ↓



Example:

```
class point2D
{
int x;
int y;
void display() {
System.out.println("x="+x+"y="+y);
class Point3D extends point2D
{
int z;
void display()
{
System.out.println("x="+x+"y="+y+
                   "z="+z);
}
}
class SimpleSingleInheritance {
public static void main(String args[])
{
Point2D P1 = new Point2D()
Point3D P2 = new Point3D();
```

```
P1.x = 10;
P1.y = 20;
P1.display();
P2.x = 5;
P2.y = 6;
P2.z = 15;
P2.display();
}
}
```

output x=10 y=20
x=5 y=6 z=15

Note

~~Point2D P3 = new Point2D();~~

```
Point2D P3 = (Point2D)P2
P3.display();
```

output = (10, 20)

// Dynamic binding
// Call to an instance of class Point2D
// Run time polymorphism

Why we use inheritance in Java?

1. for method overriding (So run time polymorphism achieved)
2. for Code Reusability

Note IS-A Relationship (Inheritance)

[Single Inheritance]

Example 2

```
class Employee
{
float salary = 40000;
}
class Programmer extends Employee {
int bonus = 10000;
public static void main (String args[])
{
Programmer P1 = new Programmer();
System.out.println("Bonus" + P1.salary);
System.out.println("salary" + P1.Bonus);
}
}
```

Programmer salary is: 40000.0
Bonus of Program is 10000

Direct Program

```
class Employee {
float salary = 40000;
}
class Programmer
{
static int bonus = 10000;
public static void main (String args[])
  {
  Employee p = new Employee();
  Sop("Programmer salary=" + P.salary);
  Sop("Bonus of Programmer=" + Bonus);
  }
}
```

Some other Examples


```
class Animal
{
void eat() { S.op("Eating...."); }
class Dog extends Animal
{
~~Sop("barking");~~ void bark()
{
  Sop("barking....");
}
class Test-Inheritance
{
public static void main (String args[])
{
Dog d = new Dog();
   d.bark();
   d.eat();
}
}
```

Multilevel Inheritance ✓

```
class Animal
 {
void eat() { sop("eating ....."); }
 }
class extends Dog
   {
void bark() { sop(" barking ....."); }
   }
class Baby Animal (Dog) extends Dog
   {
  void weep() { sop("weeping ....."); }
   }
class Multilevels ~~extends~~
   {
 public static void main (String args[])
        {
 Baby Animal = b1 = Baby Animal();
        b1.weep();
        b1.bark();
        b.eat(); }}
```

output
weeping ...
barking ...

Q: Why multiple inheritance is not Supported by Java

Ans: To reduce the Complexity and Simplify language, multiple Inheritance is not Supported in Java

```
class A
 {
void msg() { sop("Hello"); }}

class B
  {
void msg() { sop("welcome"); }}

 class C extends A, B // Suppose if we wer
   {
public static void main (String args[])
    {
  C object1 = new C();
      object1.msg(); Now which msg() method involved
    } Compile time error
```

Access Modifier

Access Modifier	with in class	with in package	out side package by Subclass	outside package
Private	Yes	No	No	No
Default	yes	yes	No	No
Protected	yes	yes	yes	No
Public	yes	yes	yes	yes

Example

```
class A
{
private int data = 40;
private void msg()
  {
  Sop("Hello Java");
  }
class B
  {
public static void main(String args[])
  {
  A a = new A();
  Sop(a.data);  Compile time error
  a.msg();  // Compile time error
  }
}
```

[Method overriding]

(1) Method overriding is used to provide specific implementation of method which already provided by its super class.

(2) Method overriding is used for runtime polymorphism.

[Rules for Java Method Overriding]

(1) The method must have name as in the parent class

(2) The method must have same parameter as in the parent class.

(3) There must be an IS-A relationship. (Inheritance)

Note: (1) A sub class object can reference a super class variable or method if it is not overridden.

(2) A super class object can not reference a variable or method which is explicit to the sub class object.

[Method override Example]

```
class Vehicle
{
void run()
{
System.out.println("vehicle is running");
}
}
class CBShin extends bike
{
void run()
{
System.out.println("CBShining is running");
}
}
class Testoverride
{
public static void main(String args[])
{
CBShin c = new CBShin();
c.run();
}
}
```

output:

CBShin is running.

[Super Keyword]

(1) The super keyword in Java is reference variable which is used to refer immediate parent class members.

(2) Whenever you create an instance of subclass, an instance of its parent class is created implicity, which is reffered by super keyword.

[Usage of Keyword]

(1) Super can be used to refer immediate parent class instance variable.

(2) Super can be used to invoke immediate parent class method.

(3) super() can be used to invoke immediate parent class constructor.

[with super keyword]

```
class Animal
  {
  String color = "white";
  }
class Dog extends Animal
  {
  String color = "black";
  void printColor() {
    System.out.println(color);
    system.out.println(super.color);
    }
  }
class TestSuper1
  {
public static void main(String args[])
  {
Dog d = new Dog();
  d.printColor();
  }
  }
```

out black
white

Note
with out super

```
String color =
  "white";
```

(With out super)

```
class Animal
 {
String color1 = "White";
 }

class Dog extends Animal
  {
 String color2 = "black";
 void Color ( )
   {
   sop(color1);
   sop(color2);
   }
class Test with out super
  {→ public static void main (String args[]){
Dog d = new Dog()
   d.color()
  }
  }
```

output White
black

Super: Invoking parent class Method

```
class Animal
 {
void eat ( )
 {
sop("Eating Eating ... ");
 }
class Dog extends Animal
  {
void eat ( )
   {
sop("Dog eating ...");
 }
void bark ( ) { sop ("barking"); }
 void work ( )
  {
  super.eat(); //parent eat() invoke
   bark();
   eat();      // Child eat() invoke
  }
  }
Class Test Super 2
  {
public static void main ( String args[])
```

```
Dog d = new Dog();
d.work()
}
```

Animal and Dog both the classes have eat() method. If you call eat() method from Dog class. It will be call eat method of Dog class by default priority is given do local.

To call the super class (Parent class) method, we need use super keyword.

output Eating Eating ...
barking ...
Dog eating ...

[super :- invoking parent class constructor]

```
class Animal
{
Animal ()
{
sop ("Animal is created");
}}
class Dog extends Animal
{
Dog ()
{
sop super ();  // call super class Constructor Animal()
sop ("dog is created");
}}
class Test Super
{
public static void main (String args[])
{
Dog d = new Dog()
}
{
```

output
Animal is created
dog is created

super : invoking parent class Constructor (

[Constructor over loading]

```
class Point2D
  {
  double x, y;
  Point2D( ) {
    x = 0.0;
    y = 0.0;     // Default Initialization
    }
  Point2D (double x, double y, double z)
    {
    this.x = x;     // or  x = a;
    this.y = y;     // or  y = b;
    }
class Point3D extends Point2D
  {
  double z;
  Point3D ( )
    {
    super( );   // Call Parent reference Parent class Constructor
    z = 0.0;
    }   // Default Initi
```

```
Point3D (double x, double y, double z)
    {
    super(x, y);
    this.z = z;     // or  z = a;
    }
  }
class Test Super 4 {
  public static void main (String args[])
    {
  Point3D P = new 3D(2.0, 3.0, 4.0);
  Point3D P1 = new 3D( );
    }
```

(Dynamic binding)
[Dynamic Dispatch method]
*(Run time polymorphism)

```
class Bike
  {
void run() { sop("running"); }
  }

class Splender extends Bike
  {
void run() { sop("splender is running"); }
  }

class Dynamic_Dispatch
  {
public static void main(String args[])
  {
Splender b1 = new Splender();
   b1.run();
Bike b2 = new Bike();
  b2.run();
Bike b3 = new Splender(); //Up casting
   b3.run();
  }
}
```

output
running
splender is running
splender is running

For the b3 object, we are calling the run() method by reference variable of Super class. Since it refers to Sub class object and Subclass method overrides the Super class method, the Subclass method invoke Run time.

Other Example Dynamic binding

```
class A
  {
void callme() {("I am here");}}
class B extends A
  {
void callme() { sop("I am in college");}
  }
class C
  {
public static void main(String args[])
  {
A a = new B();
  a.callme();
B b = new B();
```

output
I am in college
I am in college

[Inheritance through Constructor] (Program)

```
class Box {
double width, height, depth;
 Box()  // default Constructor
  {
  width = 0.0;
  height = 0.0;
  depth = 0.0;
  }
Box(double w, double h, double d)
   {
  width = w;
  height = h;
  depth = d;
   }
double volume()
   {
  return (width * height * depth);
   }
}
class BoxWeight extends Box
   {
  double weight;
  BoxWeight (double w, double h
             double d, double m)
  {
  width = w;
  height = h;
  depth = d;
  weight = m;
  }
}
```

Using Super

```
BoxWeight(double w, double h, double d, double m)
  {
  super();
  weight = m;
  }
}
class Demo extends
  {
Box b1 = new Box();
Box b3 = new Box(1,1,1);
BoxWeight b2 = new BoxWeight(2,3,4,0.076);
double vol;
vol = b1.volume();
System.out.println("volum of Box" + vol);
                    //using default
vol = b2.volume();
Sop("volum of Box using IInd
        Constructor" + vol);
Sop(b2.weight);
  }
```

Arguments

output

```
0
24
.076
```

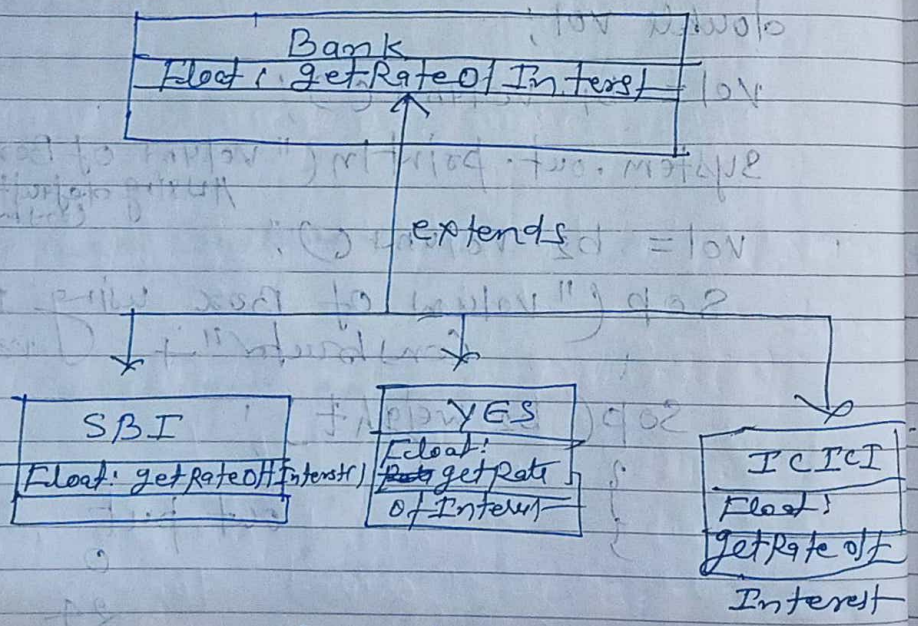
Example:

[overriding]

A Real Life Example of Java Method overriding

Consider a Scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to the bank.

For Example, SBI, ICICI, YES banks provides 8%, 7%, 9% rate of interest.



Java method overriding is mostly used in Runtime Polymorphism

```
class Bank
{
int getRateOfInterest()
{
return 0;
}}

class SBI extends Bank
{
int getRateOfInterest() {
return 8;
}}

Class ICICI extends Bank
{
int getRate of Interest()
{
return 9;
}}

class YES extends Bank
{
int getRate of Interest()
{
return 10;
}}
```

```
class Test {
public static void
main(String args[])
{
SBI S1 = new SBI();
ICICI S2 = new ICICI();
YES S3 = new YES();
sop("SBI" + S1.getRate
of Interest());
sop("ICICI" +
S2.getRateOfInterest());
sop("yes"
+ S3.getRate
of Interest())
}
}
```

output

8
9
10

* Final method can not be override

* Final class can not be extented

```
final class Bike
{ } // if we write final run()
                    { sop( )
                    }
class Honda extends bike
{                   // Error final
                       class can not
                       (inherited)
void run ()
  {
  sop ("bike is running");
  } }
```

Not possible

```
class Demo Final class Demo
  {
public  static void main(String a[])
  {
Honda h = new Honda();
   h.run();
  }
}
```

[Interface]

(1) Interface just like a class, It is collection of field and abstract method, final and static field.

(2) An Interface does not have instance variable.

(3) All the method of an interface are automatically public.

Interface Declaration

```
interface Animal
  {
    void show ();
  }
class Dog implements Animal
  {
  void show ()
    {
    sop ("Animal is eating");
    } }
class TestInterface
  {
  Dog d1 = new ()
```

```
d1.show();
  }
}
```

Example [Interface - Example]

```
interface A
{
void show();
void Sound();
}

interface B
{
void show1();
void Sound1();
}

class C implements A,B
{
public void show()
{
int a=10;
int b=20;
Sop("Sum is="+(a+b));
}
void Sound()
{
Sop("chi -- chi -- chi");
}
public void show1()
{
int a = 10;
int b = 20;
Sop("mul"+(a*b));
}
public void Sound()
{
Sop("mew -- mew");
}}

class Interface
{
public static void
main(String args[])
C a = new C();
a.show();
a.Sound();
a.show1();
a.Sound1();
}
}
```

out put

```
30
chi -- chi --
200
mew -- --
```

```
A a1 = new C();
a1.show();
a1.Sound();
```

out put

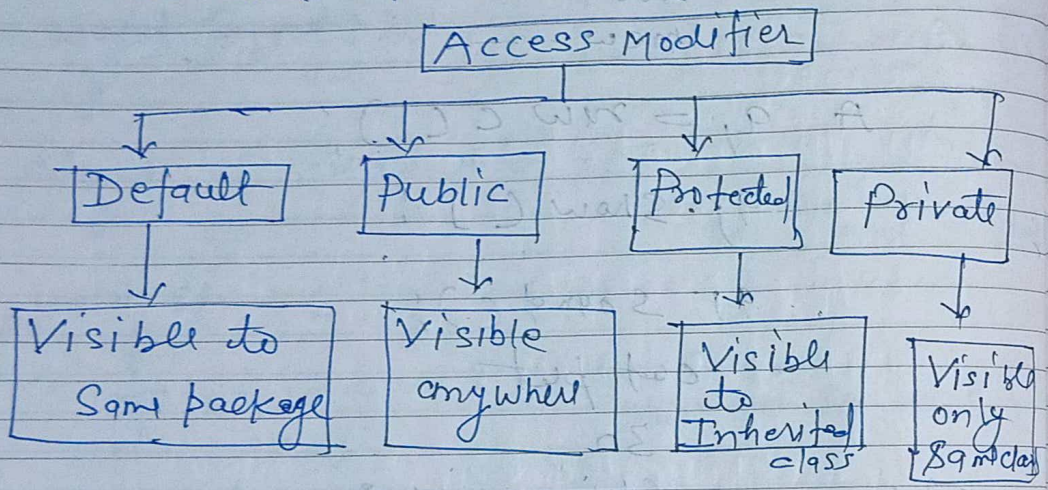
```
30
chi -- chi
```

```
B b1 = new C();
b1.show1();
b1.Sound1();
```

```
200
mew -- --
```

(1) Access Specifier
(2) Package
(3) Interfaces

Concept of access modifiers



The access modifier in Java accessibility (Scope) of data member, method, constructor, or class.

Note: class is only default, public

class A and class B is not accessible to each other vice-versa

Access Level / Modifier	class	Package	Subclass	Everywhere
Public	✓	✓	✓	✓
Protected	✓	✓	✓	X
Default	✓	✓	X	X
Private	✓	X	X	X

Default:

```
class A
{
void show() { Sop("I am in classA"); }
}
```

Save this class A.class in Subdirectory "Pack1"

```
class B
{
public static void main(String args[])
{
A a = new A(); // Compile time error.
a.show(); // Compile time error.
}}
```

Save: B.java in Subdirectory Pack2

// Here default modifier if we save both files in Pack1 Then it no error.