**Program 4:**

**Title:** Write a java program to implement Box class application to depict constructor overloading .

**Problem Description:** Given three classes called A, B, and C, C can be a subclass of B, which is a

subclass of A. When this type of situation occurs, each subclass inherits all of the traits

found in all of its super classes. methods with different parameters.

**Method:** In this program, C inherits all aspects of B and A. To see how a multilevel hierarchy can be useful, consider the following program. In it, the subclass BoxWeight is used as a superclass to create the subclass called Shipment. Shipment inherits all of the traits of BoxWeight and Box, and adds a field called cost, which holds the cost of shipping such a parcel.

**Theory Reference:** Module 3

class Box {

double length;

double width;

double height;

// Default constructor

Box() {

length = -1;

width = -1;

height = -1;

}

// Parameterized constructor

Box(double len, double wid, double hgt) {

length = len;

width = wid;

height = hgt;

}

// Copy constructor

Box(Box ob) {

length = ob.length;

width = ob.width;

height = ob.height;

}

Box(double len)

{

width=height=length=len;

}

// Method to calculate volume

double volume() {

return length \* width \* height;

}

}

// Define the BoxWeight class that extends Box

class BoxWeight extends Box {

double weight;

// Default constructor

BoxWeight() {

super();

weight = -1;

}

// Parameterized constructor

BoxWeight(double len, double wid, double hgt, double wt) {

super(len, wid, hgt);

weight = wt;

}

BoxWeight(BoxWeight ob) {

super(ob);

weight = ob.weight;

}

BoxWeight(double len, double wt)

{

super(len);

weight=wt;

}

}

// Define the Shipment class that extends BoxWeight

class Shipment extends BoxWeight {

double cost;

// Default constructor

Shipment() {

super();

cost = -1;

}

// Parameterized constructor

Shipment(double len, double wid, double hgt, double wt, double c) {

super(len, wid, hgt, wt);

cost = c;

}

// Copy constructor

Shipment(Shipment ob) {

super(ob);

cost = ob.cost;

}

Shipment(double len, double wt, double c)

{

super(len,wt);

cost=c;

}

}

public class BoxClassApplication

{

public static void main(String[] args)

{

Shipment shipment1 =new Shipment(10, 20, 15, 10, 3.41);

Shipment shipment2 =new Shipment(2, 3, 4, 0.76, 1.28);

double vol;

vol = shipment1.volume();

System.out.println("Volume of shipment1 is " + vol);

System.out.println("Weight of shipment1 is "+ shipment1.weight);

System.out.println("Shipping cost: $" + shipment1.cost);

System.out.println();

vol = shipment2.volume();

System.out.println("Volume of shipment2 is " + vol);

System.out.println("Weight of shipment2 is "+ shipment2.weight);

System.out.println("Shipping cost: $" + shipment2.cost);

}

}

**Program 5:**

**Title:** Write a java program to solve Tower of Hanoi Problem using Stack .

**Problem Description:** Move all the disks stacked on the first tower over to the last tower using a helper tower in the middle. While moving the disks, certain rules must be followed. Only one diskcan be moved.A larger disk can not be placed on a smaller disk.

**Method:** In this program, Solving the Tower of Hanoi problem using a stack is an elegant and efficient approach. The Tower of Hanoi problem involves three rods and a number of disks of different sizes that can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, with the smallest disk at the top, and the objective is to move the entire stack to another.

**Theory Reference:** Module 3

**Code:**

import java.util.\*;

 /\* Class TowerOfHanoiUsingStacks \*/

 public class p5

 {

     public static int N;

     /\* Creating Stack array  \*/

     public static Stack<Integer>[] tower = new Stack[4];

     public static void main(String[] args)

     {

         Scanner scan = new Scanner(System.in);

         tower[1] = new Stack<Integer>();

         tower[2] = new Stack<Integer>();

         tower[3] = new Stack<Integer>();

         /\* Accepting number of disks \*/

         System.out.println("Enter number of disks");

         int num = scan.nextInt();

         N = num;

         toh(num);

     }

 /\* Function to push disks into stack \*/

     public static void toh(int n)

     {

         for (int d = n; d > 0; d--)

             tower[1].push(d);

         display();

         move(n, 1, 2, 3);

     }

     /\* Recursive Function to move disks \*/

     public static void move(int n, int a, int b, int c)

     {

         if (n > 0)

         {

             move(n-1, a, c, b);

             int d = tower[a].pop();

             tower[c].push(d);

             display();

             move(n-1, b, a, c);

         }

     }

     /\*  Function to display \*/

     public static void display()

     {

         System.out.println("  A  |  B  |  C");

         System.out.println("---------------");

         for(int i = N - 1; i >= 0; i--)

         {

             String d1 = " ", d2 = " ", d3 = " ";

             try

             {

                 d1 = String.valueOf(tower[1].get(i));

             }

             catch (Exception e){

             }

             try

             {

                 d2 = String.valueOf(tower[2].get(i));

             }

             catch(Exception e){

             }

             try

             {

                 d3 = String.valueOf(tower[3].get(i));

             }

             catch (Exception e){

             }

             System.out.println("  "+d1+"  |  "+d2+"  |  "+d3);

         }

         System.out.println("\n");

     }

 }