1. Illustrate the importance of Constructor Overloading with appropriate example

**package** Pranav221047001;

**class** Box {

**double** width, height, depth;

// constructor used when all dimensions

// specified

Box(**double** w, **double** h, **double** d) {

width = w;

height = h;

depth = d;

}

// constructor used when no dimensions

// specified

Box() {

width = height = depth = 0;

}

// constructor used when cube is created

Box(**double** len) {

width = height = depth = len;

}

// compute and return volume

**double** volume() {

**return** width \* height \* depth;

}

}

**class** construtor{

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

// create boxes using the various

// constructors

Box mybox1 = **new** Box(30, 15, 30);

Box mybox2 = **new** Box(5);

Box mycube = **new** Box(10);

**double** vol;

// get volume of first box

vol = mybox1.volume();

System.***out***.println(" Volume of mybox1 is " + vol);

// get volume of second box

vol = mybox2.volume();

System.***out***.println(" Volume of mybox2 is " + vol);

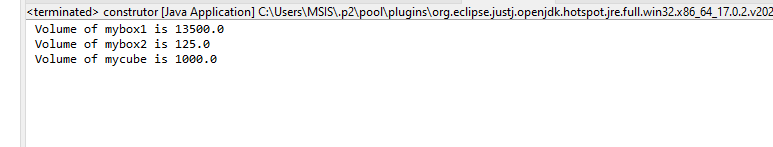
// get volume of cube

vol = mycube.volume();

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

}

}



2a. Java’s support to multi-level inheritance

**package** Pranav221047001;

**class** Shape {

**public** **void** display() {

System.***out***.println("Inside display function");

}

}

**class** Rectangle **extends** Shape { //class rectangle inherits properties of shape

**public** **void** area() {

System.***out***.println("Inside area function");

}

}

**class** Cube **extends** Rectangle { //class cube inherits properties of both shape and rectangle

**public** **void** volume() {

System.***out***.println("Inside volume function");

}

}

**public** **class** threeclasses{

**public** **static** **void** main(String[] arguments) {

Cube cube = **new** Cube();

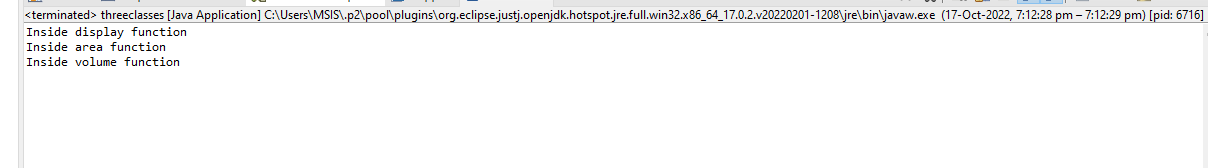
cube.display();

cube.area();

cube.volume();

}

}



2.b Usage of Super from at method level and constructor level

**package** Pranav221047001;

**class** Animal { // Superclass (parent)

**public** **void** animalSound() {

System.***out***.println("The animal makes a sound");

}

}

**class** Dog **extends** Animal { // Subclass (child)

**public** **void** animalSound() {

**super**.animalSound(); // Call the superclass method

System.***out***.println("The dog says: bhow bhow");

}

}

**class** Cclass {

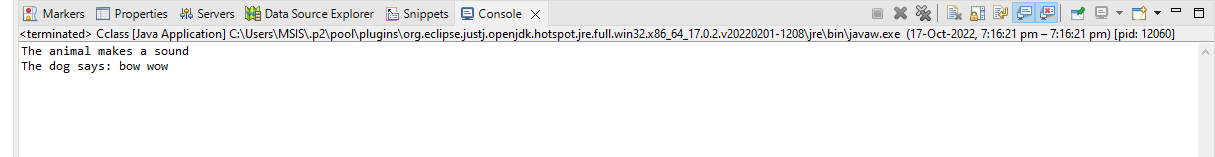
**public** **static** **void** main(String[] args) {

Animal myDog = **new** Dog(); // Create a Dog object

myDog.animalSound(); // Call the method on the Dog object

}

}



2.c Working of Protected access

**package** Pranav221047001;

**public** **class** Paccess{

**protected** **void** msg(){System.***out***.println("Hello World !!");}

}

**package** Pranav221047001;

**class** Protect **extends** Paccess{

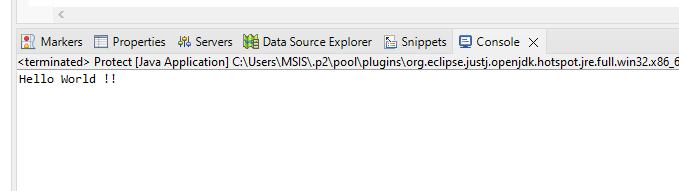
**public** **static** **void** main(String args[]){

Paccess obj = **new** Paccess();

obj.msg();

}

}



3. Differentiate between method overloading and overriding with appropriate example

**package** Pranav221047001;

**class** Adder{

**static** **int** add(**int** a,**int** b)//Overloading Class add with two parameters

{

**return** a+b;

}

**static** **int** add(**int** a,**int** b,**int** c)// //Overloading the same Class add with three parameters

{

**return** a+b+c;

}

}

**class** Overload

{

**public** **static** **void** main(String[] args)

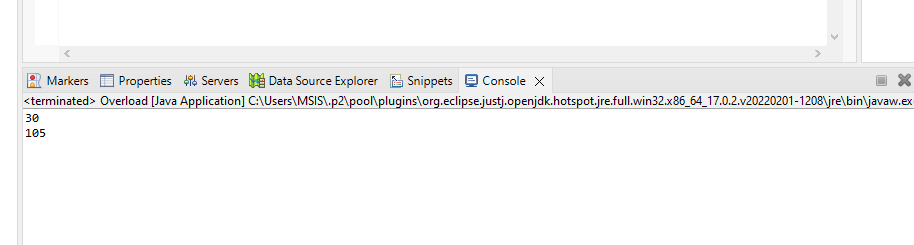
{

System.***out***.println(Adder.*add*(10,20));

System.***out***.println(Adder.*add*(2,3,100));

}

}



Example 2: Method Overloading: changing data type of arguments

**package** Pranav221047001;

**class** add1{

**static** **int** add(**int** a, **int** b){**return** a+b;

}

**static** **double** add(**double** a, **double** b){**return** a+b;}

}

**class** Over{

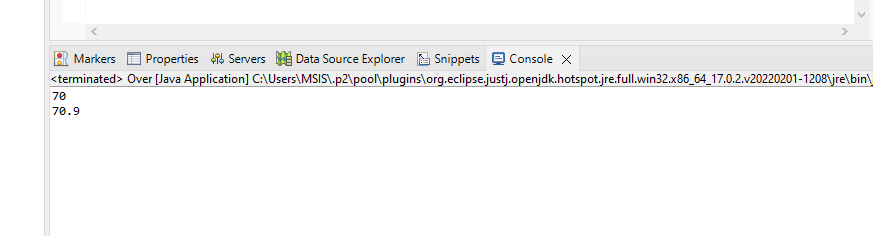
**public** **static** **void** main(String[] args){

System.***out***.println(add1.*add*(20,50));

System.***out***.println(add1.*add*(20.3,50.6));

}

}



Method Overriding

**package** Pranav221047001;

**class** Bank{

**int** getRateOfInterest()//Method

{

**return** 0;

}

}

//Creating child classes.

**class** SBI **extends** Bank{

**int** getRateOfInterest()//Same methodname

{

**return** 8;

}

}

**class** ICICI **extends** Bank

{

**int** getRateOfInterest()//Same methodname

{

**return** 7;

}

}

**class** AXIS **extends** Bank

{

**int** getRateOfInterest()//Same methodname

{

**return** 9;

}

}

**class** Override{

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

SBI s=**new** SBI();

ICICI i=**new** ICICI();

AXIS a=**new** AXIS();

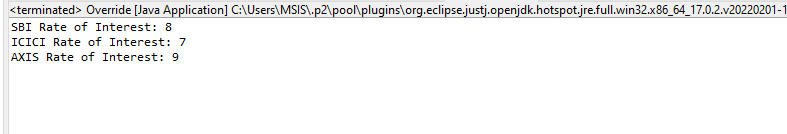
System.***out***.println("SBI Rate of Interest: "+s.getRateOfInterest()); //Same methodname of class SBI

System.***out***.println("ICICI Rate of Interest: "+i.getRateOfInterest()); // Same methodname of class ICICI

System.***out***.println("AXIS Rate of Interest: "+a.getRateOfInterest()); //Same methodname of class AXIX

}

}



4. Demonstrate the usefulness of finalize() method

**package** Pranav221047001;

**class** Finaliza

{

**public** **static** **void** main(String[ ] args) // A "main thread" gets introduced

{

String s = **new** String("Gate Vidyalay"); // A String object gets created

s = **null**; // String Object becomes eligible for garbage collection

System.*gc*( ); // A request is made to JVM for running garbage collector ; A "gc thread" gets introduced

System.***out***.println("End of main method");

}

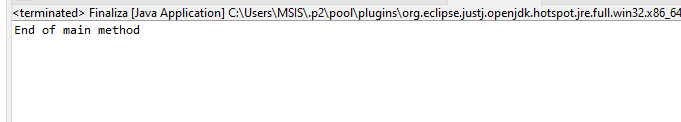
**public** **void** finalize( ) // Test class finalize( ) method

{

System.***out***.println("Finalize method of Test class");

}

}



5. Illustrate the concepts of Abstract class and Interface with appropriate example

**package** Pranav221047001;

**interface** A{

**void** a();//bydefault, public and abstract

**void** b();

**void** c();

**void** d();

}

//Creating abstract class that provides the implementation of one method of A interface

**abstract** **class** B **implements** A{

**public** **void** c(){System.***out***.println("I am :C");}

}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods

**class** M **extends** B{

**public** **void** a(){System.***out***.println("I am :A");}

**public** **void** b(){System.***out***.println("I am :B");}

**public** **void** d(){System.***out***.println("I am :D");}

}

//Creating a test class that calls the methods of A interface

**class** Interface{

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

A a=**new** M();

a.a();

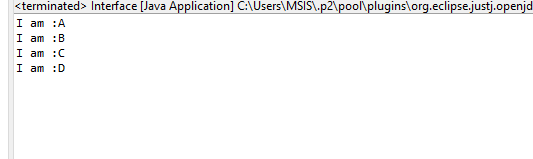
a.b();

a.c();

a.d();

}

}



6. Illustrate the significance of Encapsulation – namely the control the concept provides in your application through appropriate examples.

**package** Pranav221047001;

**class** Student {

**private** **int** Student\_Id;

**private** String name;

//getters, setters for Student\_Id and name fields.

**public** **int** getId() {

**return** Student\_Id;

}

**public** **void** setId(**int** s\_id) {

**this**.Student\_Id = s\_id;

}

**public** String getname() {

**return** name;

}

**public** **void** setname(String s\_name) {

**this**.name = s\_name;

}

}

**public** **class** Encap{

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

//create an object of Student class

Student s=**new** Student();

//set fields values using setter methods

s.setId (221047001);

s.setname("Pranav Khedekar");

//print values using getter methods

System.***out***.println("Student Data:" + "\nStudent ID:" + s.getId()

+ "\nStudent Name:" + s.getname());

}

}

