Name: Shaikh Adnan Shaukat Ali

Roll Number: 55

Sem: III

Subject: OOPM(JAVA)

Number of Experiments: 15

Branch: Computer

Experiment NO. 1

Aim: To understand basic concept of programming like branching and looping in Object Oriented Programming (OOP) Java

Problem Statement: 1) Write a program to print all Even and Odd Numbers between 2 to 100 using if statement and for loop.

2) Write a program to print Star pattern

Theory:

Program 1:

public class Exp\_1{

public static void main(String[] args){

System.out.print("Even no: ");

for(int i=2;i<=100;i++){

if(i%2==0){

System.out.print(i+"\t");

}

}

System.out.println("\n Odd no: ");

for(int i=2;i<=100;i++){

if(i%2!=0){

System.out.print(i+"\t");

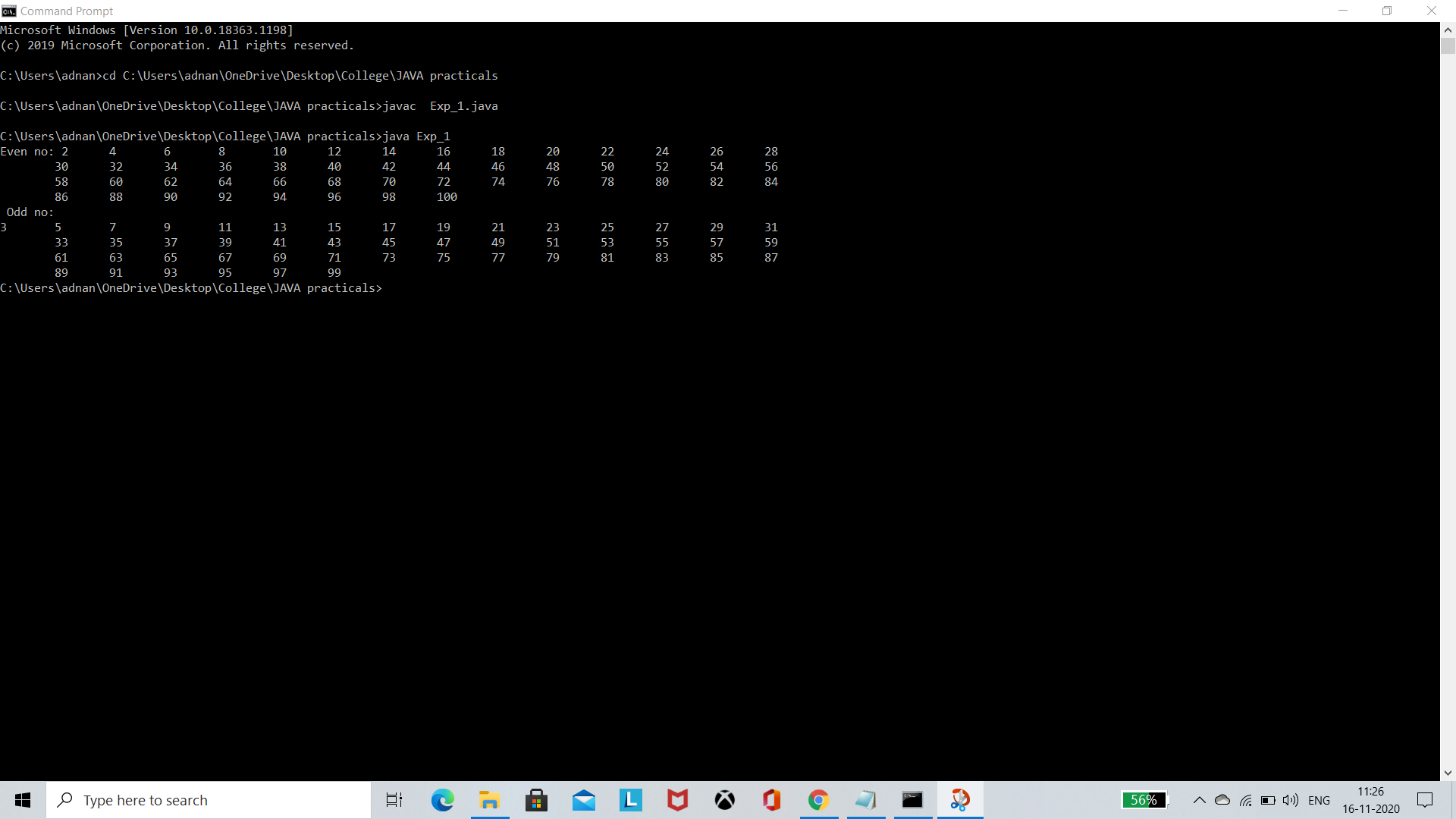
}

}

}

}

Output:



Program 2:

public class StarPlatinum{

public static void main(String[] args){

for(int i=1;i<=4;i++){

for(int j=1;j<=i;j++)

System.out.print("\*");

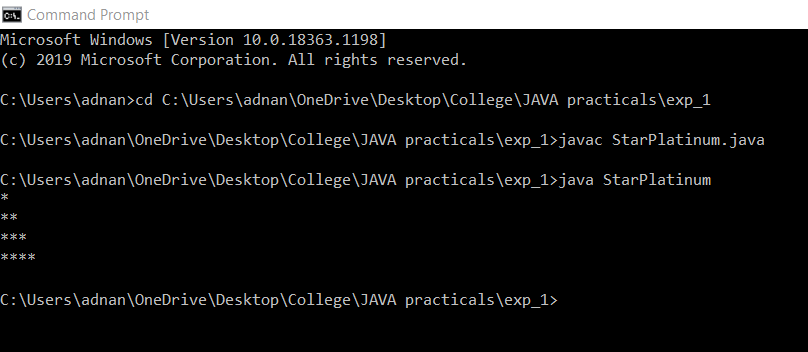
System.out.println();

}

}

}

Output:



Conclusion: Hence, we learned the basic concepts of programming in OOP Java

Experiment NO. 2

Aim: To understand the concept of accepting input through keyboard in Object Oriented Programming (OOP) Java

Problem Statement: To accept String from user and print it on a terminal as a output

Theory:

import java.util.Scanner;

public class Exp\_2 {

public static void main (String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Your Input here: ");

String inputString = scanner.nextLine();

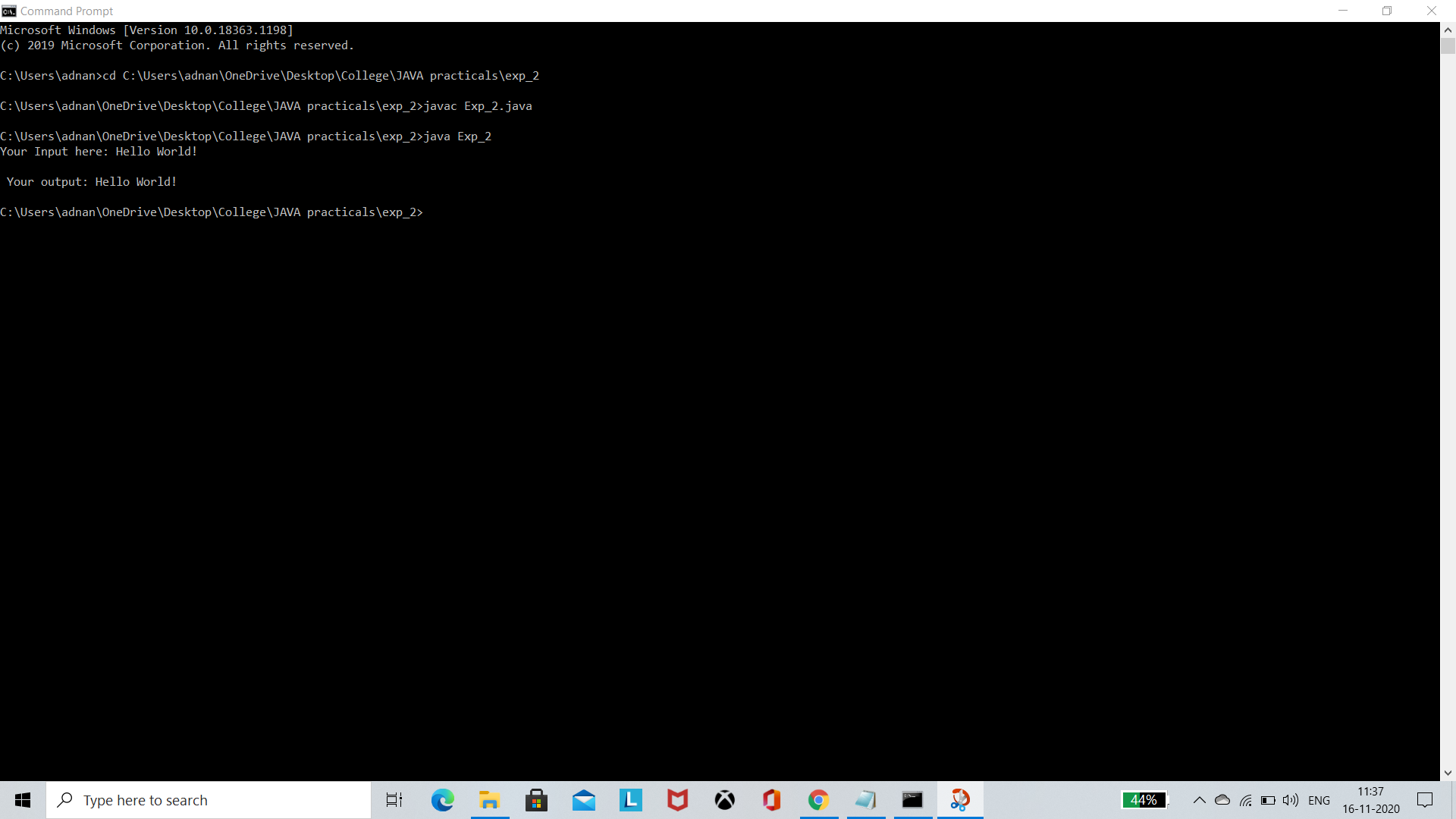
System.out.print(" \n Your output: ");

System.out.println(inputString);

}

}

Output:



Conclusion: Hence, we learned the Scanner class and how to accept input through keyboard using Scanner object in OOP Java

Experiment NO. 3

Aim: To understand the concept of class and objects in Object Oriented Programming (OOP) Java

Problem Statement: Create a class and its objects and output how many objects are created of a class

Theory:

public class CandO {

static int x = 1;

public CandO(){

System.out.println("Object "+x+" is created");

x++;

}

public static void main(String[] args) {

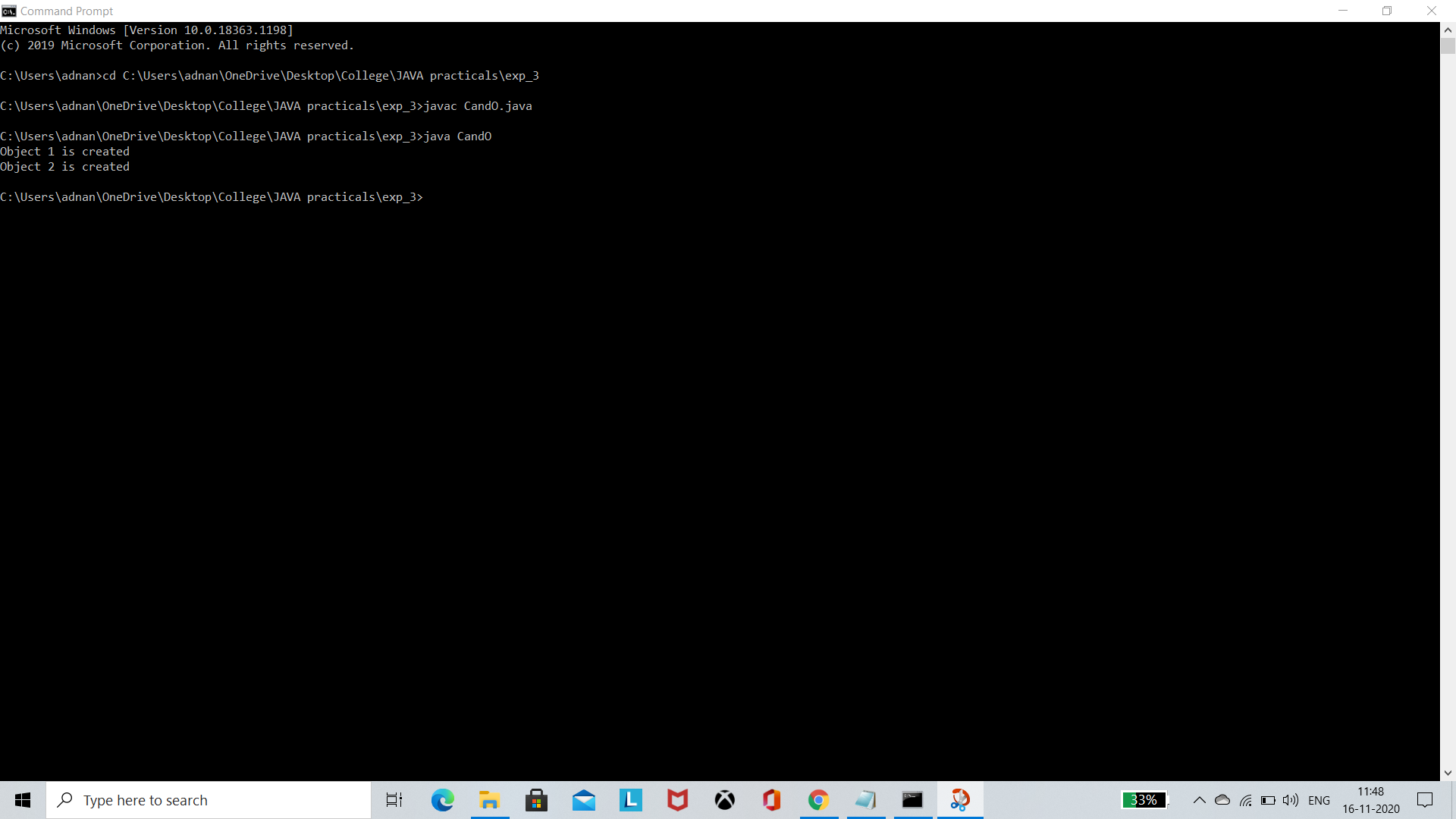
CandO myObj1 = new CandO(); // Object 1

CandO myObj2 = new CandO(); // Object 2

}

}

Output:



Conclusion: Hence, we learned how to create class and its objects in OOP Java

Experiment NO. 4

Aim: To understand the concept of method and constructor overloading in Object Oriented Programming (OOP) Java

Problem Statement: Create a class which have two overloaded methods one calculate area of square and other one calculate area of rectangle and this class should demonstrate the constructor overloading

Theory:

import java.util.Scanner;

public class Exp\_4{

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

double l,b,s;

System.out.print("Enter Rectangle length and breadth: ");

l = sc.nextDouble();

b = sc.nextDouble();

System.out.print("\n Enter side of square: ");

s = sc.nextDouble();

System.out.println();

Overload o1 = new Overload();

Overload o2 = new Overload("Constructor overloaded successfully ");

System.out.println("Area of square: "+ o1.Area(s)+"\n Area of recatangle: "+o1.Area(l,b));

}

}

class Overload{

public Overload(){

System.out.println("Empty overloaded Constructor");

}

public Overload(String v){

System.out.println("This overloaded constructor takes String \n your given string is "+v);

}

double Area(double l, double b){

return l\*b;

}

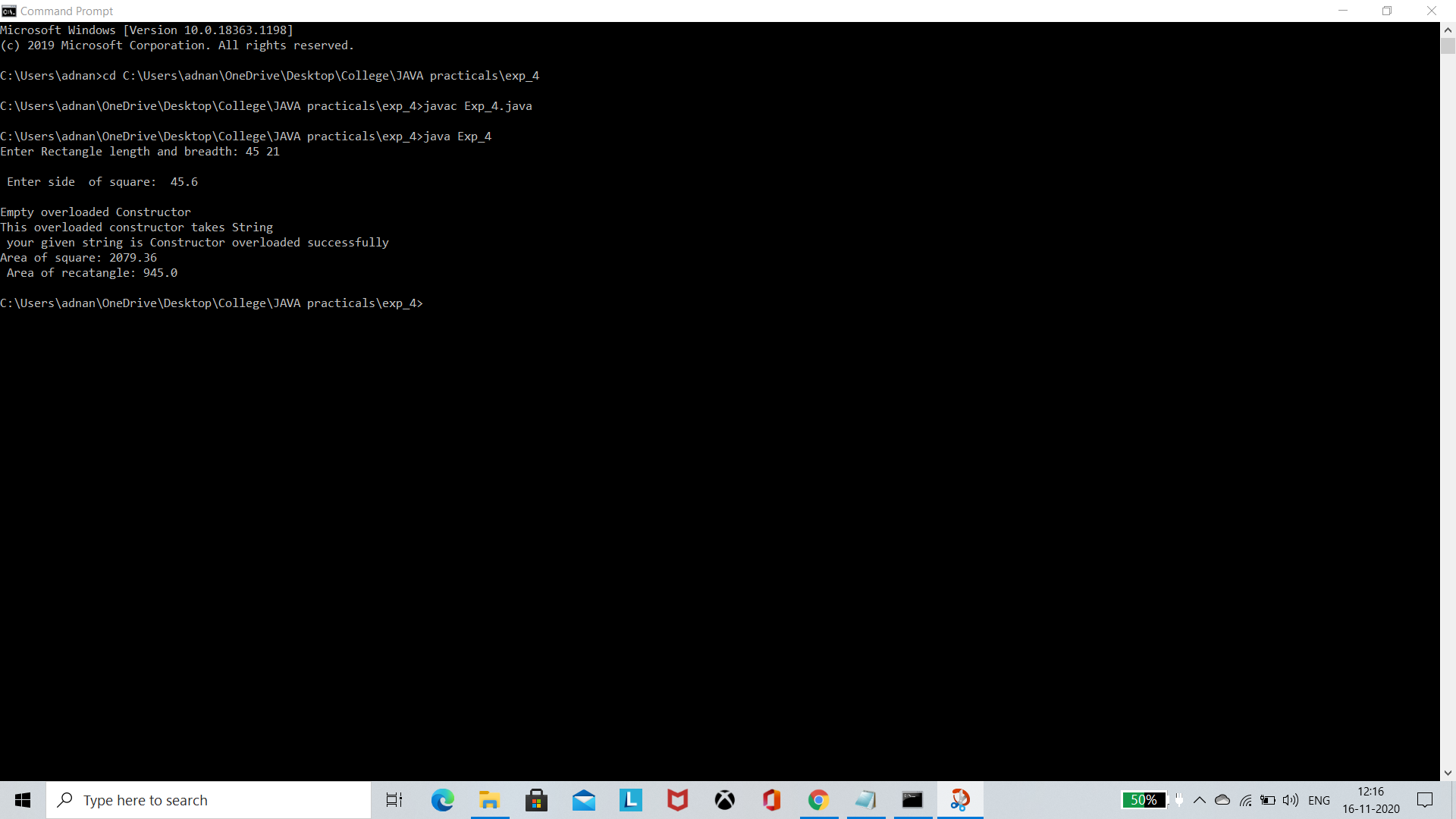
double Area(double s){

return s\*s;

}

}

Output:



Conclusion: Hence, we learned how to overload methods and constructors in OOP Java

Experiment NO. 5

Aim: To understand the concept of packages in Object Oriented Programming (OOP) Java

Problem Statement: Create a user defined package in which contain class A and create an object this class A in other class B which is located in another package.

Theory:

Package of class A:

package pack;

public class A{

public void msg(){System.out.println("Hello");}

}

Class B:

package mypack;

import pack.A;

class B{

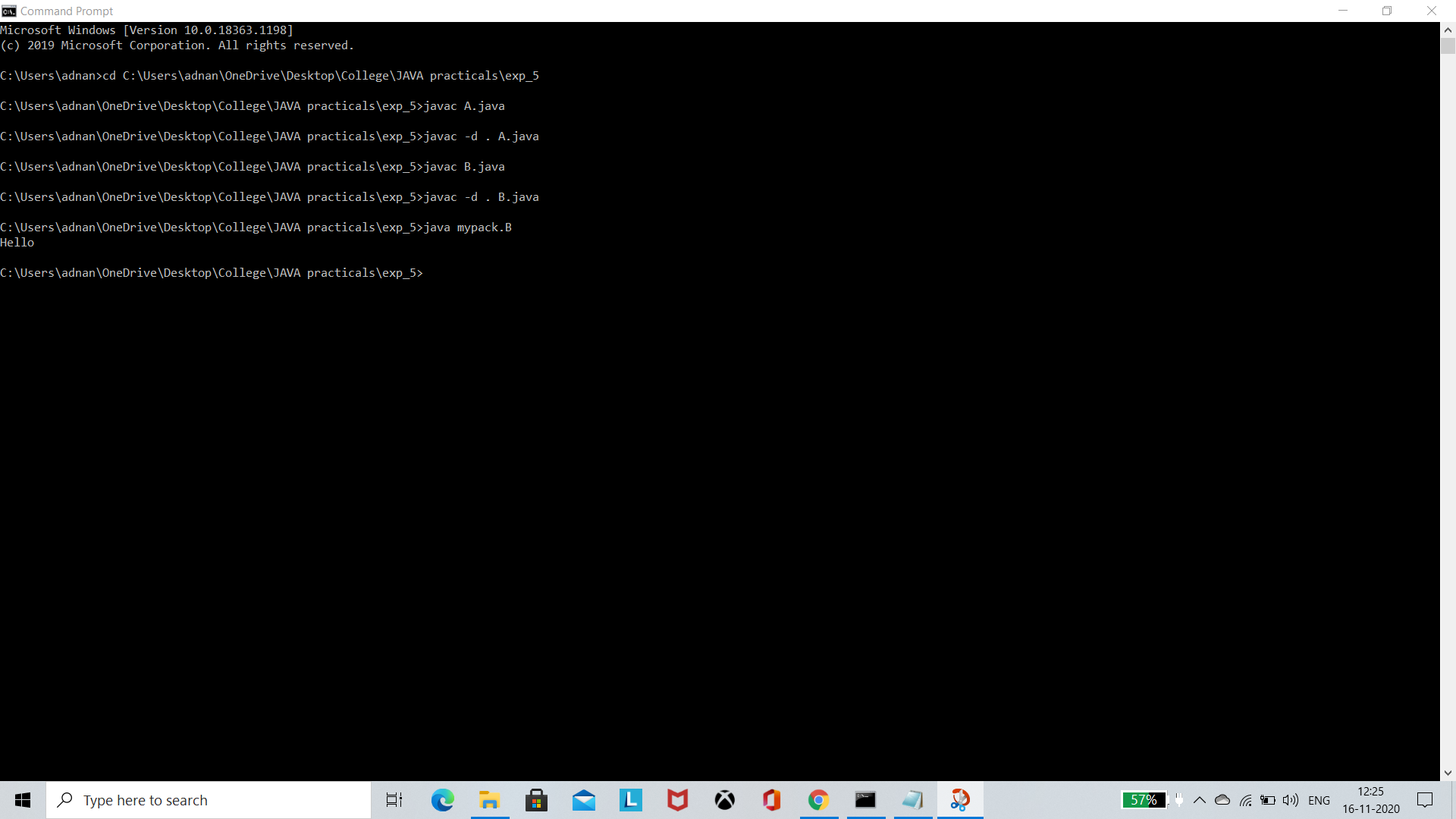
public static void main(String args[]){

A obj = new A();

obj.msg();

}

}

Output: 

Conclusion: Hence, we learned how to create packages and use it in OOP Java

Experiment NO. 6

Aim: To understand the concept of 2D arrays and String functions in Object Oriented Programming (OOP) Java

Problem Statement: Create a program to calculate the addition of 3x3 matrices and demonstrate the use of String functions

Theory:

import java.util.Scanner;

public class Exp\_6{

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

int z;

do{

System.out.println("Enter : 1) for matrix addition 2) for String Operations 3) To exit ");

z = sc.nextInt();

switch(z){

case 1: Matrix m = new Matrix();

m.op();

break;

case 2: St s = new St();

s.op();

break;

case 3: break;

default : System.out.println("You Entered Wrong key please try again");

}

}while(z!=3);

}

}

class Matrix{

void op(){

Scanner sc = new Scanner(System.in);

int[][] arr\_1 = new int[3][3];

int[][] arr\_2 = new int[3][3];

System.out.println("Enter your first 3x3 Matrix \n");

for(int i=0; i<3;i++){

for(int j=0;j<3;j++){

arr\_1[i][j] = sc.nextInt();

}

}

System.out.println("Enter your second 3x3 Matrix \n");

for(int i=0; i<3;i++){

for(int j=0;j<3;j++){

arr\_2[i][j] = sc.nextInt();

}

}

System.out.println("Your Matrix addition: ");

for(int i=0;i<3;i++){

for(int j=0;j<3;j++){

System.out.print(arr\_1[i][j]+arr\_2[i][j]+"\t");

}

System.out.println();

}

}

}

class St{

void op(){

Scanner sc = new Scanner(System.in);

String s = "";

System.out.println("Checking if string is empty or not ");

if(s.isEmpty())

System.out.println("String is empty");

else

System.out.println("String is not empty");

System.out.print("Enter your String: ");

s = sc.nextLine();

System.out.println("\n your String: "+s);

System.out.println("Checking again if string is empty or not ");

if(s.isEmpty())

System.out.println("String is empty");

else

System.out.println("String is not empty");

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

System.out.println("Character at "+i+" Position is :"+s.charAt(i));

String b;

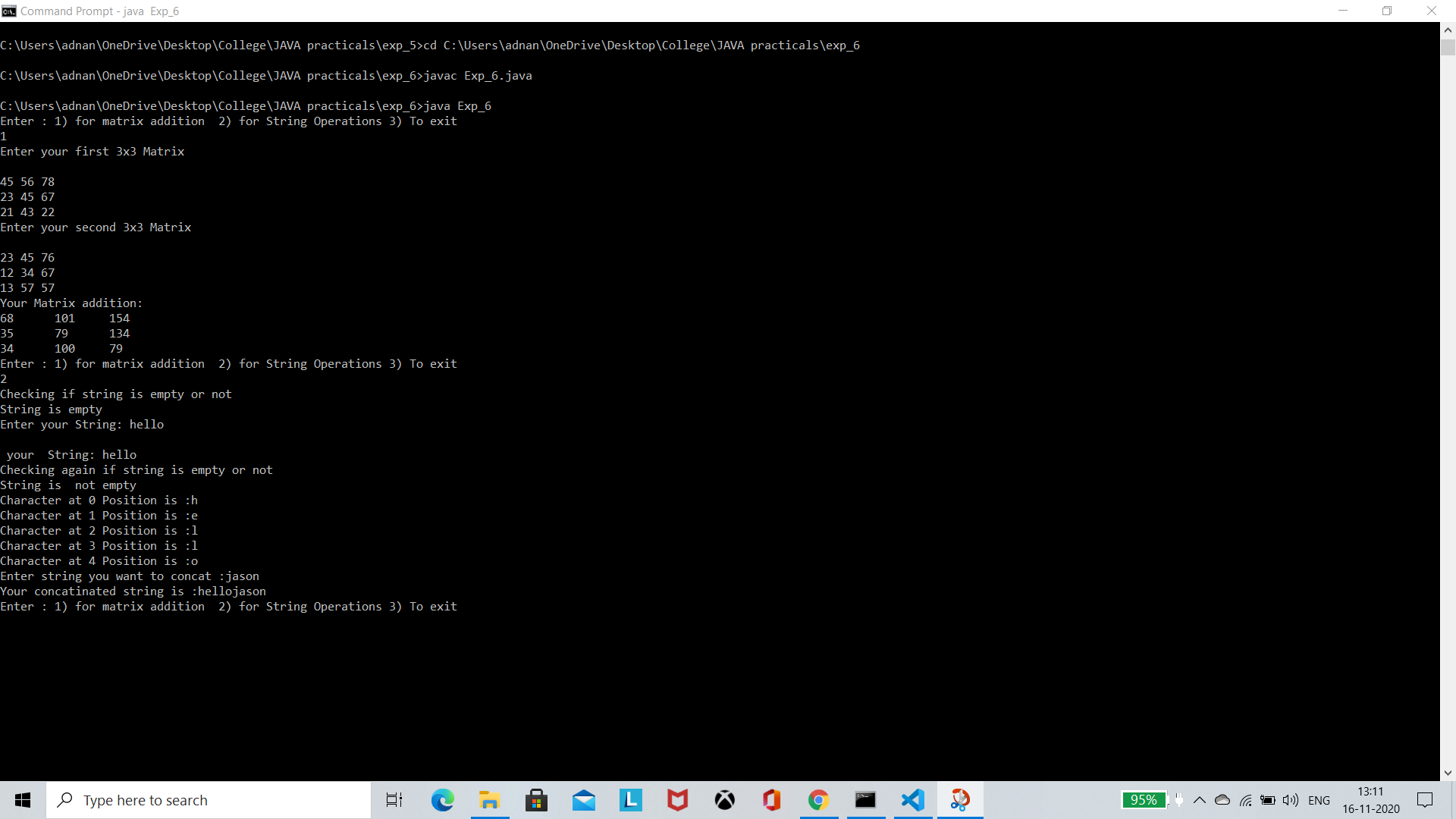
System.out.print("Enter string you want to concat :");

b = sc.nextLine();

System.out.println("Your concatinated string is :"+ s.concat(b));

}

}

Output:

Conclusion: Hence, we learned the concept of 2D array and String functions in OOP Java

Experiment NO. 7

Aim: To understand the concept of String Buffer and Vector in Object Oriented Programming (OOP) Java

Problem Statement: Create a program to demonstrate the use of String Buffer and Vector

Theory:

import java.util.Scanner;

import java.util.Vector;

public class Exp\_7 {

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

int z;

do{

System.out.println("Enter : 1) for String Buffer 2) for Vectors 3) To exit ");

z = sc.nextInt();

switch(z){

case 1: SB s = new SB();

s.op();

break;

case 2: VC v = new VC();

v.op();

break;

case 3: break;

default : System.out.println("You Entered Wrong key please try again");

}

}while(z!=3);

}

}

class SB{

void op(){

StringBuffer sb = new StringBuffer("study");

System.out.println("Before appending: "+sb);

sb.append("tonight");

System.out.println("After Appending: "+sb);

System.out.println("Now passing int variable to append");

StringBuffer str = new StringBuffer("test");

System.out.println("Before appending: "+str);

str.append(123);

System.out.println("After Appending:"+str);

System.out.println("Inserting variable in between the string: ");

str.insert(4, "code");

System.out.println("After inserting in middle:"+str);

System.out.println("Reversing the string ");

str.reverse();

System.out.println("After reversing: "+str);

StringBuffer st = new StringBuffer("Hello Jason");

System.out.println("Replacing Substring from the string: ");

System.out.println("Before replacing: "+st);

st.replace(6, 11, "Java");

System.out.println("After replacing: "+st);

System.out.println("Initial Capacity: "+st.capacity());

st.ensureCapacity(30);

System.out.println( "After Increasing the capacity: "+st.capacity());

}

}

class VC{

void op(){

Vector<String> vec = new Vector<String>(4);

vec.add("Tiger");

vec.add("Lion");

vec.add("Dog");

vec.add("Elephant");

System.out.println("Size is: "+vec.size());

System.out.println("Default capacity is: "+vec.capacity());

System.out.println("Vector element is: "+vec);

vec.addElement("Rat");

vec.addElement("Cat");

vec.addElement("Deer");

System.out.println("Size after addition: "+vec.size());

System.out.println("Capacity after addition is: "+vec.capacity());

System.out.println("Elements are: "+vec);

if(vec.contains("Tiger"))

{

System.out.println("Tiger is present at the index " +vec.indexOf("Tiger"));

}

else

{

System.out.println("Tiger is not present in the list.");

}

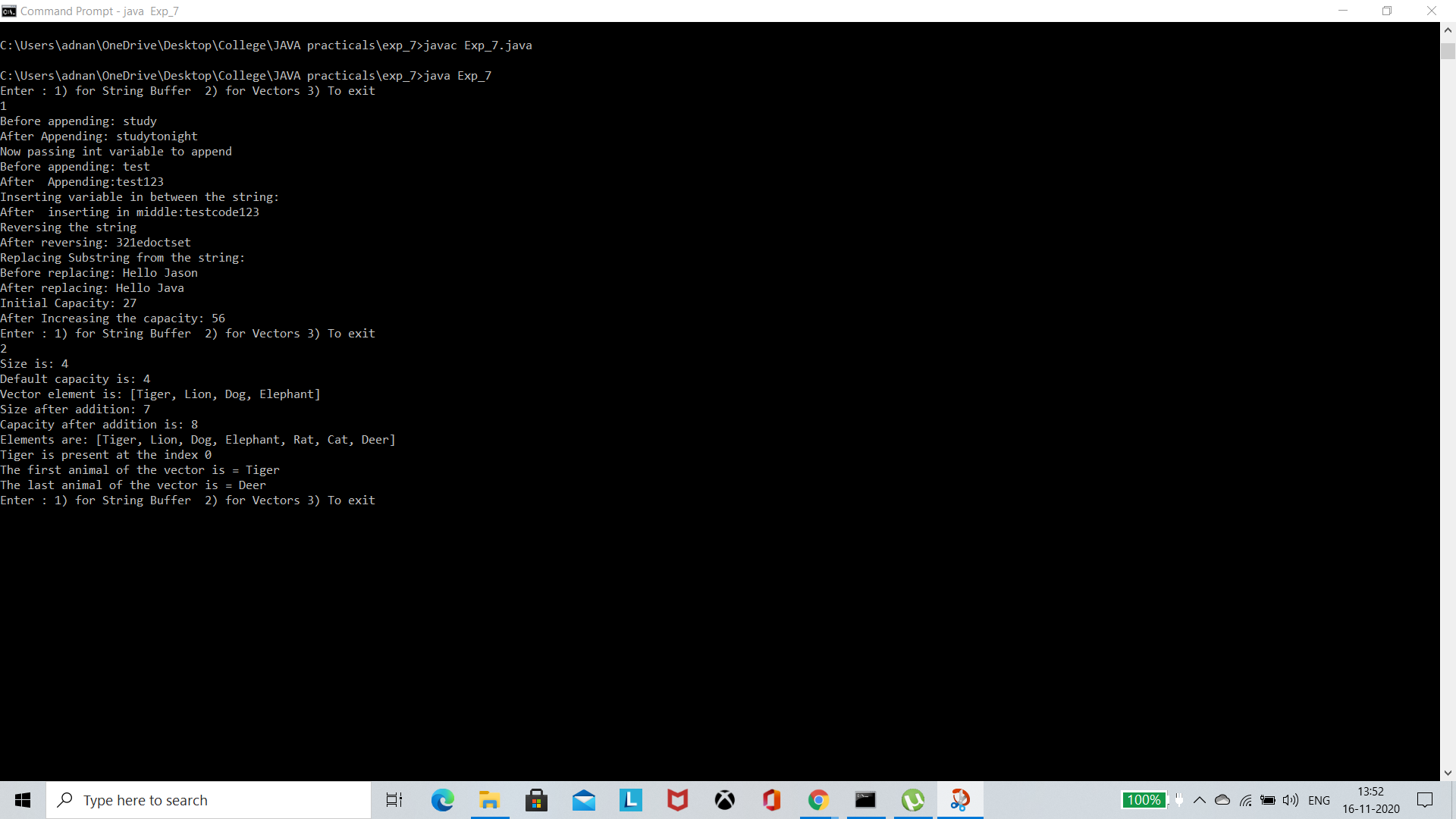
System.out.println("The first animal of the vector is = "+vec.firstElement());

System.out.println("The last animal of the vector is = "+vec.lastElement());

}

}

Output:



Conclusion: Hence, we learned the concept of String Buffer and Vector in OOP Java

Experiment NO. 8

Aim: To understand the concept of types of inheritance in Object Oriented Programming (OOP) Java

Problem Statement: Create a program to demonstrate the concept of inheritance

Theory:

import java.util.\*;

public class Exp\_8 {

public static void main(String[] args) {

SingleH s = new SingleH();

s.display();

MultipleH m = new MultipleH();

m.display();

SuperClass h = new HierarchicalH();

h.display();

}

}

class SuperClass{

void display(){

System.out.println("Super class");

}

}

class SingleH extends SuperClass{

@Override

void display(){

System.out.println("You're in SingleH class");

System.out.println("Calling super class in subclass");

super.display();

}

}

class MultipleH extends SingleH{

@Override

void display(){

System.out.println("You're in MultipleH class");

System.out.println("calling previous super class which will call root class");

super.display();

}

}

class HierarchicalH extends SuperClass{

void display(){

System.out.println("You're in HierarchicalH class which extends SuperClass (SuperClass is extended by 2 class viz SingleH and HierarchicalH which is an example of Hierarchical inheritance )");

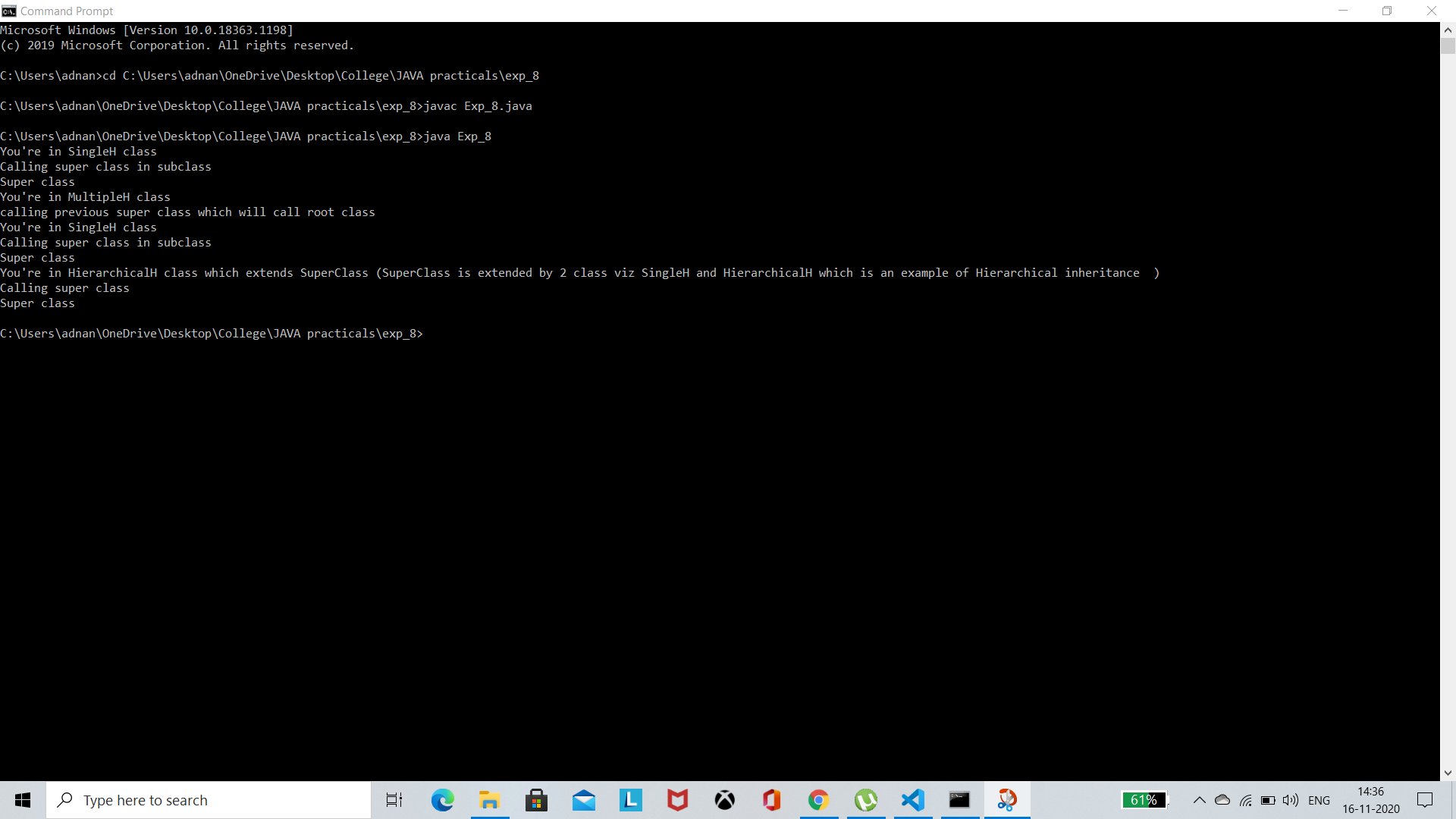
System.out.println("Calling super class ");

super.display();

}

}

Output:



Conclusion: Hence, we learned the concept of Inheritance and types of inheritance in OOP Java

Experiment NO. 9

Aim: To understand the concept of Multiple inheritance in Object Oriented Programming (OOP) Java

Problem Statement: Create a program to demonstrate Multiple inheritance using Interfaces

Theory:

public class Exp\_9  {

    public static void main(String[] args) {

        Aniko a = new Aniko("JoJo's the bizarre adventure", "Action, Adventure and Comedy", 2012, 24);

        a.display();

        }

    }

 interface Info{

    public void genre(String g);

    public void release\_date(int r);

    public void episodes(int e);

}

 interface Name{

    public void name(String n);

}

interface Anime extends Info,Name{

 public void display();

}

class Aniko implements Anime{

    private String genr,name\_1;

    private int releasedate,episode;

    public void genre(String genr){

        this.genr = genr;

    }

    public void name(String name\_1){

        this.name\_1 = name\_1;

    }

    public void episodes(int episode){

        this.episode = episode;

    }

    public void release\_date(int r){

        this.releasedate = r;

    }

    public Aniko(String name,String genre,int release\_date, int episodes){

        genre(genre);

        name(name);

        release\_date(release\_date);

        episodes(episodes);

    }

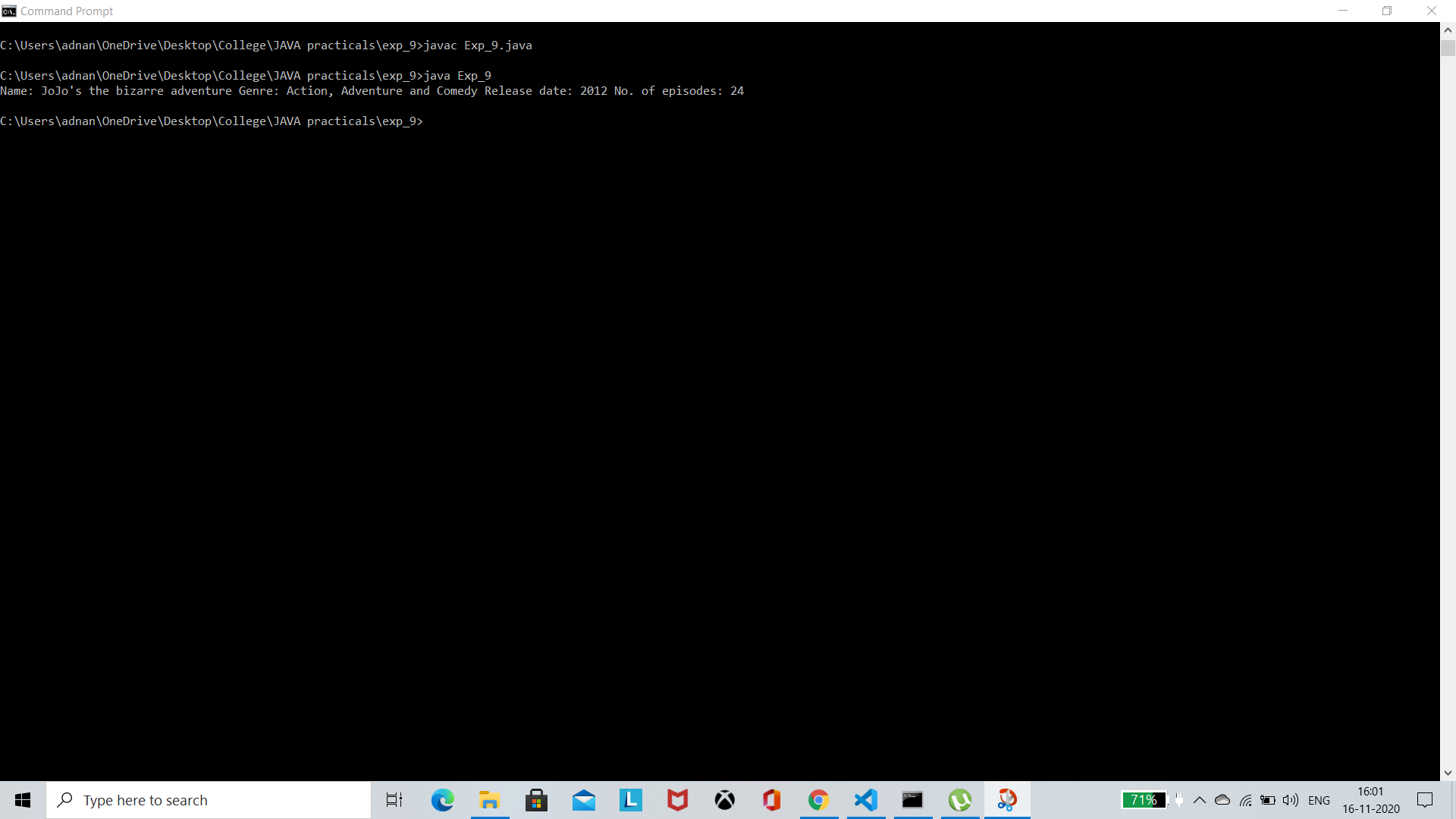
    public void display(){

        System.out.println("Name: "+this.name\_1+" Genre: "+this.genr+" Release date: "+this.releasedate+" No. of episodes: "+this.episode);

    }

}

Output:



Conclusion: Hence, we learned the concept of Multiple Inheritance using Interfaces in OOP Java

Experiment NO. 10

Aim: To understand the concept of abstract class and abstract methods in Object Oriented Programming (OOP) Java

Problem Statement: Create a program which have one abstract class and this abstract class should have at least one abstract method

Theory:

public static void main(String[] args) {

Student s = new Info("Jonathan Joester", 68, "11/B", 72323);

s.display();

}

}

abstract class Student{

private String name;

private int roll\_no;

private String grade;

public Student(String name,int roll\_no,String grade){

this.name = name;

this.roll\_no = roll\_no;

this.grade = grade;

}

public abstract void display();

public void show(){

System.out.println("Student name: "+this.name+"\n Roll no: "+this.roll\_no+"\n Standard: "+this.grade);

}

}

class Info extends Student{

int gr\_no;

public Info(String name,int roll\_no,String grade,int gr\_no){

super(name, roll\_no,grade);

this.gr\_no = gr\_no;

}

public void display(){

System.out.println("Your entered information is : ");

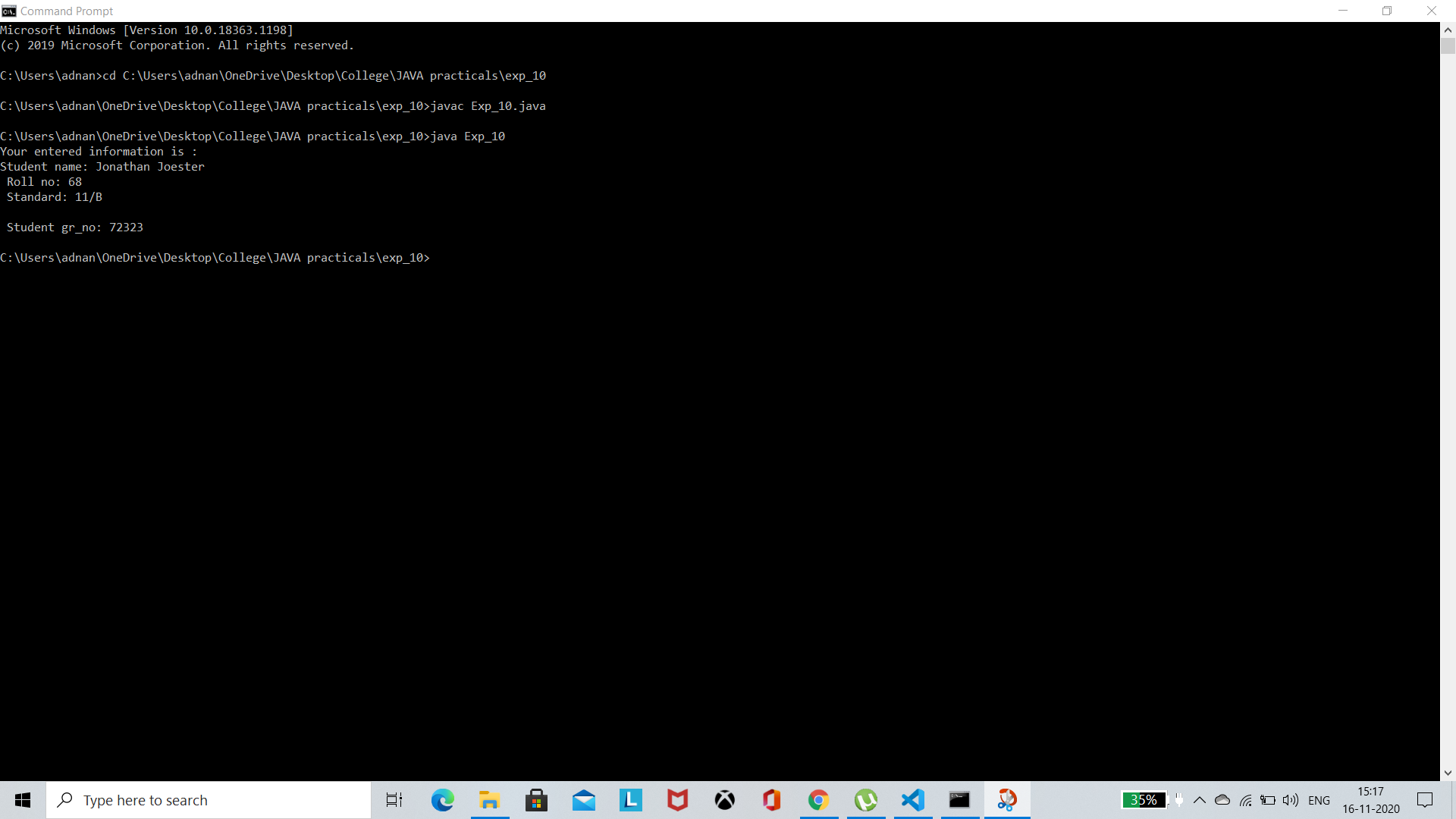
super.show();

System.out.println("\n Student gr\_no: "+this.gr\_no);

}

}

Output:



Conclusion: Hence, we learned the concept of abstract class and methods in OOP Java

Experiment NO. 11

Aim: To understand the concept of Super and Final keywords in Object Oriented Programming (OOP) Java

Problem Statement: Create a program which uses Super and final keywords

Theory:

public class Exp\_11 {

public static void main(String[] args) {

SuperClass h = new SingleH();

h.display();

NoHerit n = new NoHerit();

n.display();

}

}

class SuperClass{

void display(){

System.out.println("Super class");

}

}

class SingleH extends SuperClass{

void display(){

System.out.println("You're in SingleH class");

System.out.println("Calling super class in subclass");

super.display();

}

}

final class NoHerit{

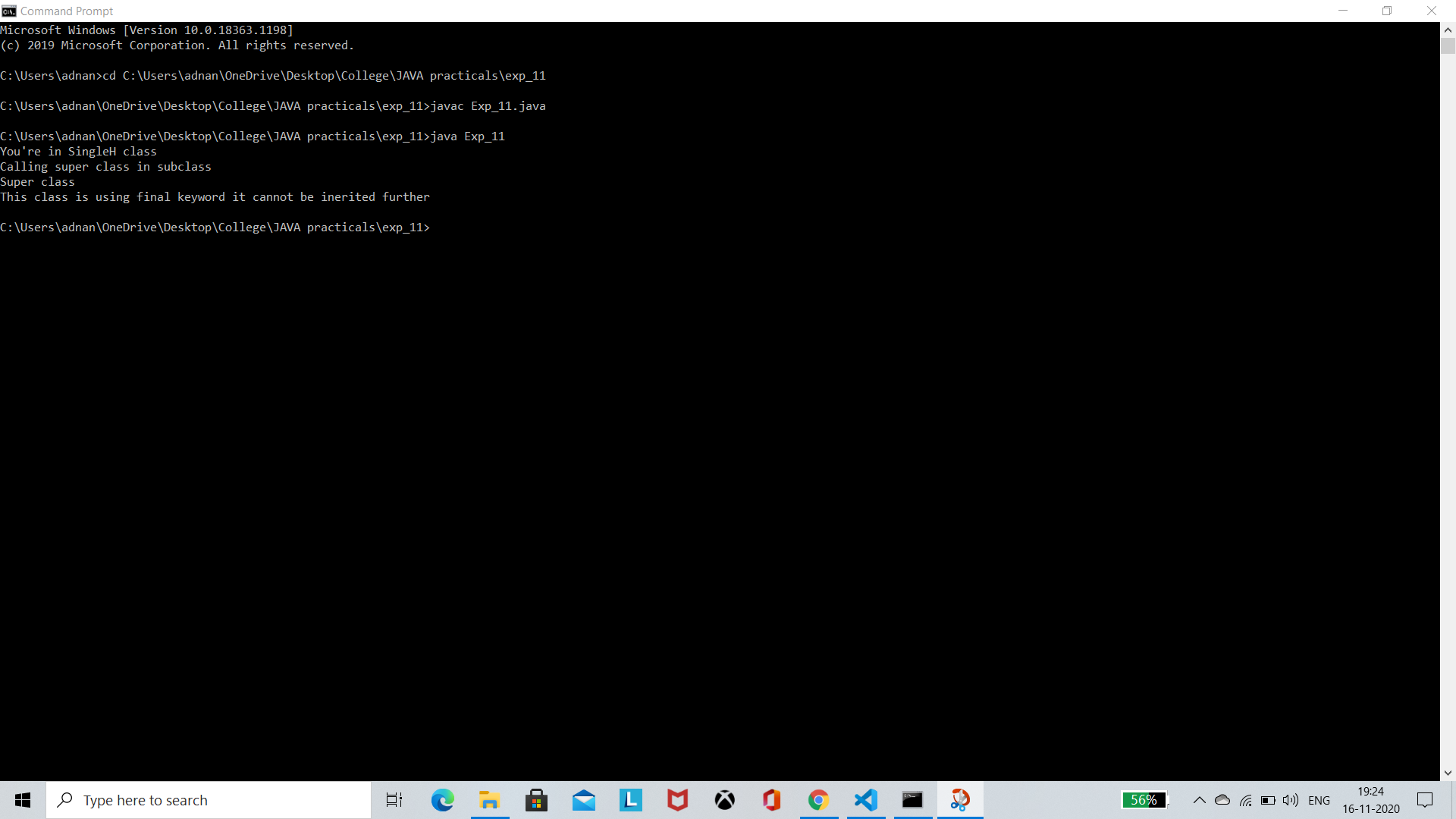
void display(){

System.out.println("This class is using final keyword it cannot be inerited further ");

}

}

Output:



Conclusion: Hence, we learned the concept of Super and Final keywords and how to use them OOP Java

Experiment NO. 12

Aim: To understand the concept of Exceptional Handling in Object Oriented Programming (OOP) Java

Problem Statement: Create a class which throw at least two exceptions and handle it using try and catch keywords

Theory:

public class Exp\_12 {

public static void main(String[] args) {

try

{

int arr[]= {15,53,25,78};

System.out.println(arr[8]);

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println(e);

}

try

{

int data=50/0;

}

catch(ArithmeticException e)

{

System.out.println(e);

}

finally{

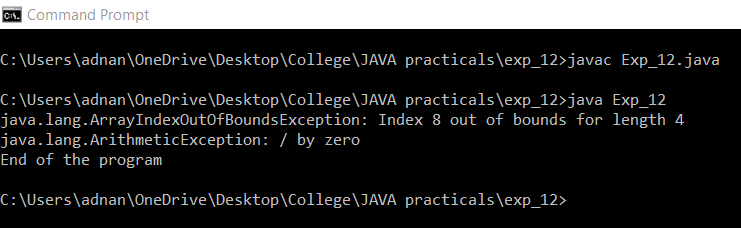
System.out.println("End of the program");

}

}

}

Output:



Conclusion: Hence, we learned the concept of Exceptional handling in OOP Java

Experiment NO. 13

Aim: To understand the concept of user defined exception in Object Oriented Programming (OOP) Java

Problem Statement: Create a class which throw user defined exception and handle it using try and catch block

Theory:

class InvalidProductException extends Exception

{

public InvalidProductException(String s)

{

super(s);

}

}

public class Exp\_13

{

void productCheck(int weight) throws InvalidProductException{

if(weight<100){

throw new InvalidProductException("Product Invalid");

}

}

public static void main(String args[])

{

Exp\_13 obj = new Exp\_13();

try

{

obj.productCheck(60);

}

catch (InvalidProductException ex)

{

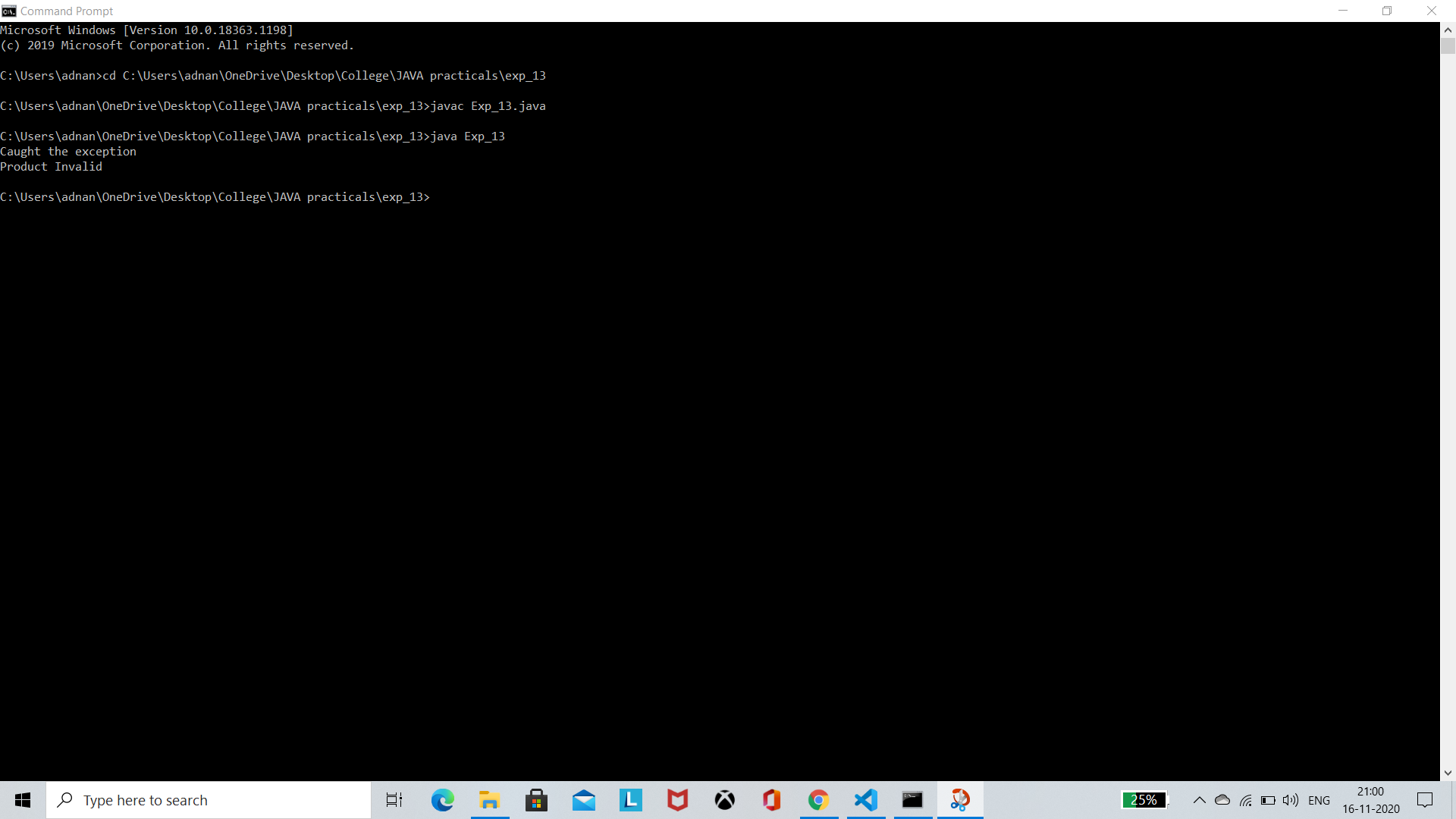
System.out.println("Caught the exception");

System.out.println(ex.getMessage());

}

}

}

Output: 

Conclusion: Hence, we learned the concept of user defined exception in OOP Java

Experiment NO. 14

Aim: To understand the concept of Multithreading in Object Oriented Programming (OOP) Java

Problem Statement: Create a class which inherits Thread class and use it to demonstrate the concept of Multithreading

Theory:

class Count extends Thread

{

Count()

{

super("my extending thread");

System.out.println("my thread created" + this);

start();

}

public void run()

{

try

{

for (int i=0 ;i<10;i++)

{

System.out.println("Printing the count " + i);

Thread.sleep(1000);

}

}

catch(InterruptedException e)

{

System.out.println("my thread interrupted");

}

System.out.println("My thread run is over" );

}

}

class Exp\_14

{

public static void main(String args[])

{

Count cnt = new Count();

try

{

while(cnt.isAlive())

{

System.out.println("Main thread will be alive till the child thread is live");

Thread.sleep(1500);

}

}

catch(InterruptedException e)

{

System.out.println("Main thread interrupted");

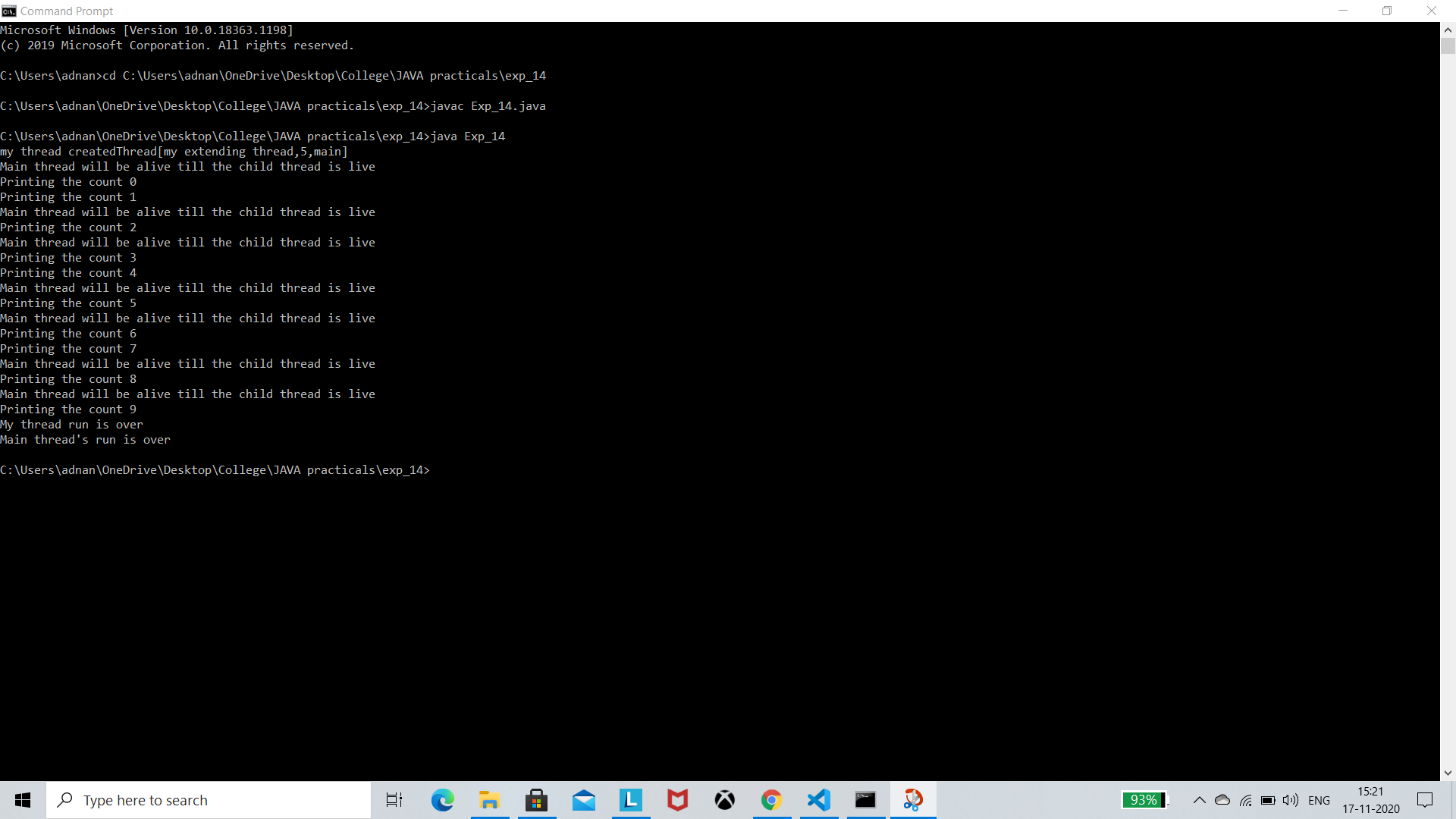
}

System.out.println("Main thread's run is over" );

}

}

Output:



Conclusion: Hence, we learned the concept of Multithreading in OOP Java

Experiment NO. 15

Aim: To understand the concept of Graphics class in Object Oriented Programming (OOP) Java

Problem Statement: Write a program in java using AWT and Applet class.

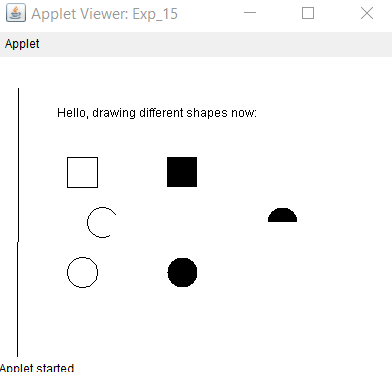
Theory:

Program 1:

import java.applet.Applet;  
import java.awt.\*;  
public class Exp\_15 extends Applet  
{  
 public void paint(Graphics g)  
 {  
 g.setColor(Color.black);  
 g.drawString("Hello, drawing different shapes now:",60, 60);  
 g.setColor(Color.black);  
 g.fillOval(170,200,30,30);  
 g.drawArc(90,150,30,30,30,270);  
 g.fillArc(270,150,30,30,0,180);  
 g.drawLine(21,31,20,300);  
 g.drawRect(70,100,30,30);  
 g.fillRect(170,100,30,30);  
 g.drawOval(70,200,30,30);  
 }  
}

**Html Body:** <applet code="Exp\_15.class" width="300" height="300">

**output**:



Program 2:

import java.awt.\*;

import java.applet.\*;

import java.awt.event.\*;

public class student extends Frame implements ActionListener

{String msg;

Button b1=new Button("save");

Label l11=new Label("Student details",Label.CENTER);

Label l1=new Label("Name:",Label.LEFT);

Label l2=new Label("age:",Label.LEFT);

Label l3=new Label("Sex(M/F):",Label.LEFT);

Label l4=new Label("Address:",Label.LEFT);

Label l5=new Label("Course:",Label.LEFT);

Label l6=new Label("Semester:",Label.LEFT);

Label l7=new Label("",Label.RIGHT);

TextField t1=new TextField();

Choice c1=new Choice();

CheckboxGroup cbg=new CheckboxGroup();

Checkbox ck1=new Checkbox("Male",false,cbg);

Checkbox ck2=new Checkbox("Female",false,cbg);

TextArea t2=new TextArea("",180,90,TextArea.SCROLLBARS\_VERTICAL\_ONLY);

Choice course=new Choice();

Choice sem=new Choice();

Choice age=new Choice();

public student()

{addWindowListener(new myWindowAdapter());

setBackground(Color.white);

setForeground(Color.black);

setLayout(null);

add(l11);

add(l1);

add(l2);

add(l3);

add(l4);

add(l5);

add(l6);

add(l7);

add(t1);

add(t2);

add(ck1);

add(ck2);

add(course);

add(sem);

add(age);

add(b1);

b1.addActionListener(this);

add(b1);

course.add("BE CSE");

course.add("BE IT");

course.add("BE Mechanical");

course.add("BE EXTC");

course.add("BE Instrumental ");

sem.add("1");

sem.add("2");

sem.add("3");

sem.add("4");

sem.add("5");

sem.add("6");

age.add("17");

age.add("18");

age.add("19");

age.add("20");

age.add("21");

l1.setBounds(25,65,90,20);

l2.setBounds(25,90,90,20);

l3.setBounds(25,120,90,20);

l4.setBounds(25,185,90,20);

l5.setBounds(25,260,90,20);

l6.setBounds(25,290,90,20);

l7.setBounds(25,260,90,20);

l11.setBounds(10,40,280,20);

t1.setBounds(120,65,170,20);

t2.setBounds(120,185,170,60);

ck1.setBounds(120,120,50,20);

ck2.setBounds(170,120,60,20);

course.setBounds(120,260,100,20);

sem.setBounds(120,290,50,20);

age.setBounds(120,90,50,20);

b1.setBounds(120,350,50,30);

}

public void paint(Graphics g)

{g.drawString(msg,200,450);}

public void actionPerformed(ActionEvent ae)

{if(ae.getActionCommand().equals("save"))

{msg="Student details saved!";

setForeground(Color.red); }

}

public static void main(String g[])

{student stu=new student();

stu.setSize(new Dimension(500,500));

stu.setTitle("student registration");

stu.setVisible(true);

}

}

class myWindowAdapter extends WindowAdapter

{public void windowClosing(WindowEvent we)

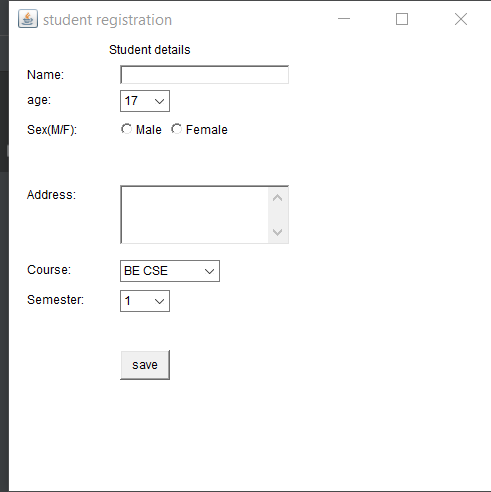
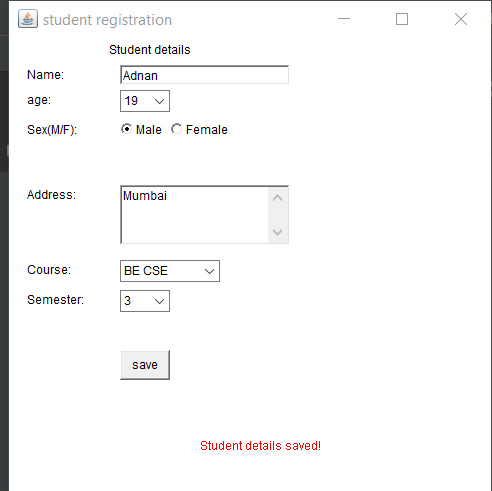
{

System.exit(0);

}

}

Output:

Conclusion: Hence, we learned the concept of Graphics class in OOP Java