Practical No. 1

Aim:- Create debug and run java programs based on constant, variable and operators.

Theory:- Java is Object-Oriented Programming language that supports java without need for recomplication.

i Constant - A constant is a variable whose value cannot changes once. Java doesn't hava built-in support for constants. A constant can make our ~~pro~~ programme more easily read and understand by others.

ii Variable - A variable provide us with named entger that our programms can manipulate each variable. Java has a specific type which determine the sise and layout of the variable. ~~among~~ ~~there~~ There are 3 Types of variable : i) local variables
ii) Instance variables.
iii) Static variables.

iii Operators - Java provides a rich set of operators to manipulate variable.

i) Arithmatic Operator

These operator are used to execute mathematical expressions

e.x. + = Addition
- = subtraction
* = multiplication
%/ = devision.

ii) Relational Operator

i. == (equal to) : to check values of operands are equal.

ii. != (not equal to) : to check values of operands are not equals

iii. > (greater than) : to check the left operand is greater than right operand.

iv. < (less than) : to check the left operand is smaller that right operand.

iii) Bitwise Operator

This operator are used in integer types i.e. long, short, char and bytes.

e.x. & - Bitwise and

~ - Bitwise compliment

^ - Bitwise XOR

>> - Bitwise sift

iv) Logical Operator.

Assume Boolean variable A holds true and variable B holds false then

i. && (logical and) : if both are not true equal, condition is true.

ii. || (logical or) : if any of two are not zero, condition is true

Program :-

```
import java.util.Scanner;
class practical1 {
    public static void main(String[] args) {
        int r, s, l, b;
        double pi = 3.14, circle, square, rect;
        Scanner c = new Scanner(System.in);
        System.out.println("Enter radius of circle : ");
        r = c.nextInt();
        circle = pi * r * r;
        System.out.println("Area of Circle : " + circle);
        System.out.println("Enter side of square : ");
        s = c.nextInt();
        square = s*s;
        System.out.println("Area of Square : " + square);
        System.out.println("Enter Length and Breadth of
                                Rectangle : ");
        l = c.nextInt();
        b = c.nextInt();
        rect = l*b;
        System.out.println("Area of Rectangle : " + rect);
    }
}
```

Conclusion :- we have successfully created, debugged and run java program based on constant, variable and operators.

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical1.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical1
Enter radius of circle:5
Area of circle:78.5
Enter Side of Square:5
Area of Square:25.0
Enter Length and Breadth of Rectangle:10
10
Area of Rectangle:100.0
```

Practical No. 2

Aim:- create, debug and run java programs based on decision making and branching.

Theory:- Java uses a control statement to control the flow of execution of programmed based on certain decision.

i) if - else - This if statement alone tells us that if a condition is true, it will execute a block of statement and otherwise the else statement can be executed.

syntax:-

```
if (condition) {
    // executable
        code
}
else {
    // executable
        code
}
```

ii) if - else - if - ladder :- a user can decide among multiple options. The if statement are executed from the top down as one of the conditions is true. If the statement associated with that if is executed, then the rest of the ladder will bypassed, if none of the condition is true, then the final else statement will be executed.

Camlin Page
Date / /

Syntax :
```
if (condition1)
    statement1;
else if (condition2)
    statement2;
else if (condition3)
    statement3;
else
    final statement;
```

iii) Switch Case :- The switch statement is a multi-branch statement. It provides an easeway to dispatch execution to different parts of code based on value of the expression.

Syntax :-
```
switch (expression) {
    case 1 :
        statement1;
        break;
    case 2 :
        statement2;
        break;
    default :
        default statement;
}
```

Camlin Page
Date / /

Programs :-

1st program

```
import java.util.Scanner;
class practical2 {
    public static void main(String[] args){
        int n;
        Scanner s = new Scanner(System.in);
        System.out.println("Enter a No. :");
        n = s.nextInt();
        if(n%2==0){
            System.out.println("Given number is even.");
        else
            System.out.println("Given nomber is Odd.");
        }
}
```

2nd program

```
import java.util.Scanner;
class practical2_2{
    public static void main(String[] args){
        float tmarks, pmarks=28F;
        char grade;
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the total Marks:");
        tmarks = s.nextfloat();
```

```
if (tmark >= pmarks) {
    if ( tmarks >90)
        grade = 'A';
    else if ( tmarks > 75)
        grade = 'B';
    else if ( tmarks > 60)
        grade = 'c';
    else
        grade = 'D';

    System.out.println ("You passed the exam with"
                        + grade + ' grade");
    }
else {
    grade = 'F';
    System.out.println ("You failed the exam with"
                        + grade + " grade");
    }
}
}
```

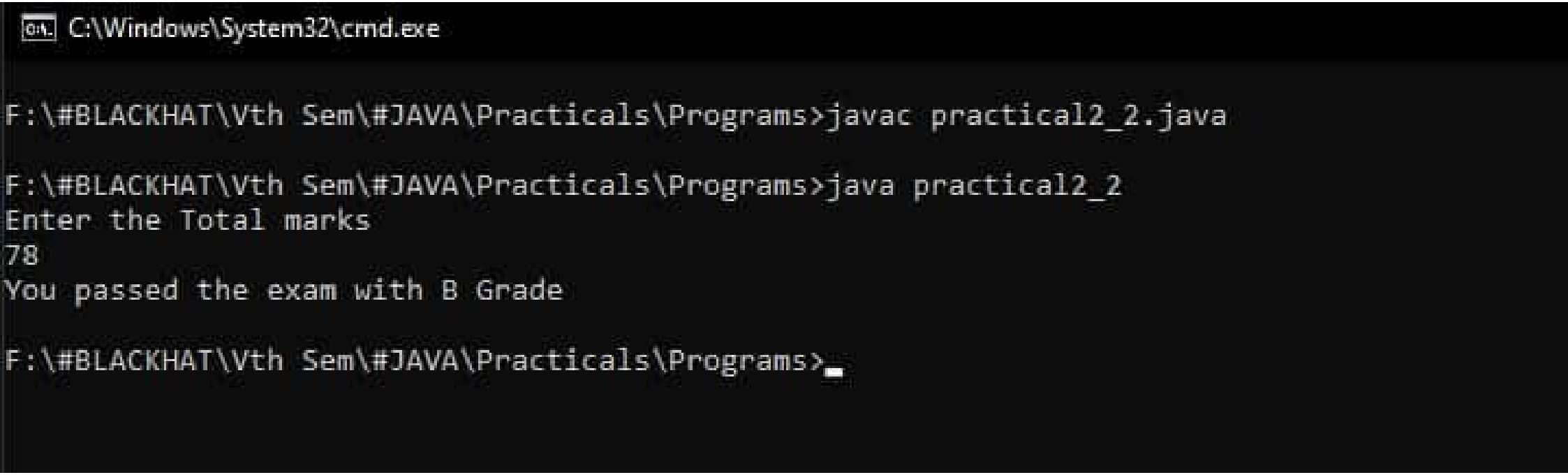
3rd program

```
class practical2_3 {
    public static void main (String [] args) {
        String levelname = " Beginner";
        int level = 0;
        switch (levelname) {
        case " Beginner" : level = 1;
        break;
```

```
            case "Intermediate" : level = 2;
            break;
            case "Expert" : level = 3;
            break;
            default : level = 0;
            break;
            }
        System.out.println("Your level is : " +level);
    }
}
```

Conclusion:- we have successfully performed, debug and run java program based on decision making and branching.

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical2
Enter a No. :
7
Given Number is Odd

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>
```



```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical2_3.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical2_3
Your Level is: 1

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>
```

Practical No. 3

Aim :- Create, debug and run java programs based on decision making and looping.

Theory :- Looping is a feature which facilitates the occution of set of instructions / functions repeatedly while some conditions evaluates to true.

i while loop :- A while loop is a control flow statement that allows code to be executed repeatedly based on a given boolean condition.

syntax :-

```
while (boolean condition)
{
    loop statement ;
}
```

ii for loop :- for loop provides a concise way of writing the loop structure. Unlike while loop, a for loop statement contains, initialization, condition and increament / decrement in one line.

syntax :-

```
for ( initialization ; condition ; increment/decrement)
{
    looping statement ;
}
```

iii Enhanced for :- Java also includes another version of for loop which is enhanced for loop

syntax :- for(I elements : collection obj/array) {
statements;
}

Programs :-

1st program

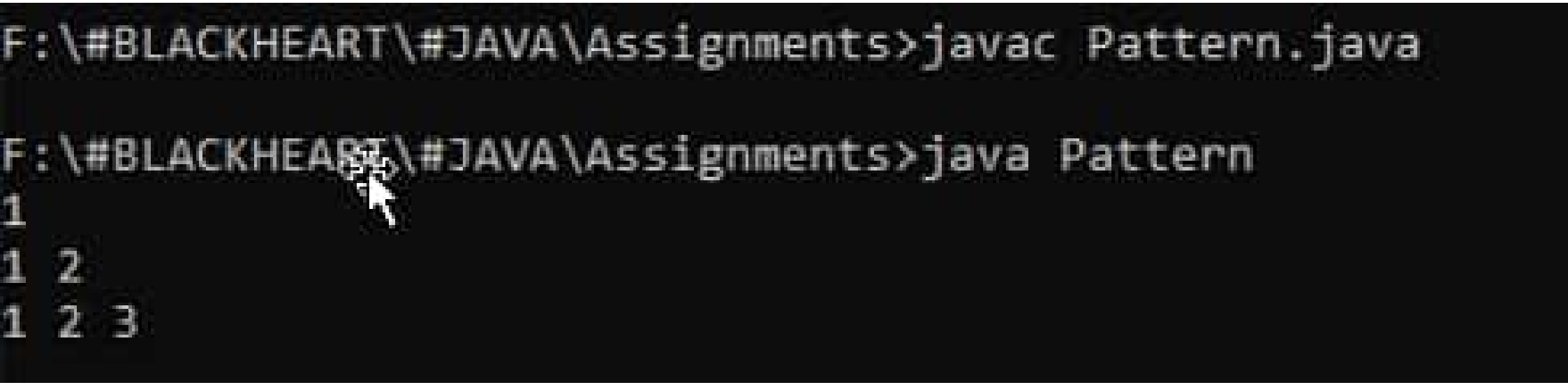
```
public class practical3 {
    public static void main(String[]args){
        for (int i = 1; i<=3; i++) {
            for (int j= 1; j<=i; j++){
                System.out.print(j+" ");
            }
            System.out.println();
        }
    }
}
```

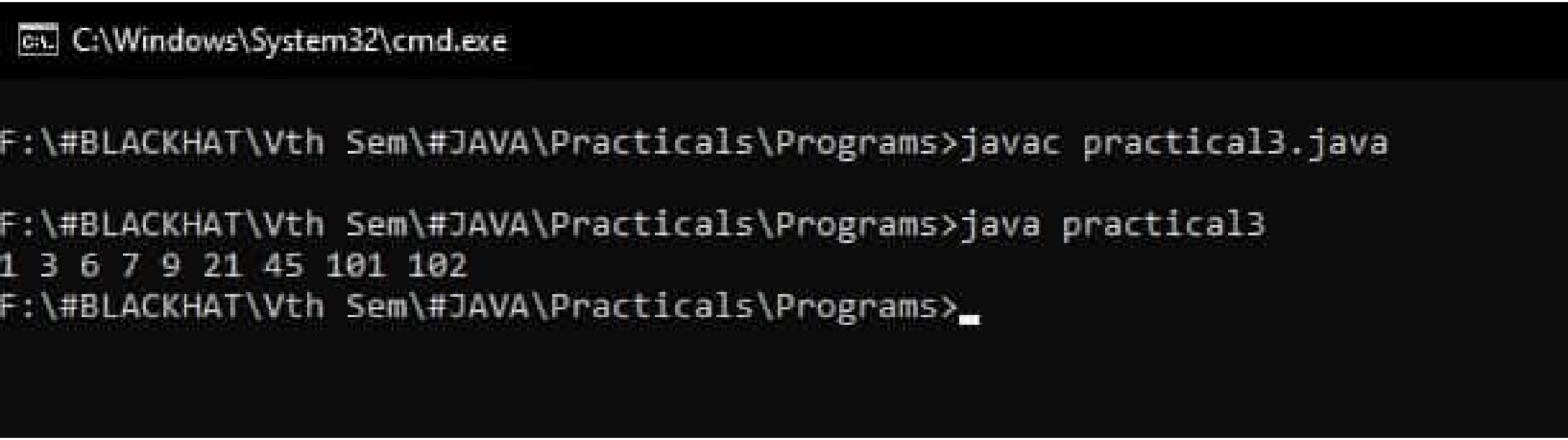
2nd program

```
class practical3_2 {
    public static void main (String[] args) {
    int temp;
    int [] arr = {1, 3, 7, 6, 45, 21, 9, 103, 102};
    for (int = 0; i<(arr.length-1); i++) {
        for (int j = i+1 ; j< arr.length ;j++){
            if ( arr[i] > arr[j]) {
                temp= arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
```

```
        for (int m = arr) {
            System.out.print(m+" ");
        }
    }
}
```

Conclusion:- We have successfully performed a java program based on decision making and looping.





Practical No. 4

Aim: create debug and run java programms based on string and stringbuffer

Theory: String in java are objects that are backed internally by a char array, since arrays are immutable. Strings is made an entity new string is created.

Syntax :-

<string_type> <string_name> = "<seq_of_string>";

There are two ways to create string in Java

i) String literal

- string s = "Blackheart";

ii) Using new keyword

- string s = new string ("Blackhat");

String Buffer - stringbuffer is a peer class of string that provides much of the functionality of strings It represents fixed-length immutable character sequences, while string buffer represents growable and writeable characters sequences.

Syntax :-

String Buffer s = new StringBuffer ("Blackhat");

Camlin Page
Date / /

Programs :-

1st program

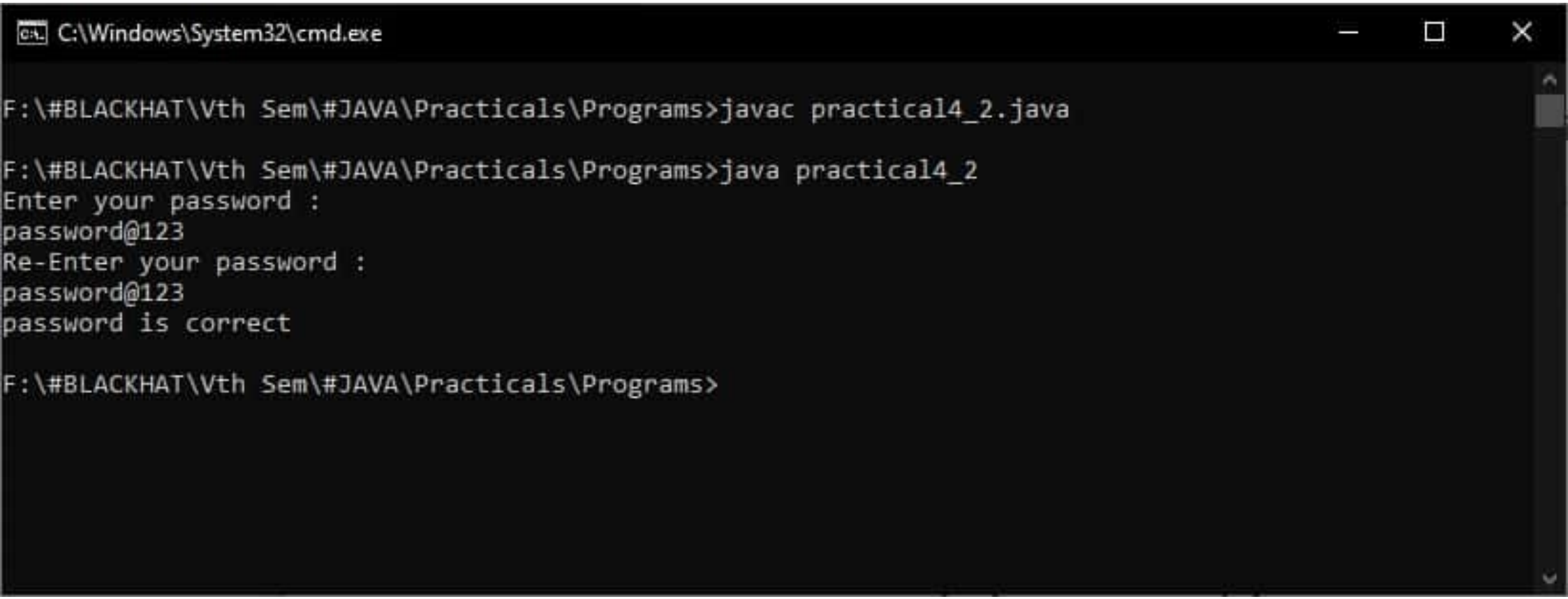
```
import java.util.Scanner
public class practical4{
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        System.out.println("Enter your password:");
        String s1 = s.nextLine();
        System.out.println("Re-Enter your password:");
        String s2 = s.nextLine();
        if (s1.equals(s2)){
            System.out.println("Password is correct");
        else
            System.out.println("Password is wronged");
    }
}
```

2nd program

```
import java.util.Scanner;

public class practical4_2{
    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        System.out.println("Enter a String:");
        String string1 = s.nextLine();
```

```
StringBuffer string2 = new StringBuffer(string1);
string2.reverse();
if (string1.equals(string2.toString()))
    System.out.println(" string1 + " is palindrome");
else
    System.out.println(string1 + " is not a
                        palindrome");
    }
}
```

Conclusion: we have successfully performed the java program based on strings and StringBuffer



```
F:\#BLACKHEART\#JAVA\Assignments>javac palandrome.java

F:\#BLACKHEART\#JAVA\Assignments>java palandrome
Enter a String :
radar
radar is palandrome
```

Practical No. 5

Aim :- Create, debug and run java program based on wrapper class or vector

Theory :- wrapper classes provide a way to use primitive data types (int, boolean, etc) as objects.

The given table shows the primitive type and the equivalent wrapper class:

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Vector in Java - vector is like the dynamic array which can grow or shrink to it's size unlike array. we can store n number of elements in it as there is no size.

Java Vector class declaration

public class Vector <E>
extends Objects <E>
implements List <E>, clonable, similizable.

Camlin | Page
Date / /

Program :

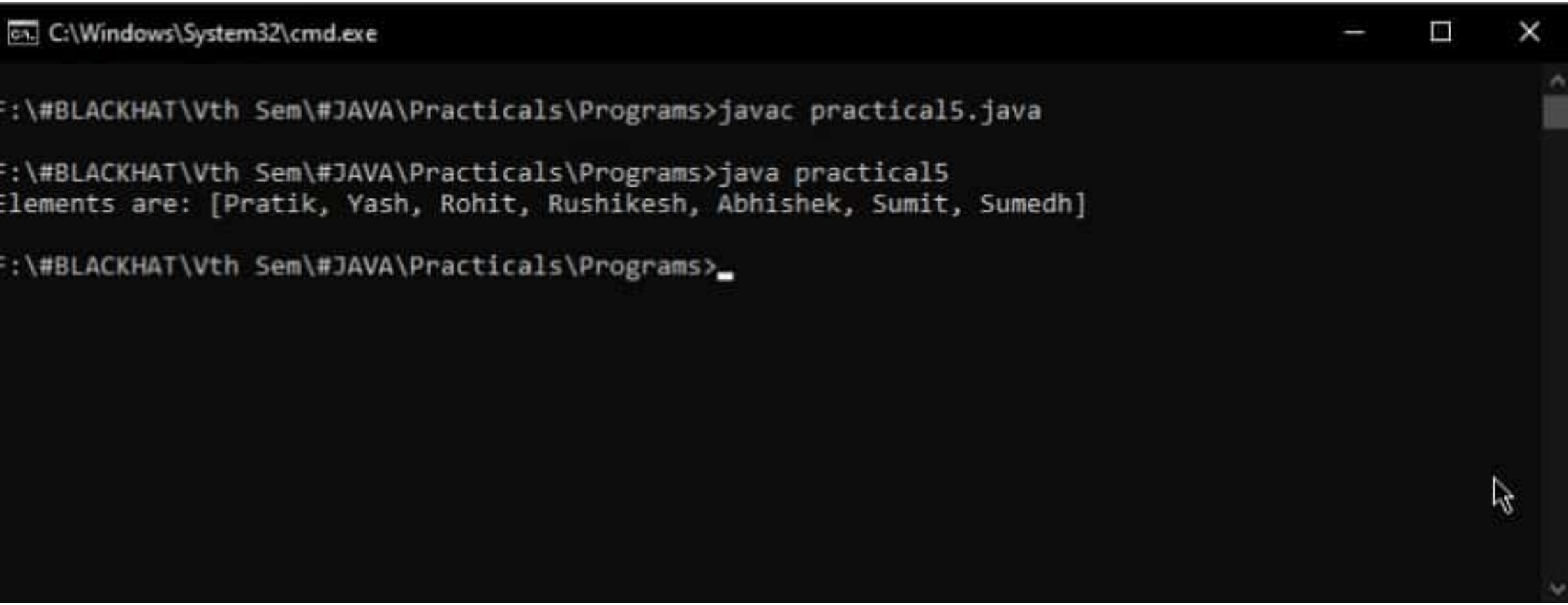
1st Program

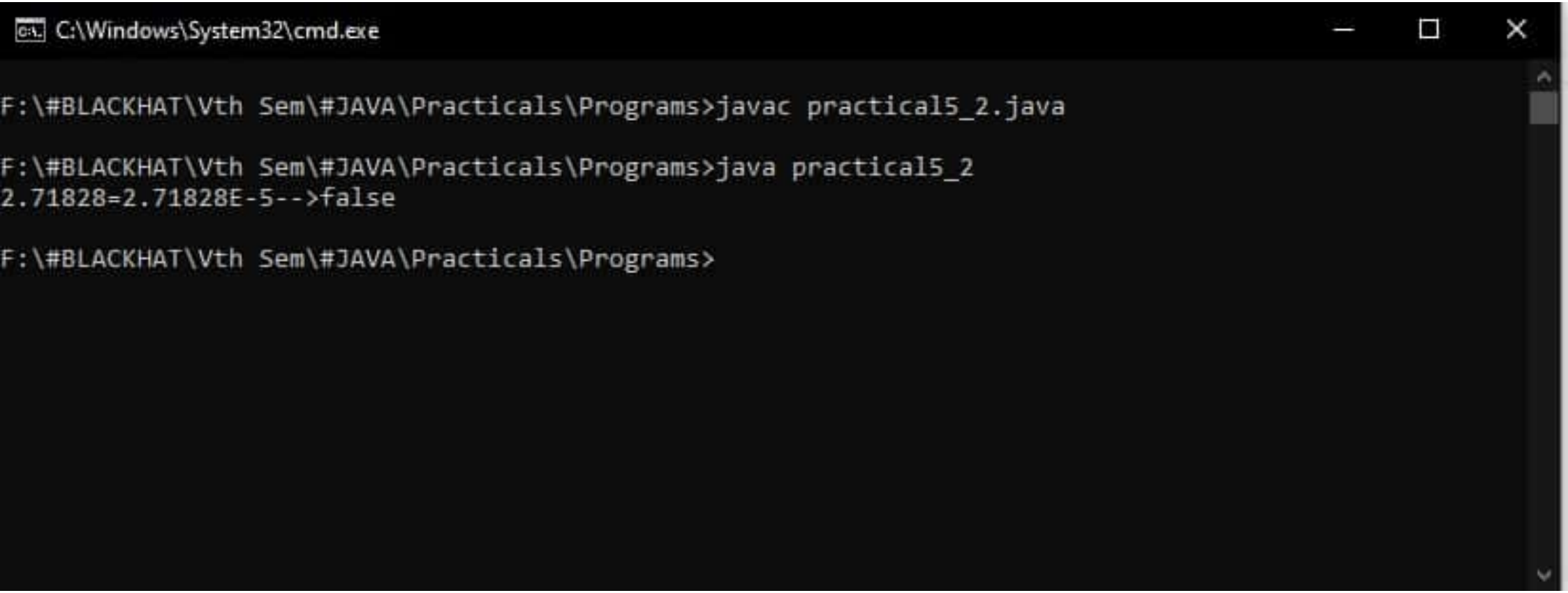
```
Import java.util.*;
class practical5{
    public static void main (String args[]){
    Vector <String> vec = new Vector <String> ();
    vec.add (" Pratik ");
    vec.add (" Yash ");
    vec.add (" Rohit ");
    vec.add (" Rushikesh ");

    vec.addElement (" Abhishek ");
    vec.addElement (" Sumit ");
    vec.addElement (" Sumedh ");
    System.out.println (" Elements are : " + vec);
    }
}
```

2nd Program

```
class practical5_2{
    public static void main (String arg[]){
        Double d1 = new Double (2.71828);
        Double d2 = new Double ("2.71828E-5 ");
        System.out.println (d1 + " = " + d2 + " --> "
                            + d1.equals(d2));
    }
}
```

Conclusion :- we have successfully created, debugged and compile the java program based on wrapper class and vectors.





Camlin | Page
Date / /

Practical No. 6

Aim :- Create, debug and run java programms based on classes with objects.

Theory :- A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

Syntax :-

```
public class Test {
    code Block ;
}
```

A class in java contains : Fields, methods, constructors, blocks, etc.

Objects - It is a basic unit of object oriented programming and represents the real life entities. A java programme creats many objects.

syntax :-

```
Test t = new Test();
```

Program :-

```
class Box {
    double width, height, depth;
}
public class practical6 {
    public static void main(String args[]) {
        Box b1;
        b1.width = 2;
        b1.height = 3;
        b1.depth = 4;
        double vol;
        vol = b1.width * b1.height * b1.depth;
        System.out.println("Volume of Box B1"
                + vol);
    }
}
```

Conclusion :- we have successfully performed a java program based on classes and objects.

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical6.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical6
Volume of b1 :24.0

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>
```

Practical No. 7

Aim :- Create, debug and run java programmms based on method overloading

Theory :- Method Overloading is a concept that allows to declare multiple methods with same name but different parameters in same class. Java supports method overriding and always occurs in same class. It is way through which java supports polymorphism. Method overloading can be done by

i) Changing No. of Arguments
ii) Changing the data types of arguments

In java, we can overload the main method using different number and types of parameter but the JVM only understand original main () method.

Programs :-

1st program

```
class practical7 {
    public static void main(String[] args){
        System.out.println(" Main with String ");
    }
```

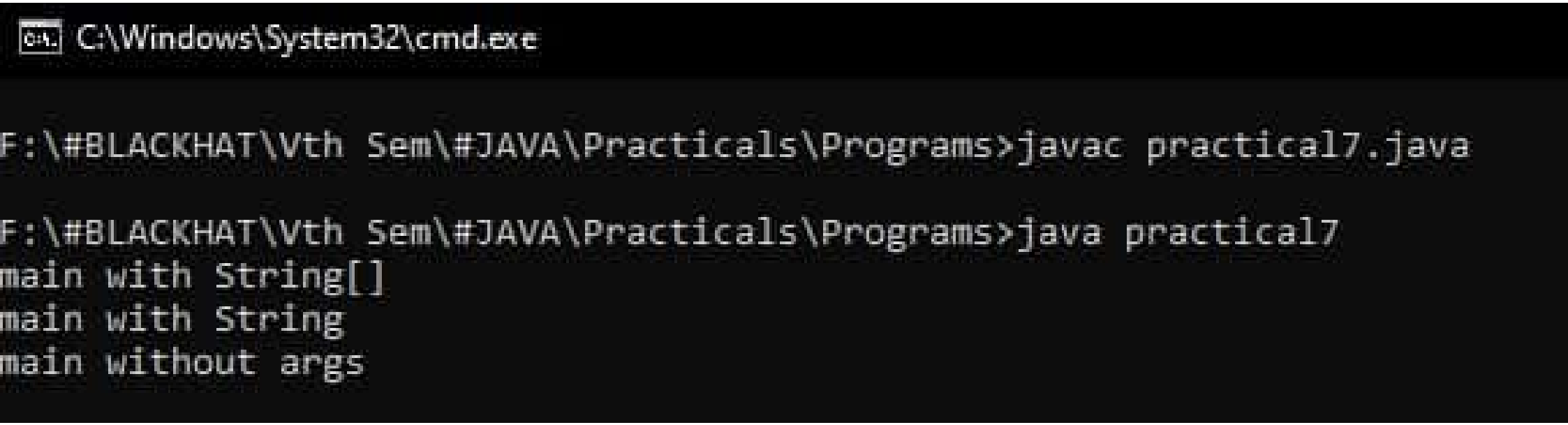
```
    public static void main(){
        System.out.println(" Main without args");
    }
    public static void main(String[] args){
        practical7 t = practical7;
        t.main("BLACKHAT");
        t.main();
    }
}
```

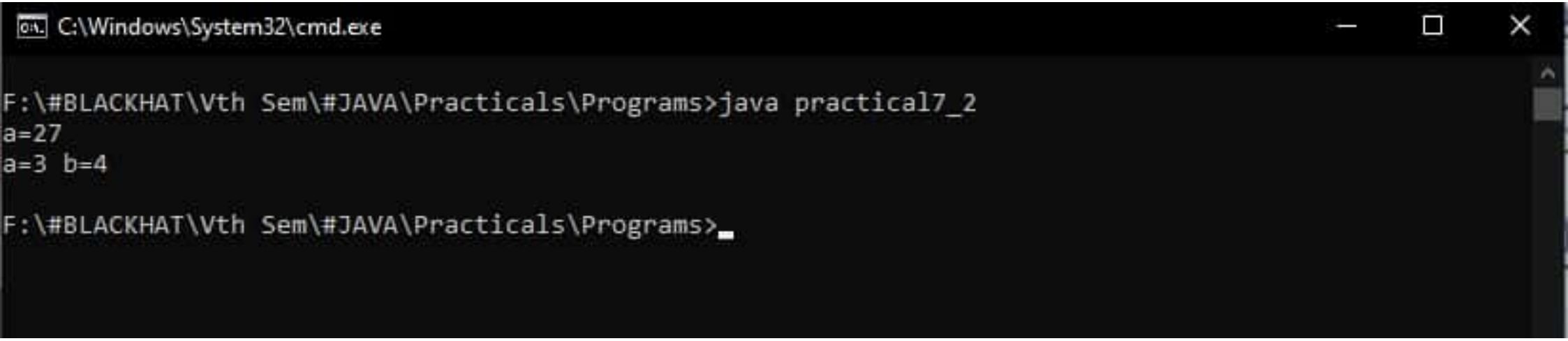
2nd program

```
class practical7_2{
    int a, b;
    void meth1(){
        a=3; b=4;
        System.out.println("a = "+a+" b = "+b);
    }
    void meth1(int i, int j){
        a=i; b=j;
        System.out.println("a = "+a+" b = "+b);
    }
    int meth1(int i, int j, int k){
        return i+j+k;
    }

    public static void main(String args[]){

        practical7_2 pr = new practical7_2;
```

```
int a = pr.meth1(6,9,10);
System.out.println("a="+a);
pr.meth1();
}
}
```

Conclusion :- we have successfully performed a javac program based on method overloading





Practical No. 8

Aim:- Create, debug and run java programm based on constructor overloading.

Theory :- Constructor Overloading in Java - a constructor is just like a method but without return type. It can also be overloaded like java methods.

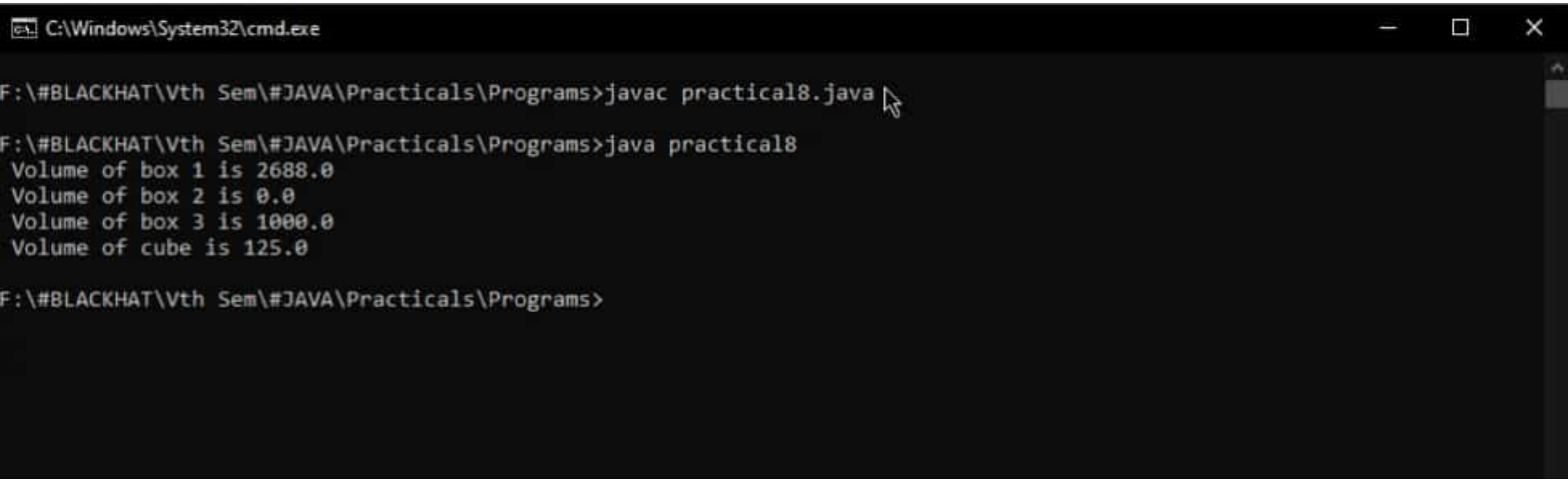
Constructor overloading in java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the lists and their types.

Program :-

```
class Box {
    double width, height, depth;
    Box(double w, ~~height~~double h, double d)
    {
        width = d;
        height = h;
        depth = d;
    }
    Box(double len){
        width = heigh = depth = len;
    }
```

```
    double volume () {
        return width * height * depth;
    }
}

public class practical8 {
    public static void main (String args[]) {
        Box b1 = new Box (5);
        Box b2 = new Box ();
        Box c1 = new Box (10);
        Box b3 = new Box (12, 14, 16);

        double vol;
        vol = ~~mybox1~~ b1.volume();
        System.out.println ("Volume of Box 1 is" +vol);
        vol = b2.volume();
        System.out.println ("Volume of Box 2 is" +vol);
        vol = b3.volume ();
        System.out.println ("Volume of Box 3 is" +vol);
        vol = c1.volume ();
        System.out.println ("Volume of cube is" +vol);
```

Conclusion :- we have successfully perform a constructor overloading program in java.



Practical No. 9

Aim:- Create, debug and run java program based on nested and inner class.

Theory :-

Nested classes - just like methods, variable of a class have another class as it's member, writing a class within another is allowed in java.

Syntax :-

~~Class Outer_class {~~

~~Class~~

```
class outer_class {
    class inner_class {
        ---
        ----
    }
}
```

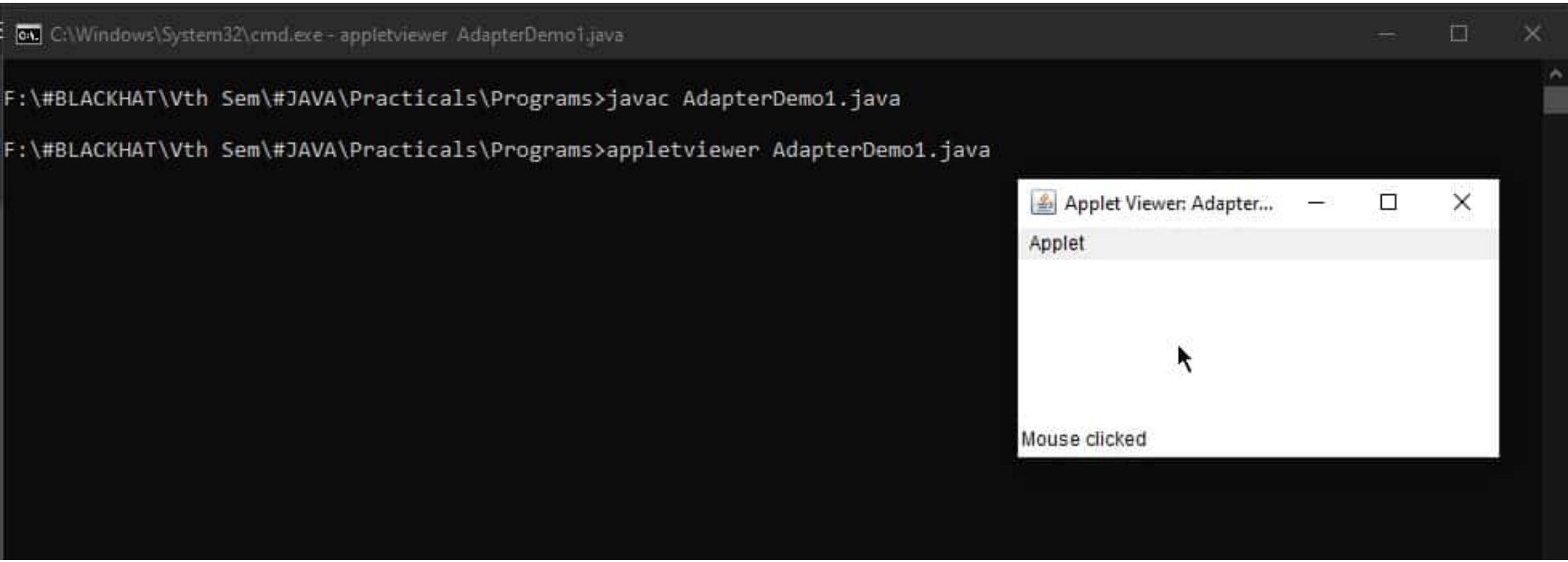
Inner classes - java inner classes or nested class is a class which is declared inside the class interface. we use inner classes to logically group classes and interface in one place so that it can be more readable and maintainable. Additional It can access all members of outer class including private data members and methods.

Program :-

Inner Class

```
import java.awt.*;
import java.applet.*;
/* <applet code = "AdapterDemo1" width = 300 height = 100>
  */
public class AdapterDemo1 extends Applet {
    public void init() {
        addMouseListener (new MyMouseAdapter());
    }
    class MyMouseListener extends MouseAdapter {
        public void mouseClicked (MouseEvent me) {
            showStatus ("Mouse clicked");
        }
    }
}
```

Conclusion :- we have successfully created, debugged and compiled a java program based on nested and inner class using applets



Practical No. 10

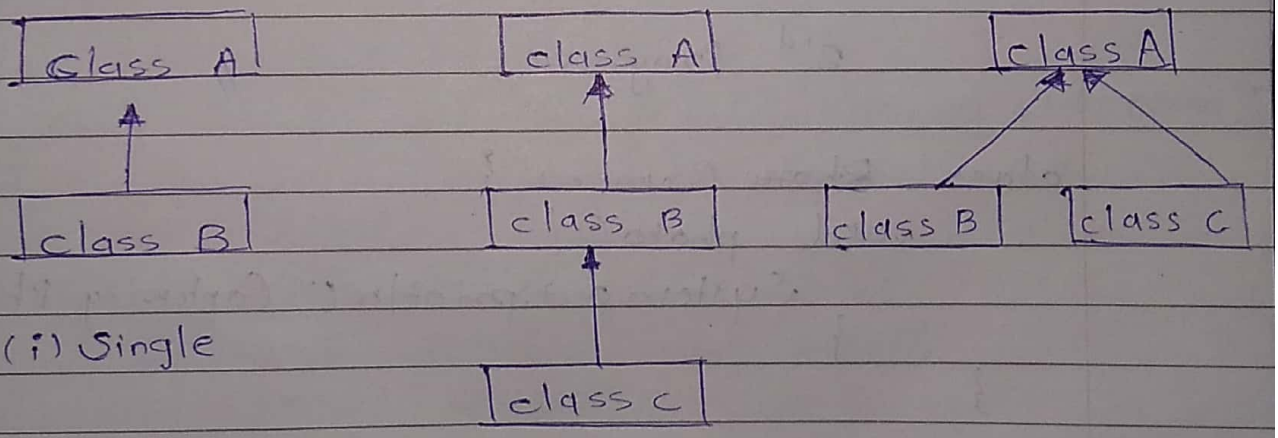
Aim:- create, debug and run java program based on inheritances.

Theory :- Inheritance is used to borrow properties and methods from an existing class.

Ex. phone → Smartphone
Superclass → subclass.

When a class inherits ~~it's~~ from a superclass, it's inherits parts of superclass methods and fields. Java doesn't support multiple inheritance i.e. two classes cannot be super class for a subclass.

There are three types of inheritance
i> Single
ii> multilevel
iii> hierchical



(i) Single

(ii) Multilevel

Programs :-

1st program

```
class Phone {
    void ring(){
        System.out.println("Ringing..."); }
    }
class SmartPhone extends Phone {
    void vcall(){
        System.out.println(" Video calling... "); }
    }

class practical10 {
    public static void main (String args[]){

    SmartPhone s = new SmartPhone();
        s.vcall();
        s.ring();
    }
}
```

2nd Program

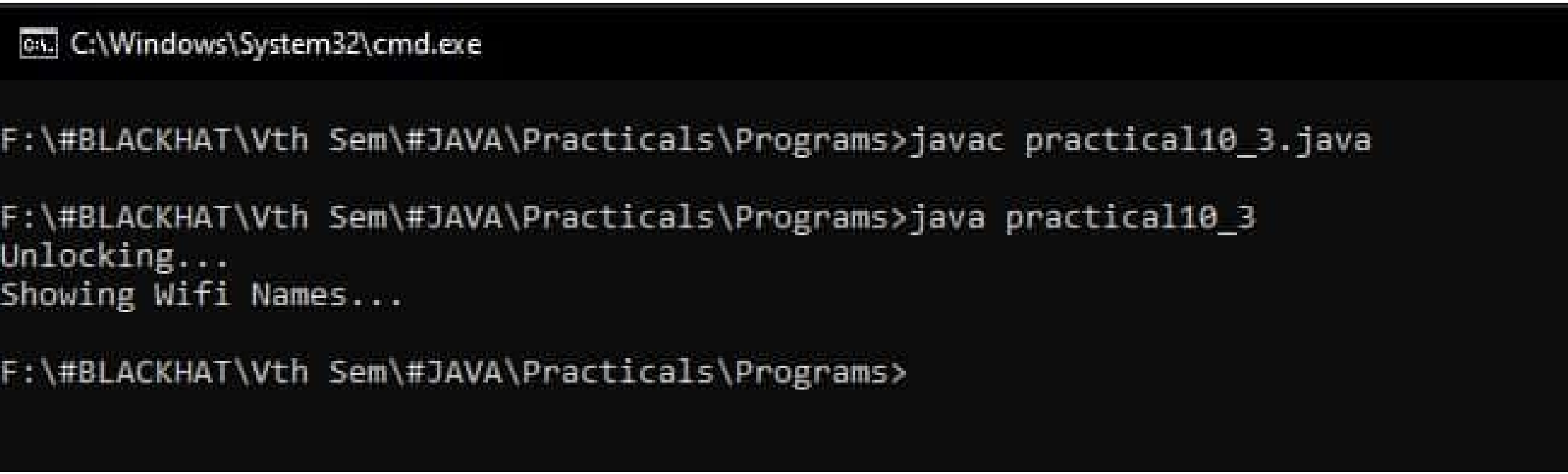
```
class Phone Camera {
    void photo(){
        System.out.println(" Capturing Photo");
        }
    }
class Vedio extend Camera {
    void vedio(){
        System.out.println(" Recording Vedio");
        }
    }
```

```
class SmartPhone extends Vedio {
        void Gallery(){
            System.out.println(" Opening Gallery...");
        }
}
class practical10_2 {
    Public static void main(String args[]) {
            SmartPhone s = new SmartPhone();
            s.photo();
            s. vedio();
            s. Gallery();
    }
}
    3rd program
class SmartPhone {
    void Unlock() { System. out. println("Unlocking.... "); }
}
class Location extend SmartPhone {
    void Gps() { System.out.println("Searching Location...");
}
class Wifi extend SmartPhone {
    void names() { System.out.println("Showing Wifi Name");
}
class practical10_3 {
    public static void main(String args[]){
        SmartPhone Wifi w = new WiFi();
        w. Unlock(); , w. wiFi();
    }
}
```

Conclusion :- we have successfully performed java program based on Inheritance and it's types.

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical10.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical10
Video Calling...
Ringing..

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>
```

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical10_2.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical10_2
Opening Gallery...
Recording Video...
Capturing Photo...

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>
```



Practical No. 11

Aim :- Create, debug and run java programs based on interface

Theory :- An interface in java is a blueprint of a class. it has static constants and abstract methods

The Interface is group of related methods with Empty bodies.

Example :-

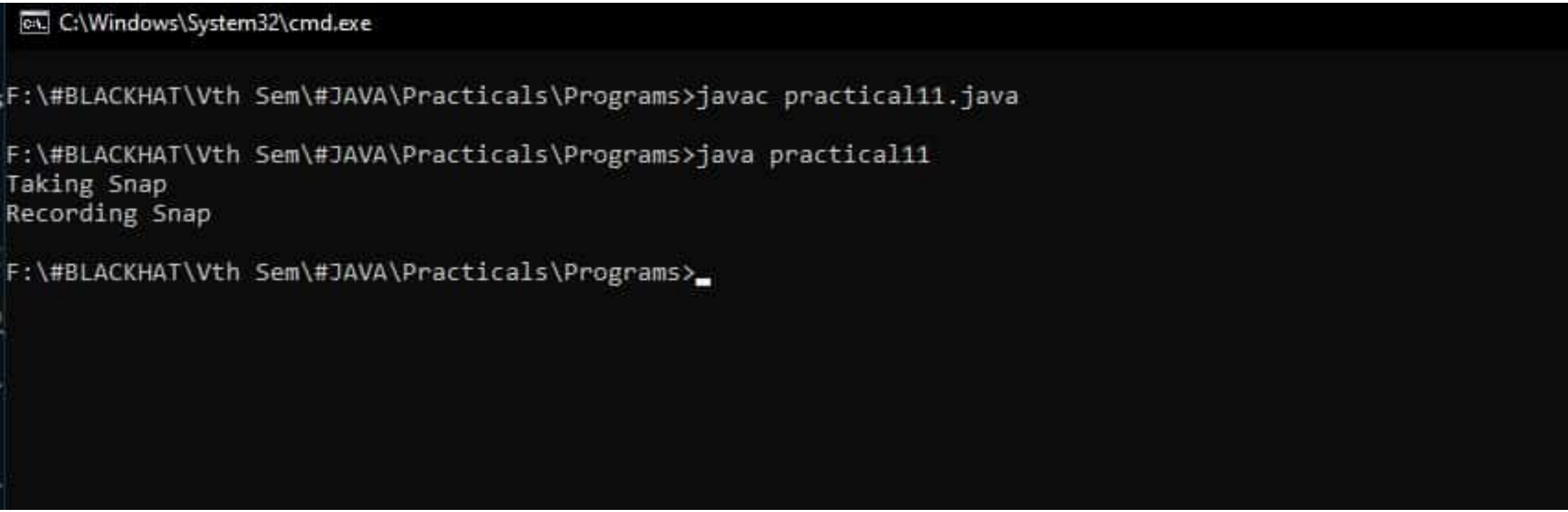
```
interface Bicycle {
    void applyBrake (int decrement)
    void speedUp (int increment)
}
class AvonCycle implements ~~Bicly~~ Bicycle {
    executable code :
    ------ ;
    ------ ;
}
```

An interface have default and static methods. Default methods enable us to add new functionality to existing interface.

Program :-

```
interface SnapChat {
        void takesnap();
        void recordsnap();
}

class Mysmartphone implements SnapChat {
        public void takesnap() { System.out.println("Taking snap"); }
        public void recordsnap() { System.out.println("Recording snap"); }
}

public class practical11 {
    public static void main(String args[]) {
            Mysmartphone s = new Mysmartphone();
            s.takesnap();
            s.recordsnap();
    }
}
```

Conclusion :- we have successfully performed a java program based on interface.

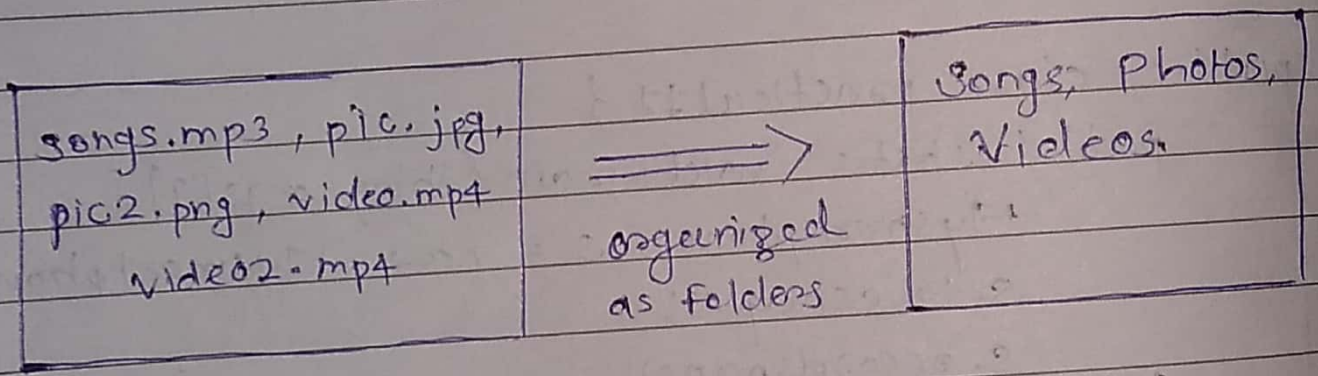


Practical No. 12

Aim:- Create, debug and run java programs based on ~~inheritance~~ package

Theory. A package is used to group related classes. packages helps in avoiding name conflicts.

There are two types of packages
i) Built-in Packages - Java API
ii) User define Packages - custom packages

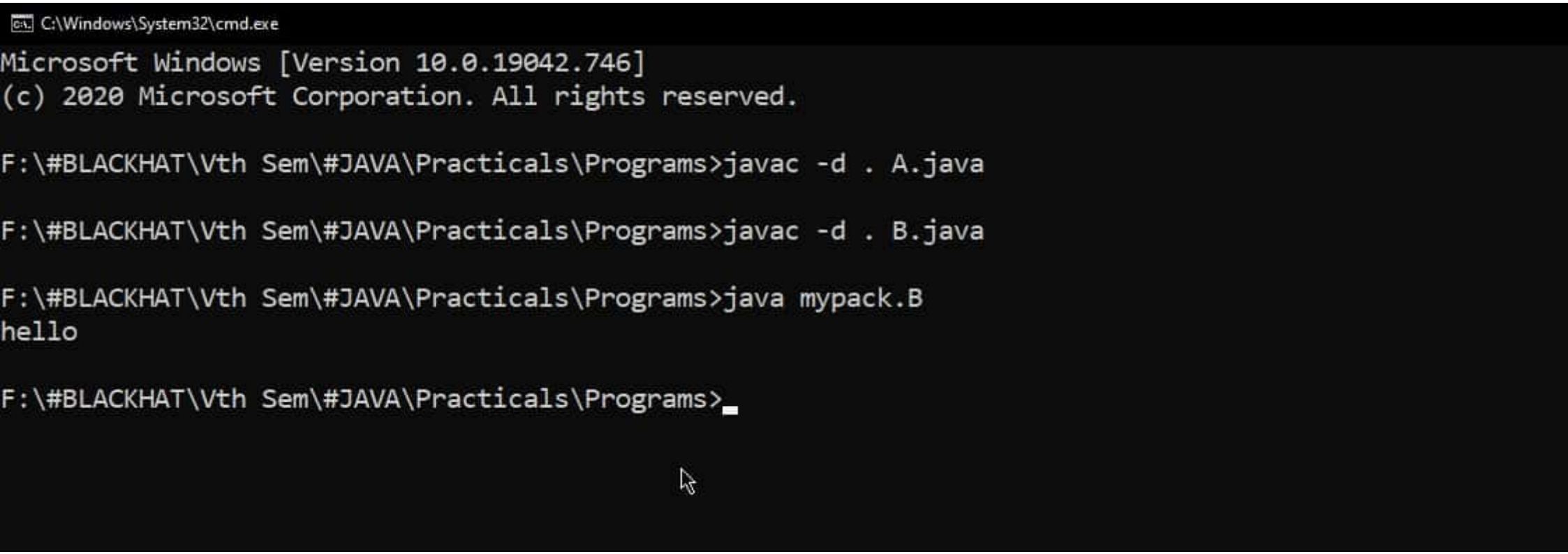


Program :-

```
package pack;
public class A {
    public void msg() {
        System.out.println("Hello");
    }
}
```

```
package mypack;
import pack.*;
class B {
    public static void main(String args[]) {
        A obj = new A();
        obj.msg();
    }
}
```

Conclusion :- we have successfully performed a java program based on packages.



Practical No. 13

Aim :- Create, debug and run java programs based on exception handling.

Theory :-

The java Exception Handling is one of the powerful mechanism to handle the runtime errors so that normal Flow of the application can be maintained.

Java Exception handling is managed by five keywords

i) try,
ii) catch
iii) throw
iv) throws
v) finally.

Java Exception can be generated by the java run-time system, or they can be manually generated by your code.

General form of exception-handling block:

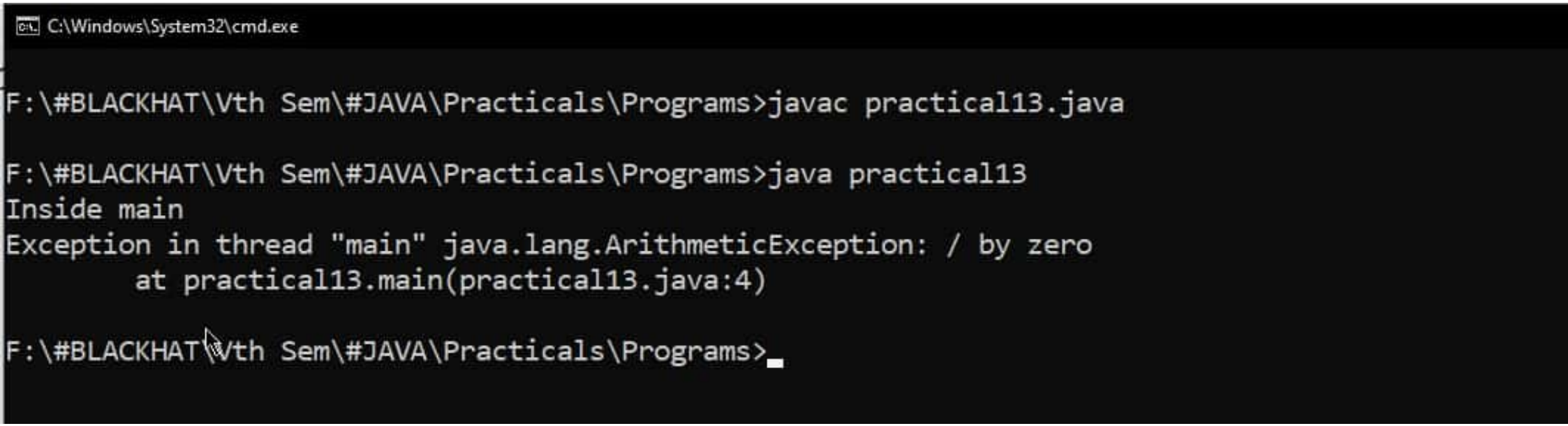
```
try {
    // block of code
}
catch (ExceptionType1 exOb) {
    // exception handler
}
Finally {
    // block of code before try block }
```

Camlin Page
Date / /

Program :-

```
class practical13 {
    public static void main (String args[]){
        System.out.println("Inside main");

        int a = 10/0;
        System.out.println("a = "+a);
        System.out.println("Program Exiting....");
    }
}
```

Conclusion :- we have successfully executed an exception handling program in java.



Camlin Page
Date / /

Practical No. 14

Aim :- Create, debug and run java programs based on user defined Exception.

Theory :-

User defined Exception - if you are creating your own exception that is know as custom exception or user defined exception.

This exceptions are used to customize the exception according to user need. you can have your own exception and message.

There is no need to override any methods available in the exception class.

Program :-

```
class BankException extends Exception {
    double b, w;
    BankException () { }
    BankException (double b, double w) {
        b = b;
        w = w;
    }
    public String toString () {
        return "Insufficient funds";
    }
}
class Bank {
    double balance;
    Bank (){
        balance = 50000;
    }
}
```

```
public class BankDemo {
    static void withdraw(double wd) throws
                                    BankException {
        if (wd <= b.balance) {
            System.out.println("Collect your cash");
        }
        else
            throw new BankException(b.balance, wd);
    }
    public static void main(String args[]) {
        try {
            withdraw(55000);
        }
        catch (BankException e) {
            System.out.println("Error : "+e);
        }
    }
}
```

Conclusion :- We have successfully performed a java program based on user defined exception.

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical14.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical14
Error : Insufficient funds

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>
```

Practical. No. 15

Aim:- Create, debug and run java programs based on Threads by implementing Runnable interface

Theory :-

Threads - Facilities to allow multiple activities within a single process. Every java program creates at least one thread ~~as~~ [main() thread]. Additional threads are created through the Thread constructor or by instantiating classes that extend the Thread class

Thread implementation in java can be achieve in two ways :

i) ~~Implementing~~ Extending the java.lang.Runnable Interface.

ii) Extending the java.lang.Thread class.

Runnable Interface

It is used to execute code on a concurrent thread. It is an interface which is implemented by any class.

public void run()

This method takes in no arguments. When the object of class implementing Runnable class is

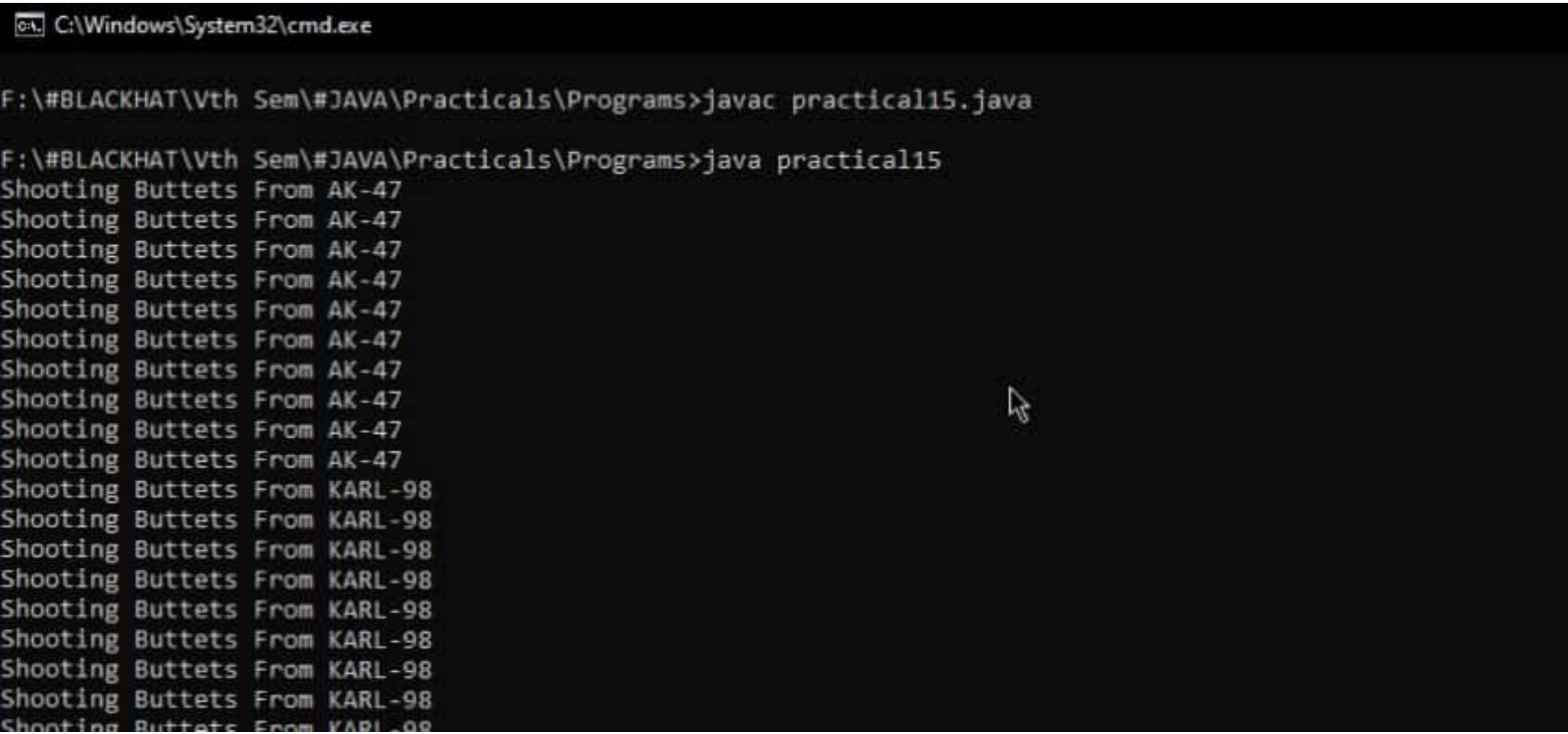
used to create a thread, then the run method is invoked in the thread which executes seperately.

Program 8-

```
class MyRunnableGun1 implements Runnable {
    public void run(){
        int i = 0;
        while (true){
            i++;
            System.out.println("Shooting Bullets
                from AK-47");
        }
    }
}

class MyRunnableGun2 implements Runnable {
    public void run(){
        int i = 0;
        while (true){
            i++;
            System.out.print("Shooting Bullets
                from KARL-98");
        }
    }
}
```

```
public class practical575 {
    public static void main(String[] args) {
        MyRunnableGun1 ak47 = new MyRunnableGun1();
        Thread gun1 = new Thread(ak47);
        MyRunnableGun2 kar98 = new MyRunnableGun2();
        Thread gun2 = new Thread(kar98);
        gun1.start();
        gun2.start();
    }
}
```

Conclusion :- we have successfully performed a java program based on Threads by implementing Runnable interface.



Practical No. 16

Aim :- Create, debug and run java program based on Threads by extending Thread class

Theory :

Extending Thread class

The class should extend Java Thread class. The class should override run() method. The functionality that is expected by the Thread to be executed is written in the run() method.

void start(): Creates a new thread and makes it runnable.

void run(): The new thread begins it's life inside this method.

Program :

```
class MyThread1 extends Thread {
        public void run(){
            while (true){
                System.out.println("Chatting with Friend");
            try{
                sleep(1000);
            }catch (InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

```
class Mythread2 extends Thread {
    public void run(){
        while (true) {
            System.out.println("Coding Java Programs");
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
public class practical16 {
    public static void main(String[] args){
        Mythread1 t1 = new Mythread1();
        Mythread2 t2 = new Mythread1();
        t1.start();
        t2.start();
    }
}
```

conclusion :- we have successfully performed and executed a java program based on Threads by extending Thread class.

```
C:\Windows\System32\cmd.exe

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical16.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical16
Chatting with friend
--------------------------
Coding Java Programs
======================
Chatting with friend
--------------------------
Coding Java Programs
======================
Chatting with friend
--------------------------
Coding Java Programs
======================
Coding Java Programs
======================
Chatting with friend
--------------------------
```

Practical. No. 17

Aim :- Create, debug and run java programs based on Applets.

Theory :-

Applets - An applet is an extension of the java program that run in a Web browser. An applet can be a fully functional java application because it has the entire java API at it's disposal.

Every applet is an extension of the java.applet.Applet class.

life Cycle of the Applet -

There are four methods in the Applet class gives you the framework.

i) init
ii) start
iii) Stop
iv) destroy
v) paint

An appletviewer executes your applet in a window. This is generally the fastest and easiest way to test applet.

Program :

1st program

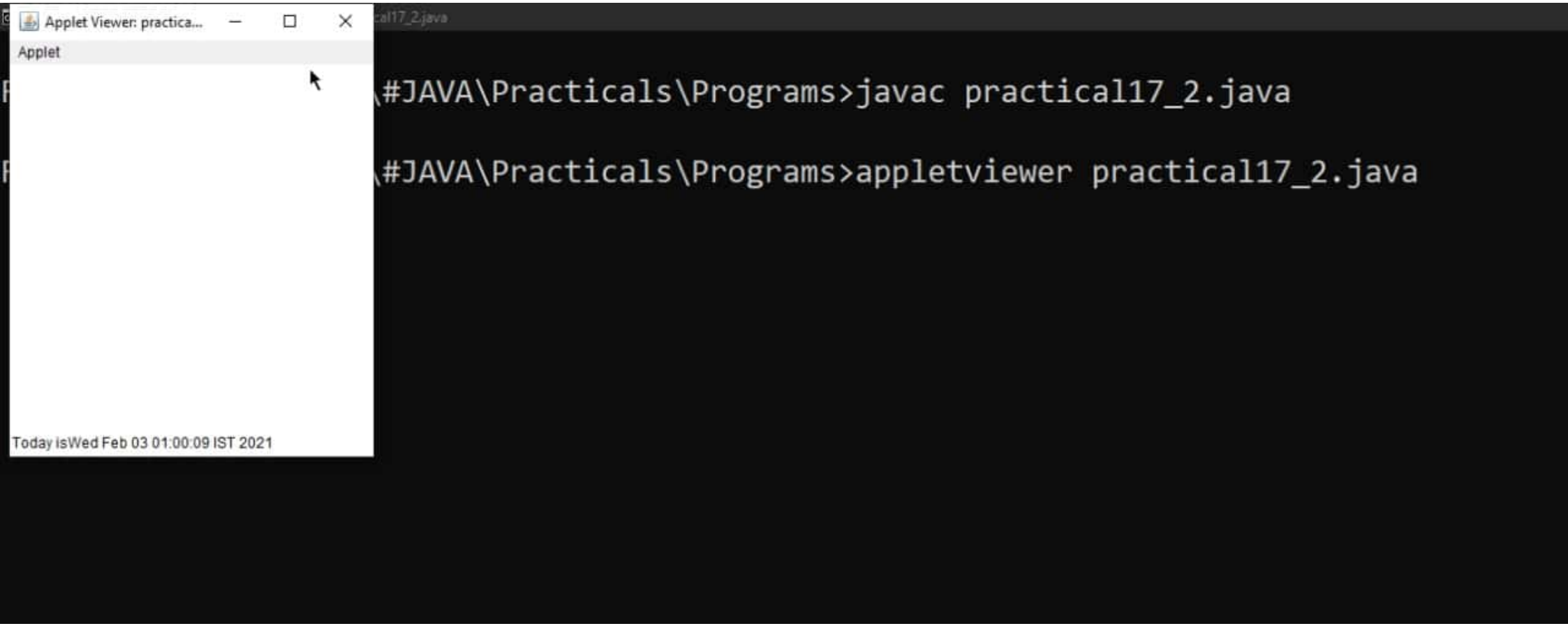
```
import java.applet.*;
/* <applet code = "Applet1" width = 600 height = 300>
   </applet> */

public class Applet1 extends Applet {
    public void paint (Graphics g) {
        g.drawString ("A Simple Applet", 20, 20);
    }
}
```

2nd program

```
import java.applet.*;
import java.util.Date;

/* <applet code = "practical17_2" width = 300 heigh = 300>
   </applet> */

public class practical17_2 extends Applet {
    public void paint ( Graphics g) {
        Date dt = new Date();
        super.showStatus ("Today is " + dt);
    }
}
```

Conclusion :- we have successfully perform a java programs based on applets.

```
C:\Windows\System32\cmd.exe - appletviewer Applet1.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac Applet1.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>appletviewer Applet1.java
load: class Applete1 not found.
java.lang.ClassNotFoundException: Applete1
        at sun.applet.AppletClassLoader.findClass(AppletClassLoader.java:219)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
        at sun.applet.AppletClassLoader.loadClass(AppletClassLoader.java:152)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
        at sun.applet.AppletClassLoader.loadCode(AppletClassLoader.java:635)
        at sun.applet.AppletPanel.createApplet(AppletPanel.java:799)
        at sun.applet.AppletPanel.runLoader(AppletPanel.java:728)
        at sun.applet.AppletPanel.run(AppletPanel.java:378)
        at java.lang.Thread.run(Thread.java:748)
```



Practical No. 18

Aim :- Create, debug and execute java program based on graphics to draw, Fill, different shapes.

Theory :- The graphics class is the abstract base class for all graphics contexts. that allows an application to draw onto components that are realized on various devices, as well as onto off-screen images.

A Graphics object encapsulates state information needed for the basic rendering operations that java supports. This state information includes the following properties:

i) The Coponet object on which to draw.
ii) A translation option for rendering and clipping co-ordinates.
iii) The current clip.
iv) The current font.
v) The current color
vi) The current logical pixel operation function
vii) The current XOR alternation color.

Program :-

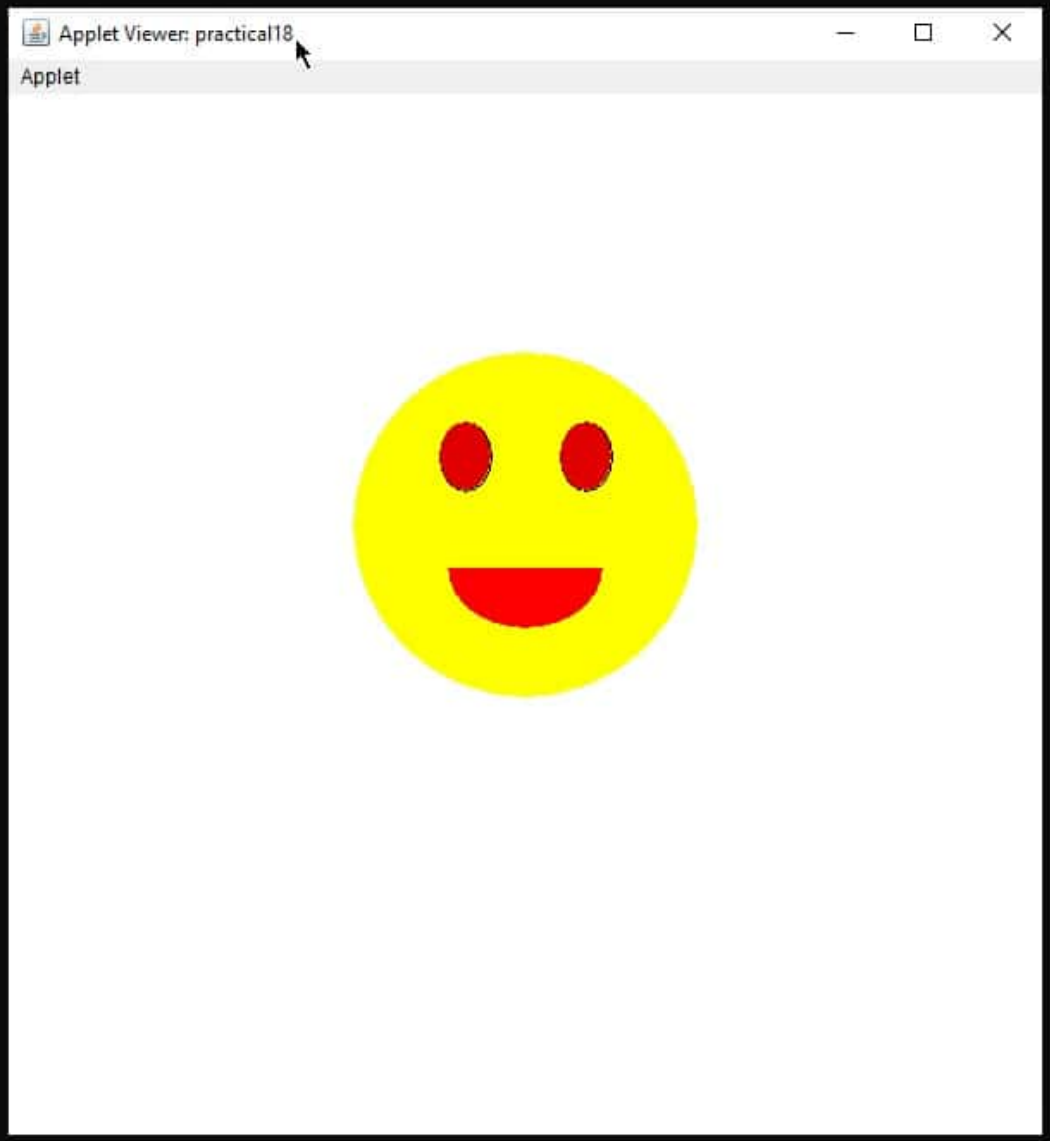
```
import java.awt.*;
import java.applet.*;
/*<applet code="Practical18" width = 600 height = 600>
  </applet> */

public class Practical18 extends implements Runnable{
      Thread t = new Thread();
      Color c;
      public void init(){}
      public void run(){}
      public void paint(Graphics g){
            boolean blink = false;
            int i = 225;
            int j = 0;
            int k = 0;
            while(true){
              g.setColor(Color.yellow);
              g.fillOval(200,150,200,200); // Face
              g.setColor(Color.black);
              g.drawOval(250,190,30,45); // left eye
              g.setColor(Color.white);
              g.fillOval(250,190,28,40);
              g.setColor(Color.black);
              g.fillOval(255,199,20,20);
              g.setColor(Color.white);
              g.fillOval(260,205,10,10);
```

```
g.setColor(Color.black);
g.drawOval(320,190,30,10); // right eye
g.setColor(Color.white);
g.fillOval(320,190,28,40);
g.setColor(Color.black);
g.fillOval(325,199,20,20);
g.setColor(Color.white);
g.fillOval(330,205,10,10);
if(!blink){
        c = new Color(i,j,k);
        g.setColor(c);
        g.fillOval(250,190,30,40);
        g.fillOval(320,190,30,40);
        if(j==0 && k==0){
                j=100;
        }
        else if(j==100 && k==0){
                j=225;
                k=255;
        }
        else if(j==255 && k==255){
                j=0;
                k=0;
        }
}
g.setColor(Color.red);
g.fillArc(255,240,90,70,0,-180);
blink = !blink;
try{
        t.sleep(500); }
catch(Exception e){}
}
}
```

Conclusion :- we have successfully created, debugged and excuted a applet program in java based on graphics to draw, fill with different colors and shapes.

C:\Windows\System32\cmd.exe - appletviewer practical18.java

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical18.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>appletviewer practical18.java
```



Camlin Page
Date / /

Practical No. 19

Aim:- Create, debug and execute java programs based on mouse events and keyboard events

Theory :- Java applets are mainly event driven programs. User activities related to keyboard and mouse are called events.

Handling events.

step 1: import classes from the packages.

step 2: Implement the appropriate listener interface.

step 3: Register the listener object with the appropriate event source.

step 4: Define all the methods of the implemented listener interfaces.

To handle mouse event, you must implement the MouseListener and the MouseMotionListener interfaces.

Handling keyboard Events: You will be implementing the KeyListener interface here.

Program :-

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/* <applet code = "Practical19" width = "600" height = "600">
    </applet> */

public class Practical19 extends Applet {
        public void init(){
            Button btn = new Button("Button");
        setLayout(null);
        btn.setBounds(250,200,100,20);
        btn.addMouseListener(new HelperListener(this));
        add(btn);
        setVisible(true);
        }
}
class HelperListener extends MouseAdapter{
        Assignment13 ad;
        public HelperListener(Assignment13 ad){
            this.ad = ad;
        }
```

```
public void mouseClicked(MouseEvent e){
    if((e.getModifiers() & InputEvent.BUTTON1_
                                MASK) != 0){
        System.out.println("Left click detected");
    }
    if((e.getModifiers() & InputEvent.BUTTON2_
                                MASK) != 0){
        System.out.println("Middle Click detected");
    }
    if((e.getModifiers() & InputEvent.BUTTON3_MASK)
                                != 0){
        System.out.println("Right Click detected");
    }
    int count = e.getClickCount();
    switch (count) {
        case 1:
            ad.setBackground(Color.Black);
            break;
        case 2:
            ad.setBackground(Color.Yellow);
            break;
        case 3:
            ad.setBackground(Color.red);
            break;
        case 4:
            ad.setbackgoound(Color.blue);
            break;
        case 5:
            ad.setbackground(Color.green);
            break;
        default: break;
    }
}
```

Program for keyboard events.

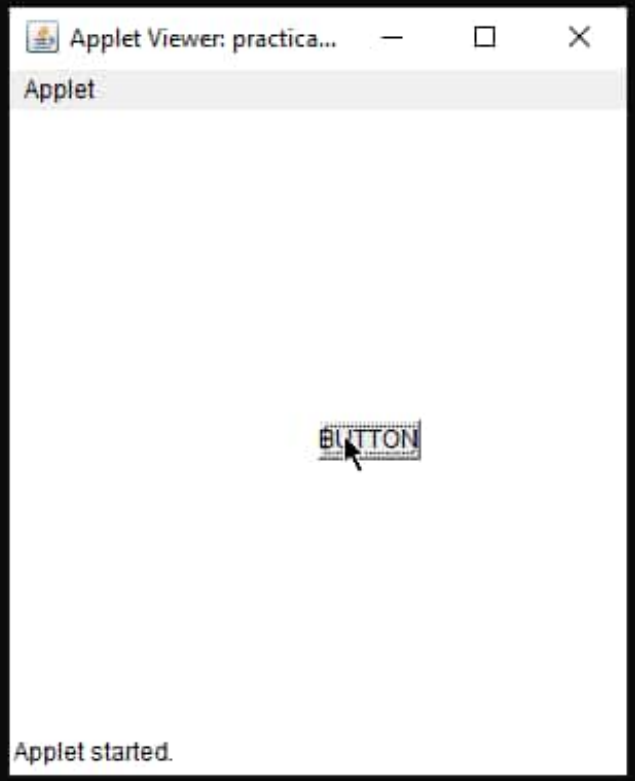
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code = "practical19_2" width = 300
    height = 100>
  </applet> */

public class practical19_2 extends Applet implements
                                    KeyListener {
    String msg = " ";
    int X = 10, Y = 20;
    public void init(){
      addKeyListener(this);
      requestFocus();
    }
    public void keyPressed(KeyEvent ke){
          showStatus("Key Down");
    }
    public void keyReleased(KeyEvent ke){
          showStatus("Key Up");
    }
    public void keyTyped(KeyEvent ke){
          msg += ke.getKeyChar();
          repaint();
    }
```

```
public void paint(Graphics g){
        g.drawString(msg,x,y);
    }
}
```

Conclusion :- We have successfully performed applets in java based on mouse events and keyboard events.

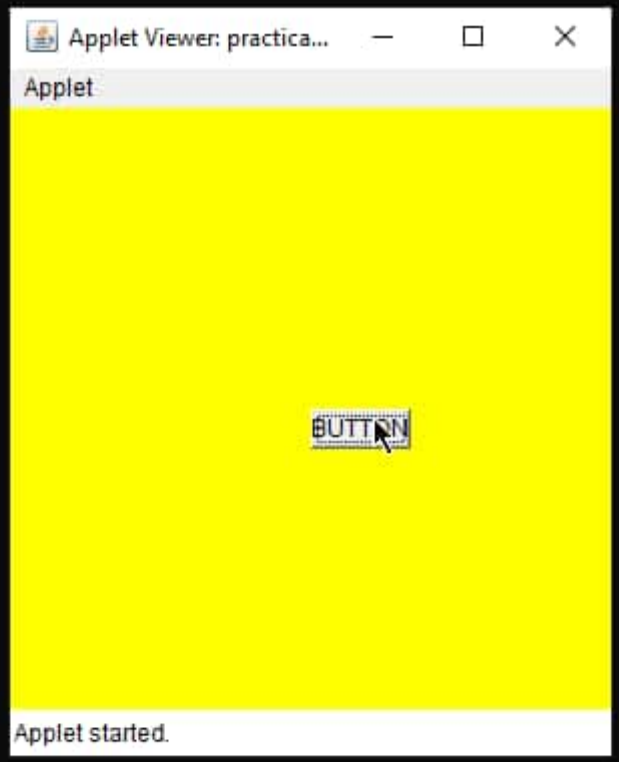
C:\Windows\System32\cmd.exe - appletviewer practical19.java

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical19.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>appletviewer practical19.java
```



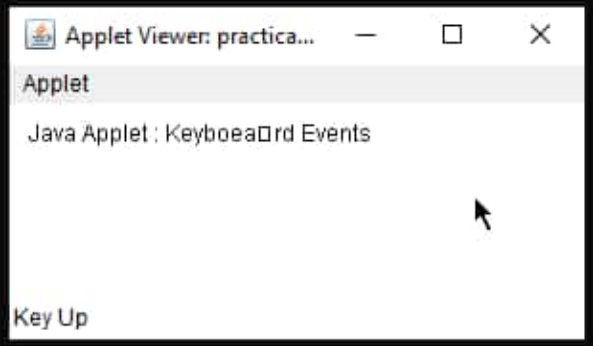
C:\Windows\System32\cmd.exe - appletviewer practical19.java

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical19.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>appletviewer practical19.java
Left click detected
```



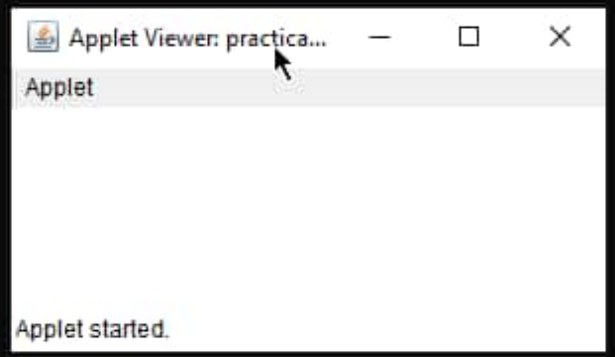
C:\Windows\System32\cmd.exe - appletviewer practical19_2.java

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical19_2.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>appletviewer practical19_2.java
```



C:\Windows\System32\cmd.exe - appletviewer practical19_2.java

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical19_2.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>appletviewer practical19_2.java
```



Practical No. 20

Aim :- Create, debug and run java programs based on read and write characters from a file using input/output stream.

Theory :- FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in java.

A FileOutputStream is an output stream for writing data in file. If you want to write primitive values then use FileOutputStream. Instead for character-oriented data, prefer FileWriter. But you can write byte-oriented as well as character-oriented data.

Constructure :-
- FileOutputStream (File, File)
- FileOutputStream (File file, boolean app
- FileOutputStream (String name)
- FileOutputStream (String name, boolean appe

A FileInputStream obtains input bytes from a file. It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader. It should be used to read byte-oriented data. For example to read image etc.

Constructor :- • FileInputStream (String Filename)

• FileInputStream (File file)

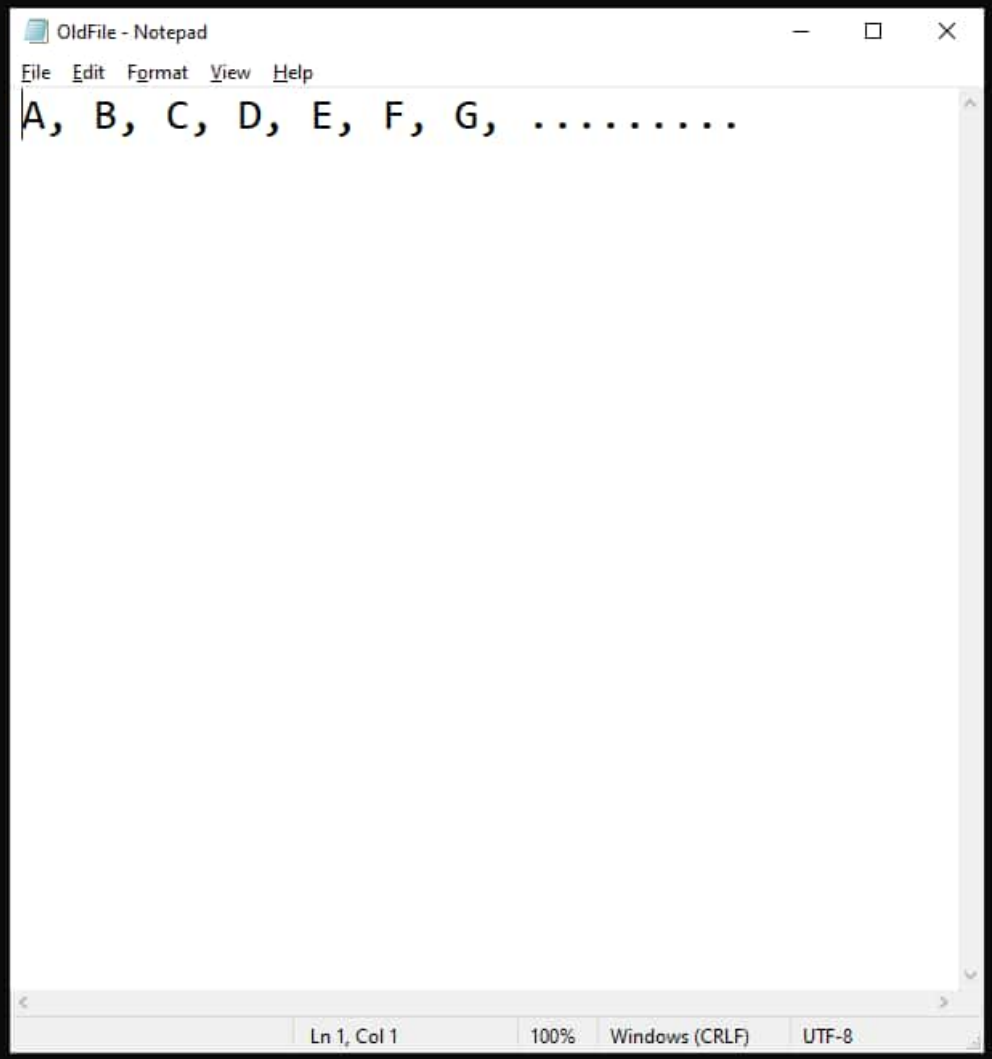
Program :-

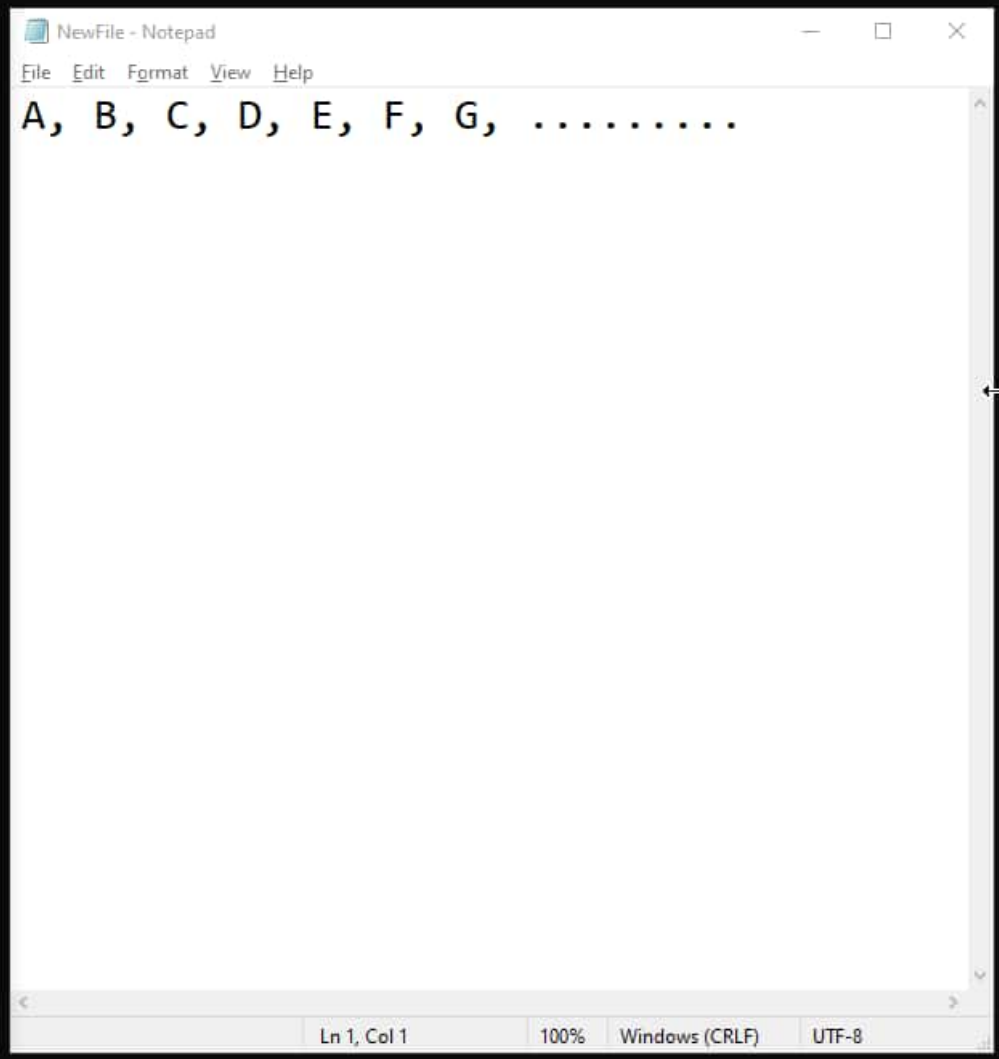
```
import java.io.*;
class practical20{
    public static void main(String args[]) throws
                    Exception{
        FileInputStream fin = new FileInputStream
                        ("OldFile.txt");
        FileOutputStream fout = new FileOutputStream
                        ("NewFile.txt");
        int i=0;
        while((i=fin.read())!=-1){
            fout.write((byte)i);
        }
        fin.close();
    }
}
```

Conclusion :- we have successfully created, debugged and executed the java program based on read and write characters from a file using input/output stream.

C:\Windows\System32\cmd.exe

```
F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>javac practical20.java

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>java practical20

F:\#BLACKHAT\Vth Sem\#JAVA\Practicals\Programs>
```





Practical No. 21

Aim :- Create, debug and run java programs based on object serialization.

Theory :- Serialization of a class is enables by the class implementing the java.io.serializeable interface. The serialization interface has no methods or fields and serves only to identify the semanties of being serializable.

ObjectOutputStream :-

An objectoutputstream is used to write primitive datatype and java objects. To an outputstream. Only object that supports the java.io.serializable interface can be written to string.

ObjectInputStream :-

An ObjectInputStream serializes objects and primitive data written using an object output stream.

Program :

1st program

```
import java.io.serializable;
public class Student implements Serializable {
        int id;
        String name;
        public Student(int id, String name){
                this.id = id;
                this.name = name;
        }
}
```

2nd program

```
import java.io.*;
class Persist {
        public static void main(String args[]) throws
                                        Exception {
                Student s1 = new Student(108, "Rahul");
                FileOutputStream fout = new FileOutputStream
                                        ("F.txt");
                ObjectOutputStream out = new ObjectOutputStream
                                        (fout);
                out.writeObject(s1);
                out.flush();
                System.out.println("Success");
        }
}
```

3rd program

```
import java.io.*;
class deperesist{
    public static void main (String[] args) throws
                                        Exception{

        ObjectInputStream in = new ObjectInputStream
                                    (new FileInput
                                    Stream("f.txt"))
        Student s = (Student) in.readObject();
        System.out.println(s.id + "  " + s.name);
        in.close();
    }
}
```

Conclusion :- we have successfully performed a java program based on objects of serialization.