

GURU NANAK DEV ENGINEERING COLLEGE,
LUDHIANA

JAVA LABORTARAY
Code- LPC IT 109



Submitted by-

Submitted to-

Name	URN	CRN
------	-----	-----

Aanshika	2004878	2021001
----------	---------	---------

Abhinav	2004879	2021004
---------	---------	---------

Abhishek	2004880	2021005
----------	---------	---------

Aditi	2004881	2021006
-------	---------	---------

INDEX

S.NO	PRACRICAL NAME	PAGE NO
1	Handling various data types	4-6
2	Type casting	7-8
3	Arrays – 1D and 2 D	9-11
4	Various control structures	12-14
5	Various decision structures	15-19
6	Recursion	20-21
7	Method Overloading by passing objects as arguments	22-25
8	Constructor Overloading by passing objects as arguments	26-27
9	Various access control and usage of static, final and finalize ()	28-30
10	Command line arguments	31
11	Various types of inheritance by applying various access controls to its data members and methods	32
12	Method overriding	33
13	Abstract class	34
14	Nested class	35
15	Constructor chaining	36
16	Importing classes from user defined package and creating packages using access protection	37-39
17	Interfaces, nested interfaces and use of extending interfaces	40-42

18	Exception Handling - using predefined exception	43-44
19	Exception Handling - creating user defined exceptions	45
20	Multithreading by extending Thread Class	46-47
21	Multithreading by implementing Runnable Interface	48-49
22	Thread life cycle	50-51
25	Event Handling	52-56
28	String class and its methods	57-58
29	StringBuffer class and its methods	59-60

PRACTICAL 1

Write a program of Data Types In Java

Java has two categories in which data types are segregated

1. Primitive Data Type: such as Boolean, char, int, short, byte, long, float, and double

1. Boolean type

The boolean data type has two possible values, either true or false.

Default value: false.

They are usually used for true/false conditions.

2. byte type

The byte data type can have values from -128 to 127 (8-bit signed two's complement integer).

If it's certain that the value of a variable will be within -128 to 127, then it is used instead of int to save memory.

Default value: 0

3. short type

The short data type in Java can have values from -32768 to 32767 (16-bit signed two's complement integer).

If it's certain that the value of a variable will be within -32768 and 32767, then it is used instead of other integer data types (int, long).

Default value: 0

4. int type

The int data type can have values from -2³¹ to 2³¹-1 (32-bit signed two's complement integer).

If you are using Java 8 or later, you can use an unsigned 32-bit integer.

This will have a minimum value of 0 and a maximum value of 2³²-1. To

learn more, visit [How to use the unsigned integer in java 8?](#)

Default value: 0

5. long type

The long data type can have values from -2^{63} to $2^{63}-1$ (64-bit signed two's complement integer).

If you are using Java 8 or later, you can use an unsigned 64-bit integer with a minimum value of 0 and a maximum value of $2^{64}-1$.

Default value: 0

6. double type

The double data type is a double-precision 64-bit floating-point.

It should never be used for precise values such as currency.

Default value: 0.0 (0.0d)

7. float type

The float data type is a single-precision 32-bit floating-point. Learn more about single-precision and double-precision floating-point if you are interested.

It should never be used for precise values such as currency.

Default value: 0.0 (0.0f)

8. char type

It's a 16-bit Unicode character.

The minimum value of the char data type is '\u0000' (0) and the maximum value of the is '\uffff'.

Default value: '\u0000'

9.String type

Java also provides support for character strings via java.lang.String class

Strings in Java are not primitive types. Instead, they are objects

Code-

```
public class Main
{
    public static void main(String[] args) {
        String name="Aanshika";
        int crn=2021001;
        int urn=2004878;
        String department="Inforamation Technology";
        float cgpa=8.89f;

        System.out.println("Student Name"+name);
        System.out.println("Student C.R.N"+crn);
        System.out.println("Student U.R.N"+urn);
        System.out.println("Student Department"+department);
        System.out.println("Student CGPA"+cgpa);
    }
}
```

OUTPUT-

```
Student Name -Aanshika
Student C.R.N - 2021001
Student U.R.N - 2004878
Student Department -Inforamation Technology
Student CGPA -8.89
```

PRACTICAL 2

Write a program of type conversion

Java provides various data types just like any other dynamic languages such as Boolean, char, int, unsigned int, signed int, float, double, long, etc

Total providing 7 types where every datatype acquires different space while storing in memory.

When you assign a value of one data type to another, the two types might not be compatible with each other.

If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion, and if not then they need to be cast or converted explicitly.

For example, assigning an int value to a long variable.

Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

The two data types are compatible.

When we assign a value of a smaller data type to a bigger data type.

For Example, in java, the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

CODE-

```
public class Main
{
    public static void main(String[] args) {
        //type conversion int to long to float
        int i=2000;
        long l=i;
        float f=l;
```

```
        System.out.println("Int value - "+i);  
        System.out.println("Long value - "+l);  
        System.out.println("Float value - "+f);  
    }  
}
```

OUTPUT-

```
Int value -2000  
Long value -2000  
Float value -2000.0
```


PRACTICAL 3

Write a program of Array 1D and 2D

Array- Array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Types of Array

There are two types of array.

- o Single Dimensional Array
- o Multidimensional

Array Code

```
public class Main

{

    public static void main(String[] args) {

        int a[]=new int[5];//declaration and instantiation

        a[0]=10;//initialization

        a[1]=20;

        a[2]=70;

        a[3]=40;

        a[4]=50;

        System.out.println("Name- Aanshika");
```

```
System.out.println("Crn-2021001");
```

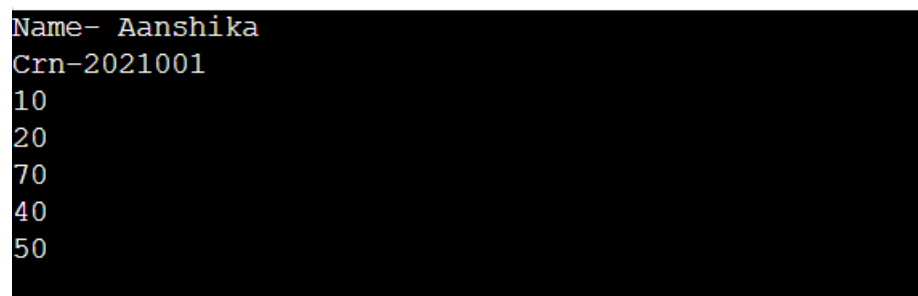
```
for(int i=0;i<a.length;i++)
```

```
System.out.println(a[i]);
```

```
    }
```

```
}
```

Output

A screenshot of a terminal window with a black background and white text. The output of the program is displayed line by line: "Name- Aanshika", "Crn-2021001", "10", "20", "70", "40", and "50".

```
Name- Aanshika
Crn-2021001
10
20
70
40
50
```

2D Array

Code

```
public class Main
```

```
{
```

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

```
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
```

```
System.out.println("Name- Aanshika");
```

```
System.out.println("Crn-2021001");
```

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

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

```
        System.out.print(arr[i][j]+"
```

```
    ");
```

```
}  
  
System.out.println();  
  
}  
  
}  
  
}
```

Output

```
Name- Aanshika  
Crn-2021001  
1 2 3  
2 4 5  
4 4 5
```

PRACTICAL 4

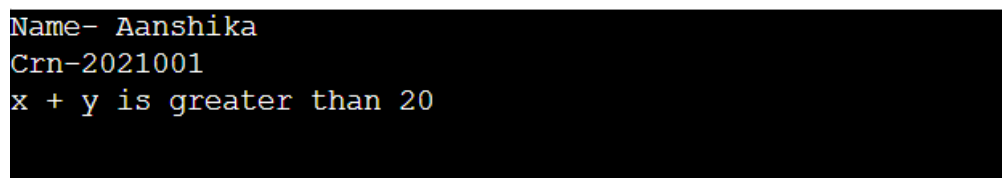
Write a program of Decision Statements

Code

```
public class Main
{
    public static void main(String[] args) {
        int x = 10;
        int y = 12;
        System.out.println("Name- Aanshika");
        System.out.println("Crn-2021001");

        if(x+y < 10) {
            System.out.println("x + y is less than 10");
        } else {
            System.out.println("x + y is greater than 20");
        }
    }
}
```

Output

A screenshot of a terminal window with a black background and white text. The output consists of three lines: "Name- Aanshika", "Crn-2021001", and "x + y is greater than 20".

```
Name- Aanshika
Crn-2021001
x + y is greater than 20
```

Switch Statement

Code

```
public class Main
{
    public static void main(String[] args) {
        int num = 4;
        switch (num){
            case 0:
                System.out.println("Aanshika");
                break;
            case 1:
                System.out.println("Anamika");
                break;
            case 2:
                System.out.println("Vishakha");
                break;
            case 3:
                System.out.println("Kritika");
                break;

            default:
                System.out.println("NAME DOESNOT EXIST");
        }
    }
}
```

Output

Name- Aanshika

Crn-2021001

NAME DOESNOT EXIST

PRACTICAL 5

Write a program of Control Statements

For Loop

Code

```
public class Main
{
    public static void main(String[] args) {
        System.out.println("Name- Aanshika");
        System.out.println("Crn-2021001");
```

```
int sum = 0;
for(int j = 1; j<=10; j++) {
    sum = sum + j;
}

System.out.println("The sum of first 10 natural numbers is " + sum);
}
}
```

Output

```
Name- Aanshika
Crn-2021001
The sum of first 10 natural numbers is 55
```

While Loop

Code

```
public class Main
```

```

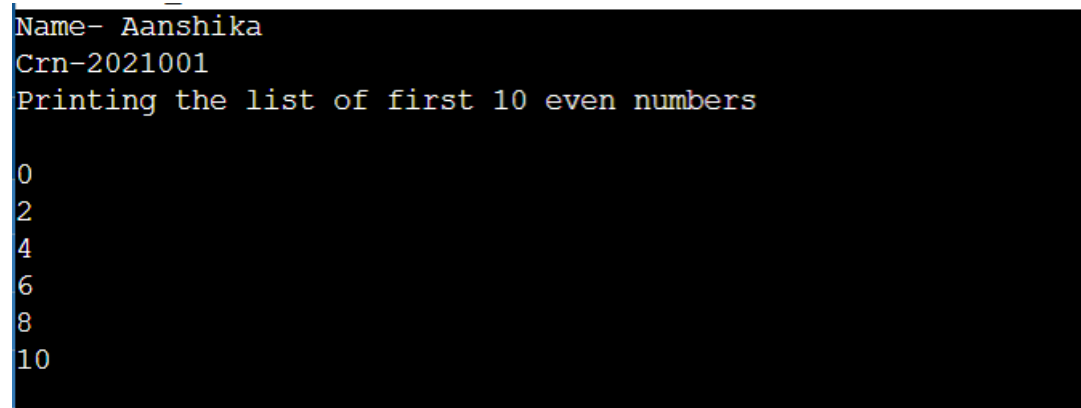
{
    public static void main(String[] args) {
        System.out.println("Name- Aanshika");
        System.out.println("Crn-2021001");

        int i = 0;

        System.out.println("Printing the list of first 10 even numbers \n");
        while(i<=10) {
            System.out.println(i);
            i = i + 2;
        }
    }
}

```

Output



```

Name- Aanshika
Crn-2021001
Printing the list of first 10 even numbers
0
2
4
6
8
10

```

Break Statement

Code

```

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

```



```
System.out.println("Name- Aanshika");  
System.out.println("Crn-2021001");
```

a:

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

b:

```
for(int j = 0; j<=15;j++) {
```

c:

```
for (int k = 0; k<=20; k++) {
```

```
System.out.println(k);
```

```
if(k==5) {
```

```
break a;
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

Output

```
Name- Aanshika  
Crn-2021001  
0  
1  
2  
3  
4  
5
```

Continue Statement

Code

```
public class Main
{
    public static void main(String[] args) {
        System.out.println("Name- Aanshika");
        System.out.println("Crn-2021001");

        for(int i = 0; i<= 2; i++) {

            for (int j = i; j<=5; j++) {

                if(j == 4) {
                    continue;
                }

                System.out.println(j);
            }
        }
    }
}
```

Output

Name- Aanshika

Crn-2021001

0

1

2

3

5

1

2

3

5

2

3

5

PRACTICAL 6

Write a program of Recursion

Recursion - Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method. It makes the code compact but complex to understand.

Code-

```
public class Main
{
    static int factorial( int n ) {
        if (n != 0) // termination condition
            return n * factorial(n-1); // recursive call
        else
            return 1;
    }

    public static void main(String[] args) {
        int number = 10, result;
        result = factorial(number);
        System.out.println(number + " factorial = " + result);
    }
}
```

Output

```
10 factorial = 3628800
```

PRACTICAL 7

Write a program of method overloading by passing object as arguments

Method overloading - If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Code –

```
class Product {  
    public int multiply(int a, int b)  
    {  
        int prod = a * b;  
        return prod;  
    }  
    public int multiply(int a, int b, int c)  
    {  
        int prod = a * b * c;  
        return prod;  
    }  
}
```

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

```
{  
    Product ob = new Product();  
    int prod1 = ob.multiply(1, 2,3)  
    System.out.println(  
        "Product of the two integer value:" + prod1);  
    int prod2 = ob.multiply(1, 2, 3);  
    System.out.println(  
        "Product of the three integer value:" + prod2);  
}  
}
```

Output

```
Product of the two integer value :2  
Product of the three integer value :6
```

Practical 7

Method Overloading

Method Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters, or a mixture of both.

Method overloading is also known as **Compile-time Polymorphism**, *Static Polymorphism*, or **Early binding** in Java. In Method overloading compared to parent argument, child argument will get the highest priority.

Code->

```
public class ComplexNumber
{
    int real, image;

    public ComplexNumber(int r, int i)
    { this.real = r;
      this.image = i;
    }

    public void showC()
    { System.out.print(this.real + " +i" + this.image);
    }

    public static ComplexNumber add(ComplexNumber n1, ComplexNumber n2) { ComplexNumber res = new ComplexNumber(0, 0);
    res.real = n1.real + n2.real; res.image = n1.image + n2.image;
    return res;
    }

    public static void main(String arg[]){
    System.out.println("Aditi Sharma");
    {

    ComplexNumber c1 = new ComplexNumber(4, 5);
    ComplexNumber c2 = new ComplexNumber(10, 5);
    System.out.print("first Complex number: ");
    c1.showC();

    System.out.print("\nSecond Complex number: ");
    c2.showC();
    ComplexNumber res = add(c1, c2);
    System.out.print("\nAddition is :");
    res.showC();
    }
    }
    }
    }
```


Output:-

Output

```
^ java -cp /tmp/5qPcoTm2PU ComplexNumber  
Aditi Sharma  
first Complex number: 4 +i5  
Second Complex number: 10 +i5  
Addition is :14 +i10|
```

Practical No:- 8

Constructor Overloading by passing objects as arguments

```
import
java.util.*;
class area {
double length, breadth;

area()
{ length =
0; breadth
= 0;
}

area(int l, int b)
{ length = l; breadth = b;
}

area(int l)
{
length = breadth = l;
}

double area()
{
return length * breadth;
}
}

public class Rectangle
{

public static void main(String args[])
{System.out.println("Aditi Sharma");
{ area r1 = new
area(); area r2 =
new area(5); area r3
= new area(2, 6);
System.out.println("area of rectangle 1 is: " +
r1.area());    System.out.println("area    of
rectangle    2    is:    "    +    r2.area());
System.out.println("area of rectangle 3 is: " +
r3.area());
}
}
}
```

Output:-

```
Aditi Sharma  
area of rectangle 1 is: 0.0  
area of rectangle 2 is: 25.0  
area of rectangle 3 is: 12.0
```

Practical No:- 9

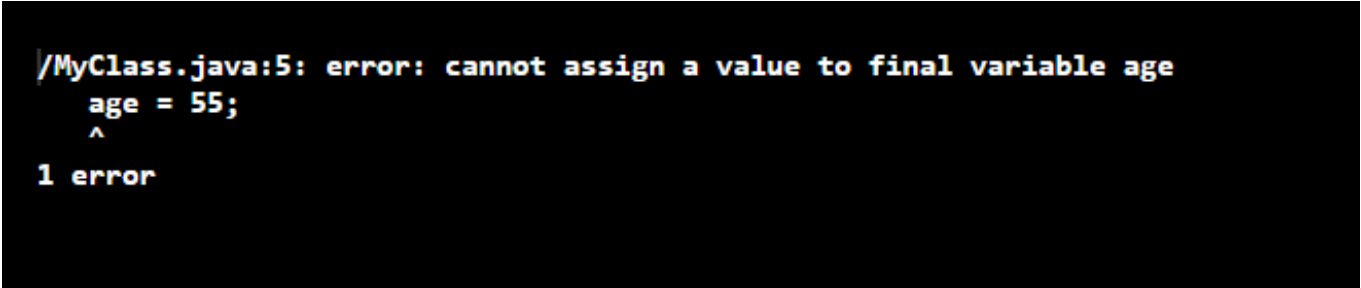
Various access control and usage of static, final and finalize ()

```
public class MyClass
{ final int age = 18;

void showAge()
{ age = 55;
}

public static void main(String[] args)
{System.out.println("Aditi Sharma");
{ MyClass student = new
MyClass(); student.showAge();
}
}
}
```

Output:->

A screenshot of a Java compiler error message displayed on a black background with white and yellow text. The message indicates an error on line 5 of a file named MyClass.java, stating that a final variable 'age' cannot be assigned a new value. The error points to the line 'age = 55;' with a caret under the 'age' variable. Below the message, it says '1 error'.

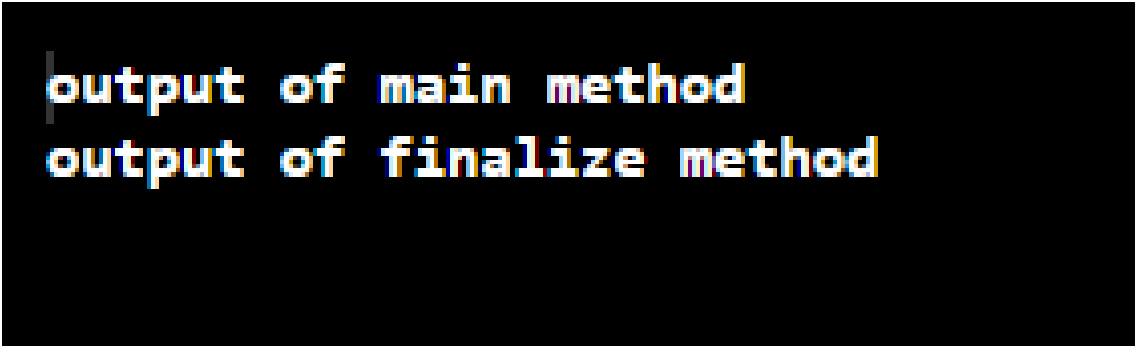
```
/MyClass.java:5: error: cannot assign a value to final variable age
    age = 55;
    ^
1 error
```

finalize:

Code->

```
public class finalize {  
    public static void main(String[] args) {  
        finalize str2 = new finalize();  
        str2 = null;  
  
        System.gc();  
        System.out.println("output of main method");  
    }  
  
    protected void finalize() {  
        System.out.println("output of finalize  
method");  
    }  
}
```

Output->

A screenshot of a terminal window with a black background. It displays two lines of output in a yellow, monospaced font. The first line is "output of main method" and the second line is "output of finalize method".

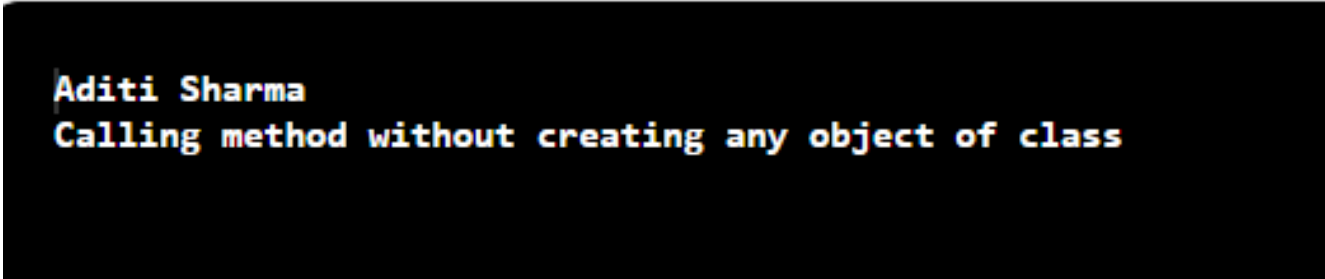
```
output of main method  
output of finalize method
```

Static:

Code->

```
public class static1 {  
    // static method  
    static void show()  
    {  
        System.out.println("Calling method without creating any object of class");  
    }  
  
    public static void main(String[] args)  
    {System.out.println("Aditi Sharma");  
    { show();  
    }  
    }  
}
```

Output:-

A screenshot of a terminal window with a black background and white text. The output consists of two lines: "Aditi Sharma" on the first line and "Calling method without creating any object of class" on the second line.

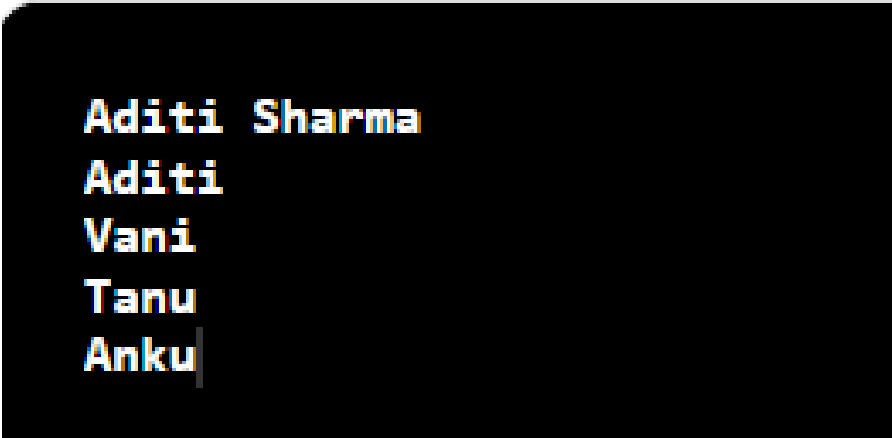
```
Aditi Sharma  
Calling method without creating any object of class
```

Practical No :- 10

Command line arguments

```
public class Main {  
    public static void main(String args[])  
    {System.out.println("Aditi Sharma");  
    { for (int i = 0; i < args.length; i++)  
    {  
        System.out.println("args[" + i + "]: " + args[i]);  
    }  
    }  
    }  
}
```

Output->

A screenshot of a terminal window with a black background. The output of the Java program is displayed in a yellow, monospaced font. The text is arranged vertically, showing the first argument 'Aditi Sharma' on the first line, followed by the subsequent arguments 'Aditi', 'Vani', 'Tanu', and 'Anku' on separate lines. A vertical cursor is visible at the end of the last line.

```
Aditi Sharma  
Aditi  
Vani  
Tanu  
Anku
```

Practical No :- 11

Inheritance in Java

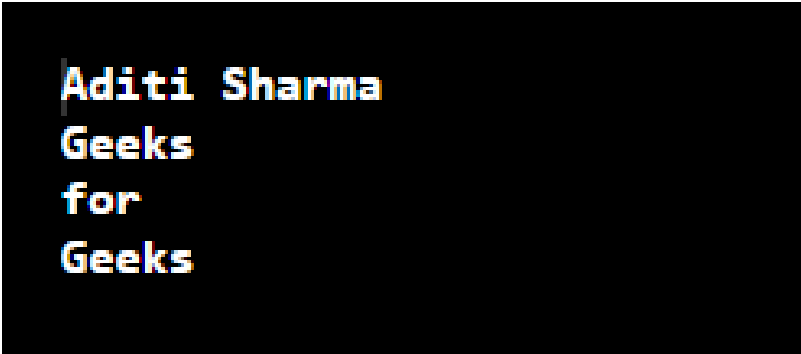
```
import java.io.*;
import
java.lang.*;
import java.util.*;

class one {
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}

class two extends one {
    public void
    print_for()
    { System.out.println("for"); }
}

// Driver class
public class Main {
    public static void main(String[] args)
    {System.out.println("Aditi Sharma");
    {
        two g = new
        two();
        g.print_geek();
        g.print_for();
        g.print_geek();
    }
}
}
```

Output->

A screenshot of a terminal window with a black background. The output text is displayed in a yellow, monospaced font. The text is arranged in four lines: "Aditi Sharma", "Geeks", "for", and "Geeks".

Aditi Sharma
Geeks
for
Geeks

Practical No :- 12

Method Overriding

```
class Parent { void fun() {  
    System.out.println("parent's fun");  
}  
}  
  
class Child extends Parent { @Override  
    void fun()  
    { System.out.println("child's fun");  
    }  
}  
  
class GrandChild extends Child {  
    @Override void fun()  
    { System.out.println("Grandchild's fun");  
    }  
}  
  
public class overriding {  
    public static void main(String args[])  
    {System.out.println("Aditi Sharma");  
    { Parent p = new Child();  
    Parent ch = new  
    GrandChild(); p.fun();  
    ch.fun();  
  
    }  
    }  
}
```

Output:-

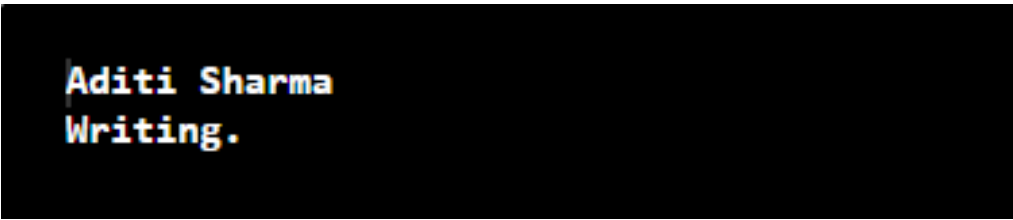
```
Aditi Sharma  
child's fun  
Grandchild's fun
```

Practical No :-13

Abstract classes

```
abstract class parent {  
    abstract void write();  
}  
  
class child extends parent {  
    @Override void write()  
    { System.out.println("Writing.    ");  
    }  
}  
  
public class Abstract{  
  
    public static void main(String args[])  
    {System.out.println("Aditi Sharma");  
    { child a = new  
child(); a.write();  
    }  
    }  
}
```

Output:-

A screenshot of a terminal window with a black background. The text "Aditi Sharma" is displayed on the first line, and "Writing." is displayed on the second line. Both lines of text are in a light blue or cyan monospaced font.

```
Aditi Sharma  
Writing.
```

Practical No :-14

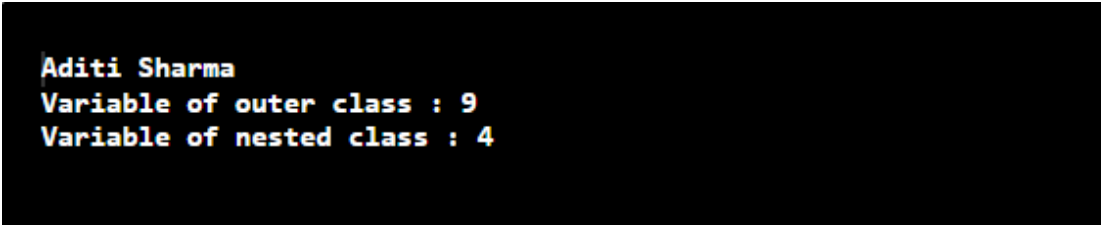
Nested class

```
class outer { int x = 9;
```

```
class nested { int y = 4;  
}  
}
```

```
public class nestedClass {  
public static void main(String[] args)  
{System.out.println("Aditi Sharma");  
{ outer otr = new outer();  
outer.nested nstd = otr.new nested();  
  
System.out.println("Variable of outer class : " + otr.x);  
System.out.println("Variable of nested class : " + nstd.y);  
}  
}  
}
```

Output:-

A screenshot of a terminal window with a black background and white text. The output of the Java program is displayed in three lines: 'Aditi Sharma', 'Variable of outer class : 9', and 'Variable of nested class : 4'.

```
Aditi Sharma  
Variable of outer class : 9  
Variable of nested class : 4
```

Practical No : 15

Constructor chaining

Input Code:

```
public class Constructor_Chaining { Constructor_Chaining(){
    this(40,50);

    System.out.println ("Base class Default constructor called"); }
    Constructor_Chaining(int x, int y){
        System.out.println ("Base class Parameterized constructor");
    }

    static class Prototype extends Constructor_Chaining{
        Prototype(){
            this("Java", "Lang");

            System.out.println ("Derived class default constructor called
(Abhishek+Aanshika+Aditi+Abhinav)");
        }

        Prototype(String str1, String str2){ super();
            System.out.println ("Derived class parameterized constructor"); }
    }

    public static void main(String[] args) {
        Prototype obj = new Prototype(); }
}
```

Output :

```
/Users/avhi/IdeaProjects/Student/out/production/Student Constructor_Chaining
Base class Parameterized constructor
Base class Default constructor called
Derived class parameterized constructor
Derived class default constructor called (Abhishek+Aanshika+Aditi+Abhinav)

Process finished with exit code 0
```

Practical No : 16

Importing classes from user defined package and creating packages using access protection

::Importing classes from user defined package

Input Code:

```
package FirstPackage1; public
class MyClass {
    int a;

    public void set_value(int n){ a
        = n; }
    public void show(){
        System.out.println("(Abhishek+Abhinav+Aanshika+Aditi)" + "The
            value of a is : "+a); }
}
```

```
package MyPackage2;
import FirstPackage1.MyClass;

public class PackageDemo {
    public static void main(String[] args) {
        MyClass obj = new MyClass (); obj.set_value (10); obj.show
        ();
    } }
```

Output :

```
/Users/avhi/IdeaProjects/Student/out/production/Student MyPackage2.PackageDemo
(Abhishek+Abhinav+Aanshika+Aditi)The value of a is : 10

Process finished with exit code 0
```

::Creating packages using access protection

Input Code:

```
package Test;

public class TestClass { public int a; int b;
    private int c; public
    void show(){
        System.out.println ("The value of a is "+a);
    }

    public void method1(){
        System.out.println ("Method1 is allowed because this is a
public(Abhishek+Abhinav+Aditi+Aanshika)); }
    void method2(){
        System.out.println ("Method2 is not allowed");
    }

    private void method3(){
        System.out.println ("Method3 is not allowed"); }
}
```

```
package MainPackage; import
```

```
Test.TestClass;

public class AccessProtection {
    public static void main(String[] args) {
        TestClass obj = new TestClass ();
        obj.a = 30; // allowed because a is public obj.show ();
        //obj.b = 30; // error cannot access because b is default Modifier // obj.c =
        25; // error : cannot access because c is private
        obj.method1 (); // allowed because fun1 is public

        // obj.method2(); // error cannot access because fun2 is default //
        obj.method3(); // error cannot access because fun3 is private
    } }
}
```

Output :

```
/Users/avhi/IdeaProjects/Student/out/production/Student MainPackage.AccessProtection
The value of a is 30
Method1 is allowed because this is a public(Abhishek+Abhinav+Aditi+Aanshika)

Process finished with exit code 0
```

Practical No : 17

Interfaces, nested interfaces and use of extending interfaces

::Interfaces

Input Code:

```
interface printable{ void print();  
}  
  
public class Interfaces implements printable{  
    @Override  
    public void print() { System.out.println  
        ("Introduction  
Interfaces\n(Abhishek+Aditi+Abhinav+Aanshika)"); }  
  
    public static void main(String[] args) { Interfaces obj = new Interfaces();  
obj.print ();  
    } }
```

Output :

```
/Users/avhi/IdeaProjects/Student/out/production/Student Interfaces  
Introduction Interfaces  
(Abhishek+Aditi+Abhinav+Aanshika)  
  
Process finished with exit code 0
```


::Nested Interfaces

Input Code:

```
class A1 {  
    interface Message {  
        void msg(); }  
}  
  
class TesNestedInterface2 implements A1.Message { public void  
    msg() {  
        System.out.println ("Introduction Nested Interface \n  
(Abhishek+Aanshika+Abhinav+Aditi)");  
    }  
}  
  
public class NestedInterfaces {  
    public static void main(String[] args) {  
        A1.Message message= new TesNestedInterface2 ();  
        message.msg (); }  
}
```

Output :

```
/Users/avhi/IdeaProjects/Student/out/production/Student NestedInterfaces  
Introduction Nested Interface  
(Abhishek+Aanshika+Abhinav+Aditi)  
  
Process finished with exit code 0
```

::Use of extending interface

Input Code:

```
interface Printable{ void print();
}
interface show extends Printable{ void
    show(); }
public class InterfaceInheritance implements show{ public void print(){
    System.out.println ("Hello"); }
    public void show(){ System.out.println
("Interface\n(ABHISHEK+ABHINAV+ADITI+AANSHIKA)");
    }
    public static void main(String[] args) {
        InterfaceInheritance obj = new InterfaceInheritance (); obj.print
();
                                obj.show ();
                                } }
```

Output :

```
/Users/avhi/IdeaProjects/Student/out/production/Student InterfaceInheritance
Hello
Interface
(ABHISHEK+ABHINAV+ADITI+AANSHIKA)

Process finished with exit code 0
```

Practical No : 18

Exception Handling - using predefined exception

Input Code:

```
import java.util.Scanner;

public class BuiltException extends Throwable {

    public static <ArrayIndexOutOfBoundsException> void
main(String[] args) {
// public static void main(String args[])throws IOException{ int
marks[] = new int[2];
    marks[0] = 3;

    marks[1] = 4;

// marks[4] = 8;

    Scanner sc = new Scanner(System.in); System.out.println("Enter the
index"); int index = sc.nextInt();
    System.out.println("Enter the you want to divide with"); int number =
sc.nextInt();
    try{

        System.out.println("The value at array index entered
is:"+marks[index]);// ArrayIndexOutOfBoundsException occur
        System.out.println("The value of array = value/number
"+marks[index]/ number); // Arithmetic Exception occur
    }

    catch(ArithmeticException e){
        System.out.println("ArithmeticException occur :");
        System.out.println(e+ "\n (Abhishek+Aditi+Abhinav+Aanshika)");
    }

    catch(ArrayIndexOutOfBoundsException e){

        System.out.println("ArrayIndexOutOfBoundsException occur
:");

        System.out.println(e); }
}
```

```
        catch(Exception e){
            System.out.println("Some other exception occur :");
System.out.println(e);
        } }
    }
```

Output :

```
-classpath /Users/avhi/IdeaProjects/Student/out/production/Student BuiltException
Enter the index
1
Enter the you want to divide with
0
The value at array index entered is:4
ArithmeticException occur :
java.lang.ArithmeticException: / by zero
(Abhishek+Aditi+Abhinav+Aanshika)

Process finished with exit code 0
```

Practical No : 19

Exception Handling - user defined exception

Input Code:

```
class MyException extends Exception{ String str1;
    MyException(String str2) {
        str1=str2; }

    public String toString(){
        return ("MyException Occurred: "+str1) ;
    } }

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

        System.out.println("Starting of try block");

// I'm throwing the custom exception using throw throw
new MyException("This is My error
Message(Abhishek+Aditi+Aanshika+Abhinav)");
    }

    catch(MyException exp){
        System.out.println("Catch Block") ;
        System.out.println(exp) ; }
} }
```

Output :

```
-classpath /Users/avhi/IdeaProjects/Student/out/production/Student UserDefinedExceptions
Starting of try block
Catch Block
MyException Occurred: This is My error Message(Abhishek+Aditi+Aanshika+Abhinav)

Process finished with exit code 0
```

Practical No : 20

Multithreading by extending Threads Class

Input Code:

```
class Thread1 extends Thread{ @Override
public void run(){
    while(true){
        System.out.println ("Thread1 is Running(Abhishek+Abhinav)");
    }
}

class Thread2 extends Thread{
    @Override
    public void run() {
        while(true) {
            System.out.println ("Thread2 is Running(Aditi+Aanshika)");
        }
    }
}

public class ThreadClass {
    public static void main(String[] args) { Thread1 t1 = new Thread1 ();
    Thread2 t2 = new Thread2 (); t1.start ();
        t2.start(); }
}
```

Output :

```
Thread1 is Running(Abhishek+Abhinav)
Thread1 is Running(Abhishek+Abhinav)
Thread1 is Running(Abhishek+Abhinav)
Thread1 is Running(Abhishek+Abhinav)
Thread1 is Running(Abhishek+Abhinav)
Thread1 is Running(Abhishek+Abhinav)
Thread1 is Running(Abhishek+Abhinav)
Thread2 is Running(Aditi+Aanshika)
Thread2 is Running(Aditi+Aanshika)
Thread2 is Running(Aditi+Aanshika)
Thread2 is Running(Aditi+Aanshika)
Thread2 is Running(Aditi+Aanshika)
Thread2 is Running(Aditi+Aanshika)
```

Practical No : 21

Multithreading by implementing Runnable Interface

Input Code:

```
class ThreadRunnable1 implements Runnable{ @Override
public void run() {
    int i = 0; while(i<5){
        System.out.println ("Thread1 is Running (Abhishek+Abhinav)"); i++;
    }
} }

class ThreadRunnable2 implements Runnable{ @Override
public void run() {
    int i = 0; while(i<5){
        System.out.println ("Thread2 is Running (Aditi+Aanshika)"); i++; }
} }

public class RunnableInterface {
    public static void main(String[] args) {
        ThreadRunnable1 bullet1 = new ThreadRunnable1 (); Thread
gun1 = new Thread (bullet1); ThreadRunnable2 bullet2 = new
ThreadRunnable2 (); Thread gun2 = new Thread(bullet2);
        gun1.start ();
        gun2.start (); }
}
```


Output :

```
-classpath /Users/avhi/IdeaProjects/Student/out/production/Student RunnableInterface
Thread1 is Running (Abhishek+Abhinav)
Thread1 is Running (Abhishek+Abhinav)
Thread1 is Running (Abhishek+Abhinav)
Thread1 is Running (Abhishek+Abhinav)
Thread1 is Running (Abhishek+Abhinav)
Thread2 is Running (Aditi+Aanshika)
Thread2 is Running (Aditi+Aanshika)
Thread2 is Running (Aditi+Aanshika)
Thread2 is Running (Aditi+Aanshika)
Thread2 is Running (Aditi+Aanshika)
```

Practical No : 22

Thread life cycle

Input Code:

```
class thread implements Runnable{ @Override public
void run() {
    try{
        Thread.sleep (500);
    }
    catch (InterruptedException e){
        e.printStackTrace (); }
    System.out.println(
        "State of thread1 while it called join() method on thread2 -"
        + ThreadLifeCycle.thread1.getState()
        +"\n(Abhishek+Abhinav+Aditi+Aanshika)"); try {
        Thread.sleep (200);
    } catch (InterruptedException e){ e.printStackTrace ();
    } }
}

public class ThreadLifeCycle implements Runnable { public
    static Thread thread1;
    public static ThreadLifeCycle obj; public static void main(String[] args)
    { obj = new ThreadLifeCycle ();
        thread1 = new Thread (obj); System.out.println(
        "State of thread1 after creating it - " + thread1.getState()); thread1.start();
    System.out.println(
        "State of thread1 after calling .start() method on it - "
```

```

        + thread1.getState());}
@Override
public void run() {

    thread myThread = new thread ();

    Thread thread2 = new Thread(myThread);
    System.out.println(
        "State of thread2 after creating it - "
            + thread2.getState()); thread2.start();
    System.out.println(
        "State of thread2 after calling .start() method on it - "
            + thread2.getState());

    try{
        Thread.sleep (100);
    }catch (InterruptedException e){ e.printStackTrace ();
    } System.out.println(
        "State of thread2 after calling .sleep() method on it - " +
thread2.getState());
    try{
        thread2.join ();
    }

    catch (InterruptedException e) { e.printStackTrace ();
System.out.println (
        "State of thread2 when it has finished it's execution - " +
thread2.getState ());
    } } }

```

Output :

```

-classpath /Users/avhi/IdeaProjects/Student/out/production/Student ThreadLifeCycle
State of thread1 after creating it - NEW
State of thread1 after calling .start() method on it - RUNNABLE
State of thread2 after creating it - NEW
State of thread2 after calling .start() method on it - RUNNABLE
State of thread2 after calling .sleep() method on it - TIMED_WAITING
State of thread1 while it called join() method on thread2 -WAITING
(Abhishek+Abhinav+Aditi+Aanshika)

Process finished with exit code 0

```

Practical No : 25

Event Handling

We can put the event handling code into one of the following places:

- 1. Within class**
- 2. Other class**
- 3. Anonymous class**

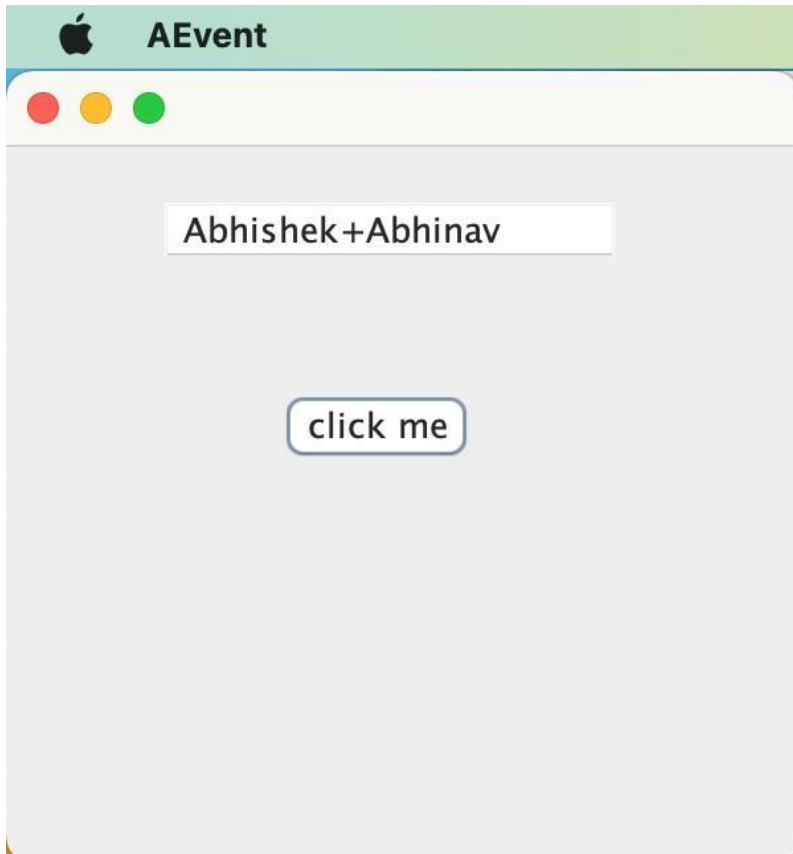
::Event handling Within Class Input Code:

```
import java.awt.*;

import java.awt.event.ActionEvent; import java.awt.event.ActionListener;
class AEvent extends Frame implements ActionListener{ TextField tf;
    AEvent() {
        tf = new TextField ();
        tf.setBounds (60, 50, 170, 20); Button b = new Button ("click
me"); b.setBounds (100, 120, 80, 30);
        b.addActionListener (this); add (b);
        add (tf);
        setSize (300, 300); setLayout (null); setVisible (true);
    }
}
```

```
public void actionPerformed(ActionEvent e){ tf.setText  
    ("Abhishek+Abhinav"); }  
public static void main(String args[]){ new AEvent ();  
} }
```

Output :



::Event handling by Other Class

Input Code:

```
import java.awt.*; import java.awt.event.*;
public class AEvent2 extends Frame {
    public Label textField;
    TextField tf;

    AEvent2() {
        tf = new TextField(); tf.setBounds(100,
        50, 170, 20); Button b = new
        Button("Click Here"); b.setBounds(100,
        120, 80, 30); Other o = new Other(this);
        b.addActionListener(o);
        add(tf);

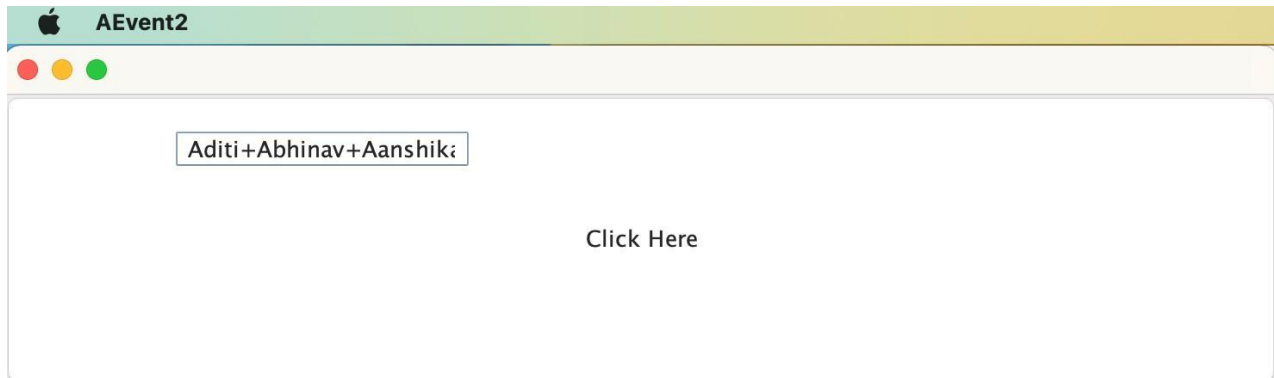
        add(b);
        setVisible(true);
    }

    public static void main(String[] args) { new
        AEvent2();
    }
}

import java.awt.event.*;
public class Other implements ActionListener { AEvent2
    obj;
    Other(AEvent2 gj) {
        this.obj = obj;
    }

    public void actionPerformed(ActionEvent e) {
    }
}
```

Output :

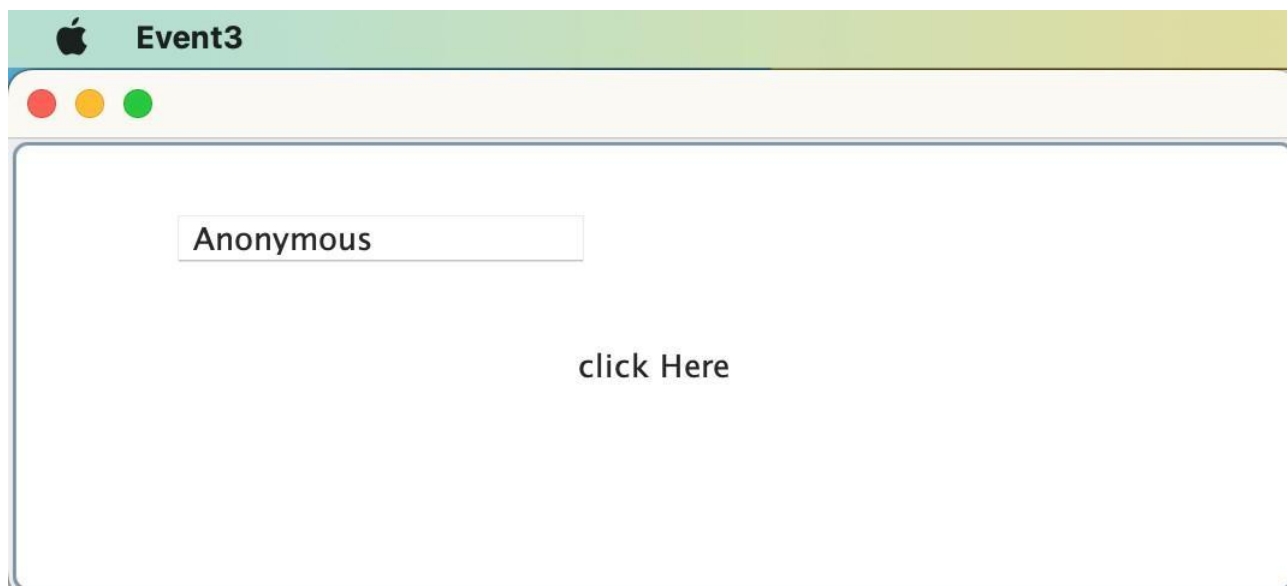


::Event handling by Anonymous Class Input

Code:

```
import javax.swing.*; import java.awt.*; import java.awt.event.*; public
class Event3 extends Frame{ TextField textField;
    Event3() {
        textField = new TextField ();
        textField.setBounds (70,60,170,20); Button button = new Button
("click Here"); button.setBounds (70,120,80,30);
        button.addActionListener (e -> textField.setText ("Anonymous"));
add(textField);
        add(button);
        setVisible (true); }
    public static void main(String[] args) { new Event3 ();
    } }
```

Output :



Practical No : 28

String class and its methods

Input Code:

```
import java.util.*; import java.util.*; public
class StringClass {
    public static void main(String[] args) { String s
        = "Abhishek";

        System.out.println ("String length = "+s.length ()); System.out.println
        ("Character at 4th position = "+s.charAt
(4));
        System.out.println
        ("Substring "+s.substring
(2)); String s2 = "Gne
        College";
        System.out.println ("Index of C
        character = "+s2.indexOf("C"));

//Concatenates string2 to the of string1 String
    s3 = "Aditi";
    String s4 = " Aanshika";

    System.out.println ("Concatenated string = "+s3.concat (s4));

//Replacing characters String
    str1 = "dbhinav";
    System.out.println ("Original String "+str1); String str2 = "Abhinav";
    System.out.println ("Replaced d with A -> "+str2);

//Checking equality of Strings

    Boolean out = "Abhishek".equals ("abhishek"); System.out.println
```

```

("Checking Equality " +out); out
"Abhishek".equals ("Abhishek"); System.out.println ("Checking Equality " +
out);
    out = "Abhishek".equals ("abhsek"); System.out.println ("Checking
Equality "+out);
//If ASCII difference is zero then the two strings are similar int
    out1 = s3.compareTo (s4);
    System.out.println ("The difference between ASCII value is =
"+out1);
    String word1 = "GneCollege";

    System.out.println ("Changing to lower Case "+word1.toLowerCase
());
//converting the word

    String word2 = "GneCollege";

    System.out.println ("Changing to Upper Case "+word2.toUpperCase
());
//Trimming the word

    String word4 = " Name and Fame"; System.out.println
("Trim the word "+word4.trim ());
} }

```

Output :

```

-classpath /Users/avhi/IdeaProjects/Student/out/production/Student StringClass
String length = 8
Character at 4th position = s
Substring hishek
Index of C character = 4
Concatenated string = Aditi Aanshika
Original String dbhinav
Replaced d with A -> Abhinav
Checking Equality false
Checking Equality true
Checking Equality false
The difference between ASCII value is = 33
Changing to lower Case gnecollege
Changing to Upper Case GNECOLLEGE
Trim the word Name and Fame

Process finished with exit code 0

```

Practical No : 29

StringBuffer class and its methods

Input Code:

```
public class StringBufferClass {  
    public static void main(String[] args) {  
        //append() Method System.out.println("append  
        Method:");  
        StringBuffer sb1 = new StringBuffer("Aditi ");  
        sb1.append("Aanshika"); System.out.println(sb1);  
        // insert() Method System.out.println("insert  
        Method:");  
        StringBuffer sb2 = new StringBuffer("Hello");  
        sb2.insert(1, "java"); System.out.println(sb2);  
        //replace() Method System.out.println("replace  
        method:");  
        StringBuffer sb3 = new StringBuffer("Hello");  
        sb3.replace(1, 2, "java"); System.out.println(sb3);  
        //capacity Method System.out.println("capacity()  
        method:"); StringBuffer sb4 = new  
        StringBuffer();  
        System.out.println(sb4.capacity());  
        //default 16  
        sb4.append("Abhishek");  
  
        System.out.println(sb4.capacity()); sb4.append("cricket  
        is my favourite game");  
        System.out.println(sb4.capacity());  
        System.out.println("Introduction insureCapacity:");  
        sb4.ensureCapacity(10);  
        System.out.println(sb4.capacity());
```

```
sb4.ensureCapacity(50);
System.out.println(sb4.capacity());
```

```
//delete() method System.out.println("delete
method:");
StringBuffer sb5 = new StringBuffer("Abhinav");
sb5.delete(3, 5);
System.out.println(sb5);

//reverse() method

System.out.println("reverse method:(reverse of Abhishek)");
StringBuffer sb6 = new StringBuffer("Abhishek"); sb6.reverse();
System.out.println(sb6);

}}
```

Output :

```
-classpath /Users/avhi/IdeaProjects/Student/out/production/Student StringBufferClass
append Method:
Aditi Aanshika
insert Method:
Hjavaello
replace method:
Hjavallo
capacity() method:
16
16
36
Introduction insureCapacity:
36
74
delete method:
Abhav
reverse method:(reverse of Abhishek)
kehsihbA

Process finished with exit code 0
```

