

# **JAVA PROGRAMMING LAB**

## **M.Sc (COMPUTER SCIENCE)**

### **SEMESTER-I, PAPER-VII**

#### **Lesson Writer:**

Dr. U. Surya Kameswari  
Asst. Professor  
Dept. of CS&E  
Acharya Nagarjuna University  
Nagarjunanagar – 522 510

#### **Editor**

**Dr. Kampa Lavanya**  
Asst. Professor  
Dept. of CS&E  
Acharya Nagarjuna University  
Nagarjunanagar – 522 510

**Director, I/c.**  
**Prof. V. Venkateswarlu**  
M.A., M.P.S., M.S.W., M.Phil., Ph.D.  
Professor  
Centre for Distance Education  
Acharya Nagarjuna University  
Nagarjuna Nagar 522 510

Ph: 0863-2346222, 2346208  
0863- 2346259 (Study Material)  
Website [www.anucde.info](http://www.anucde.info)  
E-mail: [anucdedirector@gmail.com](mailto:anucdedirector@gmail.com)

# **M.Sc Computer Science**

**First Edition : 2025**

**No. of Copies :**

**© Acharya Nagarjuna University**

**This book is exclusively prepared for the use of students of M.Sc (Computer Science), Centre for Distance Education, Acharya Nagarjuna University and this book is meant for limited circulation only.**

**Published by:**

*Director I/c*  
**Prof. V. Venkateswarlu,**  
M.A., M.P.S., M.S.W . M.Phil., Ph.D.  
**Centre for Distance Education,**  
**Acharya Nagarjuna University**

***Printed at:***

## FOREWORD

*Since its establishment in 1976, Acharya Nagarjuna University has been forging ahead in the path of progress and dynamism, offering a variety of courses and research contributions. I am extremely happy that by gaining 'A+' grade from the NAAC in the year 2024, Acharya Nagarjuna University is offering educational opportunities at the UG, PG levels apart from research degrees to students from over 221 affiliated colleges spread over the two districts of Guntur and Prakasam.*

*The University has also started the Centre for Distance Education in 2003-04 with the aim of taking higher education to the door step of all the sectors of the society. The centre will be a great help to those who cannot join in colleges, those who cannot afford the exorbitant fees as regular students, and even to housewives desirous of pursuing higher studies. Acharya Nagarjuna University has started offering B.Sc., B.A., B.B.A., and B.Com courses at the Degree level and M.A., M.Com., M.Sc., M.B.A., and L.L.M., courses at the PG level from the academic year 2003-2004 onwards.*

*To facilitate easier understanding by students studying through the distance mode, these self-instruction materials have been prepared by eminent and experienced teachers. The lessons have been drafted with great care and expertise in the stipulated time by these teachers. Constructive ideas and scholarly suggestions are welcome from students and teachers involved respectively. Such ideas will be incorporated for the greater efficacy of this distance mode of education. For clarification of doubts and feedback, weekly classes and contact classes will be arranged at the UG and PG levels respectively.*

*It is my aim that students getting higher education through the Centre for Distance Education should improve their qualification, have better employment opportunities and in turn be part of country's progress. It is my fond desire that in the years to come, the Centre for Distance Education will go from strength to strength in the form of new courses and by catering to larger number of people. My congratulations to all the Directors, Academic Coordinators, Editors and Lesson-writers of the Centre who have helped in these endeavors.*

*Prof. K. Gangadhara Rao  
M.Tech., Ph.D.,  
Vice-Chancellor I/c  
Acharya Nagarjuna University*

**M.Sc. Computer Science**  
**Semester-I, Paper-VI**  
**DATA STRUCTURES USING C LAB**

**Lab Cycle**

1. Write a Java Program to define a class, describe its constructor, overload the constructors and instantiate its object.
2. Write a Java Program to define a class, define instance methods for setting and retrieving values of instance variables and instantiate its object
3. Write a java program to practice using String class and its methods
4. Write a java program to implement inheritance and demonstrate use of method overriding
5. Write a java program to implement multilevel inheritance by applying various access controls to its data members and methods.
6. Write a program to demonstrate use of implementing interfaces
7. Design a Java interface for ADT Stack. Develop two different classes that implement this interface, one using array and the other using linked-list. Provide necessary exception handling in both the implementations.
8. Write a Java program to implement the concept of importing classes from user defined package and creating packages
9. Write a program to implement the concept of threading by implementing Runnable Interface
10. write a java program to store and read objects from a file
11. Write a Java program that displays the number of characters, lines and words in a text file.
12. write a java program to illustrate object serialization
13. Create a java program to illustrate user defined exception
14. Write a java program to create a thread using runnable interface
15. Write a Java program that creates three threads. First thread displays "Good Morning" every one second, the second thread displays "Hello" every two seconds and the third thread displays "Welcome" every three seconds
16. Write an applet To create multiple threads that correctly implements producer consumer problem using the concept of Inter thread communication
17. Write an applet To handling the mouse events
18. Write a Program That works as a simple calculator using Grid layout to arrange buttons for the digits and +,-,\* % operations. Add a text field to print the result.
19. Build and run "Celsius Converter" sample application using swings
20. Develop an applet that receives an integer in one text field, and computes its factorial Value and returns it in another text field, when the button named "Compute" is clicked

# **JAVA PROGRAMMING LAB**

## **OBJECTIVES:**

The objective of this lab is to master the JAVA PROGRAMMING concepts and to learn how to work with real time applications using JAVA PROGRAMMING. These programs are widely used in most real-time scenarios. After the end of lab, students will be able know the complete practical exposure on JAVA PROGRAMMING.

## **STRUCTURE**

1. Write a Java Program to define a class, describe its constructor, overload the constructors and instantiate its object.
2. Write a Java Program to define a class, define instance methods for setting and retrieving values of instance variables and instantiate its object
3. Write a java program to practice using String class and its methods
4. Write a java program to implement inheritance and demonstrate use of method overriding
5. Write a java program to implement multilevel inheritance by applying various access controls to its data members and methods.
6. Write a program to demonstrate use of implementing interfaces
7. Design a Java interface for ADT Stack. Develop two different classes that implement this interface, one using array and the other using linked-list. Provide necessary exception handling in both the implementations.
8. Write a Java program to implement the concept of importing classes from user defined package and creating packages
9. Write a program to implement the concept of threading by implementing Runnable Interface
10. write a java program to store and read objects from a file
11. Write a Java program that displays the number of characters, lines and words in a text file.
12. write a java program to illustrate object serialization
13. Create a java program to illustrate user defined exception
14. Write a java program to create a thread using runnable interface
15. Write a Java program that creates three threads. First thread displays “Good Morning” every one second, the second thread displays “Hello” every two seconds and the third thread displays “Welcome” every three seconds

16. Write an applet To create multiple threads that correctly implements producer consumer problem using the concept of Inter thread communication
17. Write an applet To handling the mouse events
18. Write a Program That works as a simple calculator using Grid layout to arrange buttons for the digits and +,-,\* % operations. Add a text field to print the result.
19. Build and run "CelsiusConverter" sample application using swings
20. Develop an applet that receives an integer in one text field, and computes its factorial Value and returns it in another text field, when the button named "Compute" is clicked

**LAB EXERCISE 1:**

**Write a Java Program to define a class, describe its constructor, overload the constructors and instantiate its object.**

---

**PROGRAM DESCRIPTION**

This program demonstrates the concept of class definition, constructor usage, and constructor overloading in Java. It defines a class First with attributes a, b, and c, provides multiple constructors to initialize these attributes in different ways: a default constructor, a constructor with two parameters, and a constructor with three parameters. The program highlights the flexibility of constructor overloading by instantiating objects of the First class with varying levels of detail, showcasing how object initialization can be tailored to specific requirements.

**SOURCE CODE:**

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

class First          // define a class
{
    int a,b,c;

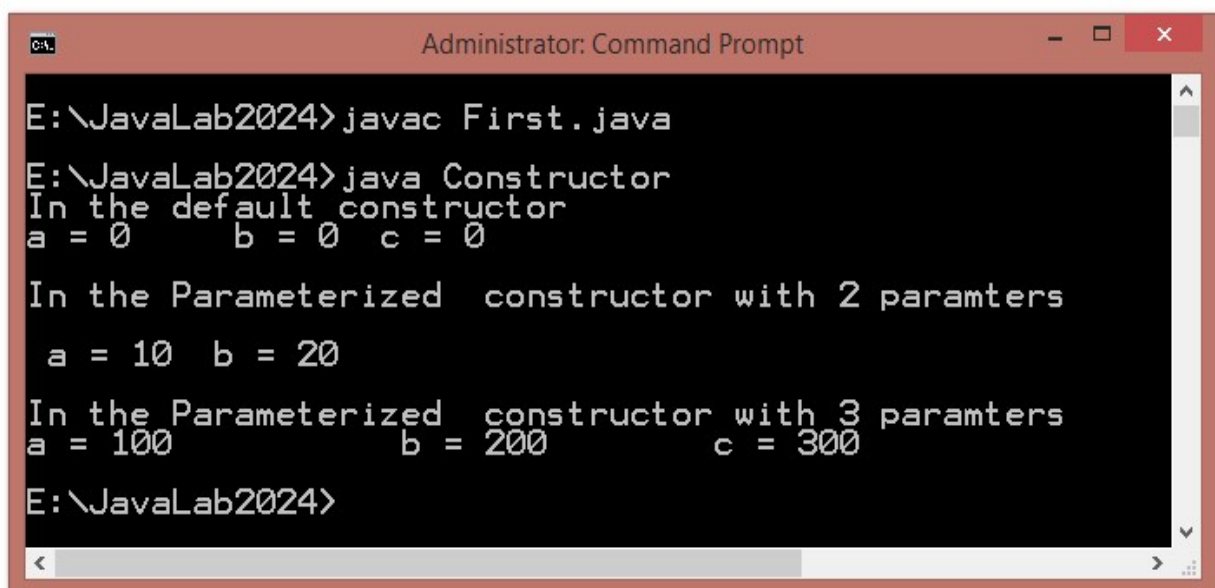
    First()          // default constructor
    {
        a=0;
        b=0;
        c=0;
        System.out.println("In the default constructor");
        System.out.println("a = " + a + " \t " + " b = " + b + "\t" + " c = " + c);
    }

    First(int a,int b)    // parameterized constructor
    {
        this.a=a;
```

```
        this.b=b;
        System.out.println("\nIn the Parameterized constructor with 2 paramters");
        System.out.println("\n a = " + a + "\t" + " b = " + b );
    }

    First(int a, int b, int c)        // parameterized constructor overloaded
    {
        this.a=a;
        this.b=b;
        this.c=c;
        System.out.println("\nIn the Parameterized constructor with 3 paramters");
        System.out.println("a = " + a + " \t " + " b = " + b + "\t" + " c = " + c);
    }
}

class Constructor
{
    public static void main(String args[ ])
    {
        First f1 = new First();
        First f2 = new First(10,20);
        First f3 = new First(100,200,300);
    }
}
```

**OUTPUT:**

```
C:\>Administrator: Command Prompt

E:\JavaLab2024>javac First.java
E:\JavaLab2024>java Constructor
In the default constructor
a = 0      b = 0      c = 0

In the Parameterized constructor with 2 paramters
a = 10     b = 20

In the Parameterized constructor with 3 paramters
a = 100    b = 200    c = 300
E:\JavaLab2024>
```

**LAB EXERCISE 2:**

**Write a Java Program to define a class, define instance methods for setting and retrieving values of instance variables and instantiate its object**

---

**PROGRAM DESCRIPTION**

This program illustrates how to define a class in Java with instance variables and corresponding instance methods for setting and retrieving their values. The class, designed to encapsulate data, contains private fields to ensure encapsulation and provides public setter and getter methods to modify and access these fields. The program demonstrates object instantiation by creating an object of the class and using the setter methods to assign values to the instance variables, followed by the getter methods to retrieve and display these values. This approach emphasizes the principles of encapsulation and data hiding in object-oriented programming.

**SOURCE CODE:**

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

class Employee          // Define the class
{
    private String ename;
    private int deptno;   // Instance variables

    // Method to set the ename
    public void setename(String ename)
    {
        this.ename = ename;
    }

    // Method to get the ename
    public String getename()
    {
        return this.ename;
    }

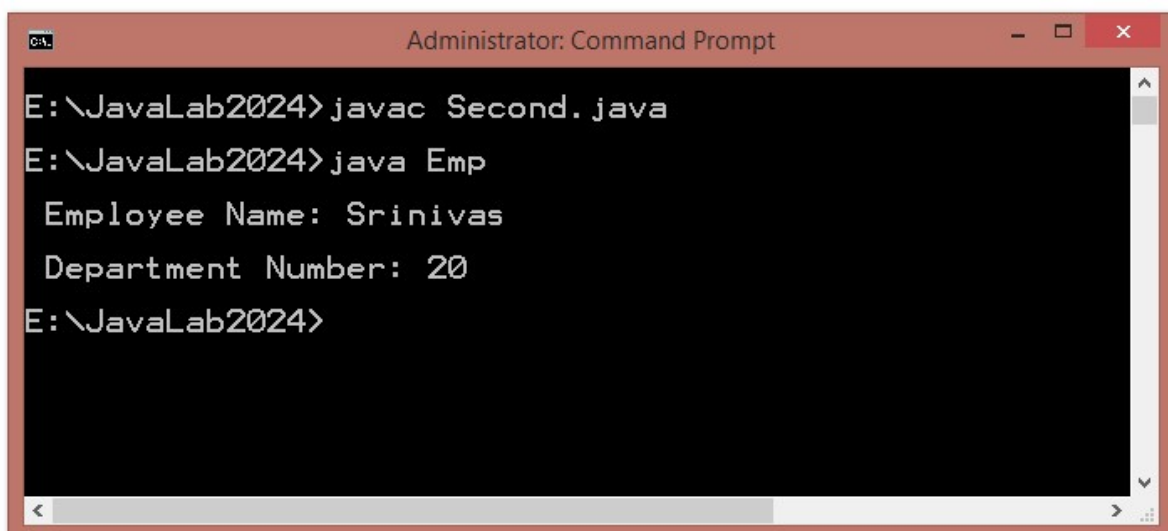
    // Method to set the deptno
    public void setdeptno(int deptno)
    {
        if (deptno >= 0)          // Validate deptno to be non-negative
            this.deptno = deptno;
        else
            System.out.println("deptno cannot be negative!");
    }
}
```

```
// Method to get the deptno
public int getdeptno()
{
    return this.deptno;
}

class Emp
{
    public static void main(String[] args)
    {
        // Instantiate an object of the Employee class
        Employee Employee = new Employee();

        // Set values using setter methods
        Employee.setename("Srinivas");
        Employee.setdeptno(20);

        // Retrieve and display values using getter methods
        System.out.println("\n Employee Name: " + Employee.getename());
        System.out.println("\n Department Number: " + Employee.getdeptno());
    }
}
```

**OUTPUT:**

```
Administrator: Command Prompt

E:\JavaLab2024>javac Second.java
E:\JavaLab2024>java Emp
Employee Name: Srinivas
Department Number: 20
E:\JavaLab2024>
```

**LAB EXERCISE 3:**

**Write a java program to practice using String class and its methods**

**PROGRAM DESCRIPTION**

This Java program demonstrates the versatility of the String class by showcasing various commonly used methods on the string "acharya nagarjuna university". It includes operations such as finding the length of the string, converting to uppercase and lowercase, finding the index of specific characters, checking prefixes and suffixes, extracting substrings, replacing words, splitting the string into an array, checking for the presence of substrings, and trimming leading and trailing spaces. The program also compares strings for equality both case-sensitively and case-insensitively. These examples highlight the power of the String class in handling and manipulating textual data in Java, making it an essential part of working with strings effectively.

**SOURCE CODE:**

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

class StringMethodsExample
{
    public static void main(String[] args)
    {
        // String element
        String str = "acharya nagarjuna university";

        // Display original string
        System.out.println("\n Original String: " + str);

        // Length of the string
        System.out.println("\n Length of String: " + str.length());

        // Convert to uppercase
        System.out.println("\n Uppercase: " + str.toUpperCase());

        // Convert to lowercase
        System.out.println("\n Lowercase: " + str.toLowerCase());

        // Find the index of a specific character
        System.out.println("\n Index of 'n': " + str.indexOf('n'));

        // Check if the string starts with a specific prefix
        System.out.println("\n Starts with 'acharya': " + str.startsWith("acharya"));

        // Check if the string ends with a specific suffix
        System.out.println("\n Ends with 'university': " + str.endsWith("university"));

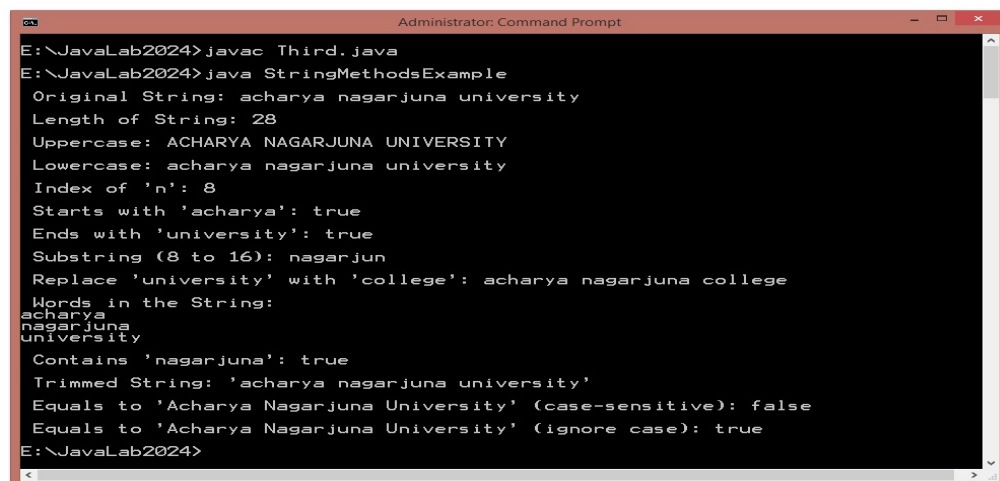
        // Extract a substring
        System.out.println("\n Substring (8 to 16): " + str.substring(8, 16));
```

```
// Replace a word in the string
System.out.println("\n Replace 'university' with 'college': " + str.replace("university",
"college"));

// Split the string into words
String[] words = str.split(" ");
System.out.println("\n Words in the String:");
for (String word : words) {
    System.out.println(word);
}

// Check if the string contains a specific sequence
System.out.println("\n Contains 'nagarjuna': " + str.contains("nagarjuna"));

// Remove leading and trailing spaces (if any)
String strWithSpaces = "  acharya nagarjuna university  ";
System.out.println("\n Trimmed String: '" + strWithSpaces.trim() + "'");
// Compare strings
String str2 = "Acharya Nagarjuna University";
System.out.println("\n Equals to 'Acharya Nagarjuna University' (case-sensitive): " +
str.equals(str2));
System.out.println("\n Equals to 'Acharya Nagarjuna University' (ignore case): " +
str.equalsIgnoreCase(str2));
}
}
```

**OUTPUT:**

```
Administrator: Command Prompt
E:\JavaLab2024>javac Third.java
E:\JavaLab2024>java StringMethodsExample
Original String: acharya nagarjuna university
Length of String: 28
Uppercase: ACHARYA NAGARJUNA UNIVERSITY
Lowercase: acharya nagarjuna university
Index of 'n': 8
Starts with 'acharya': true
Ends with 'university': true
Substring (8 to 16): nagarjun
Replace 'university' with 'college': acharya nagarjuna college
Words in the String:
acharya
nagarjuna
university
Contains 'nagarjuna': true
Trimmed String: 'acharya nagarjuna university'
Equals to 'Acharya Nagarjuna University' (case-sensitive): false
Equals to 'Acharya Nagarjuna University' (ignore case): true
E:\JavaLab2024>
```

**LAB EXERCISE 4:**

**Write a java program to implement inheritance and demonstrate use of method overriding**

**PROGRAM DESCRIPTION:**

This program demonstrates the concept of inheritance and method overriding in Java. The Animal class serves as the parent class with a generic sound() method, while the Dog and Cat classes inherit from Animal and override the sound() method to provide specific implementations ("Dog barks" and "Cat meows"). The main method creates objects of these classes to show how the overridden methods are called. Additionally, it illustrates polymorphism by using a reference of the parent class (Animal) to call the overridden methods of the child classes (Dog and Cat). This program highlights key object-oriented principles like inheritance, overriding, and polymorphism.

**SOURCE CODE:**

```
// Parent class
class Animal
{
    // Method to describe the sound of the animal
    public void sound()
    {
        System.out.println("Animals make different sounds.");
    }
}

// Child class
class Dog extends Animal {
    // Overriding the sound method in the parent class
    @Override
    public void sound() {
        System.out.println("Dog barks.");
    }
}

// Another child class
class Cat extends Animal {
    // Overriding the sound method in the parent class
    @Override
    public void sound() {
        System.out.println("Cat meows.");
    }
}

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

```

```
// Create an Animal object
Animal animal = new Animal();
animal.sound(); // Calls the sound method of Animal

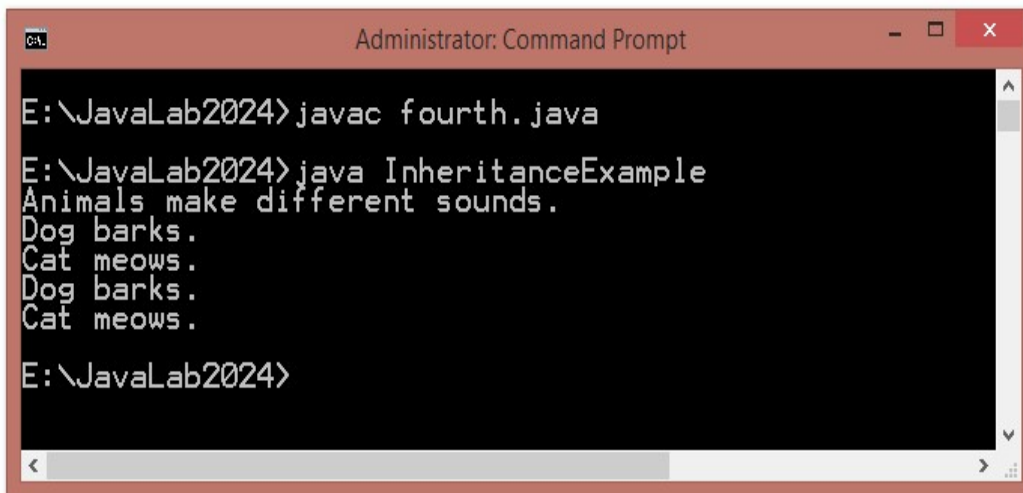
// Create a Dog object
Dog dog = new Dog();
dog.sound(); // Calls the overridden sound method of Dog

// Create a Cat object
Cat cat = new Cat();
cat.sound(); // Calls the overridden sound method of Cat

// demonstrating polymorphism
Animal myAnimal;

myAnimal = dog;
myAnimal.sound();    // Calls Dog's sound method

myAnimal = cat;
myAnimal.sound();    // Calls Cat's sound method
}
}
```

**OUTPUT:**

```
Administrator: Command Prompt

E:\JavaLab2024>javac fourth.java

E:\JavaLab2024>java InheritanceExample
Animals make different sounds.
Dog barks.
Cat meows.
Dog barks.
Cat meows.

E:\JavaLab2024>
```

**LAB EXERCISE 5:**

Write a java program to implement multilevel inheritance by applying various access controls to its data members and methods. Write a java program to implement inheritance and demonstrate use of method overriding

**PROGRAM DESCRIPTION:**

This Java program demonstrates multilevel inheritance and the use of various access control modifiers (public, protected, private, and default) in a class hierarchy. The program consists of three classes: Vehicle, Car (which inherits from Vehicle), and SportsCar (which inherits from Car). The Vehicle class contains data members and methods with different access levels, such as public for general accessibility, protected for subclass access, private for internal use only, and default for package-level access. The Car and SportsCar classes override and use these inherited methods to display details of the vehicle, car, and sports car, demonstrating how access control affects visibility and method invocation in multilevel inheritance.

**SOURCE CODE:**

```
// Base class
class Vehicle
{
    public String brand;      // Public data member: Accessible from anywhere
    protected int maxSpeed;   // Protected data member: Accessible within the same package and
                               // subclasses
    private String engineType; // Private data member: Accessible only within this class
    String fuelType;          // Default (package-private) data member: Accessible within the same
                               // package

    // Public method: Accessible from anywhere
    public void displayBrand()
    {
        System.out.println("Vehicle Brand: " + brand);
    }

    // Protected method: Accessible within the same package and subclasses
    protected void displayMaxSpeed()
    {
        System.out.println("Max Speed: " + maxSpeed + " km/h");
    }

    // Private method: Accessible only within this class
    private void displayEngineType() {
        System.out.println("Engine Type: " + engineType);
    }

    // Default (package-private) method: Accessible within the same package
    void displayFuelType() {
        System.out.println("Fuel Type: " + fuelType);
    }

    // Constructor
    public Vehicle(String brand, int maxSpeed, String engineType, String fuelType) {
        this.brand = brand;
    }
}
```

```
        this.maxSpeed = maxSpeed;
        this.engineType = engineType;
        this.fuelType = fuelType;
    }
}

// Derived class
class Car extends Vehicle {
    // Constructor
    public Car(String brand, int maxSpeed, String engineType, String fuelType) {
        super(brand, maxSpeed, engineType, fuelType); // Call the superclass constructor
    }

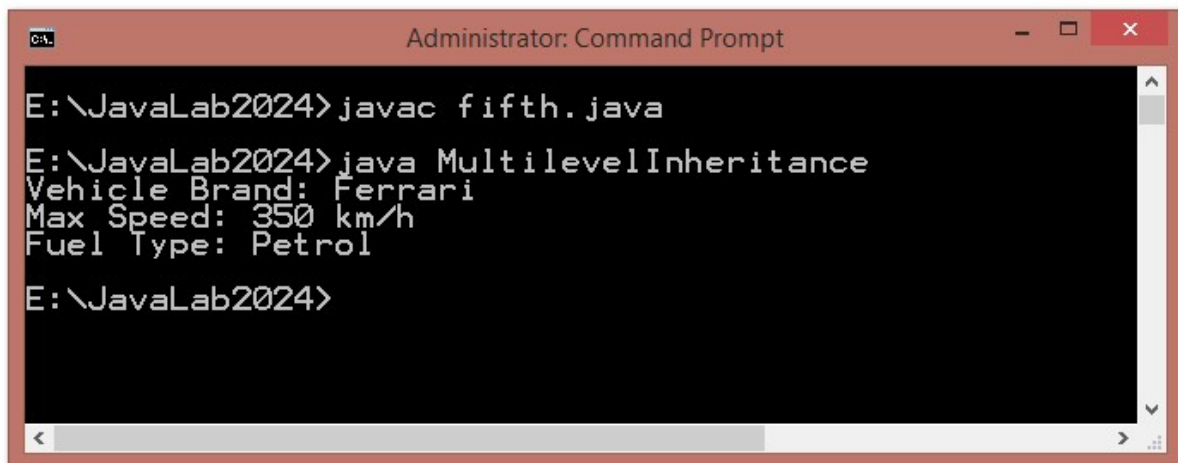
    // Overriding method to display car-specific details
    public void displayCarDetails() {
        displayBrand();          // Public method from the base class
        displayMaxSpeed();       // Protected method from the base class
        // displayEngineType(); // Error: Cannot access private method from base class
        displayFuelType();       // Default method from the base class
    }
}

// Further derived class
class SportsCar extends Car {
    // Constructor
    public SportsCar(String brand, int maxSpeed, String engineType, String fuelType) {
        super(brand, maxSpeed, engineType, fuelType); // Call the superclass constructor
    }

    // Method to display sports car-specific details
    public void displaySportsCarDetails() {
        displayBrand();          // Public method from the base class
        displayMaxSpeed();       // Protected method from the base class
        // displayEngineType(); // Error: Cannot access private method from base class
        displayFuelType();       // Default method from the base class
    }
}

// Main class
class MultilevelInheritance
{
    public static void main(String[] args)
    {
        // Create an object of SportsCar (which inherits from Car and Vehicle)
        SportsCar sportsCar = new SportsCar("Ferrari", 350, "V8", "Petrol");
    }
}
```

```
        // Call the method from SportsCar to display details
        sportsCar.displaySportsCarDetails();
    }
}
```

**OUTPUT:**

```
Administrator: Command Prompt

E:\JavaLab2024>javac fifth.java
E:\JavaLab2024>java MultilevelInheritance
Vehicle Brand: Ferrari
Max Speed: 350 km/h
Fuel Type: Petrol
E:\JavaLab2024>
```

**LAB EXERCISE 6:**

**Write a program to demonstrate use of implementing interfaces**

**PROGRAM DESCRIPTION:**

This Java program demonstrates the use of interfaces by modeling student information through an interface called Student. The interface defines methods for displaying and updating student details, along with a default method for calculating tuition fees. Two classes, Undergraduate and Postgraduate, implement this interface, providing their own specific implementations for the displayDetails(), updateDetails(), and calculateTuitionFee() methods. The program creates objects for both classes, updates their details, and calls the implemented methods to display the information and calculate the respective tuition fees. The use of interfaces in this program illustrates how common behaviors can be defined and shared across different classes while allowing for flexibility in implementation.

**SOURCE CODE:**

```
// Define the interface
interface Student
{
    // Abstract method to display student details
    void displayDetails();

    // Abstract method to update student details
    void updateDetails(String name, int age, String course);
}
```

```
// Default method to calculate tuition fee (can be overridden by implementing classes)
default void calculateTuitionFee()
{
    System.out.println("Calculating general tuition fee...");
}
}
```

```
// Undergraduate class implementing the Student interface
class Undergraduate implements Student
```

```
{
    private String name;
    private int age;
    private String course;
```

```
// Constructor
```

```
public Undergraduate(String name, int age, String course)
{
    this.name = name;
    this.age = age;
    this.course = course;
}
```

```
// Implementing displayDetails method
```

```
@Override
public void displayDetails()
{
    System.out.println("Undergraduate Student: ");
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Course: " + course);
}
```

```
// Implementing updateDetails method
```

```
@Override
public void updateDetails(String name, int age, String course)
{
    this.name = name;
    this.age = age;
    this.course = course;
}
```

```
// Overriding the calculateTuitionFee method
```

```
@Override
public void calculateTuitionFee() {
    System.out.println("Calculating undergraduate tuition fee...");
}
```

```
}  
}  
  
// Postgraduate class implementing the Student interface  
class Postgraduate implements Student {  
    private String name;  
    private int age;  
    private String course;  
  
    // Constructor  
    public Postgraduate(String name, int age, String course) {  
        this.name = name;  
        this.age = age;  
        this.course = course;  
    }  
  
    // Implementing displayDetails method  
    @Override  
    public void displayDetails() {  
        System.out.println("Postgraduate Student: ");  
        System.out.println("Name: " + name);  
        System.out.println("Age: " + age);  
        System.out.println("Course: " + course);  
    }  
  
    // Implementing updateDetails method  
    @Override  
    public void updateDetails(String name, int age, String course)  
    {  
        this.name = name;  
        this.age = age;  
        this.course = course;  
    }  
  
    // Overriding the calculateTuitionFee method  
    @Override  
    public void calculateTuitionFee()  
    {  
        System.out.println("Calculating postgraduate tuition fee...");  
    }  
}  
  
// Main class to test the implementation  
class StudentInformation  
{
```

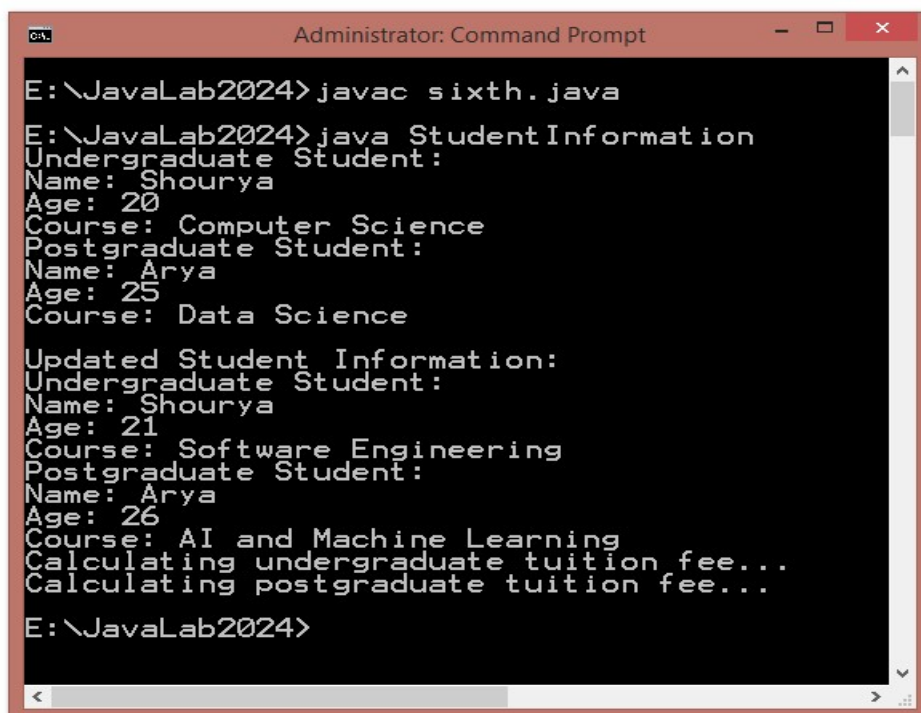
```
public static void main(String[] args)
{
    // Create Undergraduate and Postgraduate student objects
    Student undergrad = new Undergraduate("Shourya", 20, "Computer Science");
    Student postgrad = new Postgraduate("Arya", 25, "Data Science");

    // Display student details
    undergrad.displayDetails();
    postgrad.displayDetails();

    // Update student details
    undergrad.updateDetails("Shourya", 21, "Software Engineering");
    postgrad.updateDetails("Arya", 26, "AI and Machine Learning");

    // Display updated details
    System.out.println("\nUpdated Student Information:");
    undergrad.displayDetails();
    postgrad.displayDetails();

    // Calculate tuition fees
    undergrad.calculateTuitionFee();
    postgrad.calculateTuitionFee();
}
}
```

**OUTPUT:**

```
Administrator: Command Prompt
E:\JavaLab2024>javac sixth.java
E:\JavaLab2024>java StudentInformation
Undergraduate Student:
Name: Shourya
Age: 20
Course: Computer Science
Postgraduate Student:
Name: Arya
Age: 25
Course: Data Science

Updated Student Information:
Undergraduate Student:
Name: Shourya
Age: 21
Course: Software Engineering
Postgraduate Student:
Name: Arya
Age: 26
Course: AI and Machine Learning
Calculating undergraduate tuition fee...
Calculating postgraduate tuition fee...

E:\JavaLab2024>
```

**LAB EXERCISE 7:**

**Design a Java interface for ADT Stack. Develop two different classes that implement this interface, one using array and the other using linked-list. Provide necessary exception handling in both the implementations**

**PROGRAM DESCRIPTION:**

This Java program demonstrates the implementation of stack operations using both an array-based stack (Astack) and a linked list-based stack (liststack). The stackoperation interface defines two methods: push(int i) and pop(), which are implemented by both stack classes. The Astack class uses an array to store stack elements, handling overflow and underflow conditions, while the liststack class uses a linked list, with a node class to represent individual elements. The program allows the user to interactively choose between the array-based or linked list-based stack and perform operations like pushing, popping, and displaying elements. The main method provides a menu-driven interface to select the stack type and operations, demonstrating how different data structures can be used to implement the same stack functionality

**SOURCE CODE:**

```
import java.io.*;

interface stackoperation
{
    public void push(int i);
    public void pop();
}

class Astack implements stackoperation
{
    int stack[];
    int top;
    Astack()
    {
        stack=new int[10]; top=0;
    }

    public void push(int item)
    {
        if (stack[top]==10)
            System.out.println("overflow");
        else
        {
            stack[++top]=item;
            System.out.println("item pushed");
        }
    }
}
```

```
public void pop()
{
    if (stack[top]<=0)
        System.out.println("underflow");
    else
    {
        stack[top]=top--;
        System.out.println("item popped");
    }
}

public void display()
{
    for(int i=1;i<=top;i++)
        System.out.println("element:"+stack[i]);
}
}
class liststack implements stackoperation
{
    node top,q;
    int count;

    public void push(int i)
    {
        node n=new node(i);
        n.link=top;
        top=n;
        count++;
    }

    public void pop()
    {
        if(top==null)
            System.out.println("under flow");
        else
        {
            int p=top.data;
            top=top.link;
            count--;
            System.out.println("popped element:"+p);
        }
    }

    void display()
```

```
        {
            for(q=top;q!=null;q=q.link)
            {
                System.out.println("the elements are:"+q.data);
            }
        }
    }

class node
{
    int data;
    node link;

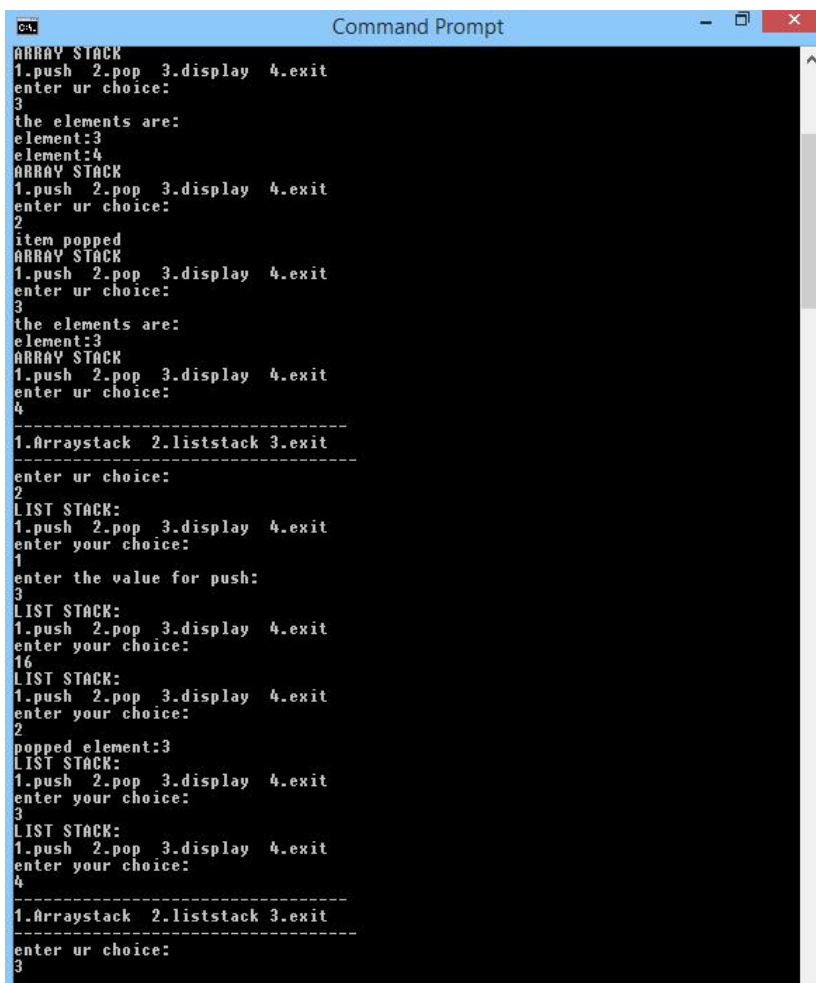
    node(int i)
    {
        data=i;
        link=null;
    }
}

class sample
{
    public static void main(String args[]) throws IOException
    {
        int ch, x=1,p=0,t=0;
        DataInputStream in=new DataInputStream(System.in); do
        {
            try
            {
                System.out.println("-----");
                System.out.println("1.Arraystack 2.liststack 3.exit");
                System.out.println("-----");
                System.out.println("enter ur choice:");
                int c=Integer.parseInt(in.readLine());
                Astack s=new Astack();
                switch(c)
                {
                    case 1:
                        do
                        {
                            if(p==1)
                                break;
                            System.out.println("ARRAY STACK");
                        }
                    }
                }
            }
            catch (Exception e)
            {
                System.out.println("Invalid choice");
            }
        } while (ch!=3);
    }
}
```

```
        System.out.println("1.push 2.pop 3.display 4.exit");
        System.out.println("enter ur choice:");
        ch=Integer.parseInt(in.readLine());

        switch(ch)
        {
        case 1:
            System.out.println("enter the value to push:");
            int i=Integer.parseInt(in.readLine());
            s.push(i);
            break;
        case 2:
            s.pop();
            break;
        case 3:
            System.out.println("the elements are:");
            s.display();
            break;
        case 4:
            p=1;
            continue;
        }
    } while(x!=0);
    break;
case 2:
    liststack l=new liststack();
    do
    {
        if(t==1)
            break;
        System.out.println("LIST STACK:");
        System.out.println("1.push 2.pop 3.display 4.exit");
        System.out.println("enter your choice:");
        ch=Integer.parseInt(in.readLine());
        switch(ch)
        {
        case 1:
            System.out.println("enter the value for push:");
            int a=Integer.parseInt(in.readLine());
            l.push(a);
            break;
        case 2:
            l.pop();
            break;
        case 3:
```

```
                l.display();
                break;
            case 4:
                t=1;
                continue;
        }
    }
    while(x!=0);
    break;
case 3:
    System.exit(0);
}
}
catch(IOException e)
{
    System.out.println("io error");
}
} while(x!=0);
}
}
```

**OUTPUT:**

```
cmd Command Prompt
ARRAY STACK
1.push 2.pop 3.display 4.exit
enter ur choice:
3
the elements are:
element:3
element:4
ARRAY STACK
1.push 2.pop 3.display 4.exit
enter ur choice:
2
item popped
ARRAY STACK
1.push 2.pop 3.display 4.exit
enter ur choice:
3
the elements are:
element:3
ARRAY STACK
1.push 2.pop 3.display 4.exit
enter ur choice:
4
-----
1.Arraystack 2.liststack 3.exit
-----
enter ur choice:
2
LIST STACK:
1.push 2.pop 3.display 4.exit
enter your choice:
1
enter the value for push:
3
LIST STACK:
1.push 2.pop 3.display 4.exit
enter your choice:
16
LIST STACK:
1.push 2.pop 3.display 4.exit
enter your choice:
2
popped element:3
LIST STACK:
1.push 2.pop 3.display 4.exit
enter your choice:
3
LIST STACK:
1.push 2.pop 3.display 4.exit
enter your choice:
4
-----
1.Arraystack 2.liststack 3.exit
-----
enter ur choice:
3
```

**LAB EXERCISE 8:**

**Write a Java program to implement the concept of importing classes from user defined package and creating packages**

**PROGRAM DESCRIPTION:**

The provided Java code defines a simple banking system with two classes: Bank and Account. The Bank class contains two attributes: name (a String) and balance (a double), which are initialized via its constructor. The show() method in the Bank class prints the account holder's name and balance to the console. The Account class contains the main method, where an instance of the Bank class is created with the name "ram" and a balance of 5678.23. The show() method is then called to display this information.

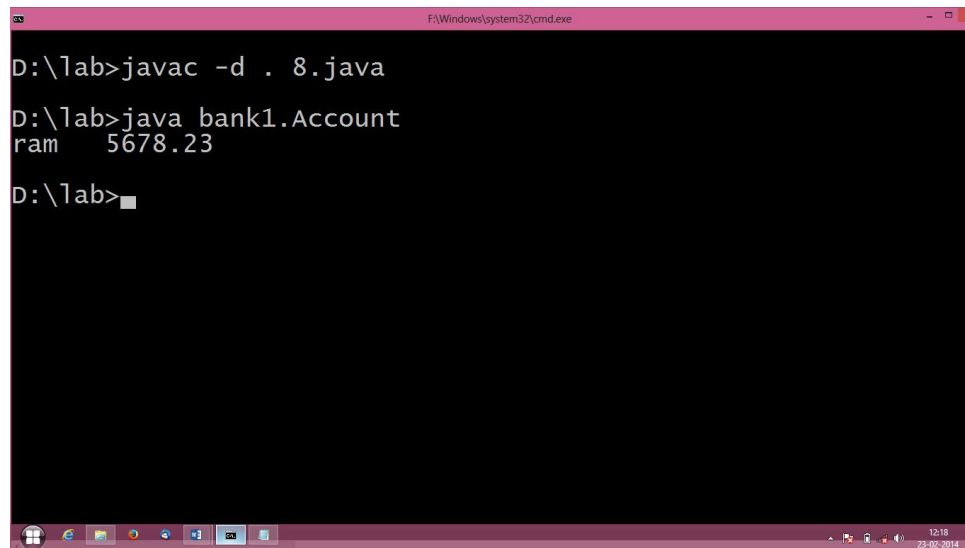
**SOURCE CODE:**

```
package bank1;

class Bank
{
    String name; double balance;
    Bank(String n,double bal)
    {
        name=n;
        balance=bal;
    }

    void show()
    {
        System.out.println(name+" "+balance);
    }
}

class Account
{
    public static void main(String args[ ])
    {
        Bank b=new Bank("ram",5678.23);
        b.show();
    }
}
```

**OUTUT:**A screenshot of a Windows command prompt window. The title bar reads 'F:\Windows\system32\cmd.exe'. The command prompt shows the following sequence of commands and output:  
D:\lab>javac -d . 8.java  
D:\lab>java bank1.Account  
ram 5678.23  
D:\lab>  
The window has a standard Windows taskbar at the bottom with various icons and a system clock showing 12:18 on 23-02-2014.**LAB EXERCISE 9:**

**Write a program to implement the concept of threading by implementing Runnable Interface**

**PROGRAM DESCRIPTION:**

This Java program demonstrates the use of multithreading by creating a child thread alongside the main thread. The NewThread class implements the Runnable interface and defines a run() method, where it counts down from 5 to 1, pausing for 500 milliseconds between each iteration. The main() method in the ThreadDemo class creates an instance of NewThread, starting the child thread, and then the main thread independently counts down from 5 to 1, pausing for 1 second between each iteration. Both threads print messages to the console to indicate their progress. The program also handles interruptions, and at the end, both threads print a message before exiting. The Thread.sleep() method is used to simulate delays in the execution of both threads.

**SOURCE CODE:**

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

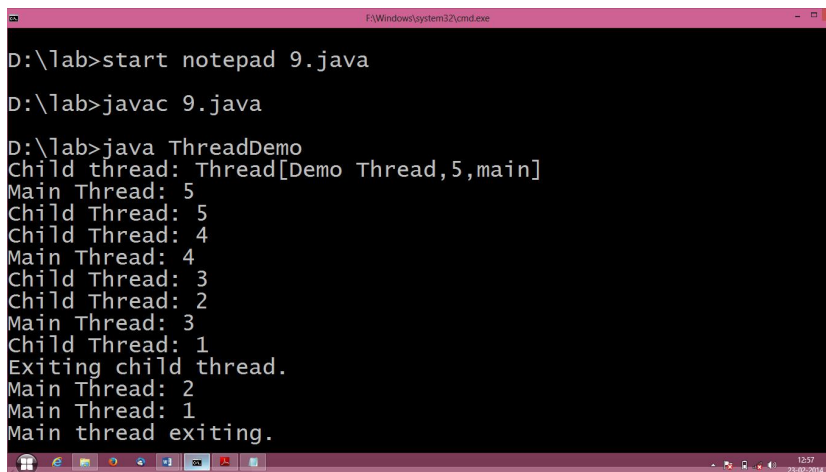
class NewThread implements Runnable
{
    Thread t; NewThread()
    {
        t = new Thread(this, "Demo Thread");
        System.out.println("Child thread: " + t);
```

```
        t.start();
    }

    public void run()
    {
        try
        {
            for(int i = 5; i > 0; i--)
            {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Child interrupted.");
        }

        System.out.println("Exiting child thread.");
    }
}

class ThreadDemo
{
    public static void main(String args[])
    {
        new NewThread();
        try
        {
            for (int i = 5; i > 0; i--)
            {
                System.out.println("Main Thread: " + i);
                Thread.sleep(1000);
            }
        }
        catch(InterruptedException e)
        {
            System.out.println("Main thread interrupted.");
        }
        System.out.println("Main thread exiting.");
    }
}
```

**OUTPUT:**

```
D:\lab>start notepad 9.java
D:\lab>javac 9.java
D:\lab>java ThreadDemo
Child thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Main Thread: 4
Child Thread: 3
Child Thread: 2
Main Thread: 3
Child Thread: 1
Exiting child thread.
Main Thread: 2
Main Thread: 1
Main thread exiting.
```

**LAB EXERCISE 10:**

**Write a java program to store and read objects from a file**

**PROGRAM DESCRIPTION:**

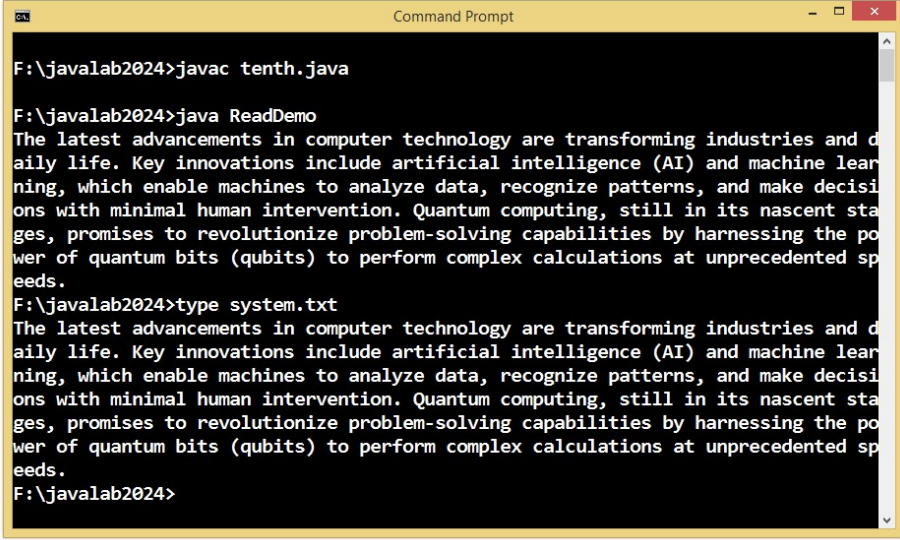
The provided Java program demonstrates file handling by reading the contents of one file and writing it to another. It uses `FileInputStream` to open and read data from a file named "technology.txt" and `FileOutputStream` to write the read data into a new file called "system.txt." Inside the while loop, the program reads each byte from the input file, prints the corresponding character to the console, and writes the byte to the output file. The program continues this process until all bytes have been read (when `fis.read()` returns -1). Finally, the program closes both the input and output file streams to release system resources. This program handles potential file input/output exceptions using the throws `Exception` declaration.

**SOURCE CODE:**

```
import static java.lang.System.*;
import java.io.*;

class ReadDemo
{
    public static void main(String args[])throws Exception
    {
        FileInputStream fis=new FileInputStream("technology.txt");
        FileOutputStream fos=new FileOutputStream("system.txt");
        int r=0;
        while((r=fis.read())!=-1)
        {
            out.print((char)r);
            fos.write(r);
        }
    }
}
```

```
        fis.close();
        fos.close();
    }
}
```

**OUTPUT:**

```
Command Prompt

F:\javab2024>javac tenth.java

F:\javab2024>java ReadDemo
The latest advancements in computer technology are transforming industries and d
aily life. Key innovations include artificial intelligence (AI) and machine lear
ning, which enable machines to analyze data, recognize patterns, and make decisi
ons with minimal human intervention. Quantum computing, still in its nascent sta
ges, promises to revolutionize problem-solving capabilities by harnessing the po
wer of quantum bits (qubits) to perform complex calculations at unprecedented sp
eeds.

F:\javab2024>type system.txt
The latest advancements in computer technology are transforming industries and d
aily life. Key innovations include artificial intelligence (AI) and machine lear
ning, which enable machines to analyze data, recognize patterns, and make decisi
ons with minimal human intervention. Quantum computing, still in its nascent sta
ges, promises to revolutionize problem-solving capabilities by harnessing the po
wer of quantum bits (qubits) to perform complex calculations at unprecedented sp
eeds.

F:\javab2024>
```

**LAB EXERCISE 11:**

**Write a Java program that displays the number of characters, lines and words in a text file.**

**PROGRAM DESCRIPTION:**

This Java program calculates and displays the number of characters, lines, and words in a specified text file. It uses `BufferedReader` and `FileReader` to read the file line by line. For each line, the program increments the line count, splits the line into words (using regular expression `\\s+` to handle spaces), and counts the words by measuring the length of the resulting array. It also counts the characters by determining the length of each line. After processing all lines, the program prints the total number of characters, lines, and words. If there is an error while reading the file, the program catches the `IOException` and displays an error message. The file path can be changed by modifying the `filePath` variable.

**SOURCE CODE:**

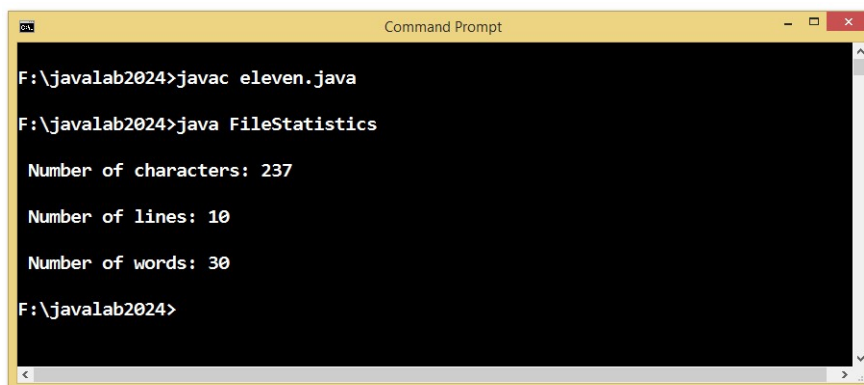
```
import java.io.*;

class FileStatistics
{
    public static void main(String[] args)
    {
        String filePath = "sample.txt";
        // Change to the path of your text file. Initialize counters for characters, lines, and words
```

```
int characterCount = 0;
int lineCount = 0;
int wordCount = 0;

try (BufferedReader br = new BufferedReader(new FileReader(filePath)))
{
    String line;          //Read each line from the file
    while ((line = br.readLine()) != null)
    {
        lineCount++; // Count lines
        String[] words = line.split("\\s+"); // Count words in the current line
        wordCount += words.length;
        characterCount += line.length(); // Count characters in the current line
    }
    // Display the results
    System.out.println("\n Number of characters: " + characterCount);
    System.out.println("\n Number of lines: " + lineCount);
    System.out.println("\n Number of words: " + wordCount);
}
catch (IOException e)
{
    System.out.println("An error occurred while reading the file.");
    e.printStackTrace();
}
}
```

## OUTPUT



```
Command Prompt
F:\javalab2024>javac eleven.java
F:\javalab2024>java FileStatistics

Number of characters: 237

Number of lines: 10

Number of words: 30
F:\javalab2024>
```

**LAB EXERCISE 12:****Write a java program to illustrate object serialization****PROGRAM DESCRIPTION:**

This Java program demonstrates object serialization and deserialization using the Voter class. The Voter class, which implements the Serializable interface, contains two fields: name and age, along with a constructor to initialize these fields and a display() method to print the object's details. The program creates an instance of the Voter class, serializes it into a file named Voter.ser using ObjectOutputStream, and prints a confirmation message. Then, it deserializes the object from the file using ObjectInputStream, and the deserialized object's details are displayed. The program also includes exception handling to catch errors during both serialization and deserialization. This example highlights the process of converting an object into a byte stream and restoring it back to its original form in Java.

**SOURCE CODE:**

```
import java.io.*;

// Class to represent a simple object that can be serialized
class Voter implements Serializable
{
    private String name;
    private int age;

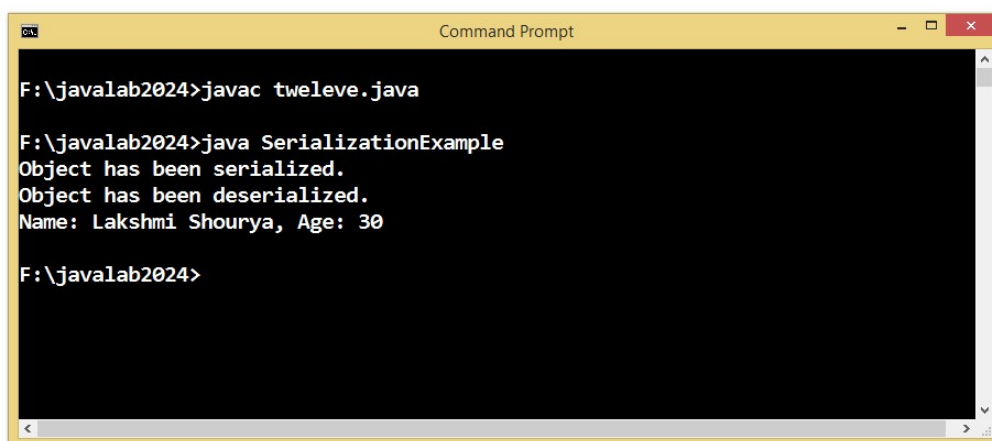
    // Constructor to initialize the object
    public Voter(String name, int age)
    {
        this.name = name;
        this.age = age;
    }

    // Method to display the object's information
    public void display()
    {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}

class SerializationExample
{
    public static void main(String[] args)
    {
        // Create an object of Voter class
        Voter Voter1 = new Voter("Lakshmi Shourya", 30);
        // Serialize the object to a file
```

```
try (ObjectOutputStream out = new ObjectOutputStream(new
    FileOutputStream("Voter.ser")))
{
    out.writeObject(Voter1); // Serialize the object
    System.out.println("Object has been serialized.");
}
catch (IOException e)
{
    System.out.println("Error during serialization: " + e.getMessage());
}

// Deserialize the object from the file
try (ObjectInputStream in = new ObjectInputStream(new
    FileInputStream("Voter.ser")))
{
    Voter deserializedVoter = (Voter) in.readObject(); // Deserialize the
object
    System.out.println("Object has been deserialized.");
    deserializedVoter.display(); // Display the deserialized object's data
}
catch (IOException | ClassNotFoundException e)
{
    System.out.println("Error during deserialization: " + e.getMessage());
}
}
```

**OUTPUT:**

```
Command Prompt

F:\javab2024>javac tweleve.java

F:\javab2024>java SerializationExample
Object has been serialized.
Object has been deserialized.
Name: Lakshmi Shourya, Age: 30

F:\javab2024>
```

**LAB EXERCISE 13:****Create a java program to illustrate user defined exception****PROGRAM DESCRIPTION:**

This Java program demonstrates the use of a user-defined exception by simulating a bank account system. It defines a custom exception, `InsufficientFundsException`, which is thrown when a user attempts to withdraw more money than their account balance. The `BankAccount` class includes a `withdraw` method that checks if the withdrawal amount exceeds the available balance, throwing the custom exception if insufficient funds are detected. In the `main` method, the program first tries to withdraw an amount larger than the balance, catches the exception, and then successfully processes a valid withdrawal. The program helps illustrate how custom exceptions can enhance error handling in real-world scenarios, like financial transactions.

**SOURCE CODE:**

```
import java.io.*;
// Define the user-defined exception
class InsufficientFundsException extends Exception
{
    // Constructor to accept a custom error message
    public InsufficientFundsException(String message)
    {
        super(message);
    }
}

// BankAccount class that performs withdrawal operations
class BankAccount
{
    private double balance; // Constructor to initialize balance
    public BankAccount(double initialBalance)
    {
        this.balance = initialBalance;
    }

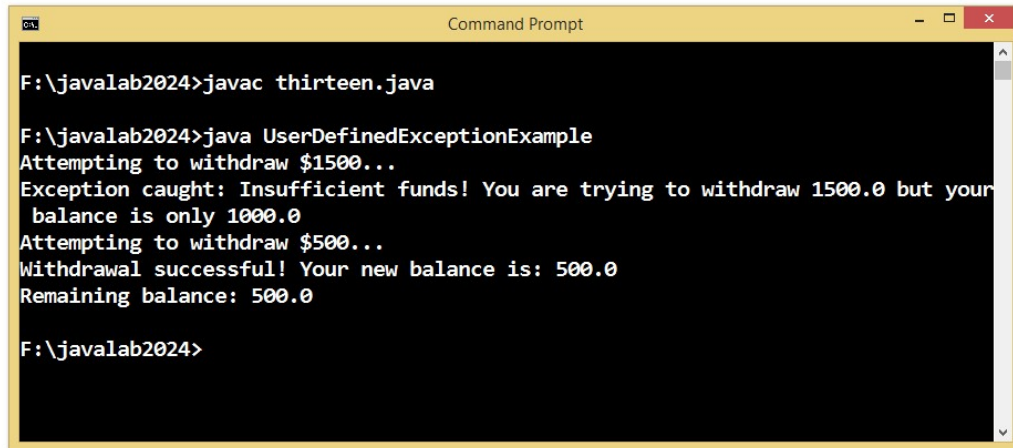
    // Method to withdraw money
    public void withdraw(double amount) throws InsufficientFundsException
    {
        if (amount > balance)
        {
            // Throwing user-defined exception if withdrawal amount is greater than the balance
            throw new InsufficientFundsException("Insufficient funds! You are trying to withdraw " + amount + " but your balance is only " + balance);
        }
    }
}
```

```
        balance -= amount; // Deduct the amount from the balance
        System.out.println("Withdrawal successful! Your new balance is: " +
                                balance);
    }

    // Method to check balance
    public double getBalance()
    {
        return balance;
    }
}

class UserDefinedExceptionExample
{
    public static void main(String[] args)
    {
        // Create a BankAccount with an initial balance of 1000
        BankAccount account = new BankAccount(1000.00);
        // Try to withdraw an amount greater than the balance
        try
        {
            System.out.println("Attempting to withdraw $1500...");
            account.withdraw(1500.00); // This should throw the exception
        }
        catch (InsufficientFundsException e)
        {
            // Handle the exception
            System.out.println("Exception caught: " + e.getMessage());
        }
        // Try a successful withdrawal
        try
        {
            System.out.println("Attempting to withdraw $500...");
            account.withdraw(500.00); // This should succeed
        }
        catch (InsufficientFundsException e)
        {
            // Handle the exception
            System.out.println("Exception caught: " + e.getMessage());
        }

        // Display remaining balance
        System.out.println("Remaining balance: " + account.getBalance());
    }
}
```

**OUTPUT:**

```
F:\javab2024>javac thirteen.java

F:\javab2024>java UserDefinedExceptionExample
Attempting to withdraw $1500...
Exception caught: Insufficient funds! You are trying to withdraw 1500.0 but your
balance is only 1000.0
Attempting to withdraw $500...
Withdrawal successful! Your new balance is: 500.0
Remaining balance: 500.0

F:\javab2024>
```

**LAB EXERCISE 14:**

**Write a Java Program to create a thread using runnable interface**

---

**PROGRAM DESCRIPTION**

The above Java program demonstrates the use of a user-defined exception by simulating a bank account system. It defines a custom exception, 'InsufficientFundsException', which is thrown when a user attempts to withdraw more money than their account balance. The 'BankAccount' class includes a 'withdraw' method that checks if the withdrawal amount exceeds the available balance, throwing the custom exception if insufficient funds are detected. In the 'main' method, the program first tries to withdraw an amount larger than the balance, catches the exception, and then successfully processes a valid withdrawal. The program helps illustrate how custom exceptions can enhance error handling in real-world scenarios, like financial transactions.

**SOURCE CODE:**

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

class Tdemo implements Runnable
{
    boolean stop=false; Tdemo()
    {
        Thread t=new Thread(this); t.start();
    }

    public void run()
    {
        for(;;)
```



starts three threads ('t1', 't2', and 't3'), each corresponding to one of the classes. The use of the 'synchronized' keyword ensures that the messages are printed in a thread-safe manner, although there are no specific shared resources being accessed. This program illustrates basic multithreading concepts in Java and how threads can work concurrently with different sleep durations.

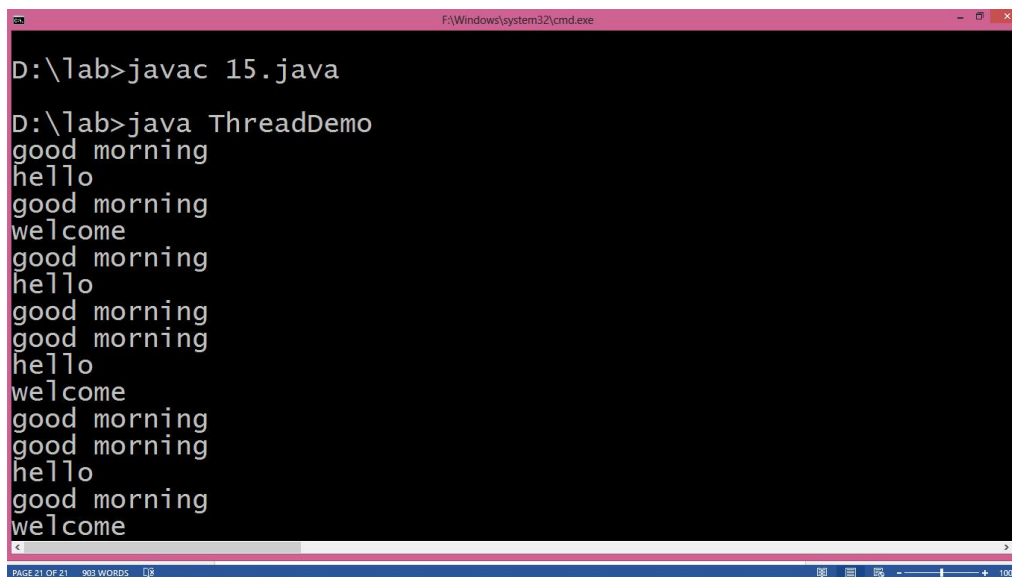
**SOURCE CODE:**

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

class A extends Thread
{
    synchronized public void run()
    {
        try
        {
            while(true)
            {
                sleep(10);
                System.out.println("good morning");
            }
        }
        catch(Exception e)
        {
        }
    }
}

class B extends Thread
{
    synchronized public void run()
    {
        try
        {
            while(true)
            {
                sleep(20);
                System.out.println("hello");
            }
        }
        catch(Exception e)
        {
        }
    }
}
```

```
}  
class C extends Thread  
{  
    synchronized public void run()  
    {  
        try  
        {  
            while(true)  
            {  
                sleep(30);  
                System.out.println("welcome");  
            }  
        }  
        catch(Exception e)  
        {  
        }  
    }  
}  
  
}  
  
class ThreadDemo  
{  
    public static void main(String args[])  
    {  
        A t1=new A();  
        B t2=new B();  
        C t3=new C();  
        t1.start();  
        t2.start();  
        t3.start();  
    }  
}
```

**OUTPUT:**

```
D:\lab>javac 15.java  
D:\lab>java ThreadDemo  
good morning  
hello  
good morning  
welcome  
good morning  
hello  
good morning  
good morning  
hello  
welcome  
good morning  
good morning  
hello  
good morning  
welcome
```

**LAB EXERCISE 16:**

**Write a java program to create multiple threads that correctly implements producer consumer problem using the concept of Inter thread communication**

---

**PROGRAM DESCRIPTION**

This Java program demonstrates a basic producer-consumer problem using multithreading and synchronization. It involves two threads, 'Producer' and 'Consumer', which communicate with each other by sharing a resource. The 'Producer' thread generates values (from 1 to 10) and assigns them to a variable 'j' in the 'Consumer' class. The producer uses 'synchronized' blocks to ensure that the 'Consumer' thread consumes the produced value in a controlled manner. After producing a value, the producer calls 'wait()' to release the lock and allow the consumer to consume the value. The 'Consumer' thread, in turn, consumes the value, prints it, and notifies the producer using 'notify()' to signal that the producer can continue. This cycle repeats until both threads have processed 10 values. The program uses 'Thread.sleep()' to simulate delays and control the timing of production and consumption. The synchronization ensures that the producer and consumer work in a coordinated manner without conflicts.

**SOURCE CODE:**

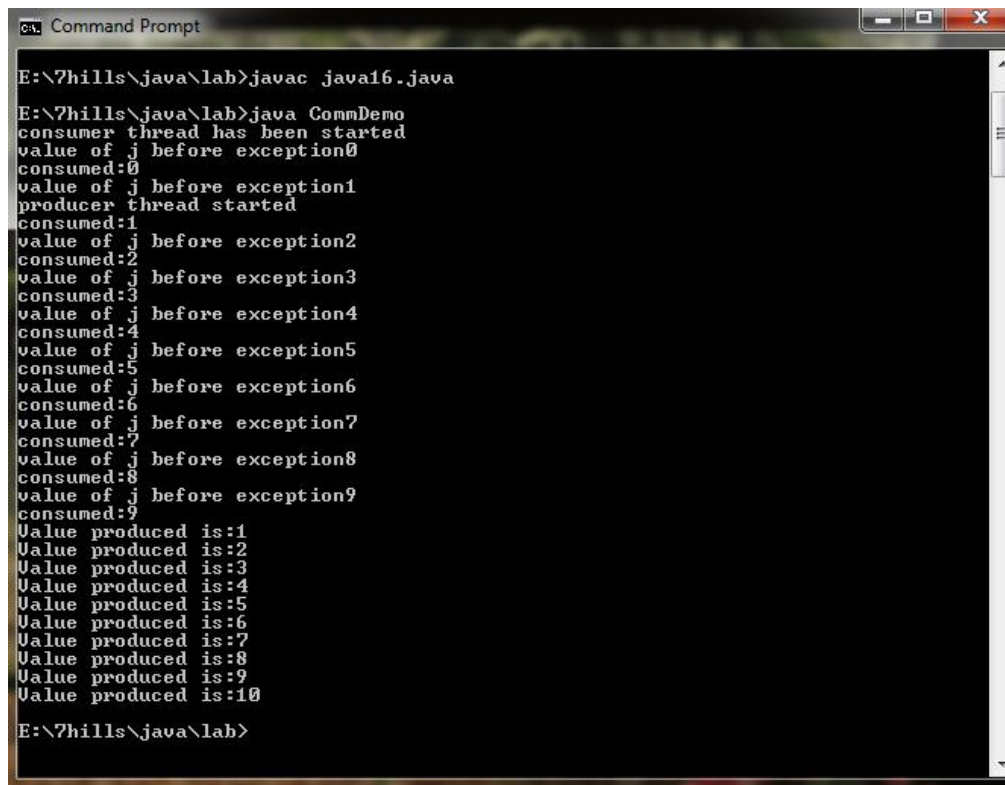
```
import java.lang.*;

class Producer extends Thread
{
    static int i=0;
    Consumer obj;
    Producer(Consumer obj1)
    {
        obj=obj1;
    }

    public void run()
    {
        System.out.println("producer thread started");
        synchronized(obj)
        {
            while(i<10)
            {
                try
                {
                    System.out.println("Value produced is:"+i);
                    obj.j=++i;
                    Thread.sleep(10);
                    wait();
                    System.out.println("lock is released by producer"); i++;
                }
            }
        }
    }
}
```

```
        }
        catch(Exception e)
        {
        }
    }
class Consumer extends Thread
{
    int j;
    public void run()
    {
        System.out.println("consumer thread has been started");
        synchronized(this)
        {
            while(j<10)
            {
                try
                {
                    System.out.println("value of j before exception"+j);
                    Thread.sleep(10);
                    System.out.println("consumed:"+j);
                    j++;
                }
                notify();
            }
            catch(Exception e)
            {
            }
        }
    }
}

class CommDemo
{
    public static void main(String[] args)throws Exception
    {
        Consumer con=new Consumer();
        Producer pod=new Producer(con);
        con.start();
        Thread.sleep(5);
        pod.start();
    }
}
```

**OUTPUT:**

```
Command Prompt
E:\7hills\java\lab>javac java16.java
E:\7hills\java\lab>java CommDemo
consumer thread has been started
value of j before exception0
consumed:0
value of j before exception1
producer thread started
consumed:1
value of j before exception2
consumed:2
value of j before exception3
consumed:3
value of j before exception4
consumed:4
value of j before exception5
consumed:5
value of j before exception6
consumed:6
value of j before exception7
consumed:7
value of j before exception8
consumed:8
value of j before exception9
consumed:9
Value produced is:1
Value produced is:2
Value produced is:3
Value produced is:4
Value produced is:5
Value produced is:6
Value produced is:7
Value produced is:8
Value produced is:9
Value produced is:10
E:\7hills\java\lab>
```

**LAB EXERCISE 17:**

**Write an applet to handling the mouse events**

---

**PROGRAM DESCRIPTION**

The Java program demonstrates the use of mouse event handling in an applet using `MouseListener` and `MouseMotionListener` interfaces. It creates a graphical interface where various mouse actions trigger different events. The `MouseDemo` applet reacts to mouse events like entering, exiting, clicking, pressing, and releasing. Depending on the action, it changes the background color of the applet and updates a message (msg). For example, when the mouse is pressed, the background changes to green, and the message is updated to "Java." The program also captures mouse movement and drag events, updating the message and background color accordingly. The `paint()` method is used to display the updated message at specific coordinates on the applet's canvas. This interactive applet showcases how Java handles mouse inputs and provides feedback through visual and textual updates.

**SOURCE CODE:**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseDemo" width=500 height=500> </applet>
```

\*/

```
class MouseDemo extends Applet implements MouseListener, MouseMotionListener
{
    int X=0,Y=20;

    String msg="MouseEvents"; public void init()
    {
        addMouseListener(this);
        addMouseMotionListener(this);
        setBackground(Color.black);
        setForeground(Color.red);
    }

    public void mouseEntered(MouseEvent m)
    {
        setBackground(Color.magenta); showStatus("Mouse Entered"); repaint();
    }

    public void mouseExited(MouseEvent m)
    {
        setBackground(Color.black); showStatus("Mouse Exited"); repaint();
    }

    public void mousePressed(MouseEvent m)
    {
        X=10;
        Y=20;
        msg="Java";
        setBackground(Color.green);
        repaint();
    }

    public void mouseReleased(MouseEvent m)
    {
        X=10;
        Y=20;
        msg="Programming";
        setBackground(Color.blue);
        repaint();
    }

    public void mouseMoved(MouseEvent m)
    {
        X=m.getX();
```

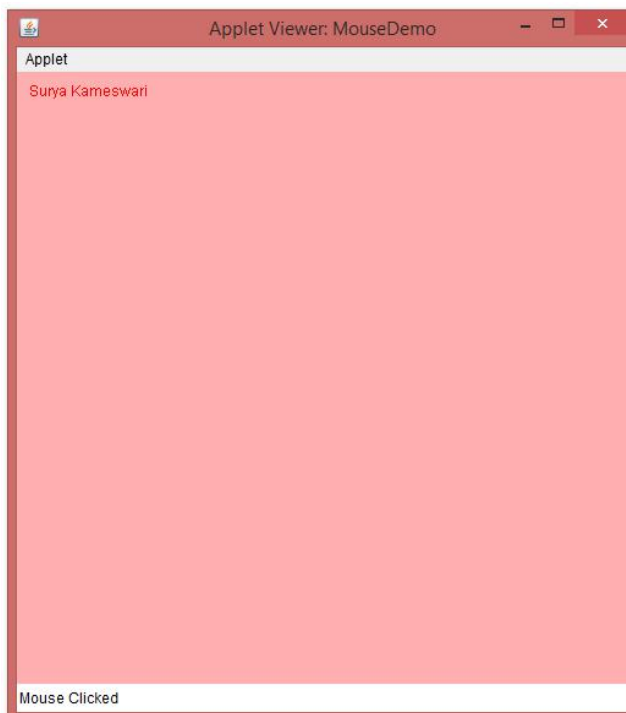
```
        Y=m.getY();
        msg="MCA";
        setBackground(Color.white);
        showStatus("Mouse Moved");
        repaint();
    }

    public void mouseDragged(MouseEvent m)
    {
        msg="ANU";
        setBackground(Color.yellow);
        showStatus("Mouse Moved"+m.getX()+" "+m.getY()); repaint();
    }

    public void mouseClicked(MouseEvent m)
    {
        msg="Surya Kameswari";
        setBackground(Color.pink);
        showStatus("Mouse Clicked");
        repaint();
    }

    public void paint(Graphics g)
    {
        g.drawString(msg,X,Y);
    }

}
```

**OUTPUT:**

**LAB EXERCISE 18:**

**Write a Program That works as a simple calculator using Grid layout to arrange buttons for the digits and +,-,\* % operations. Add a text field to print the result.**

---

**PROGRAM DESCRIPTION**

The given Java program implements a simple calculator applet using the AWT (Abstract Window Toolkit) library to handle basic arithmetic operations like addition, subtraction, multiplication, division, and modulus. The applet provides a user interface with buttons for digits (0-9), operations (+, -, \*, /, %, =), and a clear button ("C"). It uses a TextField for displaying input and results. The actionPerformed() method processes the actions triggered by the user, such as digit entries and operations, and calculates the result based on the operator selected. The result is displayed after pressing the "=" button. The calculator uses a GridLayout to organize the buttons in a 4x5 grid and handles user inputs with event listeners that perform the appropriate calculations based on the operator selected. The program also provides functionality for clearing the input field with the "C" button.

**SOURCE CODE:**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

/*    <applet code="Calculator" width=300 height=300>    </applet>    */

public class Calculator extends Applet implements ActionListener
{
    String msg=" "; int v1,v2,result; TextField t1;
    Button b[]=new Button[10];
    Button add,sub,mul,div,clear,mod,EQ; char OP;

    public void init()
    {
        Color k=new Color(120,89,90); setBackground(k);
        t1=new TextField(10);
        GridLayout gl=new GridLayout(4,5); setLayout(gl);
        for(int i=0;i<10;i++)
        {
            b[i]=new Button(""+i);
        }

        add=new Button("+");
        sub=new Button("-");
```

```
        mul=new Button("*");
        div=new Button("/");
        mod=new Button("%");
        clear=new Button("C");
        EQ=new Button("=");
        t1.addActionListener(this);
        add(t1);

        for(int i=0;i<10;i++)
        {
            add(b[i]);
        }

        add(add);
        add(sub);
        add(mul);
        add(div);
        add(mod);
        add(clear);
        add(EQ);

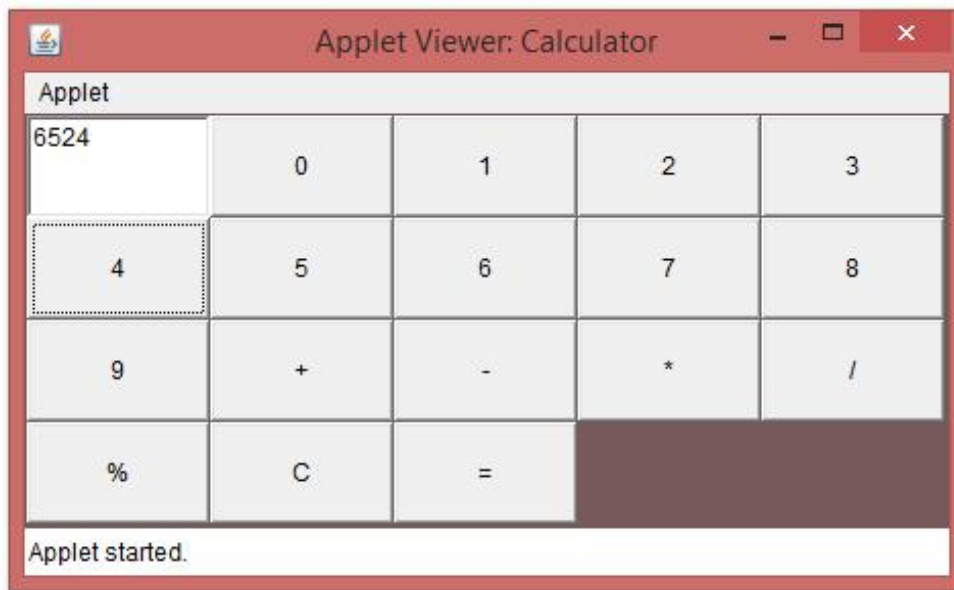
        for(int i=0;i<10;i++)
        {
            b[i].addActionListener(this);
        }
        add.addActionListener(this);
        sub.addActionListener(this);
        mul.addActionListener(this);
        div.addActionListener(this);
        mod.addActionListener(this);
        clear.addActionListener(this);
        EQ.addActionListener(this);
    }

    public void actionPerformed(ActionEvent ae)
    {
        String str=ae.getActionCommand();
        char ch=str.charAt(0);
        if ( Character.isDigit(ch))
            t1.setText(t1.getText()+str);
        else
            if(str.equals("+"))
            {
                v1=Integer.parseInt(t1.getText());
                OP='+';
```

```
        t1.setText("");
    }
    else if(str.equals("-"))
    {
        v1=Integer.parseInt(t1.getText()); OP='-';
        t1.setText("");
    }
    else if(str.equals("*"))
    {
        v1=Integer.parseInt(t1.getText());
        OP='*';
        t1.setText("");
    }
    else if(str.equals("/"))
    {
        v1=Integer.parseInt(t1.getText());
        OP='/';
        t1.setText("");
    }
    else if(str.equals("%"))
    {
        v1=Integer.parseInt(t1.getText());
        OP='%';
        t1.setText("");
    }
    if(str.equals("="))
    {
        v2=Integer.parseInt(t1.getText());
        if(OP=='+')
            result=v1+v2; else if(OP=='-')
            result=v1-v2; else if(OP=='*')
            result=v1*v2; else if(OP=='/')
            result=v1/v2; else if(OP=='%')
            result=v1%v2;
        t1.setText(""+result);
    }

    if(str.equals("C"))
    {
        t1.setText("");
    }
}
}
```

**OUTPUT:**

**LAB EXERCISE 19:****Build and run Celsius converter sample application using swings**

---

**PROGRAM DESCRIPTION**

The Java program implements a simple Celsius to Fahrenheit temperature converter using the Swing GUI framework. It creates a graphical user interface (GUI) with a 'JFrame' containing a 'JPanel' and components arranged in a 'GridLayout'. The user inputs a temperature in Celsius into a 'JTextField', and upon pressing the "Convert..." button, the program converts the Celsius value to Fahrenheit and displays the result in a 'JLabel'. The conversion formula used is 'Fahrenheit = (Celsius \* 1.8) + 32'. The program implements the 'ActionListener' interface to handle the button click event, triggering the conversion process. The look and feel of the application is set to the cross-platform default using 'UIManager'. This program demonstrates basic event handling and GUI design in Java using Swing components.

**SOURCE CODE:**

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.UIManager;
```

```
public class CelsiusConverter implements ActionListener
{
    JFrame converterFrame;
    JPanel converterPanel;
    JTextField tempCelsius;
    JLabel celsiusLabel, fahrenheitLabel;
    JButton convertTemp;

    public CelsiusConverter()
    {
        converterFrame = new JFrame("Convert Celsius to Fahrenheit");
        converterPanel = new JPanel();
        converterPanel.setLayout(new GridLayout(2, 2));
        addWidgets();
        converterFrame.getContentPane().add(converterPanel,
        BorderLayout.CENTER);
        converterFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        converterFrame.pack();
        converterFrame.setVisible(true);
    }

    // Create and add the widgets for converter.
    private void addWidgets()
    {
        tempCelsius = new JTextField(2);
        celsiusLabel = new JLabel("Celsius", SwingConstants.LEFT); convertTemp =
        new JButton("Convert...");
        fahrenheitLabel = new JLabel("Fahrenheit", SwingConstants.LEFT);
        convertTemp.addActionListener(this);
        converterPanel.add(tempCelsius);
        converterPanel.add(celsiusLabel);
        converterPanel.add(convertTemp);
        converterPanel.add(fahrenheitLabel);
        celsiusLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
        fahrenheitLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
    }

    // Implementation of ActionListener interface.
    public void actionPerformed(ActionEvent event)
    {
        int tempFahr = (int) ((Double.parseDouble(tempCelsius.getText())) * 1.8 +
        32); fahrenheitLabel.setText(tempFahr + " Fahrenheit");
    }

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

```
{
    // Set the look and feel.
    try
    {
        UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    }
    catch (Exception e)
    {
    }
    CelsiusConverter converter = new CelsiusConverter();
}
}
```

**OUTPUT:****LAB EXERCISE 20:**

**Develop an applet that receives an integer in one text field, and computes its factorial Value and returns it in another text field, when the button named “Compute” is clicked**

---

**PROGRAM DESCRIPTION**

The given Java program implements a simple applet to compute the factorial of a number entered by the user. It uses the AWT (Abstract Window Toolkit) for creating the graphical user interface (GUI). The applet contains a ‘TextField’ for the user to input a number (‘n’), a ‘Button’ labeled "compute" to trigger the calculation, and another ‘TextField’ to display the calculated factorial. When the user enters a number and clicks the "compute" button, the ‘actionPerformed()’ method is invoked, which calculates the factorial by multiplying all

integers from 1 to 'n' and displays the result in the second 'TextField'. The program demonstrates event handling in Java applets, allowing interaction with GUI elements to perform a mathematical calculation.

**SOURCE CODE:**

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*<applet code="fact.class" width=300 height=300></applet>*/
public class fact extends Applet implements ActionListener
{
    int n;
    TextField t1, t2;
    Label l1;
    Button b;

    public void init()
    {
        l1=new Label("enter n value",Label.LEFT); t1=new TextField(20);
        b=new Button("compute"); t2=new TextField(20); add(l1);
        add(t1);
        add(b);
        add(t2);
        b.addActionListener(this);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String s1=t1.getText(); int f=1;
        int n=Integer.parseInt(s1); for(int i=1;i<=n;i++)
        f=f*i;
        String s="fact="+f; t2.setText(s);
    }
}
```

**OUTPUT:**

