Department of Information Technology
BABU BANARASI DAS
Institute of Technology & Management, Lucknow
Recognized by AICTE, Govt.of India & Affiliated to Dr. A.P.J Abdul Kalam Technical University,Lucknow,AKTU College Code=054

LAB MANUAL

OF

OBJECT ORIENTED PROGRAMMING WITH JAVA

BCS452

B.TECH, 2nd Year, Semester -IV

Dr. A.P.J. Abdul Kalam Tech. University
Uttar Pradesh

2025-26

INFORMATION TECHNOLOGY

Faculty Name:

Mr.Ankit Singh (Assistant Professor)

Approved by:

(Department Head/Coordinator)

MANUAL CONTENTS

This manual is intended for the **2nd year students of Information Technology,** in the subject of **Object Oriented Programming with Java BCS452**. This manual typically contains practical sessions related to OOPs with Java covering various aspects related to the subject of enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions.

Vision of the Institute

To become a leading institute of providing professionally competent and socially responsive technocrats with high moral values.

Mission of the Institute

M1: To create an eco-system for the dissemination of technical knowledge, to achieve academic excellence.

M2: To develop technocrats with creative skills and leadership qualities, to solve local and global challenges.

M3: To impart human values and ethics in students, to make them socially and Eco-friendly responsible.

Vision of the Department

To produce globally competent professionals having social values and committed to serve the global needs with ability to work in an interdisciplinary environment

Mission of the Department

M1: To impart quality education to enhance ethical, professional and leadership qualities to make them globally competitive.

M2: To create a conducive environment in which students can explore computational problems, analyze them and identify their optimal solutions.

M3: To strive for continual enhancement of technical knowledge & innovation through industry interface to accomplish global needs.

Program Educational Objectives (PEOs)

PE01: Students must be able to apply software engineering principles to analyze complex computing problems and identify their solutions.

PE02: Students must be able to analyze, design, and implement the latest technology-driven projects.

PE03: Students must be able to work in a collaborative environment and understand the ethical, social, and economic impact of their work.

Program Specific Outcomes (PSOs)

PSO 1: Able to design and implement the data structures and algorithms to deliver quality software products.

PSO 2: Able to apply Artificial Intelligence and Machine Learning concepts to solve society related needs.

Program Outcome (POs)

Engineering Graduates will be able to:

PO: 1 Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO: 2 Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO: 3 Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO: 4 Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO: 5 Engineering tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO: 6 The engineer and The World: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO: 7 Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO: 8 Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO: 9 Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO: 10 Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO: 11 Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Evaluation Scheme

S. No	Subject Code	Subject	Periods			Evaluation Scheme		End Semester	Total	Credit
			L	T	P	PS	Total	PE		
1.	BCS452	Object Oriented Programming with Java Lab	0	0	2	50	50	50	100	1

Course Objectives:

The teacher will explain:

1	**Understand Java Fundamentals:** Gain a comprehensive understanding of Java's history, environment, source file structure, and compilation process, along with fundamental programming structures including classes, methods, and control flow.
2	**Master Object-Oriented Programming Concepts**: Learn and apply core object-oriented programming principles such as inheritance, encapsulation, polymorphism, and abstraction, including the use of interfaces and abstract classes.
3	**Implement Exception Handling:** Develop skills in managing exceptions in Java, including understanding different types of exceptions, control flow in exception handling, and the use of try-catch-finally blocks.
4	**Utilize Java Collections Framework:** Explore the Java Collections Framework, including various collection types (List, Set, Map), their interfaces, and how to effectively use iterators and sorting mechanisms.
5	**Leverage Spring Framework and Spring Boot:** Understand the fundamentals of the Spring Framework, including dependency injection and aspect-oriented programming, and learn to build RESTful web services using Spring Boot

Pre- requisite:

1. Introduction to Programming Concepts: Understanding fundamental programming concepts like variables, loops, and functions.
2. Basic Knowledge of installing JVM and any editor and working on command prompt.
3. Java Extension Pack: Install this extension pack in VS Code to enable Java support

Course Outcome of Practical

Course Outcome (CO) At the end of course, the student will be able to:		Bloom's Level (L)
CO 1	Develop and execute Java programs using Java compiler and Eclipse IDE	**L3**
CO 2	Demonstrate understanding of Object-Oriented Programming (OOP) concepts in Java.	**L3**
CO 3	Implement error handling techniques using exception handling and multithreading in Java.	**L4**
CO 4	Use and apply the Java Collections Framework, focusing on different collection types (List, Set, Map), their respective interfaces, and utilizing iterators and sorting techniques effectively.	**L4**
CO 5	Build Java applications using Spring Framework and test RESTful web services with Spring Boot.	**L4**

CO-PO-PSO Mapping

COURSE OUTCOME	COs	PROGRAM OUTCOMES(POs) & PROGRAM SPECIFIC OUTCOME(PSO)												
		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
	CO1	3	3	3	2	2	–	–	–	1	1	2	3	2	1
	CO2	3	3	3	2	2	–	–	–	1	1	2	3	2	1
	CO3	3	3	2	3	2	–	–	–	1	1	2	3	1	1
	CO4	2	2	2	2	3	–	–	–	1	1	2	3	3	1
	CO5	3	3	3	3	3	–	–	–	2	1	3	3	1	2
CO (Average)		3	3	3	2	2	–	–	–	1	1	2	3	2	1

S. No	List of Programs as per AKTU
1	Use Java compiler and eclipse platform to write and execute java program.
2	Creating simple java programs using command line arguments.
3	Understand OOP concepts and basics of Java programming.
4	Create Java programs using inheritance and polymorphism.
5	Implement error-handling techniques using exception handling and multithreading.
6	Create java program with the use of java packages
7	Construct java program using Java I/O package.
8	Create industry oriented application using Spring Framework.
9	Test RESTful web services using Spring Boot.
10	Test Frontend web application with Spring Boot

S. No	List of Programs with enhancement of faculty
1	a. Use Java compiler and any editor like eclipse or VS Code platform to write and execute java program. b. Creating simple java programs using command line arguments.
2	a. Demonstrate key OOP concepts such as inheritance, and polymorphism (Overloading and Overriding) with simple examples. b. Demonstrate encapsulation and abstraction with simple examples.
3	a. Demonstrate try-catch, finally, and throws in exception handling. b. Implement multithreading by creating a program where two threads calculate Fibonacci numbers simultaneously, and handle potential issues (e.g., race conditions) with synchronization.
4	a. Write a program to read from a file and display its contents on the console. b. Write a program to write user input to a file.
5	Implement the interface using both a traditional anonymous inner class and a lambda expression.
6	Demonstrate the use of try-with-resources to handle automatic resource management with BufferedReader, FileReader, or database connections.
7	Implement programs that demonstrate the use of various collection classes at different levels of the hierarchy (e.g., Collection, List, Set, Queue, Map).
8	Write a program that demonstrates the use of the Iterator interface to traverse a List and a Set. Use hasNext() and next() methods to iterate over collections like ArrayList and HashSet.
9	Test RESTful web services using Spring Boot.
10	Test Frontend web application with Spring Boot.
	List of Value added Program
	Implement the consumer producer problem with the synchronized methods produce() and consume(), with proper wait-notify mechanisms.

INDEX

S. No	Program Name	CO	BTL	Date	Sign
1	a.Use Java compiler and any editor like eclipse or VS Code platform to write and execute java program. b. Creating simple java programs using command line arguments.	CO1	L3		
2	a.Demonstrate key OOP concepts such as inheritance, and polymorphism (Overloading and Overriding) with simple examples. b. Demonstrate encapsulation and abstraction with simple examples.	CO1	L3		
3	a.Demonstrate try-catch, finally, and throws in exception handling. b. Implement multithreading by creating a program where two threads calculate Fibonacci numbers simultaneously, and handle potential issues (e.g., race conditions) with synchronization.	CO2	L3		
4	a.Write a program to read from a file and display its contents on the console b. Write a program to write user input to a file.	CO2	L3		
5	Implement the interface using both a traditional anonymous inner class and a lambda expression.	CO3	L4		
6	Demonstrate the use of try-with-resources to handle automatic resource management with BufferedReader, FileReader, or database connections.	CO3	L4		
7	Implement programs that demonstrate the use of various collection classes at different levels of the hierarchy (e.g., Collection, List, Set, Queue, Map).	CO4	L4		
8	Write a program that demonstrates the use of the Iterator interface to traverse a List and a Set. Use hasNext() and next() methods to iterate over collections like ArrayList and HashSet.	CO4	L4		
9	Test RESTful web services using Spring Boot.	CO5	L4		
10	Test Frontend web application with Spring Boot.	CO5	L4		
11	Implement the consumer producer problem with the synchronized methods produce() and consume(), with proper wait-notify mechanisms.	CO2	L3		

Softwares & Hardwares Required

1. Computer- Multi-core processor, Min 8 GB RAM, Min 256 GB SSD storage, monitor
2. Input Devices- keyboard, mouse
3. Development Environment- Operating System: Windows, macOS, or Linux
4. Development Tools:
   a) Text Editor/IDE: Visual Studio Code, Sublime Text, or Eclipse for writing Java code.
   b) Java Development Kit (JDK): JDK (version 8 or higher) for developing Java applications, including Servlets and Java Beans.

Practical No. 1(a)

1. **Experiment Description:** To use java compiler and editor VSCode to create and run program

Setting Up the Environment:

- Install JDK and set up the environment variables (JAVA_HOME and PATH) to include the JDK bin directory.
- Open Visual Studio Code and install the Java Extension Pack by searching for it in the Extensions Marketplace.

Creating a New Java Project:

- Open VS Code and create a new folder for your Java project.
- Open the folder in VS Code by selecting File > Open Folder.
- Create a new file with a .java extension (e.g., HelloWorld.java).

Compiling the Java Program:

- Open the integrated terminal in VS Code by selecting Terminal > New Terminal.
- Navigate to the directory where your Java file is located (if not already there).
- Compile the Java program using the following command: javac HelloWorld.java
- This command will generate a HelloWorld.class file if there are no syntax errors.

Executing the Java Program:

- After successful compilation, run the program using the following command: java HelloWorld

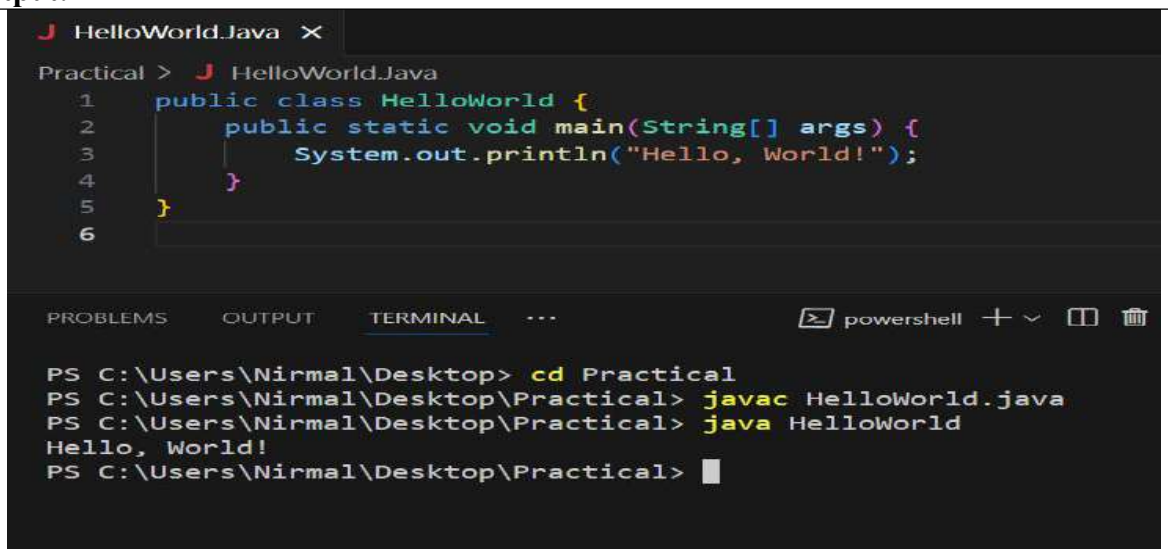
2. **Objective:** Use Java compiler and any editor like eclipse or VS Code platform to write and execute java program.

3. **Code and Procedure:**

HelloWorld.java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

4. **Output:**



Output of HelloWorld.java file showing result in terminal of VSCode

5. **Outcome of the Practical:** By the end of this practical, student will be able to Set up a Java development environment using Visual Studio Code. Write, compile, and execute a simple Java program. Understand the basic structure of a Java program, including the main method. Identify and resolve common compilation errors. Utilize the debugging tools available in VS Code to troubleshoot their code. Gain confidence in modifying and extending Java programs for more complex tasks.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**

- What are the steps to install and set up the Java Development Kit (JDK) on your machine?
- How do you install the Java Extension Pack in Visual Studio Code, and why is it important for Java development?
- Can you describe the basic structure of a Java program? What are the key components that must be included?
- What is the purpose of the main method in a Java program, and what is its signature?
- What command do you use to compile a Java program in Visual Studio Code, and what does this command do?
- If you encounter a compilation error, what steps would you take to identify and resolve the issue?
- What are some common compilation errors you might face when writing Java code, and how can you fix them?
- How can you use the debugging tools in Visual Studio Code to troubleshoot your Java code? Can you name at least two debugging features?
- What modifications can you make to a simple Java program to enhance its functionality? Can you provide an example?
- Why is it important to gain confidence in modifying and extending Java programs? How does this skill benefit your overall programming ability?

Practical No. 1(b)

1. **Experiment Description:** To Crete a simple java programs using command line arguments

Steps to Follow:

- Creating a New Java File and Open your text editor Notepad++ or any IDE.
- Create a new file named CommandLineArgsExample.java.

Compiling the Java Program:

- Open the command line or terminal.
- Navigate to the directory where your CommandLineArgsExample.java file is located.
- Compile the Java program using the command: javac CommandLineArgsExample.java
- This command will generate a CommandLineArgsExample.class file if there are no syntax errors.

Executing the Java Program with Command Line Arguments:

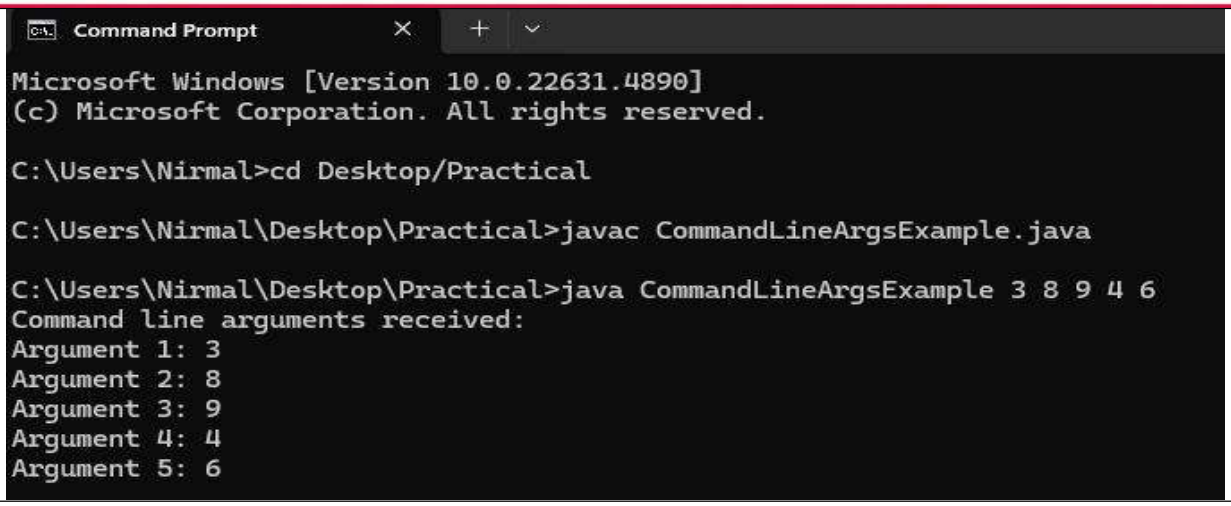
- Run the program and pass some command line arguments using the following command: java CommandLineArgsExample. Two possible two cases are listed:
- when arguments are given in command line - : java CommandLineArgsExample 3 8 9 4 6
- when arguments are not given in command line- : java CommandLineArgsExample

2. **Objective:** Creating simple java programs using command line arguments.
3. **Code and Procedure:**

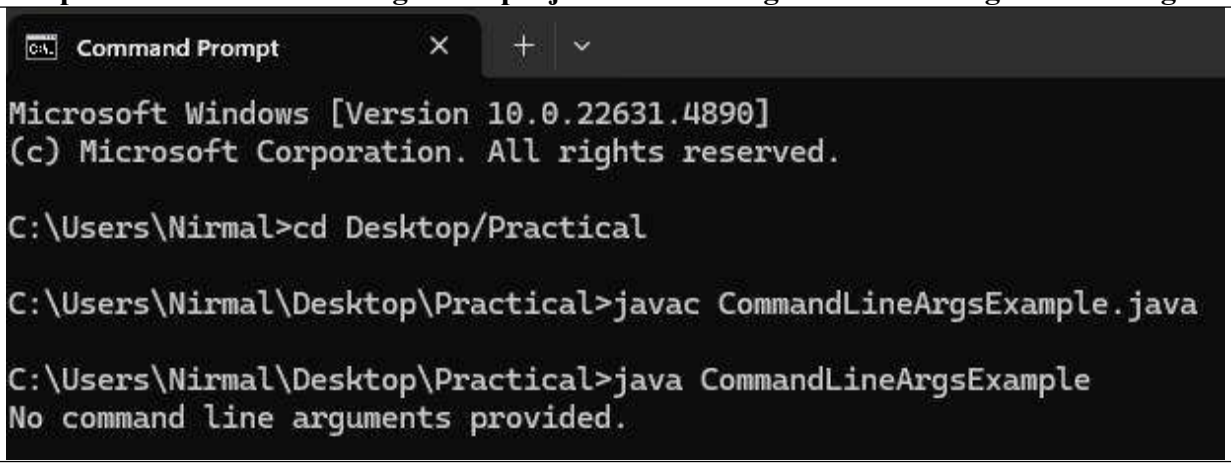
CommandLineArgsExample.java

```
public class CommandLineArgsExample {
    public static void main(String[] args) {
        // Check if any command line arguments are provided
        if (args.length == 0) {
            System.out.println("No command line arguments provided.");
        } else {
            System.out.println("Command line arguments received:");
            // Loop through the command line arguments and print them
            for (int i = 0; i < args.length; i++) {
                System.out.println("Argument " + (i + 1) + ": " + args[i]);
            }
        }
    }
}
```

4. **Output:**



Output of CommandLineArgsExample.java file showing result when arguments are given



1. **Output of CommandLineArgsExample.java file showing result when arguments are not given**

5. Outcome of the Practical: By the end of this practical, student will be able to create a Java program that accepts and processes command line arguments. Compile and run Java programs from the command line. Understand how to access and manipulate command line arguments within a Java program. Gain confidence in using command line arguments for various programming tasks.

6. Assessment Criteria:

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. Viva Questions/Review Questions:

- What is a command line argument in Java?
- How do you access command line arguments in a Java program?
- What is the name of the array that holds command line arguments in the main method?
- What is the data type of the command line arguments passed to a Java program?
- How do you compile a Java program that uses command line arguments?
- What command do you use to run a Java program with command line arguments?
- If you run a Java program without any command line arguments, what will the length of the args array be?

- Can command line arguments be of different data types (e.g., integers, strings) in Java? Why or why not?
- What happens if you try to access an index of the args array that does not exist?
- How can you print the first command line argument passed to a Java program?

Practical No. 2 (a)

1. **Experiment Description:** To demonstrate key Object-Oriented Programming (OOP) concepts, specifically inheritance and polymorphism (method overloading and method overriding), using a simple example involving shapes. Students will learn how to create a base class and derived classes, override methods, and overload methods in Java.

   Steps to Follow:
   - Create a New Java File:
   - Open your text editor or IDE.
   - Create a new file named ShapeExample.java.

   Write the Base Class: Shape:
   - In the ShapeExample.java file, start by defining the base class Shape with an overloaded method for displaying information and a method for calculating the area.

   Create Derived Classes: Circle and Rectangle:
   - Define the Circle class that extends Shape and overrides the area() and displayInfo() methods.
   - Define the Rectangle class that also extends Shape and overrides the area() and displayInfo() methods.

   Implement the Main Class:
   - In the same file, implement the main method to create instances of Circle and Rectangle, and demonstrate the functionality.

   Compile the Java Program:
   - Open the command line or terminal.
   - Navigate to the directory where your ShapeExample.java file is located.
   - Compile the Java program using the following command:javac ShapeExample.java

   Run the Java Program: After compilation, use the command to run: java ShapeExample

2. **Objective:** Demonstrate key OOP concepts such as inheritance, and polymorphism (Overloading and Overriding) with simple examples.
3. **Code and Procedure:**

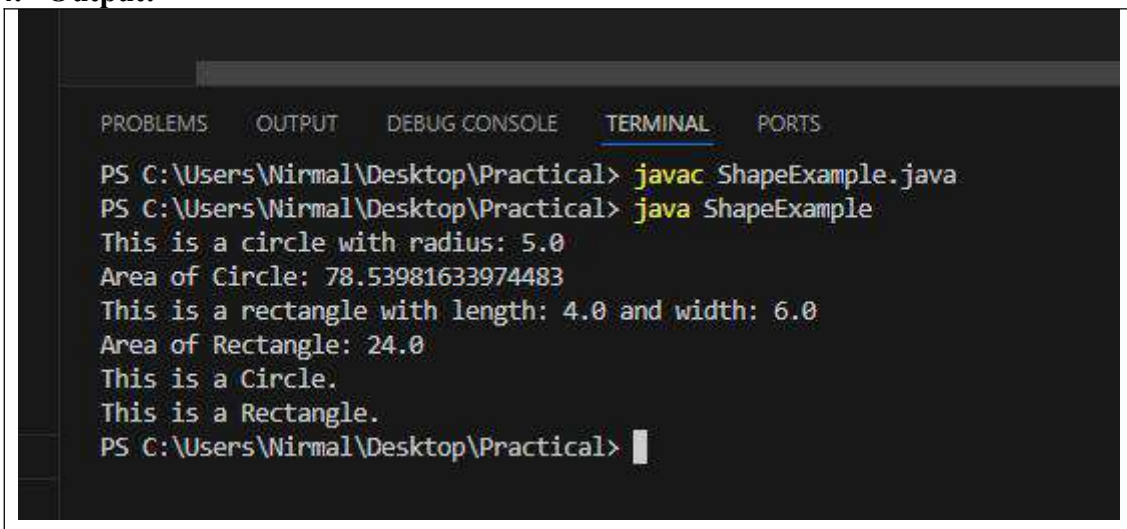
ShapeExample.java

```
// Base class
class Shape {
  // Overloaded method to display shape information
  void displayInfo() {
    System.out.println("This is a shape.");
```

```
    }
    void displayInfo(String name) {
        System.out.println("This is a " + name + ".");
    }
    // Method to calculate area (to be overridden)
    double area() {
        return 0; // Default implementation
    }
}
// Derived class: Circle –  Inheritance is done here
class Circle extends Shape {
    double radius;
    Circle(double radius) {
        this.radius = radius;
    }
    // Overriding the area method
    @Override
    double area() {
        return Math.PI * radius * radius; // Area of circle
    }
    // Overriding the displayInfo method
    @Override
    void displayInfo() {
        System.out.println("This is a circle with radius: " + radius);
    }
}
// Derived class: Rectangle
class Rectangle extends Shape {
    double length;
    double width;
    Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }
    // Overriding the area method
    @Override
    double area() {
        return length * width; // Area of rectangle
    }
    // Overriding the displayInfo method
    @Override
    void displayInfo() {
        System.out.println("This is a rectangle with length: " + length + " and width: " + width);
    }
}
// Main class to demonstrate the functionality
public class ShapeExample {
    public static void main(String[] args) {
        Shape myCircle = new Circle(5); // Create a Circle object
        Shape myRectangle = new Rectangle(4, 6); // Create a Rectangle object
        // Display information and area for Circle
```

```
        myCircle.displayInfo(); // Calls the overridden method in Circle
        System.out.println("Area of Circle: " + myCircle.area()); // Calls the overridden area method
        // Display information and area for Rectangle
        myRectangle.displayInfo(); // Calls the overridden method in Rectangle
        System.out.println("Area of Rectangle: " + myRectangle.area()); // Calls the overridden area
method
        // Using the overloaded method in Shape
        myCircle.displayInfo("Circle"); // Calls the overloaded method in Shape
        myRectangle.displayInfo("Rectangle"); // Calls the overloaded method in Shape
    }
}
```

4. **Output:**



Output of ShapeExample.java file showing Overloading by displayInfo() , Overriding by area() and inheritance of shape class by Circle and Rectangle Class

5. **Outcome of** the Practical: Students will learn how to override and Overload methods in classes to provide specific implementations, demonstrating polymorphism in action. Also, they understand how to create a base class and derived classes, as the concept of inheritance enabling code reuse and establishing hierarchical relationships between classes and thus will build confidence in modifying and extending Java applications, preparing them for more complex programming challenges in future projects.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**

- What is inheritance in Java, and why is it used?
- What is the difference between a base class (parent class) and a derived class (child class)?
- How do you declare a class that inherits from another class in Java?

- What is polymorphism in Java, and how does it enhance code flexibility?
- What are the two types of polymorphism in Java? Can you briefly explain each?
- What is method overloading, and how does it differ from method overriding?
- Can you overload a method by changing only its return type? Why or why not?
- What is method overriding, and when would you use it?
- What are the rules for overriding a method in Java?
- How does polymorphism allow for dynamic method resolution in Java?

Practical No. 2 (b)

1. **Experiment Description:** To illustrate the key Object-Oriented Programming (OOP) concepts of encapsulation and abstraction using a simple banking application. Students will learn how to encapsulate data within classes and abstract complex functionalities, allowing for a clear and organized code structure.

Steps to Follow:

- Create a New Java File:
- Open your text editor or IDE.
- Create a new file named BankAccountExample.java.

Define the Abstract Class: BankAccount: Start by defining an abstract class BankAccount that will serve as the blueprint for different types of bank accounts. This class will encapsulate the account number and balance as private fields and provide abstract methods for depositing and withdrawing money.

Create the Derived Class: SavingsAccount: Define a concrete class SavingsAccount that extends the BankAccount class. This class will implement the abstract methods for depositing and withdrawing money, while also encapsulating the logic for managing the account balance.

Implement the Main Class: In the same file, implement the main method to create an instance of SavingsAccount and demonstrate the functionality of depositing and withdrawing money.

Compile the Java Program:

- Open the command line or terminal.
- Navigate to the directory where your BankAccountExample.java file is located.
- Compile the Java program using the following command: javac BankAccountExample.java

Run the Java Program: After compilation, run the program using: java BankAccountExample

2. **Objective:** Demonstrate encapsulation and abstraction with simple examples.
3. **Code and Procedure:**

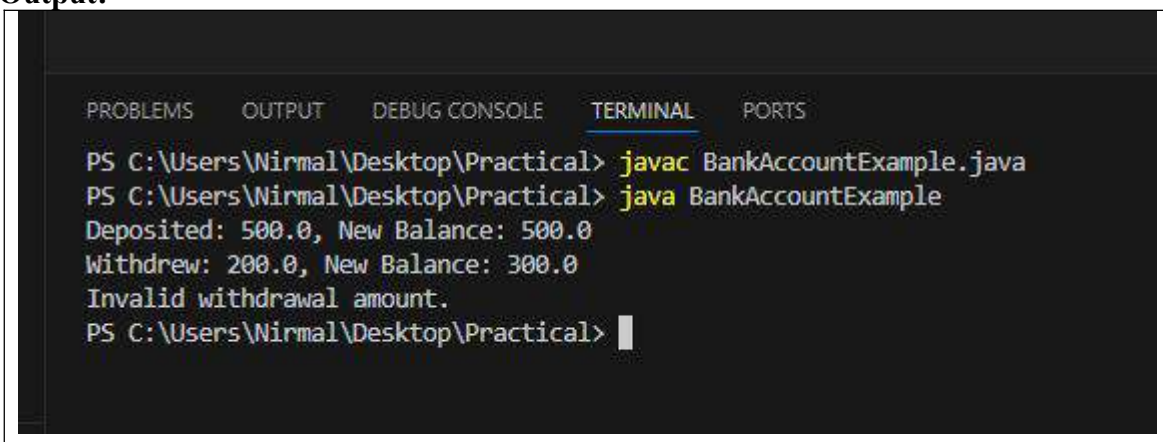
BankAccountExample.java

```
// Abstract class representing a generic bank account
abstract class BankAccount {
    // Encapsulated fields
    private String accountNumber;
    private double balance;
    // Constructor
    public BankAccount(String accountNumber) {
```

```
        this.accountNumber = accountNumber;
        this.balance = 0.0; // Initial balance is 0
    }
    // Method to get the account number (getter)
    public String getAccountNumber() {
        return accountNumber;
    }
    // Method to get the balance (getter)
    public double getBalance() {
        return balance;
    }
    // Abstract method to deposit money (to be implemented by subclasses)
    public abstract void deposit(double amount);
    // Abstract method to withdraw money (to be implemented by subclasses)
    public abstract void withdraw(double amount);
}
// Concrete class representing a savings account
class SavingsAccount extends BankAccount {
    // Constructor
    public SavingsAccount(String accountNumber) {
        super(accountNumber);
    }
    // Implementing the deposit method
    @Override
    public void deposit(double amount) {
        if (amount > 0) {
            double newBalance = getBalance() + amount;
            // Accessing the private balance field through a method
            setBalance(newBalance);
            System.out.println("Deposited: " + amount + ", New Balance: " + getBalance());
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }
    // Implementing the withdraw method
    @Override
    public void withdraw(double amount) {
        if (amount > 0 && amount <= getBalance()) {
            double newBalance = getBalance() - amount;
            setBalance(newBalance);
            System.out.println("Withdrew: " + amount + ", New Balance: " + getBalance());
        } else {
            System.out.println("Invalid withdrawal amount.");
        }
    }
    // Encapsulated method to set the balance
    private void setBalance(double balance) {
        // This method is private to restrict direct access
        // You can add additional logic here if needed
        this.balance = balance;
    }
```

```
}
// Main class to demonstrate the functionality
public class BankAccountExample {
    public static void main(String[] args) {
        // Creating a savings account
        SavingsAccount myAccount = new SavingsAccount("123456789");
        // Depositing money
        myAccount.deposit(500.0);
        // Withdrawing money
        myAccount.withdraw(200.0);
        // Attempting to withdraw more than the balance
        myAccount.withdraw(400.0);
    }
}
```

4. **Output:**



Output of BankAccountExample.java showing the concepts of encapsulation (use of setter and getter method) and abstraction for related methods

5. **Outcome of the Practical:** Students learn to encapsulate data using private fields in the BankAccount class, ensuring controlled access through public methods. They also learned the use of Setter and Getter methods .Also they gain the knowledge of abstraction by using an abstract class to define essential behaviors without revealing implementation details.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is encapsulation in Java? Can you provide a simple definition?
   - How does encapsulation enhance data security in a class?
   - What are the access modifiers used to implement encapsulation?
   - What is abstraction in Java, and how does it differ from encapsulation?
   - Can you explain the purpose of an abstract class in Java?
   - What is the difference between an abstract method and a concrete method?
   - How do you declare an abstract class in Java?

- In the provided program, how is the account balance encapsulated?
- Why is the setBalance method private in the SavingsAccount class?
- How does using an abstract class promote code reusability in Java?

Practical No. 3 (a)

1. **Experiment Description**: Exception handling is a powerful mechanism in Java that allows developers to manage runtime errors, ensuring that the program can continue executing or terminate gracefully. This practical will cover three key components of exception handling:

try-catch: Used to catch exceptions that may occur during the execution of a block of code.
finally: A block that executes after the try-catch blocks, regardless of whether an exception was thrown or caught.
throws: A keyword used in method signatures to indicate that a method may throw exceptions, allowing the caller to handle them

Program Structure: Main Class:
The main class will contain a method that performs division and may throw an ArithmeticException if division by zero occurs.
It will also demonstrate the use of try-catch to handle exceptions, a finally block to execute cleanup code, and the throws keyword in method signatures.

- The divide method takes two integers as parameters and performs division. It declares that it may throw an ArithmeticException using the throws keyword.
- In the main method, a try block attempts to call the divide method with a denominator of zero, which will throw an exception.
- The catch block catches the ArithmeticException and prints an error message.
- The finally block executes regardless of whether an exception was thrown, indicating that the execution of the try-catch block is complete.
- The program also demonstrates a successful division operation to show how the exception handling works when no error occurs.

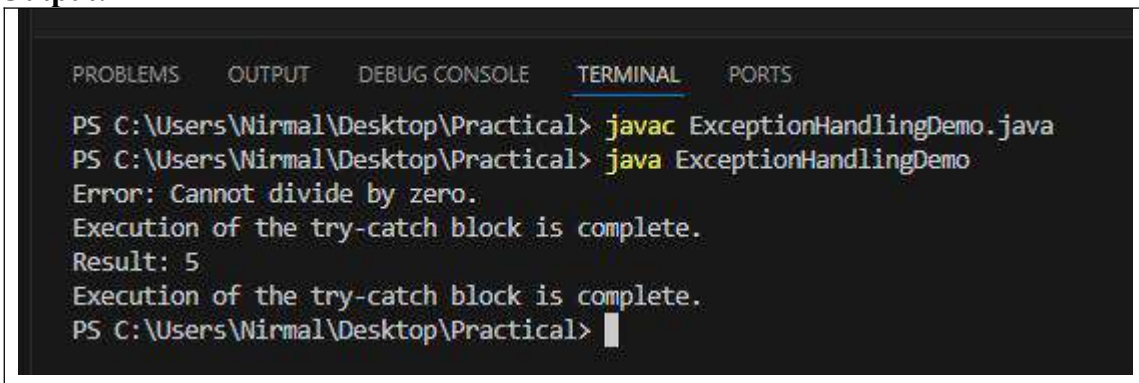
2. **Objective:** Demonstrate try-catch, finally, and throws in exception handling.
3. **Code and Procedure:**

ExceptionHandlingDemo.java

```
public class ExceptionHandlingDemo {
    // Method that may throw an exception
    public static int divide(int numerator, int denominator) throws ArithmeticException {
        return numerator / denominator; // May throw ArithmeticException
```

```
    }
    public static void main(String[] args) {
        int result = 0;
        try {
            result = divide(10, 0); // This will throw an exception
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero.");
        } finally {
            System.out.println("Execution of the try-catch block is complete.");
        }
        // Demonstrating a successful division
        try {
            result = divide(10, 2); // This will not throw an exception
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero.");
        } finally {
            System.out.println("Execution of the try-catch block is complete.");
        }
    }
}
```

4. **Output:**



Output of the ExceptionHandlingDemo.java file for ArithmeticException

5. **Outcome of the Practical**: Students will learn to effectively manage runtime errors using try-catch blocks to prevent crashes and enhance user experience. They will understand the role of finally blocks for executing cleanup code, the significance of the throws keyword in method signatures, and how to provide meaningful error messages. They will also distinguish between checked and unchecked exceptions and adopt best practices in exception handling.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is exception handling in Java?
   - What is the purpose of the try block?
   - How does the catch block work in exception handling?

- What is the role of the finally block?
- What does the throws keyword indicate in a method signature?
- What happens if an exception is not caught in a program?
- Can you have multiple catch blocks for a single try block? If so, why?
- What types of exceptions can be handled using the catch block?
- How can you create a custom exception in Java?
- Why is it important to use exception handling in Java applications?

Practical No. 3 (b)

1. **Experiment Description:** The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, typically starting with 0 and 1. The sequence can be defined recursively, which makes it a suitable candidate for demonstrating multithreading. In this practical, we will create a FibonacciCalculator class to compute Fibonacci numbers, a FibonacciThread class to run calculations in separate threads, and a Main class to execute the program.

Threads: A thread is the smallest unit of processing that can be scheduled by an operating system. In Java, threads can be created by extending the Thread class or implementing the Runnable interface.

Implementation: The FibonacciThread class extends the Thread class and overrides the run() method. This method contains the logic to execute the Fibonacci calculation by invoking the run() method of the FibonacciCalculator instance.

Multithreading: Multithreading allows multiple threads to run concurrently, enabling a program to perform multiple operations simultaneously. This is particularly useful for tasks that can be executed independently, such as calculating Fibonacci numbers for different inputs.

Implementation: In our program, we create two threads to calculate Fibonacci numbers for different values (10 and 15) at the same time, demonstrating how multithreading can improve performance and responsiveness.

Synchronization: Synchronization is a mechanism that ensures that two or more concurrent threads do not simultaneously execute a particular section of code, which can lead to inconsistent data or race conditions.

Race Condition: A race condition occurs when multiple threads access shared data and try to change it at the same time. If not properly synchronized, this can lead to unpredictable results.

Implementation: The calculateFibonacci method in the FibonacciCalculator class is declared as synchronized. This ensures that only one thread can execute this method at a time, preventing race conditions when calculating Fibonacci numbers.

Program Structure: FibonacciCalculator Class:

Attributes: private int n: Holds the Fibonacci number to be calculated.

Constructor: public FibonacciCalculator(int n): Initializes the n attribute with the provided value.

Methods:

- public synchronized long calculateFibonacci(int num): A synchronized method that calculates the Fibonacci number for a given integer num. The method uses recursion to compute the Fibonacci value.
- public void run(): Calls calculateFibonacci(n) and prints the result to the console.

FibonacciThread Class:
Attributes: private FibonacciCalculator calculator: Holds a reference to a FibonacciCalculator instance.
Constructor: public FibonacciThread(FibonacciCalculator calculator): Initializes the calculator attribute with the provided FibonacciCalculator instance.
Methods: @Override public void run(): Overrides the run method of the Thread class to execute the Fibonacci calculation.
Main Method: public static void main(String[] args): The entry point of the program. It creates two instances of FibonacciCalculator for different Fibonacci numbers (10 and 15), creates two FibonacciThread instances, starts the threads, and waits for both threads to finish using join().

2. **Objective:** Implement multithreading by creating a program where two threads calculate Fibonacci numbers simultaneously, and handle potential issues (e.g., race conditions) with synchronization.

3. **Code and Procedure:**

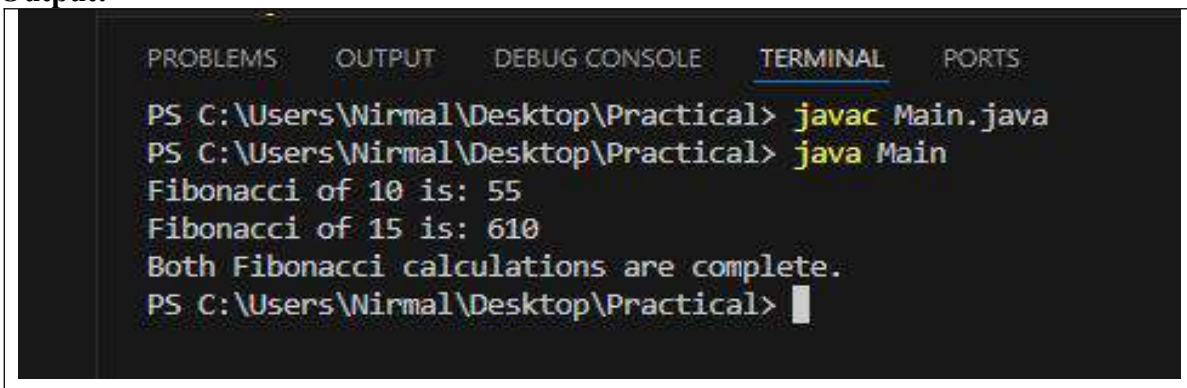
Main.java

```
class FibonacciCalculator {
    private int n;
    public FibonacciCalculator(int n) {
        this.n = n;
    }
    // Synchronized method to calculate Fibonacci
    public synchronized long calculateFibonacci(int num) {
        if (num <= 1) {
            return num;
        }
        return calculateFibonacci(num - 1) + calculateFibonacci(num - 2);
    }
    public void run() {
        long result = calculateFibonacci(n);
        System.out.println("Fibonacci of " + n + " is: " + result);
    }
}
class FibonacciThread extends Thread {
    private FibonacciCalculator calculator;

    public FibonacciThread(FibonacciCalculator calculator) {
        this.calculator = calculator;
    }
    @Override
    public void run() {
        calculator.run();
    }
}
public class Main {
    public static void main(String[] args) {
```

```
        // Create two FibonacciCalculator instances for different Fibonacci numbers
        FibonacciCalculator calculator1 = new FibonacciCalculator(10);
        FibonacciCalculator calculator2 = new FibonacciCalculator(15);
        // Create two threads
        FibonacciThread thread1 = new FibonacciThread(calculator1);
        FibonacciThread thread2 = new FibonacciThread(calculator2);
        // Start the threads
        thread1.start();
        thread2.start();
        // Wait for both threads to finish
        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Both Fibonacci calculations are complete.");
    }
}
```

4. **Output:**



Output of the Main.java file

5. **Outcome of the Practical:** Students will gain hands-on experience with key Java concepts such as multithreading, synchronization, and thread management. They will learn how to create and manage threads to execute tasks concurrently, enhancing their understanding of parallel processing. They will also understand the importance of synchronization in preventing race conditions when multiple threads access shared resources. This practical will also highlight the challenges of recursive algorithms in a multithreaded context, encouraging students to think critically about performance optimization and thread safety in real-world applications.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	4 Marks

7. **Viva Questions/Review Questions:**
   - What is multithreading in Java?
   - How do you create a thread in Java?
   - What is the purpose of the synchronized keyword in Java?

- What is a race condition?
- In the Fibonacci calculator program, why is the calculateFibonacci method synchronized?
- What is the output of the Fibonacci calculator for the input value of 10?
- What happens if you remove the synchronized keyword from the calculateFibonacci method?
- What is the purpose of the join() method in the main class?
- Can you explain the structure of the FibonacciThread class?
- What are some potential optimizations for calculating Fibonacci numbers?

Practical No. 4 (a)

1. **Experiment Description**: File handling is a crucial aspect of programming that allows applications to interact with external data sources. In this practical, students will create a Java program that reads a text file line by line and prints its contents to the console. This exercise will help students understand how to work with files, handle exceptions, and manage resources effectively.

Program Structure: File Creation: Before running the program, students should create a text file (e.g., sample.txt) with some sample text content.
Main Class: The main class will contain the logic to read from the specified file and display its contents.

- The program imports necessary classes for file handling: BufferedReader, FileReader, and IOException.
- The filePath variable specifies the location of the text file to be read.
- A try-with-resources statement is used to create a BufferedReader object, which automatically closes the resource after use.
- The program reads the file line by line using a while loop and prints each line to the console.
- If an IOException occurs (e.g., if the file is not found), an error message is displayed

2. **Objective:** Write a program to read from a file and display its contents on the console.

3. **Code and Procedure:**

FileReadDemo.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

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

```
        String filePath = "C:/Users/Nirmal/Desktop/Practical/sample.txt"; // Specify the path to the file

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line); // Display each line on the console
            }
        }
        catch (IOException e) {
            System.out.println("An error occurred while reading the file: " + e.getMessage());
        }
    }
}
```

4. **Output:**



Output of FileReadDemo.java file reading the data line by line from sample.txt file

5. **Outcome of the Practical:** Students will learn how to read data from files in Java, utilize file handling classes such as FileReader and BufferedReader, and implement exception handling to manage potential errors. They will also gain experience in using the try-with-resources statement for efficient resource management, enhancing their overall programming skills in file operations.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**

- What is the purpose of the FileReader class in Java?
- How does the BufferedReader class improve file reading performance?
- What is the significance of using a try-with-resources statement?
- How do you handle exceptions that may occur during file reading?
- What happens if the specified file does not exist when running the program?
- Can you explain the difference between FileReader and FileInputStream?
- How would you read a file character by character instead of line by line?
- What is the role of the readLine() method in the BufferedReader class?
- How can you modify the program to read from a different file?
- Why is it important to close file resources after use?

Practical No. 4 (b)

1. **Experiment Description**: Using Java's file handling capabilities, including the FileWriter and BufferedWriter classes, to capture user input and save it to a text file. students will create a Java program that prompts the user for input, captures that input, and writes it to a specified text file. This exercise will help students understand how to work with files, handle exceptions, and manage resources effectively.

Program Structure:
User Input: The program will prompt the user to enter text that they want to save to a file.
File Writing: The program will use FileWriter and BufferedWriter to write the user input to a specified file.

- The program imports necessary classes for file handling: BufferedWriter, FileWriter, IOException, and Scanner.
- A Scanner object is created to capture user input from the console.
- The filePath variable specifies the name of the output file where the user input will be saved.
- A try-with-resources statement is used to create a BufferedWriter object, which automatically closes the resource after use.
- The program prompts the user to enter text, and it continues to read input until the user types "exit".
- Each line of input is written to the file, followed by a new line.
- If an IOException occurs (e.g., if there is an issue with file writing), an error message is displayed.
- The finally block ensures that the Scanner resource is closed after use.

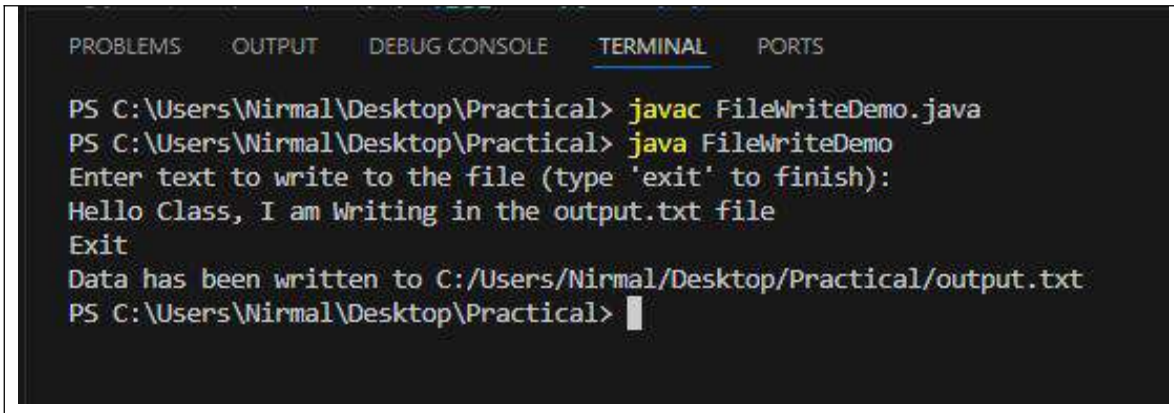
2. **Objective:** Write a program to write user input to a file
3. **Code and Procedure:**

FileWriteDemo.java

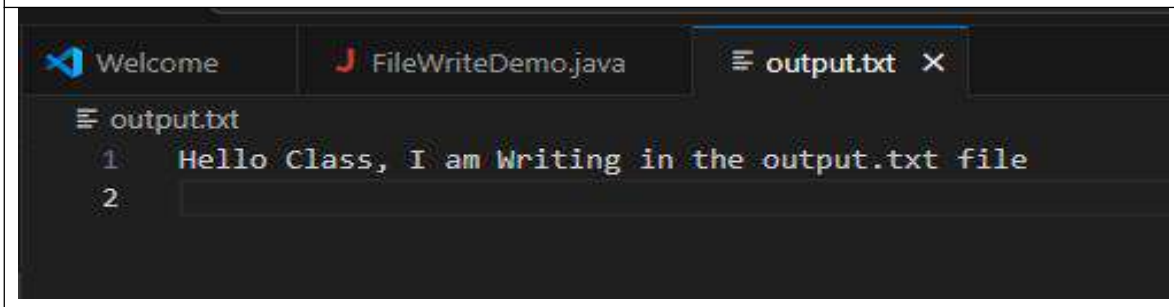
```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
```

```
import java.util.Scanner;
public class FileWriteDemo {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
         String filePath = "C:/Users/Nirmal/Desktop/Practical/output.txt"; // Specify the path to the output file
      System.out.println("Enter text to write to the file (type 'exit' to finish):");
      try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
         String input;
         while (!(input = scanner.nextLine()).equalsIgnoreCase("exit")) {
            writer.write(input);
            writer.newLine(); // Write a new line after each input
         }
         System.out.println("Data has been written to " + filePath);
      } catch (IOException e) {
         System.out.println("An error occurred while writing to the file: " + e.getMessage());
      } finally {
         scanner.close(); // Close the scanner resource
      }
   }
}
```

4. **Output:**



Output of FileWriteDemo.Java file showing written content in output.txt file



Output of output.txt file showing written content by FileWriteDemo.java file

5. **Outcome of the Practical:** Students will learn how to write data to files in Java, utilize file handling classes such as FileWriter and BufferedWriter, and implement exception handling to

manage potential errors. They will also gain experience in capturing user input and managing resources effectively, enhancing their overall programming skills in file operations.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is the purpose of the FileWriter class in Java?
   - How does the BufferedWriter class improve file writing performance?
   - What is the significance of using a try-with-resources statement?
   - How do you handle exceptions that may occur during file writing?
   - What happens if the specified file already exists when writing to it?
   - Can you explain the difference between FileWriter and PrintWriter?
   - How would you modify the program to append data to an existing file instead of overwriting?
   - What is the role of the newLine() method in the BufferedWriter class?
   - How can you change the output file name based on user input?
   - Why is it important to close file resources after use?

Practical No. 5

1. **Experiment Description:** Interfaces are used to define a contract that classes can implement. With the introduction of lambda expressions in Java 8, developers can write more concise and readable code, especially when working with functional interfaces (interfaces with a single abstract method). This practical will involve creating a simple functional interface and implementing it using both an anonymous inner class and a lambda expression.

Program Structure:

Functional Interface: Define a functional interface with a single abstract method.

Anonymous Inner Class Implementation: Implement the interface using an anonymous inner class.

Lambda Expression Implementation: Implement the same interface using a lambda expression.

Functional Interface: The Greeting interface is defined with a single abstract method greet(String name). The @FunctionalInterface annotation indicates that this interface is intended to be a functional interface.

Anonymous Inner Class: In the main method, an instance of the Greeting interface is created using an anonymous inner class. The greet method is overridden to provide the implementation, which prints a greeting message.

Lambda Expression: The same interface is implemented using a lambda expression, which provides a more concise way to define the greet method. The lambda expression takes a single parameter name and prints the greeting message.

When the program is executed, it will display two greeting messages: one from the anonymous inner class implementation and another from the lambda expression implementation. This practical will help students understand how to implement interfaces in different ways and appreciate the simplicity and readability of lambda expressions.

2. **Objective:** Implement the interface using both a traditional anonymous inner class and a lambda expression.
3. **Code and Procedure:**

InterfaceImplementationDemo.java

```
// Step 1: Define a functional interface
@FunctionalInterface
interface Greeting {
    void greet(String name);
}
public class InterfaceImplementationDemo {
    public static void main(String[] args) {
        // Step 2: Implementing the interface using an anonymous inner class
        Greeting greetingAnonymous = new Greeting() {
            @Override
            public void greet(String name) {
                System.out.println("Hello, " + name + "! (from Anonymous Inner Class)");
            }
        };
        greetingAnonymous.greet("Alice");
        // Step 3: Implementing the interface using a lambda expression
        Greeting greetingLambda = (name) -> System.out.println("Hello, " + name + "! (from
Lambda Expression)");
        greetingLambda.greet("Bob");
    }
}
```

4. **Output:**



Output of InterfaceImplementationDemo.Java file showing results for Anonymous Inner Class and Lambda Expression

5. **Outcome of the Practical:** Students will learn how to define and implement functional interfaces in Java. They will understand the differences between traditional anonymous inner classes and lambda expressions, including the benefits of using lambda expressions for cleaner and more concise code. Additionally, students will gain experience in functional programming concepts and how they can be applied in Java.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is a functional interface in Java?
   - How does the @FunctionalInterface annotation help in defining a functional interface?
   - What are the advantages of using lambda expressions over anonymous inner classes?
   - Can a functional interface have multiple abstract methods? Why or why not?
   - How do you implement a functional interface using an anonymous inner class?
   - What is the syntax of a lambda expression in Java?
   - Can lambda expressions access variables from the enclosing scope? If so, how?
   - What are some common functional interfaces provided by the Java standard library?
   - How do lambda expressions improve code readability and maintainability?
   - Can you provide an example of a situation where using a lambda expression would be more beneficial than using an anonymous inner class?

Practical No. 6

1. **Experiment Description:** Managing resources such as file streams, database connections, and network sockets is crucial for building robust applications. The try-with-resources statement, introduced in Java 7, simplifies resource management by automatically closing resources when they are no longer needed. This practical will involve reading data from a file using BufferedReader and FileReader within a try-with-resources block.

Program Structure:

File Creation: Before running the program, students should create a text file (e.g., input.txt) with some sample text content.

Resource Management: The program will read the contents of the file using BufferedReader and FileReader within a try-with-resources statement.

- The program imports necessary classes for file handling: BufferedReader, FileReader, and IOException.
- The filePath variable specifies the location of the text file to be read.
- A try-with-resources statement is used to create a BufferedReader object, which automatically closes the resource after use.
- The program reads the file line by line using a while loop and prints each line to the console.
- If an IOException occurs (e.g., if the file is not found), an error message is displayed.
- The BufferedReader is automatically closed at the end of the try block, ensuring that resources are managed efficiently.
- When the program is executed, it will read the contents of input.txt and display them on the console. If the file does not exist or an error occurs during reading, an appropriate error message will be shown. This practical will help students understand the benefits of using try-with-resources for automatic resource management and how it simplifies code by eliminating the need for explicit resource closing.

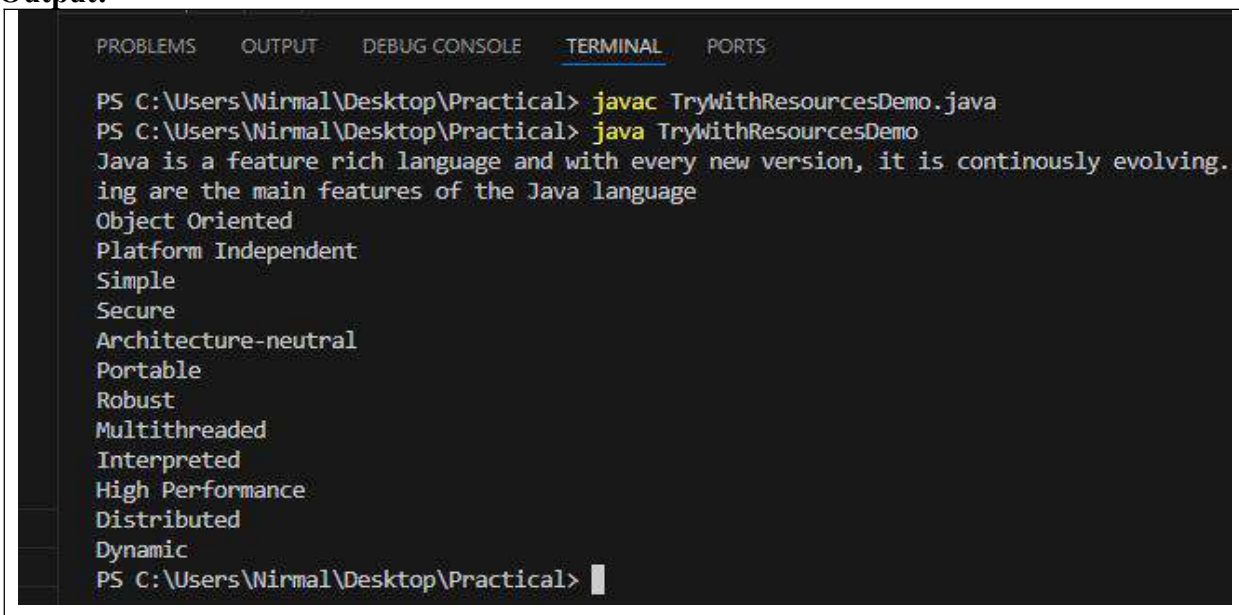
2. **Objective:** Demonstrate the use of try-with-resources to handle automatic resource management with BufferedReader, FileReader, or database connections.
3. **Code and Procedure:**

TryWithResourcesDemo.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class TryWithResourcesDemo {
   public static void main(String[] args) {
      String filePath = "input.txt"; // Specify the path to the input file
      // Step 1: Using try-with-resources for automatic resource management
      try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
         String line;
         // Step 2: Read the file line by line
         while ((line = br.readLine()) != null) {
            System.out.println(line); // Display each line on the console
         }
      } catch (IOException e) {
         System.out.println("An error occurred while reading the file: " + e.getMessage());
      }
      // No need to explicitly close BufferedReader; it is closed automatically
   }
}
```

4. **Output:**



Output of TryWithResourcesDemo.java file showing results input.txt content



Output of TryWithResourcesDemo.java file showing error results when input.txt file was not available

5. **Outcome of the Practical:** Students will learn how to use the try-with-resources statement in Java for effective resource management. They will understand how this feature helps prevent resource leaks and simplifies error handling. Additionally, students will gain experience in file input operations and exception handling, enhancing their overall programming skills.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**

- What is the purpose of the try-with-resources statement in Java?
- How does try-with-resources help in managing resources?
- What types of resources can be used with try-with-resources?
- Can you explain the difference between try-with-resources and traditional try-catch-finally blocks?
- What happens if an exception occurs within the try block?
- How does Java ensure that resources are closed automatically in try-with-resources?
- Can you use multiple resources in a single try-with-resources statement? If so, how?
- What is the significance of the AutoCloseable interface in relation to try-with-resources?
- How would you modify the program to handle multiple files using try-with-resources?
- Can you provide an example of a situation where not using try-with-resources could lead to resource leaks?

Practical No. 7

1. **Experiment Description:** The Java Collection Framework provides a set of classes and interfaces for storing and manipulating groups of objects. This practical will involve creating programs that utilize different collection classes to illustrate their functionalities and use cases. Students will implement examples using ArrayList, HashSet, PriorityQueue, and HashMap.

Program Structure:

List Implementation: Use ArrayList to demonstrate dynamic array capabilities.

Set Implementation: Use HashSet to demonstrate unique element storage.

Queue Implementation: Use PriorityQueue to demonstrate priority-based ordering.

Map Implementation: Use HashMap to demonstrate key-value pair storage.

ArrayList: Demonstrates a dynamic array that can grow in size. Elements can be added, accessed, and printed.

HashSet: Demonstrates a collection that does not allow duplicate elements. It shows how to add elements and the result of trying to add a duplicate.

PriorityQueue: Demonstrates a queue that orders elements based on their natural ordering or a specified comparator. It shows how to add elements and poll the head of the queue.

HashMap: Demonstrates a collection that stores key-value pairs. It shows how to add entries and retrieve values based on keys.

When the program is executed, it will demonstrate the functionalities of various collection classes in Java. Students will see how to use ArrayList, HashSet, PriorityQueue, and HashMap to manage data effectively. This practical will help students understand the differences between these collections and when to use each type.

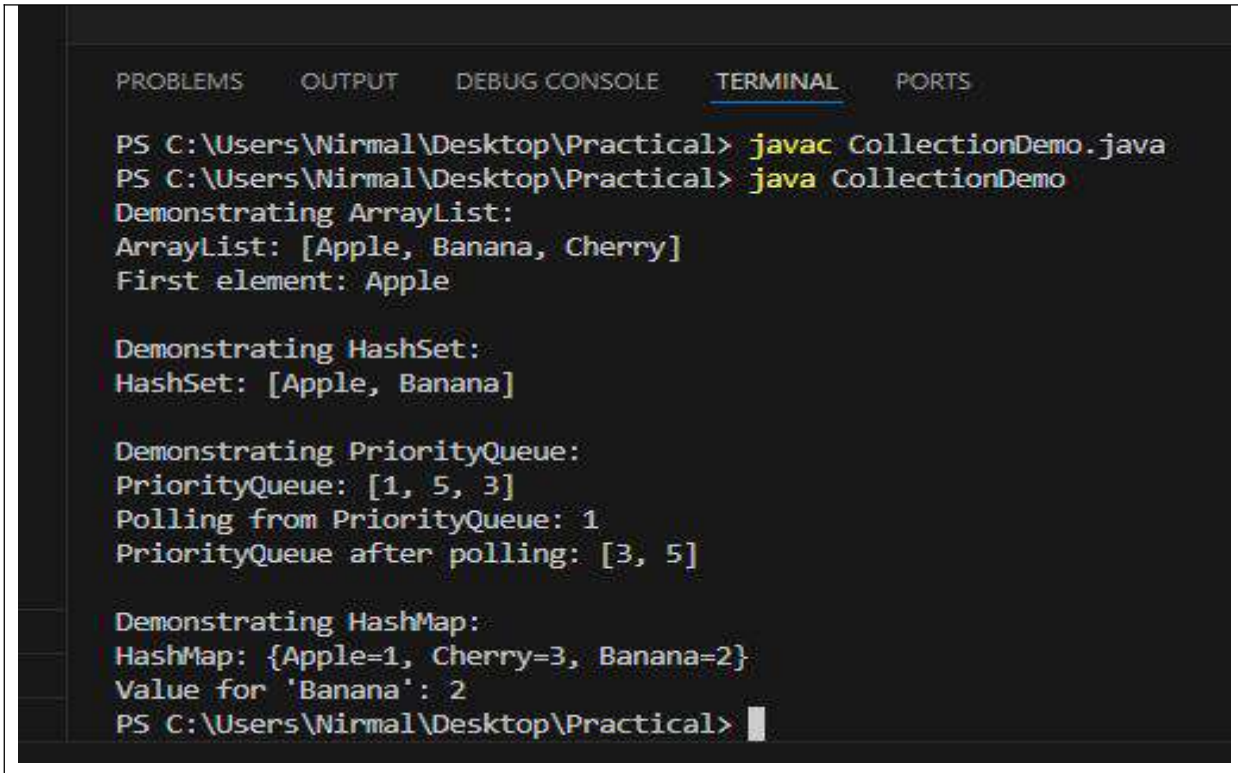
2. **Objective:** Implement programs that demonstrate the use of various collection classes at different levels of the hierarchy (e.g., Collection, List, Set, Queue, Map).
3. **Code and Procedure:**

CollectionDemo.java

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.PriorityQueue;
import java.util.HashMap;
public class CollectionDemo {
   public static void main(String[] args) {
      // 1. Demonstrating List using ArrayList
      System.out.println("Demonstrating ArrayList:");
      ArrayList<String> arrayList = new ArrayList<>();
      arrayList.add("Apple");
      arrayList.add("Banana");
      arrayList.add("Cherry");
      System.out.println("ArrayList: " + arrayList);
       // Accessing elements
      System.out.println("First element: " + arrayList.get(0));
       // 2. Demonstrating Set using HashSet
      System.out.println("\nDemonstrating HashSet:");
      HashSet<String> hashSet = new HashSet<>();
      hashSet.add("Apple");
      hashSet.add("Banana");
      hashSet.add("Apple"); // Duplicate, will not be added
      System.out.println("HashSet: " + hashSet);
       // 3. Demonstrating Queue using PriorityQueue
      System.out.println("\nDemonstrating PriorityQueue:");
      PriorityQueue<Integer> priorityQueue = new PriorityQueue<>();
      priorityQueue.add(5);
      priorityQueue.add(1);
      priorityQueue.add(3);
      System.out.println("PriorityQueue: " + priorityQueue);
      // Polling elements (removing the head of the queue)
      System.out.println("Polling from PriorityQueue: " + priorityQueue.poll());
      System.out.println("PriorityQueue after polling: " + priorityQueue);
       // 4. Demonstrating Map using HashMap
      System.out.println("\nDemonstrating HashMap:");
      HashMap<String, Integer> hashMap = new HashMap<>();
      hashMap.put("Apple", 1);
      hashMap.put("Banana", 2);
      hashMap.put("Cherry", 3);
      System.out.println("HashMap: " + hashMap);
      // Accessing value by key
      System.out.println("Value for 'Banana': " + hashMap.get("Banana"));
   }
```

}

4. **Output:**



Output of CollectionDemo.java file showing results different collection

5. **Outcome of the Practical:** Students will learn how to effectively utilize different collection classes such as ArrayList, HashSet, PriorityQueue, and HashMap to store, manipulate, and retrieve data. Students will develop the ability to choose the appropriate collection type based on specific requirements, such as the need for ordered elements, unique values, or key-value pairs.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**

- What is the Java Collection Framework, and why is it important?
- Can you explain the differences between List, Set, and Map interfaces in Java?
- What are the key features of an ArrayList, and how does it differ from an array?
- How does a HashSet handle duplicate elements, and what are its advantages?
- What is the purpose of a PriorityQueue, and how does it determine the order of its elements?
- How do you retrieve a value from a HashMap using a key, and what happens if the key does not exist?
- Can you explain the time complexity of common operations (add, remove, access) for ArrayList, HashSet, and HashMap?

- What are the differences between LinkedList and ArrayList in terms of performance and use cases?
- How can you iterate over the elements of a HashMap?
- In what scenarios would you prefer to use a TreeSet over a HashSet?

Practical No. 8

1. **Experiment Description:** The Iterator interface provides a standard way to traverse collections in Java. It allows for the removal of elements during iteration and provides a way to access elements sequentially without exposing the underlying structure of the collection. This practical will involve creating a program that uses an ArrayList and a HashSet, demonstrating how to iterate through both collections using the Iterator interface.

Program Structure:

List Implementation: Use ArrayList to demonstrate iteration.

Set Implementation: Use HashSet to demonstrate iteration.

ArrayList: An ArrayList is created and populated with fruit names. An Iterator is obtained from the ArrayList using the iterator() method. The while loop uses hasNext() to check if there are more elements to iterate over and next() to retrieve the next element.

HashSet: A HashSet is created and populated with animal names. Similarly, an Iterator is obtained from the HashSet, and the same iteration process is followed to print the animal names.

When the program is executed, it will display the elements of the ArrayList followed by the elements of the HashSet. This practical will help students understand how to use

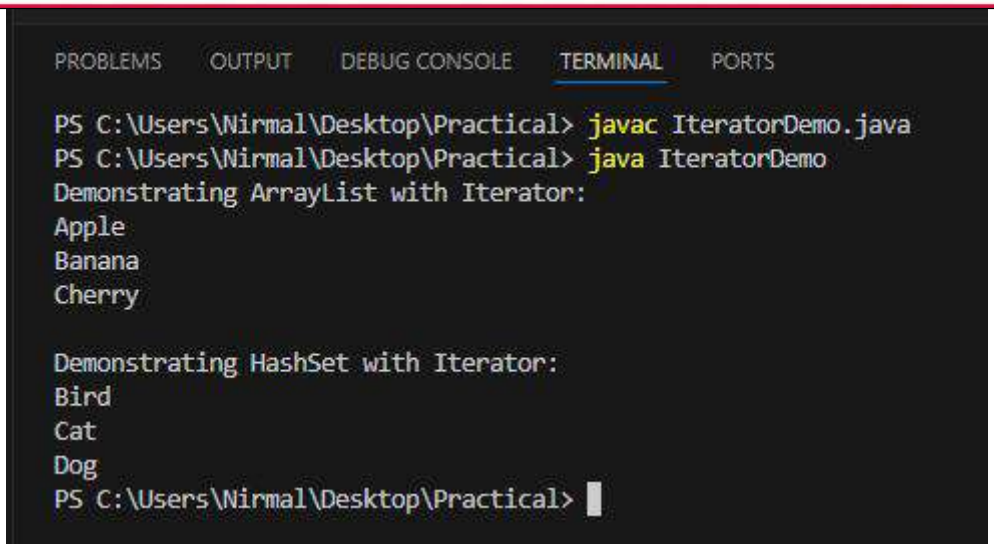
the Iterator interface to traverse collections in Java, highlighting the differences in iteration between a List and a Set.

2. **Objective:** Write a program that demonstrates the use of the Iterator interface to traverse a List and a Set. Use hasNext() and next() methods to iterate over collections like ArrayList and HashSet.
3. **Code and Procedure:**

IteratorDemo.java

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
public class IteratorDemo {
   public static void main(String[] args) {
      // 1. Demonstrating Iterator with ArrayList
      System.out.println("Demonstrating ArrayList with Iterator:");
      ArrayList<String> arrayList = new ArrayList<>();
      arrayList.add("Apple");
      arrayList.add("Banana");
      arrayList.add("Cherry");
      // Creating an iterator for the ArrayList
      Iterator<String> listIterator = arrayList.iterator();
      while (listIterator.hasNext()) {
         String fruit = listIterator.next();
         System.out.println(fruit);}
       // 2. Demonstrating Iterator with HashSet
      System.out.println("\nDemonstrating HashSet with Iterator:");
      HashSet<String> hashSet = new HashSet<>();
      hashSet.add("Dog");
      hashSet.add("Cat");
      hashSet.add("Bird");
      // Creating an iterator for the HashSet
      Iterator<String> setIterator = hashSet.iterator();
      while (setIterator.hasNext()) {
         String animal = setIterator.next();
         System.out.println(animal);
      }
   }
}
```

4. **Output:**



Output of IteratorDemo.Java file showing results for Iterator in ArrayList and HashSet

5. **Outcome of the Practical:** Students will learn how to effectively use the Iterator interface to traverse collections in Java. They will understand the importance of the hasNext() and next() methods in the iteration process and how to apply these methods to different collection types such as ArrayList and HashSet. This experience will enhance their ability to manipulate collections and improve their overall programming skills in Java.
6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is the purpose of the Iterator interface in Java?
   - How does the hasNext() method work in the context of iteration?
   - What is the difference between using an Iterator and a traditional for loop for traversing collections?
   - Can you explain how the next() method functions in the Iterator interface?
   - What happens if you call next() when there are no more elements to iterate over?
   - How can you remove an element from a collection while iterating using an Iterator?
   - What are the key differences between List and Set in terms of iteration?
   - Why is it important to use an Iterator instead of directly accessing elements in a collection?
   - Can you use an Iterator with a Map? If so, how?
   - What are some common pitfalls to avoid when using the Iterator interface?

Practical No. 9

1. **Experiment Description:** To test RESTful web services using Spring Boot, focus on writing unit and integration tests. This involves setting up a test environment, using tools and testing various aspects for simulating HTTP requests.

Procedure:

Tools to be required: VSCode IDE, Spring Boot Extension Pack, REST Client extension

After installing SpringBoot Extension pack the dashboard for Springboot will be available

Steps to follow

- Go to View ->Command Pallet – Type Spring Initialiser – select Create a Maven Project
- Select lowest version available eg 3.3.6 then select java
- Then name the group id org.restsample
- Name the Artifact id springtestexample then select jar and keep the version 17 or above available of java
- For Dependency -type web and select Spring Web Web
- Press Enter to continue (it will take you to the project path select your project and click on Generate into this folder)
- Now navigate to your folder in VSCode to open it

To check For correct JDK installation and working Ensure the file pom.xml for <properties>

<java.version>17</java.version>

</properties>

- Go to manage -> settings search with runtimes and click on Edit in settings.json

Provide the code inside "java.configuration.runtimes": [

{ "name":"JavaSE-17",

"path":"C:\\Program Files\\Java\\jdk-21",

"default": true}

- Save the file and close it.

Now have a look in the file SpringtestexampleApplication.java which have the path C:\Users\Nirmal\Desktop\Practical\gs-rest-service-main\springtestexample\src\main\java\org\restexample\springtestexample\SpringtestexampleApplication.java

- Now adding REST controller with two different methods

1. GET Request
2. POST Request

For this create a new file with SampleRestController.java name

- Now run the main class which is SpringtestexampleApplication.java. this will result a SPRING page in terminal.(output 1)
- open the local host and see the result for /api/greetings (get method)
- for testing these, under the project folder Springtestexample create a http file with name samplerequest.http (ensure REST Client extension is installed properly)
- Test the GET and POST method and get output accordingly

2. **Objective:** Test RESTful web services using Spring Boot.

3. **Code and Procedure:**

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.3.10</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>org.restexample</groupId>
    <artifactId>springtestexample</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springtestexample</name>
    <description>Demo project for Spring Boot</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
          </plugin>
        </plugins>
    </build>
</project>
```

Settings.json

```
{
   "debug.javascript.defaultRuntimeExecutable": {
      "pwa-node": "node"
   },
   "java.configuration.runtimes": [
            { "name":"JavaSE-17",
   "path":"C:\\Program Files\\Java\\jdk-21",
"default": true}
   ]
}
```

SpringtestexampleApplication.java

```
package org.restexample.springtestexample;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringtestexampleApplication {
   public static void main(String[] args) {
      SpringApplication.run(SpringtestexampleApplication.class, args);
   }
}
```

SampleRestController.java

```
package org.restexample.springtestexample;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@RestController
@RequestMapping("/api")
public class SampleRestController {
@GetMapping("/greetings")
public String getGreetings() {
   return "Hello, World";
```

```
}
@PostMapping("/echo")
public String echoMessage(@RequestBody String message) {
   return "You Sent:" + message;
}
}
```

samplerequest.http

For get

```
get http:/localhost:8080/api/greetings
```

For post

```
post http://localhost:8080/api/echo
Content-Type: application/json

{
   "message":"Testing RESTful web services using Springboot"
}
```

4. **Output:**

```
PS C:\Users\Nirmal\Desktop\Practical\gs-rest-service-main\springtestexample>  & 'C:\Program Files\J
a\Local\Temp\cp_dc64xz6vn5vjbfgb84rv66hqq.argfile' 'org.restexample.springtestexample.Springtestexa


  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

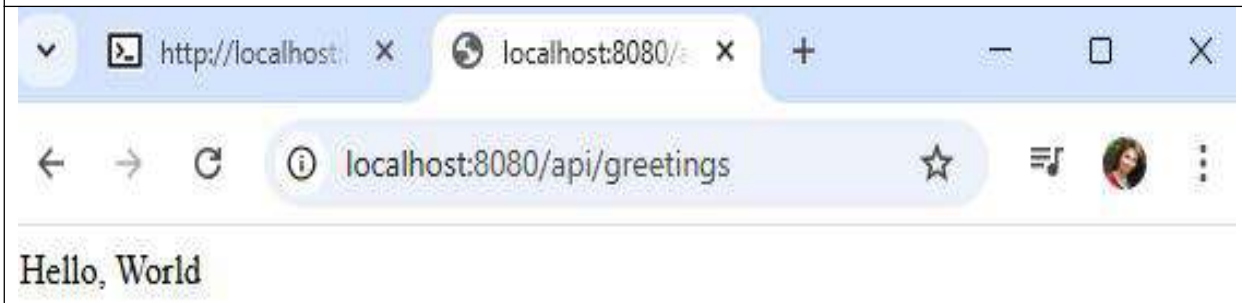
 :: Spring Boot ::                (v3.3.10)

2025-04-01T23:38:10.747+05:30  INFO 12656 --- [springtestexample] [           main] o.r.s.Springtes
ampleApplication using Java 21.0.6 with PID 12656 (C:\Users\Nirmal\Desktop\Practical\gs-rest-servic
y Nirmal in C:\Users\Nirmal\Desktop\Practical\gs-rest-service-main\springtestexample)
2025-04-01T23:38:10.750+05:30  INFO 12656 --- [springtestexample] [           main] o.r.s.Springtes
, falling back to 1 default profile: "default"
2025-04-01T23:38:12.096+05:30  INFO 12656 --- [springtestexample] [           main] o.s.b.w.embedde
th port 8080 (http)
ess running for 4.962)
2025-04-01T23:41:06.236+05:30  INFO 12656 --- [springtestexample] [nio-8080-exec-2] o.a.c.c.C.[Tomc
ispatcherServlet 'dispatcherServlet'
2025-04-01T23:41:06.236+05:30  INFO 12656 --- [springtestexample] [nio-8080-exec-2] o.s.web.servlet
'dispatcherServlet'
2025-04-01T23:41:06.246+05:30  INFO 12656 --- [springtestexample] [nio-8080-exec-2] o.s.web.servlet
ion in 10 ms
```

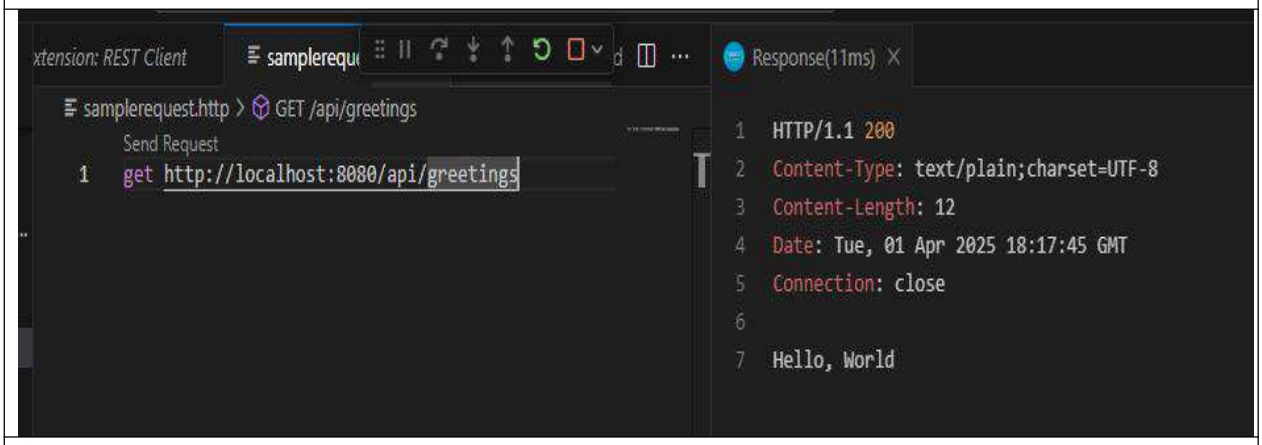
1.Output of SpringtestexampleApplication.java after creating SampleRestController.java



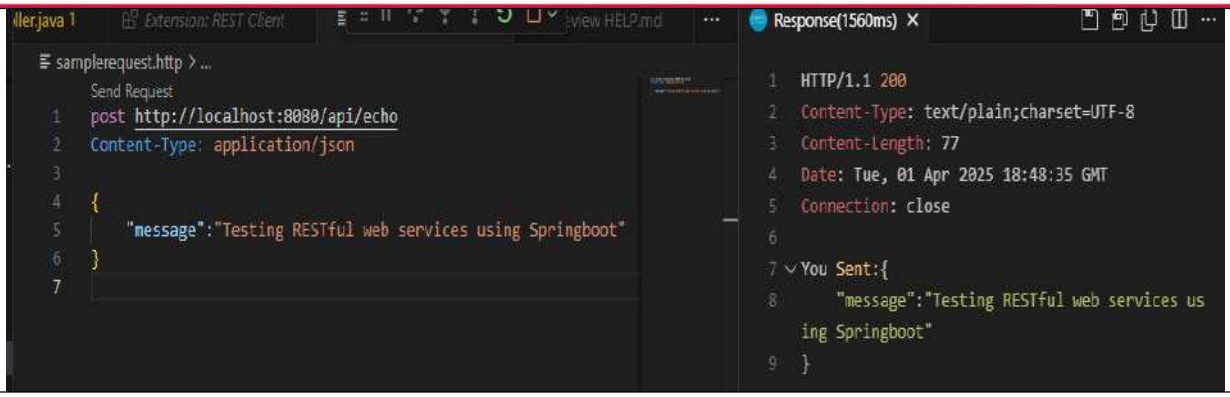
Output of default page of localhost



Output after giving path of greetings



Output of Samplerequest.http file for testing get method on clicking Send Request



Output of Samplerequest.http file for testing post method on clicking Send Request

5. **Outcome of the Practical:** Students will gain hands-on experience in writing unit and integration tests for RESTful services. They will also gain the knowledge of testing the GET and POST methods and how to handle http requests.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What does REST stand for, and what are its main principles?
   - Can you explain what a resource is in the context of RESTful web services?
   - What HTTP methods are commonly used in RESTful APIs, and what do they represent?
   - How does Spring Boot simplify the development of RESTful web services?
   - What is the purpose of the @RestController annotation in Spring Boot?
   - How do you handle exceptions in a Spring Boot RESTful application?
   - What is the role of the @RequestMapping annotation in Spring Boot?
   - How can you implement pagination in a Spring Boot REST API?
   - What is the difference between @GetMapping and @PostMapping in Spring Boot?
   - How do you secure a RESTful web service in Spring Boot?

Practical No. 10

1. **Experiment Description:** To demonstrate how to test a frontend web application that interacts with a Spring Boot backend. This will involve setting up a simple Spring Boot application, creating a frontend using a JavaScript framework (like React or Angular), and implementing tests to ensure that both the frontend and backend work correctly together.

Tools to be required : JavaScript and a frontend framework (React, Angular, or Vue.js). Development environment set up with Java, Maven/Gradle, Node.js, and npm/yarn.

Program Structure:

Spring Boot Backend:

- Create a RESTful API using Spring Boot.
- Implement basic CRUD operations.
- Write unit tests for the API endpoints.

Frontend Application:

- Set up a simple frontend application using React (or another framework).
- Create components that interact with the Spring Boot API.
- Write tests for the frontend components.

Integration Testing:

- Test the interaction between the frontend and backend.
- Use tools like Postman or Cypress for end-to-end testing.

Step-by-Step Implementation

Step 1: Set Up Spring Boot Backend

- Create a New Spring Boot Project:
- Use Spring Initializr (https://start.spring.io/) to create a new project with dependencies: Spring Web, Spring Data JPA, and H2 Database.
- Create a Model Class:User.java
- Create a Repository Interface: public interface UserRepository extends JpaRepository<User, Long>{}
- Create a REST Controller:UserController.java
- Write Unit Tests: UserControllerTest.java

Step 2: Set Up Frontend Application

- Create a New React Application: npx create-react-app frontend then cd frontend
- Install Axios for API Calls: npm install axios
- Create a User Component: UserComponent.js
- Write Tests for the User Component:UserTest.js

Step 3: Integration Testing

- Use Postman for Manual Testing: Test the API endpoints using Postman to ensure they return the expected results.
- Use Cypress for End-to-End Testing: Install Cypress: npm install cypress --save-dev
- Create a test file in cypress/integration to test the full flow of the application.

2. **Objective:** Test Frontend web application with Spring Boot

3. **Code and Procedure:**

User.java (Model Class)

```
@Entity
public class User {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
```

```
    private String name;
    private String email;     // Getters and Setters
}
```

UserController.java

```
@RestController
@RequestMapping("/api/users")
public class UserController {
    @Autowired
    private UserRepository userRepository;
    @GetMapping
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }
    @PostMapping
    public User createUser (@RequestBody User user) {
        return userRepository.save(user);
    }
}
```

UserControllerTest.java

```
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerTest {
    @Autowired
    private MockMvc mockMvc;
    @Test
    public void testGetAllUsers() throws Exception {
        mockMvc.perform(get("/api/users"))
            .andExpect(status().isOk());
    }
    @Test
    public void testCreateUser () throws Exception {
        String userJson = "{\"name\":\"John Doe\", \"email\":\"john@example.com\"}";
        mockMvc.perform(post("/api/users")
            .contentType(MediaType.APPLICATION_JSON)
            .content(userJson))
            .andExpect(status().isCreated());
    }
}
```

UserComponent.js

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
const UserList = () => {
    const [users, setUsers] = useState([]);
    useEffect(() => {
        axios.get('/api/users')
            .then(response => setUsers(response.data));
    }, []);
    return (
```

```
      <div>
        <h1>User List</h1>
        <ul>
          {users.map(user => (
            <li key={user.id}>{user.name} - {user.email}</li>
          ))}
        </ul>
      </div>
    );
};
export default UserList;
```

UserTest.js

```
import { render, screen } from '@testing-library/react';
import UserList from './User List';
import axios from 'axios';
jest.mock('axios');
test('renders user list', async () => {
  axios.get.mockResolvedValueOnce({ data: [{ id: 1, name: 'John Doe', email: 'john@example.com' }] });
  render(<User List />);
  const userElement = await screen.findByText(/John Doe/i);
  expect(userElement).toBeInTheDocument();
});
```

4. **Output: (only for reference)**

```
Tomcat started on port(s): 8080 (http) with context path ''
Started Application in X seconds
```

Output of Successful API Startup

```
[
    {
        "id": 1,
        "name": "John Doe",
        "email": "john@example.com"
    }
]
```

Output of json file after adding a user GET Request to /api/users

```
{
    "id": 1,
    "name": "John Doe",
    "email": "john@example.com"
}
```

Output of json file for response of POST Request to /api/users

```
User List
John Doe - john@example.com
```

Output of Frontend Userlist Display

```
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
```

Output of Unit Test for Spring Boot

```
PASS  src/UserList.test.js
  ✓ renders user list (X ms)
```

Output of Unit Test for React Components

```
PASS  src/UserList.test.js
  ✓ renders user list (X ms)
```

Output of Cypress End-to-End Tests

5. **Outcome of the Practical:** Students will gain hands-on experience in:
   - Setting up a Spring Boot application and creating RESTful APIs.
   - Building a frontend application using a JavaScript framework.
   - Writing unit and integration tests to validate the functionality of both the frontend and backend.
   - Understanding the importance of testing in software development.

6. **Assessment Criteria:**

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. **Viva Questions/Review Questions:**
   - What is the purpose of using Spring Boot for backend development?
   - How do you create a RESTful API in Spring Boot?
   - What is the role of the @RestController annotation?
   - How do you handle exceptions in a Spring Boot application?
   - What is the purpose of using Axios in a React application?
   - How do you test a React component using Jest?
   - What is the difference between unit testing and integration testing?
   - How can you test the interaction between the frontend and backend?
   - What tools can you use for end-to-end testing in web applications?
   - Why is it important to write tests for your application?

List of Value Added Programs

Practical No. 11

1. **Experiment Description:** The Producer-Consumer problem is a classic synchronization problem in concurrent programming. It involves two types of threads: producers, which generate data and place it into a shared buffer, and consumers, which take data from the buffer. Proper synchronization is required to ensure that the producer does not add data to a full buffer and that the consumer does not attempt to consume data from an empty buffer.

   Program Structure:

   Shared Buffer: A class that holds the data produced and consumed.

   Producer Class: A class that generates data and adds it to the buffer.

   Consumer Class: A class that retrieves data from the buffer.

   Main Class: A class to run the producer and consumer threads.

   SharedBuffer: This class contains a queue to hold the produced items and methods for producing and consuming items. The produce() method adds items to the buffer and uses wait() and notifyAll() to manage synchronization.

   Producer: This class implements Runnable and generates integers to be added to the shared buffer. It simulates production time with Thread.sleep().

   Consumer: This class also implements Runnable and retrieves items from the shared buffer, simulating consumption time.

   Main Class: The ProducerConsumerDemo class initializes the shared buffer and starts the producer and consumer threads.

   When the program is executed, the producer will generate integers and add them to the shared buffer, while the consumer will retrieve and print these integers. The wait-notify mechanism ensures that the producer waits when the buffer is full and the consumer waits when the buffer is empty, demonstrating proper synchronization.

2. **Objective:** Implement the consumer producer problem with the synchronized methods produce() and consume(), with proper wait-notify mechanisms.

3. **Code and Procedure:**

ProducerConsumerDemo.java

```
import java.util.LinkedList;
import java.util.Queue;
class SharedBuffer {
   private final Queue<Integer> buffer = new LinkedList<>();
   private final int capacity;

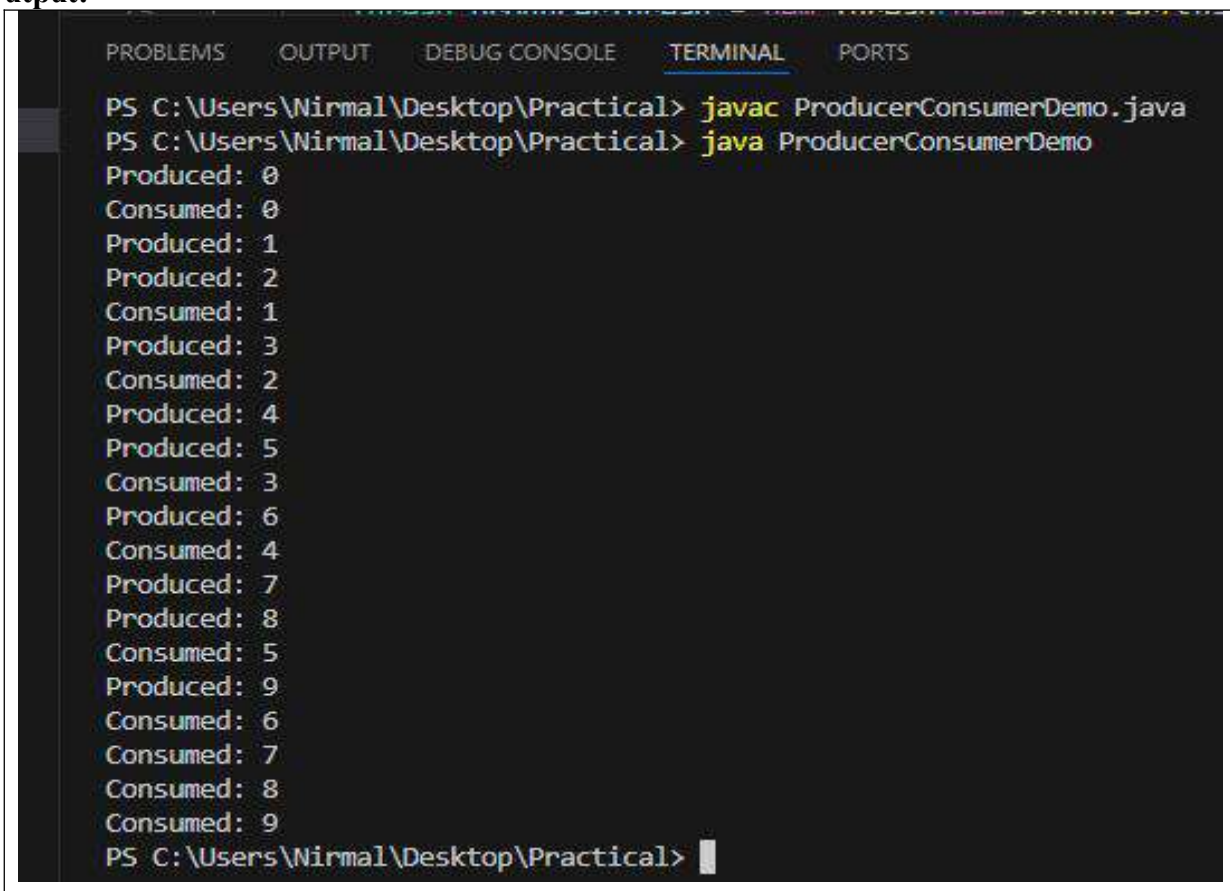
   public SharedBuffer(int capacity) {
      this.capacity = capacity;
   }
   public synchronized void produce(int value) throws InterruptedException {
      while (buffer.size() == capacity) {
         wait(); // Wait until there is space in the buffer
      }
      buffer.add(value);
      System.out.println("Produced: " + value);
      notifyAll(); // Notify consumers that an item has been produced
   }
   public synchronized int consume() throws InterruptedException {
      while (buffer.isEmpty()) {
```

```
            wait(); // Wait until there is an item to consume
        }
        int value = buffer.poll();
        System.out.println("Consumed: " + value);
        notifyAll(); // Notify producers that an item has been consumed
        return value;
    }
}
class Producer implements Runnable {
    private final SharedBuffer sharedBuffer;

    public Producer(SharedBuffer sharedBuffer) {
        this.sharedBuffer = sharedBuffer;
    }
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                sharedBuffer.produce(i);
                Thread.sleep(100); // Simulate time taken to produce
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
class Consumer implements Runnable {
    private final SharedBuffer sharedBuffer;
    public Consumer(SharedBuffer sharedBuffer) {
        this.sharedBuffer = sharedBuffer;
    }
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                sharedBuffer.consume();
                Thread.sleep(150); // Simulate time taken to consume
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
public class ProducerConsumerDemo {
    public static void main(String[] args) {
        SharedBuffer sharedBuffer = new SharedBuffer(5); // Buffer capacity of 5
        Thread producerThread = new Thread(new Producer(sharedBuffer));
        Thread consumerThread = new Thread(new Consumer(sharedBuffer));
        producerThread.start();
        consumerThread.start();
    }
```

}

4. Output:



Output of ProducerConsumerDemo.Java file showing use of produce and consume with proper synchronization

5. Outcome of the Practical: Students will learn how to implement the Producer-Consumer problem using synchronized methods in Java. They will understand the concepts of thread synchronization, the use of wait-notify mechanisms, and how to manage shared resources in a concurrent environment. This experience will enhance their understanding of multithreading and concurrency in Java.

6. Assessment Criteria:

Record	3 Marks
Execution	4 Marks
Viva	3 Marks

7. Viva Questions/Review Questions:

- What is the Producer-Consumer problem, and why is it important in concurrent programming?
- How does the synchronized keyword work in Java?
- Explain the purpose of the wait() and notifyAll() methods in the context of thread synchronization.
- What happens if a producer tries to produce an item when the buffer is full?
- How does the consumer know when it can consume an item from the buffer?
- What are the potential issues that can arise if proper synchronization is not implemented?
- Can you explain the difference between wait() and sleep() methods in Java?
- How would you modify the program to allow multiple producers and consumers?

- What is the role of the SharedBuffer class in this implementation?
- How can you test the functionality of the Producer-Consumer implementation?

Textbooks and References

Textbooks

Book Name	Author	Description
Head First Java	Kathy Sierra, Bert Bates	An engaging introduction to Java focusing on object-oriented programming concepts.
Effective Java	Joshua Bloch	A must-read for Java developers, offering best practices and design patterns.
Java: The Complete Reference	Herbert Schildt	A comprehensive guide covering both core and advanced Java topics.
Java Programming and Problem Solving	D. S. Malik	Focuses on problem-solving techniques using Java, suitable for beginners.
Java in a Nutshell	Benjamin J. Evans, David Flanagan	A concise reference for Java developers, covering essential features and libraries.
Java: A Beginner's Guide	Herbert Schildt	A beginner-friendly introduction to Java programming with practical examples.
Programming in Java	E. Balagurusamy	A widely used textbook in Indian universities, covering basic to advanced Java concepts.

Reference Books

Book Name	Author	Description
Foundational Java: Key Elements and Practical Programming	N/A	A comprehensive guide with exercises to build Java skills.
Java: A Complete Practical Solution	Swati Saxena	Ideal for beginners and professionals, focusing on dynamic web applications.
Introduction to Programming Using Java	N/A	A freely available PDF resource for foundational Java concepts.

Online Recourses Available

Online Resources for Java Programming

Resource Name	**Type**	**Description**	Link
Oracle Java Documentation	Official Documentation	The official documentation for Java, including tutorials, API references, and guides.	Oracle Java Docs
Codecademy Java Course	Online Course	An interactive platform offering a beginner-friendly Java course with hands-on exercises.	Codecademy
Coursera Java Programming and Software Engineering Fundamentals	Online Course	A series of courses covering Java programming and software engineering principles.	Coursera
Udacity Java Programming Basics	Online Course	A free course that introduces the basics of Java programming and object-oriented concepts.	Udacity
W3Schools Java Tutorial	Online Tutorial	A comprehensive tutorial covering Java basics, syntax, and examples.	W3Schools
GeeksforGeeks Java Programming	Online Articles & Tutorials	A vast collection of articles, tutorials, and coding challenges for Java programming.	GeeksforGeeks
Java Brains	YouTube Channel	A YouTube channel offering video tutorials on Java and related technologies.	Java Brains
freeCodeCamp Java Course	Online Course	A free comprehensive course on Java programming available on YouTube.	freeCodeCamp
SoloLearn Java Course	Mobile App & Online	An interactive platform and mobile app for learning Java through quizzes and coding challenges.	SoloLearn