Q1. Sort a list of students by roll number (ascending) using Comparable.

Program:

class Student implements Comparable<Student> {

int rollNo;

String name;

double marks;

public Student(int rollNo, String name, double marks) {

this.rollNo = rollNo;

this.name = name;

this.marks = marks;

}

public int compareTo(Student other) {

return this.rollNo - other.rollNo;

}

public String toString() {

return rollNo + " " + name + " " + marks;

}

}

Create a Student class with fields: rollNo, name, and marks. Implement the Comparable interface to sort students by their roll numbers.

Q2. Create a Product class and sort products by price using Comparable.

Implement Comparable<Product> and sort a list of products using Collections.sort().

Program:

class Product implements Comparable<Product> {

String name;

double price;

public Product(String name, double price) {

this.name = name;

this.price = price;

}

public int compareTo(Product other) {

return Double.compare(this.price, other.price);

}

public String toString() {

return name + " - ₹" + price;

}

}

Q3. Create an Employee class and sort by name using Comparable.

Use the compareTo() method to sort alphabetically by employee names.

Program:

class Employee implements Comparable<Employee> {

String name;

double salary;

public Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

public int compareTo(Employee other) {

return this.name.compareTo(other.name);

}

public String toString() {

return name + " - ₹" + salary;

}

}

Q4. Sort a list of Book objects by bookId in descending order using Comparable.

Hint: Override compareTo() to return the reverse order.

Program:

class Book implements Comparable<Book> {

int bookId;

String title;

public Book(int bookId, String title) {

this.bookId = bookId;

this.title = title;

}

public int compareTo(Book other) {

return other.bookId - this.bookId; // Descending

}

public String toString() {

return bookId + ": " + title;

}

}

Q5. Implement a program that sorts a list of custom objects using Comparable, and displays them before and after sorting.

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

// Custom class implementing the Comparable interface

class Student implements Comparable<Student> {

private int rollNumber;

private String name;

public Student(int rollNumber, String name) {

this.rollNumber = rollNumber;

this.name = name;

}

@Override

public String toString() {

return "Student{" +

"rollNumber=" + rollNumber +

", name='" + name + '\'' +

'}';

}

// Compares this object with the specified object for order.

// Returns a negative integer, zero, or a positive integer as this object is

// less than, equal to, or greater than the specified object.

@Override

public int compareTo(Student other) {

return this.rollNumber - other.rollNumber;

}

}

public class ComparableSortingExample {

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(101, "Alice"));

students.add(new Student(103, "Charlie"));

students.add(new Student(102, "Bob"));

System.out.println("Unsorted List: \n" + students);

// Sorting the list using the natural ordering defined by Comparable

Collections.sort(students);

System.out.println("\nSorted List (by roll number): \n" + students);

}

}

Q6. Sort a list of students by marks (descending) using Comparator.

Create a Comparator class or use a lambda expression to sort by marks.

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Student {

private String name;

private int marks;

public Student(String name, int marks) {

this.name = name;

this.marks = marks;

}

public int getMarks() {

return marks;

}

@Override

public String toString() {

return "Student{" +

"name='" + name + '\'' +

", marks=" + marks +

'}';

}

}

public class ComparatorSortingExample {

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student("Alice", 85));

students.add(new Student("Charlie", 92));

students.add(new Student("Bob", 78));

students.add(new Student("David", 92));

System.out.println("Unsorted List:\n" + students);

// Sorting by marks in descending order using a lambda expression

Comparator<Student> marksComparator = (s1, s2) -> Integer.compare(s2.getMarks(), s1.getMarks());

Collections.sort(students, marksComparator);

System.out.println("\nSorted List (by marks descending):\n" + students);

}

}

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Product {

private String name;

private double price;

public Product(String name, double price) {

this.name = name;

this.price = price;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

@Override

public String toString() {

return "Product{" +

"name='" + name + '\'' +

", price=" + price +

'}';

}

}

public class ProductSortingStrategies {

public static void main(String[] args) {

List<Product> products = new ArrayList<>();

products.add(new Product("Laptop", 1200.50));

products.add(new Product("Mouse", 25.00));

products.add(new Product("Keyboard", 75.99));

products.add(new Product("Monitor", 350.75));

System.out.println("Unsorted List:\n" + products);

// Sorting by price ascending

Comparator<Product> byPriceAsc = (p1, p2) -> Double.compare(p1.getPrice(), p2.getPrice());

Collections.sort(products, byPriceAsc);

System.out.println("\nSorted by Price (Ascending):\n" + products);

// Sorting by price descending

Comparator<Product> byPriceDesc = (p1, p2) -> Double.compare(p2.getPrice(), p1.getPrice());

Collections.sort(products, byPriceDesc);

System.out.println("\nSorted by Price (Descending):\n" + products);

// Sorting by name alphabetically

Comparator<Product> byNameAlpha = (p1, p2) -> p1.getName().compareTo(p2.getName());

Collections.sort(products, byNameAlpha);

System.out.println("\nSorted by Name (Alphabetical):\n" + products);

}

}

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

Program:

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Employee {

private String name;

private LocalDate joiningDate;

public Employee(String name, LocalDate joiningDate) {

this.name = name;

this.joiningDate = joiningDate;

}

public LocalDate getJoiningDate() {

return joiningDate;

}

@Override

public String toString() {

return "Employee{" +

"name='" + name + '\'' +

", joiningDate=" + joiningDate +

'}';

}

}

public class SortByJoiningDate {

public static void main(String[] args) {

List<Employee> employees = new ArrayList<>();

employees.add(new Employee("Alice", LocalDate.of(2020, 1, 15)));

employees.add(new Employee("Charlie", LocalDate.of(2018, 5, 10)));

employees.add(new Employee("Bob", LocalDate.of(2021, 11, 22)));

System.out.println("Unsorted List:\n" + employees);

// Sorting by joining date using a lambda expression

Comparator<Employee> byJoiningDate = (e1, e2) -> e1.getJoiningDate().compareTo(e2.getJoiningDate());

Collections.sort(employees, byJoiningDate);

System.out.println("\nSorted List (by joining date):\n" + employees);

}

}

Q9. Write a program that sorts a list of cities by population using Comparator.

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class City {

private String name;

private int population;

public City(String name, int population) {

this.name = name;

this.population = population;

}

public int getPopulation() {

return population;

}

@Override

public String toString() {

return "City{" +

"name='" + name + '\'' +

", population=" + population +

'}';

}

}

public class SortByPopulation {

public static void main(String[] args) {

List<City> cities = new ArrayList<>();

cities.add(new City("New York", 8467513));

cities.add(new City("Los Angeles", 3849297));

cities.add(new City("Chicago", 2699347));

cities.add(new City("Houston", 2304580));

System.out.println("Unsorted List:\n" + cities);

// Sorting by population using a lambda expression

Comparator<City> byPopulation = (c1, c2) -> Integer.compare(c1.getPopulation(), c2.getPopulation());

Collections.sort(cities, byPopulation);

System.out.println("\nSorted List (by population):\n" + cities);

}

}

Q10. Use an anonymous inner class to sort a list of strings by length.

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class SortByStringLength {

public static void main(String[] args) {

List<String> strings = new ArrayList<>();

strings.add("Apple");

strings.add("Banana");

strings.add("Grapes");

strings.add("Kiwi");

strings.add("Orange");

System.out.println("Unsorted List:\n" + strings);

// Sorting strings by length using an anonymous inner class

Collections.sort(strings, new Comparator<String>() {

@Override

public int compare(String s1, String s2) {

return Integer.compare(s1.length(), s2.length());

}

});

System.out.println("\nSorted List (by length):\n" + strings);

}

}

Q11. Create a program where:

Student implements Comparable to sort by name

Use Comparator to sort by marks

Demonstrate both sorting techniques in the same program.

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Student implements Comparable<Student> {

private String name;

private int marks;

public Student(String name, int marks) {

this.name = name;

this.marks = marks;

}

public String getName() {

return name;

}

public int getMarks() {

return marks;

}

@Override

public String toString() {

return "Student{" +

"name='" + name + '\'' +

", marks=" + marks +

'}';

}

// Natural sorting order by name

@Override

public int compareTo(Student other) {

return this.name.compareTo(other.name);

}

}

public class ComparableAndComparatorExample {

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student("Charlie", 85));

students.add(new Student("Alice", 92));

students.add(new Student("Bob", 78));

// Sorting using Comparable (natural order by name)

Collections.sort(students);

System.out.println("Sorted by Name (using Comparable):\n" + students);

// Sorting using Comparator (by marks)

Comparator<Student> marksComparator = (s1, s2) -> Integer.compare(s1.getMarks(), s2.getMarks());

Collections.sort(students, marksComparator);

System.out.println("\nSorted by Marks (using Comparator):\n" + students);

}

}

Q12. Sort a list of Book objects using both Comparable (by ID) and Comparator (by title, then author).

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Book implements Comparable<Book> {

private int id;

private String title;

private String author;

public Book(int id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

}

public int getId() {

return id;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

@Override

public String toString() {

return "Book{" +

"id=" + id +

", title='" + title + '\'' +

", author='" + author + '\'' +

'}';

}

// Natural sorting order by ID

@Override

public int compareTo(Book other) {

return Integer.compare(this.id, other.id);

}

}

public class BookSortingExample {

public static void main(String[] args) {

List<Book> books = new ArrayList<>();

books.add(new Book(102, "The Lord of the Rings", "J.R.R. Tolkien"));

books.add(new Book(101, "The Hobbit", "J.R.R. Tolkien"));

books.add(new Book(103, "Pride and Prejudice", "Jane Austen"));

System.out.println("Unsorted List:\n" + books);

// Sort by ID using Comparable (natural order)

Collections.sort(books);

System.out.println("\nSorted by ID (using Comparable):\n" + books);

// Sort by title, then author using Comparator

Comparator<Book> byTitleThenAuthor = (b1, b2) -> {

int titleComparison = b1.getTitle().compareTo(b2.getTitle());

if (titleComparison != 0) {

return titleComparison;

}

return b1.getAuthor().compareTo(b2.getAuthor());

};

Collections.sort(books, byTitleThenAuthor);

System.out.println("\nSorted by Title, then Author (using Comparator):\n" + books);

}

}

Q13. Write a menu-driven program to sort Employee objects by name, salary, or department using Comparator.

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

import java.util.Scanner;

class Employee {

private String name;

private double salary;

private String department;

public Employee(String name, double salary, String department) {

this.name = name;

this.salary = salary;

this.department = department;

}

public String getName() {

return name;

}

public double getSalary() {

return salary;

}

public String getDepartment() {

return department;

}

@Override

public String toString() {

return "Employee{" +

"name='" + name + '\'' +

", salary=" + salary +

", department='" + department + '\'' +

'}';

}

}

public class MenuDrivenSort {

public static void main(String[] args) {

List<Employee> employees = new ArrayList<>();

employees.add(new Employee("Alice", 75000.0, "HR"));

employees.add(new Employee("Charlie", 90000.0, "Engineering"));

employees.add(new Employee("Bob", 65000.0, "IT"));

employees.add(new Employee("David", 90000.0, "Engineering"));

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nSelect a sorting option:");

System.out.println("1. Sort by Name");

System.out.println("2. Sort by Salary");

System.out.println("3. Sort by Department");

System.out.println("4. Exit");

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

int choice = scanner.nextInt();

List<Employee> sortedEmployees = new ArrayList<>(employees);

switch (choice) {

case 1:

Collections.sort(sortedEmployees, Comparator.comparing(Employee::getName));

System.out.println("Sorted by Name:");

break;

case 2:

Collections.sort(sortedEmployees, Comparator.comparingDouble(Employee::getSalary));

System.out.println("Sorted by Salary:");

break;

case 3:

Collections.sort(sortedEmployees, Comparator.comparing(Employee::getDepartment));

System.out.println("Sorted by Department:");

break;

case 4:

System.out.println("Exiting program.");

scanner.close();

return;

default:

System.out.println("Invalid choice. Please try again.");

continue;

}

for (Employee emp : sortedEmployees) {

System.out.println(emp);

}

}

}

}

Q14. Use Comparator.comparing() with method references to sort objects in Java 8+.

Program:

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

@Override

public String toString() {

return "Person{" +

"name='" + name + '\'' +

", age=" + age +

'}';

}

}

public class Java8SortingExample {

public static void main(String[] args) {

List<Person> persons = new ArrayList<>();

persons.add(new Person("Charlie", 35));

persons.add(new Person("Alice", 25));

persons.add(new Person("Bob", 40));

System.out.println("Unsorted List:\n" + persons);

// Sorting by name using Comparator.comparing() and a method reference

persons.sort(Comparator.comparing(Person::getName));

System.out.println("\nSorted by Name:\n" + persons);

// Sorting by age descending using Comparator.comparing() and then reversed()

persons.sort(Comparator.comparing(Person::getAge).reversed());

System.out.println("\nSorted by Age (Descending):\n" + persons);

}

}

Q15. Use TreeSet with a custom comparator to sort a list of persons by age.

Program:

import java.util.Comparator;

import java.util.Set;

import java.util.TreeSet;

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

@Override

public String toString() {

return "Person{" +

"name='" + name + '\'' +

", age=" + age +

'}';

}

}

public class TreeSetWithComparator {

public static void main(String[] args) {

// Create a custom comparator to sort persons by age

Comparator<Person> byAgeComparator = (p1, p2) -> Integer.compare(p1.getAge(), p2.getAge());

// Create a TreeSet and pass the custom comparator to its constructor

Set<Person> persons = new TreeSet<>(byAgeComparator);

persons.add(new Person("Charlie", 35));

persons.add(new Person("Alice", 25));

persons.add(new Person("Bob", 40));

// The TreeSet automatically keeps the elements sorted by age

System.out.println("Sorted list of persons (by age):\n" + persons);

}

}

**Q1. Create and Write to a File**

Write a Java program to create a file named student.txt and write 5 lines of student names using FileWriter.

Program:

import java.io.FileWriter;

import java.io.IOException;

public class WriteToFile {

public static void main(String[] args) {

String fileName = "student.txt";

String[] students = {"Alice", "Bob", "Charlie", "David", "Eve"};

try (FileWriter writer = new FileWriter(fileName)) {

for (String student : students) {

writer.write(student + "\n");

}

System.out.println("Successfully wrote to the file: " + fileName);

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

**Q2. Read from a File**

Write a program to read the contents of student.txt and display them line by line using BufferedReader.

Program:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class ReadFromFile {

public static void main(String[] args) {

String fileName = "student.txt";

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

String line;

System.out.println("Contents of the file:");

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

System.out.println("An error occurred while reading the file.");

e.printStackTrace();

}

}

}

**Q3. Append Data to a File**

Write a Java program to append a new student name to the existing student.txt file without overwriting existing data.

Program:

import java.io.FileWriter;

import java.io.IOException;

public class AppendToFile {

public static void main(String[] args) {

String fileName = "student.txt";

String newStudent = "Frank";

try (FileWriter writer = new FileWriter(fileName, true)) { // `true` for append mode

writer.write("\n" + newStudent);

System.out.println("Successfully appended to the file: " + fileName);

} catch (IOException e) {

System.out.println("An error occurred while appending to the file.");

e.printStackTrace();

}

}

}

**Q4. Count Words and Lines**

Write a program to count the number of words and lines in a given text file notes.txt.

Program:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class CountWordsAndLines {

public static void main(String[] args) {

String fileName = "notes.txt";

int lineCount = 0;

int wordCount = 0;

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

String line;

while ((line = reader.readLine()) != null) {

lineCount++;

String[] words = line.split("\\s+");

wordCount += words.length;

}

System.out.println("File: " + fileName);

System.out.println("Number of lines: " + lineCount);

System.out.println("Number of words: " + wordCount);

} catch (IOException e) {

System.out.println("An error occurred while reading the file.");

e.printStackTrace();

}

}

}

**Q5. Copy Contents from One File to Another**

Write a program to read from source.txt and write the same content into destination.txt.

Program:

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class CopyFile {

public static void main(String[] args) {

String sourceFile = "source.txt";

String destinationFile = "destination.txt";

try (BufferedReader reader = new BufferedReader(new FileReader(sourceFile));

BufferedWriter writer = new BufferedWriter(new FileWriter(destinationFile))) {

String line;

while ((line = reader.readLine()) != null) {

writer.write(line);

writer.newLine(); // Writes a line separator

}

System.out.println("Content successfully copied from " + sourceFile + " to " + destinationFile);

} catch (IOException e) {

System.out.println("An error occurred during file copy.");

e.printStackTrace();

}

}

}

**Q6. Check if a File Exists and Display Properties**

Create a program to check if report.txt exists. If it does, display its:

* Absolute path
* File name
* Writable (true/false)
* Readable (true/false)
* File size in bytes

Program:

import java.io.File;

public class FileProperties {

public static void main(String[] args) {

String fileName = "report.txt";

File file = new File(fileName);

if (file.exists()) {

System.out.println("File found: " + fileName);

System.out.println("Absolute Path: " + file.getAbsolutePath());

System.out.println("File Name: " + file.getName());

System.out.println("Writable: " + file.canWrite());

System.out.println("Readable: " + file.canRead());

System.out.println("File Size in Bytes: " + file.length());

} else {

System.out.println("File not found: " + fileName);

}

}

}

**Q7. Create a File and Accept User Input**

Accept input from the user (using Scanner) and write the input to a file named userinput.txt.

Program:

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

public class UserInputToFile {

public static void main(String[] args) {

String fileName = "userinput.txt";

try (Scanner scanner = new Scanner(System.in);

FileWriter writer = new FileWriter(fileName)) {

System.out.println("Enter text to write to the file. Type 'exit' to stop.");

String line;

while (true) {

line = scanner.nextLine();

if (line.equalsIgnoreCase("exit")) {

break;

}

writer.write(line + "\n");

}

System.out.println("Successfully wrote user input to " + fileName);

} catch (IOException e) {

System.out.println("An error occurred while writing to the file.");

e.printStackTrace();

}

}

}

**Q8. Reverse File Content**

Write a program to read a file data.txt and create another file reversed.txt containing the lines in reverse order.

Program:

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class ReverseFileContent {

public static void main(String[] args) {

String sourceFile = "data.txt";

String destinationFile = "reversed.txt";

List<String> lines = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader(sourceFile))) {

String line;

while ((line = reader.readLine()) != null) {

lines.add(line);

}

} catch (IOException e) {

System.out.println("An error occurred while reading the source file.");

e.printStackTrace();

return;

}

Collections.reverse(lines);

try (BufferedWriter writer = new BufferedWriter(new FileWriter(destinationFile))) {

for (String line : lines) {

writer.write(line);

writer.newLine();

}

System.out.println("Content successfully reversed and written to " + destinationFile);

} catch (IOException e) {

System.out.println("An error occurred while writing to the destination file.");

e.printStackTrace();

}

}

}

**Q9. Store Objects in a File using Serialization**

Create a Student class with id, name, and marks. Serialize one object and save it in a file named student.ser.

Program:

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.io.Serializable;

class Student implements Serializable {

private int id;

private String name;

private double marks;

public Student(int id, String name, double marks) {

this.id = id;

this.name = name;

this.marks = marks;

}

@Override

public String toString() {

return "Student{" +

"id=" + id +

", name='" + name + '\'' +

", marks=" + marks +

'}';

}

}

public class SerializeObject {

public static void main(String[] args) {

Student student = new Student(1, "Alice", 95.5);

String fileName = "student.ser";

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(fileName))) {

oos.writeObject(student);

System.out.println("Student object successfully serialized and saved to " + fileName);

} catch (IOException e) {

System.out.println("An error occurred during serialization.");

e.printStackTrace();

}

}

}

**Q10. Read Serialized Object from File**

Deserialize the student.ser file and display the object's content on the console.

Program:

import java.io.FileInputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

public class DeserializeObject {

public static void main(String[] args) {

Student student = null;

String fileName = "student.ser";

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(fileName))) {

student = (Student) ois.readObject();

System.out.println("Student object successfully deserialized from " + fileName);

System.out.println("Object Content: " + student);

} catch (IOException | ClassNotFoundException e) {

System.out.println("An error occurred during deserialization.");

e.printStackTrace();

}

}

}

**Q11. Print All Files in a Directory**

Write a program to list all files (not directories) inside a folder path given by the user.

Program:

import java.io.File;

import java.util.Scanner;

public class ListDirectoryFiles {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a directory path: ");

String directoryPath = scanner.nextLine();

scanner.close();

File directory = new File(directoryPath);

if (!directory.exists() || !directory.isDirectory()) {

System.out.println("Invalid directory path or directory does not exist.");

return;

}

File[] files = directory.listFiles();

if (files == null || files.length == 0) {

System.out.println("No files found in the directory.");

} else {

System.out.println("Files in " + directoryPath + ":");

for (File file : files) {

if (file.isFile()) {

System.out.println(file.getName());

}

}

}

}

}

**Q12. Delete a File**

Write a program to delete a file (given by file name) if it exists.

Program:

import java.io.File;

import java.util.Scanner;

public class DeleteFile {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the name of the file to delete: ");

String fileName = scanner.nextLine();

scanner.close();

File fileToDelete = new File(fileName);

if (fileToDelete.exists()) {

if (fileToDelete.delete()) {

System.out.println("Successfully deleted " + fileName);

} else {

System.out.println("Failed to delete " + fileName + ". Check permissions.");

}

} else {

System.out.println("File not found: " + fileName);

}

}

}

**Q13. Word Search in a File**

Ask the user to enter a word and check whether it exists in the file notes.txt.

Program:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Scanner;

public class WordSearch {

public static void main(String[] args) {

String fileName = "notes.txt";

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the word to search for: ");

String searchWord = scanner.nextLine();

scanner.close();

boolean found = false;

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {

String line;

while ((line = reader.readLine()) != null) {

if (line.contains(searchWord)) {

found = true;

break;

}

}

} catch (IOException e) {

System.out.println("An error occurred while reading the file.");

e.printStackTrace();

return;

}

if (found) {

System.out.println("The word '" + searchWord + "' was found in the file.");

} else {

System.out.println("The word '" + searchWord + "' was not found in the file.");

}

}

}

**Q14. Replace a Word in a File**

Read content from story.txt, replace all occurrences of the word "Java" with "Python", and write the updated content to updated\_story.txt

Program:

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class ReplaceWordInFile {

public static void main(String[] args) {

String sourceFile = "story.txt";

String destinationFile = "updated\_story.txt";

try (BufferedReader reader = new BufferedReader(new FileReader(sourceFile));

BufferedWriter writer = new BufferedWriter(new FileWriter(destinationFile))) {

String line;

while ((line = reader.readLine()) != null) {

String updatedLine = line.replaceAll("Java", "Python");

writer.write(updatedLine);

writer.newLine();

}

System.out.println("Successfully replaced 'Java' with 'Python' and saved to " + destinationFile);

} catch (IOException e) {

System.out.println("An error occurred during file operation.");

e.printStackTrace();

}

}

}