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

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

Code:

import java.util.\*;

class Student implements Comparable<Student> {

int rollNo;

String name;

double marks;

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

this.rollNo = rollNo;

this.name = name;

this.marks = marks;

}

@Override

public int compareTo(Student s) {

return this.rollNo - s.rollNo; // ascending order

}

public String toString() {

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

}

}

public class Main {

public static void main(String[] args) {

ArrayList<Student> list = new ArrayList<>();

list.add(new Student(102, "Alice", 88.5));

list.add(new Student(101, "Bob", 92.0));

list.add(new Student(105, "Charlie", 75.0));

list.add(new Student(103, "David", 80.0));

System.out.println("Before Sorting:");

for (Student s : list) {

System.out.println(s);

}

Collections.sort(list);

System.out.println("\nAfter Sorting by Roll No (Ascending):");

for (Student s : list) {

System.out.println(s);

}

}

}

Output:

Before Sorting:

102 Alice 88.5

101 Bob 92.0

105 Charlie 75.0

103 David 80.0

After Sorting by Roll No (Ascending):

101 Bob 92.0

102 Alice 88.5

103 David 80.0

105 Charlie 75.0

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

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

Code:

import java.util.\*;

class Product implements Comparable<Product> {

int id;

String name;

double price;

Product(int id, String name, double price) {

this.id = id;

this.name = name;

this.price = price;

}

@Override

public int compareTo(Product p) {

// Ascending order by price

if (this.price > p.price)

return 1;

else if (this.price < p.price)

return -1;

else

return 0;

}

public String toString() {

return id + " " + name + " " + price;

}

}

public class Main {

public static void main(String[] args) {

ArrayList<Product> list = new ArrayList<>();

list.add(new Product(101, "Laptop", 75000));

list.add(new Product(102, "Phone", 25000));

list.add(new Product(103, "Tablet", 45000));

list.add(new Product(104, "Headphones", 2000));

System.out.println("Before Sorting:");

for (Product p : list) {

System.out.println(p);

}

Collections.sort(list);

System.out.println("\nAfter Sorting by Price (Ascending):");

for (Product p : list) {

System.out.println(p);

}

}

}

Output:

Before Sorting:

101 Laptop 75000.0

102 Phone 25000.0

103 Tablet 45000.0

104 Headphones 2000.0

After Sorting by Price (Ascending):

104 Headphones 2000.0

102 Phone 25000.0

103 Tablet 45000.0

101 Laptop 75000.0

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

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

Code:

import java.util.\*;

class Employee implements Comparable<Employee> {

int id;

String name;

double salary;

Employee(int id, String name, double salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

@Override

public int compareTo(Employee e) {

// Alphabetical order by name

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

}

public String toString() {

return id + " " + name + " " + salary;

}

}

public class Main {

public static void main(String[] args) {

ArrayList<Employee> list = new ArrayList<>();

list.add(new Employee(101, "John", 55000));

list.add(new Employee(102, "Alice", 65000));

list.add(new Employee(103, "Bob", 60000));

list.add(new Employee(104, "David", 70000));

System.out.println("Before Sorting:");

for (Employee e : list) {

System.out.println(e);

}

Collections.sort(list);

System.out.println("\nAfter Sorting by Name (Alphabetical):");

for (Employee e : list) {

System.out.println(e);

}

}

}

Output:

Before Sorting:

101 John 55000.0

102 Alice 65000.0

103 Bob 60000.0

104 David 70000.0

After Sorting by Name (Alphabetical):

102 Alice 65000.0

103 Bob 60000.0

104 David 70000.0

101 John 55000.0

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

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

Code:

import java.util.\*;

class Book implements Comparable<Book> {

int bookId;

String title;

String author;

Book(int bookId, String title, String author) {

this.bookId = bookId;

this.title = title;

this.author = author;

}

@Override

public int compareTo(Book b) {

// Descending order by bookId

return b.bookId - this.bookId;

}

public String toString() {

return bookId + " " + title + " " + author;

}

}

public class Main {

public static void main(String[] args) {

ArrayList<Book> list = new ArrayList<>();

list.add(new Book(101, "Java Basics", "James Gosling"));

list.add(new Book(105, "Data Structures", "Mark Allen"));

list.add(new Book(103, "Algorithms", "Robert Sedgewick"));

list.add(new Book(110, "Operating Systems", "Abraham Silberschatz"));

System.out.println("Before Sorting:");

for (Book b : list) {

System.out.println(b);

}

Collections.sort(list);

System.out.println("\nAfter Sorting by Book ID (Descending):");

for (Book b : list) {

System.out.println(b);

}

}

}

Output:

Before Sorting:

101 Java Basics James Gosling

105 Data Structures Mark Allen

103 Algorithms Robert Sedgewick

110 Operating Systems Abraham Silberschatz

After Sorting by Book ID (Descending):

110 Operating Systems Abraham Silberschatz

105 Data Structures Mark Allen

103 Algorithms Robert Sedgewick

101 Java Basics James Gosling

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

Code:

import java.util.\*;

class Item implements Comparable<Item> {

int id;

String name;

double price;

Item(int id, String name, double price) {

this.id = id;

this.name = name;

this.price = price;

}

@Override

public int compareTo(Item i) {

// Ascending order by price

if (this.price > i.price)

return 1;

else if (this.price < i.price)

return -1;

else

return 0;

}

public String toString() {

return id + " " + name + " " + price;

}

}

public class Main {

public static void main(String[] args) {

ArrayList<Item> list = new ArrayList<>();

list.add(new Item(101, "Pen", 10.5));

list.add(new Item(102, "Book", 50.0));

list.add(new Item(103, "Pencil", 5.0));

list.add(new Item(104, "Notebook", 30.0));

System.out.println("Before Sorting:");

for (Item i : list) {

System.out.println(i);

}

Collections.sort(list);

System.out.println("\nAfter Sorting by Price (Ascending):");

for (Item i : list) {

System.out.println(i);

}

}

}

Output:

Before Sorting:

101 Pen 10.5

102 Book 50.0

103 Pencil 5.0

104 Notebook 30.0

After Sorting by Price (Ascending):

103 Pencil 5.0

101 Pen 10.5

104 Notebook 30.0

102 Book 50.0

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

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

Code:

import java.util.\*;

class Student {

int rollNo;

String name;

double marks;

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

this.rollNo = rollNo;

this.name = name;

this.marks = marks;

}

public String toString() {

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

}

}

// Comparator class for sorting by marks (descending)

class MarksDescendingComparator implements Comparator<Student> {

@Override

public int compare(Student s1, Student s2) {

// Descending order

if (s1.marks < s2.marks)

return 1;

else if (s1.marks > s2.marks)

return -1;

else

return 0;

}

}

public class Main {

public static void main(String[] args) {

ArrayList<Student> list = new ArrayList<>();

list.add(new Student(101, "Alice", 88.5));

list.add(new Student(102, "Bob", 92.0));

list.add(new Student(103, "Charlie", 75.0));

list.add(new Student(104, "David", 80.0));

System.out.println("Before Sorting:");

for (Student s : list) {

System.out.println(s);

}

// Using Comparator class

Collections.sort(list, new MarksDescendingComparator());

System.out.println("\nAfter Sorting by Marks (Descending) [Using Comparator Class]:");

for (Student s : list) {

System.out.println(s);

}

// Using Lambda Expression

Collections.sort(list, (s1, s2) -> Double.compare(s2.marks, s1.marks));

System.out.println("\nAfter Sorting by Marks (Descending) [Using Lambda]:");

for (Student s : list) {

System.out.println(s);

}

}

}

output:

Before Sorting:

101 Alice 88.5

102 Bob 92.0

103 Charlie 75.0

104 David 80.0

After Sorting by Marks (Descending) [Using Comparator Class]:

102 Bob 92.0

101 Alice 88.5

104 David 80.0

103 Charlie 75.0

After Sorting by Marks (Descending) [Using Lambda]:

102 Bob 92.0

101 Alice 88.5

104 David 80.0

103 Charlie 75.0

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

Code:

import java.util.\*;

class Product {

int id;

String name;

double price;

Product(int id, String name, double price) {

this.id = id;

this.name = name;

this.price = price;

}

public String toString() {

return id + " " + name + " " + price;

}

}

// Comparator for Price Ascending

class PriceAscendingComparator implements Comparator<Product> {

@Override

public int compare(Product p1, Product p2) {

return Double.compare(p1.price, p2.price);

}

}

// Comparator for Price Descending

class PriceDescendingComparator implements Comparator<Product> {

@Override

public int compare(Product p1, Product p2) {

return Double.compare(p2.price, p1.price);

}

}

// Comparator for Name Alphabetical

class NameComparator implements Comparator<Product> {

@Override

public int compare(Product p1, Product p2) {

return p1.name.compareTo(p2.name);

}

}

public class Main {

public static void main(String[] args) {

ArrayList<Product> list = new ArrayList<>();

list.add(new Product(101, "Laptop", 75000));

list.add(new Product(102, "Phone", 25000));

list.add(new Product(103, "Tablet", 45000));

list.add(new Product(104, "Headphones", 2000));

System.out.println("Original List:");

for (Product p : list) {

System.out.println(p);

}

// Price Ascending

Collections.sort(list, new PriceAscendingComparator());

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

for (Product p : list) {

System.out.println(p);

}

// Price Descending

Collections.sort(list, new PriceDescendingComparator());

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

for (Product p : list) {

System.out.println(p);

}

// Name Alphabetically

Collections.sort(list, new NameComparator());

System.out.println("\nSorted by Name (Alphabetically):");

for (Product p : list) {

System.out.println(p);

}

}

}

Output:

Original List:

101 Laptop 75000.0

102 Phone 25000.0

103 Tablet 45000.0

104 Headphones 2000.0

Sorted by Price (Ascending):

104 Headphones 2000.0

102 Phone 25000.0

103 Tablet 45000.0

101 Laptop 75000.0

Sorted by Price (Descending):

101 Laptop 75000.0

103 Tablet 45000.0

102 Phone 25000.0

104 Headphones 2000.0

Sorted by Name (Alphabetically):

104 Headphones 2000.0

101 Laptop 75000.0

102 Phone 25000.0

103 Tablet 45000.0

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

Code:

import java.util.\*;

import java.time.LocalDate;

class Employee {

int id;

String name;

LocalDate joiningDate;

Employee(int id, String name, LocalDate joiningDate) {

this.id = id;

this.name = name;

this.joiningDate = joiningDate;

}

public String toString() {

return id + " " + name + " " + joiningDate;

}

}

public class Main {

public static void main(String[] args) {

ArrayList<Employee> list = new ArrayList<>();

list.add(new Employee(101, "Alice", LocalDate.of(2022, 5, 10)));

list.add(new Employee(102, "Bob", LocalDate.of(2020, 3, 15)));

list.add(new Employee(103, "Charlie", LocalDate.of(2023, 1, 5)));

list.add(new Employee(104, "David", LocalDate.of(2021, 8, 20)));

System.out.println("Before Sorting:");

for (Employee e : list) {

System.out.println(e);

}

// Sort by joining date (earliest first)

Collections.sort(list, Comparator.comparing(emp -> emp.joiningDate));

System.out.println("\nAfter Sorting by Joining Date (Earliest First):");

for (Employee e : list) {

System.out.println(e);

}

// Sort by joining date (latest first)

Collections.sort(list, Comparator.comparing((Employee emp) -> emp.joiningDate).reversed());

System.out.println("\nAfter Sorting by Joining Date (Latest First):");

for (Employee e : list) {

System.out.println(e);

}

}

}

Output:

Before Sorting:

101 Alice 2022-05-10

102 Bob 2020-03-15

103 Charlie 2023-01-05

104 David 2021-08-20

After Sorting by Joining Date (Earliest First):

102 Bob 2020-03-15

104 David 2021-08-20

101 Alice 2022-05-10

103 Charlie 2023-01-05

After Sorting by Joining Date (Latest First):

103 Charlie 2023-01-05

101 Alice 2022-05-10

104 David 2021-08-20

102 Bob 2020-03-15

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

Code:

import java.util.\*;

// City class

class City {

String name;

int population;

City(String name, int population) {

this.name = name;

this.population = population;

}

@Override

public String toString() {

return name + " - Population: " + population;

}

}

public class SortCitiesByPopulation {

public static void main(String[] args) {

// Create list of cities

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

cities.add(new City("Mumbai", 20411000));

cities.add(new City("Delhi", 16787941));

cities.add(new City("Bangalore", 8443675));

cities.add(new City("Hyderabad", 6809970));

// Sort using Comparator (population ascending)

Collections.sort(cities, new Comparator<City>() {

@Override

public int compare(City c1, City c2) {

return Integer.compare(c1.population, c2.population);

}

});

System.out.println("Cities sorted by population (ascending):");

for (City c : cities) {

System.out.println(c);

}

// Sort using lambda (population descending)

cities.sort((c1, c2) -> Integer.compare(c2.population, c1.population));

System.out.println("\nCities sorted by population (descending):");

cities.forEach(System.out::println);

}

}

Output:

Cities sorted by population (ascending):

Hyderabad - Population: 6809970

Bangalore - Population: 8443675

Delhi - Population: 16787941

Mumbai - Population: 20411000

Cities sorted by population (descending):

Mumbai - Population: 20411000

Delhi - Population: 16787941

Bangalore - Population: 8443675

Hyderabad - Population: 6809970

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

Code:

import java.util.\*;

public class SortStringsByLength {

public static void main(String[] args) {

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

names.add("Bharadwaj");

names.add("Raj");

names.add("Mohitha");

names.add("Anu");

// Sort using an anonymous inner class

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

@Override

public int compare(String s1, String s2) {

return Integer.compare(s1.length(), s2.length()); // ascending by length

}

});

System.out.println("Strings sorted by length (ascending):");

for (String name : names) {

System.out.println(name);

}

}

}

Output:

Strings sorted by length (ascending):

Raj

Anu

Mohitha

Bharadwaj

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.

Code:

import java.util.\*;

// Student class implementing Comparable

class Student implements Comparable<Student> {

String name;

int marks;

Student(String name, int marks) {

this.name = name;

this.marks = marks;

}

// Sort by name (natural ordering)

@Override

public int compareTo(Student other) {

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

}

@Override

public String toString() {

return name + " - Marks: " + marks;

}

}

public class StudentSortingDemo {

public static void main(String[] args) {

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

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

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

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

students.add(new Student("Raj", 88));

// Sort by name using Comparable

Collections.sort(students);

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

for (Student s : students) {

System.out.println(s);

}

// Sort by marks using Comparator

Collections.sort(students, new Comparator<Student>() {

@Override

public int compare(Student s1, Student s2) {

return Integer.compare(s1.marks, s2.marks); // ascending order

}

});

System.out.println("\nSorted by Marks (Comparator):");

for (Student s : students) {

System.out.println(s);

}

}

}

Output:

Sorted by Name (Comparable):

Anu - Marks: 92

Bharadwaj - Marks: 85

Mohitha - Marks: 78

Raj - Marks: 88

Sorted by Marks (Comparator):

Mohitha - Marks: 78

Bharadwaj - Marks: 85

Raj - Marks: 88

Anu - Marks: 92

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

Code:

import java.util.\*;

// Book class implementing Comparable (sort by ID)

class Book implements Comparable<Book> {

int id;

String title;

String author;

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

this.id = id;

this.title = title;

this.author = author;

}

// Natural ordering: sort by ID

@Override

public int compareTo(Book other) {

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

}

@Override

public String toString() {

return "ID: " + id + ", Title: " + title + ", Author: " + author;

}

}

public class BookSortingDemo {

public static void main(String[] args) {

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

books.add(new Book(3, "Java Programming", "James Gosling"));

books.add(new Book(1, "C Programming", "Dennis Ritchie"));

books.add(new Book(4, "Python Basics", "Guido van Rossum"));

books.add(new Book(2, "Java Programming", "Herbert Schildt"));

// Sort by ID using Comparable

Collections.sort(books);

System.out.println("Sorted by ID (Comparable):");

for (Book b : books) {

System.out.println(b);

}

// Sort by title, then author using Comparator

Collections.sort(books, new Comparator<Book>() {

@Override

public int compare(Book b1, Book b2) {

int titleCompare = b1.title.compareTo(b2.title);

if (titleCompare == 0) {

return b1.author.compareTo(b2.author); // secondary sort

}

return titleCompare;

}

});

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

for (Book b : books) {

System.out.println(b);

}

}

}

Code:

Sorted by ID (Comparable):

ID: 1, Title: C Programming, Author: Dennis Ritchie

ID: 2, Title: Java Programming, Author: Herbert Schildt

ID: 3, Title: Java Programming, Author: James Gosling

ID: 4, Title: Python Basics, Author: Guido van Rossum

Sorted by Title, then Author (Comparator):

ID: 1, Title: C Programming, Author: Dennis Ritchie

ID: 2, Title: Java Programming, Author: Herbert Schildt

ID: 3, Title: Java Programming, Author: James Gosling

ID: 4, Title: Python Basics, Author: Guido van Rossum

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

Code:

import java.util.\*;

// Employee class

class Employee {

String name;

double salary;

String department;

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

this.name = name;

this.salary = salary;

this.department = department;

}

@Override

public String toString() {

return name + " | Salary: " + salary + " | Dept: " + department;

}

}

public class EmployeeSortMenu {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

employees.add(new Employee("Bharadwaj", 50000, "IT"));

employees.add(new Employee("Anu", 60000, "Finance"));

employees.add(new Employee("Raj", 55000, "HR"));

employees.add(new Employee("Mohitha", 65000, "IT"));

employees.add(new Employee("Vikram", 48000, "Finance"));

int choice;

do {

System.out.println("\n--- Employee Sorting Menu ---");

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: ");

choice = sc.nextInt();

switch (choice) {

case 1:

Collections.sort(employees, new Comparator<Employee>() {

@Override

public int compare(Employee e1, Employee e2) {

return e1.name.compareTo(e2.name);

}

});

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

employees.forEach(System.out::println);

break;

case 2:

Collections.sort(employees, new Comparator<Employee>() {

@Override

public int compare(Employee e1, Employee e2) {

return Double.compare(e1.salary, e2.salary);

}

});

System.out.println("\nSorted by Salary:");

employees.forEach(System.out::println);

break;

case 3:

Collections.sort(employees, new Comparator<Employee>() {

@Override

public int compare(Employee e1, Employee e2) {

return e1.department.compareTo(e2.department);

}

});

System.out.println("\nSorted by Department:");

employees.forEach(System.out::println);

break;

case 4:

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

break;

default:

System.out.println("Invalid choice! Try again.");

}

} while (choice != 4);

sc.close();

}

}

Output:

1. Sort by Name

2. Sort by Salary

3. Sort by Department

4. Exit

Enter your choice: 1

Sorted by Name:

Anu | Salary: 60000.0 | Dept: Finance

Bharadwaj | Salary: 50000.0 | Dept: IT

Mohitha | Salary: 65000.0 | Dept: IT

Raj | Salary: 55000.0 | Dept: HR

Vikram | Salary: 48000.0 | Dept: Finance

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

Code:

import java.util.\*;

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;

}

// Getters

public String getName() { return name; }

public double getSalary() { return salary; }

public String getDepartment() { return department; }

@Override

public String toString() {

return name + " | Salary: " + salary + " | Dept: " + department;

}

}

public class ComparatorMethodRefDemo {

public static void main(String[] args) {

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

employees.add(new Employee("Bharadwaj", 50000, "IT"));

employees.add(new Employee("Anu", 60000, "Finance"));

employees.add(new Employee("Raj", 55000, "HR"));

employees.add(new Employee("Mohitha", 65000, "IT"));

employees.add(new Employee("Vikram", 48000, "Finance"));

// Sort by Name

employees.sort(Comparator.comparing(Employee::getName));

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

employees.forEach(System.out::println);

// Sort by Salary

employees.sort(Comparator.comparing(Employee::getSalary));

System.out.println("\nSorted by Salary:");

employees.forEach(System.out::println);

// Sort by Department, then Name

employees.sort(Comparator.comparing(Employee::getDepartment)

.thenComparing(Employee::getName));

System.out.println("\nSorted by Department, then Name:");

employees.forEach(System.out::println);

}

}

Output:

Anu | Salary: 60000.0 | Dept: Finance

Bharadwaj | Salary: 50000.0 | Dept: IT

Mohitha | Salary: 65000.0 | Dept: IT

Raj | Salary: 55000.0 | Dept: HR

Vikram | Salary: 48000.0 | Dept: Finance

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

Code:

import java.util.\*;

// Person class

class Person {

String name;

int age;

Person(String name, int age) {

this.name = name;

this.age = age;

}

@Override

public String toString() {

return name + " - Age: " + age;

}

}

public class TreeSetCustomComparator {

public static void main(String[] args) {

// TreeSet with custom comparator (sort by age)

Set<Person> persons = new TreeSet<>(new Comparator<Person>() {

@Override

public int compare(Person p1, Person p2) {

return Integer.compare(p1.age, p2.age); // ascending age

}

});

// Adding persons

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

persons.add(new Person("Anu", 30));

persons.add(new Person("Raj", 22));

persons.add(new Person("Mohitha", 28));

// Display sorted set

System.out.println("Persons sorted by age:");

for (Person p : persons) {

System.out.println(p);

}

}

}

Output:

Persons sorted by age:

Raj - Age: 22

Bharadwaj - Age: 25

Mohitha - Age: 28

Anu - Age: 30

**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.

Code:

import java.io.FileWriter;

import java.io.IOException;

public class WriteStudentFile {

public static void main(String[] args) {

try {

// Create FileWriter object (will create file if it doesn't exist)

FileWriter writer = new FileWriter("student.txt");

// Write 5 student names (each on a new line)

writer.write("Bharadwaj\n");

writer.write("Anu\n");

writer.write("Raj\n");

writer.write("Mohitha\n");

writer.write("Vikram\n");

// Close the FileWriter

writer.close();

System.out.println("Successfully wrote student names to student.txt");

} catch (IOException e) {

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

e.printStackTrace();

}

}

}

Output:

Bharadwaj

Anu

Raj

Mohitha

Vikram

**Q2. Read from a File**

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

Code:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class ReadStudentFile {

public static void main(String[] args) {

try {

// Create FileReader and wrap it with BufferedReader

BufferedReader reader = new BufferedReader(new FileReader("student.txt"));

String line;

System.out.println("Contents of student.txt:");

// Read each line until end of file (null)

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

System.out.println(line);

}

// Close the reader

reader.close();

} catch (IOException e) {

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

e.printStackTrace();

}

}

}

Output:

Contents of student.txt:

Bharadwaj

Anu

Raj

Mohitha

Vikram

**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.

Code:

import java.io.FileWriter;

import java.io.IOException;

public class AppendStudentFile {

public static void main(String[] args) {

try {

// FileWriter in append mode (true)

FileWriter writer = new FileWriter("student.txt", true);

// Append a new student name (with newline)

writer.write("Suresh\n");

// Close the FileWriter

writer.close();

System.out.println("New student name appended successfully.");

} catch (IOException e) {

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

e.printStackTrace();

}

}

}

Output:

Bharadwaj

Anu

Raj

Mohitha

Vikram

>After appending :  
Bharadwaj

Anu

Raj

Mohitha

Vikram

Suresh

**Q4. Count Words and Lines**

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

Code:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class CountWordsAndLines {

public static void main(String[] args) {

int lineCount = 0;

int wordCount = 0;

try {

BufferedReader reader = new BufferedReader(new FileReader("notes.txt"));

String line;

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

lineCount++;

// Split line into words using whitespace regex

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

// Handle case for empty lines

if (!line.trim().isEmpty()) {

wordCount += words.length;

}

}

reader.close();

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();

}

}

}

Output:

Java is fun

I like programming

This is a test  
Number of lines: 3

Number of words: 9

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

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

Code:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class CopyFile {

public static void main(String[] args) {

try {

// Reader for source file

BufferedReader reader = new BufferedReader(new FileReader("source.txt"));

// Writer for destination file

FileWriter writer = new FileWriter("destination.txt");

String line;

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

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

}

// Close both streams

reader.close();

writer.close();

System.out.println("File copied successfully from source.txt to destination.txt");

} catch (IOException e) {

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

e.printStackTrace();

}

}

}

Output:

**source.txt :**

Java is fun.

File handling is useful.

**destination.txt:**

Java is fun.

File handling is useful.

**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

Code:

import java.io.File;

public class FileDetails {

public static void main(String[] args) {

// Create a File object for report.txt

File file = new File("report.txt");

// Check if the file exists

if (file.exists()) {

System.out.println("File exists!");

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 (bytes): " + file.length());

} else {

System.out.println("The file report.txt does not exist.");

}

}

}

Output:

File exists!

Absolute Path: C:\Users\Bharadwaj\Documents\report.txt

File Name: report.txt

Writable: true

Readable: true

File Size (bytes): 102

**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.

Code:

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

public class UserInputToFile {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

try {

// Create FileWriter for userinput.txt

FileWriter writer = new FileWriter("userinput.txt");

System.out.println("Enter text to write to userinput.txt (type 'exit' to stop):");

while (true) {

String line = sc.nextLine();

// Stop when user types 'exit'

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

break;

}

// Write line to file with a newline

writer.write(line + System.lineSeparator());

}

writer.close();

System.out.println("Successfully written to userinput.txt");

} catch (IOException e) {

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

e.printStackTrace();

}

sc.close();

}

}

Output:

Enter text to write to userinput.txt (type 'exit' to stop):

Hello, this is Bharadwaj.

Java file writing example.

exit

Successfully written to userinput.txt

**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.

Code:

import java.io.\*;

import java.util.\*;

public class ReverseFileLines {

public static void main(String[] args) {

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

try (BufferedReader br = new BufferedReader(new FileReader("data.txt"))) {

String line;

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

lines.add(line);

}

} catch (IOException e) {

System.out.println("Error reading from data.txt");

e.printStackTrace();

return;

}

Collections.reverse(lines);

try (BufferedWriter bw = new BufferedWriter(new FileWriter("reversed.txt"))) {

for (String line : lines) {

bw.write(line);

bw.newLine();

}

System.out.println("Successfully created reversed.txt with reversed lines.");

} catch (IOException e) {

System.out.println("Error writing to reversed.txt");

e.printStackTrace();

}

}

}

Output:

If **data.txt** contains:

nginx

CopyEdit

Apple

Banana

Cherry

**reversed.txt** will contain:

Cherry

Banana

Apple

**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.

Code:

import java.io.\*;

// Student class implementing Serializable

class Student implements Serializable {

private static final long serialVersionUID = 1L; // For version control

int id;

String name;

double marks;

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

this.id = id;

this.name = name;

this.marks = marks;

}

}

public class SerializeStudent {

public static void main(String[] args) {

// Create a student object

Student student = new Student(101, "John Doe", 88.5);

// Serialize object to student.ser

try (FileOutputStream fos = new FileOutputStream("student.ser");

ObjectOutputStream oos = new ObjectOutputStream(fos)) {

oos.writeObject(student);

System.out.println("Student object serialized and saved to student.ser");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Ouput:

Student object serialized and saved to student.ser

**Q10. Read Serialized Object from File**

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

Code:

import java.io.\*;

// Student class should be Serializable (same as in serialization program)

class Student implements Serializable {

int id;

String name;

double marks;

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

this.id = id;

this.name = name;

this.marks = marks;

}

}

public class DeserializeStudent {

public static void main(String[] args) {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.ser"))) {

// Reading the object from file

Student s = (Student) ois.readObject();

// Displaying object's content

System.out.println("Student Details:");

System.out.println("ID: " + s.id);

System.out.println("Name: " + s.name);

System.out.println("Marks: " + s.marks);

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

Output:

Student Details:

ID: 101

Name: John Doe

Marks: 85.5

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

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

import java.io.File;

import java.util.Scanner;

public class ListFilesInFolder {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Get folder path from user

System.out.print("Enter folder path: ");

String folderPath = sc.nextLine();

File folder = new File(folderPath);

// Check if folder exists and is a directory

if (folder.exists() && folder.isDirectory()) {

File[] files = folder.listFiles();

System.out.println("Files in the folder:");

boolean foundFile = false;

if (files != null) {

for (File file : files) {

if (file.isFile()) { // Only files, no directories

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

foundFile = true;

}

}

}

if (!foundFile) {

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

}

} else {

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

}

sc.close();

}

}

Output:

Files in the folder:

file1.txt

file2.pdf

**Q12. Delete a File**

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

import java.io.File;

import java.util.Scanner;

public class DeleteFileExample {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Get file path from user

System.out.print("Enter file path to delete: ");

String filePath = sc.nextLine();

File file = new File(filePath);

// Check if file exists

if (file.exists() && file.isFile()) {

if (file.delete()) {

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

} else {

System.out.println("Failed to delete the file.");

}

} else {

System.out.println("File does not exist.");

}

sc.close();

}

}

Output:

Enter file path to delete: C:\ExampleFolder\test.txt

File deleted successfully: test.txt

**Q13. Word Search in a File**

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

Code:

import java.io.\*;

import java.util.Scanner;

public class SearchWordInFile {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Ask for word to search

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

String word = sc.nextLine();

File file = new File("notes.txt");

boolean found = false;

try (BufferedReader br = new BufferedReader(new FileReader(file))) {

String line;

int lineNumber = 1;

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

if (line.toLowerCase().contains(word.toLowerCase())) {

System.out.println("Word found on line " + lineNumber + ": " + line);

found = true;

}

lineNumber++;

}

} catch (IOException e) {

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

}

if (!found) {

System.out.println("Word not found in the file.");

}

sc.close();

}

}

Output:

Case 1:

Enter word to search: java

Word found on line 2: I am learning Java programming.

**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

Code:

import java.io.\*;

public class ReplaceWordInFile {

public static void main(String[] args) {

File inputFile = new File("story.txt");

File outputFile = new File("updated\_story.txt");

try (BufferedReader br = new BufferedReader(new FileReader(inputFile));

BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile))) {

String line;

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

// Replace "Java" with "Python"

String updatedLine = line.replaceAll("\\bJava\\b", "Python");

bw.write(updatedLine);

bw.newLine();

}

System.out.println("File updated successfully. Check updated\_story.txt");

} catch (IOException e) {

System.out.println("Error processing the file: " + e.getMessage());

}

}

}

Output:

**story.txt**

I love Java programming.

Java is a popular language.

JavaScript is different from Java.