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.

ANS:

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;

}

public int compareTo(Student other) {

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

}

public String toString() {

return "RollNo: " + rollNo + ", Name: " + name + ", Marks: " + marks;

}

}

public class SortStudents {

public static void main(String[] args) {

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

students.add(new Student(3, "Alice", 85.5));

students.add(new Student(1, "Bob", 78.0));

students.add(new Student(2, "Charlie", 92.3));

Collections.sort(students);

for (Student s : students) {

System.out.println(s);

}

}

}

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

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

ANS:  
import java.util.\*;

class Product implements Comparable<Product> {

String name;

double price;

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;

}

}

public class SortProducts {

public static void main(String[] args) {

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

products.add(new Product("Laptop", 1500.0));

products.add(new Product("Phone", 800.0));

products.add(new Product("Tablet", 400.0));

Collections.sort(products);

for (Product p : products) {

System.out.println(p);

}

}

}

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

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

ANS:  
import java.util.\*;

class Employee implements Comparable<Employee> {

int id;

String name;

Employee(int id, String name) {

this.id = id;

this.name = name;

}

public int compareTo(Employee other) {

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

}

public String toString() {

return "ID: " + id + ", Name: " + name;

}

}

public class SortEmployees {

public static void main(String[] args) {

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

employees.add(new Employee(101, "Zara"));

employees.add(new Employee(102, "Mike"));

employees.add(new Employee(103, "Alice"));

Collections.sort(employees);

for (Employee e : employees) {

System.out.println(e);

}

}

}

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

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

ANS:

import java.util.\*;

class Book implements Comparable<Book> {

int bookId;

String title;

Book(int bookId, String title) {

this.bookId = bookId;

this.title = title;

}

public int compareTo(Book other) {

return Integer.compare(other.bookId, this.bookId); // descending order

}

public String toString() {

return "BookId: " + bookId + ", Title: " + title;

}

}

public class SortBooksDescending {

public static void main(String[] args) {

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

books.add(new Book(3, "Java Basics"));

books.add(new Book(1, "Data Structures"));

books.add(new Book(2, "Algorithms"));

Collections.sort(books);

for (Book b : books) {

System.out.println(b);

}

}

}

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

ANS:

import java.util.\*;

class Person implements Comparable<Person> {

int id;

String name;

Person(int id, String name) {

this.id = id;

this.name = name;

}

public int compareTo(Person other) {

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

}

public String toString() {

return "ID: " + id + ", Name: " + name;

}

}

public class SortCustomObjects {

public static void main(String[] args) {

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

people.add(new Person(3, "Zara"));

people.add(new Person(1, "Bob"));

people.add(new Person(2, "Alice"));

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

for (Person p : people) {

System.out.println(p);

}

Collections.sort(people);

System.out.println("\nAfter sorting:");

for (Person p : people) {

System.out.println(p);

}

}

}

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

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

ANS:  
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: " + rollNo + ", Name: " + name + ", Marks: " + marks;

}

}

public class SortByMarks {

public static void main(String[] args) {

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

students.add(new Student(1, "Alice", 85.5));

students.add(new Student(2, "Bob", 78.0));

students.add(new Student(3, "Charlie", 92.3));

// Using Comparator class (anonymous inner class)

/\*

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

public int compare(Student s1, Student s2) {

return Double.compare(s2.marks, s1.marks);

}

});

\*/

// Using lambda expression

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

for (Student s : students) {

System.out.println(s);

}

}

}

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

ANS:

import java.util.\*;

class Product {

String name;

double price;

Product(String name, double price) {

this.name = name;

this.price = price;

}

public String toString() {

return name + ": $" + price;

}

}

public class ProductSorting {

public static void main(String[] args) {

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

products.add(new Product("Laptop", 1500));

products.add(new Product("Phone", 800));

products.add(new Product("Tablet", 400));

Comparator<Product> priceAsc = Comparator.comparingDouble(p -> p.price);

Comparator<Product> priceDesc = (p1, p2) -> Double.compare(p2.price, p1.price);

Comparator<Product> nameAlpha = Comparator.comparing(p -> p.name);

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

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

products.sort(priceAsc);

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

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

products.sort(priceDesc);

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

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

products.sort(nameAlpha);

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

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

}

}

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

ANS:

import java.util.\*;

import java.time.LocalDate;

class Employee {

String name;

LocalDate joiningDate;

Employee(String name, LocalDate joiningDate) {

this.name = name;

this.joiningDate = joiningDate;

}

public String toString() {

return name + " joined on " + joiningDate;

}

}

public class SortByJoiningDate {

public static void main(String[] args) {

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

employees.add(new Employee("Alice", LocalDate.of(2020, 5, 10)));

employees.add(new Employee("Bob", LocalDate.of(2019, 3, 15)));

employees.add(new Employee("Charlie", LocalDate.of(2021, 1, 5)));

employees.sort(Comparator.comparing(e -> e.joiningDate));

for (Employee e : employees) {

System.out.println(e);

}

}

}

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

ANS:  
import java.util.\*;

class City {

String name;

int population;

City(String name, int population) {

this.name = name;

this.population = population;

}

public String toString() {

return name + " (Population: " + population + ")";

}

}

public class SortCitiesByPopulation {

public static void main(String[] args) {

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

cities.add(new City("New York", 8419000));

cities.add(new City("Los Angeles", 3980000));

cities.add(new City("Chicago", 2716000));

cities.sort(Comparator.comparingInt(c -> c.population));

for (City city : cities) {

System.out.println(city);

}

}

}

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

ANS:  
import java.util.\*;

public class SortStringsByLength {

public static void main(String[] args) {

List<String> strings = new ArrayList<>(Arrays.asList("apple", "banana", "fig", "date", "cherry"));

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

public int compare(String s1, String s2) {

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

}

});

for (String s : strings) {

System.out.println(s);

}

}

}

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.

ANS:  
import java.util.\*;

class Student implements Comparable<Student> {

String name;

double marks;

Student(String name, double marks) {

this.name = name;

this.marks = marks;

}

public int compareTo(Student other) {

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

}

public String toString() {

return name + ": " + marks;

}

}

public class StudentSorting {

public static void main(String[] args) {

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

students.add(new Student("Alice", 85.5));

students.add(new Student("Bob", 78.0));

students.add(new Student("Charlie", 92.3));

System.out.println("Sort by name (Comparable):");

Collections.sort(students);

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

System.out.println("\nSort by marks (Comparator):");

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

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

}

}

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

ANS:  
import java.util.\*;

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;

}

public int compareTo(Book other) {

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

}

public String toString() {

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

}

}

public class BookSorting {

public static void main(String[] args) {

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

books.add(new Book(3, "Java Basics", "John Doe"));

books.add(new Book(1, "Data Structures", "Alice Smith"));

books.add(new Book(2, "Algorithms", "Bob Johnson"));

books.add(new Book(4, "Algorithms", "Alice Smith"));

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

Collections.sort(books);

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

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

books.sort(Comparator.comparing((Book b) -> b.title)

.thenComparing(b -> b.author));

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

}

}

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

ANS:

import java.util.\*;

class Employee {

String name;

double salary;

String department;

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

this.name = name;

this.salary = salary;

this.department = department;

}

public String toString() {

return name + ", $" + 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("Alice", 70000, "HR"));

employees.add(new Employee("Bob", 85000, "IT"));

employees.add(new Employee("Charlie", 65000, "Finance"));

employees.add(new Employee("David", 90000, "IT"));

while (true) {

System.out.println("\nSort Employees By:");

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

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

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

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

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

int choice = sc.nextInt();

if (choice == 4) break;

switch (choice) {

case 1:

employees.sort(Comparator.comparing(e -> e.name));

break;

case 2:

employees.sort(Comparator.comparingDouble(e -> e.salary));

break;

case 3:

employees.sort(Comparator.comparing(e -> e.department));

break;

default:

System.out.println("Invalid choice");

continue;

}

System.out.println("\nSorted Employees:");

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

}

sc.close();

}

}

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

ANS:  
import java.util.\*;

class Employee {

String name;

double salary;

Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

public String getName() { return name; }

public double getSalary() { return salary; }

public String toString() {

return name + ": $" + salary;

}

}

public class ComparatorMethodReference {

public static void main(String[] args) {

List<Employee> employees = Arrays.asList(

new Employee("Alice", 70000),

new Employee("Bob", 85000),

new Employee("Charlie", 65000)

);

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

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

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

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

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

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

}

}

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

ANS:

import java.util.\*;

class Person {

String name;

int age;

Person(String name, int age) {

this.name = name;

this.age = age;

}

public String toString() {

return name + " (" + age + ")";

}

}

public class TreeSetCustomComparator {

public static void main(String[] args) {

Comparator<Person> ageComparator = Comparator.comparingInt(p -> p.age);

TreeSet<Person> people = new TreeSet<>(ageComparator);

people.add(new Person("Alice", 30));

people.add(new Person("Bob", 25));

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

people.add(new Person("David", 28));

for (Person p : people) {

System.out.println(p);

}

}

}

**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.

ANS:

import java.io.FileWriter;

import java.io.IOException;

public class CreateWriteFile {

public static void main(String[] args) {

try (FileWriter writer = new FileWriter("student.txt")) {

writer.write("Alice\n");

writer.write("Bob\n");

writer.write("Charlie\n");

writer.write("David\n");

writer.write("Eve\n");

} catch (IOException e) {

System.out.println("Error writing to file: " + e.getMessage());

}

}

}

**Q2. Read from a File**

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

ANS:  
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class ReadFile {

public static void main(String[] args) {

try (BufferedReader reader = new BufferedReader(new FileReader("student.txt"))) {

String line;

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

System.out.println(line);

}

} catch (IOException e) {

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

}

}

}

**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.

ANS:

import java.io.FileWriter;

import java.io.IOException;

public class AppendToFile {

public static void main(String[] args) {

try (FileWriter writer = new FileWriter("student.txt", true)) { // true for append mode

writer.write("Frank\n");

} catch (IOException e) {

System.out.println("Error appending to file: " + e.getMessage());

}

}

}

**Q4. Count Words and Lines**

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

ANS:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class CountWordsLines {

public static void main(String[] args) {

int lines = 0;

int words = 0;

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

String line;

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

lines++;

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

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

words += wordArray.length;

}

}

} catch (IOException e) {

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

}

System.out.println("Lines: " + lines);

System.out.println("Words: " + words);

}

}

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

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

ANS:

import java.io.\*;

public class CopyFileContents {

public static void main(String[] args) {

try (BufferedReader br = new BufferedReader(new FileReader("source.txt"));

BufferedWriter bw = new BufferedWriter(new FileWriter("destination.txt"))) {

String line;

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

bw.write(line);

bw.newLine();

}

} catch (IOException e) {

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

}

}

}

**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

ANS:  
import java.io.File;

public class FileProperties {

public static void main(String[] args) {

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

if (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("File does not exist.");

}

}

}

**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.

ANS:  
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);

System.out.println("Enter text to write to file:");

String input = sc.nextLine();

try (FileWriter writer = new FileWriter("userinput.txt")) {

writer.write(input);

} catch (IOException e) {

System.out.println("Error writing to file: " + e.getMessage());

}

sc.close();

}

}

**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.

ANS:

import java.io.\*;

import java.util.\*;

public class ReverseFileContent {

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 file: " + e.getMessage());

return;

}

Collections.reverse(lines);

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

for (String l : lines) {

bw.write(l);

bw.newLine();

}

} catch (IOException e) {

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

}

}

}

**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.

ANS:

import java.io.\*;

class Student implements Serializable {

private static final long serialVersionUID = 1L;

int id;

String name;

double marks;

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) {

Student s = new Student(101, "Alice", 88.5);

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("student.ser"))) {

oos.writeObject(s);

} catch (IOException e) {

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

}

}

}

**Q10. Read Serialized Object from File**

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

ANS:

import java.io.\*;

class Student implements Serializable {

private static final long serialVersionUID = 1L;

int id;

String name;

double marks;

public String toString() {

return "ID: " + id + ", Name: " + name + ", Marks: " + marks;

}

}

public class DeserializeStudent {

public static void main(String[] args) {

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

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

System.out.println(s);

} catch (IOException | ClassNotFoundException e) {

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

}

}

}

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

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

ANS:

import java.io.File;

import java.util.Scanner;

public class ListFilesInDirectory {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String path = sc.nextLine();

File dir = new File(path);

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

File[] files = dir.listFiles();

if (files != null) {

for (File f : files) {

if (f.isFile()) {

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

}

}

}

} else {

System.out.println("Invalid directory path.");

}

sc.close();

}

}

**Q12. Delete a File**

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

ANS:

import java.io.File;

import java.util.Scanner;

public class DeleteFile {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String filename = sc.nextLine();

File file = new File(filename);

if (file.exists()) {

if (file.delete()) {

System.out.println("File deleted successfully.");

} else {

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

}

} else {

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

}

sc.close();

}

}

**Q13. Word Search in a File**

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

ANS:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Scanner;

public class WordSearchInFile {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String word = sc.nextLine();

boolean found = false;

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

String line;

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

if (line.contains(word)) {

found = true;

break;

}

}

} catch (IOException e) {

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

}

System.out.println(found ? "Word found." : "Word not found.");

sc.close();

}

}

**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

ANS:

import java.io.\*;

public class ReplaceWordInFile {

public static void main(String[] args) {

StringBuilder content = new StringBuilder();

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

String line;

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

content.append(line.replace("Java", "Python")).append("\n");

}

} catch (IOException e) {

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

return;

}

try (BufferedWriter bw = new BufferedWriter(new FileWriter("updated\_story.txt"))) {

bw.write(content.toString());

} catch (IOException e) {

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

}

}

}