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.

ANSWER

```
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 other) {
        return Integer.compare(this.rollNo, other.rollNo); // ascending
    }


    @Override
    public String toString() {
        return rollNo + " - " + name + " - " + marks;
    }
}
```

```
public class StudentSortDemo {
  public static void main(String[] args) {
    List<Student> list = new ArrayList<>();
    list.add(new Student(3, "Sam", 88.5));
    list.add(new Student(1, "Vikram", 92.0));
    list.add(new Student(2, "Neha", 76.3));

    Collections.sort(list);
    System.out.println("Sorted Students by RollNo:");
    list.forEach(System.out::println);
  }
}
```

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

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

ANSWER

```
import java.util.*;

class Product implements Comparable<Product> {
  String name;
  double price;

  Product(String name, double price) {
    this.name = name;
    this.price = price;
  }

  @Override
  public int compareTo(Product other) {
        return Double.compare(this.price, other.price); // ascending by price
    }

    @Override
    public String toString() {
        return name + " - ₹" + price;
    }
}

public class ProductSortDemo {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        products.add(new Product("Laptop", 55000));
        products.add(new Product("Mobile", 20000));
        products.add(new Product("Tablet", 30000));

        Collections.sort(products);
        System.out.println("Products sorted by Price:");
        products.forEach(System.out::println);
    }
}
```

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

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

ANSWER

```
import java.util.*;

class Employee implements Comparable<Employee> {
    int id;
    String name;

    Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public int compareTo(Employee other) {
        return this.name.compareTo(other.name); // alphabetical order
    }

    @Override
    public String toString() {
        return id + " - " + name;
    }
}

public class EmployeeSortDemo {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee(101, "Ramesh"));
        employees.add(new Employee(103, "Anita"));
        employees.add(new Employee(102, "Vikas"));

        Collections.sort(employees);
        System.out.println("Employees sorted by Name:");
        employees.forEach(System.out::println);
```

```
  }
}
```

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

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

ANSWER

```
import java.util.*;

class Book implements Comparable<Book> {
  int bookId;
  String title;

  Book(int bookId, String title) {
    this.bookId = bookId;
    this.title = title;
  }

  @Override
  public int compareTo(Book other) {
    return Integer.compare(other.bookId, this.bookId); // descending order
  }

  @Override
  public String toString() {
    return bookId + " - " + title;
  }
}
```

```
public class BookSortDemo {
  public static void main(String[] args) {
    List<Book> books = new ArrayList<>();
    books.add(new Book(201, "Java Basics"));
    books.add(new Book(105, "Python Guide"));
    books.add(new Book(301, "C++ Advanced"));

    Collections.sort(books);
    System.out.println("Books sorted by bookId (Descending):");
    books.forEach(System.out::println);
  }
}
```

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

ANSWER

```
import java.util.*;

class Car implements Comparable<Car> {
  int modelNo;
  String brand;

  Car(int modelNo, String brand) {
    this.modelNo = modelNo;
    this.brand = brand;
  }

  @Override
```

```
    public int compareTo(Car other) {
        return Integer.compare(this.modelNo, other.modelNo);
    }

    @Override
    public String toString() {
        return modelNo + " - " + brand;
    }
}

public class CustomSortDemo {
    public static void main(String[] args) {
        List<Car> cars = new ArrayList<>();
        cars.add(new Car(2020, "BMW"));
        cars.add(new Car(2018, "Audi"));
        cars.add(new Car(2022, "Tesla"));

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

        Collections.sort(cars);

        System.out.println("\nAfter Sorting by ModelNo:");
        cars.forEach(System.out::println);
    }
}
```

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

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

ANSWER

```
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;
    }


    @Override
    public String toString() {
        return rollNo + " - " + name + " - " + marks;
    }
}


public class SortByMarks {
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();
        list.add(new Student(1, "Sam", 85.6));
        list.add(new Student(2, "Ravi", 91.2));
        list.add(new Student(3, "Neha", 78.4));


        // Comparator using lambda
        list.sort((s1, s2) -> Double.compare(s2.marks, s1.marks));

        System.out.println("Students sorted by Marks (Descending):");
        list.forEach(System.out::println);
    }
}
```

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

ANSWER

```
import java.util.*;

class Product {
    String name;
    double price;

    Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public String toString() {
        return name + " - ₹" + price;
    }
}

public class ProductSortStrategies {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        products.add(new Product("Laptop", 50000));
        products.add(new Product("Mobile", 20000));
        products.add(new Product("Tablet", 30000));

        // Price ascending
        products.sort(Comparator.comparingDouble(p -> p.price));
        System.out.println("Sorted by Price (Ascending): " + products);

        // Price descending
        products.sort((p1, p2) -> Double.compare(p2.price, p1.price));
        System.out.println("Sorted by Price (Descending): " + products);

        // Name alphabetically
        products.sort(Comparator.comparing(p -> p.name));
        System.out.println("Sorted by Name: " + products);
    }
}
```

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

ANSWER

```
import java.time.LocalDate;
import java.util.*;

class Employee {
    String name;
    LocalDate joiningDate;

    Employee(String name, LocalDate joiningDate) {
        this.name = name;
        this.joiningDate = joiningDate;
    }

    @Override
    public String toString() {
        return name + " - Joined: " + joiningDate;
    }
}

public class EmployeeSortByDate {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Amit", LocalDate.of(2020, 5, 10)));
        employees.add(new Employee("Neha", LocalDate.of(2019, 3, 15)));
        employees.add(new Employee("Ravi", LocalDate.of(2021, 1, 5)));

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

        System.out.println("Employees sorted by Joining Date:");
        employees.forEach(System.out::println);
    }
}
```

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

ANSWER

```
import java.util.*;

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 CitySort {
   public static void main(String[] args) {
      List<City> cities = new ArrayList<>();
      cities.add(new City("Delhi", 19000000));
      cities.add(new City("Mumbai", 21000000));
      cities.add(new City("Pune", 7000000));
```

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

        System.out.println("Cities sorted by Population:");
        cities.forEach(System.out::println);
    }
}
```

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

ANSWER

```
import java.util.*;

public class SortStringsByLength {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Java", "SpringBoot", "AI", "Python");

        Collections.sort(list, new Comparator<String>() {
            @Override
            public int compare(String s1, String s2) {
                return Integer.compare(s1.length(), s2.length());
            }
        });

        System.out.println("Strings sorted by length: " + list);
    }
}
```

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.

ANSWER

```
import java.util.*;

class Student implements Comparable<Student> {
    String name;
    double marks;

    Student(String name, double marks) {
        this.name = name;
        this.marks = marks;
    }

    @Override
    public int compareTo(Student other) {
        return this.name.compareTo(other.name); // Comparable = by name
    }

    @Override
    public String toString() {
        return name + " - " + marks;
    }
}

public class StudentSortDemo {
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();
        list.add(new Student("Ravi", 88.0));
        list.add(new Student("Amit", 92.5));
        list.add(new Student("Neha", 75.3));

        Collections.sort(list); // Comparable = by Name
        System.out.println("Sorted by Name (Comparable): " + list);

        list.sort((s1, s2) -> Double.compare(s2.marks, s1.marks)); // Comparator
        System.out.println("Sorted by Marks (Comparator): " + list);
    }
}
```

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

ANSWER

```
import java.util.*;

class Book implements Comparable<Book> {
    int id;
    String title, author;

    Book(int id, String title, String author) {
        this.id = id;
        this.title = title;
        this.author = author;
    }

    @Override
    public int compareTo(Book other) {
        return Integer.compare(this.id, other.id); // Comparable = by ID
    }

    @Override
    public String toString() {
        return id + " - " + title + " by " + author;
    }
}

public class BookSortDemo {
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();
        books.add(new Book(3, "Java", "James"));
        books.add(new Book(1, "Python", "Guido"));
        books.add(new Book(2, "C++", "Bjarne"));

        Collections.sort(books); // Comparable = by ID
        System.out.println("Sorted by ID: " + books);

        books.sort(Comparator.comparing((Book b) -> b.title).thenComparing(b -> b.author));
        System.out.println("Sorted by Title, then Author: " + books);
    }
}
```

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

ANSWER

```
import java.util.*;

class Employee {
    String name, dept;
    double salary;

    Employee(String name, String dept, double salary) {
        this.name = name;
        this.dept = dept;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return name + " - " + dept + " - ₹" + salary;
    }
}

public class EmployeeMenuSort {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Ravi", "HR", 50000));
        employees.add(new Employee("Amit", "IT", 60000));
        employees.add(new Employee("Neha", "Finance", 55000));

        Scanner sc = new Scanner(System.in);
        System.out.println("1. Sort by Name\n2. Sort by Salary\n3. Sort by Department");
        int choice = sc.nextInt();
        switch (choice) {
            case 1 -> employees.sort(Comparator.comparing(e -> e.name));
            case 2 -> employees.sort(Comparator.comparingDouble(e -> e.salary));
            case 3 -> employees.sort(Comparator.comparing(e -> e.dept));
        }

        System.out.println("Sorted Employees:");
        employees.forEach(System.out::println);
    }
}
```

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

ANSWER

```
import java.util.*;

class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return name + " - " + age;
    }
}

public class ComparatorWithMethodRef {
  public static void main(String[] args) {
    List<Person> list = new ArrayList<>();
    list.add(new Person("Ravi", 25));
    list.add(new Person("Neha", 30));
    list.add(new Person("Amit", 22));

    list.sort(Comparator.comparing(Person::getName)); // method reference
    list.forEach(System.out::println);
  }
}

// Need getter:
class Person {
  String name;
  int age;
  Person(String name, int age) { this.name = name; this.age = age; }
  String getName() { return name; }
  int getAge() { return age; }
  public String toString() { return name + " - " + age; }
}
```

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

ANSWER

```
import java.util.*;
```

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return name + " - " + age;
    }
}

public class TreeSetCustomComparator {
    public static void main(String[] args) {
        TreeSet<Person> set = new TreeSet<>(Comparator.comparingInt(p -> p.age));

        set.add(new Person("Ravi", 25));
        set.add(new Person("Neha", 30));
        set.add(new Person("Amit", 22));

        System.out.println("Persons sorted by Age (TreeSet):");
        set.forEach(System.out::println);
    }
}
```

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.

ANSWER

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

public class Q1_CreateFile {
  public static void main(String[] args) {
    try (FileWriter fw = new FileWriter("student.txt")) {
      fw.write("Aman\n");
      fw.write("Riya\n");
      fw.write("Samarth\n");
      fw.write("Karan\n");
      fw.write("Priya\n");
      System.out.println("student.txt created and data written.");
    } catch (IOException e) {
      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.

ANSWER

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

import java.io.IOException;


public class Q2_ReadFile {

  public static void main(String[] args) {

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

      String line;

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

        System.out.println(line);

      }

    } catch (IOException e) {

      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.

ANSWER

```
import java.io.FileWriter;

import java.io.IOException;


public class Q3_AppendFile {

  public static void main(String[] args) {

    try (FileWriter fw = new FileWriter("student.txt", true)) {

      fw.write("NewStudent\n");

      System.out.println("Data appended successfully.");
```

```
        } catch (IOException e) {
            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.

ANSWER

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

public class Q4_CountWordsLines {
    public static void main(String[] args) {
        int lineCount = 0, wordCount = 0;
        try (BufferedReader br = new BufferedReader(new FileReader("notes.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                lineCount++;
                wordCount += line.split("\\s+").length;
            }
            System.out.println("Lines: " + lineCount);
            System.out.println("Words: " + wordCount);
        } catch (IOException e) {
            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.

ANSWER

```
import java.io.*;


public class Q5_CopyFile {
    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();
            }
            System.out.println("File copied successfully.");
        } catch (IOException e) {
            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

ANSWER

```
import java.io.File;


public class Q6_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: " + file.length() + " bytes");
    } 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.

ANSWER

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
```

```
public class Q7_UserInputFile {
  public static void main(String[] args) {
    try (Scanner sc = new Scanner(System.in);
       FileWriter fw = new FileWriter("userinput.txt")) {
      System.out.println("Enter text (type 'exit' to stop): ");
      while (true) {
        String input = sc.nextLine();
        if (input.equalsIgnoreCase("exit")) break;
        fw.write(input + "\n");
      }
      System.out.println("Data written to userinput.txt");
    } catch (IOException e) {
      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.

ANSWER

```
import java.io.*;
import java.util.*;


public class Q8_ReverseFile {
  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) {
            e.printStackTrace();
        }

        try (BufferedWriter bw = new BufferedWriter(new FileWriter("reversed.txt"))) {
            Collections.reverse(lines);
            for (String l : lines) {
                bw.write(l);
                bw.newLine();
            }
            System.out.println("Content reversed into reversed.txt");
        } catch (IOException e) {
            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.

ANSWER

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

```
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 Q9_Serialize {
  public static void main(String[] args) {
    Student s = new Student(101, "Samarth", 85.5);
    try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("student.ser"))) {
      oos.writeObject(s);
      System.out.println("Object serialized to student.ser");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Q10. Read Serialized Object from File

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

ANSWER

```
import java.io.*;


public class Q10_Deserialize {
  public static void main(String[] args) {
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.ser"))) {
      Student s = (Student) ois.readObject();
      System.out.println("Deserialized Student: " + s.id + ", " + s.name + ", " + s.marks);
    } catch (Exception e) {
      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.

ANSWER

```
import java.io.File;
import java.util.Scanner;


public class Q11_ListFiles {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter directory path: ");
    String path = sc.nextLine();
    File folder = new File(path);


    if (folder.isDirectory()) {
      for (File file : folder.listFiles()) {
```

```
                if (file.isFile()) {
                    System.out.println(file.getName());
                }
            }
        } else {
            System.out.println("Invalid directory.");
        }
    }
}
```

Q12. Delete a File

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

ANSWER

```
import java.io.File;
import java.util.Scanner;

public class Q12_DeleteFile {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter file name 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 file.");
        }
      } else {
        System.out.println("File does not exist.");
      }
    }
}
```

Q13. Word Search in a File

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

ANSWER

```
import java.io.*;
import java.util.Scanner;

public class Q13_WordSearch {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter word to search: ");
        String word = sc.nextLine();

        try (BufferedReader br = new BufferedReader(new FileReader("notes.txt"))) {
            String line;
            boolean found = false;
            while ((line = br.readLine()) != null) {
                if (line.contains(word)) {
                    found = true;
                    break;
                }
            }
            System.out.println(found ? "Word found in file." : "Word not found.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

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

ANSWER

```
import java.io.*;

public class Q14_ReplaceWord {
    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.replaceAll("Java", "Python")).append("\n");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        try (BufferedWriter bw = new BufferedWriter(new FileWriter("updated_story.txt"))) {
            bw.write(content.toString());
            System.out.println("Word replaced successfully into updated_story.txt");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```