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.

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

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

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

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

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

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

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

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

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

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

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

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

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.

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

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

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

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

**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.

**Q2. Read from a File**

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

**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.

**Q4. Count Words and Lines**

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

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

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

**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

**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.

**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.

**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.

**Q10. Read Serialized Object from File**

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

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

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

**Q12. Delete a File**

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

**Q13. Word Search in a File**

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

**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