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.

```
import java.util.*;

class Student implements Comparable<Student> {
    int rollNo;
    String name;
    int marks;

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

    public int compareTo(Student s) {
        return this.rollNo - s.rollNo;
    }

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

public class SortStudentsByRollNo {
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();
        list.add(new Student(3, "Amit", 85));
        list.add(new Student(1, "Rahul", 92));
        list.add(new Student(2, "Kiran", 78));

        Collections.sort(list);
```

```
        for (Student s : list) {
            System.out.println(s);
        }
    }
}
```

Output:

1 Rahul 92

2 Kiran 78

3 Amit 85

Q2. Create a Product class and sort products by price using Comparable. Implement Comparable<Product> and sort a list of products using Collections.sort().

```
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 p) {
        return Double.compare(this.price, p.price);
    }

    public String toString() {
        return name + " " + price;
    }
}
```

```
public class SortProductsByPrice {
    public static void main(String[] args) {
        List<Product> list = new ArrayList<>();
        list.add(new Product("Pen", 10.5));
        list.add(new Product("Pencil", 5.0));
        list.add(new Product("Notebook", 20.0));

        Collections.sort(list);

        for (Product p : list) {
            System.out.println(p);
        }
    }
}
```

Output:

Pencil 5.0

Pen 10.5

Notebook 20.0

Q3. Create an Employee class and sort by name using Comparable. Use the compareTo() method to sort alphabetically by employee names.

```
import java.util.*;
class Employee implements Comparable<Employee> {
    String name;
    int id;

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

    public int compareTo(Employee e) {
```

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

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

public class SortEmployeeByName {
    public static void main(String[] args) {
        List<Employee> list = new ArrayList<>();
        list.add(new Employee("Raj", 101));
        list.add(new Employee("Anita", 102));
        list.add(new Employee("Zara", 103));

        Collections.sort(list);

        for (Employee e : list) {
            System.out.println(e);
        }
    }
}
```

Output:

```
102 Anita
101 Raj
103 Zara
```

Q4. Sort a list of Book objects by bookId in descending order using Comparable. Hint: Override compareTo() to return the reverse order.

```
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 b) {
        return b.bookId - this.bookId;
    }

    public String toString() {
        return bookId + " " + title;
    }
}

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

        Collections.sort(list);

        for (Book b : list) {
            System.out.println(b);
        }
    }
}
```

Output:

3 Java

2 C++

1 Python

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

```
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 p) {
      return this.id - p.id;
   }

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

public class SortPerson {
   public static void main(String[] args) {
      List<Person> list = new ArrayList<>();
      list.add(new Person(3, "John"));
      list.add(new Person(1, "Alice"));
      list.add(new Person(2, "Bob"));
```

```
        System.out.println("Before Sorting:");
        for (Person p : list)
            System.out.println(p);

        Collections.sort(list);

        System.out.println("After Sorting:");
        for (Person p : list)
            System.out.println(p);
    }
}
```

Output:

```
Before Sorting:
3 John
1 Alice
2 Bob
After Sorting:
1 Alice
2 Bob
3 John
```

Q6. Sort a list of students by marks (descending) using Comparator. Create a Comparator class or use a lambda expression to sort by marks.

```
import java.util.*;
class Student {
    String name;
    int marks;

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

```
    public String toString() {
        return name + " " + marks;
    }
}

public class SortStudentsByMarks {
    public static void main(String[] args) {
        List<Student> list = new ArrayList<>();
        list.add(new Student("Amit", 85));
        list.add(new Student("Rahul", 92));
        list.add(new Student("Kiran", 78));

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

        for (Student s : list) {
            System.out.println(s);
        }
    }
}
```

Output:

Rahul 92

Amit 85

Kiran 78

Q7. Create multiple sorting strategies for a Product class. Implement comparators to sort by: Price ascending, Price descending, Name alphabetically.

```
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 SortProductStrategies {
    public static void main(String[] args) {
        List<Product> list = new ArrayList<>();
        list.add(new Product("Pen", 10.5));
        list.add(new Product("Pencil", 5.0));
        list.add(new Product("Notebook", 20.0));

        Collections.sort(list, Comparator.comparingDouble(p -> p.price));
        System.out.println("Price ascending:");
        list.forEach(System.out::println);

        Collections.sort(list, (p1, p2) -> Double.compare(p2.price, p1.price));
        System.out.println("Price descending:");
        list.forEach(System.out::println);

        Collections.sort(list, Comparator.comparing(p -> p.name));
        System.out.println("Name alphabetically:");
        list.forEach(System.out::println);
    }
}
```

Output:

Price ascending:

Pencil 5.0

Pen 10.5

Notebook 20.0

Price descending:

Notebook 20.0

Pen 10.5

Pencil 5.0

Name alphabetically:

Notebook 20.0

Pen 10.5

Pencil 5.0

Q8. Sort Employee objects by joining date using Comparator. Use Comparator to sort employees based on LocalDate or Date.

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

class Employee {
   String name;
   LocalDate joiningDate;

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

   public String toString() {
      return name + " " + joiningDate;
   }
}
```

```
public class SortEmployeeByDate {
    public static void main(String[] args) {
        List<Employee> list = new ArrayList<>();
        list.add(new Employee("Raj", LocalDate.of(2020, 5, 10)));
        list.add(new Employee("Anita", LocalDate.of(2018, 3, 15)));
        list.add(new Employee("Zara", LocalDate.of(2021, 1, 20)));

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

        list.forEach(System.out::println);
    }
}
```

Output:

```
Anita 2018-03-15
Raj 2020-05-10
Zara 2021-01-20
```

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

```
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;
    }
}
```

```
public class SortCitiesByPopulation {
    public static void main(String[] args) {
        List<City> list = new ArrayList<>();
        list.add(new City("Delhi", 19000000));
        list.add(new City("Mumbai", 20000000));
        list.add(new City("Chennai", 10000000));

        Collections.sort(list, Comparator.comparingInt(c -> c.population));

        list.forEach(System.out::println);
    }
}
```

Output:

Chennai 10000000

Delhi 19000000

Mumbai 20000000

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

```
import java.util.*;
public class SortStringsByLength {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("apple");
        list.add("banana");
        list.add("kiwi");

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

```
        list.forEach(System.out::println);
    }
}
```

Output:

kiwi

apple

banana

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.

```
import java.util.*;
class Student implements Comparable<Student> {
    String name;
    int marks;

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

    public int compareTo(Student s) {
        return this.name.compareTo(s.name);
    }

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

public class SortStudentNameAndMarks {
    public static void main(String[] args) {
```

```
        List<Student> list = new ArrayList<>();
        list.add(new Student("Amit", 85));
        list.add(new Student("Rahul", 92));
        list.add(new Student("Kiran", 78));

        System.out.println("Sort by Name:");
        Collections.sort(list);
        list.forEach(System.out::println);

        System.out.println("Sort by Marks:");
        Collections.sort(list, Comparator.comparingInt(s -> s.marks));
        list.forEach(System.out::println);
    }
}
```

Output:

Sort by Name:

Amit 85

Kiran 78

Rahul 92

Sort by Marks:

Kiran 78

Amit 85

Rahul 92

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

```
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 b) {
        return this.id - b.id;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public String toString() {
        return id + " " + title + " " + author;
    }
}

public class SortBookByIdAndTitle {
    public static void main(String[] args) {
        List<Book> list = new ArrayList<>();
        list.add(new Book(3, "Java", "Author1"));
        list.add(new Book(1, "Python", "Author2"));
        list.add(new Book(2, "C++", "Author3"));

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

```
        Collections.sort(list);
        list.forEach(System.out::println);

        System.out.println("Sort by Title and Author:");
        Collections.sort(list, Comparator.comparing(Book::getTitle).thenComparing(Book::getAuthor));
        list.forEach(System.out::println);
    }
}
```

Output:

```
Sort by ID:
1 Python Author2
2 C++ Author3
3 Java Author1
Sort by Title and Author:
2 C++ Author3
3 Java Author1
1 Python Author2
```

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

```
import java.util.*;
import java.util.Scanner;
class Employee {
    String name, department;
    int salary;

    Employee(String name, String department, int salary) {
        this.name = name;
        this.department = department;
        this.salary = salary;
    }
```

```
    public String toString() {
        return name + " " + department + " " + salary;
    }
}

public class EmployeeSortMenu {
    public static void main(String[] args) {
        List<Employee> list = new ArrayList<>();
        list.add(new Employee("Raj", "IT", 50000));
        list.add(new Employee("Anita", "HR", 60000));
        list.add(new Employee("Zara", "Finance", 55000));

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter 1: Sort by Name, 2: Salary, 3: Department");
        int choice = sc.nextInt();

        switch (choice) {
            case 1:
                Collections.sort(list, Comparator.comparing(e -> e.name));
                break;
            case 2:
                Collections.sort(list, Comparator.comparingInt(e -> e.salary));
                break;
            case 3:
                Collections.sort(list, Comparator.comparing(e -> e.department));
                break;
            default:
                System.out.println("Invalid Choice");
                sc.close();
                return;
        }
```

```
        list.forEach(System.out::println);
        sc.close();
    }
}
```

Output :

```
Anita HR 60000
Raj IT 50000
Zara Finance 55000
```

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

```
import java.util.*;
class Person {
    String name;
    int age;

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

    public String getName() { return name; }
    public int getAge() { return age; }

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

public class ComparatorMethodReference {
    public static void main(String[] args) {
        List<Person> list = new ArrayList<>();
```

```
        list.add(new Person("John", 25));
        list.add(new Person("Alice", 30));
        list.add(new Person("Bob", 20));

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

        list.forEach(System.out::println);
    }
}
```

Output:

```
Alice 30
Bob 20
John 25
```

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

```
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) {
```

```
        TreeSet<Person> set = new TreeSet<>(Comparator.comparingInt(p -> p.age));
        set.add(new Person("John", 25));
        set.add(new Person("Alice", 30));
        set.add(new Person("Bob", 20));

        set.forEach(System.out::println);
    }
}
```

Output:

Bob 20

John 25

Alice 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.

```
import java.io.FileWriter;
import java.io.IOException;
public class WriteToFile {
    public static void main(String[] args) {
        try {
            FileWriter fw = new FileWriter("student.txt");
            fw.write("Rahul\n");
            fw.write("Anita\n");
            fw.write("Kiran\n");
            fw.write("Amit\n");
            fw.write("Priya\n");
            fw.close();
            System.out.println("File written successfully");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
}
```

Output:

File written successfully

Q2. Read from a File. Write a program to read the contents of student.txt and display them line by line using BufferedReader.

```
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class ReadFromFile {

   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);

         }

         br.close();

      } catch (IOException e) {

         e.printStackTrace();

      }

   }

}
```

Output:

Rahul

Anita

Kiran

Amit

Priya

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.

```
import java.io.FileWriter;

import java.io.IOException;
```

```
public class AppendToFile {
   public static void main(String[] args) {
      try {
         FileWriter fw = new FileWriter("student.txt", true); // append mode true
         fw.write("Sunil\n");
         fw.close();
         System.out.println("Data appended successfully");
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Output:

Data appended successfully

Q4. Count the number of words and lines in a text file.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class CountWordsLines {
   public static void main(String[] args) {
      int lines = 0, words = 0;
      try {
         BufferedReader br = new BufferedReader(new FileReader("student.txt"));
         String line;
         while ((line = br.readLine()) != null) {
            lines++;
            String[] wordList = line.trim().split("\\s+");
            words += wordList.length;
         }
         br.close();
```

```
            System.out.println("Lines: " + lines);
            System.out.println("Words: " + words);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output :

Lines: 6

Words: 6

Q5. Copy contents from one file to another. Copy the content of student.txt to copy.txt.

```
import java.io.*;
public class CopyFile {
    public static void main(String[] args) {
        try {
            BufferedReader br = new BufferedReader(new FileReader("student.txt"));
            FileWriter fw = new FileWriter("copy.txt");

            String line;
            while ((line = br.readLine()) != null) {
                fw.write(line + "\n");
            }
            br.close();
            fw.close();
            System.out.println("File copied successfully");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

File copied successfully

Q6. Delete a specific line from a text file. Delete the line containing "Amit" from student.txt.

```
import java.io.*;
import java.util.*;
public class DeleteLineFromFile {
   public static void main(String[] args) {
      File inputFile = new File("student.txt");
      File tempFile = new File("temp.txt");

      try (BufferedReader br = new BufferedReader(new FileReader(inputFile));
          PrintWriter pw = new PrintWriter(new FileWriter(tempFile))) {
         String line;
         while ((line = br.readLine()) != null) {
            if (line.trim().equals("Amit")) continue; // skip line to delete
            pw.println(line);
         }
      } catch (IOException e) {
         e.printStackTrace();
      }

      inputFile.delete();
      tempFile.renameTo(inputFile);
      System.out.println("Line deleted successfully");
   }
}
```

Output:

Line deleted successfully

Q7. Serialize an object. Write a Java program to serialize a Student object to a file.

```
import java.io.*;
class Student implements Serializable {
   int id;
```

```
    String name;

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

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

public class SerializeObject {
    public static void main(String[] args) {
        Student s = new Student(1, "Rahul");
        try {
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("student.ser"));
            oos.writeObject(s);
            oos.close();
            System.out.println("Object serialized successfully");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

Object serialized successfully

Q8. Deserialize an object. Read the Student object from the serialized file and display its data.

```
import java.io.*;
public class DeserializeObject {
    public static void main(String[] args) {
```

```
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.ser"));
            Student s = (Student) ois.readObject();
            ois.close();
            System.out.println("Deserialized Student: " + s);
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

Deserialized Student: 1 Rahul

Q9. Count occurrences of a specific word in a file. Count how many times "Rahul" appears in student.txt.

```
import java.io.*;
public class CountWordOccurrences {
    public static void main(String[] args) {
        String word = "Rahul";
        int count = 0;
        try (BufferedReader br = new BufferedReader(new FileReader("student.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] words = line.split("\\s+");
                for (String w : words) {
                    if (w.equals(word))
                        count++;
                }
            }
            System.out.println("Word '" + word + "' found " + count + " times");
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }
    }
}
```

Output:

```
Word 'Rahul' found 1 times
```

Q10. Read a file using Scanner class and display contents.

```
import java.io.*;
import java.util.*;
public class ReadFileUsingScanner {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(new File("student.txt"));
            while (sc.hasNextLine()) {
                System.out.println(sc.nextLine());
            }
            sc.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
Rahul
Anita
Kiran
Priya
Sunil
```

Q11. Write a program to create a directory if it does not exist.

```
import java.io.File;
public class CreateDirectory {
    public static void main(String[] args) {
```

```
        File dir = new File("dataFolder");
        if (!dir.exists()) {
            if (dir.mkdir()) {
                System.out.println("Directory created");
            } else {
                System.out.println("Failed to create directory");
            }
        } else {
            System.out.println("Directory already exists");
        }
    }
}
```

Output :

Directory created

Q12. Rename a file from old.txt to new.txt.

```
import java.io.File;
public class RenameFile {
    public static void main(String[] args) {
        File oldFile = new File("old.txt");
        File newFile = new File("new.txt");

        if (oldFile.renameTo(newFile)) {
            System.out.println("File renamed successfully");
        } else {
            System.out.println("Rename failed");
        }
    }
}
```

Output:

File renamed successfully

Q13. Delete a file named temp.txt.

```
import java.io.File;
public class DeleteFile {
   public static void main(String[] args) {
      File file = new File("temp.txt");
      if (file.delete()) {
         System.out.println("File deleted");
      } else {
         System.out.println("Failed to delete file");
      }
   }
}
```

Output :

File deleted

Q14. List all files and directories in a folder named "dataFolder".

```
import java.io.File;
public class ListFiles {
   public static void main(String[] args) {
      File folder = new File("dataFolder");
      File[] list = folder.listFiles();

      if (list != null) {
         for (File f : list) {
            System.out.println((f.isDirectory() ? "Directory: " : "File: ") + f.getName());
         }
      } else {
         System.out.println("Folder not found or empty");
      }
   }
}
```

Output :

File: student.txt

File: copy.txt

Directory: subFolder