1.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.

```
package Assignment_Day9;

import java.util.*;

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

public 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);
}

@Override
public String toString() {
return "Student{rollNo=" + rollNo + ", name='" + name + "', marks=" + marks + "}";
}
}

public class Problem1 {
public static void main(String[] args) {
List<Student> students = new ArrayList<>();
students.add(new Student(3, "hari", 85.5));
students.add(new Student(1, "eswar", 90.0));
students.add(new Student(2, "kumar", 75.5));

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

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

Collections.sort(students);

System.out.println("After sorting");
students.forEach(System.out::println);
}
}
```

```
Before sorting:
Student{rollNo=3, name='hari', marks=85.5}
Student{rollNo=1, name='eswar', marks=90.0}
Student{rollNo=2, name='kumar', marks=75.5}
After sorting
Student{rollNo=1, name='eswar', marks=90.0}
Student{rollNo=2, name='kumar', marks=75.5}
Student{rollNo=3, name='hari', marks=85.5}
```

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

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

```
package Assignment_Day9;

import java.util.*;

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

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

```java
this.price = price;
}

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

@Override
public String toString() {
return "Product{name='" + name + "', price=" + price + "}";
}
}

public class Problem2 {
public static void main(String[] args) {
List<Product> products = new ArrayList<>();
products.add(new Product("Laptop", 999.99));
products.add(new Product("Phone", 699.99));
products.add(new Product("Tablet", 299.99));

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

Collections.sort(products);

System.out.println("After sorting");
products.forEach(System.out::println);
}
}
```

```
Before sorting:
Product{name='Laptop', price=999.99}
Product{name='Phone', price=699.99}
Product{name='Tablet', price=299.99}
After sorting
Product{name='Tablet', price=299.99}
Product{name='Phone', price=699.99}
Product{name='Laptop', price=999.99}
```

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

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

```
package Assignment_Day9;

import java.util.*;

class Employee implements Comparable<Employee> {
String name;
double salary;

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

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

@Override
public String toString() {
return "Employee{name='" + name + "', salary=" + salary + "}";
}
}

public class Problem3 {
public static void main(String[] args) {
List<Employee> employees = new ArrayList<>();
employees.add(new Employee("Hari", 60000));
employees.add(new Employee("Eswar", 50000));
employees.add(new Employee("Kumar", 70000));

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

Collections.sort(employees);
```

```
System.out.println("After sorting");
employees.forEach(System.out::println);
}
}
```

```
Before sorting:
Employee{name='Hari', salary=60000.0}
Employee{name='Eswar', salary=50000.0}
Employee{name='Kumar', salary=70000.0}
After sorting
Employee{name='Eswar', salary=50000.0}
Employee{name='Hari', salary=60000.0}
Employee{name='Kumar', salary=70000.0}
```

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

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

```
package Assignment_Day9;

import java.util.*;

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

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

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

```java
@Override
public String toString() {
return "Book{bookId=" + bookId + ", title='" + title + "'}";
}
}

public class Problem4 {
public static void main(String[] args) {
List<Book> books = new ArrayList<>();
books.add(new Book(1, "Java Programming"));
books.add(new Book(3, "Python Basics"));
books.add(new Book(2, "C for Beginners"));

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

Collections.sort(books);

System.out.println("\nAfter sorting by bookid in descending order");
books.forEach(System.out::println);
}
}
```

```
Before sorting:
Book{bookId=1, title='Java Programming'}
Book{bookId=3, title='Python Basics'}
Book{bookId=2, title='C for Beginners'}

After sorting by bookid in descending order
Book{bookId=3, title='Python Basics'}
Book{bookId=2, title='C for Beginners'}
Book{bookId=1, title='Java Programming'}
```

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

```java
package Assignment_Day9;

import java.util.*;

class Person implements Comparable<Person> {
String name;
int age;

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

@Override
public int compareTo(Person other) {
return this.name.compareTo(other.name);
}

@Override
public String toString() {
return "Person{name='" + name + "', age=" + age + "}";
}
}

public class Problem5 {
public static void main(String[] args) {
List<Person> people = new ArrayList<>();
people.add(new Person("Hari", 25));
people.add(new Person("Eswar", 30));
people.add(new Person("Kumar", 20));

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

Collections.sort(people);

System.out.println("After sorting");
people.forEach(System.out::println);
}
}
```

```
Before sorting:
Person{name='Hari', age=25}
Person{name='Eswar', age=30}
Person{name='Kumar', age=20}
After sorting
Person{name='Eswar', age=30}
Person{name='Hari', age=25}
Person{name='Kumar', age=20}
```

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

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

```
package Assignment_Day9;

import java.util.*;

class Student {
int rollNo;
String name;
double marks;

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

@Override
public String toString() {
return "Student{rollNo=" + rollNo + ", name='" + name + "', marks=" + marks + "}";
}
}

public class Problem6 {
public static void main(String[] args) {
List<Student> students = new ArrayList<>();
```

```java
students.add(new Student(1, "Hari", 85.5));
students.add(new Student(2, "Eswar", 90.0));
students.add(new Student(3, "Kumar", 75.5));

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

Comparator<Student> marksComparator = (s1, s2) -> Double.compare(s2.marks, s1.marks);
Collections.sort(students, marksComparator);

System.out.println("After sorting by marks in descending order");
students.forEach(System.out::println);
}
}
```

```
Before sorting:
Student{rollNo=1, name='Hari', marks=85.5}
Student{rollNo=2, name='Eswar', marks=90.0}
Student{rollNo=3, name='Kumar', marks=75.5}
After sorting by marks in descending order
Student{rollNo=2, name='Eswar', marks=90.0}
Student{rollNo=1, name='Hari', marks=85.5}
Student{rollNo=3, name='Kumar', marks=75.5}
```

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

```java
package Assignment_Day9;
```

```java
import java.util.*;

class Product {
String name;
double price;

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

@Override
public String toString() {
return "Product{name='" + name + "', price=" + price + "}";
}
}

public class Problem7 {
public static void main(String[] args) {
List<Product> products = new ArrayList<>();
products.add(new Product("Laptop", 999.99));
products.add(new Product("Phone", 699.99));
products.add(new Product("Tablet", 299.99));

Comparator<Product> priceAsc = Comparator.comparingDouble(p -> p.price);
Collections.sort(products, priceAsc);
System.out.println("Sorted by price ascending:");
products.forEach(System.out::println);

Comparator<Product> priceDesc = (p1, p2) -> Double.compare(p2.price, p1.price);
Collections.sort(products, priceDesc);
System.out.println("Sorted by price descending:");
products.forEach(System.out::println);

Comparator<Product> nameComparator = Comparator.comparing(p -> p.name);
Collections.sort(products, nameComparator);
System.out.println("Sort by name:");
products.forEach(System.out::println);
}
}
```

```
Sorted by price ascending:
Product{name='Tablet', price=299.99}
Product{name='Phone', price=699.99}
Product{name='Laptop', price=999.99}
Sorted by price descending:
Product{name='Laptop', price=999.99}
Product{name='Phone', price=699.99}
Product{name='Tablet', price=299.99}
Sort by name:
Product{name='Laptop', price=999.99}
Product{name='Phone', price=699.99}
Product{name='Tablet', price=299.99}
```

Q8. Sort Employee objects by joining date using Comparator.

Use Comparator to sort employees based on LocalDate or Date.

```java
package Assignment_Day9;

import java.time.*;
import java.util.*;

class Employee1 {
String name;
LocalDate joiningDate;

public Employee1(String name, LocalDate joiningDate) {
this.name = name;
this.joiningDate = joiningDate;
}

@Override
public String toString() {
return "Employee{name='" + name + "', joiningDate=" + joiningDate + "}";
}
```

```
}

public class Problem8 {
public static void main(String[] args) {
List<Employee1> employees = new ArrayList<>();
employees.add(new Employee1("Hari", LocalDate.of(2020, 5, 15)));
employees.add(new Employee1("Eswar", LocalDate.of(2019, 3, 10)));
employees.add(new Employee1("Kumar", LocalDate.of(2021, 1, 20)));

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

Comparator<Employee1> dateComparator = Comparator.comparing(e -> e.joiningDate);
Collections.sort(employees, dateComparator);

System.out.println("After sorting by joining date:");
employees.forEach(System.out::println);
}
}
```

```
Before sorting:
Employee{name='Hari', joiningDate=2020-05-15}
Employee{name='Eswar', joiningDate=2019-03-10}
Employee{name='Kumar', joiningDate=2021-01-20}
After sorting by joining date:
Employee{name='Eswar', joiningDate=2019-03-10}
Employee{name='Hari', joiningDate=2020-05-15}
Employee{name='Kumar', joiningDate=2021-01-20}
```

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

```
package Assignment_Day9;

import java.util.*;

class City {
String name;
int population;

public City(String name, int population) {
```

```java
this.name = name;
this.population = population;
}

@Override
public String toString() {
return "City{name='" + name + "', population=" + population + "}";
}
}

public class Problem9 {
public static void main(String[] args) {
List<City> cities = new ArrayList<>();
cities.add(new City("Guntur", 8419000));
cities.add(new City("Vijayawada", 13960000));
cities.add(new City("Hyderabad", 8982000));

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

Comparator<City> populationComparator = (c1, c2) -> Integer.compare(c2.population,
c1.population);
Collections.sort(cities, populationComparator);

System.out.println("After sorting by population in descending order:");
cities.forEach(System.out::println);
}
}
```

```
Before sorting:
City{name='Guntur', population=8419000}
City{name='Vijayawada', population=13960000}
City{name='Hyderabad', population=8982000}
After sorting by population in descending order:
City{name='Vijayawada', population=13960000}
City{name='Hyderabad', population=8982000}
City{name='Guntur', population=8419000}
```

Q10. Write a menu-driven program to sort Employee objects by name, salary, or department using Comparator.

```java
package Assignment_Day9;

import java.util.*;

class Employee2 {
String name;
double salary;
String department;

public Employee2(String name, double salary, String department) {
this.name = name;
this.salary = salary;
this.department = department;
}

@Override
public String toString() {
return "Employee{name='" + name + "', salary=" + salary + ", department='" +
department + "'}";
}
}

public class Problem10 {
public static void main(String[] args) {
List<Employee2> employees = new ArrayList<>();
employees.add(new Employee2("Hari", 60000, "HR"));
employees.add(new Employee2("Eswar", 70000, "IT"));
employees.add(new Employee2("Kumar", 50000, "Finance"));

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

switch (choice) {
case 1:
Collections.sort(employees, Comparator.comparing(e -> e.name));
break;
case 2:
Collections.sort(employees, Comparator.comparingDouble(e -> e.salary));
break;
case 3:
```

```
Collections.sort(employees, Comparator.comparing(e -> e.department));
break;
default:
System.out.println("Invalid choice!");
}

System.out.println("After sorting:");
employees.forEach(System.out::println);
}
}
```

```
Sort by:
1. Name
2. Salary
3. Department
Enter choice: 1
After sorting:
Employee{name='Eswar', salary=70000.0, department='IT'}
Employee{name='Hari', salary=60000.0, department='HR'}
Employee{name='Kumar', salary=50000.0, department='Finance'}
```

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.

```
package Assignment_Day9;

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

public class Problem11 {
public static void main(String[] args) {
String fileName = "student1234.txt";

try {
FileWriter writer = new FileWriter(fileName);
writer.write("Hari\n");
writer.write("Kmar\n");
writer.write("eswar\n");
writer.write("vinay\n");
writer.write("rao\n");
writer.close();
System.out.println("file created ");

System.out.println("Reading from file " + fileName);
BufferedReader reader = new BufferedReader(new FileReader(fileName));
String line;
while ((line = reader.readLine()) != null) {
System.out.println(line);
}
reader.close();

writer = new FileWriter(fileName, true);
```

```java
writer.write("krishna\n");
writer.close();
System.out.println("Data appended Successfully");


int lineCount = 0;
int wordCount = 0;
reader = new BufferedReader(new FileReader(fileName));
while ((line = reader.readLine()) != null) {
lineCount++;
String[] words = line.split("\\s+");
wordCount += words.length;
}
reader.close();
System.out.println("Number of lines: " + lineCount);
System.out.println("Number of words: " + wordCount);

} catch (Exception e) {
System.out.println(e);
}
}
}
```

```
file created
Reading from file  student1234.txt
Hari
Kmar
eswar
vinay
rao
Data appended Successfully
Number of lines: 6
Number of words: 6
```

Q5. Copy Contents from One File to Another

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

Q12. Delete a File

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

```
package File_Handling;


import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;

public class CopyFile {

public static void main(String[] args) throws IOException {
Path path=Paths.get("Sample121.txt");
Files.createFile(path);
System.out.println("File Created");

//copy file
Path copypath=Paths.get("Samplecopy1.txt");
Files.copy(path, copypath,StandardCopyOption.REPLACE_EXISTING);
System.out.println("File Copied");

//Move File
Path movedPath=Paths.get("movedfile1.txt");
Files.move(copypath, movedPath, StandardCopyOption.REPLACE_EXISTING);
System.out.println("File moved");

//Delete File
Files.deleteIfExists(movedPath);
Files.deleteIfExists(path);
```

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

}

}
```

```
File Created
File Copied
File moved
File deleted
```

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

```
package File_Handling;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
```

```
import java.util.List;

public class all_pract {

   public static void main(String[] args) throws IOException {
          File f=new File("sample1.txt");
          System.out.println(f.getName());
          System.out.println(f.length());
          System.out.println(f.getAbsolutePath());
          System.out.println(f.canRead());
          System.out.println(f.canWrite());

      try {
         BufferedWriter w = new BufferedWriter(new FileWriter("sample1.txt"));
         w.write("Hello");
         w.newLine();
         w.write("Welcome to Java Learning Sessions");
         w.close();
         System.out.println("\nWriting Completed");
      } catch (Exception e) {
         System.out.println(e);
      }


      try {
         List<String> l = new ArrayList<String>();
         BufferedReader r = new BufferedReader(new FileReader("sample1.txt"));
         String line;
         System.out.println("\nReading File");
         while ((line = r.readLine()) != null) {
            l.add(line);
         }
         for (String k : l) {
            System.out.println(k);
         }
         r.close();
      } catch (Exception e) {
         System.out.println(e);
```

```
        }


        try {
            PrintWriter p = new PrintWriter(new FileWriter("sample1.txt"));
            p.println("Hello world");
            p.print("Using PrintWriter");
            p.close();
            System.out.println("\nWriting Completed");
        } catch (Exception e) {
            System.out.println(e);
        }


        try {
            List<String> l = new ArrayList<String>();
            BufferedReader r = new BufferedReader(new FileReader("sample1.txt"));
            String line;
            System.out.println("\nReading file ");
            while ((line = r.readLine()) != null) {
                l.add(line);
            }
            for (String k : l) {
                System.out.println(k);
            }
            r.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

```
sample1.txt
30
C:\Users\user\Desktop\java1807\java_practice\sample1.txt
true
true

Writing Completed

Reading File
Hello
Welcome to Java Learning Sessions

Writing Completed

Reading file
Hello world
Using PrintWriter
```