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.

```
package Day9_Assessments;
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 s) {
	        return this.rollNo - s.rollNo;
	    }
	    public String toString() {
	        return rollNo + "  " + name + "  " + marks;
	    }
	}
	public class SortRollNoUsingComparator {
	    public static void main(String[] args) {
	        List<Student> students = new ArrayList<>();
	        students.add(new Student(102, "Natasha", 88.5));
	        students.add(new Student(101, "Manisha", 92.0));
	        students.add(new Student(105, "Ravi", 76.3));
	        students.add(new Student(103, "Arjun", 85.4));
	        Collections.sort(students);
	        System.out.println("Students sorted by Roll Number:");
	        for (Student s : students) {
	            System.out.println(s);
```

```
        }
    }
}
```

Output:

```
Students sorted by Roll Number:
101  Manisha  92.0
102  Natasha  88.5
103  Arjun  85.4
105  Ravi  76.3
```

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 Day9_Assessments;
import java.util.*;
class Product implements Comparable<Product> {
    private String name;
    private double price;
    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }
    public String getName() {
        return name;
    }
    public double getPrice() {
        return price;
    }
    public int compareTo(Product other) {
```

```
        return Double.compare(this.price, other.price); // ascending
order
    }
    public String toString() {
        return name + " - ₹" + price;
    }
}
public class ProductSort {
    public static void main(String[] args) {
        List<Product> products = new ArrayList<>();
        products.add(new Product("Laptop", 75000));
        products.add(new Product("Smartphone", 25000));
        products.add(new Product("Headphones", 3000));
        products.add(new Product("Smartwatch", 12000));
        System.out.println("Before sorting:");
        for (Product p : products) {
            System.out.println(p);
        }
        Collections.sort(products);
        System.out.println("\nAfter sorting by price (Ascending):");
        for (Product p : products) {
            System.out.println(p);
        }
    }
}
```

Output:

```
Before sorting:
Laptop - ₹75000.0
Smartphone - ₹25000.0
Headphones - ₹3000.0
Smartwatch - ₹12000.0

After sorting by price (Ascending):
Headphones - ₹3000.0
Smartwatch - ₹12000.0
Smartphone - ₹25000.0
```

Laptop - ₹75000.0

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

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

```
package Day9_Assessments;
import java.util.*;
class Employee implements Comparable<Employee> {
    private String name;
    private int id;
    public Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public int getId() {
        return id;
    }
    public int compareTo(Employee other) {
        return this.name.compareTo(other.name); // Alphabetical order
    }
    public String toString() {
        return id + " - " + name;
    }
}
public class EmployeeSorting {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Rahul", 103));
        employees.add(new Employee("Anita", 101));
```

```
        employees.add(new Employee("Vikram", 105));
        employees.add(new Employee("Meena", 102));
        System.out.println("Before sorting:");
        for (Employee e : employees) {
            System.out.println(e);
        }
        Collections.sort(employees);
        System.out.println("\nAfter sorting by name (Alphabetical):");
        for (Employee e : employees) {
            System.out.println(e);
        }
    }
}
```

Output:

```
Before sorting:
103 - Rahul
101 - Anita
105 - Vikram
102 - Meena

After sorting by name (Alphabetical):
101 - Anita
102 - Meena
103 - Rahul
105 - Vikram
```

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

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

```
package Day9_Assessments;
import java.util.*;
```

```
class Book implements Comparable<Book> {
    private int bookId;
    private String title;
    public Book(int bookId, String title) {
        this.bookId = bookId;
        this.title = title;
    }
    public int getBookId() {
        return bookId;
    }
    public String getTitle() {
        return title;
    }
    public int compareTo(Book other) {
        return Integer.compare(other.bookId, this.bookId); // Reverse
order
    }
    public String toString() {
        return "Book ID: " + bookId + ", Title: " + title;
    }
}
public class BookSorting{
    public static void main(String[] args) {
        List<Book> books = new ArrayList<>();
        books.add(new Book(105, "Java Programming"));
        books.add(new Book(102, "Data Structures"));
        books.add(new Book(110, "Algorithms"));
        books.add(new Book(101, "Operating Systems"));
        System.out.println("Before sorting:");
        for (Book b : books) {
            System.out.println(b);
        }
        Collections.sort(books);
        System.out.println("\nAfter sorting by bookId (Descending):");
        for (Book b : books) {
            System.out.println(b);
        }
```

```
    }
}
```

Output:

```
Before sorting:
Book ID: 105, Title: Java Programming
Book ID: 102, Title: Data Structures
Book ID: 110, Title: Algorithms
Book ID: 101, Title: Operating Systems

After sorting by bookId (Descending):
Book ID: 110, Title: Algorithms
Book ID: 105, Title: Java Programming
Book ID: 102, Title: Data Structures
Book ID: 101, Title: Operating Systems
```

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

```
package Day9_Assessments;
import java.util.*;
class Student implements Comparable<Student> {
    private String name;
    private int marks;
    public Student(String name, int marks) {
        this.name = name;
        this.marks = marks;
    }
    public String getName() {
        return name;
    }
    public int getMarks() {
        return marks;
    }
```

```
    public int compareTo(Student other) {
        return Integer.compare(this.marks, other.marks);
    }
    public String toString() {
        return name + " - Marks: " + marks;
    }
}
public class ComparableExample {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student("Ravi", 85));
        students.add(new Student("Anjali", 92));
        students.add(new Student("Vikram", 76));
        students.add(new Student("Meena", 88));
        System.out.println("Before Sorting:");
        for (Student s : students) {
            System.out.println(s);
        }
        Collections.sort(students);
        System.out.println("\nAfter Sorting by Marks (Ascending):");
        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

Output:

Ravi   - Marks: 85

Anjali - Marks: 92

Vikram - Marks: 76

Meena - Marks: 88

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 Day9_Assessments;
import java.util.*;
class Student {
    private String name;
    private int marks;
    public Student(String name, int marks) {
        this.name = name;
        this.marks = marks;
    }
    public String getName() {
        return name;
    }
    public int getMarks() {
        return marks;
    }
    public String toString() {
        return name + " - Marks: " + marks;
    }
}
class MarksDescendingComparator implements
Comparator<Student> {
    public int compare(Student s1, Student s2) {
        return Integer.compare(s2.getMarks(), s1.getMarks()); //
Descending
    }
}
public class ComparatorExample {
    public static void main(String[] args) {
        List<Student> students = new ArrayList<>();
        students.add(new Student("Ravi", 85));
        students.add(new Student("Anjali", 92));
        students.add(new Student("Vikram", 76));
        students.add(new Student("Meena", 88));
        System.out.println("Before Sorting:");
        for (Student s : students) {
```

```
            System.out.println(s);
        }
        Collections.sort(students, new MarksDescendingComparator());
        System.out.println("\nAfter Sorting by Marks (Descending):");
        for (Student s : students) {
            System.out.println(s);
        }
    }
}
```

Output:

Ravi - Marks: 85

Anjali - Marks: 92

Vikram - Marks: 76

Meena - Marks: 88

Q7. Create multiple sorting strategies for a Product class.

Implement comparators to sort by:

Price ascending

Price descending

Name alphabetically

```
package Day9_Assessments;
import java.util.*;
class Product {
 private String name;
 private double price;
 public Product(String name, double price) {
    this.name = name;
    this.price = price;
 }
 public String getName() {
    return name;
```

```
 }
 public double getPrice() {
    return price;
 }
 public String toString() {
    return name + " - ₹" + price;
 }
}
class PriceAscendingComparator implements Comparator<Product>
{
 public int compare(Product p1, Product p2) {
    return Double.compare(p1.getPrice(), p2.getPrice());
 }
}
class PriceDescendingComparator implements
Comparator<Product> {
 public int compare(Product p1, Product p2) {
    return Double.compare(p2.getPrice(), p1.getPrice());
 }
}
class NameComparator implements Comparator<Product> {
 public int compare(Product p1, Product p2) {
    return p1.getName().compareToIgnoreCase(p2.getName());
 }
}
public class MultipleSortingStrategies {
 public static void main(String[] args) {
    List<Product> products = new ArrayList<>();
    products.add(new Product("Laptop", 75000));
    products.add(new Product("Smartphone", 25000));
    products.add(new Product("Headphones", 3000));
    products.add(new Product("Smartwatch", 12000));
    System.out.println("Original List:");
    products.forEach(System.out::println);
    Collections.sort(products, new PriceAscendingComparator());
    System.out.println("\nSorted by Price (Ascending):");
    products.forEach(System.out::println);
```

```
        Collections.sort(products, new PriceDescendingComparator());
        System.out.println("\nSorted by Price (Descending):");
        products.forEach(System.out::println);
        Collections.sort(products, new NameComparator());
        System.out.println("\nSorted by Name (Alphabetical):");
        products.forEach(System.out::println);
    }
}
```

Output:

Laptop - ₹75000.0

Smartphone - ₹25000.0

Headphones - ₹3000.0

Smartwatch - ₹12000.0

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.*;

// Employee class
class Employee {
    private String name;
    private LocalDate joiningDate;

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

```
        this.joiningDate = joiningDate;
    }

    public String getName() {
        return name;
    }

    public LocalDate getJoiningDate() {
        return 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("Ravi", LocalDate.of(2022, 5,
10)));
        employees.add(new Employee("Anjali", LocalDate.of(2020, 8,
21)));
```

```
        employees.add(new Employee("Meena", LocalDate.of(2021, 1,
15)));
        employees.add(new Employee("Vikram", LocalDate.of(2023, 3,
5)));

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

        // Sort by joining date (oldest first)

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

        System.out.println("\nAfter Sorting by Joining Date (Oldest
First):");
        employees.forEach(System.out::println);

        // Sort by joining date (newest first)

employees.sort(Comparator.comparing(Employee::getJoiningDate).re
versed());

        System.out.println("\nAfter Sorting by Joining Date (Newest
First):");
        employees.forEach(System.out::println);
    }
}
```

Output:

Before Sorting:

Ravi - Joined: 2022-05-10

Anjali - Joined: 2020-08-21

Meena - Joined: 2021-01-15

Vikram - Joined: 2023-03-05

After Sorting by Joining Date (Oldest First):

Anjali - Joined: 2020-08-21

Meena - Joined: 2021-01-15

Ravi - Joined: 2022-05-10

Vikram - Joined: 2023-03-05

After Sorting by Joining Date (Newest First):

Vikram - Joined: 2023-03-05

Ravi - Joined: 2022-05-10

Meena - Joined: 2021-01-15

Anjali - Joined: 2020-08-21

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

```
package Day9_Assessments;
import java.util.*;
class City {
 private String name;
 private long population;
 public City(String name, long population) {
    this.name = name;
    this.population = population;
```

```
 }
 public String getName() {
    return name;
 }
 public long getPopulation() {
    return population;
 }
 public String toString() {
    return name + " - Population: " + population;
 }
}
public class CitySortByPopulation {
 public static void main(String[] args) {
    List<City> cities = new ArrayList<>();
    cities.add(new City("Mumbai", 20411000));
    cities.add(new City("Delhi", 16787941));
    cities.add(new City("Bangalore", 8443675));
    cities.add(new City("Hyderabad", 6809970));
    System.out.println("Before Sorting:");
    cities.forEach(System.out::println);
    cities.sort(Comparator.comparingLong(City::getPopulation));
    System.out.println("\nAfter Sorting by Population (Ascending):");
    cities.forEach(System.out::println);

cities.sort(Comparator.comparingLong(City::getPopulation).reversed(
));
    System.out.println("\nAfter Sorting by Population
(Descending):");
    cities.forEach(System.out::println);
 }
}
```

Output:

```
Before Sorting:
Mumbai - Population: 20411000
Delhi - Population: 16787941
Bangalore - Population: 8443675
```

```
Hyderabad - Population: 6809970

After Sorting by Population (Ascending):
Hyderabad - Population: 6809970
Bangalore - Population: 8443675
Delhi - Population: 16787941
Mumbai - Population: 20411000

After Sorting by Population (Descending):
Mumbai - Population: 20411000
Delhi - Population: 16787941
Bangalore - Population: 8443675
Hyderabad - Population: 6809970
```

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

```
package day9_Assessments;
import java.util.*;
public class SortStringsByLength {
    public static void main(String[] args) {
        List<String> strings = new ArrayList<>();
        strings.add("Banana");
        strings.add("Apple");
        strings.add("Strawberry");
        strings.add("Kiwi");
        strings.add("Mango");
        System.out.println("Before Sorting:");
        System.out.println(strings);
        Collections.sort(strings, new Comparator<String>() {
            public int compare(String s1, String s2) {
                return Integer.compare(s1.length(), s2.length());
            }
        });
        System.out.println("\nAfter Sorting by Length (Ascending):");
        System.out.println(strings);
    }
```

```
}
```

Output:

Before Sorting:
[Banana, Apple, Strawberry, Kiwi, Mango]

After Sorting by Length (Ascending):
[Kiwi, Apple, Mango, Banana, Strawberry]

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.

```
package day9_Assessments;
import java.util.*;
class Student implements Comparable<Student> {
 private String name;
 private int marks;
 public Student(String name, int marks) {
    this.name = name;
    this.marks = marks;
 }
 public String getName() {
    return name;
 }
 public int getMarks() {
    return marks;
 }
 public int compareTo(Student other) {
    return this.name.compareToIgnoreCase(other.name);
 }
```

```
 public String toString() {
    return name + " - Marks: " + marks;
 }
}
public class StudentSortingDemo {
 public static void main(String[] args) {
    List<Student> students = new ArrayList<>();
    students.add(new Student("Ravi", 85));
    students.add(new Student("Anjali", 92));
    students.add(new Student("Vikram", 76));
    students.add(new Student("Meena", 88));
    Collections.sort(students);
    System.out.println("Sorted by Name (Alphabetical):");
    students.forEach(System.out::println);
    students.sort(new Comparator<Student>() {
       public int compare(Student s1, Student s2) {
          return Integer.compare(s2.getMarks(), s1.getMarks()); //
Descending
       }
    });
    System.out.println("\nSorted by Marks (Descending):");
    students.forEach(System.out::println);
 }
}
```

Output:

Anjali - Marks: 92

Meena - Marks: 88

Ravi - Marks: 85

Vikram - Marks: 76

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

```java
package day9_Assessments;
import java.util.*;
class Book implements Comparable<Book> {
 private int bookId;
 private String title;
 private String author;

 public Book(int bookId, String title, String author) {
    this.bookId = bookId;
    this.title = title;
    this.author = author;
 }
 public int getBookId() {
    return bookId;
 }
 public String getTitle() {
    return title;
 }
 public String getAuthor() {
    return author;
 }
 public int compareTo(Book other) {
    return Integer.compare(this.bookId, other.bookId);
 }
 public String toString() {
    return "BookID: " + bookId + ", Title: " + title + ", Author: " +
author;
 }
}
public class BookSortingDemo {
 public static void main(String[] args) {
    List<Book> books = new ArrayList<>();
    books.add(new Book(103, "Java Programming", "John Smith"));
    books.add(new Book(101, "Data Structures", "Alice Johnson"));
    books.add(new Book(105, "Algorithms", "Bob Williams"));
    books.add(new Book(102, "Java Programming", "Charlie
Brown"));
```

```
        Collections.sort(books);
        System.out.println("Sorted by Book ID:");
        books.forEach(System.out::println);
        books.sort(Comparator
                .comparing(Book::getTitle)
                .thenComparing(Book::getAuthor));
        System.out.println("\nSorted by Title, then Author:");
        books.forEach(System.out::println);
    }
}
```

Output:

Sorted by Book ID

BookID: 101, Title: Data Structures, Author: Alice Johnson

BookID: 102, Title: Java Programming, Author: Charlie Brown

BookID: 103, Title: Java Programming, Author: John Smith

BookID: 105, Title: Algorithms, Author: Bob Williams

Sorted by Title, then Author:

BookID: 105, Title: Algorithms, Author: Bob Williams

BookID: 101, Title: Data Structures, Author: Alice Johnson

BookID: 102, Title: Java Programming, Author: Charlie Brown

BookID: 103, Title: Java Programming, Author: John Smith

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.

```
package day9_Assessments;
```

```
import java.io.FileWriter;
import java.io.IOException;
public class CreateAndWriteFile {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("student.txt");
            writer.write("Ravi Kumar\n");
            writer.write("Anjali Sharma\n");
            writer.write("Meena Patel\n");
            writer.write("Vikram Singh\n");
            writer.write("Priya Verma\n");
            writer.close();
            System.out.println("File created and student names written
successfully.");
        } catch (IOException e) {
            System.out.println("An error occurred while writing to the
file.");
            e.printStackTrace();
        }
    }
}
```

Output:

File created and student names 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.

```
package day9_Assessments;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class ReadFileExample {
    public static void main(String[] args) {
```

```
        try {
            BufferedReader reader = new BufferedReader(new
FileReader("student.txt"));
            String line;
            System.out.println("Contents of student.txt:");
                    while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();
        } catch (IOException e) {
            System.out.println("An error occurred while reading the
file.");
            e.printStackTrace();
        }
    }
}
```

Output:

Contents of student.txt:
Ravi Kumar
Anjali Sharma
Meena Patel
Vikram Singh
Priya Verma

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.

```
package day9_Assessments;
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);
            writer.write("Arjun Malhotra\n");
            writer.close();
            System.out.println("New student name appended
successfully.");
        } catch (IOException e) {
            System.out.println("An error occurred while appending to the
file.");
            e.printStackTrace();
        }
    }
}
```

Output:

New student name appended successfully.

Q4. Count Words and Lines

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

```
package day9_Assessments;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class CountWordsAndLines {
    public static void main(String[] args) {
        int lineCount = 0;
        int wordCount = 0;
        try {
            BufferedReader reader = new BufferedReader(new
FileReader("notes.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                lineCount++;
```

```
                String[] words = line.trim().split("\\s+");
                if (!line.trim().isEmpty()) {
                    wordCount += words.length;
                }
            }
            reader.close();
            System.out.println("Number of lines: " + lineCount);
            System.out.println("Number of words: " + wordCount);
        }
        catch (IOException e) {
            System.out.println("An error occurred while reading the
file.");
            e.printStackTrace();
        }
    }
}
```

Output:

Number of lines: 40

Number of words: 240

Q5. Copy Contents from One File to Another

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

```
package day9_Assessments;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
public class CopyFiles {
    public static void main(String[] args) {
        try {
```

```
            BufferedReader reader = new BufferedReader(new
FileReader("source.txt"));
            BufferedWriter writer = new BufferedWriter(new
FileWriter("destination.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                writer.write(line);
                writer.newLine();
            }
            reader.close();
            writer.close();
            System.out.println("File copied successfully.");
        }
        catch (IOException e) {
            System.out.println("An error occurred while copying the
file.");
            e.printStackTrace();
        }
    }
}
```

Output:

Enter a character:

5

5 is a digit.

Enter a character:

A

A is not a digit.

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 day9_Assessments;
import java.io.File;

public class FileProperties {
   public static void main(String[] args) {
      File file = new File("notes.txt");

      if (file.exists()) {
         System.out.println("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 in bytes: " + file.length());
      } else {
         System.out.println("The file 'report.txt' does not exist.");
      }
   }
}
```

Output:

File exists.

Absolute path: C:\Users\Admin\Documents\notes.txt

File name: report.txt

Writable: true

Readable: true

File size in bytes: 124