ChatGpt Analysis:

Here’s a detailed analysis of the provided solution for Exercise 7, with grades across **correctness**, **accuracy**, and **style**:

**1. Correctness (Grade: 9.5/10)**

* **Positive Points**:
  + The program implements two Comparator instances correctly:
    - One for alphabetical ascending order using String::compareTo.
    - Another for sorting strings by their length in descending order.
  + The stream API and the sorted method are used appropriately to perform sorting.
  + The program prints the results for the original list, alphabetically sorted list, and length-sorted list, meeting the exercise requirements.
* **Minor Issue**:
  + The strList is immutable after being wrapped in List.of(). While stream().sorted() ensures immutability for sorting purposes, this approach might fail if there’s a need for modifications. Wrapping it in new ArrayList<>(...) resolves this issue and makes the list mutable.

**2. Accuracy (Grade: 9/10)**

* **Positive Points**:
  + The alphabeticalAscendingComparator and lengthDescendingComparator logic are accurate and appropriate for the use case.
  + The printCollection method is simple and works for displaying collections effectively.
* **Suggestions for Improvement**:
  + **Reuse of Sorted List**:
    - Currently, sorting is done and directly printed using the forEach method without capturing the result in a new collection. This prevents reuse of the sorted list later in the program. Capturing the result in a variable could make the code more reusable:

java

CopyEdit

List<String> alphabeticalSortedList = strList.stream()

.sorted(alphabeticalAscendingComparator)

.toList();

alphabeticalSortedList.forEach(System.out::println);

* + **Edge Case Handling**:
    - The program doesn’t account for null values in the strList. Adding a null check in the comparators would improve robustness:

java

CopyEdit

Comparator<String> alphabeticalAscendingComparator = (str1, str2) -> {

if (str1 == null) return -1;

if (str2 == null) return 1;

return str1.compareTo(str2);

};

**3. Style (Grade: 8.5/10)**

* **Positive Points**:
  + The code is clear and readable, with meaningful variable and method names (printCollection, alphabeticalAscendingComparator, etc.).
  + The use of functional programming (Comparator, stream, and forEach) is clean and demonstrates a good grasp of modern Java.
* **Suggestions for Improvement**:
  + **Encapsulation of Sorting Logic**:
    - Extracting the sorting logic into methods can improve readability and reusability. For example:

java

CopyEdit

private static List<String> sortAlphabetically(List<String> list) {

return list.stream()

.sorted(String::compareTo)

.toList();

}

private static List<String> sortByLengthDescending(List<String> list) {

return list.stream()

.sorted((str1, str2) -> Integer.compare(str2.length(), str1.length()))

.toList();

}

* + - This would also make the main method cleaner.
  + **Improve Output Format**:
    - Add a more descriptive output for clarity. For example:

java

CopyEdit

System.out.println("Original List:\n" + strList);

System.out.println("Sorted Alphabetically (Ascending):\n" + alphabeticalSortedList);

System.out.println("Sorted By Length (Descending):\n" + lengthSortedList);

* + - This ensures the program output is structured and easy to follow.

**Execution Example (Sample Output)**

For the given input ["abc", "bcd", "def", "cde", "fjdksal;fjdsla;", "asdfdf"], the output will look like:

markdown

CopyEdit

------------

Original List:

abc

bcd

def

cde

fjdksal;fjdsla;

asdfdf

------------

Sorted Alphabetically Ascending List:

abc

asdfdf

bcd

cde

def

fjdksal;fjdsla;

------------

Sorted By Length Descending List:

fjdksal;fjdsla;

asdfdf

abc

bcd

def

cde

**Revised Solution with Improvements**

Here’s an improved version of the solution based on the analysis:

java

CopyEdit

import java.util.\*;

import java.util.stream.Collectors;

public class Main {

private static void printCollection(String title, Collection<?> col) {

System.out.println(title);

col.forEach(System.out::println);

}

private static List<String> sortAlphabetically(List<String> list) {

return list.stream().sorted(String::compareTo).toList();

}

private static List<String> sortByLengthDescending(List<String> list) {

return list.stream()

.sorted((str1, str2) -> Integer.compare(str2.length(), str1.length()))

.toList();

}

public static void main(String[] args) {

List<String> strList = new ArrayList<>(List.of("abc", "bcd", "def", "cde", "fjdksal;fjdsla;", "asdfdf"));

printCollection("Original List:", strList);

System.out.println("------------");

List<String> alphabeticalSortedList = sortAlphabetically(strList);

printCollection("Sorted Alphabetically (Ascending):", alphabeticalSortedList);

System.out.println("------------");

List<String> lengthSortedList = sortByLengthDescending(strList);

printCollection("Sorted By Length (Descending):", lengthSortedList);

}

}

**Overall Grades**

| **Aspect** | **Grade** | **Comments** |
| --- | --- | --- |
| **Correctness** | 9.5/10 | Correct and functional, though null handling is missing. |
| **Accuracy** | 9/10 | Logic is accurate; edge cases (like null values) can be considered. |
| **Style** | 8.5/10 | Clean but can benefit from modularization and better output formatting. |

Feel free to let me know if you want further refinements or exercises!