**Experiment Report - 05 – test1\_rankingData**

1. **Summary Table of Errors Found**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Error ID | Line Number | Error Type | CSCR? | Self-Detected? | LLM? |
| E01 | Line 23 | Semantic | √ |  | √ |
| E02 | Line 48 | Syntax | √ |  | √ |
| E03 | Line 72 | Semantic | √ |  | √ |
| E04 | Line 43 (88) | Semantic |  | × | × |
| E05 | Line 66 (111) | Semantic |  | √ | × |
| E06 | Line 67 (112) | Semantic |  | √ | × |

Additional Errors Found by Self: 0

CSCR Rate: 100%

Self-Review Detection Rate: 67%

LLM Rate: 50%

1. **Source Code**
2. import java.io.BufferedReader;
3. import java.io.BufferedWriter;
4. import java.io.FileReader;
5. import java.io.FileWriter;
6. import java.io.IOException;
7. import java.util.ArrayList;
8. import java.util.Collections;
9. import java.util.Comparator;
10. import java.util.List;
11. public class rankingData {
12. public void ExtractData(String inputFilePath, String outputFilePath, int num) {
13. try (BufferedReader br = new BufferedReader(new FileReader(inputFilePath));
14. BufferedWriter bw = new BufferedWriter(new FileWriter(outputFilePath))) {
15. String line;
16. while ((line = br.readLine()) != null) {
17. *// 各行を"/"で分割*
18. String[] data = line.split("/");
19. *// 3つ目と7つ目のデータを抽出*
20. if (data.length >= num) {
21. String thirdData = data[3];
22. String seventhData = data[num - 1];
23. *// 抽出したデータを"/"で結合*
24. String result = thirdData + "/" + seventhData;
25. *// 結果を新しいファイルに書き込む*
26. bw.write(result);
27. bw.newLine();
28. }
29. }
30. System.out.println("データの抽出と書き込みが完了しました。");
31. } catch (IOException e) {
32. e.printStackTrace();
33. }
34. }
35. public static Result sort(String inputFilePath) {
36. List<Person> persons = new ArrayList<>();
37. try (BufferedReader br = new BufferedReader(new FileReader(inputFilePath))) {
38. String line;
39. while ((line = br.readLine()) != null) {
40. String[] data = line.split("/");
41. if (data.length = 2) {
42. String name = data[0];
43. String grip = data[1];
44. persons.add(new Person(name, grip));
45. }
46. }
47. } catch (IOException e) {
48. e.printStackTrace();
49. }
50. *// 握力の強さが大きい順にソート*
51. Collections.sort(persons, new Comparator<Person>() {
52. @Override
53. public int compare(Person p1, Person p2) {
54. *// Compare grips as integers for sorting*
55. return Integer.compare(Integer.parseInt(p2.grip), Integer.parseInt(p1.grip)); *// 降順*
56. }
57. });
58. *// 上位3名を抽出*
59. int topN = Math.min(3, persons.size());
60. String[] names = new String[3];
61. String[] grips = new String[3];
62. for (int i = 0; i <= 3; i++) {
63. if (i < topN) {
64. names[i] = persons.get(i).name;
65. grips[i] = persons.get(i).grip;
66. } else {
67. names[i] = "-";
68. grips[i] = "-"; *// ここは適切な初期値（例えば0）を設定*
69. }
70. }
71. *// Result オブジェクトを返す*
72. return new Result(names, grips);
73. }
74. public static void main(String[] args) {
75. rankingData rankingData = new rankingData();
76. rankingData.ExtractData("input.txt", "output.txt", 7);
77. Result result = rankingData.sort("output.txt");
78. for (int i = 0; i < result.names.length; i++) {
79. System.out.println("Name: " + result.names[i] + ", Grip: " + result.grips[i]);
80. }
81. }
82. }
83. class Person {
84. public String name;
85. public String grip;
86. public Person(String name, String grip) {
87. this.name = name;
88. this.grip = grip;
89. }
90. }
91. class Result {
92. public String[] names;
93. public String[] grips;
94. public Result(String[] names, String[] grips) {
95. this.names = names;
96. this.grips = grips;
97. }
98. }