Alexander Keith

CS 152 section 1

MW 3:00

1. public class Student {
2. private String lastName;
3. private double gpa;
4. private int age;
5. public Student(String lastName, double gpa, int age) {
6. this.lastName = lastName;
7. this.gpa = gpa;
8. this.age = age;
9. }
10. public double getGPA() {
11. return gpa;
12. }
13. public String toString() {
14. return lastName + "\t" + gpa + "\t" + age;
15. }
16. int compareTo(Student s) {
17. if (gpa < s.getGPA()) {
18. return -1;
19. } else if (gpa == s.getGPA()) {
20. return 0;
21. } else {
22. return 1;
23. }
24. }
25. }
26. public class LinkedList {
27. private Node list;
28. public LinkedList() {
29. list = new Node(null);
30. }
31. public void addFront(Student s) {
32. if (list == null) {
33. list = new Node(s);
34. } else {
35. Node temp = list;
36. list = new Node(s);
37. list.next = temp;
38. }
39. }
40. public void addTail(Student s) {
41. if (list == null) {
42. list = new Node(s);
43. } else {
44. Node temp = new Node(s);
45. Node currentNode = list;
46. while (currentNode.next != null) {
47. currentNode = currentNode.next;
48. }
49. currentNode.next = temp;
50. }
51. }
52. public Student bestStudent() {
53. if (list.next != null) {
54. Node currentBest = list;
55. Node currentNode = list;
56. while (currentNode.next != null) {
57. currentNode = currentNode.next;
58. if (currentBest.data.getGPA() < currentNode.data.getGPA()) {
59. currentBest = currentNode;
60. }
61. }
62. return currentBest.data;
63. } else {
64. return list.data;
65. }
66. }
67. public void printLinkedList() {
68. Node currentNode = list;
69. System.out.println(currentNode.data.toString());
70. while (currentNode.next != null) {
71. currentNode = currentNode.next;
72. System.out.println(currentNode.data.toString());
73. }
74. }
75. public int countNodesRec(Node list) {
76. if (list.next != null) {
77. return 1 + countNodesRec(list.next);
78. } else {
79. return 1;
80. }
81. }
82. public Student worstStudentRec(Node list) {
83. if (list.next != null) {
84. if (list.data.getGPA() < worstStudentRec(list.next).getGPA()) {
85. return list.data;
86. } else {
87. return worstStudentRec(list.next);
88. }
89. } else {
90. return list.data;
91. }
92. }
93. public Student findWorstStudent() {
94. return worstStudentRec(list);
95. }
96. public int findNodeCount() {
97. return countNodesRec(list);
98. }
99. public void addInOrder(Student s) {
100. if (list == null) {
101. list = new Node(s);
102. } else {
103. if (s.compareTo(list.data) == -1) {
104. Node temp = list;
105. list = new Node(s);
106. list.next = temp;
107. } else {
108. if (list.next == null) {
109. list.next = new Node(s);
110. } else {
111. Node prev = list;
112. Node currentNode = list.next;
113. Node temp;
114. if (s.compareTo(currentNode.data) == -1) {
115. list.next = new Node(s);
116. temp = currentNode;
117. currentNode = list.next;
118. currentNode.next = temp;
119. } else {
120. if (currentNode.next == null) {
121. currentNode.next = new Node(s);
122. } else {
123. prev = currentNode;
124. currentNode = currentNode.next;
125. if (s.compareTo(currentNode.data) == -1) {
126. prev.next = new Node(s);
127. temp = currentNode;
128. currentNode = prev.next;
129. currentNode.next = temp;
130. } else {
131. if (currentNode.next == null) {
132. currentNode.next = new Node(s);
133. } else {
134. prev = currentNode;
135. currentNode = currentNode.next;
136. if (s.compareTo(currentNode.data) == -1) {
137. prev.next = new Node(s);
138. temp = currentNode;
139. currentNode = prev.next;
140. currentNode.next = temp;
141. } else {
142. if (currentNode.next == null) {
143. currentNode.next = new Node(s);
144. } else {
145. prev = currentNode;
146. currentNode = currentNode.next;
147. if (s.compareTo(currentNode.data) == -1) {
148. prev.next = new Node(s);
149. temp = currentNode;
150. currentNode = prev.next;
151. currentNode.next = temp;
152. } else {
153. if (currentNode.next == null) {
154. currentNode.next = new Node(s);
155. } else {
156. prev = currentNode;
157. currentNode = currentNode.next;
158. if (s.compareTo(currentNode.data) == -1) {
159. prev.next = new Node(s);
160. temp = currentNode;
161. currentNode = prev.next;
162. currentNode.next = temp;
163. } else {
164. if (currentNode.next == null) {
165. currentNode.next = new Node(s);
166. } else {
167. prev = currentNode;
168. currentNode = currentNode.next;
169. if (s.compareTo(currentNode.data) == -1) {
170. prev.next = new Node(s);
171. temp = currentNode;
172. currentNode = prev.next;
173. currentNode.next = temp;
174. } else {
175. if (currentNode.next == null) {
176. currentNode.next = new Node(s);
177. } else {
178. prev = currentNode;
179. currentNode = currentNode.next;
180. if (s.compareTo(currentNode.data) == -1) {
181. prev.next = new Node(s);
182. temp = currentNode;
183. currentNode = prev.next;
184. currentNode.next = temp;
185. }
186. }
187. }
188. }
189. }
190. }
191. }
192. }
193. }
194. }
195. }
196. }
197. }
198. }
199. }
200. }
201. }
202. private class Node {
203. Student data;
204. Node next;
205. public Node(Student s) {
206. data = s;
207. next = null;
208. }
209. }
210. }
211. public class TestList {
212. public static void main(String[] args) {
213. Student s1 = new Student("Adams", 3.9, 26);
214. Student s2 = new Student("Lewis", 2.1, 29);
215. Student s3 = new Student("Lopez", 4.0, 53);
216. Student s4 = new Student("Smith", 3.2, 22);
217. Student s5 = new Student("Zeeler", 3.6, 38);
218. LinkedList list1 = new LinkedList();
219. list1.addFront(s1);
220. list1.addFront(s2);
221. list1.addFront(s3);
222. list1.addFront(s4);
223. list1.addFront(s5);
224. System.out.println("List 1:");
225. list1.printLinkedList();
226. System.out.println("\nBest Student:");
227. Student l1b = list1.bestStudent();
228. System.out.println(l1b.toString());
229. System.out.println("\nWorst Student:");
230. Student l1w = list1.findWorstStudent();
231. System.out.println(l1w.toString());
232. System.out.println("\nNumber of Nodes: " + list1.findNodeCount());
233. LinkedList list2 = new LinkedList();
234. list2.addTail(s1);
235. list2.addTail(s2);
236. list2.addTail(s3);
237. list2.addTail(s4);
238. list2.addTail(s5);
239. System.out.println("\nList 2:");
240. list2.printLinkedList();
241. System.out.println("\nBest Student:");
242. Student l2b = list2.bestStudent();
243. System.out.println(l2b.toString());
244. System.out.println("\nWorst Student:");
245. Student l2w = list2.findWorstStudent();
246. System.out.println(l2w.toString());
247. System.out.println("\nNumber of Nodes: " + list2.findNodeCount());
248. LinkedList list3 = new LinkedList();
249. list3.addInOrder(s1);
250. list3.addInOrder(s2);
251. list3.addInOrder(s3);
252. list3.addInOrder(s4);
253. list3.addInOrder(s5);
254. System.out.println("\nList 3:");
255. list3.printLinkedList();
256. System.out.println("\nBest Student:");
257. Student l3b = list3.bestStudent();
258. System.out.println(l3b.toString());
259. System.out.println("\nWorst Student:");
260. Student l3w = list3.findWorstStudent();
261. System.out.println(l3w.toString());
262. System.out.println("\nNumber of Nodes: " + list3.findNodeCount());
263. }
264. }

|  |
| --- |
| LinkedList |
| - List : Node |
| + LinkedList : void  + addFront(s : Student) : void  + addTail(s : Student) : void  + bestStudent() : Student  + printLinkedList() : void  + countNodesRec(list : Node) : int  + worstStudentRec(list : Node) : Student  + findWorstStudent() : Student  + findNodeCount() : int  + addInOrder(s : Student) : void |

|  |
| --- |
| TestList |
|  |
| + main(args : String[]) : void |

|  |
| --- |
| Student |
| - lastName : String  - gpa : double  - age : int |
| + Student(lastName : String, gpa: double, age : int) : void  + getGPA() : double  + toString() : String  + compareTo : int |

|  |
| --- |
| Node |
| + data : Student  + next : Node |
| + Node(s : Student) : void |

