Java Program to Implement Doubly Linked List

*/\**

1. *\* Java Program to Implement Doubly Linked List*
2. *\*/*
4. **import** java.util.Scanner;
6. */\* Class Node \*/*
7. **class** Node
8. {
9. **protected** **int** data;
10. **protected** Node next, prev;
12. */\* Constructor \*/*
13. **public** Node()
14. {
15. next = **null**;
16. prev = **null**;
17. data = 0;
18. }
19. */\* Constructor \*/*
20. **public** Node(**int** d, Node n, Node p)
21. {
22. data = d;
23. next = n;
24. prev = p;
25. }
26. */\* Function to set link to next node \*/*
27. **public** **void** setLinkNext(Node n)
28. {
29. next = n;
30. }
31. */\* Function to set link to previous node \*/*
32. **public** **void** setLinkPrev(Node p)
33. {
34. prev = p;
35. }
36. */\* Funtion to get link to next node \*/*
37. **public** Node getLinkNext()
38. {
39. **return** next;
40. }
41. */\* Function to get link to previous node \*/*
42. **public** Node getLinkPrev()
43. {
44. **return** prev;
45. }
46. */\* Function to set data to node \*/*
47. **public** **void** setData(**int** d)
48. {
49. data = d;
50. }
51. */\* Function to get data from node \*/*
52. **public** **int** getData()
53. {
54. **return** data;
55. }
56. }
58. */\* Class linkedList \*/*
59. **class** linkedList
60. {
61. **protected** Node start;
62. **protected** Node end ;
63. **public** **int** size;
65. */\* Constructor \*/*
66. **public** linkedList()
67. {
68. start = **null**;
69. end = **null**;
70. size = 0;
71. }
72. */\* Function to check if list is empty \*/*
73. **public** **boolean** isEmpty()
74. {
75. **return** start == **null**;
76. }
77. */\* Function to get size of list \*/*
78. **public** **int** getSize()
79. {
80. **return** size;
81. }
82. */\* Function to insert element at begining \*/*
83. **public** **void** insertAtStart(**int** val)
84. {
85. Node nptr = **new** Node(val, **null**, **null**);
86. **if**(start == **null**)
87. {
88. start = nptr;
89. end = start;
90. }
91. **else**
92. {
93. start.setLinkPrev(nptr);
94. nptr.setLinkNext(start);
95. start = nptr;
96. }
97. size++;
98. }
99. */\* Function to insert element at end \*/*
100. **public** **void** insertAtEnd(**int** val)
101. {
102. Node nptr = **new** Node(val, **null**, **null**);
103. **if**(start == **null**)
104. {
105. start = nptr;
106. end = start;
107. }
108. **else**
109. {
110. nptr.setLinkPrev(end);
111. end.setLinkNext(nptr);
112. end = nptr;
113. }
114. size++;
115. }
116. */\* Function to insert element at position \*/*
117. **public** **void** insertAtPos(**int** val , **int** pos)
118. {
119. Node nptr = **new** Node(val, **null**, **null**);
120. **if** (pos == 1)
121. {
122. insertAtStart(val);
123. **return**;
124. }
125. Node ptr = start;
126. **for** (**int** i = 2; i <= size; i++)
127. {
128. **if** (i == pos)
129. {
130. Node tmp = ptr.getLinkNext();
131. ptr.setLinkNext(nptr);
132. nptr.setLinkPrev(ptr);
133. nptr.setLinkNext(tmp);
134. tmp.setLinkPrev(nptr);
135. }
136. ptr = ptr.getLinkNext();
137. }
138. size++ ;
139. }
140. */\* Function to delete node at position \*/*
141. **public** **void** deleteAtPos(**int** pos)
142. {
143. **if** (pos == 1)
144. {
145. **if** (size == 1)
146. {
147. start = **null**;
148. end = **null**;
149. size = 0;
150. **return**;
151. }
152. start = start.getLinkNext();
153. start.setLinkPrev(**null**);
154. size--;
155. **return** ;
156. }
157. **if** (pos == size)
158. {
159. end = end.getLinkPrev();
160. end.setLinkNext(**null**);
161. size-- ;
162. }
163. Node ptr = start.getLinkNext();
164. **for** (**int** i = 2; i <= size; i++)
165. {
166. **if** (i == pos)
167. {
168. Node p = ptr.getLinkPrev();
169. Node n = ptr.getLinkNext();
171. p.setLinkNext(n);
172. n.setLinkPrev(p);
173. size-- ;
174. **return**;
175. }
176. ptr = ptr.getLinkNext();
177. }
178. }
179. */\* Function to display status of list \*/*
180. **public** **void** display()
181. {
182. System.out.print("**\n**Doubly Linked List = ");
183. **if** (size == 0)
184. {
185. System.out.print("empty**\n**");
186. **return**;
187. }
188. **if** (start.getLinkNext() == **null**)
189. {
190. System.out.println(start.getData() );
191. **return**;
192. }
193. Node ptr = start;
194. System.out.print(start.getData()+ " <-> ");
195. ptr = start.getLinkNext();
196. **while** (ptr.getLinkNext() != **null**)
197. {
198. System.out.print(ptr.getData()+ " <-> ");
199. ptr = ptr.getLinkNext();
200. }
201. System.out.print(ptr.getData()+ "**\n**");
202. }
203. }
205. */\* Class DoublyLinkedList \*/*
206. **public** **class** DoublyLinkedList
207. {
208. **public** **static** **void** main(String[] args)
209. {
210. Scanner scan = **new** Scanner(System.in);
211. */\* Creating object of linkedList \*/*
212. linkedList list = **new** linkedList();
213. System.out.println("Doubly Linked List Test**\n**");
214. **char** ch;
215. */\* Perform list operations \*/*
216. **do**
217. {
218. System.out.println("**\n**Doubly Linked List Operations**\n**");
219. System.out.println("1. insert at begining");
220. System.out.println("2. insert at end");
221. System.out.println("3. insert at position");
222. System.out.println("4. delete at position");
223. System.out.println("5. check empty");
224. System.out.println("6. get size");
226. **int** choice = scan.nextInt();
227. **switch** (choice)
228. {
229. **case** 1 :
230. System.out.println("Enter integer element to insert");
231. list.insertAtStart( scan.nextInt() );
232. **break**;
233. **case** 2 :
234. System.out.println("Enter integer element to insert");
235. list.insertAtEnd( scan.nextInt() );
236. **break**;
237. **case** 3 :
238. System.out.println("Enter integer element to insert");
239. **int** num = scan.nextInt() ;
240. System.out.println("Enter position");
241. **int** pos = scan.nextInt() ;
242. **if** (pos < 1 || pos > list.getSize() )
243. System.out.println("Invalid position**\n**");
244. **else**
245. list.insertAtPos(num, pos);
246. **break**;
247. **case** 4 :
248. System.out.println("Enter position");
249. **int** p = scan.nextInt() ;
250. **if** (p < 1 || p > list.getSize() )
251. System.out.println("Invalid position**\n**");
252. **else**
253. list.deleteAtPos(p);
254. **break**;
255. **case** 5 :
256. System.out.println("Empty status = "+ list.isEmpty());
257. **break**;
258. **case** 6 :
259. System.out.println("Size = "+ list.getSize() +" **\n**");
260. **break**;
261. **default** :
262. System.out.println("Wrong Entry **\n** ");
263. **break**;
264. }
265. */\* Display List \*/*
266. list.display();
267. System.out.println("**\n**Do you want to continue (Type y or n) **\n**");
268. ch = scan.next().charAt(0);
270. } **while** (ch == 'Y'|| ch == 'y');
271. }
272. }

# Java Program to Implement Singly Linked List

1. */\**
2. *\* Java Program to Implement Singly Linked List*
3. *\*/*
5. **import** java.util.Scanner;
7. */\* Class Node \*/*
8. **class** Node
9. {
10. **protected** **int** data;
11. **protected** Node link;
13. */\* Constructor \*/*
14. **public** Node()
15. {
16. link = **null**;
17. data = 0;
18. }
19. */\* Constructor \*/*
20. **public** Node(**int** d,Node n)
21. {
22. data = d;
23. link = n;
24. }
25. */\* Function to set link to next Node \*/*
26. **public** **void** setLink(Node n)
27. {
28. link = n;
29. }
30. */\* Function to set data to current Node \*/*
31. **public** **void** setData(**int** d)
32. {
33. data = d;
34. }
35. */\* Function to get link to next node \*/*
36. **public** Node getLink()
37. {
38. **return** link;
39. }
40. */\* Function to get data from current Node \*/*
41. **public** **int** getData()
42. {
43. **return** data;
44. }
45. }
47. */\* Class linkedList \*/*
48. **class** linkedList
49. {
50. **protected** Node start;
51. **protected** Node end ;
52. **public** **int** size ;
54. */\* Constructor \*/*
55. **public** linkedList()
56. {
57. start = **null**;
58. end = **null**;
59. size = 0;
60. }
61. */\* Function to check if list is empty \*/*
62. **public** **boolean** isEmpty()
63. {
64. **return** start == **null**;
65. }
66. */\* Function to get size of list \*/*
67. **public** **int** getSize()
68. {
69. **return** size;
70. }
71. */\* Function to insert an element at begining \*/*
72. **public** **void** insertAtStart(**int** val)
73. {
74. Node nptr = **new** Node(val, **null**);
75. size++ ;
76. **if**(start == **null**)
77. {
78. start = nptr;
79. end = start;
80. }
81. **else**
82. {
83. nptr.setLink(start);
84. start = nptr;
85. }
86. }
87. */\* Function to insert an element at end \*/*
88. **public** **void** insertAtEnd(**int** val)
89. {
90. Node nptr = **new** Node(val,**null**);
91. size++ ;
92. **if**(start == **null**)
93. {
94. start = nptr;
95. end = start;
96. }
97. **else**
98. {
99. end.setLink(nptr);
100. end = nptr;
101. }
102. }
103. */\* Function to insert an element at position \*/*
104. **public** **void** insertAtPos(**int** val , **int** pos)
105. {
106. Node nptr = **new** Node(val, **null**);
107. Node ptr = start;
108. pos = pos - 1 ;
109. **for** (**int** i = 1; i < size; i++)
110. {
111. **if** (i == pos)
112. {
113. Node tmp = ptr.getLink() ;
114. ptr.setLink(nptr);
115. nptr.setLink(tmp);
116. **break**;
117. }
118. ptr = ptr.getLink();
119. }
120. size++ ;
121. }
122. */\* Function to delete an element at position \*/*
123. **public** **void** deleteAtPos(**int** pos)
124. {
125. **if** (pos == 1)
126. {
127. start = start.getLink();
128. size--;
129. **return** ;
130. }
131. **if** (pos == size)
132. {
133. Node s = start;
134. Node t = start;
135. **while** (s != end)
136. {
137. t = s;
138. s = s.getLink();
139. }
140. end = t;
141. end.setLink(**null**);
142. size --;
143. **return**;
144. }
145. Node ptr = start;
146. pos = pos - 1 ;
147. **for** (**int** i = 1; i < size - 1; i++)
148. {
149. **if** (i == pos)
150. {
151. Node tmp = ptr.getLink();
152. tmp = tmp.getLink();
153. ptr.setLink(tmp);
154. **break**;
155. }
156. ptr = ptr.getLink();
157. }
158. size-- ;
159. }
160. */\* Function to display elements \*/*
161. **public** **void** display()
162. {
163. System.out.print("**\n**Singly Linked List = ");
164. **if** (size == 0)
165. {
166. System.out.print("empty**\n**");
167. **return**;
168. }
169. **if** (start.getLink() == **null**)
170. {
171. System.out.println(start.getData() );
172. **return**;
173. }
174. Node ptr = start;
175. System.out.print(start.getData()+ "->");
176. ptr = start.getLink();
177. **while** (ptr.getLink() != **null**)
178. {
179. System.out.print(ptr.getData()+ "->");
180. ptr = ptr.getLink();
181. }
182. System.out.print(ptr.getData()+ "**\n**");
183. }
184. }
186. */\* Class SinglyLinkedList \*/*
187. **public** **class** SinglyLinkedList
188. {
189. **public** **static** **void** main(String[] args)
190. {
191. Scanner scan = **new** Scanner(System.in);
192. */\* Creating object of class linkedList \*/*
193. linkedList list = **new** linkedList();
194. System.out.println("Singly Linked List Test**\n**");
195. **char** ch;
196. */\* Perform list operations \*/*
197. **do**
198. {
199. System.out.println("**\n**Singly Linked List Operations**\n**");
200. System.out.println("1. insert at begining");
201. System.out.println("2. insert at end");
202. System.out.println("3. insert at position");
203. System.out.println("4. delete at position");
204. System.out.println("5. check empty");
205. System.out.println("6. get size");
206. **int** choice = scan.nextInt();
207. **switch** (choice)
208. {
209. **case** 1 :
210. System.out.println("Enter integer element to insert");
211. list.insertAtStart( scan.nextInt() );
212. **break**;
213. **case** 2 :
214. System.out.println("Enter integer element to insert");
215. list.insertAtEnd( scan.nextInt() );
216. **break**;
217. **case** 3 :
218. System.out.println("Enter integer element to insert");
219. **int** num = scan.nextInt() ;
220. System.out.println("Enter position");
221. **int** pos = scan.nextInt() ;
222. **if** (pos <= 1 || pos > list.getSize() )
223. System.out.println("Invalid position**\n**");
224. **else**
225. list.insertAtPos(num, pos);
226. **break**;
227. **case** 4 :
228. System.out.println("Enter position");
229. **int** p = scan.nextInt() ;
230. **if** (p < 1 || p > list.getSize() )
231. System.out.println("Invalid position**\n**");
232. **else**
233. list.deleteAtPos(p);
234. **break**;
235. **case** 5 :
236. System.out.println("Empty status = "+ list.isEmpty());
237. **break**;
238. **case** 6 :
239. System.out.println("Size = "+ list.getSize() +" **\n**");
240. **break**;
241. **default** :
242. System.out.println("Wrong Entry **\n** ");
243. **break**;
244. }
245. */\* Display List \*/*
246. list.display();
247. System.out.println("**\n**Do you want to continue (Type y or n) **\n**");
248. ch = scan.next().charAt(0);
249. } **while** (ch == 'Y'|| ch == 'y');
250. }
251. }

# Java Program to Implement Circular Doubly Linked List

1. */\**
2. *\* Java Program to Implement Circular Doubly Linked List*
3. *\*/*
5. **import** java.util.Scanner;
7. */\* Class Node \*/*
8. **class** Node
9. {
10. **protected** **int** data;
11. **protected** Node next, prev;
13. */\* Constructor \*/*
14. **public** Node()
15. {
16. next = **null**;
17. prev = **null**;
18. data = 0;
19. }
20. */\* Constructor \*/*
21. **public** Node(**int** d, Node n, Node p)
22. {
23. data = d;
24. next = n;
25. prev = p;
26. }
27. */\* Function to set link to next node \*/*
28. **public** **void** setLinkNext(Node n)
29. {
30. next = n;
31. }
32. */\* Function to set link to previous node \*/*
33. **public** **void** setLinkPrev(Node p)
34. {
35. prev = p;
36. }
37. */\* Funtion to get link to next node \*/*
38. **public** Node getLinkNext()
39. {
40. **return** next;
41. }
42. */\* Function to get link to previous node \*/*
43. **public** Node getLinkPrev()
44. {
45. **return** prev;
46. }
47. */\* Function to set data to node \*/*
48. **public** **void** setData(**int** d)
49. {
50. data = d;
51. }
52. */\* Function to get data from node \*/*
53. **public** **int** getData()
54. {
55. **return** data;
56. }
57. }
59. */\* Class linkedList \*/*
60. **class** linkedList
61. {
62. **protected** Node start;
63. **protected** Node end ;
64. **public** **int** size;
66. */\* Constructor \*/*
67. **public** linkedList()
68. {
69. start = **null**;
70. end = **null**;
71. size = 0;
72. }
73. */\* Function to check if list is empty \*/*
74. **public** **boolean** isEmpty()
75. {
76. **return** start == **null**;
77. }
78. */\* Function to get size of list \*/*
79. **public** **int** getSize()
80. {
81. **return** size;
82. }
83. */\* Function to insert element at begining \*/*
84. **public** **void** insertAtStart(**int** val)
85. {
86. Node nptr = **new** Node(val, **null**, **null**);
87. **if** (start == **null**)
88. {
89. nptr.setLinkNext(nptr);
90. nptr.setLinkPrev(nptr);
91. start = nptr;
92. end = start;
93. }
94. **else**
95. {
96. nptr.setLinkPrev(end);
97. end.setLinkNext(nptr);
98. start.setLinkPrev(nptr);
99. nptr.setLinkNext(start);
100. start = nptr;
101. }
102. size++ ;
103. }
104. */\*Function to insert element at end \*/*
105. **public** **void** insertAtEnd(**int** val)
106. {
107. Node nptr = **new** Node(val, **null**, **null**);
108. **if** (start == **null**)
109. {
110. nptr.setLinkNext(nptr);
111. nptr.setLinkPrev(nptr);
112. start = nptr;
113. end = start;
114. }
115. **else**
116. {
117. nptr.setLinkPrev(end);
118. end.setLinkNext(nptr);
119. start.setLinkPrev(nptr);
120. nptr.setLinkNext(start);
121. end = nptr;
122. }
123. size++;
124. }
125. */\* Function to insert element at position \*/*
126. **public** **void** insertAtPos(**int** val , **int** pos)
127. {
128. Node nptr = **new** Node(val, **null**, **null**);
129. **if** (pos == 1)
130. {
131. insertAtStart(val);
132. **return**;
133. }
134. Node ptr = start;
135. **for** (**int** i = 2; i <= size; i++)
136. {
137. **if** (i == pos)
138. {
139. Node tmp = ptr.getLinkNext();
140. ptr.setLinkNext(nptr);
141. nptr.setLinkPrev(ptr);
142. nptr.setLinkNext(tmp);
143. tmp.setLinkPrev(nptr);
144. }
145. ptr = ptr.getLinkNext();
146. }
147. size++ ;
148. }
149. */\* Function to delete node at position \*/*
150. **public** **void** deleteAtPos(**int** pos)
151. {
152. **if** (pos == 1)
153. {
154. **if** (size == 1)
155. {
156. start = **null**;
157. end = **null**;
158. size = 0;
159. **return**;
160. }
161. start = start.getLinkNext();
162. start.setLinkPrev(end);
163. end.setLinkNext(start);
164. size--;
165. **return** ;
166. }
167. **if** (pos == size)
168. {
169. end = end.getLinkPrev();
170. end.setLinkNext(start);
171. start.setLinkPrev(end);
172. size-- ;
173. }
174. Node ptr = start.getLinkNext();
175. **for** (**int** i = 2; i <= size; i++)
176. {
177. **if** (i == pos)
178. {
179. Node p = ptr.getLinkPrev();
180. Node n = ptr.getLinkNext();
182. p.setLinkNext(n);
183. n.setLinkPrev(p);
184. size-- ;
185. **return**;
186. }
187. ptr = ptr.getLinkNext();
188. }
189. }
190. */\* Function to display status of list \*/*
191. **public** **void** display()
192. {
193. System.out.print("**\n**Circular Doubly Linked List = ");
194. Node ptr = start;
195. **if** (size == 0)
196. {
197. System.out.print("empty**\n**");
198. **return**;
199. }
200. **if** (start.getLinkNext() == start)
201. {
202. System.out.print(start.getData()+ " <-> "+ptr.getData()+ "**\n**");
203. **return**;
204. }
205. System.out.print(start.getData()+ " <-> ");
206. ptr = start.getLinkNext();
207. **while** (ptr.getLinkNext() != start)
208. {
209. System.out.print(ptr.getData()+ " <-> ");
210. ptr = ptr.getLinkNext();
211. }
212. System.out.print(ptr.getData()+ " <-> ");
213. ptr = ptr.getLinkNext();
214. System.out.print(ptr.getData()+ "**\n**");
215. }
216. }
218. */\* Class CircularDoublyLinkedList \*/*
219. **public** **class** CircularDoublyLinkedList
220. {
221. **public** **static** **void** main(String[] args)
222. {
223. Scanner scan = **new** Scanner(System.in);
224. */\* Creating object of linkedList \*/*
225. linkedList list = **new** linkedList();
226. System.out.println("Circular Doubly Linked List Test**\n**");
227. **char** ch;
228. */\* Perform list operations \*/*
229. **do**
230. {
231. System.out.println("**\n**Circular Doubly Linked List Operations**\n**");
232. System.out.println("1. insert at begining");
233. System.out.println("2. insert at end");
234. System.out.println("3. insert at position");
235. System.out.println("4. delete at position");
236. System.out.println("5. check empty");
237. System.out.println("6. get size");
239. **int** choice = scan.nextInt();
240. **switch** (choice)
241. {
242. **case** 1 :
243. System.out.println("Enter integer element to insert");
244. list.insertAtStart( scan.nextInt() );
245. **break**;
246. **case** 2 :
247. System.out.println("Enter integer element to insert");
248. list.insertAtEnd( scan.nextInt() );
249. **break**;
250. **case** 3 :
251. System.out.println("Enter integer element to insert");
252. **int** num = scan.nextInt() ;
253. System.out.println("Enter position");
254. **int** pos = scan.nextInt() ;
255. **if** (pos < 1 || pos > list.getSize() )
256. System.out.println("Invalid position**\n**");
257. **else**
258. list.insertAtPos(num, pos);
259. **break**;
260. **case** 4 :
261. System.out.println("Enter position");
262. **int** p = scan.nextInt() ;
263. **if** (p < 1 || p > list.getSize() )
264. System.out.println("Invalid position**\n**");
265. **else**
266. list.deleteAtPos(p);
267. **break**;
268. **case** 5 :
269. System.out.println("Empty status = "+ list.isEmpty());
270. **break**;
271. **case** 6 :
272. System.out.println("Size = "+ list.getSize() +" **\n**");
273. **break**;
274. **default** :
275. System.out.println("Wrong Entry **\n** ");
276. **break**;
277. }
278. */\* Display List \*/*
279. list.display();
280. System.out.println("**\n**Do you want to continue (Type y or n) **\n**");
281. ch = scan.next().charAt(0);
282. } **while** (ch == 'Y'|| ch == 'y');
283. }
284. }