|  |  |  |
| --- | --- | --- |
| Sukkur_IBA_New_Logo | **Sukkur IBA University**  **Department of Computer Science** | **C:\Users\Saif Hassan\Downloads\CS logo (3).jpg** |

**DATA STRUCTURES**

**Lab01 – Arrays, LinkesLists**

**Instructor: Saif Hassan**

**READ IT FIRST**

Prior to start solving the problems in this assignments, please give full concentration on following points.

1. WORKING – This is individual lab. If you are stuck in a problem contact your teacher, but, in mean time start doing next question (don’t waste time).
2. DEADLINE – 11th March, 2022
3. SUBMISSION – This assignment needs to be submitted in a soft copy.
4. WHERE TO SUBMIT – Please visit your LMS.
5. WHAT TO SUBMIT – Submit this docx and pdf file.

**KEEP IT WITH YOU!**

1. Indent your code inside the classes and functions. It’s a good practice!
2. It is not bad if you keep your code indented inside the loops, if and else blocks as well.
3. Comment your code, where it is necessary.
4. Read the entire question. Don’t jump to the formula directly.

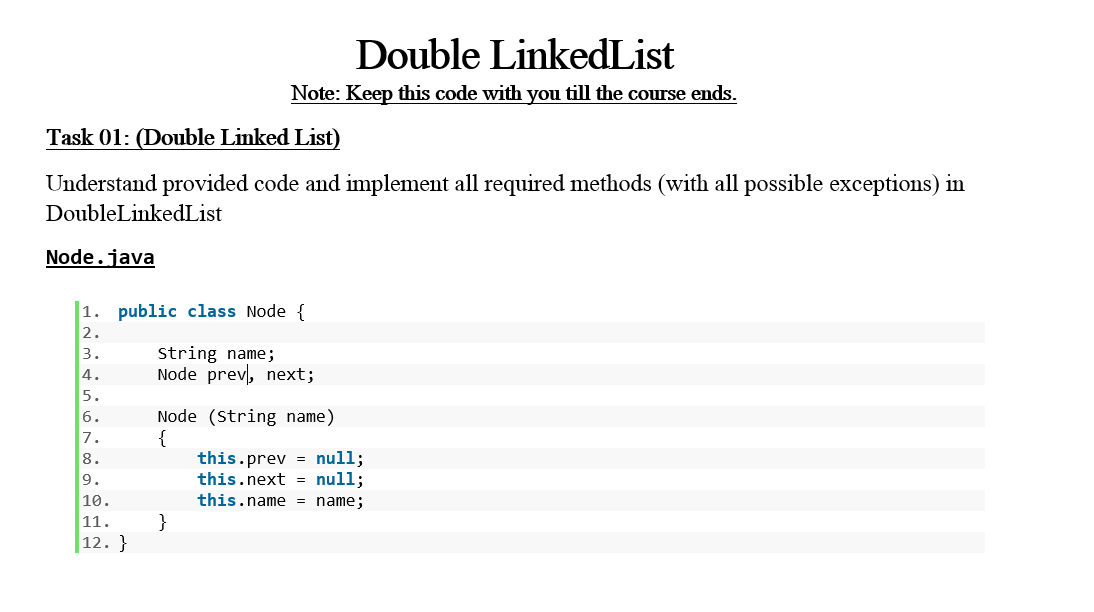
I, \_**Amjad Ali**\_ with student ID \_191-21-0001\_

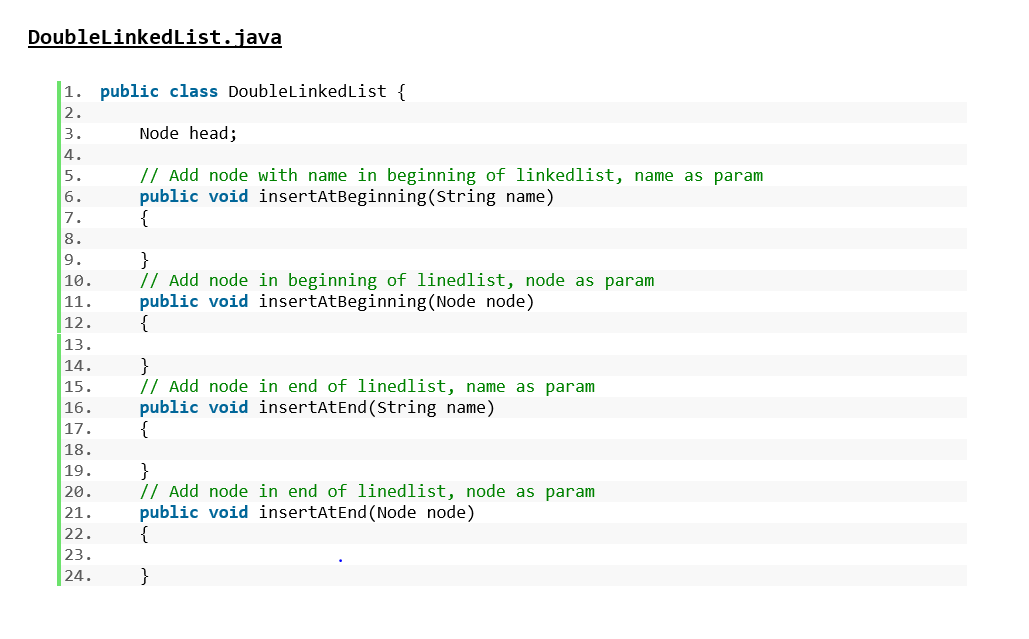
Section \_**A**\_hereby declare that I do understand the instructions above and follow them. This is

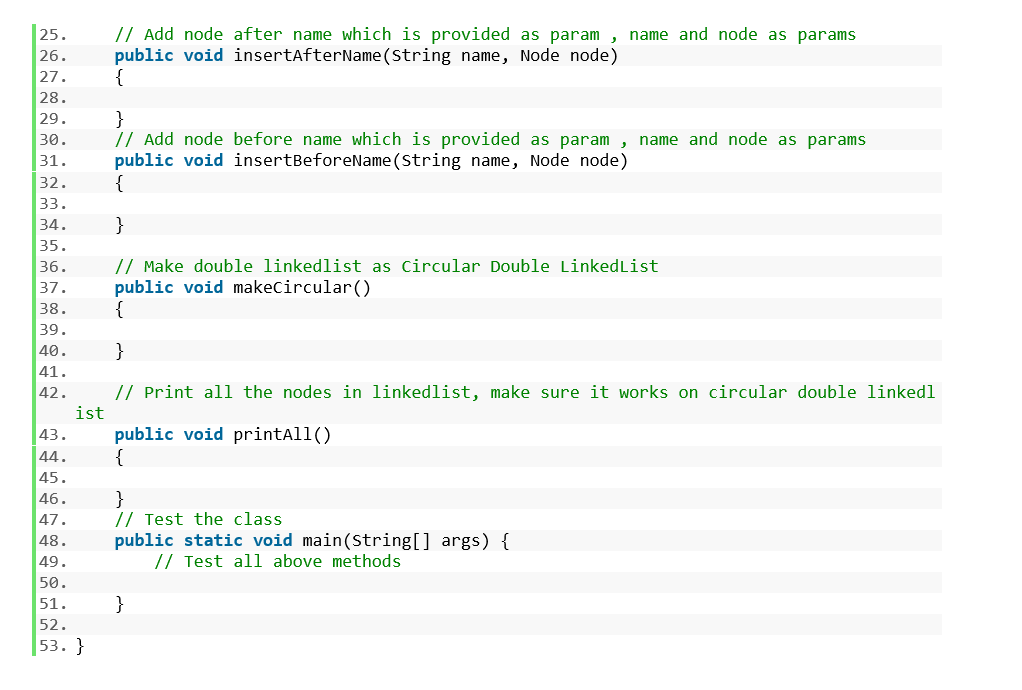
my own work.

**Exercises**

**Task1 Description**







Solution:

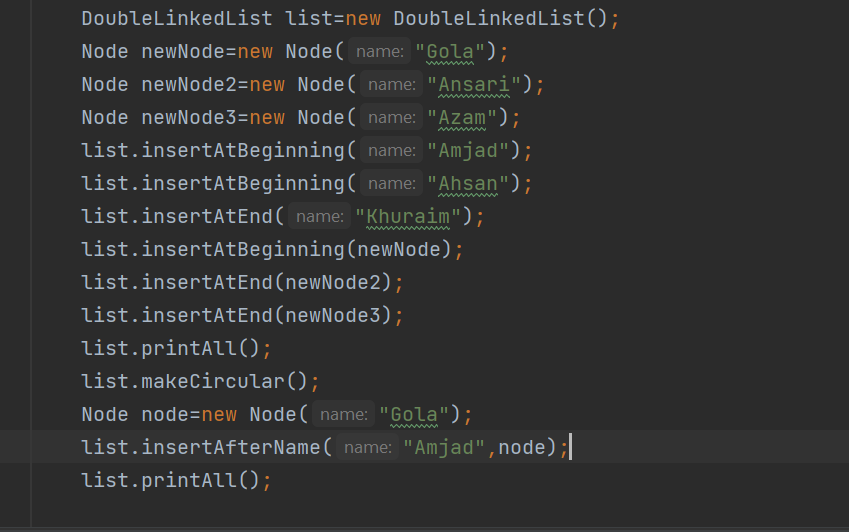
**Code**

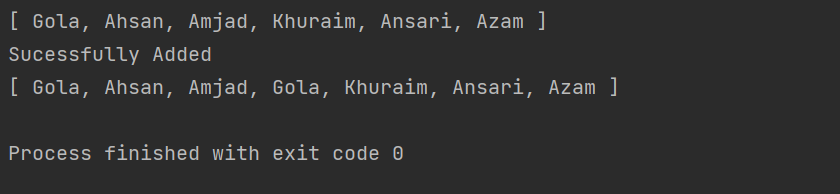
1. public class DoubleLinkedList {
3. Node head;
5. // Add node with name in beginning of linkedlist, name as param
6. public void insertAtBeginning(String name) {
8. Node newNode = new Node(name);
9. if (head == null) {
10. head = newNode;
11. } else {
12. head.prev = newNode;
13. newNode.next = head;
14. head = newNode;
16. }
17. }
18. // Add node in beginning of linedlist, node as param
20. public void insertAtBeginning(Node node) {
22. if (head == null) {
23. head = node;
24. } else {
25. head.prev = node;
26. node.next = head;
27. head = node;
29. }
30. }
31. // Add node in end of linedlist, name as param
33. public void insertAtEnd(String name) {
35. Node newNode = new Node(name);
36. if (head == null) {
37. head = newNode;
38. } else {
39. Node current=head;
40. while(current.next!=null)
41. {
42. current=current.next;
43. }
45. current.next=newNode;
46. newNode.prev=current;
47. }
49. }
50. // Add node in end of linedlist, node as param
52. public void insertAtEnd(Node node) {
54. if (head == null) {
55. head = node;
56. } else {
57. Node current=head;
58. while(current.next!=null)
59. {
60. current=current.next;
61. }
63. current.next=node;
64. node.prev=current;
65. }
67. }
68. // Add node after name which is provided as param , name and node as params
70. public void insertAfterName(String name, Node node) {
71. boolean a=false;
72. Node current=head;
73. while(current!=null && current.next!=head)
74. {
75. if(current.name==name)
76. {
77. a=true;
78. break;
79. }
80. current=current.next;
81. }
82. if(a)
83. {
84. Node temp=current.next;
85. current.next=node;
86. node.prev=current;
87. node.next=temp;
88. temp.prev=node;
89. System.out.println("Sucessfully Added");
90. }
91. else{
92. System.out.println("Name Doesn't Exist");
93. }

96. }
98. // Add node before name which is provided as param , name and node as params
99. public void insertBeforeName(String name, Node node) {
100. boolean a=false;
101. Node current=head;
102. while(current!=null && current.next!=head)
103. {
105. if(current.name==name)
106. {
107. a=true;
108. break;
109. }
110. current=current.next;
111. }
112. if(a)
113. {
114. Node temp=current.prev;
115. temp.next=node;
116. node.prev=temp;
117. node.next=current;
118. current.prev=node;
119. System.out.println("Sucessfully Added");
120. }
121. else{
122. System.out.println("Name Doesn't Exist");
123. }
125. }
127. // Make double linkedlist as Circular Double LinkedList
128. public void makeCircular() {
129. Node current=head;
130. while(current.next!=null)
131. current=current.next;
133. current.next=head;
134. head.prev=current;
136. }
138. // Print all the nodes in linkedlist, make sure it works on circular double linkedlist
139. public void printAll() {
140. Node current=head;
141. StringBuffer str=new StringBuffer();
142. str.append("[ ");
143. while(current.next!=null && current.next!=head )
144. {
145. str.append(current.name+", ");
146. current=current.next;
147. }
148. str.append(current.name+" ]");
149. System.out.println(str);
151. }
152. // Test the class
154. public static void main(String[] args) {
155. // Test all above methods
156. DoubleLinkedList list=new DoubleLinkedList();
157. Node newNode=new Node("Gola");
158. Node newNode2=new Node("Ansari");
159. Node newNode3=new Node("Azam");
160. list.insertAtBeginning("Amjad");
161. list.insertAtBeginning("Ahsan");
162. list.insertAtEnd("Khuraim");
163. list.insertAtBeginning(newNode);
164. list.insertAtEnd(newNode2);
165. list.insertAtEnd(newNode3);
166. list.printAll();
167. list.makeCircular();
168. Node node=new Node("Gola");
169. list.insertAfterName("Amjad",node);
171. list.printAll();

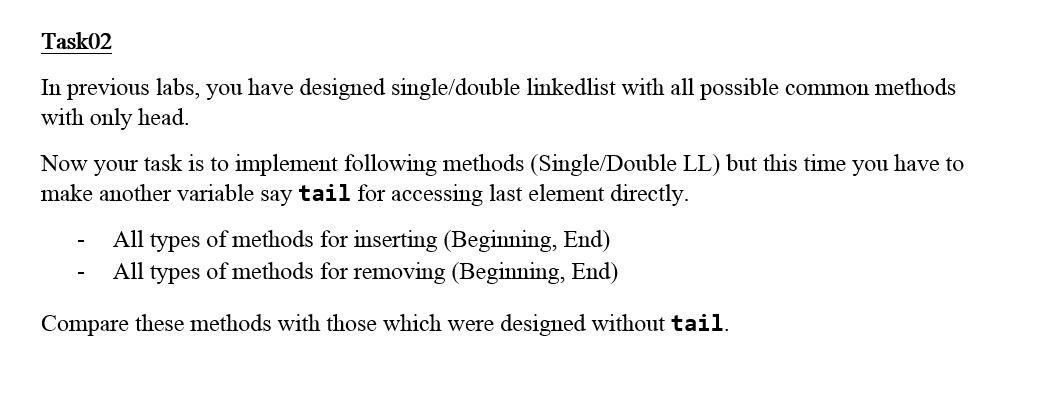

175. }
177. }

**Sample Input:**

**Sample Output**



**Task2 Description**



Solution:

**(Part 1 For Double Linked List with Tail.)**

**Code**

1. public class DoubleLinkedList {

4. Node head;
5. Node tail;
7. // Add node with name in beginning of linkedlist, name as param
8. public void insertAtBeginning(String name) {
10. Node newNode = new Node(name);
11. if (head == null) {
12. head = newNode;
13. tail = newNode;
14. } else {
15. head.prev = newNode;
16. newNode.next = head;
17. head = newNode;
19. }
20. }
21. // Add node in beginning of linedlist, node as param
23. public void insertAtBeginning(Node node) {
25. if (head == null) {
26. head = node;
27. tail = node;
28. } else {
29. head.prev = node;
30. node.next = head;
31. head = node;
33. }
34. }
35. public void RemoveAtBeginning()
36. {
37. if(head==null)
38. System.out.println("List is Empty");
39. else{
40. Node temp=head.next;
41. head=temp;
42. head.prev=null;
43. System.out.println("Succesfully Deleted");
44. }
45. }
47. public void RemoveAtEnding()
48. {
49. if(tail==null)
50. System.out.println("List is Empty");
51. else{
52. Node temp=tail.prev;
53. tail=temp;
54. tail.next=null;
55. System.out.println("Succesfully Deleted");
56. }
57. }
59. // Add node in end of linedlist, name as param
61. public void insertAtEnd(String name) {
63. Node newNode = new Node(name);
64. if (head == null) {
65. head = newNode;
66. tail = newNode;
67. } else {
68. newNode.prev = tail;
69. tail.next = newNode;
70. tail = newNode;
71. }
73. }
74. // Add node in end of linedlist, node as param
76. public void insertAtEnd(Node node) {
78. if (head == null) {
79. head = node;
80. tail = node;
81. } else {
82. node.prev = tail;
83. tail.next = node;
84. tail = node;
85. }
87. }
88. // Add node after name which is provided as param , name and node as params
90. public void insertAfterName(String name, Node node) {
91. boolean a=false;
92. Node current=head;
93. while(current!=null && current.next!=head)
94. {
95. if(current.name==name)
96. {
97. a=true;
98. break;
99. }
100. current=current.next;
101. }
102. if(a)
103. {
104. Node temp=current.next;
105. current.next=node;
106. node.prev=current;
107. node.next=temp;
108. temp.prev=node;
109. System.out.println("Sucessfully Added");
110. }
111. else{
112. System.out.println("Name Doesn't Exist");
113. }

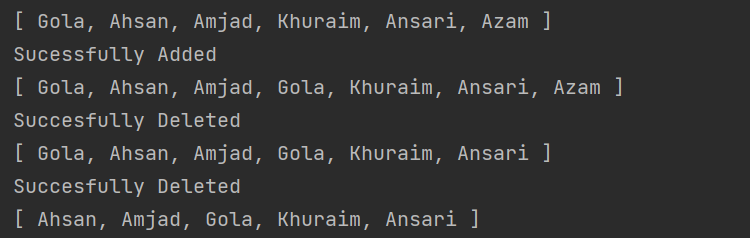
116. }
118. // Add node before name which is provided as param , name and node as params
119. public void insertBeforeName(String name, Node node) {
120. boolean a=false;
121. Node current=head;
122. while(current!=null && current.next!=head)
123. {
125. if(current.name==name)
126. {
127. a=true;
128. break;
129. }
130. current=current.next;
131. }
132. if(a)
133. {
134. Node temp=current.prev;
135. temp.next=node;
136. node.prev=temp;
137. node.next=current;
138. current.prev=node;
139. System.out.println("Sucessfully Added");
140. }
141. else{
142. System.out.println("Name Doesn't Exist");
143. }
145. }
147. // Make double linkedlist as Circular Double LinkedList
148. public void makeCircular() {
149. tail.next=head;
150. head.prev=tail;
151. System.out.println("Successfully Made Circular");
152. }
154. // Print all the nodes in linkedlist, make sure it works on circular double linkedlist
155. public void printAll() {
156. Node current=head;
157. StringBuffer str=new StringBuffer();
158. str.append("[ ");
159. while(current.next!=null && current.next!=head )
160. {
161. str.append(current.name+", ");
162. current=current.next;
163. }
164. str.append(current.name+" ]");
165. System.out.println(str);
167. }
168. // Test the class
170. public static void main(String[] args) {
171. // Test all above methods
172. DoubleLinkedList list=new DoubleLinkedList();
173. Node newNode=new Node("Gola");
174. Node newNode2=new Node("Ansari");
175. Node newNode3=new Node("Azam");
176. list.insertAtBeginning("Amjad");
177. list.insertAtBeginning("Ahsan");
178. list.insertAtEnd("Khuraim");
179. list.insertAtBeginning(newNode);
180. list.insertAtEnd(newNode2);
181. list.insertAtEnd(newNode3);
182. list.printAll();
183. Node node=new Node("Gola");
184. list.insertAfterName("Amjad",node);
185. list.printAll();
186. list.RemoveAtEnding();
187. list.printAll();
188. list.RemoveAtBeginning();
189. list.printAll();



194. }
196. }

**Sample Input:**

**Sample Output**



**(Part 2 For Single Linked List with Tail.)**

**Code:**

1. package com.company;
2. public class Linked\_List implements List{
4. private int size=0;
5. private Node Head;
6. private Node Tail;
8. public void reverseList(){
9. if(Head==null)
10. System.out.println("List is Empty");
11. else{
12. Node current=Head;
13. Node previous=null;
14. while(current.next!=null)
15. {
16. Node temp=current.next;
17. current.next=previous;
18. previous=current;
19. current=temp;
20. }
21. current.next=previous;
22. Head=current;
23. System.out.print("List Successfully Reversed");

26. }
28. }
29. @Override
30. public void incSize(){
31. size++;
32. }
34. @Override
35. public void decSize()
36. {
37. size--;
38. }

41. @Override
42. public boolean isEmpty()
43. {
44. return Head==null;
45. }
47. @Override
48. public void insertAtBeginning(int data)
49. {
50. Node newNode=new Node(data);
51. if(isEmpty())
52. {
53. Head=newNode;
54. Tail=newNode;
55. }
56. else{
57. Node temp=Head;
58. newNode.next=Head;
59. Head=newNode;
60. }
61. incSize();
63. }
64. @Override
65. public void insertAtEnding(int data){
66. Node newNode=new Node(data);
67. if(Tail==null)
68. {
69. Head=Tail=newNode;
70. }
71. else
72. {
73. Tail.next=newNode;
74. Tail=newNode;
75. }
76. incSize();
77. }
79. public void insertAtEnding(Node node){
81. if(Tail==null)
82. {
83. Head=Tail=node;
84. }
85. else
86. {
87. Tail.next=node;
88. Tail=node;
89. }
90. incSize();
91. }
93. public void insertAtBeginning(Node node){
95. if(isEmpty())
96. {
97. Head=node;
98. Tail=node;
99. }
100. else{
101. Node temp=Head;
102. node.next=Head;
103. Head=node;
104. }
105. incSize();
106. }
107. public void removeAtBeginning(){
108. if(Head==null)
109. System.out.println("List is Empty");
110. else if(Head.next==null)
111. {
112. Head=null;
113. decSize();
114. }
115. else{
116. Head=Head.next;
117. decSize();
118. }
119. }
120. public void removeAtEnding(){
121. if(Head==null)
122. System.out.println("List is Empty");
123. else if(Head.next==null)
124. {
125. Head=null;
126. decSize();
127. }
128. else{
129. Node current=Head;
130. Node previous=Head;
131. while(current.next!=null)
132. {
133. previous =current;
134. current=current.next;
136. }
137. previous.next=null;
139. decSize();
140. }
141. }
143. @Override
144. public int size() {
146. return size;
147. }
149. @Override
150. public void add(int index, int item) {
152. if(index>size)
153. System.out.println("index Out of Bound");
154. else{
155. int i=1;
156. Node current=Head;
157. while(i<index)
158. {
159. current=current.next;
160. i++;
161. }
162. current.data=item;
163. System.out.println("Successfully Added");
164. }
165. }
167. @Override
168. public void removeIndex(int index) {
169. if(size<index)
170. System.out.println("Index out of bound");
171. else if(index==1)
172. {
173. Head=Head.next;
174. }
175. else{
176. int i=1;
177. Node current=Head;
178. while(i+1!=index)
179. {
180. current=current.next;
181. i++;
182. }
183. current.next=current.next.next;
184. System.out.println("Successfully Removed");
185. size--;
186. }
188. }
190. public void remove(int item) {
191. if (size == 0) {
192. System.out.println("List Is Empty.");
193. } else {
194. boolean cond = false;
195. Node Current = Head;
196. Node oneBackCurrent = Head;
197. while (Current.next != null) {
199. if (Current.data==item) {
200. cond = true;
201. break;
203. }
204. oneBackCurrent = Current;
205. Current = Current.next;


209. }
210. if (Current.data==item)
211. cond = true;
213. if(Head.data==item)
214. {
215. Head=Head.next;
216. size--;
217. }
218. else if (cond) {
219. oneBackCurrent.next= oneBackCurrent.next.next;
220. System.out.println("Succesfully Removed");
221. size--;
222. } else {
223. System.out.println("No Such Element In the List");
224. }
225. }
226. }
227. public List duplicateReversed()
228. {
229. Linked\_List list=new Linked\_List();
230. Node Current=Head;
231. for(int i=0;i<size;i++)
232. {
234. list.insertAtEnding(Current.data);
235. Current=Current.next;
236. }

239. return list;
241. }
243. public List duplicate()
244. {
245. Linked\_List list=new Linked\_List();
246. Node Current=Head;
247. for(int i=0;i<size;i++)
248. {
249. Node newNode=new Node(Current.data);
251. if(list.isEmpty()) {
252. list.Head = newNode;
253. }
254. else{
255. Node Check=list.Head;
256. while(Check.next!=null)
257. {
258. Check=Check.next;
259. }
260. Check.next=newNode;

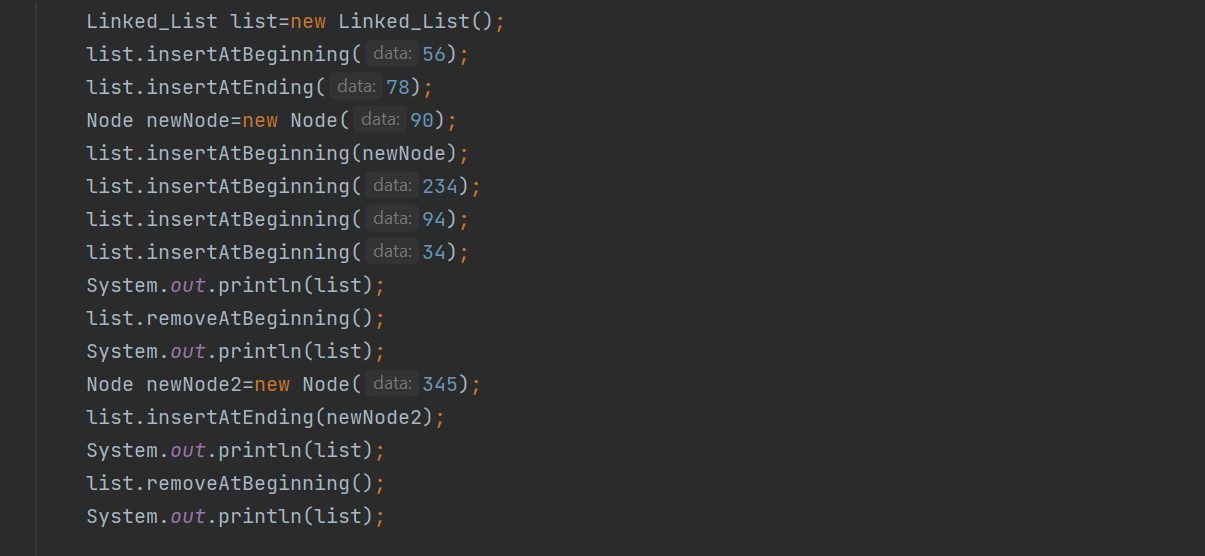
263. }
264. list.incSize();
265. Current=Current.next;
267. }
269. return list;
270. }

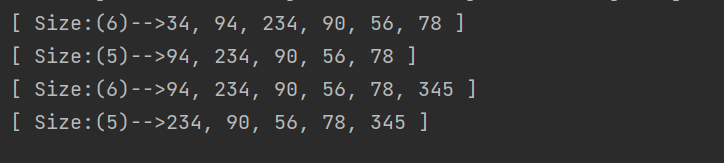
273. public String toString()
274. {
275. String Str ="[ Size:("+size+")-->";
276. Node Current=Head;
277. while(Current.next!=null)
278. {
279. Str+=Current.data+", ";
280. Current=Current.next;
281. }
282. Str+=Current.data+" ]";
284. return Str;
285. }
287. public static void main(String[] args) {
288. Linked\_List list=new Linked\_List();
289. list.insertAtBeginning(56);
290. list.insertAtEnding(78);
291. Node newNode=new Node(90);
292. list.insertAtBeginning(newNode);
293. list.insertAtBeginning(234);
294. list.insertAtBeginning(94);
295. list.insertAtBeginning(34);
296. System.out.println(list);
297. list.removeAtBeginning();
298. System.out.println(list);
299. Node newNode2=new Node(345);
300. list.insertAtEnding(newNode2);
301. System.out.println(list);
302. list.removeAtBeginning();
303. System.out.println(list);



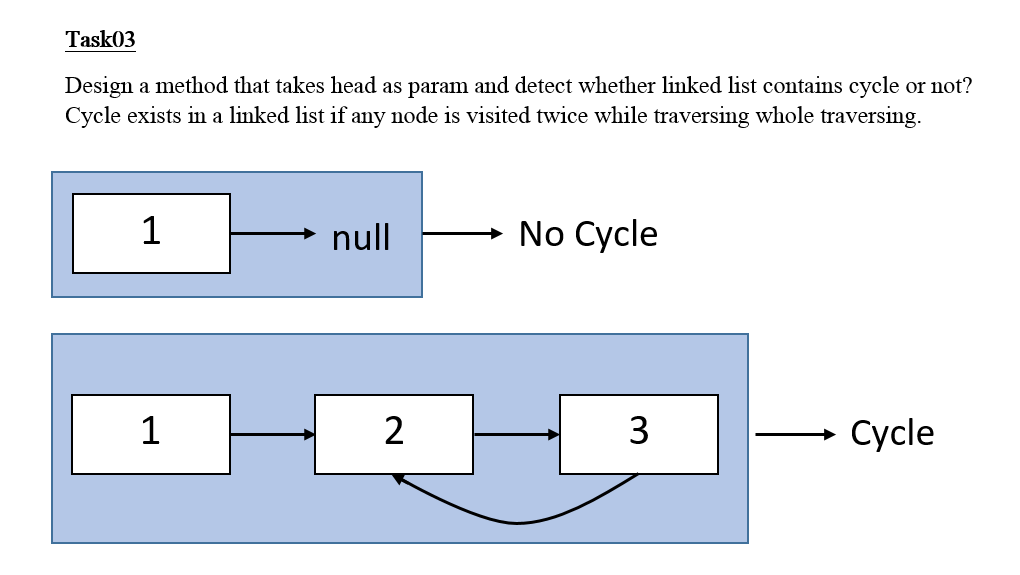

309. }
311. }

**Sample Input:**



**Sample Output:** 

**Task3 Description**



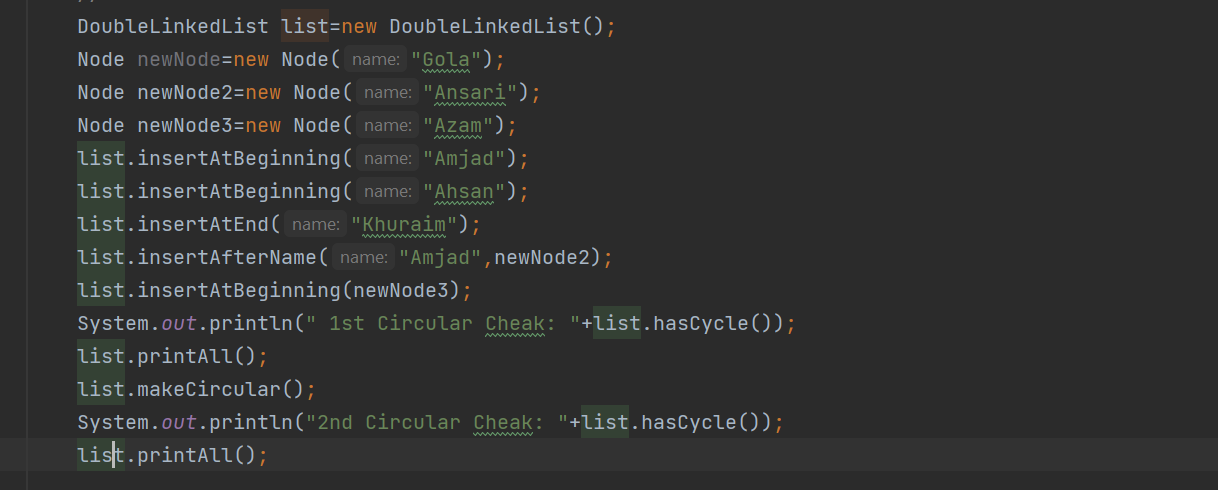
Solution:

**Code:**

1. public boolean hasCycle(ListNode head) {
2. if(head==null)
3. return false;

6. ListNode current=head;
7. ArrayList<ListNode> Array=new ArrayList<>();
8. while(current.next!=null)
9. {
10. if(Array.contains(current))
11. {
12. return true;
13. }
14. else
15. {
16. Array.add(current);
17. }
19. current=current.next;
21. }
23. return false;
25. }

**Sample Input:**

**Sample Output**

