Java Program to Implement Circular Doubly Linked List

This is a Java Program to implement a Circular Doubly Linked List. A linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data and a reference (in other words, a link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence. In a circular doubly linked list each node has two links one pointing to the next node in the list and one pointing to the previous node in the list and the last node's 'next link' points to the first node and the first node's 'previous link' points to the last node.

Here is the source code of the Java program to implement Circular Doubly Linked List. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.

```
1. /*
2.  *  Java Program to Implement Circular Doubly Linked List
3.  */
4.
```

```
5. import java.util.Scanner;
6.
7. /*  Class Node  */
8. class Node
9. {
10.     protected int data;
11.     protected Node next, prev;
12.
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. }
58.
59. /* Class linkedList */
60. class linkedList
61. {
```

```
62.     protected Node start;
63.     protected Node end ;
64.     public int size;
65.
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();
181. 
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("\nCircular 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. }
217.
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("\nCircular 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");
238.
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("\nDo you want to continue
    (Type y or n) \n");
281.             ch = scan.next().charAt(0);
282.         } while (ch == 'Y'|| ch == 'y');
```

```
283.      }

284. }
```

```
Circular Doubly Linked List Test


Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
1
Enter integer element to insert
5

Circular Doubly Linked List = 5 <-> 5

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
```

```
5. check empty
6. get size
1
Enter integer element to insert
7

Circular Doubly Linked List = 7 <-> 5 <-> 7

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
2
Enter integer element to insert
3

Circular Doubly Linked List = 7 <-> 5 <-> 3 <-> 7

Do you want to continue (Type y or n)

y
```

```
Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
3
Enter integer element to insert
2
Enter position
2

Circular Doubly Linked List = 7 <-> 2 <-> 5 <-> 3 <-> 7

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
3
Enter integer element to insert
```

```
4
Enter position
4

Circular Doubly Linked List = 7 <-> 2 <-> 5 <-> 4 <-> 3 <-> 7

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
6
Size = 5


Circular Doubly Linked List = 7 <-> 2 <-> 5 <-> 4 <-> 3 <-> 7

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations
```

```
1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
1

Circular Doubly Linked List = 2 <-> 5 <-> 4 <-> 3 <-> 2

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
1
Enter integer element to insert
1

Circular Doubly Linked List = 1 <-> 2 <-> 5 <-> 4 <-> 3 <-> 1
```

```
Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
1

Circular Doubly Linked List = 2 <-> 5 <-> 4 <-> 3 <-> 2

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
```

```
4
Enter position
1

Circular Doubly Linked List = 5 <-> 4 <-> 3 <-> 5

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
2

Circular Doubly Linked List = 5 <-> 3 <-> 5

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations
```

```
1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
1

Circular Doubly Linked List = 3 <-> 3

Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
4
Enter position
1

Circular Doubly Linked List = empty
```

```
Do you want to continue (Type y or n)

y

Circular Doubly Linked List Operations

1. insert at begining
2. insert at end
3. insert at position
4. delete at position
5. check empty
6. get size
5
Empty status = true

Circular Doubly Linked List = empty

Do you want to continue (Type y or n)

n
```

Sanfoundry Global Education & Learning Series – 1000 Java Programs.

If you wish to look at all Java Programming examples, go to Java Programs.

If you liked this Java Program, kindly share, recommend or like below!

Manish Bhojasia, a technology veteran with 17+ years @ Cisco & Wipro, is Founder and CTO at Sanfoundry. He is Linux Kernel Developer and SAN Architect and is passionate about competency developments in these areas. He lives in Bangalore and delivers focused training sessions to IT professionals in Linux Kernel, Linux Debugging, Linux Device Drivers, Linux Networking, Linux Storage & Cluster Administration, Advanced C

Programming, SAN Storage Technologies, SCSI Internals and Storage Protocols such as iSCSI & Fiber Channel. Stay connected with him below.