Name: M. Hamza Shahzad

Submitted to: Ms. Anum Munawar

Reg No: DSAI231103053

Department: Artificial Intelligence

Khajwa Fareed University of Engineering and Information Technology, RYK

```java
1  public class Singlylist {
2      Node head;
3
4      static class Node {
5          int value;
6          Node next;
7
8          Node(int val) {
9              this.value = val;
10             // this.next = null;
11         }
12     }
13
14     void printList() {
15         StringBuilder ret_str = new StringBuilder("[");
16         Node last = this.head;
17         while (last ≠ null) {
18             ret_str.append(last.value).append(", ");
19             last = last.next;
20         }
21         if (ret_str.length() > 1) {
22             ret_str.setLength(ret_str.length() - 2); // Remove trailing comma
    and space
23         }
24         ret_str.append("]");
25         System.out.println(ret_str);
26     }
27
28     void push(int val) {
29         Node new_node = new Node(val);
30
31         // If the list is empty, set the new node as the head
32         if (this.head == null) {
33             this.head = new_node;
34             return;
35         }
36
37         // Otherwise, traverse to the end of the list and add the new node
38         Node last = this.head;
39         while (last.next ≠ null) {
40             last = last.next;
41         }
42         last.next = new_node;
43     }
44
45     void insert(int val, int idx) {
46         Node new_node = new Node(val);
47
48         // Insert at the head if index is 0
49         if (idx == 0) {
```

```
50              new_node.next = this.head;
51              this.head = new_node;
52              return;
53          }

54

55          // Traverse to the specified index
56          Node last = this.head;
57          Node prev = null;
58          int counter = 0;
59          while (last ≠ null && counter < idx) {
60              prev = last;
61              last = last.next;
62              counter++;
63          }

64

65          // Insert the new node at the specified index
66          if (prev ≠ null) {
67              new_node.next = last;
68              prev.next = new_node;
69          } else {
70              throw new IndexOutOfBoundsException("Invalid Index! No Insert");
71          }
72      }

73

74      void remove(int idx) {
75          // Remove the head node if index is 0
76          if (idx == 0) {
77              if (this.head ≠ null) {
78                  this.head = this.head.next;
79              } else {
80                  throw new IndexOutOfBoundsException("Invalid Index! No
    Delete");
81              }
82              return;
83          }

84

85          // Traverse to the specified index
86          Node last = this.head;
87          Node prev = null;
88          int counter = 0;
89          while (last ≠ null && counter < idx) {
90              prev = last;
91              last = last.next;
92              counter++;
93          }

94

95          // Remove the node at the specified index
96          if (last ≠ null) {
97              prev.next = last.next;
98          } else {
99              throw new IndexOutOfBoundsException("Invalid Index! No Delete");
```

```
100             }
101         }
102
103     void pop() {
104         if (this.head == null) {
105             throw new IndexOutOfBoundsException("Invalid Index! No Delete");
106         }
107         if (this.head.next == null) {
108             this.head = null;
109             return;
110         }
111         Node last = this.head;
112         Node prev = null;
113         while (last.next ≠ null) {
114             prev = last;
115             last = last.next;
116         }
117         System.out.println("Popped: " + last.value);
118
119         prev.next = null;
120     }
121
122     public static void main(String[] args) {
123         Singlylist li = new Singlylist();
124         li.push(10);
125         li.push(20);
126         // li.push(30);
127         // li.push(40);
128
129         // Uncomment to test insertions
130         // li.insert(50, 0);
131         // li.insert(100, 3);
132         // li.insert(3434, 232); // This will throw an exception
133
134         // li.remove(0);
135         // li.remove(213); // This will throw an exception
136
137         // li.pop();
138         li.pop();
139         // li.pop();
140         li.printList(); // Output: [20, 30, 40]
141     }
142
143 }
144 // * doubly
145 class Doublylist {
146     Node head;
147
148     static class Node {
149         int value;
150         Node next;
```

```java
151            Node prev; // # changes in double
152
153            Node(int val) {
154                this.value = val;
155                this.next = null;
156                this.prev = null; // # changes in double
157            }
158        }
159
160    void printList() {
161        StringBuilder ret_str = new StringBuilder("[");
162        Node last = this.head;
163        while (last ≠ null) {
164            ret_str.append(last.value).append(", ");
165            last = last.next;
166        }
167        if (ret_str.length() > 1) {
168            ret_str.setLength(ret_str.length() - 2); // Remove trailing comma
    and space
169        }
170        ret_str.append("]");
171        System.out.println(ret_str);
172    }
173
174    void push(int val) {
175        Node new_node = new Node(val);
176
177        // If the list is empty, set the new node as the head
178        if (this.head == null) {
179            this.head = new_node;
180            return;
181        }
182
183        // Otherwise, traverse to the end of the list and add the new node
184        Node last = this.head;
185        while (last.next ≠ null) {
186            last = last.next;
187        }
188        last.next = new_node;
189        new_node.prev = last; // # changes in double
190    }
191
192    void insert(int val, int idx) {
193        Node new_node = new Node(val);
194
195        // Insert at the head if index is 0
196        if (idx == 0) {
197            new_node.next = this.head;
198            if (this.head ≠ null) {
199                this.head.prev = new_node; // # changes in double
200            }
```

```
201                  this.head = new_node;
202                  return;
203              }
204
205              // Traverse to the specified index
206              Node last = this.head;
207              Node prev = null;
208              int counter = 0;
209              while (last ≠ null && counter < idx) {
210                  prev = last;
211                  last = last.next;
212                  counter++;
213              }
214
215              // Insert the new node at the specified index
216              if (prev ≠ null) {
217                  new_node.next = last;
218                  new_node.prev = prev; // # changes in double
219                  prev.next = new_node;
220                  if (last ≠ null) {
221                      last.prev = new_node; // # changes in double
222                  }
223              } else {
224                  throw new IndexOutOfBoundsException("Invalid Index! No Insert");
225              }
226          }
227
228      void remove(int idx) {
229              // Remove the head node if index is 0
230              if (idx = 0) {
231                  if (this.head ≠ null) {
232                      this.head = this.head.next;
233                      if (this.head ≠ null) {
234                          this.head.prev = null; // # changes in double
235                      }
236                  } else {
237                      throw new IndexOutOfBoundsException("Invalid Index! No
        Delete");
238                  }
239                  return;
240              }
241
242              // Traverse to the specified index
243              Node last = this.head;
244              Node prev = null;
245              int counter = 0;
246              while (last ≠ null && counter < idx) {
247                  prev = last;
248                  last = last.next;
249                  counter++;
250              }
```

```java
251
252            // Remove the node at the specified index
253            if (last ≠ null) {
254                prev.next = last.next;
255                if (last.next ≠ null) {
256                    last.next.prev = prev; // # changes in double
257                }
258            } else {
259                throw new IndexOutOfBoundsException("Invalid Index! No Delete");
260            }
261        }
262
263        void pop() {
264            if (this.head == null) {
265                throw new IndexOutOfBoundsException("Invalid Index! No Delete");
266            }
267            if (this.head.next == null) {
268                this.head = null;
269                return;
270            }
271            Node last = this.head;
272            while (last.next ≠ null) {
273                last = last.next;
274            }
275            System.out.println("Popped: " + last.value);
276
277            if (last.prev ≠ null) {
278                last.prev.next = null; // # changes in double
279            }
280        }
281
282        public static void main(String[] args) {
283            Doublylist list = new Doublylist();
284            list.push(1);
285            list.push(2);
286            list.push(3);
287            list.printList(); // Output: [1, 2, 3]
288            list.insert(4, 1);
289            list.printList(); // Output: [1, 4, 2, 3]
290            list.remove(2);
291            list.printList(); // Output: [1, 4, 3]
292            list.pop();
293            list.printList(); // Output: [1, 4]
294        }
295    }
296
297  // * circular
298  public class Circularlist {
299      Node head;
300
301      static class Node {
```

```java
302             int value;
303             Node next;
304
305             Node(int val) {
306                 this.value = val;
307                 // this.next = null;
308             }
309         }
310
311     void printList() {
312         StringBuilder ret_str = new StringBuilder("[");
313         if (this.head ≠ null) { // # changes in circular
314             Node last = this.head;
315             do {
316                 ret_str.append(last.value).append(", ");
317                 last = last.next;
318             } while (last ≠ this.head); // # changes in circular
319             if (ret_str.length() > 1) {
320                 ret_str.setLength(ret_str.length() - 2); // Remove trailing
    comma and space
321             }
322         }
323         ret_str.append("]");
324         System.out.println(ret_str);
325     }
326
327     void push(int val) {
328         Node new_node = new Node(val);
329
330         // If the list is empty, set the new node as the head
331         if (this.head == null) {
332             this.head = new_node;
333             new_node.next = this.head; // # changes in circular
334             return;
335         }
336
337         // Otherwise, traverse to the end of the list and add the new node
338         Node last = this.head;
339         while (last.next ≠ this.head) { // # changes in circular
340             last = last.next;
341         }
342         last.next = new_node;
343         new_node.next = this.head; // # changes in circular
344     }
345
346     void insert(int val, int idx) {
347         Node new_node = new Node(val);
348
349         // Insert at the head if index is 0
350         if (idx == 0) {
351             if (this.head == null) {
```

```
352                    this.head = new_node;
353                    new_node.next = this.head; // # changes in circular
354                } else {
355                    Node last = this.head;
356                    while (last.next ≠ this.head) { // # changes in circular
357                        last = last.next;
358                    }
359                    new_node.next = this.head;
360                    this.head = new_node;
361                    last.next = this.head; // # changes in circular
362                }
363                return;
364            }
365
366            // Traverse to the specified index
367            Node last = this.head;
368            Node prev = null;
369            int counter = 0;
370            do {
371                prev = last;
372                last = last.next;
373                counter++;
374            } while (last ≠ this.head && counter < idx); // # changes in circular
375
376            // Insert the new node at the specified index
377            if (prev ≠ null && counter == idx) {
378                new_node.next = last;
379                prev.next = new_node;
380            } else {
381                throw new IndexOutOfBoundsException("Invalid Index! No Insert");
382            }
383        }
384
385        void remove(int idx) {
386            // Remove the head node if index is 0
387            if (idx == 0) {
388                if (this.head ≠ null) {
389                    Node last = this.head;
390                    while (last.next ≠ this.head) { // # changes in circular
391                        last = last.next;
392                    }
393                    if (this.head.next == this.head) { // Only one node in the list
394                        this.head = null;
395                    } else {
396                        this.head = this.head.next;
397                        last.next = this.head; // # changes in circular
398                    }
399                } else {
400                    throw new IndexOutOfBoundsException("Invalid Index! No
    Delete");
401                }
```

```java
402                return;
403            }
404
405            // Traverse to the specified index
406            Node last = this.head;
407            Node prev = null;
408            int counter = 0;
409            do {
410                prev = last;
411                last = last.next;
412                counter++;
413            } while (last ≠ this.head && counter < idx); // # changes in circular
414
415            // Remove the node at the specified index
416            if (last ≠ this.head && counter == idx) {
417                prev.next = last.next;
418            } else {
419                throw new IndexOutOfBoundsException("Invalid Index! No Delete");
420            }
421        }
422
423        void pop() {
424            if (this.head == null) {
425                throw new IndexOutOfBoundsException("Invalid Index! No Delete");
426            }
427            if (this.head.next == this.head) { // Only one node in the list
428                this.head = null;
429                return;
430            }
431            Node last = this.head;
432            Node prev = null;
433            while (last.next ≠ this.head) { // # changes in circular
434                prev = last;
435                last = last.next;
436            }
437            System.out.println("Popped: " + last.value);
438
439            if (prev ≠ null) {
440                prev.next = this.head; // # changes in circular
441            }
442        }
443
444        public static void main(String[] args) {
445            Circularlist li = new Circularlist();
446            li.push(10);
447            li.push(20);
448            li.push(30);
449            li.push(40);
450
451            // Uncomment to test insertions
452            li.insert(50, 0);
```

```
453            li.insert(100, 3);
454            // li.insert(3434, 232); // This will throw an exception
455
456            li.remove(0);
457            // li.remove(213); // This will throw an exception
458
459            li.pop();
460            li.printList(); // Output: [10, 20, 30]
461        }
462    }
```