Homework 1: ArrayLists

Graded

Student
Vidit Dharmendra Pokharna

Total Points
100 / 100 pts

Autograder Score
100.0 / 100.0

Question 2

Feedback & Manual Grading

0 / 0 pts

✔ **+ 0 pts** Correct

Great work :) -Isabelle □‿□

Autograder Results

Autograder Output
If you're seeing this message, everything compiled and ran properly! -CS1332 TAs

Submitted Files

ArrayList.java

Download

```java
import java.util.NoSuchElementException;

/**
 * Your implementation of an ArrayList.
 *
 * @author Vidit Pokharna
 * @version 1.0
 * @userid vpokharna3
 * @GTID 903772087
 *
 * Collaborators: LIST ALL COLLABORATORS YOU WORKED WITH HERE
 *
 * Resources: LIST ALL NON-COURSE RESOURCES YOU CONSULTED HERE
 */
public class ArrayList<T> {

    /**
     * The initial capacity of the ArrayList.
     *
     * DO NOT MODIFY THIS VARIABLE!
     */
    public static final int INITIAL_CAPACITY = 9;

    // Do not add new instance variables or modify existing ones.
    private T[] backingArray;
    private int size;

    /**
     * Constructs a new ArrayList.
     *
     * Java does not allow for regular generic array creation, so you will have
     * to cast an Object[] to a T[] to get the generic typing.
     */
    public ArrayList() {
        Object[] array = new Object[INITIAL_CAPACITY];
        backingArray = (T[]) array;
    }

    /**
     * Adds the element to the specified index.
     *
     * Remember that this add may require elements to be shifted.
     *
     * Must be amortized O(1) for index size and O(n) for all other cases.
     *
     * @param index the index at which to add the new element
```

```
47      * @param data  the data to add at the specified index
48      * @throws java.lang.IndexOutOfBoundsException if index < 0 or index > size
49      * @throws java.lang.IllegalArgumentException  if data is null
50      */
51     public void addAtIndex(int index, T data) {
52         if (index > size || index < 0) {
53             throw new IndexOutOfBoundsException("The index you have provided is outside the range of
    the array");
54         } else if (data == null) {
55             throw new IllegalArgumentException("The data provided does not have a value");
56         } else if (size == backingArray.length) {
57             Object[] tempArray = new Object[2 * backingArray.length];
58             T[] newArray = (T[]) tempArray;
59             for (int a = 0; a <= size; a++) {
60                 if (a < index) {
61                     newArray[a] = backingArray[a];
62                 } else if (a == index) {
63                     newArray[index] = data;
64                 } else {
65                     newArray[a] = backingArray[a - 1];
66                 }
67             }
68             backingArray = newArray;
69             size++;
70         } else {
71             for (int b = size - 1; b >= index; b--) {
72                 if (b < backingArray.length - 1) {
73                     backingArray[b + 1] = backingArray[b];
74                 }
75             }
76             backingArray[index] = data;
77             size++;
78         }
79     }
80
81     /**
82      * Adds the element to the front of the list.
83      *
84      * Remember that this add may require elements to be shifted.
85      *
86      * Must be O(n).
87      *
88      * @param data the data to add to the front of the list
89      * @throws java.lang.IllegalArgumentException if data is null
90      */
91     public void addToFront(T data) {
92         if (data == null) {
93             throw new IllegalArgumentException("The data provided does not have a value");
94         } else if (size == backingArray.length) {
```

```
95            Object[] tempArray = new Object[2 * backingArray.length];
96            T[] newArray = (T[]) tempArray;
97            for (int a = 0; a < size; a++) {
98                newArray[a + 1] = backingArray[a];
99            }
100           newArray[0] = data;
101           backingArray = newArray;
102           size++;
103       } else if (size < backingArray.length) {
104           for (int b = size - 1; b >= 0; b--) {
105               backingArray[b + 1] = backingArray[b];
106           }
107           backingArray[0] = data;
108           size++;
109       }
110   }
111
112   /**
113    * Adds the element to the back of the list.
114    *
115    * Must be amortized O(1).
116    *
117    * @param data the data to add to the back of the list
118    * @throws java.lang.IllegalArgumentException if data is null
119    */
120   public void addToBack(T data) {
121       if (data == null) {
122           throw new IllegalArgumentException("The data provided does not have a value");
123       } else if (size == backingArray.length) {
124           Object[] tempArray = new Object[2 * backingArray.length];
125           T[] newArray = (T[]) tempArray;
126           for (int a = 0; a < size; a++) {
127               newArray[a] = backingArray[a];
128           }
129           newArray[size] = data;
130           backingArray = newArray;
131           size++;
132       } else if (size < backingArray.length) {
133           backingArray[size] = data;
134           size++;
135       }
136   }
137
138   /**
139    * Removes and returns the element at the specified index.
140    *
141    * Remember that this remove may require elements to be shifted.
142    *
143    * Must be O(1) for index size - 1 and O(n) for all other cases.
```

```
144     *
145     * @param index the index of the element to remove
146     * @return the data formerly located at the specified index
147     * @throws java.lang.IndexOutOfBoundsException if index < 0 or index >= size
148     */
149    public T removeAtIndex(int index) {
150        if (index >= size || index < 0) {
151            throw new IndexOutOfBoundsException("The index you have provided is outside the range of
    the array");
152        } else {
153            T extract = backingArray[index];
154            for (int b = index; b < size - 1; b++) {
155                backingArray[b] = backingArray[b + 1];
156            }
157            backingArray[size - 1] = null;
158            size--;
159            return extract;
160        }
161    }
162
163    /**
164     * Removes and returns the first element of the list.
165     *
166     * Remember that this remove may require elements to be shifted.
167     *
168     * Must be O(n).
169     *
170     * @return the data formerly located at the front of the list
171     * @throws java.util.NoSuchElementException if the list is empty
172     */
173    public T removeFromFront() {
174        if (size <= 0) {
175            throw new NoSuchElementException("The list is empty and therefore no element can be
    removed");
176        } else {
177            T extract = backingArray[0];
178            for (int b = 0; b < size - 1; b++) {
179                backingArray[b] = backingArray[b + 1];
180            }
181            backingArray[size - 1] = null;
182            size--;
183            return extract;
184        }
185    }
186
187    /**
188     * Removes and returns the last element of the list.
189     *
190     * Must be O(1).
```

```
191     *
192     * @return the data formerly located at the back of the list
193     * @throws java.util.NoSuchElementException if the list is empty
194     */
195    public T removeFromBack() {
196        if (size <= 0) {
197            throw new NoSuchElementException("The list is empty and therefore no element can be
    removed");
198        } else {
199            T extract = backingArray[size - 1];
200            backingArray[size - 1] = null;
201            size--;
202            return extract;
203        }
204    }
205
206    /**
207     * Returns the element at the specified index.
208     *
209     * Must be O(1).
210     *
211     * @param index the index of the element to get
212     * @return the data stored at the index in the list
213     * @throws java.lang.IndexOutOfBoundsException if index < 0 or index >= size
214     */
215    public T get(int index) {
216        if (index >= size || index < 0) {
217            throw new IndexOutOfBoundsException("The index you have provided is outside the range of
    the array");
218        } else {
219            return backingArray[index];
220        }
221    }
222
223    /**
224     * Returns whether or not the list is empty.
225     *
226     * Must be O(1).
227     *
228     * @return true if empty, false otherwise
229     */
230    public boolean isEmpty() {
231        return size <= 0;
232    }
233
234    /**
235     * Clears the list.
236     *
237     * Resets the backing array to a new array of the initial capacity and
```

```
238      * resets the size.
239      *
240      * Must be O(1).
241      */
242     public void clear() {
243         size = 0;
244         Object[] array = new Object[INITIAL_CAPACITY];
245         backingArray = (T[]) array;
246     }
247 
248     /**
249      * Returns the backing array of the list.
250      *
251      * For grading purposes only. You shouldn't need to use this method since
252      * you have direct access to the variable.
253      *
254      * @return the backing array of the list
255      */
256     public T[] getBackingArray() {
257         // DO NOT MODIFY THIS METHOD!
258         return backingArray;
259     }
260 
261     /**
262      * Returns the size of the list.
263      *
264      * For grading purposes only. You shouldn't need to use this method since
265      * you have direct access to the variable.
266      *
267      * @return the size of the list
268      */
269     public int size() {
270         // DO NOT MODIFY THIS METHOD!
271         return size;
272     }
273 }
274 
```