Homework 3: Stacks and Queues

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

ArrayQueue.java

Download

```java
import java.util.NoSuchElementException;

/**
 * Your implementation of an ArrayQueue.
 *
 * @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 ArrayQueue<T> {

    /*
     * The initial capacity of the ArrayQueue.
     *
     * 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 front;
    private int size;

    /**
     * Constructs a new ArrayQueue.
     */
    public ArrayQueue() {
        backingArray = (T[]) new Object[INITIAL_CAPACITY];
    }

    /**
     * Adds the data to the back of the queue.
     *
     * If sufficient space is not available in the backing array, resize it to
     * double the current length. When resizing, copy elements to the
     * beginning of the new array and reset front to 0.
     *
     * Must be amortized O(1).
     *
```

```
47      * @param data the data to add to the back of the queue
48      * @throws java.lang.IllegalArgumentException if data is null
49      */
50     public void enqueue(T data) {
51         if (data == null) {
52             throw new IllegalArgumentException("The data provided does not have a value");
53         } else if (size == backingArray.length) {
54             T[] newArray = (T[]) new Object[2 * backingArray.length];
55             for (int a = 0; a < size; a++) {
56                 newArray[a] = backingArray[(front + a) % backingArray.length];
57             }
58             newArray[size] = data;
59             backingArray = newArray;
60             size++;
61             front = 0;
62         } else if (size < backingArray.length) {
63             int index = (front + size) % backingArray.length;
64             backingArray[index] = data;
65             size++;
66         }
67     }
68 
69     /**
70      * Removes and returns the data from the front of the queue.
71      *
72      * Do not shrink the backing array.
73      *
74      * Replace any spots that you dequeue from with null.
75      *
76      * If the queue becomes empty as a result of this call, do not reset
77      * front to 0.
78      *
79      * Must be O(1).
80      *
81      * @return the data formerly located at the front of the queue
82      * @throws java.util.NoSuchElementException if the queue is empty
83      */
84     public T dequeue() {
85         if (size == 0) {
86             throw new NoSuchElementException("The queue is empty and therefore no element can be
    removed");
87         }
88         T extract = backingArray[front];
89         backingArray[front] = null;
90         front = (front + 1) % backingArray.length;
91         size--;
92         return extract;
93     }
94 
```

```
95      /**
96       * Returns the data from the front of the queue without removing it.
97       *
98       * Must be O(1).
99       *
100      * @return the data located at the front of the queue
101      * @throws java.util.NoSuchElementException if the queue is empty
102      */
103     public T peek() {
104         if (size == 0) {
105             throw new NoSuchElementException("The queue is empty and therefore no element can be
    removed");
106         }
107         return backingArray[front];
108     }
109
110     /**
111      * Returns the backing array of the queue.
112      *
113      * For grading purposes only. You shouldn't need to use this method since
114      * you have direct access to the variable.
115      *
116      * @return the backing array of the queue
117      */
118     public T[] getBackingArray() {
119         // DO NOT MODIFY THIS METHOD!
120         return backingArray;
121     }
122
123     /**
124      * Returns the front index of the queue.
125      *
126      * For grading purposes only. You shouldn't need to use this method since
127      * you have direct access to the variable.
128      *
129      * @return the front index of the queue
130      */
131     public int getFront() {
132         // DO NOT MODIFY THIS METHOD!
133         return front;
134     }
135
136     /**
137      * Returns the size of the queue.
138      *
139      * For grading purposes only. You shouldn't need to use this method since
140      * you have direct access to the variable.
141      *
142      * @return the size of the queue
```

```
143      */
144     public int size() {
145         // DO NOT MODIFY THIS METHOD!
146         return size;
147     }
148 }
149
```

ArrayStack.java

Download

```java
import java.util.NoSuchElementException;

/**
 * Your implementation of an ArrayStack.
 *
 * @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 ArrayStack<T> {

    /*
     * The initial capacity of the ArrayStack.
     *
     * 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 ArrayStack.
     */
    public ArrayStack() {
        backingArray = (T[]) new Object[INITIAL_CAPACITY];
    }

    /**
     * Adds the data to the top of the stack.
     *
     * If sufficient space is not available in the backing array, resize it to
     * double the current length.
     *
     * Must be amortized O(1).
     *
     * @param data the data to add to the top of the stack
     * @throws java.lang.IllegalArgumentException if data is null
```

```
47      */
48     public void push(T data) {
49         if (data == null) {
50             throw new IllegalArgumentException("The data provided has no value");
51         }  else if (size == backingArray.length) {
52             T[] newArray = (T[]) new Object[2 * backingArray.length];
53             for (int a = 0; a < size; a++) {
54                 newArray[a] = backingArray[a];
55             }
56             newArray[size] = data;
57             backingArray = newArray;
58             size++;
59         } else if (size < backingArray.length) {
60             backingArray[size] = data;
61             size++;
62         }
63 
64     }
65 
66     /**
67      * Removes and returns the data from the top of the stack.
68      *
69      * Do not shrink the backing array.
70      *
71      * Replace any spots that you pop from with null.
72      *
73      * Must be O(1).
74      *
75      * @return the data formerly located at the top of the stack
76      * @throws java.util.NoSuchElementException if the stack is empty
77      */
78     public T pop() {
79         if (size == 0) {
80             throw new NoSuchElementException("The stack is empty and no value can be found at the
    top");
81         }
82         T remove = backingArray[size - 1];
83         backingArray[size - 1] = null;
84         size--;
85         return remove;
86     }
87 
88     /**
89      * Returns the data from the top of the stack without removing it.
90      *
91      * Must be O(1).
92      *
93      * @return the data from the top of the stack
94      * @throws java.util.NoSuchElementException if the stack is empty
```

```
95      */
96     public T peek() {
97         if (size == 0) {
98             throw new NoSuchElementException("The stack is empty and no value can be found at the
    top");
99         }
100        return backingArray[size - 1];
101    }
102
103    /**
104     * Returns the backing array of the stack.
105     *
106     * For grading purposes only. You shouldn't need to use this method since
107     * you have direct access to the variable.
108     *
109     * @return the backing array of the stack
110     */
111    public T[] getBackingArray() {
112        // DO NOT MODIFY THIS METHOD!
113        return backingArray;
114    }
115
116    /**
117     * Returns the size of the stack.
118     *
119     * For grading purposes only. You shouldn't need to use this method since
120     * you have direct access to the variable.
121     *
122     * @return the size of the stack
123     */
124    public int size() {
125        // DO NOT MODIFY THIS METHOD!
126        return size;
127    }
128 }
129
```

LinkedQueue.java

Download

```java
import java.util.NoSuchElementException;

/**
 * Your implementation of a LinkedQueue. It should NOT be circular.
 *
 * @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 LinkedQueue<T> {

    /*
     * Do not add new instance variables or modify existing ones.
     */
    private LinkedNode<T> head;
    private LinkedNode<T> tail;
    private int size;

    /*
     * Do not add a constructor.
     */

    /**
     * Adds the data to the back of the queue.
     *
     * Must be O(1).
     *
     * @param data the data to add to the back of the queue
     * @throws java.lang.IllegalArgumentException if data is null
     */
    public void enqueue(T data) {
        if (data == null) {
            throw new IllegalArgumentException("The data provided does not have a value");
        } else if (size == 0) {
            head = new LinkedNode<T>(data);
            tail = head;
            size++;
        } else {
            tail.setNext(new LinkedNode<T>(data));
            tail = tail.getNext();
            size++;
```

```
47          }
48      }
49
50      /**
51       * Removes and returns the data from the front of the queue.
52       *
53       * Must be O(1).
54       *
55       * @return the data formerly located at the front of the queue
56       * @throws java.util.NoSuchElementException if the queue is empty
57       */
58      public T dequeue() {
59          if (size == 0) {
60              throw new NoSuchElementException("The queue is empty so no element can be removed from
    the linked list");
61          } else if (size == 1) {
62              T remove = head.getData();
63              head = null;
64              tail = null;
65              size--;
66              return remove;
67          } else {
68              T remove = head.getData();
69              head = head.getNext();
70              size--;
71              return remove;
72          }
73      }
74
75      /**
76       * Returns the data from the front of the queue without removing it.
77       *
78       * Must be O(1).
79       *
80       * @return the data located at the front of the queue
81       * @throws java.util.NoSuchElementException if the queue is empty
82       */
83      public T peek() {
84          if (size == 0) {
85              throw new NoSuchElementException("The queue is empty so no element can be removed from
    the linked list");
86          }
87          return head.getData();
88      }
89
90      /**
91       * Returns the head node of the queue.
92       *
93       * For grading purposes only. You shouldn't need to use this method since
```

```
94      * you have direct access to the variable.
95      *
96      * @return the node at the head of the queue
97      */
98     public LinkedNode<T> getHead() {
99         // DO NOT MODIFY THIS METHOD!
100        return head;
101    }
102
103    /**
104     * Returns the tail node of the queue.
105     *
106     * For grading purposes only. You shouldn't need to use this method since
107     * you have direct access to the variable.
108     *
109     * @return the node at the tail of the queue
110     */
111    public LinkedNode<T> getTail() {
112        // DO NOT MODIFY THIS METHOD!
113        return tail;
114    }
115
116    /**
117     * Returns the size of the queue.
118     *
119     * For grading purposes only. You shouldn't need to use this method since
120     * you have direct access to the variable.
121     *
122     * @return the size of the queue
123     */
124    public int size() {
125        // DO NOT MODIFY THIS METHOD!
126        return size;
127    }
128}
129
```

LinkedStack.java

Download

```java
import java.util.NoSuchElementException;

/**
 * Your implementation of a LinkedStack. It should NOT be circular.
 *
 * @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 LinkedStack<T> {

    /*
     * Do not add new instance variables or modify existing ones.
     */
    private LinkedNode<T> head;
    private int size;

    /*
     * Do not add a constructor.
     */

    /**
     * Adds the data to the top of the stack.
     *
     * Must be O(1).
     *
     * @param data the data to add to the top of the stack
     * @throws java.lang.IllegalArgumentException if data is null
     */
    public void push(T data) {
        if (data == null) {
            throw new IllegalArgumentException("The data provided has no value");
        } else if (size == 0) {
            head = new LinkedNode<T>(data);
            size++;
        } else {
            LinkedNode<T> newNode = new LinkedNode<T>(data);
            newNode.setNext(head);
            head = newNode;
            size++;
        }
```

```
47      }
48
49      /**
50       * Removes and returns the data from the top of the stack.
51       *
52       * Must be O(1).
53       *
54       * @return the data formerly located at the top of the stack
55       * @throws java.util.NoSuchElementException if the stack is empty
56       */
57      public T pop() {
58          if (size == 0) {
59              throw new NoSuchElementException("The stack is empty so no element can be removed from
    the linked list");
60          } else if (size == 1) {
61              T remove = head.getData();
62              head = null;
63              size--;
64              return remove;
65          } else {
66              T remove = head.getData();
67              head = head.getNext();
68              size--;
69              return remove;
70          }
71      }
72
73      /**
74       * Returns the data from the top of the stack without removing it.
75       *
76       * Must be O(1).
77       *
78       * @return the data from the top of the stack
79       * @throws java.util.NoSuchElementException if the stack is empty
80       */
81      public T peek() {
82          if (size == 0) {
83              throw new NoSuchElementException("The stack is empty so no element can be removed from
    the linked list");
84          }
85          return head.getData();
86      }
87
88      /**
89       * Returns the head node of the stack.
90       *
91       * For grading purposes only. You shouldn't need to use this method since
92       * you have direct access to the variable.
93       *
```

```
94      * @return the node at the head of the stack
95      */
96     public LinkedNode<T> getHead() {
97         // DO NOT MODIFY!
98         return head;
99     }
100
101    /**
102     * Returns the size of the stack.
103     *
104     * For grading purposes only. You shouldn't need to use this method since
105     * you have direct access to the variable.
106     *
107     * @return the size of the stack
108     */
109    public int size() {
110        // DO NOT MODIFY THIS METHOD!
111        return size;
112    }
113 }
114
```