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).
     *
     * @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 == backingArray.length) {
            T[] newArray = (T[]) new Object[2 * backingArray.length];
            for (int a = 0; a < size; a++) {
                newArray[a] = backingArray[(front + a) % backingArray.length];
            }
            newArray[size] = data;
            backingArray = newArray;
            size++;
            front = 0;
        } else if (size < backingArray.length) {
            int index = (front + size) % backingArray.length;
            backingArray[index] = data;
            size++;
        }
    }

    /**
     * Removes and returns the data from the front of the queue.
     *
     * Do not shrink the backing array.
     *
     * Replace any spots that you dequeue from with null.
     *
     * If the queue becomes empty as a result of this call, do not reset
     * front to 0.
     *
     * Must be O(1).
     *
     * @return the data formerly located at the front of the queue
     * @throws java.util.NoSuchElementException if the queue is empty
     */
    public T dequeue() {
        if (size == 0) {
            throw new NoSuchElementException("The queue is empty and therefore no element can be removed");
        }
        T extract = backingArray[front];
        backingArray[front] = null;
        front = (front + 1) % backingArray.length;
        size--;
        return extract;
    }

    /**
     * Returns the data from the front of the queue without removing it.
     *
     * Must be O(1).
     *
     * @return the data located at the front of the queue
     * @throws java.util.NoSuchElementException if the queue is empty
     */
    public T peek() {
        if (size == 0) {
            throw new NoSuchElementException("The queue is empty and therefore no element can be removed");
        }
        return backingArray[front];
    }

    /**
     * Returns the backing array of the queue.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the backing array of the queue
     */
    public T[] getBackingArray() {
        // DO NOT MODIFY THIS METHOD!
        return backingArray;
    }

    /**
     * Returns the front index of the queue.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the front index of the queue
     */
    public int getFront() {
        // DO NOT MODIFY THIS METHOD!
        return front;
    }

    /**
     * Returns the size of the queue.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the size of the queue
     */
    public int size() {
        // DO NOT MODIFY THIS METHOD!
        return size;
    }
}

```

ArrayStack.java

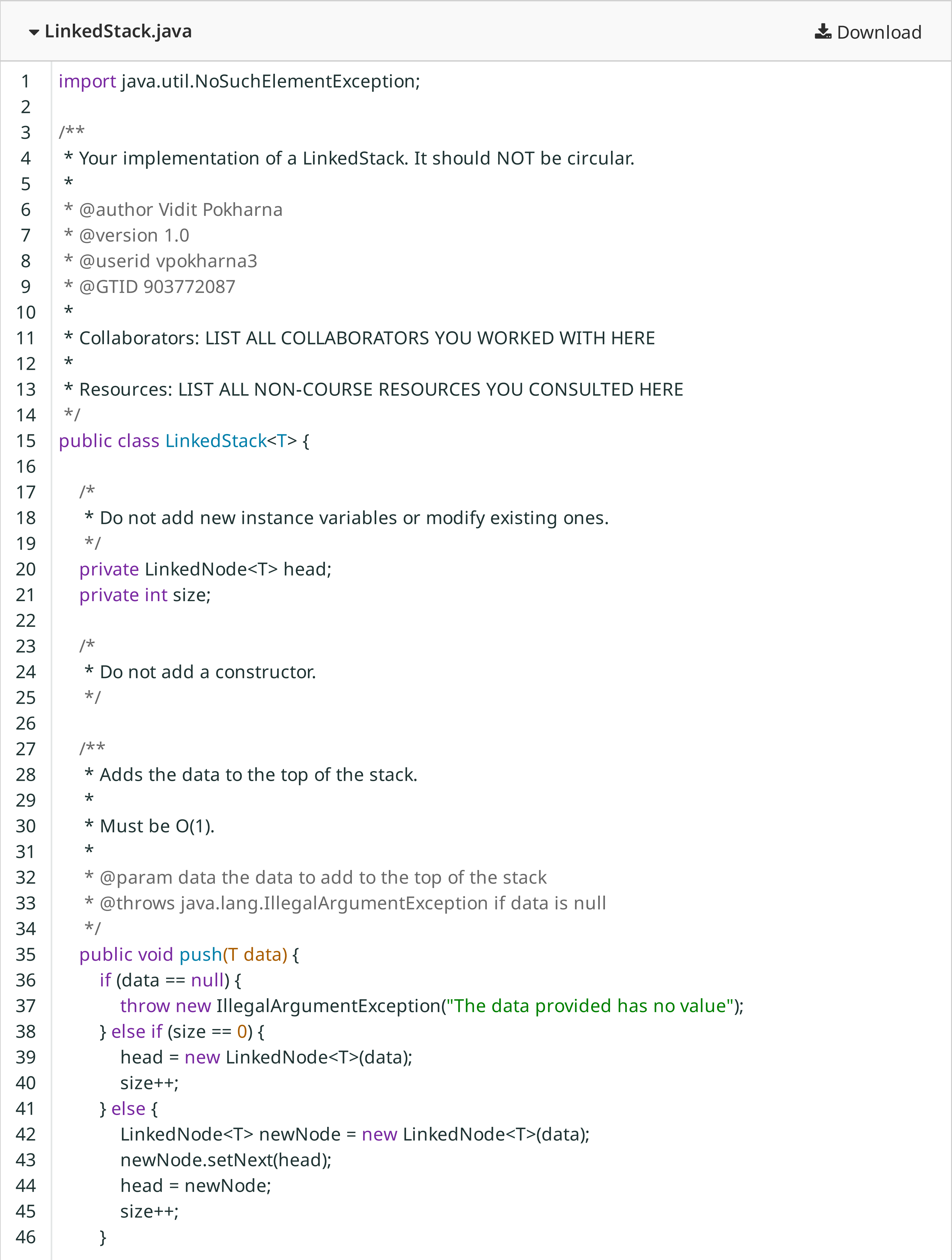
Download

```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
     */
    public void push(T data) {
        if (data == null) {
            throw new IllegalArgumentException("The data provided has no value");
        }  else if (size == backingArray.length) {
            T[] newArray = (T[]) new Object[2 * backingArray.length];
            for (int a = 0; a < size; a++) {
                newArray[a] = backingArray[a];
            }
            newArray[size] = data;
            backingArray = newArray;
            size++;
        } else if (size < backingArray.length) {
            backingArray[size] = data;
            size++;
        }

    }

    /**
     * Removes and returns the data from the top of the stack.
     *
     * Do not shrink the backing array.
     *
     * Replace any spots that you pop from with null.
     *
     * Must be O(1).
     *
     * @return the data formerly located at the top of the stack
     * @throws java.util.NoSuchElementException if the stack is empty
     */
    public T pop() {
        if (size == 0) {
            throw new NoSuchElementException("The stack is empty and no value can be found at the top");
        }
        T remove = backingArray[size - 1];
        backingArray[size - 1] = null;
        size--;
        return remove;
    }

    /**
     * Returns the data from the top of the stack without removing it.
     *
     * Must be O(1).
     *
     * @return the data from the top of the stack
     * @throws java.util.NoSuchElementException if the stack is empty
```

```java
     */
    public T peek() {
        if (size == 0) {
            throw new NoSuchElementException("The stack is empty and no value can be found at the top");
        }
        return backingArray[size - 1];
    }

    /**
     * Returns the backing array of the stack.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the backing array of the stack
     */
    public T[] getBackingArray() {
        // DO NOT MODIFY THIS METHOD!
        return backingArray;
    }

    /**
     * Returns the size of the stack.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the size of the stack
     */
    public int size() {
        // DO NOT MODIFY THIS METHOD!
        return size;
    }
}
```

▾ **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++;
        }
    }

    /**
     * Removes and returns the data from the front of the queue.
     *
     * Must be O(1).
     *
     * @return the data formerly located at the front of the queue
     * @throws java.util.NoSuchElementException if the queue is empty
     */
    public T dequeue() {
        if (size == 0) {
            throw new NoSuchElementException("The queue is empty so no element can be removed from the linked list");
        } else if (size == 1) {
            T remove = head.getData();
            head = null;
            tail = null;
            size--;
            return remove;
        } else {
            T remove = head.getData();
            head = head.getNext();
            size--;
            return remove;
        }
    }

    /**
     * Returns the data from the front of the queue without removing it.
     *
     * Must be O(1).
     *
     * @return the data located at the front of the queue
     * @throws java.util.NoSuchElementException if the queue is empty
     */
    public T peek() {
        if (size == 0) {
            throw new NoSuchElementException("The queue is empty so no element can be removed from the linked list");
        }
        return head.getData();
    }

    /**
     * Returns the head node of the queue.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the node at the head of the queue
     */
    public LinkedNode<T> getHead() {
        // DO NOT MODIFY THIS METHOD!
        return head;
    }

    /**
     * Returns the tail node of the queue.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the node at the tail of the queue
     */
    public LinkedNode<T> getTail() {
        // DO NOT MODIFY THIS METHOD!
        return tail;
    }

    /**
     * Returns the size of the queue.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the size of the queue
     */
    public int size() {
        // DO NOT MODIFY THIS METHOD!
        return size;
    }
}

```



LinkedStack.java

```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++;
        }
    }

    /**
     * Removes and returns the data from the top of the stack.
     *
     * Must be O(1).
     *
     * @return the data formerly located at the top of the stack
     * @throws java.util.NoSuchElementException if the stack is empty
     */
    public T pop() {
        if (size == 0) {
            throw new NoSuchElementException("The stack is empty so no element can be removed from the linked list");
        } else if (size == 1) {
            T remove = head.getData();
            head = null;
            size--;
            return remove;
        } else {
            T remove = head.getData();
            head = head.getNext();
            size--;
            return remove;
        }
    }

    /**
     * Returns the data from the top of the stack without removing it.
     *
     * Must be O(1).
     *
     * @return the data from the top of the stack
     * @throws java.util.NoSuchElementException if the stack is empty
     */
    public T peek() {
        if (size == 0) {
            throw new NoSuchElementException("The stack is empty so no element can be removed from the linked list");
        }
        return head.getData();
    }

    /**
     * Returns the head node of the stack.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the node at the head of the stack
     */
    public LinkedNode<T> getHead() {
        // DO NOT MODIFY!
        return head;
    }

    /**
     * Returns the size of the stack.
     *
     * For grading purposes only. You shouldn't need to use this method since
     * you have direct access to the variable.
     *
     * @return the size of the stack
     */
    public int size() {
        // DO NOT MODIFY THIS METHOD!
        return size;
    }
}

```