Oct 3

Assignment 4

Comp 2230\_02

Colton isles and kaylee crocker

2024

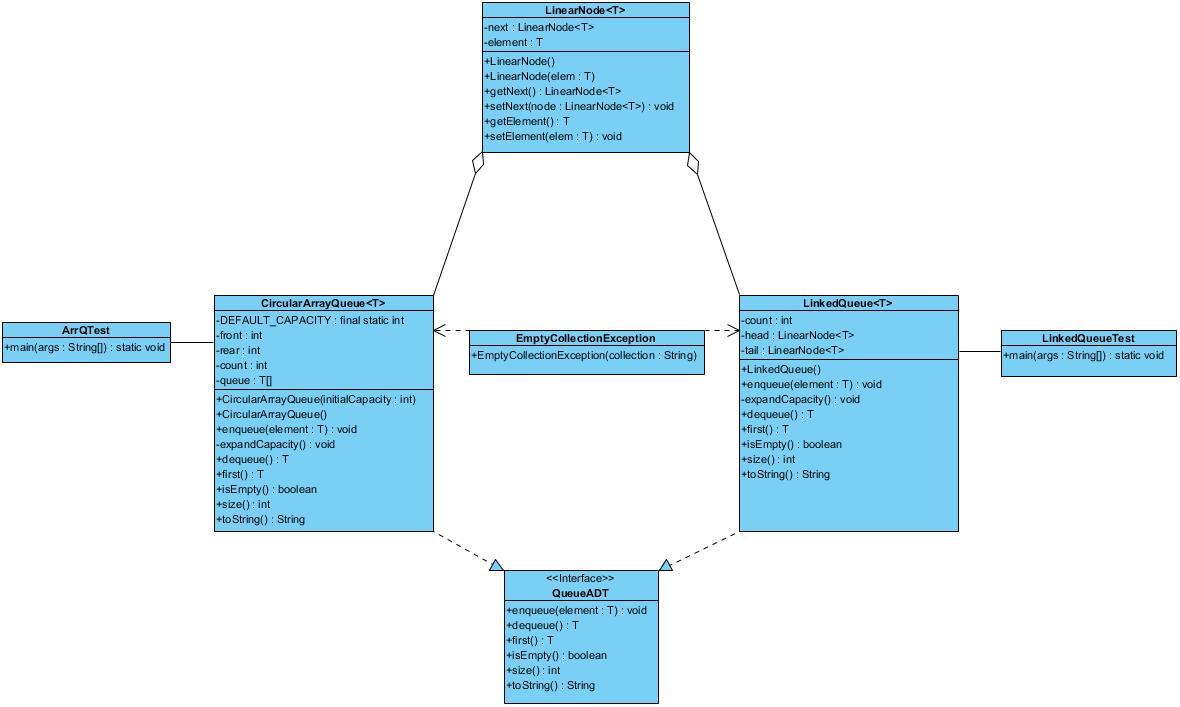


**COMP 2230 – Data Structures and Algorithm Analysis**

Assignment #4: Queues

## Due Date: Section 01 Oct 3rd, Section 02 Oct 4th

**Chapter 14: coding:**

****

**Problem 1**:

Implementation of the first(), size(), isEmpty(), and toString() methods into the LinkedQueue class.

**Problem 2**:

Implementation of the first(), size(), isEmpty(), and toString() methods into the CircularArrayQueue class.

**Problem #1 Code**

|  |
| --- |
| **LinkedQueue.java**  package Ass4\_2230;  import Ass4\_2230.exceptions.\*;  /\*\*  \* LinkedQueue represents a linked implementation of a queue.  \*  \* @author Java Foundations  \* @version 4.0  \*/  public class LinkedQueue<T> implements QueueADT<T>  {  private int count;  private LinearNode<T> head, tail;  /\*\*  \* Creates an empty queue.  \*/  public LinkedQueue()  {  count = 0;  head = tail = null;  }  /\*\*  \* Adds the specified element to the tail of this queue.  \* @param element the element to be added to the tail of the queue  \*/  public void enqueue(T element)  {  LinearNode<T> node = new LinearNode<T>(element);  if (isEmpty())  head = node;  else  tail.setNext(node);  tail = node;  count++;  }  /\*\*  \* Removes the element at the head of this queue and returns a  \* reference to it.  \* @return the element at the head of this queue  \* @throws EmptyCollectionException if the queue is empty  \*/  public T dequeue() throws EmptyCollectionException  {  if (isEmpty())  throw new EmptyCollectionException("queue");  T result = head.getElement();  head = head.getNext();  count--;  if (isEmpty())  tail = null;  return result;  }  /\*\*  \* Returns a reference to the element at the head of this queue.  \* The element is not removed from the queue.  \* @return a reference to the first element in this queue  \* @throws EmptyCollectionsException if the queue is empty  \*/  public T first() throws EmptyCollectionException  {  if(isEmpty()){  throw new EmptyCollectionException("queue");  }  return head.getElement();  }  /\*\*  \* Returns true if this queue is empty and false otherwise.  \* @return true if this queue is empty  \*/  public boolean isEmpty()  {  // To be completed as a Programming Project    return count == 0;  }  /\*\*  \* Returns the number of elements currently in this queue.  \* @return the number of elements in the queue  \*/  public int size()  {  // To be completed as a Programming Project    return count;  }  /\*\*  \* Returns a string representation of this queue.  \* @return the string representation of the queue  \*/  public String toString()  {  // To be completed as a Programming Project  String result = "";  LinearNode<T> current = head;  for (int i = 1; i <= count; i++) {  result += current.getElement() + ",";  current = current.getNext();  }  return result;  }  } |

|  |
| --- |
| **LinkedQueueTest.java**  package Ass4\_2230;  import Ass2\_2230.exceptions.EmptyCollectionException;  public class LinkedQueueTest {  public static void main(String[] args){  LinkedQueue<Integer> linkQ = new LinkedQueue<>();  //empty stack initialization  System.out.println("empty Queue: Front <- " + linkQ.toString() + " <- Rear");  for(int i = 1; i < 6; i++){  linkQ.enqueue(i);  }  System.out.println("Filled Queue: " + "Front <- " + linkQ.toString() + " <- Rear");  //test dequeue and first  System.out.println("-----dequeue() & first() Test-----");  System.out.println("first Value: " + linkQ.first());  for (int i = 1; i < 5; i++) {  linkQ.dequeue();  }  System.out.println("Queue after dequeue test: " + "Front <- " + linkQ.toString() + " <- Rear");  System.out.println("Top Value after dequeue test: " + linkQ.first());  linkQ.dequeue();  //test first with empty method  System.out.println("-----first() with empty stack test-----");  try{  linkQ.first();  } catch (EmptyCollectionException e) {  System.out.println("first() throws empty collection exception correctly");  }  //test dequeue with empty stack  System.out.println("-----dequeue() with empty stack test-----");  try {  linkQ.dequeue();  } catch (EmptyCollectionException e) {  System.out.println("dequeue() throws empty collection exception correctly");  }  System.out.println("Is the queue empty: " + linkQ.isEmpty());  System.out.println("queue size: " + linkQ.size());  }  } |

**Problem #1 Test Output**

|  |
| --- |
|  |

**Problem #2 Code**

|  |
| --- |
| **CircularArrayQueue.java**  package Ass4\_2230;  import Ass4\_2230.exceptions.\*;  import java.util.Arrays;  /\*\*  \* CircularArrayQueue represents an array implementation of a queue in  \* which the indexes for the front and rear of the queue circle back to 0  \* when they reach the end of the array.  \*  \* @author Java Foundations  \* @version 4.0  \*/  public class CircularArrayQueue<T> implements QueueADT<T>  {  private final static int DEFAULT\_CAPACITY = 100;  private int front, rear, count;  private T[] queue;  /\*\*  \* Creates an empty queue using the specified capacity.  \* @param initialCapacity the initial size of the circular array queue  \*/  public CircularArrayQueue(int initialCapacity)  {  front = rear = count = 0;  queue = (T[]) (new Object[initialCapacity]);  }  /\*\*  \* Creates an empty queue using the default capacity.  \*/  public CircularArrayQueue()  {  this(DEFAULT\_CAPACITY);  }  /\*\*  \* Adds the specified element to the rear of this queue, expanding  \* the capacity of the queue array if necessary.  \* @param element the element to add to the rear of the queue  \*/  public void enqueue(T element)  {  if (size() == queue.length)  expandCapacity();  queue[rear] = element;  rear = (rear + 1) % queue.length;  count++;  }  /\*\*  \* Creates a new array to store the contents of this queue with  \* twice the capacity of the old one.  \*/  private void expandCapacity()  {  T[] larger = (T[]) (new Object[queue.length \* 2]);  for (int scan = 0; scan < count; scan++)  {  larger[scan] = queue[front];  front = (front + 1) % queue.length;  }  front = 0;  rear = count;  queue = larger;  }  /\*\*  \* Removes the element at the front of this queue and returns a  \* reference to it.  \* @return the element removed from the front of the queue  \* @throws EmptyCollectionException if the queue is empty  \*/  public T dequeue() throws EmptyCollectionException  {  if (isEmpty())  throw new EmptyCollectionException("queue");  T result = queue[front];  queue[front] = null;  front = (front + 1) % queue.length;  count--;  return result;  }  /\*\*  \* Returns a reference to the element at the front of this queue.  \* The element is not removed from the queue.  \* @return the first element in the queue  \* @throws EmptyCollectionException if the queue is empty  \*/  public T first() throws EmptyCollectionException  {  // To be completed as a Programming Project    return queue[front];  }  /\*\*  \* Returns true if this queue is empty and false otherwise.  \* @return true if this queue is empty  \*/  public boolean isEmpty()  {  // To be completed as a Programming Project    return count == 0;  }  /\*\*  \* Returns the number of elements currently in this queue.  \* @return the size of the queue  \*/  public int size()  {  // To be completed as a Programming Project    return count;  }  /\*\*  \* Returns a string representation of this queue.  \* @return the string representation of the queue  \*/  public String toString()  {  // To be completed as a Programming Project    return Arrays.toString(queue);  }  } |

|  |
| --- |
| **ArrQTest.java**  package Ass4\_2230;  import Ass2\_2230.exceptions.EmptyCollectionException;  public class ArrQTest {  public static void main(String[] args) {  CircularArrayQueue<Integer> arrq = new CircularArrayQueue<>(5);  // Initialization with null values and capacity 5  System.out.println("Current Queue: Front <- " + arrq.toString() + " <- Rear");  // Populate the queue to fill the initial capacity  for (int i = 1; i <= 5; i++) {  arrq.enqueue(i);  }  System.out.println("Current Queue: Front <- " + arrq.toString() + " <- Rear");  // Test isEmpty and size  System.out.println("Is the queue empty: " + arrq.isEmpty());  System.out.println("Queue size: " + arrq.size());  // Test expandCapacity() method  System.out.println("-----expandCapacity() Test-----");  for (int i = 6; i <= 10; i++) {  arrq.enqueue(i);  }  System.out.println("Current Queue: Front <- " + arrq.toString() + " <- Rear");  // Test dequeue() and first() methods  System.out.println("-----dequeue() & first() Test-----");  System.out.println("First Value before dequeue: " + arrq.first());  System.out.println("Queue before dequeue: Front <- " + arrq.toString() + " <- Rear");  for (int i = 0; i < 5; i++) {  arrq.dequeue();  }  System.out.println("First Value after dequeue half the queue: " + arrq.first());  System.out.println("Queue after dequeue half the queue: Front <- " + arrq.toString() + " <- Rear");  for (int i = 0; i < 5; i++) {  arrq.dequeue();  }  System.out.println("First Value after dequeue the entire queue: " + arrq.first());  System.out.println("Queue after dequeue the entire queue: Front <- " + arrq.toString() + " <- Rear");  // Test first() with empty queue  System.out.println("-----first() with empty queue test-----");  try {  arrq.first();  } catch (EmptyCollectionException ece) {  System.out.println("first() throws empty collection exception correctly");  }  // Test dequeue() with empty queue  System.out.println("-----dequeue() with empty queue test-----");  try {  arrq.dequeue();  } catch (EmptyCollectionException ece) {  System.out.println("dequeue() throws empty collection exception correctly");  }  // Final checks  System.out.println("Is the queue empty: " + arrq.isEmpty());  System.out.println("Queue size: " + arrq.size());  }  } |

**Problem #2 Test Output**

|  |
| --- |
|  |