**/\*\* Stack implementation using an array**

**\* Created by: Matthew Casiro**

**\* Created on: April 09 2016 \*/**

package lab05;

import java.nio.BufferOverflowException;

import java.util.EmptyStackException;

import java.util.StringJoiner;

class Stack<T> {

protected int top = -1;

protected T[] stack;

**/\*\***

**\* Construct a stack object with <i>size</i> capacity.**

**\* Pre: N/A**

**\* Post: N/A**

**\* @param size is the number of elements the stack can hold**

**\* @throws IllegalArgumentException if size is less than 1 \*/**

public Stack(int size) throws IllegalArgumentException {

if (size < 1) {

throw new IllegalArgumentException();

}

stack = (T[]) new Object[size];

}

**/\*\***

**\* Check if the stack is empty.**

**\* Pre: N/A**

**\* Post: N/A**

**\* @return True if the stack is empty, otherwise False. \*/**

public boolean isEmpty() {

return top < 0;

}

**/\*\***

**\* Push <i>elem</i> on to the top of the stack.**

**\* Pre: The stack object is not full**

**\* Post: An element is added to the top of the stack**

**\* @param elem is the element to be added**

**\* @throws BufferOverflowException if pushing to a full stack \*/**

public void push(T elem) throws BufferOverflowException {

if (top >= stack.length - 1) {

throw new BufferOverflowException();

}

stack[++top] = elem;

} **/\*\***

**\* Pop the top element off of the stack.**

**\* Pre: The stack is not empty**

**\* Post: The top element of the stack is removed**

**\* @throws EmptyStackException \*/**

public void pop() throws EmptyStackException {

if (isEmpty()) {

throw new EmptyStackException();

}

top--;

}

**/\*\***

**\* Return the element at the top of the stack.**

**\* Pre: N/A**

**\* Post: N/A**

**\* @return the top element of the stack**

**\* @throws EmptyStackException if the stack is empty \*/**

public T top() throws EmptyStackException {

if (isEmpty()) {

throw new EmptyStackException();

}

return stack[top];

}

**/\*\***

**\* Build a string of stack elements in top down order.**

**\* Pre: N/A**

**\* Post: N/A**

**\* @return a comma separated string of stack elements \*/**

@Override

public String toString() {

StringJoiner str = new StringJoiner(",");

for (int i = top; i >= 0; i--) {

str.add(stack[i].toString());

}

return str.toString();

}

}

**/\*\* Queue implementation using a linked list with sentinel nodes**

**\* Created by: Matthew Casiro**

**\* Created on: April 09 2016 \*/**

package lab05;

import java.nio.BufferUnderflowException;

import java.util.StringJoiner;

class Queue<T> {

**/\*\* QueueNode containing an element and a pointer to the next element. \*/**

class QueueNode<T> {

T elem;

QueueNode<T> next;

**/\*\* Construct a QueueNode with element <i>elem</i> pointing to <i>next</i>.**

**\* Pre: N/A**

**\* Post: N/A**

**\* @param elem is the element the node contains**

**\* @param next is a pointer to the next node in the list \*/**

QueueNode(T elem, QueueNode<T> next) {

this.elem = elem;

this.next = next;

}

}

protected QueueNode<T> head, tail;

**/\*\* Construct a queue object**

**\* Pre: N/A**

**\* Post: N/A \*/**

public Queue() {

}

**/\*\* Check if the Queue is empty.**

**\* Pre: N/A**

**\* Post: N/A**

**\* @return True if queue is empty, and False otherwise \*/**

public boolean isEmpty() {

return head == null;

}

**/\*\* Add an element <i>elem</i> to the Queue.**

**\* Pre: N/A**

**\* Post: <i>elem</i> is added to the back of the queue**

**\* @param elem is the element to be added to the queue \*/**

public void enqueue(T elem) {

if(isEmpty()) {

head = tail = new QueueNode(elem, null);

return;

}

tail.next = new QueueNode(elem, null);

tail = tail.next;

} **/\*\* Remove an element from the Queue.**

**\* Pre: The queue is not empty**

**\* Post: The element at the front of the queue is removed**

**\* @throws BufferUnderflowException if queue is empty \*/**

public void dequeue() throws BufferUnderflowException {

if (isEmpty()) {

throw new BufferUnderflowException();

}

head = head.next;

if (isEmpty()) {

tail = null;

}

}

**/\*\* Return the first element in the Queue.**

**\* Pre: The queue is not empty**

**\* Post: N/A**

**\* @return the element at the front of the queue**

**\* @throws BufferUnderflowException if queue is empty \*/**

public T first() throws BufferUnderflowException {

if (isEmpty()) {

throw new BufferUnderflowException();

}

return head.elem;

}

**/\*\* Build a string of Queue elements in first to last order.**

**\* Pre: N/A**

**\* Post: N/A**

**\* @return a comma separated list of queue elements \*/**

@Override

public String toString() {

StringJoiner str = new StringJoiner(",");

QueueNode current = head;

while (current != null) {

str.add(current.elem.toString());

current = current.next;

}

return str.toString();

}

}**STACK**

**@Test**

**public void testIsEmpty() {**

System.out.println("isEmpty");

Stack s = new Stack(4);

assertEquals(true, s.isEmpty());

s.push(1);

assertEquals(false, s.isEmpty());

s.pop();

assertEquals(true, s.isEmpty());

}

**@Test**

**public void testPush() {**

System.out.println("push and top");

Stack s = new Stack(2);

s.push(5);

assertEquals("push to empty stack", "5", s.toString());

assertEquals("first element is on top", 5, s.top());

s.push(6);

assertEquals("non-empty stack", "6,5", s.toString());

assertEquals("second element is on top", 6, s.top());

System.out.println("push to full stack");

exception.expect(BufferOverflowException.class);

s.push(7);

}

**@Test**

**public void testPop() {**

System.out.println("pop and top");

Stack s = new Stack(2);

s.push("A");

s.push("B");

assertEquals("top element before pop", "B", s.top());

s.pop();

assertEquals("top element after pop", "A", s.top());

s.pop();

assertEquals("last element", true, s.isEmpty());

exception.expect(EmptyStackException.class);

s.pop();

}

**@Test**

**public void testTop() {**

System.out.println("top");

Stack s = new Stack(4);

exception.expect(EmptyStackException.class);

System.out.println(s.top());

}

**QUEUE**

**@Test**

**public void testIsEmpty() {**

System.out.println("isEmpty");

Queue q = new Queue();

assertEquals("test empty queue", true, q.isEmpty());

q.enqueue("A");

assertEquals("non-empty queue", false, q.isEmpty());

q.dequeue();

assertEquals("test emptied queue", true, q.isEmpty());

}

**@Test**

**public void testEnqueue() {**

System.out.println("enqueue and first");

Queue q = new Queue();

q.enqueue("A");

assertEquals(“to empty queue", "A", q.first());

q.enqueue("B");

assertEquals("to non-empty queue", "A", q.first());

assertEquals("test queue order", "A,B", q.toString());

}

**@Test**

**public void testDequeue() {**

System.out.println("dequeue and first");

Queue q = new Queue();

q.enqueue("A");

q.enqueue("B");

q.dequeue();

assertEquals("test dequeue order", "B", q.first());

q.dequeue();

exception.expect(BufferUnderflowException.class);

q.dequeue();

}

@Test

public void testFirst() {

System.out.println("first");

Queue q = new Queue();

exception.expect(BufferUnderflowException.class);

System.out.println(q.first());

}