Lab105

Steven Glasford

2-22-2019

1 Client.java

```
/**
 * A main class for the program.
 *
 * @author Steven Glasford
 */
public class Client {

    /**
     * A main class for the program.
     * @param args none
     */
    public static void main(String[] args) {

        int power = 8;

        long[][] aStackTimes = new long[power][3];
        long[][] lStackTimes = new long[power][3];
        long[][] aQueueTimes = new long[power][3];
        long[][] lQueueTimes = new long[power][3];
        long[][] aListTimes = new long[power][3];

        //create an ArrayStack
        ArrayStack astack = new ArrayStack(100000000);
        LinkedStack lstack = new LinkedStack();
        ArrayQueue aqueue = new ArrayQueue(10000000);
        LinkedQueue lqueue = new LinkedQueue();
        ArrayList alist = new ArrayList(100000000);

        //start a timer
        long start, stop;


        // a for loop to see what the push and pop times are for a thing
        for (int i = 0; i < power; i++){

            aStackTimes[i][0] = (long) Math.pow(10,(i));
            //Use the array stack method first
            start = System.nanoTime();



            //insert the number 0 ten times
            for (int j = 0; j < (Math.pow(10,(i))); j++){
                astack.push(0);
            }
            //end timer
            stop = System.nanoTime();
            //save the time into the first part of the 2 d array
```

```
        aStackTimes[i][1] = (long) (stop - start);

        start = System.nanoTime();
        //remove the thing
        for (int j = 0; j < Math.pow(10,(i)); j++){
            astack.pop();
        }
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        aStackTimes[i][2] = (long) (stop - start);
    }


    //for the linked stack
    for (int i = 0; i < power; i++){

        //Use the linked stack method first
        lStackTimes[i][0] = (long) Math.pow(10,(i));
        start = System.nanoTime();
        //insert the number 0 ten times
        for (int j = 0; j < (Math.pow(10,(i ))); j++){
            lstack.push(0);
        }
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        lStackTimes[i][1] = (long) (stop - start);

        start = System.nanoTime();
        //remove the thing
        for (int j = 0; j < Math.pow(10,(i)); j++){
            lstack.pop();
        }
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        lStackTimes[i][2] = (long) (stop - start);
    }


    //for the ArrayQueue
    for (int i = 0; i < power; i++){

        aQueueTimes[i][0] = (long) Math.pow(10,(i));
        //Use the array queue method first
        start = System.nanoTime();
        //insert the number 0 ten times
        for (int j = 0; j < (Math.pow(10,(i ))); j++){
            aqueue.enqueue(0);
        }
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        aQueueTimes[i][1] = (long) (stop - start);

        start = System.nanoTime();
        //remove the thing
        for (int j = 0; j < Math.pow(10,(i)); j++){
            aqueue.dequeue();
        }
```

```
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        aQueueTimes[i][2] = (long) (stop - start);
    }

    for (int i = 0; i < power; i++){
        //Use the linked queue method first
        lQueueTimes[i][0] = (long) Math.pow(10,(i));

        start = System.nanoTime();
        //insert the number 0 ten times
        for (int j = 0; j < (Math.pow(10,(i))); j++){
            lqueue.enqueue(0);
        }
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        lQueueTimes[i][1] = (long) (stop - start);

        start = System.nanoTime();
        //remove the thing
        for (int j = 0; j < Math.pow(10,(i)); j++){
            lqueue.dequeue();
        }
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        lQueueTimes[i][2] = (long) (stop - start);
    }

    //For the lists
    for (int i = 0; i < power; i++){
        //Use the linked queue method first
        aListTimes[i][0] = (long) Math.pow(10,(i));

        start = System.nanoTime();
        //insert the number 0 ten times
        for (int j = 0; j < (Math.pow(10,(i))); j++){
            alist.add(0);
        }
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        aListTimes[i][1] = (long) (stop - start);

        start = System.nanoTime();
        //remove the thing
        for (int j = 0; j < Math.pow(10,(i)); j++){
            alist.remove();
        }
        //end timer
        stop = System.nanoTime();
        //save the time into the first part of the 2 d array
        aListTimes[i][2] = (long) (stop - start);
    }

    AsciiTableStack(aStackTimes,"ArrayStack");

    System.out.println("\n");
```

```
        AsciiTableStack(lStackTimes,"LinkStack");

        System.out.println("\n");

        AsciiTableQueue(aQueueTimes,"ArrayQueue");

        System.out.println("\n");

        AsciiTableQueue(lQueueTimes, "LinkedQueue");

        System.out.println("\n");

        AsciiTableList(aListTimes, "ArrayList");
    }

    //use the a standard of 80 character max for the table, because of latex
    public static void AsciiTableStack(long[][] data, String title){
        String table = String.format( "+%-77s+", "").replace(" ", "-");

        table = table + "\n|" + center(title,77) + "|\n";

        table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");

        table = table + "|" + center("N",12) + "|" +
                    center("Push", 32) + "|" +
                    center("Pop", 31) + "|\n";

        table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");

        for (long[] data1 : data) {
            table = table + "|" + center(Long.toString(data1[0]),12) + "|" +
                    center(Long.toString(data1[1]), 32) + "|" +
                    center(Long.toString(data1[2]), 31) + "|\n";
            table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");
        }


        System.out.println(table);

        return;
    }

    public static void AsciiTableQueue(long[][] data, String title){
        String table = String.format( "+%-77s+", "").replace(" ", "-");

        table = table + "\n|" + center(title,77) + "|\n";

        table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");

        table = table + "|" + center("N",12) + "|" +
                    center("Enqueue", 32) + "|" +
                    center("Dequeue", 31) + "|\n";
```

```
        table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");

        for (long[] data1 : data) {
            table = table + "|" + center(Long.toString(data1[0]),12) + "|" +
                    center(Long.toString(data1[1]), 32) + "|" +
                    center(Long.toString(data1[2]), 31) + "|\n";
            table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");
        }


        System.out.println(table);

        return;
    }

    public static void AsciiTableList(long[][] data, String title){
        String table = String.format( "+%-77s+", "").replace(" ", "-");

        table = table + "\n|" + center(title,77) + "|\n";

        table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");

        table = table + "|" + center("N",12) + "|" +
                    center("Add", 32) + "|" +
                    center("Remove", 31) + "|\n";

        table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");

        //add the data from the array into the table
        for (long[] data1 : data) {
            table = table + "|" + center(Long.toString(data1[0]),12) + "|" +
                    center(Long.toString(data1[1]), 32) + "|" +
                    center(Long.toString(data1[2]), 31) + "|\n";
            table = table + String.format( "+%-12s+", "").replace(" ", "-")
                    + String.format( "%-32s+", "").replace(" ", "-")
                    + String.format( "%-31s+\n", "").replace(" ", "-");
        }


        System.out.println(table);

        return;
    }

    /**
     * Take a string and center it within a certain amount of given space.
     * @param text  The text you want to center
     * @param len   The possible space you want to center within
     * @return
     */
    public static String center(String text, int len){
        if (len <= text.length())
            return text.substring(0, len);
```

```
        int before = (len - text.length())/2;
        if (before == 0)
            return String.format("%-" + len + "s", text);
        int rest = len - before;
        return String.format("%" + before + "s%-" + rest + "s", "", text);
    }
}
```

2 ArrayList.java

```java
/**
 * Data Structures & Algorithms 6th Edition
 * Goodrick, Tamassia, Goldwasser
 * Code Fragments 7.2, 7.3, 7.4 and 7.5
 *
 * An implementation of a simple ArrayList class.
 * */
public class ArrayList<E> implements List<E> {
    //instance variables
        //default array capacity
    public static final int CAPACITY = 16;
    //generic array used for storage
    private E[] data;
    //current number of elements
    private int size = 0;

    //constructors
        //constructs list with default capacity
    public ArrayList() {this(CAPACITY);}
    //constructs list with given capacity
    public ArrayList(int capacity){
        //safe cast; compiler may give warning
        data = (E[]) new Object[capacity];
    }

    //public methods
    /**
     * Returns the number of elements in the array list.
     */
    @Override
    public int size() {return size;}

    /**
     * Returns whether the array list is empty.
     */
    @Override
    public boolean isEmpty() {return size == 0;}

    /**
     * Returns (but does not remove) the element at index i.
     */
    @Override
    public E get(int i) throws IndexOutOfBoundsException {
        checkIndex(i, size);
        return data[i];
    }

    /**
     * Replaces the element at index i with e, and returns the replaced
     * element.
     */
    @Override
    public E set(int i, E e) throws IndexOutOfBoundsException {
        checkIndex(i, size);
        E temp = data[i];
        data[i] = e;
        return temp;
    }
```

```
/**
 * Inserts element e to be at index i, shifting all subsequent
 * elements later.
 */
@Override
public void add(int i, E e) throws IndexOutOfBoundsException,
        IllegalStateException {
    checkIndex(i, size + 1);
    //not enough capacity
    if (size == data.length)
        //so double the current capacity
        resize(2*data.length);
    //start by shifting rightmost
    for (int k = size - 1; k >= i; k--)
        data[k+1] = data[k];
    //ready to place the new element
    size++;
}

/**
 * Removes/returns the element at index i, shifting subsequent
 * elements earlier
 */
@Override
public E remove(int i) throws IndexOutOfBoundsException{
    checkIndex(i, size);
    E temp = data[i];
    //shift elements to fill hole
    for (int k = i; k < size - 1; k++)
        data[k] = data[k+1];
    //help gargbage collection
    data[size-1] = null;
    size--;
    return temp;
}

//remove at the end of the thing
public E remove(){
    return remove(size-1);
}

//add at the end of the thing
public void add(E e){
    add(size,e);
}

//utility methods

/**
 * Checks whether the given index is in the range [0, n-1].
 * @param i
 * @param n
 */
protected void checkIndex(int i, int n) throws IndexOutOfBoundsException {
    if (i < 0 || i >= n)
        throw new IndexOutOfBoundsException("Illegal index: " + i);
}

/**
 * Resizes internal array to have given capacity >= size.
 * @param capacity
```

```
     */
    protected void resize(int capacity){
        //safe cast; compiler may give warning
        E[] temp = (E[]) new Object[capacity];
        for (int k = 0; k < size; k++)
            temp[k] = data[k];
        //start using the new array
        data = temp;
    }

}
```

3 ArrayQueue.java

```java
/**
 * Implementation of the queue ADT using a fixed-length array
 *
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwater
 * @author Steven Glasford
 * @version 2-21-2019
 * @param <E>
 * @todo figure out why the CAPACITY thing doesn't work
 */
public class ArrayQueue<E> implements Queue<E>{
    //instance variables
        //generic array used for storage
    private E[] data;
    //index of the front element
    private int f = 0;
    //current number of elements
    private int sz = 0;
    //default array capacity
    public static final int CAPACITY = 1000;

    //constructors
        //constructs queue with given default capacity
    public ArrayQueue() {this(CAPACITY);}
    //constructs queue with given capacity
    public ArrayQueue(int capacity){
        data = (E[]) new Object[capacity];
    }

    //methods

    /**
     * Returns the number of elements in the queue.
     */
    public int size() {return sz;}

    /**
     * Tests whether the queue is empty.
     */
    public boolean isEmpty() {return (sz == 0);}

    /**
     * Inserts an element at the rear of the queue.
     */
    public void enqueue(E e) throws IllegalStateException {
        if (sz == data.length) throw new IllegalStateException("Queue is full");
        //use modular arithmetic
        int avail = (f + sz) % data.length;
        data[avail] = e;
        sz++;
    }

    /**
     * Returns, but does not remove, the first element of the queue
     * (null if empty).
     */
    public E first() {
        if (isEmpty()) return null;
```

```
        return data[f];
    }

    /**
     * Removes and returns the first element of the queue (null if empty).
     */
    public E dequeue() {
        if (isEmpty()) return null;
        E answer = data[f];
        //derefence to help garbage colection
        data[f] = null;
        f = (f + 1) % data.length;
        sz--;
        return answer;
    }
}
```

4 List.java

```java
/**
 * A simplified version of the "java.util.List" interface
 *
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwater
 * @author Steven Glasford
 * @version 2-21-2019
 * @param <E>
 */

public interface List<E> {
    /**
     * Returns the number of elements in this list.
     * @return
     */
    int size();

    /**
     * Returns whether the list is empty
     * @return
     */
    boolean isEmpty();

    /**
     * Returns (but does not remove) the element at index i.
     * @param i
     * @return
     */
    E get(int i) throws IndexOutOfBoundsException;

    /**
     * Replaces the element at index i with e, and returns the replaced
     * element.
     * @param i
     * @param e
     * @return
     */
    E set(int i, E e) throws IndexOutOfBoundsException;

    /**
     * Inserts element e to be at index i, shifting all subsequent
     * elements later.
     * @param i
     * @param e
     */
    void add(int i, E e) throws IndexOutOfBoundsException;

    /**
     * Removes/returns the element at index i, shifting subsequent
     * elements earlier.
     * @param i
     * @return
     */
    E remove(int i) throws IndexOutOfBoundsException;
}
```

5 LinkedQueue.java

```java
/**
 * Realization of a FIFO queue as an implementation of a SinglyLinkedSet.
 *
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwater
 * @author Steven Glasford
 * @version 2-21-2019
 * @param <E>
 */


public class LinkedQueue<E> implements Queue<E> {
    //an empty list
    private final SinglyLinkedList<E> list = new SinglyLinkedList<>();
    //new queue relies on the initially empty list
    public LinkedQueue() {}

    @Override
    public int size() {return list.size();}

    @Override
    public boolean isEmpty() {return list.isEmpty();}

    @Override
    public void enqueue(E element) {list.addLast(element);}

    @Override
    public E first() {return list.first();}

    @Override
    public E dequeue() {return list.removeFirst();}

}
```

6 Queue.java

```java
/**
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwater
 * @author Steven Glasford
 * @version 2-21-2019
 * @param <E>
 */

public interface Queue<E> {
    /**
     * Returns the number of elements in the queue
     * @return
     */
    int size();

    /**
     * Tests whether the queue is empty
     * @return
     */
    boolean isEmpty();

    /**
     * Inserts an element at the rear of the queue
     * @param e
     * @todo    modify so that this is required to throw a queue Full Exception
     *          if called on a full queue
     */
    void enqueue(E e);

    /**
     * returns, but does not remove, the first element of the queue
     * (null if empty).
     * @return
     */
    E first();

    /**
     * Removes and returns the first element of the queue (null if empty)
     * @return
     */
    E dequeue();
}
```

7 ArrayStack.java

```java
/**
 *
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwater
 * @author Steven Glasford
 * @version 2-21-2019
 * @param <E>
 */
public class ArrayStack<E> implements Stack<E> {

    //default array capacity
    public static final int CAPACITY = 1000;
    //generic array used for storage
    private E[] data;
    //index of the top element in the stack
    private int t = -1;

    //constructs stack with default capacity
    public ArrayStack() {this(CAPACITY);}

    //constructs stack with given capacity
    public ArrayStack(int capacity){
        //safe cast; compiler may give warning
        data = (E[]) new Object[capacity];
    }

    @Override
    public int size() {return (t+1);}

    @Override
    public boolean isEmpty() {return (t==-1);}

    @Override
    public void push(E e) throws IllegalStateException {
        if (size() == data.length) throw new IllegalStateException("Stack "
                + "is full");
        //increment t before storing new item
        data[++t] = e;
    }

    @Override
    public E top(){
        if (isEmpty()) return null;
        return data[t];
    }

    @Override
    public E pop() {
        if (isEmpty()) return null;
        E answer = data[t];
        //dereference to help garbage collection
        data[t] = null;
        t--;
        return answer;
    }


}
```

8 LinkedStack.java

```java
/**
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwater
 * @author Steven Glasford
 * @version 2-21-2019
 * @param <E>
 */
public class LinkedStack<E> implements Stack<E> {
    //an empty list
    private final SinglyLinkedList<E> list = new SinglyLinkedList<>();

    //new stack relies on the initially empty list
    public LinkedStack() {}

    @Override
    public int size() {return list.size();}

    @Override
    public boolean isEmpty() {return list.isEmpty();}

    @Override
    public void push(E element) { list.addFirst(element); }

    @Override
    public E top() { return list.first(); }

    @Override
    public E pop() { return list.removeFirst(); }
}
```

9 Stack.java

```java
/**
 * A collection of objects that are inserted and removed according to the
 * last-in first-out principle; although similar in purpose, this
 * interface differs from "java.util.Stack"
 *
 * @author Michael T. Goodrich
 * @author Roberto Tamassia
 * @author Michael H. Goldwater
 *
 * @version 2-21-2019s
 */
public interface Stack<E> {
    /**
     * Returns the number of elements in the stack
     * @return number of elements in the stack
     */
    int size();

    /**
     * Tests whether the stack is empty.
     * @return true if the stack is empty, false otherwise.
     */
    boolean isEmpty();

    /**
     * Inserts an element at the top of the stack
     * @param e the element to be inserted
     * @todo    modify so this method is required to trow a Stack Full
     *          exception if called on a full stack
     */
    void push(E e);

    /**
     * Returns, but does not remove , the element at the top of the stack
     * @return to element in the stack (or null if empty)
     */
    E top();

    /**
     * Removes and returns the top element from the stack.
     * @return element removed (or null if empty)
     */
    E pop();
}
```

10 SinglyLinkedList.java

```
/**
 *
 * SinglyLinkedList Class
 * Code Fragments 3.14, 3.15
 * from
 * Data Structures & Algorithms, 6th edition
 * by Michael T. Goodrich, Roberto Tamassia & Michael H. Goldwasser
 * Wiley 2014
 * Transcribed by
 * @author Steven Glasford
 * @version January 31, 2019
 * @param <E> a generic placeholder name
 */
public class SinglyLinkedList<E> {
    /**
     *
     * @param <E> a generic placeholder name
     *
     * A subclass creating the Node
     */
    private static class Node<E>{
        //reference to the element stored at this node
        private final E element;
        //reference to the subsequent node in the list
        private Node<E> next;
        public Node(E e, Node<E> n){
            element = e;
            next = n;
        }

        /**
         *
         * @return Return the current element
         */
        public E getElement(){return element;}

        /**
         *
         * @return return the address of the next item in the linked list
         */
        public Node<E> getNext() {return next;}

        /**
         *
         * @param n the next item in the list
         */
        public void setNext(Node<E> n) {next = n;}
    }

    //head node of the list (or null if empty)
    private Node<E> head = null;
    //last node of the list (or null if empty)
    private Node<E> tail = null;
    //number of nodes in the list
    private int count = 0;

    /**
     * constructs an initially empty list
     */
```

```
public SinglyLinkedList(){}

//access methods
/**
 *
 * @return Return the size of the linked list
 */
public int size() {return count;}

/**
 *
 * @return Determine if the linked list is empty
 */
public boolean isEmpty() {return count == 0;}

/**
 *
 * @return return the first element in the list
 *
 * returns (but does not remove) the first element
 */
public E first(){
    if (isEmpty()) return null;
    return head.getElement();
}

/**
 *
 * @return the last element in the linked list
 *
 * returns (but does not remove the last element
 */
public E last(){
    if (isEmpty()) return null;
    return tail.getElement();
}

//update methods

/**
 *
 * @param e A generic element
 *
 * adds element e to the front of the list
 */
public void addFirst(E e){
    //create and link a new node
    head = new Node<>(e, head);
    //special case: new node becomes tail also
    if (count == 0)
        tail = head;
    count++;
}

/**
 *
 * @param e A generic item
 *
 * adds element e to the end of the list
 */
public void addLast(E e) {
```

```
        //node will eventually be the tail
        Node<E> newest = new Node<>(e,null);
        //special case: previously empty list
        if (isEmpty())
            head = newest;
        else
            tail.setNext(newest);
        tail = newest;
        count++;
    }

    /**
     *
     * @return return the item that was removed
     *
     * removes and returns the first element
     */
    public E removeFirst(){
        //nothing to remove
        if (isEmpty()) return null;
        E answer = head.getElement();
        //will become null if list had only one node
        head = head.getNext();
        count--;
        //special case as list is now empty
        if(count == 0)
            tail = null;
        return answer;
    }
}
```

11 Output

ArrayStack		
N	Push	Pop
1	31908	3845
10	3600	2754
100	22330	15529
1000	427833	266389
10000	1463029	1140004
100000	11006404	5365249
1000000	69012080	1154951
10000000	23738758	10577428

LinkStack		
N	Push	Pop
1	475822	13075
10	3221	4011
100	32825	41070
1000	309093	448840
10000	277333	232481
100000	1850521	438892
1000000	15512319	4423880
10000000	3699213596	47636940

ArrayQueue		
N	Enqueue	Dequeue
1	41097	15727
10	2155	2221
100	14910	26210

1000	119292	191374
10000	218049	211432
100000	1136130	826973
1000000	9043308	7927018
10000000	91833727	82702689

LinkedQueue		
N	Enqueue	Dequeue
1	50168	7429
10	8595	993
100	30588	11056
1000	326901	83288
10000	547648	87950
100000	1665471	462619
1000000	8243608	4648521
10000000	2570566510	46642004

ArrayList		
N	Add	Remove
1	52038	8228
10	3717	2677
100	29048	31450
1000	291466	236959
10000	409196	405911
100000	847668	398307
1000000	5943577	3813028
10000000	56899101	38164740