Oct 17
2024

Assignment 5

COMP 2230_02

COLTON ISLES AND KAYLEE CROCKER

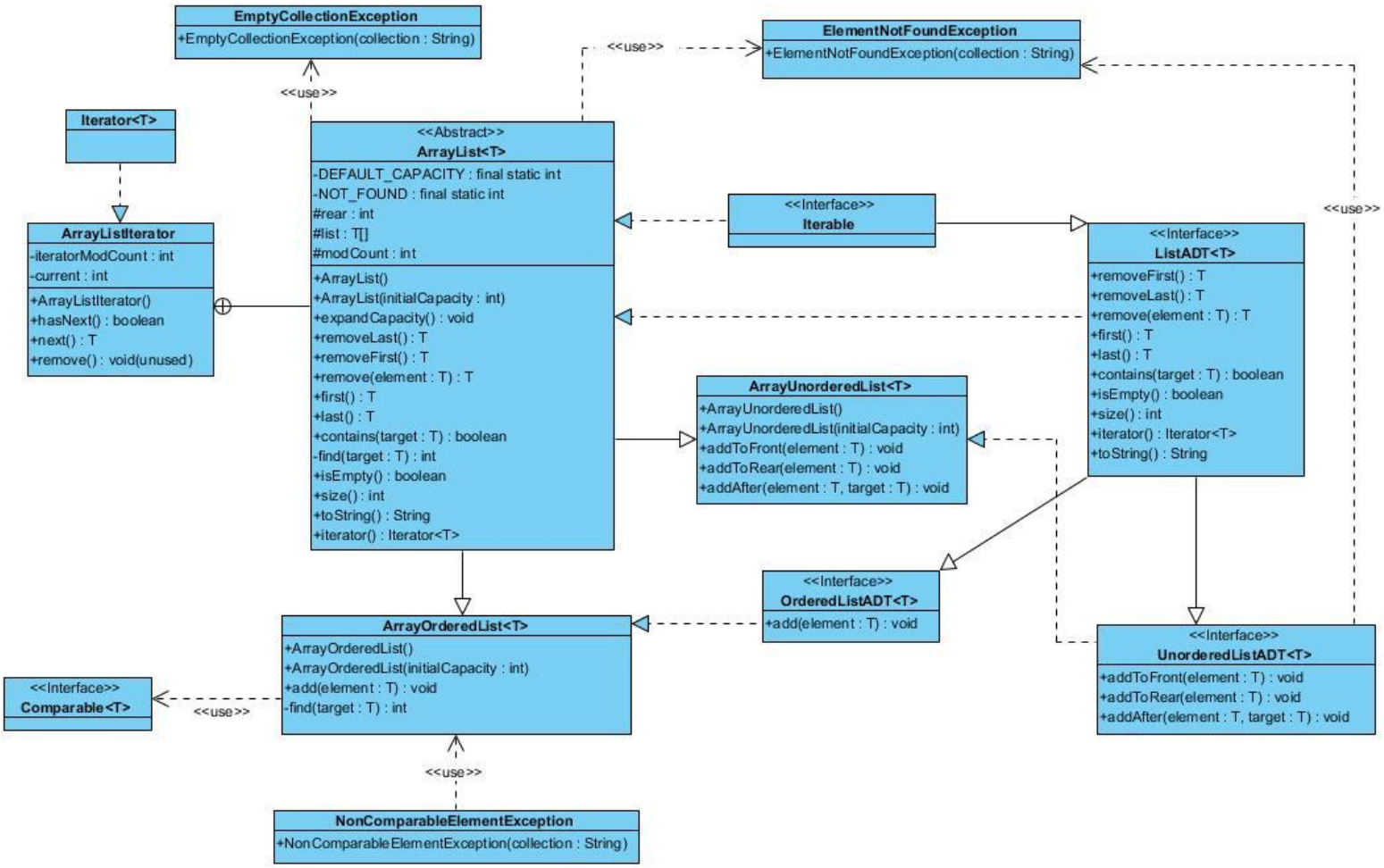
THOMPSON RIVERS UNIVERSITY

COMP 2230 – Data Structures and Algorithm Analysis

Assignment #4: Queues

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

Chapter 15



Problem #1 Code

```
package Ass5_2230;

import Ass5_2230.exceptions.*;
```

```
import java.util.*;

/**
 * ArrayList represents an array implementation of a list. The front of
 * the list is kept at array index 0. This class will be extended
 * to create a specific kind of list.
 *
 * @author Java Foundations
 * @version 4.0
 */
public abstract class ArrayList<T> implements ListADT<T>, Iterable<T>
{
    private final static int DEFAULT_CAPACITY = 100;
    private final static int NOT_FOUND = -1;

    protected int rear;
    protected T[] list;
    protected int modCount;

    /**
     * Creates an empty list using the default capacity.
     */
    public ArrayList()
    {
       this(DEFAULT_CAPACITY);
    }

    /**
     * Creates an empty list using the specified capacity.
     *
     * @param initialCapacity the integer value of the size of the array
list
     */
    public ArrayList(int initialCapacity)
    {
       rear = 0;
       list = (T[])(new Object[initialCapacity]);
       modCount = 0;
    }

    /**
     * Creates a new array to store the contents of this list with
     * twice the capacity of the old one. Called by descendant classes
     * that add elements to the list.
     */
    protected void expandCapacity()
    {
       list = Arrays.copyOf(list, 2 * list.length);
```

```
    }

    /**
     * Removes and returns the last element in this list.
     *
     * @return the last element in the list
     * @throws EmptyCollectionException if the element is not in the list
     */
    public T removeLast() throws EmptyCollectionException
    {
       if (isEmpty()) throw new EmptyCollectionException("ArrayList");

       rear--;
       modCount++;
       T result = list[rear];
       list[rear] = null;
       return result;
    }

    /**
     * Removes and returns the first element in this list.
     *
     * @return the first element in the list
     * @throws EmptyCollectionException if the element is not in the list
     */
    public T removeFirst() throws EmptyCollectionException {
       if (isEmpty()) throw new EmptyCollectionException("ArrayList");

       T result = list[0];
       //changes the structure of the list thus modCount--??
       modCount++;
       rear--;
       list = Arrays.copyOfRange(list, 1, list.length);

       return result;
    }
    /**
     * Removes and returns the specified element.
     *
     * @param  element the element to be removed and returned from the list
     * @return the removed element
     * @throws ElementNotFoundException if the element is not in the list
     */
    public T remove(T element)
    {
       T result;
       int index = find(element);
```

```
        if (index == NOT_FOUND)
           throw new ElementNotFoundException("ArrayList");

        result = list[index];
        rear--;

        // shift the appropriate elements
        for (int scan = index; scan < rear; scan++)
           list[scan] = list[scan+1];

        list[rear] = null;
        //changes the content of the list thus modCount++??
        modCount++;

        return result;
    }

    /**
     * Returns a reference to the element at the front of this list.
     * The element is not removed from the list.  Throws an
     * EmptyCollectionException if the list is empty.
     *
     * @return a reference to the first element in the list
     * @throws EmptyCollectionException if the list is empty
     */
    public T first() throws EmptyCollectionException
    {
        if (isEmpty()) throw new EmptyCollectionException("ArrayList");

        return list[0];
    }

    /**
     * Returns a reference to the element at the rear of this list.
     * The element is not removed from the list. Throws an
     * EmptyCollectionException if the list is empty.
     *
     * @return a reference to the last element of this list
     * @throws EmptyCollectionException if the list is empty
     */
    public T last() throws EmptyCollectionException
    {
        if (isEmpty()) throw new EmptyCollectionException("ArrayList");

        return list[rear - 1];
    }

    /**
```

```
     * Returns true if this list contains the specified element.
     *
     * @param target the target element
     * @return true if the target is in the list, false otherwise
     */
    public boolean contains(T target)
    {
       return (find(target) != NOT_FOUND);
    }

    /**
     * Returns the array index of the specified element, or the
     * constant NOT_FOUND if it is not found.
     *
     * @param target the target element
     * @return the index of the target element, or the
     *         NOT_FOUND constant
     */
    private int find(T target)
    {
       int scan = 0;
       int result = NOT_FOUND;

       if (!isEmpty())
          while (result == NOT_FOUND && scan < rear)
             if (target.equals(list[scan]))
                result = scan;
             else
                scan++;

       return result;
    }

    /**
     * Returns true if this list is empty and false otherwise.
     *
     * @return true if the list is empty, false otherwise
     */
    public boolean isEmpty()
    {
       return size() == 0;
    }

    /**
     * Returns the number of elements currently in this list.
     *
     * @return the number of elements in the list
     */
```

```
    public int size()
    {
       return rear;
    }

    /**
     * Returns a string representation of this list.
     *
     * @return the string representation of the list
     */
    public String toString()
    {
       return "Front -> " + Arrays.toString(list) + " <- Rear";
    }

    /**
     * Returns an iterator for the elements currently in this list.
     *
     * @return an iterator for the elements in the list
     */
    public Iterator<T> iterator()
    {
       return new ArrayListIterator();
    }

    /**
     * ArrayListIterator iterator over the elements of an ArrayList.
     */
    private class ArrayListIterator implements Iterator<T> {
       int iteratorModCount;
       int current;

       /**
        * Sets up this iterator using the specified modCount.
        *
        * @param modCount the current modification count for the ArrayList
        */
       public ArrayListIterator() {
          iteratorModCount = modCount;
          current = 0;
       }

       /**
        * Returns true if this iterator has at least one more element
        * to deliver in the iteration.
        *
        * @return true if this iterator has at least one more element to
deliver
```

```
         * in the iteration
         * @throws ConcurrentModificationException if the collection has
changed
         *                                         while the iterator is in
use
         */
        public boolean hasNext() throws ConcurrentModificationException {
           if (iteratorModCount != modCount)
              throw new ConcurrentModificationException();

           return (current < rear);
        }

        /**
         * Returns the next element in the iteration. If there are no
         * more elements in this iteration, a NoSuchElementException is
         * thrown.
         *
         * @return the next element in the iteration
         * @throws NoSuchElementException          if an element not found
exception occurs
         * @throws ConcurrentModificationException if the collection has
changed
         */
        public T next() throws ConcurrentModificationException {
           if (!hasNext())
              throw new NoSuchElementException();

           current++;

           return list[current - 1];
        }

        /**
         * The remove operation is not supported in this collection.
         *
         * @throws UnsupportedOperationException if the remove method is
called
         */
        public void remove() throws UnsupportedOperationException {
           throw new UnsupportedOperationException();
        }
     }
}
```

Problem #2 Code

```
package Ass5_2230;

import Ass5_2230.exceptions.*;

import java.util.Iterator;

/**
 * ArrayUnorderedList represents an array implementation of an unordered
list.
 *
 * @author Java Foundations
 * @version 4.0
 */
public class ArrayUnorderedList<T> extends ArrayList<T> implements
UnorderedListADT<T>
{
    /**
     * Creates an empty list using the default capacity.
     */
    public ArrayUnorderedList()
    {
       super();
    }

    /**
     * Creates an empty list using the specified capacity.
     *
     * @param initialCapacity the initial size of the list
     */
    public ArrayUnorderedList(int initialCapacity)
    {
       super(initialCapacity);
    }

    /**
     * Adds the specified element to the front of this list.
     *
     * @param element the element to be added to the front of the list
     */
    public void addToFront(T element)
    {
       if(size() == list.length){
          expandCapacity();
       }
       for(int i = rear; i > 0; i--){
```

```
            list[i] = list[i - 1];
        }
        list[0] = element;
        rear++;
        modCount++;
    }

    /**
     * Adds the specified element to the rear of this list.
     *
     * @param element the element to be added to the list
     */
    public void addToRear(T element)
    {
        if(size() == list.length){
            expandCapacity();
        }
        list[rear] = element;
        rear++;
        modCount++;
    }

    /**
     * Adds the specified element after the specified target element.
     * Throws an ElementNotFoundException if the target is not found.
     *
     * @param element the element to be added after the target element
     * @param target  the target that the element is to be added after
     */
    public void addAfter(T element, T target) throws
ElementNotFoundException
    {
        if (size() == list.length)
            expandCapacity();

        int scan = 0;

        // find the insertion point
        while (scan < rear && !target.equals(list[scan]))
            scan++;

        if (scan == rear)
            throw new ElementNotFoundException("UnorderedList");

        scan++;

        // shift elements up one
        for (int shift = rear; shift > scan; shift--)
```

```
            list[shift] = list[shift - 1];

        // insert element
        list[scan] = element;
        rear++;
        modCount++;
    }

}
```

Problem #2 Test Output

```
package Ass5_2230;
import Ass5_2230.exceptions.*;
public class UnorderedArrayListTest {
   public static void main(String[] args) {
      ArrayUnorderedList<String> list = new ArrayUnorderedList<String>(10);

      // Test addToRear method
      System.out.println("Testing addToRear method:");
      list.addToRear("Apple");
      list.addToRear("Banana");
      list.addToRear("Cherry");
      System.out.println("List after adding to rear: " + list);
      System.out.println("Size: " + list.size());
      System.out.println();

      // Test addToFront method
      System.out.println("Testing addToFront method:");
      list.addToFront("Dragon Fruit");
      System.out.println("List after adding to front: " + list);
      System.out.println();

      // Test addAfter method
      System.out.println("Testing addAfter method:");
      list.addAfter("Elderberry", "Banana");
      System.out.println("List after adding 'Elderberry' after 'Banana': " + list);
      System.out.println();

      // Test first and last methods
      System.out.println("Testing first and last methods:");
      System.out.println("First element: " + list.first());
      System.out.println("Last element: " + list.last());
      System.out.println();

      // Test remove methods
```

```
        System.out.println("Testing remove methods:");
        System.out.println("Removed first element: " + list.removeFirst());
        System.out.println("Removed last element: " + list.removeLast());
        System.out.println("List after removals: " + list);
        System.out.println();

        // Test contains method
        System.out.println("Testing contains method:");
        System.out.println("Contains 'Banana': " + list.contains("Banana"));
        System.out.println("Contains 'Pear': " + list.contains("Pear"));
        System.out.println();

        // Test iterator
        System.out.println("Testing iterator:");
        System.out.print("Elements: ");
        for (String element : list) {
            System.out.print(element + " ");
        }
        System.out.println("\n");

        // Test remove by element
        System.out.println("Testing remove by element:");
        System.out.println("Removing 'Banana': " + list.remove("Banana"));
        System.out.println("List after removal: " + list);
        System.out.println();

        // Test empty list behavior
        System.out.println("Testing empty list behavior:");
        try {
            while (!list.isEmpty()) {
                list.removeLast();
            }
            System.out.println("List is empty: " + list.isEmpty());
            list.first(); // This should throw an exception
        } catch (EmptyCollectionException e) {
            System.out.println("The EmptyCollectionException is thrown correctly");
        }

        // Test addAfter with non-existent target
        System.out.println("\nTesting addAfter with non-existent target:");
        try {
            list.addToRear("Apple");
            list.addAfter("Grape", "Pear");
        } catch (ElementNotFoundException e) {
            System.out.println("The ElementNotFoundException is thrown correctly");
        }
    }
}
```

```
Testing addToRear method:
List after adding to rear: Front -> [Apple, Banana, Cherry, null, null, null, null, null, null, null] <- Rear
Size: 3

Testing addToFront method:
List after adding to front: Front -> [Dragon Fruit, Apple, Banana, Cherry, null, null, null, null, null, null] <- Rear

Testing addAfter method:
List after adding 'Elderberry' after 'Banana': Front -> [Dragon Fruit, Apple, Banana, Elderberry, Cherry, null, null, null, null, null] <- Rear

Testing first and last methods:
First element: Dragon Fruit
Last element: Cherry

Testing remove methods:
Removed first element: Dragon Fruit
Removed last element: Cherry
List after removals: Front -> [Apple, Banana, Elderberry, null, null, null, null, null, null] <- Rear

Testing contains method:
Contains 'Banana': true
Contains 'Pear': false

Testing iterator:
Elements: Apple Banana Elderberry

Testing remove by element:
Removing 'Banana': Banana
List after removal: Front -> [Apple, Elderberry, null, null, null, null, null, null, null] <- Rear

Testing empty list behavior:
List is empty: true
The EmptyCollectionException is thrown correctly

Testing addAfter with non-existent target:
The ElementNotFoundException is thrown correctly

Process finished with exit code 0
```

Problem #3 Code

```
package Ass5_2230;

import Ass5_2230.exceptions.*;

/**
 * ArrayOrderedList represents an array implementation of an ordered list.
 *
 * @author Java Foundations
 * @version 4.0
 */
public class ArrayOrderedList<T> extends ArrayList<T>
implements OrderedListADT<T>
{
   /**
    * Creates an empty list using the default capacity.
    */
   public ArrayOrderedList()
   {
      super();
   }

   /**
    * Creates an empty list using the specified capacity.
```

```
 *
 * @param initialCapacity the initial size of the list
 */
public ArrayOrderedList(int initialCapacity)
{
  super(initialCapacity);
}

/**
 * Adds the specified Comparable element to this list, keeping
 * the elements in sorted order.
 *
 * @param element the element to be added to the list
 */
public void add(T element)
{
  if (!(element instanceof Comparable))
    throw new NonComparableElementException("OrderedList");

  Comparable<T> comparableElement = (Comparable<T>)element;

  if (size() == list.length)
    expandCapacity();

  int scan = 0;

  // find the insertion location
  while (scan < rear && comparableElement.compareTo(list[scan])  > 0)
    scan++;

  // shift existing elements up one
  for (int shift = rear; shift > scan; shift--)
    list[shift] = list[shift - 1];

  // insert element
  list[scan] = element;
  rear++;
  modCount++;
}

private <T extends Comparable<T>> int find(T target){
  int index = 0;

  if (target == null)
    throw new NonComparableElementException("OrderedList");

  if(!isEmpty()){
  while (index < rear && target.compareTo((T) list[index]) <= 0) {
    if (target.compareTo((T) list[index]) == 0) {
```

```
            return index;
          }
          index++;
          }
       }
       return -1;
    }
}
```

Problem #3 Test Output

```
package Ass5_2230;
import Ass5_2230.exceptions.*;
public class OrderedListTest {
   public static void main(String[] args) {
      ArrayOrderedList<Integer> list = new ArrayOrderedList<Integer>(5);

      // Test add method and expand capacity
      System.out.println("Testing add method and expand capacity:");
      list.add(7);
      list.add(85);
      list.add(1);
      list.add(15);
      list.add(7);
      list.add(42);
      System.out.println("List after adding: " + list);
      System.out.println("Size: " + list.size());
      System.out.println();

      // Test first and last methods
      System.out.println("Testing first and last methods:");
      System.out.println("First element: " + list.first());
      System.out.println("Last element: " + list.last());
      System.out.println();

      // Test remove methods
      System.out.println("Testing remove methods:");
      System.out.println("Removed first element: " + list.removeFirst());
      System.out.println("Removed last element: " + list.removeLast());
      System.out.println("List after removals: " + list);
      System.out.println();

      // Test contains method
      System.out.println("Testing contains method:");
      System.out.println("Contains 15: " + list.contains(15));
      System.out.println("Contains 102: " + list.contains(102));
      System.out.println();
```

```
        // Test iterator
        System.out.println("Testing iterator:");
        System.out.print("Elements: ");
        for (Integer element : list) {
            System.out.print(element + " ");
        }
        System.out.println("\n");

        // Test remove by element
        System.out.println("Testing remove by element:");
        System.out.println("Removing 15: " + list.remove(15));
        System.out.println("List after removal: " + list);
        System.out.println();

        // Test empty list behavior
        System.out.println("Testing empty list behavior:");
        try {
            while (!list.isEmpty()) {
                list.removeLast();
            }
            System.out.println("List is empty: " + list.isEmpty());
            list.first(); // This should throw an exception
        } catch (EmptyCollectionException e) {
            System.out.println("The EmptyCollectionException is thrown correctly");
        }

    }
}
```

```
Testing add method and expand capacity:
List after adding: Front -> [1, 7, 7, 15, 42, 85, null, null, null, null] <- Rear
Size: 6

Testing first and last methods:
First element: 1
Last element: 85

Testing remove methods:
Removed first element: 1
Removed last element: 85
List after removals: Front -> [7, 7, 15, 42, null, null, null, null, null] <- Rear

Testing contains method:
Contains 15: true
Contains 102: false

Testing iterator:
Elements: 7 7 15 42

Testing remove by element:
Removing 15: 15
List after removal: Front -> [7, 7, 42, null, null, null, null, null, null] <- Rear

Testing empty list behavior:
List is empty: true
The EmptyCollectionException is thrown correctly

Process finished with exit code 0
```