Daniel C. Landon Jr.

Lab 0

String ArrayList

01.25.2020

We were to download two separate “.java” files and modify them according to instructions. We were not allowed to change the existing ode in the files, only add to the files the requested methods. Following is a simple list of what we were required to do.

* Add a method where the user can enter a value and the method would add this value to an array that already contains data.
* Create a method that will search an array, return true if the user value requested exists in the array m false if not found.
* User can enter a numeric value for the index of the array. If value was not out of bounds then return the element at that index.
* User supplied index value. If not out of bounds then remove the element and update the array to show the change.
* toString method for returning a formatted representation of the array
* As an extra piece of code we created a ne wmethod that prompted the user to enter a new sentence, we would then parse the entire sentence into separate words. Nothing else was mentioned about this bu I also added the new sentence “words” to the existing array.

***CONSOLE OUTPUT***

# of words in this list is 0

Enter 12 arbitrary words one at a time.

no matter how hard your past you can always begin again buddah

Enter A New Sentence To Parse Into Words.

no matter where you go there you are

# of words in this list is 20

The words in the list are ( no[0] matter[1] how[2] hard[3] your[4] past[5] you[6] can[7] always[8] begin[9] again[10] buddah[11] no[12] matter[13] where[14] you[15] go[16] there[17] you[18] are[19] )

Enter one more word.

bob

The words in the list are ( bob[0] no[1] matter[2] how[3] hard[4] your[5] past[6] you[7] can[8] always[9] begin[10] again[11] buddah[12] no[13] matter[14] where[15] you[16] go[17] there[18] you[19] are[20] )[20] )

The word "apple" is in this list.false

The word at position 3 is how

The word at position 15 is where

Removing the word at index 8.

The removed word is can

# of words in this list is 20

The words in the list are ( bob[0] no[1] matter[2] how[3] hard[4] your[5] past[6] you[7] always[8] begin[9] again[10] buddah[11] no[12] matter[13] where[14] you[15] go[16] there[17] you[18] are[19] )

***MyStringArrayListTest.java***

/\*

\* Name:

\* Date Written:

\* Purpose: Test the methods in ADT MyStringArrrayList

\*/

package app;

import java.util.\*;

public class MyStringArrayListTest{

public static void main(String[] args){

Scanner input = new Scanner(System.in);

MyStringArrayList mylist = new MyStringArrayList();

System.out.println("# of words in this list is " + mylist.getSize());

System.out.println("Enter 12 arbitrary words one at a time. ");

for (int i = 0; i < 12; i++){

mylist.addLast(input.next());

}//for

// extra method as project instructions

mylist.enterSentence();

System.out.println("# of words in this list is " + mylist.getSize());

System.out.println("The words in the list are " + mylist);

System.out.println("Enter one more word. ");

mylist.addFront(input.next());

System.out.println("The words in the list are " + mylist);

System.out.println("The word \"apple\" is in this list." + mylist.contains("apple"));

System.out.println("The word at position 3 is "+ mylist.getElement(3));

System.out.println("The word at position 15 is "+ mylist.getElement(15));

System.out.println("Removing the word at index 8.");

String deleted = mylist.removeElementAt(8);

System.out.println("The removed word is " + deleted);

System.out.println("# of words in this list is " + mylist.getSize());

System.out.println("The words in the list are " + mylist);

// i realize that the instructions say to not modify the code, however the Scanner input object is never closed and it will bug me to no end if i do not close it

input.close();

}//main

}//class

***MyStringArrayList.java***

/\*\*

\* Date Written:

\* Purpose: Simulates an ArrayList for Strings. MyStringArrrayList

\* is an abstract data type (ADT).

\*

\* @author Suranga Hettiarachchi

\*/

package app;

import java.util.Scanner;

public class MyStringArrayList {

private String[] array; //container of this array list

private static final int CAPACITY = 10; // initial capacity of the array

private int size; // number of elements in the array

/\*\*

\* <h1>enterSentence</h1>

\*

\* <p><strong><em>Notes: </em></strong>prompts the user to enter a new sentence, parse the sentence into seperate words, add new words to existing list</p>

\*

\* @custom.precondition array must exist

\*

\* @custom.postcondition adds new words into existing array

\*

\*/

public void enterSentence(){

Scanner \_userInput = new Scanner(System.in);

System.out.println("\nEnter A New Sentence To Parse Into Words.");

String \_userSentence = \_userInput.nextLine();

System.out.println("");

String[] \_parse;

\_parse = \_userSentence.split(" ");

// loop the array

for(String \_item: \_parse){

// add to array

addLast(\_item);

} // end for

} // end parseSentence

/\*\*

\* <h1>removeElementAt</h1>

\*

\* <p><strong><em>Notes: </em></strong>removes the element at specified index, returns the value of the element that is being removed or null if out of bounds</p>

\*

\* @custom.precondition array must exist but index can be out of bounds

\*

\* @custom.postcondition returns a value that is removed, removes the value or null if out of bounds

\*

\* @param i

\* @return

\*/

public String removeElementAt(int i){

// this is an interesting method. We return NULL if the element requested is out of bounds otherwise we return the value then remove it, however the way the array is designed NULL is a possible value.

// are we looking for something out of bounds for the array

if((i < 0) || ( i > array.length - 1)){

// out of bounds

return null;

} // end if

else {

// remove the value and return what was removed

String \_removedValue = array[i];

// the new array that is created from removing the element

String[] \_tempArray = new String[array.length - 1];

// add elements to the \_tempArray that fall before the index requested

System.arraycopy(array, 0, \_tempArray, 0, i);

// add elements to the \_tempArray that fall after the index requested

System.arraycopy(array, i + 1, \_tempArray, i, (array.length - 1 - i));

// took the following lines from addLast

// set array to \_tempArray

array = \_tempArray;

// decrement the size counter

size--;

return \_removedValue;

} // end else

} // end removeElementAt;

/\*\*

\* <h1>getElement</h1>

\*

\* <p><strong><em>Notes: </em></strong>takes an index value and returns the data at that location</p>

\*

\* @custom.precondition array must exist but indaex value may be out of bounds

\*

\* @custom.postcondition returns a value or null if nothing found

\*

\* @param i which index value to search

\* @return value if found null if nothing

\*/

public String getElement(int i){

// this is an interesting method. We return NULL if the element requested is out of bounds otherwise we return the value, however the way the array is designed NULL is a possible value.

// are we looking for something out of bounds for the array

if((i < 0) || ( i > array.length - 1)){

// out of bounds

return null;

} // end if

else {

// return value;

return array[i];

} // end else

} // end getElement

/\*\*

\* <h1>contains</h1>

\*

\* <p><strong><em>Notes: </em></strong>checks the array for an existing value</p>

\*

\* @custom.precondition array must be populated with values

\*

\* @custom.postcondition true or false if requested value is found in array

\*

\* @param e value to look for in array

\* @return true value found, false if value not found

\*/

public boolean contains(String e){

// loop the array

for(String \_item: array){

// if found return true, short circuit and return

if(\_item == e) return true;

} // end for

// default to false for comparison

return false;

} // end contains

/\*\*

\* <h1>addFront</h1>

\*

\* <p><strong><em>Notes: </em></strong>modifies the arry to add a new item to the beginning</p>

\*

\* @custom.precondition array must be populated with values

\*

\* @custom.postcondition array will have new element added to index 0 and everything will be shifted up the chain

\*

\* @param e value to add to beginning fo array

\*/

public void addFront(String e){

// temp array with array size plus one

String[] \_tempArray = new String[array.length + 1];

// add new item to index 0 of temp array

\_tempArray[0] = e;

// add the original array to the end of the \_tempArray

System.arraycopy(array, 0, \_tempArray, 1, array.length);

// set array to \_tempArray

array = \_tempArray;

// increment the size counter

size++;

} // end addFront

/\*\*

\* <h1>toString</h1>

\*

\* <p><strong><em>Notes: </em></strong>Returns a string that contains the array.</p>

\*

\* @custom.precondition array must be created first

\*

\* @custom.postcondition returns the contents in a formated string

\*

\* @return the array in string format

\*/

public String toString(){

// varaibles

String \_tempString = "";

\_tempString = "( "; // fun with string

// loop the array, could use for/each here but I want the index number

for(int \_counter = 0; \_counter < array.length; \_counter++) {

\_tempString+= array[\_counter] + "[" + \_counter + "] ";

} // end for

\_tempString+= ")"; // fun with strings

return \_tempString;

} // end toString

/\*\*

\* This default constructor sets the container capacity to be 10 and size to

\* 0, where container is empty.

\*/

public MyStringArrayList() {

array = new String[CAPACITY];

size = 0;

}//comstructor

/\*\*

\*

\* @param e - element to be added

\* requires: e != null

\* ensures : e added to the end of this array list.

\* Add a new element at the end of this list

\*/

public void addLast(String e) {

if (isFull()) { // increase capacity if array is full

String[] temparray = new String[array.length + CAPACITY];

for (int i = 0; i < array.length; i++) {

temparray[i] = array[i];

}//for

array = temparray; // make temparray the container

}//if

array[size] = e; //add element e to last empty location.

size++; // increment the size of this array list

}//addLast

/\* This method is only used by this class.

\* returns true if this array list is full, otherwise returns false.

\*/

private boolean isFull() {

return size == array.length;

}//Full

/\*\*

\* requires: none

\* ensures: self.size == #self.size

\* @return current size of this array list

\* returns the current size (# of elements) of this list

\*/

public int getSize() {

return size;

}//Size

}