using System;

using System.Collections.Generic;

namespace Lab\_1\_Assignment {

internal class Program {

static void Main(string[] args) {

List l2 = new List();

Console.WriteLine(l2.toString());

l2.printAllElements();

List l = new List(1);

Console.WriteLine(l.toString());

l.printAllElements();

l.add(2);

Console.WriteLine(l.toString());

l.printAllElements();

l.add(3);

Console.WriteLine(l.toString());

l.printAllElements();

l.add(1);

Console.WriteLine(l.toString());

l.printAllElements();

l.remove(1);

Console.WriteLine(l.toString());

l.printAllElements();

l.remove(3);

Console.WriteLine(l.toString());

l.printAllElements();

l.remove(1);

Console.WriteLine(l.toString());

l.printAllElements();

l.remove(2);

Console.WriteLine(l.toString());

l.printAllElements();

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel.Design.Serialization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Net.Mime.MediaTypeNames;

namespace Lab\_1\_Assignment {

/// <summary>

/// Linked List class capable of collecting LinkedNodes

/// </summary>

public partial class List {

/// <summary>

/// It's the root of the List (the first node)

/// </summary>

LinkedNode root;

/// <summary>

/// The current number of nodes in the List

/// </summary>

int numberOfElements;

/// <summary>

/// Default constructor of the List. The root will have the value 0

/// </summary>

public List() {

root = new LinkedNode(0);

numberOfElements++;

}

/// <summary>

/// Constructor of the List. It sets the value of the root as the given one

/// </summary>

/// <param name="value">It's the value for the root</param>

public List(int value) {

root = new LinkedNode(value);

numberOfElements++;

}

/// <summary>

/// Getter for the number of nodes of the list

/// </summary>

public int NumberOfElements {

get { return numberOfElements; }

}

/// <summary>

/// Adds a new node, with the value passed as parameter, to the last position of the list

/// </summary>

/// <param name="value">It's the value for the new node to be added</param>

/// <returns>true as allways it's added</returns>

public Boolean add(int value) {

LinkedNode newNode = new LinkedNode(value);

if (root == null) {

root=newNode;

return true;

}

LinkedNode currentNode = root;

for (int i = 0; i < numberOfElements-1; i++) {

currentNode = currentNode.Next;

}

currentNode.Next=newNode;

numberOfElements++;

return true;

}

/// <summary>

/// Adds a new node, with the value passed as parameter, to the given index

/// </summary>

/// <param name="value">It's the value for the new node to be added</param>

/// <param name="index">It's the index for the new node</param>

/// <returns>true as always is added</returns>

public Boolean add(int value, int index) {

//TO DO

return true;

}

/// <summary>

/// Removes the node with the value passed as parameter

/// </summary>

/// <param name="value">It's the value of the node wanted to be removed</param>

/// <returns>true if the node with that value was removed sucessfully</returns>

public Boolean remove(int value) { //this is the remove(Object value)

LinkedNode node=root;

LinkedNode nextNode = root.Next;

if (root.Value == value) {

root = nextNode;

numberOfElements--;

return true;

}

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

if (nextNode.Value == value) {

node.Next = nextNode.Next;

numberOfElements--;

return true;

}

node.Next=node.Next;

nextNode = nextNode.Next;

}

return false;

}

//Object remove (int index)

/// <summary>

/// Removes the root

/// </summary>

/// <returns>true if the root was removed sucessfully (not null)</returns>

public Boolean remove() {

if (root != null) {

remove(root.Value);

return true;

} else {

return false;

}

}

/// <summary>

/// Returns the value of the node at the given index

/// </summary>

/// <param name="index">It's the index of the node to be retrieved</param>

/// <returns>The value of the node</returns>

/// <exception cref="IndexOutOfRangeException"></exception>

public int getElement(int index) {

if (index < 0 || index >= numberOfElements) {

throw new IndexOutOfRangeException($"The index isn't valid (it must be between 0 and {numberOfElements}");

}

LinkedNode currentNode = root;

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

currentNode = currentNode.Next;

}

return currentNode.Value;

}

/// <summary>

/// It returns a string representing the list as: (value)-(value)-....

/// </summary>

/// <returns>The string representing the list</returns>

public String toString() {

String aux = "";

LinkedNode currentNode = root;

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

aux+=$"({currentNode.Value})-";

currentNode = currentNode.Next;

}

return aux;

}

/// <summary>

/// Used for testing: prints all the elements of the list by going through it and calling getElement()

/// </summary>

public void printAllElements() {

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

Console.WriteLine(getElement(i));

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_1\_Assignment {

public partial class List {

/// <summary>

/// Node class for the LinkedList elements that are capable to store integers

/// </summary>

private class LinkedNode {

/// <summary>

/// Integer stored in the node

/// </summary>

int value;

/// <summary>

/// Reference to the next node of this node

/// </summary>

LinkedNode next;

/// <summary>

/// Constructor that creates the node with the number passed as parameter

/// </summary>

/// <param name="value">It's the numerical value for the Node</param>

public LinkedNode(int value) {

this.value = value;

next = null;

}

/// <summary>

/// Getter and setter for the value of the Node

/// </summary>

public int Value {

get { return value; }

set { this.value = value; }

}

/// <summary>

/// Getter and setter for the Next node reference

/// </summary>

public LinkedNode Next {

get { return next; }

set { this.next = value; }

}

}

}

}