Lucrarea nr. 9 - Liste dublu înlănțuite

Breviar teoretic

În cadrul elementelor studiate anterior s-a putut observa utilitatea listelor simplu înlănuite, a inserării unor elemente în cadrul listelor, a sortării acestora.
Se pune problema modului în care se poate implementa o lista cu 2 identificatori de cuplare: unul cu următorul element, iar celalalt cu predecesorul.
Prin urmare vor fi două elemente individuale, care vor identifica capatul listei elementul prim – **`First`** şi elementul ultim –**`Last.`**
Pentru ambele cazuri construcția, inserția unui element ca trebui să ia în considerare cazul în care avem de-a face cu mulțimea vida.
Structura unui program va cuprinde următoarele module:

Crearea elementului unei liste

Se observă că această clasă crează un element al listei conținând, informația efectivă - **`dData`**, informație de cuplaj (de tip clasă element al listei)- **`next`** şi **`previous`**

```
class Link
{
public double dData;
public Link next;
public Link previous;
// --------------------------------------------
```

- un constructor care permite inițializarea informației efective, (o suprainscriere a clasei)

```
public Link(double d)
{ dData = d; }
// --------------------------------------------
```

- o metodă de vizualizare a informației conținute în respectivul element:

```
public void displayLink()
{ System.out.print(dData + " "); }
```

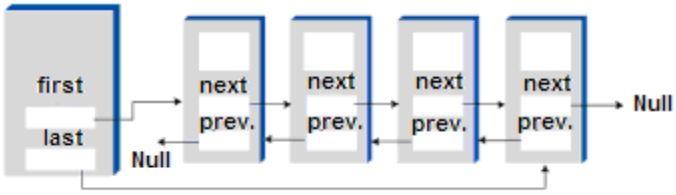
În continuare este creată structura listei dublu înlănțuite - **`DoublyLinked`**, având la bază descrierea elementelor realizată în cadrul clasei **`Link`**:

În cadrul listei sunt inserate cele două elemente particulare: primul capăt al listei – **`first`** şi cel de-al doilea capăt al listei – **`last`**, fiind inițializate cu elementul **`vid – null`**.

```
class DoublyLinkedList
{ private Link first;
  private Link last;
// ------------------------------------------
public DoublyLinkedList()
{first = null;last = null;}
// ------------------------------------------
```

Este creat indicatorul boolean - **`isEmpty()`** - de listă vidă – echivalent cu identificarea cu elementul nul al primului element – **`first`**

```
public boolean isEmpty()
{ return first==null; }
```



Este creată metoda de inserare a unui element la primul capăt al listei – **`insertFirst().`** Această metodă tretează separate, prin intermediul unei structuri **`if`** cazul în care lista era vidă (**`isEmpty`** este **`true`**), situație în care elementul introdus prin intermediul metodei este identificat cu elementul **`last`,** respectiv cazul în care pe prima poziție exista deja un element. Acest element va deveni elementul **`next`** pentru elemental nou introdus, în timp ce elementul nou introdus va deveni **`first`** (ca locație intermediară pentru interschimbare este folosită locația **`first.previous`**).

```
public void insertFirst(double dd)
{
Link newLink = new Link(dd);
if( isEmpty() )
last = newLink;
else
first.previous = newLink;
newLink.next = first;
first = newLink; }
// ------------------------------------------
```

În mod similar se construieşte metoda **`insertLast().`**

```
public void insertLast(double dd)
{ Link newLink = new Link(dd);
if( isEmpty() )
first = newLink;
else
{ last.next = newLink;
```

```
newLink.previous = last;}
last = newLink;}
// ------------------------------------------
```

Metoda de ştergere a primului element – **deleteFirst(),** tratează separate cele două cazuri posibile:

- la ştergere lista având un singur element , devine lista vidă – **first.next==null** atunci şi **last=null**
- lista conține mai multe elemente şi la ştergere trebuie refacute legăturile cu elementul de după elementul şters **(first** devine elementul care era anterior **first.next,** în timp ce elementul **previous** al elementului de după **first,** înainte de ştergere devine **null –** vezi figura. Anterior era elementul **first** ce este şters şi trimis în afara metodei prin variabila **temp)**

```
public Link deleteFirst()
{ Link temp = first;
if(first.next == null)
last = null;
else
first.next.previous = null;
first = first.next;
return temp;}
// ------------------------------------------
```

Similar metoda **deleteLast()**

```
public Link deleteLast()
{ Link temp = last;
if(first.next == null)
first = null;
else
last.previous.next = null;
last = last.previous;
return temp;}
// ------------------------------------------
```

Metoda **insertAfter()** permite inserarea unei înregistrări, în cadrul listei după un anumit element indicat în cadrul metodei. Prin funcția **while** se realizează parcurgerea listei până în momentul în care este localizată înregistrarea identică cu valoarea **key** introdusă. Structura **if** transează cazul în care nu a fost găsită nici o înregistrare corespunzătoare, când metoda întoarce rezultatul **false**, respectiv cazul în care a fost localizată corespunzător respectiva înregistrare.
Dacă locația identificată este ultima locație, atunci este introdusă direct noua înregistrare **dd** care a fost instanțiată prin variabila **newLink**, nu înainte de a fi inițializată legatura **next** cu **null** (vezi figura).
Dacă locația este diferită de ultima poziție, atunci locația curentă – a cheii identificate, va transfera legătura **next**, la înregistrarea introdusă, poziția **previous** a locației **current.next** va fi identificată prin înregistrarea introdusă, în timp ce locația **previous** a înregistrării introduce va reveni înregistrării curente, care se va lega de înregistrarea introdusă prin câmpul **next**. Transferul va fi semnalizat prin transferul variabilei booleene true, care va confirma realizarea inserării.

```
public boolean insertAfter(double key, double dd)
{ Link current = first;
while(current.dData != key)
{current = current.next;
if(current == null)
return false; }
Link newLink = new Link(dd);
if(current==last)
{newLink.next = null;
last = newLink; }
else
{ newLink.next = current.next;
current.next.previous = newLink;}
newLink.previous = current;
current.next = newLink;
return true; }
// --------------------------------------
```

Similar, metoda de ştergere a unei înregistrări situate după o anumită cheie de identificare **`deleteKey()`**

```
public Link deleteKey(double key)
{ Link current = first;
while(current.dData != key)
{current = current.next;
if(current == null)
return null; }
if(current==first)
first = current.next;
else
current.previous.next = current.next;
if(current==last)
last = current.previous;
else
current.next.previous = current.previous;
return current; }
// --------------------------------------
```

Metoda **`displayForward`** permite vizualizarea listei de la primul element la ultimul. Astfel, după afişarea mesajului de identificare a direcţiei de afişare : Lista (Primul --> Ultimul):, este citită prima înregistrare a listei, după care utilizând o structură **`while`**, care permite baleierea întregii liste (până la identificarea elementului vid), prin apelarea metodei **`displayLink()`** din clasa de creare a unei înregistrări (clasa Link).

```
public void displayForward()
{ System.out.print("Lista (Primul-->Ultimul ): ");
Link current = first;
```

```
while(current != null)
{ current.displayLink();
current = current.next;}
System.out.println(""); }
```

Metoda **`displayBackward()`** este similară cu metoda anterioară cu observația că afișarea se face de la ultimul element la primul.

```
public void displayBackward()
{ System.out.print("Lista (Ultimul -->Primul: ");
Link current = last;
while(current != null)
{ current.displayLink();
current = current.previous;}
System.out.println(""); }
```

Aplicatie : Creați şi implementați o listă dublu înlănțuită în proiectul *DoubleLinkList*:

```
import java.io.*;
class Link{
      public double dData;
      public Link next;
      public Link previous;

public Link(double d){
      dData = d;}

public void displayLink(){
      System.out.print(dData +" ");}}

class DoublyLinkedList{
      private Link first;
      private Link last;
public DoublyLinkedList(){
      first = null;
      last = null;}

public boolean isEmpty(){
      return first==null;}

public void insertFirst(double dd){
      Link newLink = new Link(dd);
      if(isEmpty())
            last=newLink;
      else {
            first.previous = newLink;
            newLink.next = first;
            first = newLink;}}

public void insertLast(double dd){
      Link newLink = new Link(dd);
      if(isEmpty())
            first = newLink;
```

```
      else {
            last.next = newLink;
            newLink.previous = last;}
      last = newLink;}

public Link deleteFirst(){
      Link temp = first ;
      if(first.next == null)
            last = null;
      else
            first.next.previous = null;
            first = first.next;
      return temp;}

public Link deleteLast(){
      Link temp = last;
      if(first.next==null)
            first = null;
      else
            last.previous.next = null;
      last = last.previous;
      return temp;}

public boolean insertAfter(double key, double dd){
      Link current = first;
      while(current.dData != key){
            current = current.next;
            if(current ==null)
                  return false;}
      Link newLink = new Link(dd);
      if(current==last){
            newLink.next=null;
            last = newLink;}
      else{
            newLink.next = current.next;
      current.next.previous = newLink;}
      newLink.previous = current;
      current.next = newLink;
      return true;}

public Link deleteKey(double key){
      Link current = first;
      while(current.dData != key){
            current = current.next;
      if(current==null)
            return null;}
      if(current==first)
            first = current.next;
      else
            current.previous.next = current.next;
      if(current==last)
            last = current.previous;
      else
        current.next.previous = current.previous;
      return current;}

public void displayForward(){
```

```
System.out.print("Lista( Primul --> Ultimul ):") ;
      Link current = first ;
      while(current != null){
            current.displayLink();
            current = current.next;}
      System.out.println("");}

public void displayBackward(){
System.out.print("Lista( Ultimul --> Primul ): ");
      Link current = last;
      while(current != null){
            current.displayLink();
            current = current.previous;}
      System.out.println("");}
}
```

Aplicatie : Creați şi implementați o listă dublu înlănțuită în proiectul *InterIterApp*:

```
import java.io.*;
class Link{
       public double dData;
       public Link next;

public Link(double dd){
       dData=dd;}

public void displayLink(){
       System.out.print(dData +" ");}}

class LinkList{
       private Link first;

public LinkList(){
       first=null;}

public Link getFirst(){
       return first;}

public void setFirst(Link f){
       first = f;}

public boolean isEmpty(){
       return first==null;}

public ListIterator getIterator(){
       return new ListIterator(this);}

public void displayList(){
       Link current = first;
       while(current!=null){
              current.displayLink();
              current = current.next;}
       System.out.println("");}
}

class ListIterator{
       private Link previous;
       private Link current;
       private LinkList ourList;

public ListIterator(LinkList list){
       ourList = list;
       reset();}
```

```
public void reset(){
	current= ourList.getFirst();
	previous = null;}

public boolean atEnd(){
	return(current.next==null);}

public void nextLink(){
	previous = current;
	current=current.next;}

public Link getCurrent(){
	return current;}

public void insertAfter(double dd){
	Link newLink = new Link(dd);
	if(ourList.isEmpty()){
		ourList.setFirst(newLink);
		current=newLink;}
	else{
		newLink.next=current.next;
		current.next=newLink;
		nextLink();}
}

public void insertBefore(double dd){
	Link newLink = new Link(dd);
	if(previous==null){
		newLink.next = ourList.getFirst();
		ourList.setFirst(newLink);
		reset();}
	else{
		newLink.next = previous.next;
		previous.next = newLink;
		current = newLink;}
}

public double deleteCurrent(){
	double value = current.dData;
	if(previous==null){
		ourList.setFirst(current.next);
		reset();}
	else{
		previous.next = current.next;
		if(atEnd())
			reset();
		else
			current = current.next;}
return value;}
}

import java.io.*;
class InterIterApp{
public static void main(String args[]) throws IOException{
	LinkList theList = new LinkList();
	ListIterator iter1 = theList.getIterator();
	double value;
iter1.insertAfter(20);
iter1.insertAfter(40);
iter1.insertAfter(80);
iter1.insertAfter(60);
while(true){
System.out.println(".....................................");
System.out.println("tastati prima litera de la Arata=show, Sterge=reset, ");
System.out.println("next, afiseaza=get, inainte=before, dupa=after, sterge=delete;");
System.out.println(".....................................");
	System.out.flush();
	int choise= getChar();
	switch(choise){
		case 's':
			if(!theList.isEmpty())
```

```
                        theList.displayList();
                else
                System.out.println("Lista este goala");
                break;
            case 'r':
                iter1.reset();
                break;
            case 'n':
                if(!theList.isEmpty() && iter1.atEnd())
                        iter1.nextLink();
                else
                  System.out.println("Nu pot gasi legatura urmatoare");
                break;
            case 'g':
                if(!theList.isEmpty()){
                        value = iter1.getCurrent().dData;
                        System.out.println("Valoarea preluata este "+value);}
                else
                        System.out.println("Lista goala");
                break;
            case 'b':
                System.out.print("Introduceti valoarea ce doriti sa o inserati: ");
                System.out.flush();
                value = getInt();
                iter1.insertBefore(value);
                break;
            case 'a':
                System.out.print("Introduceti valoarea dorita ");
                System.out.flush();
                value = getInt();
                iter1.insertAfter(value);
                break;
            case 'd':
                if(!theList.isEmpty()){
                value = iter1.deleteCurrent();
                System.out.println("Valoarea stearsa este "+value);}
                else
                System.out.println("Nu pot sterge");
                break;
            default:
            System.out.println("Intrare invalida");}}}

public static String getString() throws IOException{

InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String s = br.readLine();
        return s;}

public static int getChar() throws IOException{
        String s = getString();
        return s.charAt(0);}

public static int getInt() throws IOException{
        String s = getString();
        return Integer.parseInt(s);}
}
```

Probleme propuse

1. Rulați si comentati programul *DoubleLinkList*.
2. Modificați programul *DoubleLinkList* prin inserarea în cadrul zonei de informație a înregistrării a unui string.
3. Rulati şi comentați programul: *InterIterApp*
4. Modificați programul *InterIterApp* prin inserarea în cadrul zonei de informație a înregistrării a unui string.