Laborator Nr. 5:
Structuri de date 2

Întocmit de: Dobrinaş Alexandra

Îndrumător: Asist. Drd. Danciu Gabriel

November 3, 2011

I. NOŢIUNI TEORETICE

A. Liste

Lista reprezintă cea mai simplă structură de date înlănţuită. Se folosesc doua tipuri de liste:

- liste simplu înlănţuite;
- liste dublu înlănţuite;

B. Liste simplu înlănţuite

Pot fi definite ca fiind o secvenţă de obiecte alocate dinamic, în fiecare obiect se păstrânduse atât informaţia utilă cât şi o referinţă către succesorul sau din listă. Există o multitudine de moduri prin care se poate implementa această structură de date,însă cel mai utilizat mod este prin definirea unei clase cu doi membri, unul în care se reţine informaîia utilă şi unul În care se reţine adresa următorului element din listă.

```
public class LinkedList {

   int data;
   LinkedList next;

   public LinkedList() {

      data = 0;
      next = null;
   }

}
```

Se observă că membrul *next* este de acelaşi tip de date ca şi clasa, acesta pentru că următorul element din listă este tot de acest tip de date. Pe lângă aceşti membri se mai pot adăuga şi alţii care să conţină diverse informaţii, în funcţie de aplicaţia care cere utilizarea unei astfel de structuri.

Următorul exemplu ilustrează crearea şi afişarea unei liste simplu înlănţuite. Iniţial a fost creat un element cu ajutorul constructorului, informaaţia din acest element a primit valoarea 23, iar pentru crearea următorului element s-a instanţiat membrul *next*, ş.a.m.d. Această operaţie poate fi numită *„alipirea următorului element de elementul curent”*. Pentru a parcurge lista se foloseşte un *iterator*.

```
class LinkedListDemo {
   public static void main(String args[])
   {
      LinkedList list = new LinkedList();
      list.data =23;
      list.next = new LinkedList();
      list.next.data = 10;
      list.next.next = new LinkedList();
      list.next.next.data = 49;

   //parcurgerea listei de la primul element

      LinkedList iterator = list;
      do
      {
         System.out.println(iterator.data);
      }
      while((iterator = iterator.next)!=null);

   }
}
```

Un alt mod de folosire al listelor înlănţuite este prin declararea şi reţinerea primului element din listă:

```
class LinkedList2 {

   int data;
   LinkedList2 next;
   LinkedList2 head;

   public LinkedList2() {

      head = this;
      next = null;
      data =0;
   }

   public LinkedList2(LinkedList2 head) {

      this.head = head;
```

```
        next = null;
        data =0;
    }
}


//////////////////////////////


class LinkedList2Demo {

   public static void main(String args[]) {

        LinkedList2 list = new LinkedList2();
        list.head = list;
        list.data =23;
        list.next = new LinkedList2(list);
        list.next.data = 10;
        list.next.next = new LinkedList2(list);
        list.next.next.data = 49;

        //parcurgerea listei de la primul element
        while (list.next !=null)  {

           System.out.println(list.data);
           list = list.next;
        }

        System.out.println(list.data);
        list = list.head;

        while (list.next !=null) {

           System.out.println(list.data);
           list = list.next;
        }
        System.out.println(list.data);
    }
}
```

Se observă că prin reţinerea primunul element al listei se poate ajunge mult mai repede si mai uşor la primul element din listă. De asemenea, se poate observa că acest mod de construire al unei liste nu este eficient, deoarece, în cazul unei liste cu sute sau chiar mii de elemente adăugarea noilor elemente va fi imposibil de realizat. De aceea, *adăugarea unui element nou* precum şi *ştergerea unui element* sunt operaţii specifice pentru lucrul cu liste alături de modificarea informaţiei utile, modificarea legăturii dintre elemente, căutarea unui element în liste, aflarea poziţiei unui element.

C. Liste dublu înlănţuite

În cazul în care se doreşte parcurgerea listei şi în celalalt sens (şi nu doar de la predecesor la succesor), avem lista dublu înlănţuită. Aceasta păstreaz conceptele listei simplu înlănţuite, cu specificarea că fiecare element al listei mai conţine o referinţă şi către elementul precedent. Un mod de declararea al unei astfel de liste este următorul:

```
public class Nod {

   private Nod next;
   private Nod prev;
   private int data;

   public Nod(int data, Nod next,Nod prev)
   {
      this.data = data;
      this.next = next;
      this.prev = prev;
   }
}
```

Operaţile specifice lister simplu înlănţuite se menţin şi pentru listele dublu, doar că trebuie să se ţină cont şi de legătura *prev*.

D. Framework -ul Java pentru Colecţii

Java pune la dispoziţia utilizatorilor o serie de interfeţe şi clase ce pot fi folosite pentru crearea şi utilizarea diferitelor colecţii de date. Printre acestea amintim:

- interfaţa *Collection* - are definite metode pentru o serie de operaţii ce se aplică unei colecţii (detalii aici)

- interfaţa *Iterator* - iteratorul este acel obiect ce permite parcurgerea colecţiilor (detalii aici)
- interfaţa *List* - interfaţă ce implementează *Collection*. Există două clase ce implementează această interfaţă: *ArrayList* şi *LinkedList*. Interfaţa *List* oferă posibilitatea de a lucra ordonat, deci permite păstrarea secvenţială a elementelor dintr-o colecţie (detalii aici)
- clasa *ArrayList* - clasa este echivalentul clasei *Vector*, dar sub forma unei colecţii. Un *ArrayList* este o colecţie de elemente indexate într-o anumită ordine, dar nu neapărat sortate. (detalii)
- clasa *LinkedList* - aceasta este implementarea listei dublu înlănţuite. (detalii)

E. Mulţimi - Interfaţa *Set*

Interfaţa *Set* reprezintă un grup de elemente fără duplicate. Nu există o condiţie anume ce impune acest lucru, ci implementările din clasele Set, sunt cele care impun aceasta condiţie. Interfaţa *Set* derivă din *Collections*, deci va implementa aceleaşi metode ca şi această interfaţă. Un element, aflat deja în mulţime, nu mai poate fi modificat. API-ul interfeţei se poate găsii aici

F. Clasa *HashSet*

Face parte din interfaţa *Set* şi are ca operaţii specifice următoarele:

- adăugarea elementelor;
- ştergerea elementelor;
- verificare dacă lista este goală;
- verificare daca un anumit obiect este conţinut ţn listă;

Un exemplu de utilizare al clasei *HashSet* este următorul:

```
1  import java.util.ArrayList;
2  class Point {
3
4     public int x;
5     public int y;
6
7     public Point (int x, int y) {
8
9        this.x = x;
10       this.y = y;
11    }
12
13    public boolean equals(Object o) {
14
15       if (!(o instanceof Point))
16          return false;
17       Point pt = (Point)o;
18       return ((pt.x == this.x) &&   (pt.y==this.y));
19    }
20
21    public int hashCode() {
22
23       return 17*this.x +23*this.y+43;
24    }
25
26    public String toString() {
27       return "x = " +x+ " y = " +y + " id = "+ hashCode()+ "\n";
28    }
29 }
30
31 ////////////////////
32
33 public class Multimi
34 {
35 public static void main(String args[]) {
36
37    Set set = new HashSet();
38
39    // Adaug in multime
40    set.add(new Point(1,2));
41    set.add(new Point(2,1));
42    set.add("c");
43
```

```
44      // Sterg un element din multime
45      set.remove("c");
46
47      //Marimea unei multimi
48      int size = set.size();
49      System.out.println(size);
50
51      // Adaug un element ce exista deja
52      set.add(new Point(1,2));
53
54      //fara a avea insa efect
55      size = set.size();
56      System.out.println(size);
57
58      //Verificam daca un element este deja in multime
59      boolean b = set.contains(new Point(2,1)); // true
60      System.out.println(b);
61
62      b = set.contains("c"); // false
63      System.out.println(b);
64
65      // Parcurgem multimea
66      Iterator it = set.iterator();
67      while (it.hasNext()) {
68         //si afisam elementele
69         Object element = it.next();
70         System.out.println(element);
71      }
72  }
73  }
```

II. TEME DE LABORATOR

1. Editaţi, compilaţi şi lansaţi în execuţie aplicaţile prezentate la I B.
2. Creaţi o listă dublu înlănţuită şi afişaţi-o.
3. Editaţi, compilaţi şi lansaţi în execuţie aplicaţile prezentate la I F.
4. Se dă o clasă de numere complexe. Folosind clasa *HashSet* creaţi o listă în care să se reţină numere complexe. Adăugaţi minim 5 astfel de numere şi realizaţi un meniu prin care să se rezolve următoarele cerinţe:
   - Afişaţi dimensiunea listei.
   - Verificaţi daca un număr citit există în listă. Dacă nu există, adăugaţi-l şi afişaţi noua lungime a listei.
   - Eliminaţi un număr din listă.
   - Afişati suma numerelor din listă.
   - Afişaţi elementele listei.