Lucrare de laborator nr. 10
Colecţii

1. Scopul lucrării

În această lucrare se studiază colecţiile şi parcurgerea lor.

2. Breviar teroretic

O colecţie este un obiect ce grupează mai multe elemente într-o unitate (grup). Spre deosebire de un vector intrinsec de obiecte, la care după ce i-am stabilit o dimensiune, dacă se dovedeşte necesar în aplicaţie să introducem mai multe elemente în vector decât dimensiunea alocată, nu se mai poate, în cazul unei colecţii, nu se specifică de la început o dimensiune fixă şi deci putem adăuga ulterior câte obiecte este necesar.

În limbajul Java, colecţiile sunt tratate unitar şi punctul central al lucrului cu acestea îl reprezintă interfeţele.

Exemplu de interfeţe:

- Collection
- List
- Set

Aceste interfeţe permit utilizarea unitară a colecţiilor independent de modul lor de implementare. Aceasta este foarte avantajos deoarece reduce efortul de învăţare.

2.1 Interfaţa Collection

Collection este cea mai generală interfaţă şi ea modelează un grup de obiecte numite elemente.

În JDK nu există nici o clasă care să implementeze această colecţie, dar celelalte interfeţe extind interfaţa *Collection*.

Câteva metode ale interfeţei *Collection* :

int size();
returnează câte elemente sunt în colecţie

boolean isEmpty();
returnează *true* dacă colecţia este vidă

boolean contains(Object element);
returnează *true* dacă obiectul este în colecţie

boolean add(Object element);
pentru adăugarea unui element

boolean remove(Object element);
şterge elementul curent

Iterator iterator();
scoate un obiect de tip Iterator, obiect folosit pentru parcurgerea colecţiei

2.2 Interfaţa List

Interfaţa *List* extinde interfaţa *Collection*, astfel:
interface List extends Collection

O clasă ce implementează interfaţa List este folosită pentru a modela o listă de elemente indexate. Aceste liste de elemente indexate permit şi dubluri pentru elementele listei. Implementări ale interfeţei List sunt:

- clasa ArrayList
- clasa LinkedList
- clasa Vector

Se recomandă folosirea colecţiilor parametrizate, colecţii în care sunt specificate din declaraţia lor, tipul obiectelor conţinute. Astfel o colecţie parametrizată ArrayList de obiecte Integer, se instanţiază ca în exemplul următor:

```
ArrayList<Integer> al=new ArrayList<Integer>( );
```

Varianta neparametrizată de instanţiere este:

```
ArrayList al=new ArrayList( );
```

Această variantă neparametrizată nu se recomandă, deoarece în acest caz operaţiile asupra colecţiei sunt neverificate de către compilator.

Câteva din metodele declarate în interfaţa List sunt următoarele:

Object get(int index);
returnează obiectul de pe poziţia index

void set(int index, Object element);
pentru a modifica elementul de pe poziţia index

void add(int index, Object element);
pentru a adăuga un element, pe poziţia index

Object remove(int index);
pentru a scoate din listă elementul de pe poziţia index

int indexOf(Object o);
indexul obiectului o din listă

2.3 Interfaţa Set

Această interfaţă este folosită pentru grupuri de elemente de tip mulţime (nu permite elemente duplicate). O implementare a interfeţei *Set,* este clasa *HashSet.*

2.4 Parcurgerea colecţiilor

Pentru parcurgerea secvenţială a unei colecţii se folosesc enumerările şi iteratorii. Enumerarea este definită a fi un obiect ce implementează interfaţa *Enumeration,* în timp ce iteratorul este un obiect ce implementează interfaţa *Iterator* sau *ListIterator*.

Toate clasele ce implementează colecţii au metode ce returnează o enumerare sau un iterator pentru parcurgerea respectivei colecţii. Astfel interfaţa *Enumeration* defineşte două metode:

- *boolean hasMoreElements()*

Verifică dacă în urma parcurgerii mai sunt elemente.

- *Object nextElement()*

Returnează următorul element

Interfaţa *Iterator* declară trei metode:

- *boolean hasNext()*
  Returnează *true* dacă nu am ajuns la sfârşitul colecţiei.
- *Object next()*

Returnează obiectul următor în colecţie şi avansează cursorul.

- *void remove()*

  Şterge (elimină) din colecţie ultimul element care a fost returnat cu ajutorul obiectului *Iterator*.

Interfaţa *ListIterator* este derivată din interfaţa *Iterator*. Ea permite traversarea colecţiei atât înainte cât şi înapoi. Permite de asemenea inserarea unor noi elemente în colecţie.

Câteva metode noi declarate în interfaţa *ListIterator*:

Object previous()

Returnează elementul anterior din listă şi mută cursorul înapoi cu o poziţie.

boolean hasPrevious()

Returnează *true* dacă mai sunt elemente când traversăm lista în direcţie inversă.

void add(Object o)

Elementul este inserat imediat înainte de elementul care ar fi returnat de *next()* sau după elementul care ar fi returnat de *previous()*.

void set(Object o)

Înlocuieşte ultimul element returnat de *next()* sau *previous()* cu elementul specificat.

Există o serie de algoritmi care sunt folosiţi pentru prelucrarea unitară a colecţiei, aceştia sunt grupaţi în clasa *Collections* ce conţine metode statice care în mod tipic au ca parametru un obiect de tip *Collection*. Exemplu:

public static void sort(Collection c)

Sortează colecţia

public static void shuffle(Collection c)

Amestecă în ordine aleatoare elementele din colecţie

3. Probleme rezolvate

Problema 1

Citim repetat de la tastatură câte un număr natural pe care-l memorăm într-o colecţie ArrayList, până când se tasează numărul -1. Să se afişeze toate numerele din colecţie şi apoi să se afişeze doar cele mai mici două numere din colecţie. Dacă sunt mai puţin de două elemente în colecţie se va da un mesaj corespunzător.

```
import java.util.*;
class C1
{
 public static void main(String args[])
{
  ArrayList<Integer> al=new ArrayList<Integer>();
  Scanner sc=new Scanner(System.in);
  for(;;){
        int nr; System.out.print("NR=");
        nr=sc.nextInt();
        if(nr==-1)break;
        al.add(nr);
  }
  //al.add("mar"); Colectii parametrizate! Eroare la compilare!
  //Parcurgerea colectiei:
  Iterator<Integer> it=al.iterator();
  while(it.hasNext())
    System.out.print(it.next()+" ");
  System.out.println();
  //Cele mai mici doua elemente:
  int n=al.size();
  if(n<2){
        System.out.println("Colectia are mai putin de 2 elemente !");
        System.exit(0);
  }
  Collections.sort(al);
  //parcurgerea colectiei:
  it=al.iterator();
  System.out.println("Cele mai mici doua elemente:");
  System.out.print(it.next()+" ");
  System.out.println(it.next());
 }
}
```

Problema 2

Se va crea o lista parametrizată de obiecte String. Se va ilustra şi folosirea metodei remove() la parcurgerea listei cu iterator.

```
import java.util.*;
```

```
class AList_String
{
public static void main(String args[])
{
 List<String> al=new ArrayList<String>();
 al.add("mar");
 al.add("para");
 al.add("kiwi");
 Iterator<String> it=al.iterator();
 while(it.hasNext()){
       String s=it.next();
       if(s.equals("para"))it.remove();
 }
 System.out.println("Afisam lista:");
 it=al.iterator();
 while(it.hasNext())
     System.out.println(it.next());
 }
}
```

Problema 3

Creem o lista parametrizată de studenți (nume si nota). O afișăm in ordinea crescătoare a notelor.

```
import java.util.*;
import javax.swing.*;
class Student implements Comparable<Student>
{
 private String nume;
 private double medie;
 public Student(String nume, double medie)
 {
  this.nume=nume;
  this.medie=medie;
 }
//Folosim @Override deoarece compilatorul va verifica si ne va
// avertiza daca nu am dat corect parametrii metodei ce vrem sa o
// rescriem. De asemenea, codul scris devine mai usor de inteles.
//Metoda compareTo() este singura definita in interfata Comparable.
```

```
 @Override  //atentie,  cu O !
 public int compareTo(Student s)
 {
  if(medie>s.medie)return 1;
  else if(medie==s.medie)return 0;
  else return -1;
 }
 @Override
 public String toString()
 {
    return "["+nume+" : "+medie+" ]";
 }
}
class AList_Studenti
{
 public static void main(String args[])
 {
  List<Student> al=new ArrayList<Student>();
  for(;;){
    String s=JOptionPane.showInputDialog("nume=");
    if(s.equals("stop"))break;
    double nota=
         Double.parseDouble(JOptionPane.showInputDialog("nota="));
    al.add(new Student(s,nota));
  }
  Collections.sort(al);
  //Afisam :
  Iterator<Student> it=al.iterator();
  while(it.hasNext())
     System.out.println(it.next());
 }
}
```

Problema 4

Se citesc într-o colecție niște nume (până se tastează "stop"). Să se afișeze doar cele ce încep cu litera A.
Apoi să se elimine din colecție cele ce încep cu litera A.

```
import java.util.*;
import javax.swing.*;
```

```
class EliminareDinLista
{
 public static void main(String args[])
 {
  List<String> al=new ArrayList<String>();
  for(;;){
    String s=JOptionPane.showInputDialog("nume=");
    if(s.equals("stop"))break;
    else al.add(s);
  }
  //afisare cele ce incep cu 'A':
  Iterator<String> it=al.iterator();
  while(it.hasNext()){
     String s=it.next();
     if(s.charAt(0)=='A')System.out.println(s);
  }
  it=al.iterator();
  while(it.hasNext()){
       String s=it.next();
       if(s.charAt(0)=='A')it.remove();
  }
  //Afisarea celor ramase:
  System.out.println("Au ramas in lista:");
  it=al.iterator();
  while(it.hasNext())
      System.out.println(it.next());
 }
}
```

Problema 5

Se citesc de la tastatură nişte numere naturale (până se tastează valoarea -1). Numerele (neştiind de la început câte sunt) se vor memora într-o listă ArrayList . Vom afişa apoi dacă toate numerele prime din listă sunt diferite între ele sau nu. Pentru aceasta vom copia mai întâi toate numerele din listă într-un vector de numere întregi.

Solutie

```
import java.util.*;
import javax.swing.*;
```

```
class NumerePrimeDiferite
{
public static void main(String args[])
{
 List<Integer> al=new ArrayList<Integer>();
 Scanner sc=new Scanner(System.in);
 for(;;){
  System.out.print("nr=");
  int nr=sc.nextInt();
  if(nr==-1)break;
  else al.add(nr);
 }
 //Parcurgem lista si copiem numerele prime in vectorul a[]:
 int a[]=new int[al.size()];//dimensiune acoperitoare
 int i=0;//index in a[]
 //Parcurgem lista al:
 Iterator<Integer> it=al.iterator();
 while(it.hasNext()){
  int x=it.next();
  if(estePrim(x)){a[i]=x; i++;}
 }
 int nP=i;//numarul de numere prime
 //Afisarea numerelor prime:
 for(i=0;i<nP;i++)
  System.out.print(a[i]+"  ");
 System.out.println();
 //Sunt toate diferite intre ele?
 int j;
 for(i=0;i<nP-1;i++)
  for(j=i+1;j<nP;j++)
   if(a[i]==a[j]){System.out.println("Nu sunt toate diferite");
            return;}
  System.out.println("Sunt toate diferite");
 }
 private static boolean estePrim(int nr)
 {
 for(int i=2;i<=Math.sqrt(nr);i++)
  if(nr%i==0)return false;
 return true;
 }
```

```
}
```

Problema 6

Să se scrie clasa Umbrela ce are ca variabile de instanţă culoarea (String) şi diametrul (int) al unei umbrele. Ca metode avem: constructorul, metodele setCuloare(), setDiametru(), getCuloare(), getDiametru(), esteMare() care returnează true dacă diametrul este mai mare ca 60, esteCuloareInchisa().

Folosind obiecte de tip Umbrela, vom face o comparaţie între clasele ArrayList, Vector şi LinkedList. Se va crea un obiect ArrayList de obiecte Umbrela în care se introduc N umbrele. Se va afişa apoi această colecţie. În continuare aceeaşi problemă o vom implementa folosind un obiect Vector pentru a memora obiectele Umbrela , şi apoi un obiect LinkedList.

Pentru fiecare situaţie de implementare din cele trei vom folosi aceeaşi metodă pentru afişarea colecţiei, şi anume:

void afisareColectie(List<Umbrela> l).

Solutie

```
import java.util.*;
import java.io.*;
class Umbrela
{
 private String culoare;
 private int diametru;
 private final int MARE=60;
 public Umbrela(String c, int d)
 {
  culoare=c;
  diametru=d;
 }
 public void setCuloare(String c){culoare=c;}
 public void setDiametru(int d){diametru=d;}
 public String getCuloare(){return culoare;}
 public int getDiametru(){return diametru;}
 public boolean esteMare()
 {
  if(diametru>=MARE)return true;
  else return false;
 }
```

```
 public boolean esteCuloareInchisa()
 {
  String culoriInchise[]=
        {"NEGRU","negru","ALBASTRU","albastru"};//etc.
  for(int i=0;i<culoriInchise.length;i++)
    if(culoare.equals(culoriInchise[i]))return true;
  return false;
 }
 public void afisare()
 {
  System.out.println("Culoare="+culoare+" diametru="+diametru);
 }
}

class TestColectii
{
 public static void main(String args[]) throws IOException
 {
  int N=4;
  ArrayList<Umbrela> al_u = new ArrayList<Umbrela>();
  String culori[]={"negru","verde","albastru","galben"};
  Random r=new Random();
  int i;
  for(i=0;i<N;i++)
    al_u.add(new Umbrela(culori[r.nextInt(N)],25*(i+1)));
  afisareColectie(al_u);
  //Asteapta apasarea unei taste:
  System.in.read();
  System.out.println("colectia Vector:");
  Vector<Umbrela> v_u = new Vector<Umbrela>();
  for(i=0;i<N;i++)
     v_u.add(new Umbrela(culori[r.nextInt(N)],20*(i+1)));
  afisareColectie(v_u);
  System.in.read();
  System.out.println("colectia LinkedList:");
  LinkedList<Umbrela> l_u = new LinkedList<Umbrela>();
  for(i=0;i<N;i++)
      l_u.add(new Umbrela(culori[r.nextInt(N)],30*(i+1)));
  afisareColectie(l_u);
 }
```

```
 private static void afisareColectie(List<Umbrela> l)
 {
  Iterator i=l.iterator();
  while(i.hasNext()){
     Umbrela obj=(Umbrela)i.next();//next() returneaza un Object
     obj.afisare();
  }
 }
}
```

3. Probleme propuse

Problema 1

Se citesc nişte numere naturale (până se tastează -1). Se va folosi un obiect ArrayList pentru a memora numerele citite.
Să se afişeze apoi doar numerele prime din ArrayList.

Problema 2

Se citesc intr-o colecţie ArrayList nişte nume (până se tastează "stop"). Sa se afişeze dacă numele sunt în ordine alfabetică sau nu.

Problema 3

Pentru problema 6 (cea cu colecţiile de umbrele) să se scrie o metodă pentru a scoate primul obiect din colecţie:

void scoatePrimulElement(List<Umbrela> l)

Se vor afişa restul elementelor din colecţie.
Pentru un N mai mare comparaţi timpii de execuţie pentru fiecare din cele trei colecţii (se va folosi metoda *System.getCurrentTimeMillis()*).