Tablouri şi Liste

1 Tablouri

Un tablou poate stoca valori de acelaşi fel. Fiecare valoare poate fi accesată prin specificarea unui indice. "Tablou" în Java înseamnă *aproximativ* acelaşi lucru ca tablou, matrice sau vector în matematică.
Spre deosebire de matematică, un tablou Java trebuie declarat şi trebuie să i se aloce o cantitate fixă de memorie.

1.1 Declararea unui tablou

Un tablou este ca alte variabile – trebuie declarat, adică trebuie specificat tipul elementelor din tablou. Toate elementele trebuie să fie de acelaşi tip. Declaraţia alocă doar suficient spaţiu pentru o referinţă la un tablou (tipic 4 octeţi), dar nu creează efectiv obiectul tablou.
Pentru a declara un tablou de elemente, se foloseşte următoarea sintaxă:
<TipulElementelor> [] <nume_variabila>

Exemple:
```
String[]  args;   // args este un tablou de String
int[]     scores; // scores este un tablou de int
JButton[] bs;     // bs este un tablou de JButton
```

Nu se specifică dimensiunea în declaraţie. Spre deosebire de alte limbaje, nu se pune niciodată dimensiunea tabloului în declaraţie, deoarece o declaraţie de tablou specifică doar tipul elementelor şi numele variabilei.

Alocaţi obiectul tablou cu new. Un tablou se creează cu new. Exemplul următor creează un tablou de 100 elemente de tipul int, de la a[0] la a[99].

```
int[] a;           // Declara a ca fiind un tablou de int
a = new int[100];  // Aloca un tablou de 100 int
```

Acestea sunt adesea combinate într-o singură linie.

```
int[] a = new int[100];  // Declara si aloca.
```

Indici
Indicii sunt incluşi între paranteze pătrate „[]”. x_i din matematica este x[i] în Java.
Gamele pentru indici încep întotdeauna la zero nu de la 1, deoarece Java provine în mare parte din C.
Java verifică întotdeauna legalitatea indicilor pentru a se asigura ca indicele este >= 0 şi mai mic decât numărul de elemente din tablou. Dacă indicele este în afara acestui interval, Java arunca o excepţie de tipul `ArrayIndexOutOfBoundsException`. Atentie, această exceptie este aruncată numai la rulare, nu este eroare de compilare. Acest comportament este net superior celui din C şi C++, limbaje care permit referinţe în afara gamei corecte. În consecinţă, programele Java sunt mult mai puţin susceptibile la erori şi carenţe de securitate decât programele C/C++.

Lungimea unui tablou
Fiecare tablou are o variabilă instanţă constantă (final) care conţine lungimea tabloului. Puteţi afla câte elemente poate păstra un tablou scriindu-i numele urmat de **.length**. În exemplul anterior, `a.length` ar fi 100. Atentie, tineţi minte că acesta este numărul de elemente din tablou si este cu 1 mai mult decât indicele maxim!

Idiomul Java pentru ciclarea peste un tablou
Cea mai frecventă folosire a lui .length se regăseşte condiţia de testat în buclele for. Există două variante de a parcurge un tablou:

```
for (int i=0; i < a.length; i++)
{
    . . .
}
```
SAU
```
for (int v : a)
{
    . . .
}
```
Cel de-al doilea exemplu se poate folosi atunci când aveti nevoie *să referiţi valoarea* fiecărui element si nu vă interesează indexul elementului.

Exemplu– Adunarea elementelor unui tablou

Fragmentul următor creează un tablou şi pune 1000 de valori aleatoare în el, apoi adună toate cele 1000 elemente într-o variabilă.
Varianta 1:

```
int[] a;           // Declară un tablou de int
a = new int[1000]; // Crează un tablou de 1000 int.

//... Atribuie valori aleatoare fiecarui element.
for (int i=0; i < a.length; i++)
{
    a[i] = (int)(Math.random() * 100000);  // Numarul aleatoriu in gama 0-99999
}

//... Aduna toate valorile din tablou.
int sum = 0;           // Initializeaza suma totala la 0.
for (int i=0; i<a.length; i++)
{
    sum = sum + a[i];  // Aduna urmatorul element la total
}
```

folosest
Varianta 2:

```
. . .
int sum = 0;        // Initializeaza suma totala la 0.
for (int v : a)
{
    sum += + v;  // Aduna urmatorul element la total
}
```

Valorile iniţiale pentru elementele de tablou – zero/null/false

La alocarea unui tablou (cu new), toate elementele *primesc o valoare iniţială*. Această valoare este 0 dacă tipul este numeric (int, float, ...), `false` pentru boolean şi `null` pentru toate tipurile de obiecte.

Exemplu:
int x = new int[10]; // toate elementele sunt egale cu 0

boolean y = new boolean[10]; // toate elementele sunt egale cu false
Integer z = new Integer[10]; // toate elementele sunt egale cu null.

1.2 Iniţializarea tablourilor

La declararea unui tablou, puteţi şi să alocaţi un obiect tablou preiniţializat în aceeaşi instrucţiune. În acest caz, nu furnizaţi mărimea tabloului fiindcă Java numără valorile din iniţializare pentru a determina mărimea. Spre exemplu,

```
// stil Java 1.0 – mai scurt, dar poate fi folosit DOAR IN DECLARATII
String[] days = {"Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"};
```

Variabilele tablou sunt referite la tablouri

La declararea unei variabile tablou, Java rezervă memorie suficientă doar pentru o referinţă (numele Java pentru adresa sau poanter) la un obiect tablou. Referinţele necesită în mod tipic doar 4 octeţi. La crearea unui obiect tablou cu new se returnează o referinţă, iar acea referinţă poate fi asignată unei variabile. La asignarea unui tablou la un altul, doar referinţa este copiată. Spre exemplu:

```
int[] a = new int[] {100, 99, 98};
int[] b;
// "a" poanteaza spre un tablou, iar "b" nu poanteaza spre nimic
b = a;      // Acum b se refera la ACELASI tablou ca si "a"
b[1] = 0;   // Schimba si a[1] deoarece a si se se refera la acelasi tablou.
// Atât a cât si b se refera la acelasi tablou cu valorile {100, 0, 98}
```

1.3 Noţiuni mai complexe

Tablouri anonime
Java 2 a adăugat tablourile anonime, care vă permit să creaţi un tablou de valori nou oriunde în program, nu doar într-o iniţializare într-o declaraţie.

```
// Acest stil anonim de tablou poate fi folosit si in alte instructiuni.
String[] days = new String[] {"Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"};
```

Sintaxa tablourilor anonime poate fi folosită şi în alte părţi ale programului. Spre exemplu:

```
// In afara declaratiei puteti face aceasta atribuire.
x = new String[] {"Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"};
```

Trebuie să aveţi grijă să nu creaţi aceste tablouri anonime în bucle sau ca variabile locale, deoarece fiecare folosire a lui new va crea un alt tablou.

Alocarea dinamică
Deoarece tablourile sunt alocate dinamic, valorile de iniţializare pot fi expresii arbitrare. Spre exemplu, apelul următor creează două tablouri noi pe care le transmite ca parametri lui drawPolygon.

```
g.drawPolygon(new int[] {n, n+45, 188}, new int[] {y/2, y*2, y}, 3);
```

Declaraţii de tablouri în stil C
Java vă permite şi să scrieţi parantezele pătrate după numele variabilei în loc să le scrieţi după tip. Acesta este modul în care se scriu declaraţiile de tablouri în C, *dar nu constituie un stil bun pentru Java.* Sintaxa C poate arata foarte urât având o parte a declaraţiei înainte de variabilă şi o parte după. Java are un stil

mult mai curat, în care toată informaţia de tip poate fi scrisă fără a folosi o variabilă. Sunt situaţii în care acest stil – Java – este singura notaţie care se poate folosi.

```
int[] a;   // stil Java -- bun
int a[];   // stil C -- legal, dar nerecomandat
```

1.4 Probleme frecvente la tablouri

Câteva probleme frecvent întâlnite la folosirea tablourilor sunt:

- Se uita că indicii încep de la zero.
- Se scrie a.length() în loc de a.length. Metoda length() este folosită la String, nu la tablouri.
- Declararea unui tablou cu mărime. D.e., int[100] a; în loc de int[] a = new int[100];

folose ş t}

1.5 Tablouri multidimensionale

Tablourile din Java sunt de fapt tablouri liniare, unidimensionale. Cu toate acestea, se pot construi tablouri cu mai multe dimensiuni, întrucât exista posibilităţi de creare a lor.

Exemplele care urmează folosesc toate tablouri bidimensionale, dar sintaxa şi codificarea se poate extinde pentru oricâte dimensiuni. Prin convenţie, tablourile bidimensionale au linii (orizontale) şi coloane (verticale). Primul indice selectează linia (care este un tablou monodimensional în sine), iar cel de-al doilea selectează elementul în acea linie/acel tablou.

Declararea şi alocarea unui tablou bidimensional

Sintaxa pentru a declara si aloca un tablou bidimensional este:
<TipulElementelor>[][] <NumeVariabila> = new <TipulElementelor>[<nrRanduri>][nrColoane]
Exemplu:

```
int[][] board = new int[3][2];
```

Valori iniţiale

Se pot asigna valori iniţiale tabloului la declararea într-un mod foarte asemănător cu cel pentru tablourile monodimensionale. Dimensiunile sunt calculate din numărul de valori.

```
int[][] board = new int[][] {{0,0,0},{0,0,0},{0,0,0}};
```

Trebuie să daţi o valoare unui element înainte de a-l folosi, fie printr-o iniţializarea, fie printr-o asignare.

2 Colecţii

O colecţie este o secventă de elemente, care indeplinesc una sau mai multe reguli. Toate tipurile de tip colecţie implementează interfaţa Collection.

2.1 Liste

Interfata List este interfata ce este implementată de clasele ArrayList, LinkedList, Stack, Vector si alte clase mai putin utilizate.
Această interfată defineste o serie de metode ce pot să fie folosite de orice obiect ce implementează interfata List.

2.1.1 ArrayList

Clasa ArrayList este o implementare a clasei List. Această clasă se foloseşte atunci când se doreste să se acceseze elemente random din colecţii, în schimb are dezavantajul că este mai inceată la inserarea si stergerea de elemente în mijlocul colecţiei.

2.1.2 LinkedList

Clasa LinkedList este o altă implementare a clasei List. Această clasă ar trebui să fie folosită atunci când se doreste inserarea si stergerea de elemente din mijlocul colecţiei, dar are dezavantajul că este mai inceată la accesarea elementelor random din colecţii.

2.1.3 Stack

Clasa Stack mai este cunoscută si ca LIFO (Last in, First out), deoarece primul element ce este introdus în stack, este ultimul element ce se poate scoate din colecţie. Clasa LinkedList are niste metode ce simuleaza un Stack.

Exemplu:

```
public class StackTest {
        public static void main(String[] args) {
                Stack stack = new Stack();
                for(String s : "My dog has fleas".split(" "))
                        stack.push(s);
                while(!stack.empty())
                System.out.println(stack.pop() + " ");
        }//main
}//class
```

Intrebare:
Care este rezultatul funcţiei main?

Problema:
Simulati comportamentul unui Stack folosind metodele clasei LinkedList.

2.1.4 Metode specifice claselor ce implementeaza interfata List

2.1.4.1 Adăugarea de elemente

Adăugarea de elemente se poate face folosind una din metodele

- add(Element e) –adaugă un element la sfarsitul listei
- add(int index, Element e) – adaugă elementul e pe pozitia index
- addAll(Collection collection) – adaugă toate elementele colecţiei collection la sfarsitul colecţiei curente
- adAll(int index, Collection collection) - adaugă toate elementele colecţiei collection pe pozitia index în colecţia curentă.

Exemplu:

```
ArrayList group = new ArrayList();
group.add("primul element");
group.add(4);
group.add(new Object());

ArayList extendedGroup = new ArrayList();
extendedGroup.addAll(group);
```

Intrebari:

```
Ce elemente va contine group?
```

```
Ce elemente va contine extendedGroup?
```

2.1.4.2 Verificarea existentei unui element în colecţie

Pentru a verifica dacă exista sau nu element într-o colecţie, se poate folosi una dintre metodele:

- contains(Element e) – verifica dacă elementul e apare în colecţia curenta
- containsAll(Collection collection) – verifica dacă toate elementele din colecţia collection apar în colecţia curenta
- indexOf(Element e) – returneaza indexul la care se afla elementul e în colecţia curenta; dacă nu exista, atunci returneaza -1.

Exemplu:

```
ArrayList group = new ArrayList();
group.add("primul element");
boolean x = group.contains("primul element);
boolean y = group.contains(1);
group.add(new Object());
boolean z = group.contains(new Object());
```

```
Intrebari:
Ce valori vor avea x,y,z?
```

2.1.4.3 Stergerea elementelor dintr-o colecţie

Pentru a sterge niste elemente dintr-o colecţie, se poate folosi una din metodele:

- clear() – sterge toate elementele din colecţia curenta
- remove(int index) – sterge elementul de pe pozitia index din colecţia curenta
- remove(Element e) – sterge elementul e din colecţia curenta
- removeAll(Collection) – sterge toate elementele din colecţia curenta

Exemplu:

```
ArrayList group = new ArrayList();
group.add("primul element");
group.add(4);
group.add(new Object());
remove(1);
remove(new Object());
```

Intrebari:
Ce valoarea va avea group dupa apelarea lui remove(1)?
Ce valoarea va avea group dupa apelarea lui remove(new Object())?

Problema: aflati care este diferenta dintre operatia removeAll() si clear().

2.1.4.4 Alte metode

Pe langa metodele prezentate mai sus, mai exista urmatoarele metode ce pot sa fie utile:

- get(int index) – returneaza elementul de pe pozitia index
- size() – returneaza lungimea colecţiei
- sublist(int fromIndex, int toIndex) – returneaza o colecţie ce contine elementele colecţiei curente incepand cu indexul fromIndex si pana la indexul toIndex.
- toArray() – returneaza un sir de elemente ce va contine toate elementele din colecţia curenta

2.2 Multimi (Set)

Interfata Set este o lista speciala care nu permite elemente duplicate. Exista mai multe tipuri de clase ce implementeaza aceasta interfata, dar cele mai folosite sunt HashSet si TreeSet. Diferenta dintre implementari este modul în care este implementata ordinea elementelor, HashSet foloseşte funcţionalitea de hashing, pe când TreeSet foloseşte funcţionalitatea de arbore rosu-negru. O alta diferenta dintre clase este performanta în diferite scenarii: HasSet este superior la TreeSet la adaugarea şi căutarea de elemente. TreeSet în schimb are avantajul ca tine elementele sortate.

2.2.1 HashSet

Clasa HashSet foloseşte funcţionalitatea de hashing, adica căutam un obiect dupa un alt obiect. Pentru ca compilatorul sa stie unde stocheaza şi de unde să aduca un obiect de tip X, clasa X trebuie să implementeze o metoda speciala numita hashCode(), dar şi metoda equals().

Exemplu:

```
public class Person {
	private String name;

	@Override
	public int hashCode() {
		return 31 + ((name == null) ? 0 : name.hashCode());
	}

	@Override
	public boolean equals(Object obj) {
		Person other = (Person) obj;
		if (name == null) {
			if (other.name != null)
				return false;
		} else if (!name.equals(other.name))
			return false;
		return true;
	}
}
```

2.3 Lucrul cu colecţii

Pentru lucrul cu colecţii este necesar să se facă import la pachetul **java.util.***;

Următorul exemplu de cod prezintă crearea unui grup de studenți și accesarea elementelor din listă. Astfel, au fost create două clase: o clasă Student, pentru definirea unui element **Student** și o clasă **GrupDeStudenți**, pentru a implementa accesarea elementelor din listă.

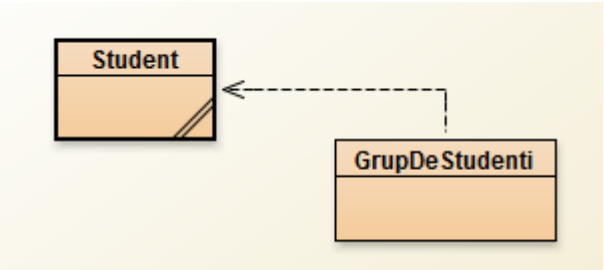

Fig. 1 Diagrama de clase

```
/**
 * Clasa Student
 */
public class Student
{
    private String nume;

    //Constructor
    public Student(String nume)
    {
        this.nume = nume;
    }

    //Modifica numele persoanei
    public void setNume(String nume)
    {
        this.nume = nume;
    }

    //acceseaza numele persoanei
    public String getNume()
    {
        return this.nume;
    }
}
```

//

```
/**
 * Clasa ce implementeaza un grup de studenti
 */
import java.util.*;

public class GrupDeStudenti
{
    public static void main(String[] args)
    {
        ArrayList<Student> grup = new ArrayList<Student>();
        grup.add(new Student("Popescu Ion"));
        grup.add(new Student("Ionescu Maria"));
        grup.add(new Student("Tudor Vasile"));
        grup.add(new Student("Pop Marius"));
        grup.add(new Student("Rus Andrei"));

        //parcurgere lista folosind FOR
        System.out.println("Parcurgere FOR1:");
        for (int i = 0; i < grup.size(); i++)
        {
            Student st = (Student)grup.get(i);
```

```
            System.out.println(st.getNume());
        }

        //parcurgere cu FOR, fara a specifica lungimea listei
        System.out.println("Parcurgere FOR2:");
        for (Student st : grup)
        {
            System.out.println(st.getNume());
        }

        //parcurgere cu Iterator
        System.out.println("Parcurgere cu Iterator:");
        Iterator it = grup.iterator();
        while(it.hasNext())                  // verifica daca exista un
element urmator
        {
            Student st = (Student)it.next(); // returneaza urmatorul element
            System.out.println(st.getNume());
        }
    }
}
```

3 Map

Map-urile sunt grupuri de tip cheie-valoare, care permit gasirea unei valori pe baza unei chei. Un exemplu în care am folosi Map-urile ar putea fie de cate ori apare un cuvant într-un text: cheia ar fi cuvant, iar valoarea ar fi numarul de aparitii. Exista mai multe implementari a interfetei Map, dar cele mai utilizate sunt HashMap şi SortedMap. Diferenta dintre existentele implementari ale interfetei sunt date de eficienta în diferite scenarii, de ordinea în care perechile (cheie, valoare) sunt stocate, cum funcţioneaza map-ul în multithread.

3.1 HashMap

La fel ca şi la HashSet, HashMap-ul foloseşte funcţionalitatea de hashing. Pentru a putea să foloseşti în mod eficient o clasa ca şi key într-un HashMap este nevoie ca clasa respectiva să implementeze metodele hashCode şi equals.

3.2 TreeMap

TreeMap foloseşte în implementare arborele rosu-negru. Pentru ca o clasa să poata fi folosita ca şi cheie într-un TreeMap, clasa trebuie să implementeze interface Comparable şi să implementeze metoda equals().

3.3 Metode specifice claselor ce implementeaza interfata Map

3.3.1 Adaugarea de elemente

Adaugarea de elemente se poate face folosind una din metodele

- put(K cheie, V valoare) –adauga o noua pereche (cheie,valoare) în map.
- putAll(Map newMap) – adauga toate perechile din map-ul newMap în map-ul curent.

Exemplu:

```
Map group = new HashMap();
group.put("ana", 1);
group.put("are", 1);
group.add("ana", 2);

Map extendedGroup = new HashMap();
extendedGroup.putAll(group);
```

Intrebari:
Ce elemente va contine group?
Ce elemente va contine extendedGroup?

3.3.2 Verificarea existentei unui element în map

Pentru a verifica daca exista sau nu element într-un map, se poate folosi una dintre metodele:

- containsKey(Element key) – verifica dacă exista vreo pereche ce are ca şi cheie valoarea e
- get(Element key) – returneaza valoarea din map a carui cheie este

Intrebari:
Pentru exemplul de mai sus, ce returneaza:

```
group.containsKey("ana");
group.get("ana");
```

3.3.3 Stergerea elementelor dintr-un map

Pentru a sterge niste elemente dintr-un map, se poate folosi una din metodele:

- clear() – sterge toate elementele din map-ul curenta=
- remove(Element key) – sterge perechea (cheie, valoare) care are ca şi cheie key.

Intrebari:
Pentru exemplul de mai sus, ce returneaza:

```
group.remove("ana");
group.remove("mere");
```

3.3.4 Alte metode

Pe langa metodele prezentate mai sus, mai exista urmatoarele metode ce pot să fie utile:

- size() – returneaza lungimea colecţiei
- keySet() – returneaza toate cheile din map-ul curent
- values() – returneaza toate valorile din map-ul curent
- isEmpty() – returneaza true dacă map-ul curent este gol, false în caz contrar.

3.4 Lucrul cu map-uri

Pentru a putea lucra cu map-urile este nevoie să se importe pachetul import java.util.*;
Exemplul urmator afiseaza de cate ori apare un cuvant într-un text.

```
public class FindIndexes {

	public Map count(String input) {
		Map words = new HashMap();
		for (String word : input.split(" ")) {
			if (words.containsKey(word)) {
				Integer noAppearances = (Integer) words.get(word);
				words.put(word, noAppearances + 1);
			} else {
				words.put(word, 1);
			}
		}
		return words;
	}

	public void displayWords(Map words) {
		for (Object word : words.keySet()) {
			System.out.println("The word " + word + " appeared "
					+ words.get(word) + " times");
```

```
            }
        }

        public static void main(String[] args) {
            FindIndexes f = new FindIndexes();
            Map indexes = f.count("am mers la curs si am invatat java");
            f.displayWords(indexes);
        }
}
```

Problema:
Implementati clasa de sus, dar folositi ca şi cheie în loc de String, un obiect de tipul Dog, unde Dog are 2 parametri: rasa şi numele. Ne intereseaza să aflam cati caini exista din aceeasi rasa şi care să aiba acelasi nume. Incercati 2 variante de implementare şi vedeti care este diferenta de afisare:

1. Nu implementati metodele equals şi hashCode pentru clasa Dog.
2. Implementati metodele equals şi hashCode pentru clasa Dog.

4 Metode de bibliotecă pentru tablouri

Există metode de bibliotecă, statice pentru manipularea tablourilor în clasa **java.util.Arrays**.

Arrays.asList()	Returnează un List (listă) pe baza tabloului.
Arrays.toString()	Returnează o formă „citibilă” a tabloului.
Arrays.binarySearch()	Execută o căutare binară pe un tablou sortat.
Arrays.equals()	Compară două tablouri dacă sunt egale..
Arrays.fill()	Umple tot tabloul sau o subgamă cu o valoare.
Arrays.sort()	Sortează un tablou.

În plus, există metoda System.arrayCopy().

Exemplu:
```
String[] words = new String[] {"ana", "are", "mere"};
List wordsAsList = Arrays.asList(words);
System.out.println(wordsAsList);
```

5 Mersul lucrării

1.1. Studiaţi şi înţelegeţi textul şi exemplele date.

1.2. Definiţi o clasă Matrix care să implementeze operaţiile de adunare, scădere, înmulţire de matrice, precum şi împărţirea cu un scalar.

1.2.1. Implementaţi problema utilizând tipul de date tablou.
1.2.2. Implementati problema folosind una dintre clasele ce implementează colecţiile de obiecte (ArrayList, Vector etc.).

1.3. Implementati o clasa PetHotel care să simuleze un registru cu toti cainii ce sunt cazati în hotel.

1.4. Extindeti clasa PetHotel ca să mai existe un registru ce contine informatii despre perioadele la care a fost cazat un caine.

1.5. Implementaţi o clasă ChessBoard care să păstreze poziţii pe tabla de şah. Figurile şi pionii sunt şi ei clase. Verificaţi corectitudinea mutărilor.

1.5.1. Implementaţi problema utilizând tipul de date tablou.
1.5.2. Implementati problema folosind una dintre clasele ce implementează colecţiile de obiecte (ArrayList, Vector etc.).