Laborator 2

Tablouri

In acest laborator vom lucra cu tablouri unidimensionale (cunoscute si ca vectori) sau bidimensionale (cunoscute si ca matrice).
Aceste tablouri, de care ne-am lovit si in laboratorul 1, sunt in fapt zone de memorie continue in care pot fi depozitate obiecte. La fel ca si in cazul variabilelor normale, daca avem un vector de primitive (int[], char[]) atunci la adresa respectiva se afla un tablou capabil sa stocheze datele efectiv. In cazul in care avem un vector de obiecte sau un vector de vectori, la adresa respectiva se afla un tablou de pointeri de tip corespunzator. In java nu exista tablouri multidimensionale. In schimb, putem sa declaram vectori de vectori, care au in fapt acelasi efect, cu mici diferente.
Sa luam urmatorul exemplu:

```
public void methodName()
{
int[] vector;
//…
}
```

La intrarea in aceasta metoda, va fi alocat spatiu pentru un pointer la un tablou de intregi, alocarea facandu-se pe stiva.

```
public void methodName()
{
int[] vector;
vector = new int[100];
//…
}
```

Pe urmatoarea linie vedem cum se aloca spatiu pentru stocarea a 100 numere intregi. Cateva precizari:

1. Pentru ca spatiul de memorie alocat este continuu, trebuie sa precizam la alocare dimensiunea exacta a tabloului. Aceasta dimensiune este apoi fixa
2. Spatiul de memorie alocat in acest fel este in HEAP, la fel ca oricare alocare dinamica.

3. Deoarece tipul elementelor din vector este un tip primitiv, spatiile de memorie vor contine efectiv date
4. Elementele tabloului vor fi initializate cu “valoarea default 0” adica in cazul numerelor intregi chiar numarul 0.

In momentul in care incercam sa afisam la ecran elementul de pe pozitia 10 sa spunem:

```
System.out.println(vector[10]);
```

vom obtine valoarea 0 in consola.

Acum sa luam un alt exemplu:

```
public void methodName()
{
String[] vector;
//…
}
```

La intrarea in aceasta metoda, va fi alocat spatiu pentru un pointer la un tablou de OBIECTE de tip String, alocarea facandu-se pe stiva.

```
public void methodName()
{
String[] vector;
vector = new String[100];
//…
}
```

Pe urmatoarea linie vedem cum se aloca spatiu pentru stocarea a 100 de POINTERI la obiecte de tip String. Cateva precizari:

1. Spatiul de memorie este alocat similar ca mai sus
2. Memoria este alocata tot in HEAP
3. Deoarece tipul elementelor NU este un tip primitiv, spatiile de memorie vor contine pointeri
4. Elementele vor fi initializate cu “valoarea default 0” care in cazul pointerilor este `null`

De aceasta data, daca incercam sa afisam la ecran valoarea de pe pozitia 10 in felul urmator:

```
System.out.println(vector[10].toString());
```

vom obtine un null pointer exception.

Putem acum sa facem o analogie:

1. int este tip de date (primitiv)
2. String este un tip de date (obiect)
3. int[] este un tip de date (array de primitive, adica un obiect)
4. String[] este un tip de date (array de obiecte, adica un obiect)

Deci nu este deloc ciudat sa vedem ceva de genul:

```
int[][] matrice;
```

care de fapt este un vector de vectori de int, iar pentru a face lucrurile mai clare il putem citi (atentie, nu este corect sintactic) asa:

```
(int[])[] matrice;
```

si astfel vedem ca este un vector de vectori. Acest rationament ne ajuta la a citi corect matrice mai complexe. Exercitiu:

Fie:
`int[][][][] matrice;`
Ce tip are:

1. matrice[0] ?
2. matrice[0][1]
3. matrice[0][1][2]?
4. matrice[0][1][2][3]?

Nu uitam de asemenea de procesul de initializare. In cazul matricelor multidimensionale in C, initializarea era facuta similar pentru o matrice multidimensionala de dimensiuni m,n ca a unui vector de dimensiuni m*n. In java in schimb nu avem matrice multidimensionale! Deci pentru o matrice cu doua dimensiuni de tip int, initializarea va fi facuta doar pentru vectorul de vectori, si mai mult initializarea va fi facuta cu pointerul null! Deci pentru a initializa o matrice multidimensionala de int in Java, trebuie sa lucram un pic, sa luam fiecare element al vectorului de vectori si sa il initializam folosind new int[dimensiune]. In practica insa, aceasta forma nu este deloc deranjanta si ne permite in plus sa avem matrice multidimensionale in care liniile au dimensiuni diferite intre ele.
Exercitiu: cum ar trebui sa initializam o matrice de tipul celei de mai sus, cu 4 dimensiuni?

Exercitiu 1:

Haideti sa facem urmatorul program/exercitiu.
Program de management al cumparaturilor clientilor unui magazin. Avem urmatoarele componente:

1. O clasa care se numeste Client, cu campul (private)
    a. String nume

    si orice alte metode mai sunt necesare
2. O clasa ProdusCumparat, cu campurile (private)
    a. String nume
    b. int cantitate

    si orice alte metode sunt necesare
3. O clasa care se numeste Magazin si are urmatoarele campuri:
    a. vector de Client
    b. matrice bidimensionala de ProdusCumparat (numita produseCumparate)

    si urmatoarele metode:
    a. userNou(String nume), care adauga un nou Client (atentie la redimensionarea vectorilor si returneaza Client-ul adaugat
    b. gasesteClient(String nume), care cauta un client dupa nume si returneaza un numar reprezentand pozitia acestuia in vector sau -1 daca aceasta nu a fost gasita.
    c. clientCumpara(String nume, String numeProdus, int cantitate) care cauta un client dupa nume, daca nu exista acest client, insereaza unul nou (folosind functia userNou) cauta un ProdusCumparat, in vectorul produseCumparate[numarUser] unde numarUser este valoarea returnata de

gasesteClient(nume), si daca exista adauga cantitate la ProdusCumparat.cantitate valoarea parametrului cantitate. Daca ProdusCumparat nu exista, se adauga unul in lista.

Pentru a simplifica lucrurile, java contine o clasa numita Arrays cu o multitudine de metode statice care ne ajuta sa manipulam vectori.
Pentru o lista completa de functionalitate, puteti sa mergeti la pagina:
https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html
dar, pe scurt, avem urmatoarele metode interesante:

1. **copyOf** care aloca memorie pentru un nou vector de dimensiunea solicitata si copiaza elementele vectorului dat ca parametru (in masura posibilitatilor) in noul vector.
2. **deepEquals** care returneaza true daca toate elementele celor doi vectori, aflate pe pozitii identice sunt egale.
3. **fill** care “umple” un array cu elementul primit ca parametru
4. **sort** care sorteaza un array primit ca parametru, folosind fie ordinea naturala intre elemente sau un Comparator oferit ca parametru (explicatii suplimentare dupa ce facem mostenirea)

In cazul nostru, copyOf ne va ajuta foarte mult in situatiile in care trebuie sa realocam memorie pentru a adauga un nou Client/ProdusCumparat la cei doi vectori.
Exercitiu: cum asezam clasele in pachete?

Citirea datelor de la tastatura

Citirea datelor de la tastatura (mai precis de la consola) este un proces simplu dar foarte rar intalnit. In primul laborator am vazut cum se afiseaza la ecran (mai precis in consola, dar si mai precis la standard output) folosind o metoda din System.out. In acest laborator vom folosi System.in, care este o instanta a unui obiect de tipul InputStream, obiect care este capabil sa ne furnizeze la cerere octeti. Acesti octeti pot fi cititi in ce mod dorim si putem sa le dam ce interpretare dorim. Deoarece lucrul cu octeti este dificil, vom folosi o clasa care ne ajuta sa citim mai usor date de la tastatura. Aceasta clasa este clasa Scanner.
Clasa scanner trebuie privita ca un obiect care se ataseaza la un obiect pentru a da forma datelor venite de la acesta. Sa facem urmatoarea analogie. Sa spunem ca vrem sa facem prajituri si sa decoram aceste prajituri cu frisca. Avem la dispozitie o “seringa” mare in care punem frisca si pe masura ce apasam un piston frisca iese. Forma in care iese frisca este una plictisitoare si rotunda. Din acest motiv, aplicam seringii un cap in forma de stea, pentru a schimba aspectul. Materialul in sine este acelasi, insa i se da o alta forma. In acelasi mod, System.in ne ofera date, iar Scanner este capabil sa le dea forma pe care o dorim. De exemplu:

```
Scanner scanner = new Scanner(System.in);
int numar = scanner.nextInt();
String nume = scanner.next();
```

Observati cum pe prima linie am declarat un obiect de tip scanner si apoi am construit o instanta de tip Scanner. Cand am construit instanta, am dat constructorului lui Scanner ca parametru pe System.in.

Pe linia urmatoare, am declarat o variabila de tip intreg si apoi am citit un intreg din scanner. **Este important de mentionat faptul ca Scanner lucreaza doar cu date de tip text**. Adica el va citi din System.in datele ca text si le va interpreta. Deci cand noi spunem cerem nextInt(), un sir de caractere (numit token) este parsat ca numar intreg. Vom reveni asupra lui Scanner si altor clase similare atunci cand ajungem la fluxuri.
Ca de obicei, cea mai buna sursa de documentare pentru clasa Scanner este pagina oficiala Java:
https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html
Haideti sa facem urmatoarea modificare programului.

1. Adaugam o functie main. (unde o adaugam, in ce clasa?)
2. In functia main, declaram si construim o instanta a clasei Magazin.
3. Afisam un meniu cu urmatoarele optiuni:
    a. 1=Afisare clienti
    b. 2=Adaugare cumparaturi
    c. 3=Adaugare client
4. Citim un numar intreg de la tastatura.
    a. Daca se citeste “1” atunci se afiseaza o lista cu toti clientii, sortati alfabetic (cum putem sorta usor clientii? Ce array sortam, daca nu vrem sa se modifice arrayul nostru?
    b. Daca se citeste “2” atunci se afiseaza “Cine cumpara?” si se citeste un string (a) apoi se afiseaza “Ce cumpara?” si se citeste un string (b) apoi se afiseaza “Cat cumpara?” si se citeste un int (c). Apoi este apelata pe magazin metoda clientCumpara cu parametrii a,b,c.
    c. Daca se citeste “3” atunci se afiseaza “Introduceti numele clientului” si se citeste un string care este folosit apoi in apelul metodei userNou.
5. Daca se citeste altceva in afara de 1,2,3, se iese din program (cum iesim din program?). Altfel, dupa apelul metodei potrivite se sare din nou la pasul 3.

Exercitiu: Cum am putea sa facem codul mai bine structurat, mai usor de citit?

Generarea numerelor pseudo-aleatoare

Exista mai multe metode pentru generarea numerelor pseudo-aleatoare in java. Una din aceste metode este folosirea clasei Random. Pentru a genera un numar aleator trebuie sa obtinem o instanta a clasei Random pe care ulterior putem apela diverse metode.

```
Random random = new Random();
int numar = random.nextInt();
boolean bool = random.nextBoolean();
double intreZeroSiUnu = random.nextDouble();
```

Exercitiu: modificati programul de mai sus astfel incat la optiunea “2” sa nu se mai citeasca de la tastatura un numar pentru cantitate, ci sa se genereze aleatoriu o cantitate intre 0 si 9 (inclusiv).

Date

O “data” este folosita pentru a reprezenta un moment in timp, cu o anumita precizie (“data” inseamna in general precizie de o zi, in timp ce “ora” inseamna precizie de un minut sau chiar milisecunda). In toate cazurile, acest “timp” este raportat la ceva. De exemplu, anul 2018 inseamna ca au trecut 2018 ani de la un moment identificat ca nasterea lui Iisus Hristos in crestinism. Inainte de asta, punctul de origine era “facerea lumii”. Din acest motiv nu ne este greu sa acceptam faptul ca obiectul Date, unul din cele mai simple obiecte folosite pentru stocarea “datei”, are ca punct de referinta epoca Unix, adica datele sunt raportate la 1 ianuarie 1970, cu precizie de milisecunda.
Date este forma cea mai simpla de stocare a unei date si are la baza un numar de tip long, reprezentand numarul de milisecunde de la 1 ianuarie 1970. De cele mai multe ori este suficienta.
O clasa mult mai complexa este clasa Calendar si subclasa ei GregorianCalendar. Aceste doua clase contin inclusiv functii pentru manipularea (modificarea) datelor (cum ar fi adaugarea de 2 ore si 15 minute, sau gasirea primei zi de luni dupa data respectiva etc).
Puteti gasi mai multe informatii aici:
https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html

Exercitiu 2:

Pentru acest laborator, haideti sa modificam clasa “ProdusCumparat” astfel incat sa contina si un camp de tip Date, numit ultimaAchizitie, care sa fie modificat de fiecare data cand un client cumpara un produs de acel tip, si sa primeasca data curenta (cu precizie de milisecunda). Hint: new Date() va produce o data reprezentand momentul curent sau Calendar.getInstance().getTime() va avea un efect similar.