

2. CONTROLUL EXECUTIEI. TABLOURI

2.1. Controlul executiei programului.....	<u>2</u>
2.1.1. Aspecte generale.....	<u>2</u>
2.1.2. Instructiuni decizionale.....	<u>2</u>
2.1.2.1. Instructiunea if.....	<u>2</u>
2.1.2.2. Instructiunea switch.....	<u>3</u>
2.1.3. Instructiuni repetitive (bucle).....	<u>5</u>
2.1.3.1. Instructiunile while si do...while.....	<u>5</u>
2.1.3.2. Instructiunea for.....	<u>5</u>
2.1.3.3. Modificarea iterarii: break si continue.....	<u>7</u>
2.2. Tablouri (arrays).....	<u>7</u>
2.2.1. Concepte.....	<u>7</u>
2.2.2. Tablouri unidimensionale.....	<u>8</u>
2.2.2.1. Declararea unei variabile de tip tablou unidimensional.....	<u>8</u>
2.2.2.2. Crearea obiectului tablou.....	<u>8</u>
2.2.2.3. Accesarea membrilor tabloului.....	<u>9</u>
2.2.2.4. Popularea tabloului.....	<u>9</u>
2.2.2.5. Parcurgerea tabloului.....	<u>9</u>
2.2.3. Tablouri multidimensionale.....	<u>10</u>
2.2.3.1. Concepte.....	<u>10</u>
2.2.3.2. Declararea variabilei de tip tablou.....	<u>10</u>
2.2.3.3. Crearea obiectului de tip tablou.....	<u>10</u>
2.2.3.4. Accesarea elementelor si popularea tabloului.....	<u>11</u>
2.2.3.5. Parcurgerea tabloului.....	<u>11</u>
2.2.4. Tablourile si operatorul de atribuire.....	<u>12</u>
2.2.4.1. Tablourile nu se pot copia prin simpla atribuire.....	<u>12</u>
2.2.4.2. Operatii de atribuire corecte.....	<u>12</u>
2.2.5. Clasa Arrays.....	<u>13</u>
2.3. BIBLIOGRAFIE.....	<u>14</u>

2.1. Controlul executiei programului

2.1.1. Aspecte generale

In programare este nevoie de alterarea curgerii liniare a programului, fie prin luarea de decizii care vor determina executarea conditionata a unor portiuni de cod, fie prin executarea repetata, in mod controlat, a altor portiuni de cod.

Controlul executiei presupune urmatoarele mecanisme:

- **instructiuni decizionale** - daca in cazul operatorului ternar aveam de-a face cu o *evaluare conditionata* a unei expresii, introducem acum *executia conditionata* a unei secvente de program, prin intermediul a doua instructiuni Java:
 - instructiunea decizionala simpla: **if**
 - instructiunea decizionala multipla: **switch**
- **instructiuni repetitive** – sunt cele care permit executarea unei portiuni de cod in mod repetat fara a fi nevoie sa scriem de tot atatea ori codul in cauza. Structuri Java folosite in acest scop: **for**, **while**, **do...while**
- **exceptii** – mecanism de management al erorilor ce va fi prezentat in cadrul unui capitol ulterior

2.1.2. Instructiuni decizionale

2.1.2.1. Instructiunea if

Instructiunea **if** permite executarea conditionata a una sau mai multe secvente de cod, in functie de rezultatul uneia sau mai multor expresii. Exista trei sintaxe posibile:

- o prima sintaxa care permite executia conditionata a unui bloc de instructiuni, in functie de valoarea expresiei primite ca argument de catre constructia if. Valoarea expresiei trebuie sa fie de tip boolean. Daca expresia se evalueaza ca true, va fi executat blocul de instructiuni ce constituie corpul lui if; daca expresia ia valoarea false, corpul lui if nu se executa

```
// sintaxa 1
if(conditie){
    instructiuni pentru conditie adevarata;
}
```

- o a doua sintaxa in care instructiunea if primeste ca argument tot o expresie cu valoare de tip boolean, insa va executa unul din doua blocuri de instructiuni in functie de valoarea expresiei:

```
// sintaxa 2
if(conditie){
    instructiuni pentru conditie adevarata;
} else
    instructiuni pentru conditie falsa;
```

- o a treia sintaxa permite testarea succesiva a unor conditii eliminatorii si executia conditionata a unuia din mai multe blocuri de instructiuni. Daca expresia 1 se evalueaza ca true, va fi executat primul bloc de instructiuni; in caz contrar se trece la evaluarea celei de-a doua expresii. Daca a doua expresie ia valoarea true, se executa cel de-al doilea bloc de instructiuni; in caz contrar se trece la a treia s.a.m.d. Blocul final else poate lipsi; daca exista, el se executa numai in cazul in care niciuna dintre expresii nu s-a evaluat ca true

```
// sintaxa 3
if(expresie1){
    set de instructiuni 1;
} else if (expresie2){
    set de instructiuni 2;
```

```

}
// ...se introduc cate if-uri este nevoie, iar la final, optional:
else{
    set de instructiuni 3;
}

```

Exemplu:

```

public class Intregi {
    public static void main(String[] args) {
        long x=Long.parseLong(args[0]);
        if(x<=255) System.out.println("Valoarea poate fi stocata ca byte");
        else if(x<65536) System.out.println("Valoarea poate fi stocata ca short");
        else if(x<4294967296L) System.out.println("Valoarea poate fi stocata ca int");
        else System.out.println("Valoarea poate fi stocata ca long");
    }
}

```

2.1.2.2. Instructiunea switch

2.1.2.2.1. Sintaxa si mod de operare

Instructiunea *switch* ofera posibilitatea executarii conditionate a diferite portiuni de cod in functie de valorile posibile ale unei expresii. Sintaxa sa este:

```

switch (expresie) {
    case valoare1:      set instructiuni 1;
    break;
    case valoare2:      set instructiuni 2;
    break;
[...pot urma alte blocuri case...]
    default:           set de instructiuni pentru cazul in care valoarea expresiei nu se
                      regaseste mai sus;
                      break;
}

```

Constructia *switch* va evalua mai intai expresia si va compara pe rand valoarea sa cu valorile prezente in case-uri, in ordinea in care acestea apar in switch. Executia poate evolu in continuare in doua moduri:

- daca este gasita o valoare *case* care corespunde, va fi executat codul incepand cu acea eticheta, pana la urmatoarea instructiune *break* sau *return*
- daca valoarea expresiei nu se regaseste ca parte a niciunui *case*:
 - daca exista clauza *default*, se va executa codul corespunzator ei pana la urmatoarea instructiune *break* sau *return*
 - daca nu exista clauza *default*, constructia *switch* se incheie si executia programului continua cu instructiunile de dedesubtul ei

Expresia testata trebuie sa aiba un rezultat de tip byte, short, int sau char (nu referinte, nu long, double sau float!) sau un tip de date enumerat. **Incepand cu Java 1.5 se accepta si obiecte de tip Byte, Short, Integer, Character, iar incepand cu Java 7 se accepta si obiecte de tip String.** Pentru String-uri, comparatia intre expresia pasata ca argument lui *switch* si fiecare dintre valorile din case va fi efectuata folosind metoda *equals()* din clasa String (vezi lectia despre String).

Atentie! Expresia pasata lui *switch* nu poate avea valoarea null! Acest caz va produce o eroare.

Nota: se observa faptul ca *switch* poate fi emulat cu ajutorul celei de-a treia sintaxe if (if..else if..else if..else). Afirmatia nu este insa adevarata si in sens opus! *switch* compara valoarea expresiei de referinta cu valorile din case-uri folosind operatorul ==, pe cand if..else if permite verificarea oricarui tip de conditie! (vezi exemplul clasei *Intregi* de mai sus)

2.1.2.2. Instructiunea break

Instructiunea *break* cauzeaza iesirea din structura switch si continuarea executiei cu instructiunile de dedesubtul acesteia. In exemplul de mai sus, fiecare bloc de instructiuni corespunzator unui *case* a fost incheiat cu o instructiune *break*. Instructiunea *break* nu este obligatorie la finalul blocului *case*; in cazul in care ea lipseste, vor fi executate in continuare instructiunile ce urmeaza (trecand si “prin dreptul” urmatorului *case* sau al default-ului) pana la intalnirea urmatorului *break* sau *return*.

Atentie! *return* este instructiunea folosita pentru a cauza iesirea dintr-o functie/metoda, si implicit din switch. Daca in blocul de instructiuni din dreptul unui *case* a fost folosit un *return* neconditonal, dupa el nu mai trebuie pus *break*!

Possibilitatea ca *break* sa lipseasca a fost introdusa pentru a putea refolosi cod. Cea mai frecventa utilizare este executarea acelasi secentage de cod pentru mai multe valori distincte ale expresiei testate, insa este posibila si refolosirea partiala:

```
switch (numar) {
    case 2:
    case 4:
        System.out.println("Numarul este par!");
        break;
    case 5:
        System.out.print("Numar prim. ");
    case 1:
    case 3:

        System.out.println("Numarul este impar!");
        break;
}
// in cazul lui 2 si 4 se afiseaza Numarul este par!
// in cazul lui 1 si 3 se afiseaza Numarul este impar!
// in cazul lui 5 se afiseaza Numar prim. Numarul este impar!
```

Nota: putem privi corpul constructiei switch ca pe un unic bloc de instructiuni, in care se poate patrunde prin dreptul diverselor etichete *case*, si din care se poate iesi prin dreptul instructiunilor *break* sau *return*.

2.1.2.3. Clauza default

Clauza default “prinde” toate valorile expresiei care nu se regasesc explicit sub forma de *case*-uri. Ea este optionala - in cazul in care nu exista, compilatorul nu emite nicio eroare.

Clauza default poate fi plasata oriunde in cadrul switch-ului, nu neaparat ultima; plasarea ei mai sus nu va anula efectul *case*-urilor ce-i urmeaza (ex: chiar daca este plasata prima, ea nu “prinde tot”, ci va actiona numai in cazul in care valoarea expresiei nu corespunde cu niciun *case*).

Există două situații în care contează unde este plasată clauza default:

- atunci cand deasupra ei se află un *case* care nu are *break* sau *return*
- atunci cand chiar instructiunile din dreptul clauzei default nu contin *break* sau *return*, iar dedesubt se află una sau mai multe clauze *case*.

2.1.3. Instructiuni repetitive (bucle)

2.1.3.1. Instructiunile while si do...while

Bucla while

```
while (expresie) {
    // instructiuni
}
```

Exemplu:

```
int i=0; // se declara & initializeaza i cu 0
while (i<13) { /* intai se verifica conditia de
executie; daca conditia de executie a fost
adevarata se executa corpul buclei. i merge
pana la 12 inclusiv */
    i+=3;
}
```

Bucla do/while

```
do {
    //instructiuni
} while (expresie);
```

Exemplu:

```
int i=1; // declararea&initializarea lui i cu 1
do { /* corpul buclei se executa deja, pana sa
se ajunga la testarea conditiei */
    i*=5;
} while (i<126); // i merge pana la 125 inclusiv
```

Succesiunea pasilor la executarea unei structuri while este urmatoarea:

- este evaluata expresia, al carei tip de date trebuie sa fie boolean. In cazul in care valoarea aceasteia este false de la bun inceput, corpul buclei while nu se va executa deloc. In cazul in care valoarea este true, se executa setul de instructiuni ce formeaza corpul lui while
- se revine la inceputul buclei, repetand aceeasi succesiune de doi pasi (evaluare expresie → executie) pana cand valoarea expresiei devine false, ceea ce va cauza incheierea buclei while si continuarea executiei cu instructiunile aflate dedesubtul acesteia

In exemplul de mai sus, la prima evaluare a expresiei variabila i are valoarea 0 si deci expresia se evalueaza ca true, avand ca efect cresterea lui i cu 3 unitati; la a doua evaluare i este 3, la a treia 6 s.a.m.d. Ultima evaluare are loc cand i este 15, ceea ce produce valoarea *false* a expresiei si deci iesirea din bucla. **Atentie! Chiar daca conditia de intrare in bucla era *i<13*, la iesirea din bucla el are valoarea 15!**

Constructia *do...while* functioneaza dupa aceeasi logica, numai ca verificarea valorii expresiei se efectueaza abia dupa executia corpului buclei.

*Nota: diferenta dintre cele doua tipuri de bucle este momentul in care se evalueaza expresia. In cazul lui **while**, verificarea se face inainte de executia oricarei instructiuni (si deci este posibil sa nu fie executate deloc instructiunile dintre acolade), pe cand in cazul **do/while** are loc cel putin o executie a setului de instructiuni, prima evaluare a expresiei avand loc abia dupa aceea.*

2.1.3.2. Instructiunea for

Instructiunea for poate fi privita ca o varianta de while cu cateva elemente ajutatoare. In cazul in care programatorul trebuie sa parcurga elementele unui tablou (vezi capitolul despre tablouri), el apeleaza la o variabila ajutatoare - asa-numita variabila contor - care trebuie in primul rand initializata si apoi incrementata cu ocazia fiecarei iterari. Desi toate aceste operatii pot fi realizate si cu ajutorul lui while, bucla *for* le automatizeaza prin prezenta unor elemente de sintaxa dedicate lor.

Sintaxa generala a instructiunii for este:

```
for (septiuneal; septiunea2; septiunea3) {
    // instructiuni - corpul buclei for
}
```

Cele trei sectiuni ce compun bucla for au urmatoarele semnificatii si caracteristici:

- prima sectiune se executa o singura data - cand executia programului atinge bucla for. Ea este folosita in general pentru initializari de variabile contor sau alte variabile utilizate in cadrul buclei
- cea de-a doua sectiune este o expresie care trebuie sa aiba tip de date boolean si care se evalueaza inaintea fiecarei executii a corpului buclei. Ea are rol identic cu expresia din while: conditioneaza continuarea repetitiilor
- sectiunea a treia contine instructiuni executate la finalul fiecarei executii a corpului buclei. Este locul in care de obicei sunt plasate instructiunile de modificare a valorii variabilelor contor

```
for (initializare variabile contor; expresie; modificare variabile contor) {
    instructiune 1;
    instructiune 2;
    ...
}
```

Exemplu:

```
for (int i=0; i< 13; i++) {
    System.out.println(i);
}
```

Mai intai, se executa instructiunile din zona de initializare: variabila i este declarata si initializata cu 0. Urmeaza verificarea conditiei (care se face intotdeauna la inceput de ciclu), apoi, daca conditia se evalueaza true, este procesat corpul buclei si se afiseaza i. La finalul fiecarui ciclu sunt executate instructiunile din sectiunea "modificare variabile contor" - in cazul nostru, se incrementeaza i. Valorile afisate vor fi cele de la 0 la 12.

Atentie! O variabila declarata in interiorul unei bucle (cum este cazul variabilei i din exemplul anterior) este o *variabila locala* si va putea fi accesata numai in corpul buclei, nu si in afara acesteia! Se poate insa folosi pe post de contor si o variabila declarata anterior.

Prima si cea de-a treia sectiune a lui *for* pot contine mai multe instructiuni, separate prin virgula:

```
for(int i=0,j=4; i<5 && j<7; i++,j+=2){...}
```

Exista si o forma mai noua a buclei for (asa-numitul for...each), introdusa odata cu Java 1.5, care automatizeaza suplimentar operatia de parcursare a unui array sau a unei colectii:

```
int[] valori = {1,3,5,7,9}; // un tablou cu numerele impare pozitive mai mici ca 10
for(int nr : valori) {
    System.out.println(nr);
}
```

Variabila *nr* va lua pe rand ca valoare fiecare dintre valorile componente ale tabloului. Pentru fiecare valoare primita de *nr* se va efectua o executie a corpului buclei.

Diferentele intre aceasta forma a lui for si cea clasica sunt urmatoarele:

- avantaje:
 - noua forma este foarte simplu de folosit pentru programator, deoarece nu mai este nevoie de infrastructura suplimentara pe care acesta trebuia sa o defineasca in forma clasica (variabila contor in cazul tablourilor, iteratori in cazul colectiilor)
 - in cazul unora dintre colectii nici nu ar putea fi folosit for-ul clasic, deoarece acestea nu mentin elementele in ordine si deci nu au conceptul de pozitie a unui element, facand variabila contor inutila
- dezavantaj: neexistand variabila contor, noua forma nu permite accesul la pozitia curenta din cadrul tabloului sau colectiei

2.1.3.3. Modificarea iterarii: break si continue

Este posibila alterarea “curgerii” normale a unei bucle, fie prin incheierea ei prematura (instructiunea **break**), fie prin saltul la finalul iterarii curente (**continue**).

```
for(int i=0; i<5; i++) {
    System.out.println(i);
    if ( i == 1 ) continue;
    if ( i == 2 ) break;
    System.out.println(i);
}
```

Executia desfasurata a acestui exemplu este urmatoarea:

- la prima iterare, i are valoarea 0, ambele conditii din if-uri se evalueaza ca false si in consecinta i este afisat de doua ori si apoi incrementat
- la urmatoarea iterare, i are valoarea 1 si deci va fi afisat o singura data, dupa care este executata instructiunea continue. Efectul este saltul peste al doilea if si a doua afisare direct la finalul corpului lui for, dupa care i este incrementat
- la ultima iterare, i are valoarea 2 si deci va fi afisat o singura data, dupa care este executata instructiunea break ce cauzeaza iesirea din intreaga bucla. Nu mai conteaza ca sectiunea a doua din for ar fi permis un i egal cu 3, 4 sau 5, deoarece ea nu mai este evaluata

O bucla for, while sau do-while poate avea atasata o eticheta, utilizabila de catre instructiunile break si continue pentru a specifica, in cazul unor bucle imbricate, care bucla este cea care trebuie intrerupta sau continuata. Atunci cand eticheta lipseste, efectul se aplică bucliei celei mai din interior (cea care contine direct instructiunea break sau continue).

```
forulExterior: for(int i=0;i<4;i++) {
    for(int j=1;j<4;j++) {
        if(j==2 && i==1) continue forulExterior; //executia sare la sfarsitul primului for
        System.out.println("i="+i+", j="+j);
    }
}
```

2.2. Tablouri (arrays)

2.2.1. Concepte

Un tablou reprezinta o modalitate convenabila de stoca intr-o zona contigua de memorie (si de a accesa apoi algoritmice) o secventa de variabile **de acelasi tip**. Spre exemplu, un sir de numere intregi; in loc sa declarăm cate o variabila pentru fiecare numar in parte (int n1,n2,n3,...) aceste numere pot fi grupate sub o denumire comună, **numele variabilei de tip tablou**, iar accesarea lor se poate face prin specificarea pozitiei in cadrul tabloului (“numarul de pe pozitia 3”). Astfel, avem de-a face cu un sir de variabile dar ale caror nume nu mai sunt independente, ci pot fi automatizate folosind instructiuni repetitive.

In Java, tabloul este un obiect, facand parte dintre cele 4 tipuri de date recunoscute de masina virtuala (celelalte 3 fiind clasele, interfetele si primitivele). Iata cateva dintre implicatii:

- la declararea unei variabile de tip tablou, se declară de fapt o referinta catre un viitor tablou (vezi mai jos)
- tabloul este creat dinamic, alocand memorie cu ajutorul operatorului **new**
- un tablou are proprietatea **length**; odata creat tabloul, numarul sau de elemente poate fi obtinut cu **nume_tablou.length** si nu poate fi modificat
- dimensiunea unui tablou **nu** poate fi specificata la declarare cu sintaxa din C (**int x[3];**), deoarece crearea obiectului tablou se face prin intermediul operatorului new. Numele tabloului corespunde unei referinte catre un vector de date de tipul specificat, care inca nu a fost creat, si a carui dimensiune va fi determinata in momentul crearii dinamice a datelor componente

Există două tipuri de tablouri:

- tablouri unidimensionale – un astfel de tabel reprezintă o singură insiruire (lista) de date de același tip
- tablouri multidimensionale – sunt tablouri în care elementele sunt la randul lor tablouri

Indiferent de tipul de tabel, lucrul cu tablouri presupune stăpanirea a 4 operații:

- declararea variabilei de tip referință către tabel
- crearea obiectului tabel
- popularea tabeloului prin accesarea locațiilor componente
- parcurgerea tabeloului

2.2.2. Tablouri unidimensionale

2.2.2.1. Declararea unei variabile de tip tabel unidimensional

Declararea unei variabile de tip tabel unidimensional presupune specificarea tipului de date al informațiilor ce vor fi stocate în tabel, urmat de simbolul “[]” ca în exemplul de mai jos:

```
int[] x;
```

Variabila x este de tip referință la tabelul de int; zona de memorie referită de numele x conține o modalitate de a accesa un obiect de tip tabel (și niciodată valoarea primului element!).

Atenție! Instrucțiunea de mai sus nu face decât să declare o variabilă de tip referință – și atât! Nu este creat tabelul și nu este alocată memorie pentru elementele tabeloului. Orice încercare de a accesa unul din membrii tabeloului în acest moment va duce la o eroare de compilare sau runtime (compilatorul și mașina virtuală Java se străduiesc să nu permită folosirea variabilelor neinitializate).

2.2.2.2. Crearea obiectului tabel

Odată variabila tabel x declarată, pasul următor este crearea obiectului tabel în memorie, cu ajutorul operatorului new. Acest operator aloca memorie și produce ca rezultat referința către zona de memorie respectivă; este suficient să memorăm rezultatul lui new în x.

La crearea unui tabel se stabilește – direct sau indirect – numărul de elemente; fiecare element va fi initializat cu valoarea implicită pentru tipul de date respectiv.

Crearea unui obiect tabel se poate realiza în două moduri:

- *cu ajutorul operatorului new (ca pentru orice obiect Java):*

```
x = new int[4];
// este permisă declararea și initializarea în cadrul aceleiași instrucțiuni:
int[] s=new int[13];
```

In ultima instrucțiune, 13 reprezintă numărul de elemente al tabeloului. Operatorul new va aloca o zonă de memorie pentru 13 primitive int și va întoarce o referință către această, care prin intermediul operatorului de atribuire va fi stocată în s. Dimensiunea acestui tabel nu mai poate fi acum schimbată! Cel mult putem “convinge” variabila s să pointeze către un alt obiect tabel din memorie, atribuindu-i o alta valoare.

- *prin initializarea explicită a variabilei tabel, specificând o lista cu valoarea fiecarui element:*

```
int[] s={1,2,3,4};
char[] c={'I','n','f','o','a','c','a','d','e','m','y'}
```

Exista si posibilitatea crearii unui asa-numit tablou anonim, atunci cand nu este necesara o variabila de tip referinta catre acesta (de exemplu, la pasarea unui tablou ca parametru intr-o metoda):

```
alege( new int[] {1,2,3}); // apelarea functiei/metodei alege() cu parametri de tip tablou
```

Atentie! Odata creat un obiect tablou, dimensiunea acestuia nu mai poate fi modificata! Daca este nevoie de elemente suplimentare, singura solutie este crearea unui nou tablou mai cuprinzator si copierea valorilor din cel vechi!

2.2.2.3. Accesarea membrilor tabloului

Un tablou reprezinta un sir de variabile care nu au nume individual – ele vor fi denumite in functie de pozitia lor in cadrul tabloului (asa-numitul **index**). Fiecare membru al unui tablou se comporta ca o variabila obisnuita, care poate fi citita sau modificata (prin atribuire, incrementare/decrementare etc.).

Accesarea unei anumite variabile componente se face folosind numele tabloului (al variabilei de tip referinta) si indexul (pozitia in tablou) cuprins intre “[“ si “]”. Indexul incepe de la 0:

```
int[] s=new int[14];
s[0]=145; //memoram valoarea 145 pe prima pozitie din tablou
```

In acest caz avem de-a face cu un sir de intregi creat in momentul declararii; primul index valid este 0 iar ultimul 13.

Atentie! Indexul este de tip int; daca un tablou are N elemente, valorile posibile ale indexului sunt cuprinse intre 0 si N-1. Incercarea de a folosi un index invalid se va solda cu o eroare de tip ArrayIndexOutOfBoundsException.

2.2.2.4. Popularea tabloului

Popularea tabloului presupune memorarea de valori in variabilele sale componente. Un tablou poate fi populat:

- la creare, folosind sintaxa prezentata mai sus (`int [] s={1,2};`)
- ulterior - in momentul alocarii memoriei pentru obiectul tablou, elementele sale sunt initializate automat cu valoarea din oficiu pentru acel tip de primitiva. Putem da elementelor tabloului valorile dorite folosind operatorul de atribuire:

```
s[0]=3; s[1]=56; // etc
```

2.2.2.5. Parcurgerea tabloului

Pentru parcurgerea unui tablou se apeleaza in cele mai dese cazuri la una dintre formele buclei for. Spre exemplu, pentru a afisa informatiile dintr-un tablou de tip `int` numit *numere* am putea scrie:

```
// varianta clasica
for(int i=0; i<numere.length; i++){
    System.out.println(numere[i]);
}

// forma noua a lui for
for(int n:numere){
    System.out.println(n);
}
```

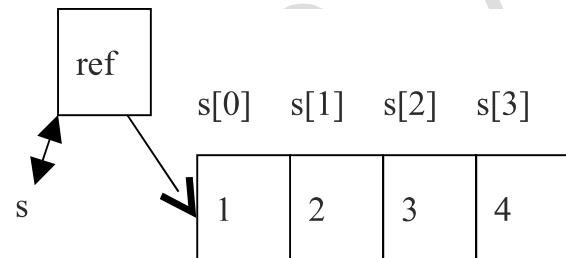


Figura 1

2.2.3. Tablouri multidimensionale

2.2.3.1. Concepte

Orice tip de date din Java poate fi folosit pentru a crea un tablou. Putem avea tablouri de numere, de caractere, de obiecte sau de...alte tablouri! Atunci cand elementele unui tablou sunt la randul lor tablouri avem de-a face cu un tablou multidimensional. In functie de nivelul pana la care se realizeaza aceasta ramificare, avem:

- tablouri unidimensionale - elementele sunt scalari
- bidimensionale - elementele tabloului de baza sunt tablouri unidimensionale, ale caror elemente sunt scalari
- tridimensionale - elementele tabloului de baza sunt tablouri bidimensionale; fiecare dintre acestea contine elemente de tip tablou unidimensional, care la randul lor stocheaza valori scalare
- s.a.m.d

Atentie! Un tablou bidimensional nu este o matrice! Toate liniile unei matrici contin acelasi numar de elemente, pe cand in cazul unui astfel de tablou sunt posibile **lungimi diferite ale tablourilor componente**. (vezi figura de mai jos)

2.2.3.2. Declararea variabilei de tip tablou

Declararea unei variabile de tip tablou multidimensional se realizeaza folosind alaturi de tipul de date un numar de paranteze [] egale cu dimensiunea tabloului:

```
int[][] s;
```

Dublul simbol [][] indica faptul ca avem de-a face cu o referinta la un tablou bidimensional - fiecare din componentele tabloului de baza este la randul sau o referinta catre un tablou de intregi.

2.2.3.3. Crearea obiectului de tip tablou

Ca si in cazul unidimensional, crearea si initializarea unui tablou multidimensional se poate face in doua moduri:

1. specificand explicit valorile elementelor in momentul declararii. Exemplu:

```
int[][] s={{13,22,11,6},{34,105,21}, {131,23,84,263,65},{1,2,3,4}};
```

Prin acest procedeu sunt create in memorie 5 tablouri:

- tabloul de baza (dimensiunea primara), care contine referinte catre viitoarele 4 tablouri secundare
- cele patru tablouri de intregi {13,22,11,6}, {34,105,21}, {131,23,84,263,65} si {1,2,3,4}

Tabloul de baza, accesibil prin intermediul referintei s, va fi populat astfel: primul sau element este referinta catre tabloul {13,22,11,6}, cel de-al doilea este referinta catre {34,105,21} s.a.m.d.

2. folosind operatorul new. In acest caz trebuie creat fiecare tablou in parte, dar in ordinea corecta (intai dimensiunea primara, apoi tablourile din dimensiunea secundara s.a.m.d.); oricarui tablou proaspat creat trebuie sa-i poata fi memorata referinta intr-o variabila deja existenta din dimensiunea imediat superioara

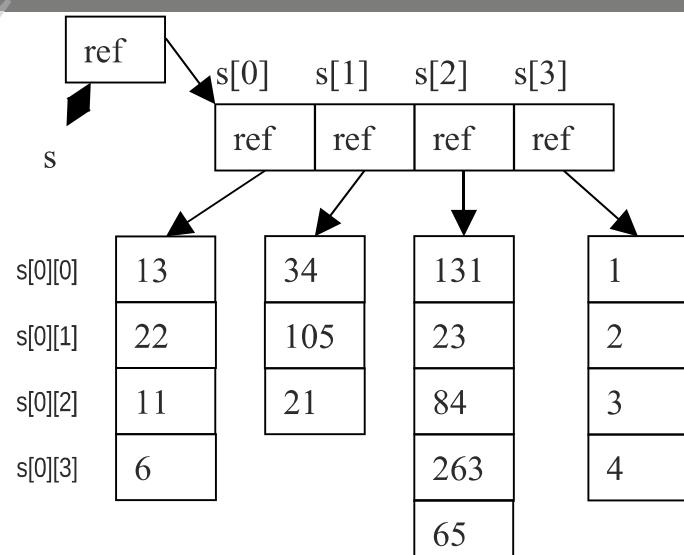


Figura 2

```
int[][] s = new int[4][]; // crearea tabloului de baza (cel primar)
s[0] = new int[4]; // crearea primului tablou secundar
s[1] = new int[3]; // crearea celui de-al doilea tablou secundar
s[2] = new int[5]; // crearea celui de-al treilea tablou secundar
s[3] = new int[4]; // crearea celui de-al patrulea tablou secundar
```

A se remarcă setul gol de paranteze de pe prima linie, care lasă tablourilor secundare posibilitatea de a avea lungimi diferite. Există și o sintaxă care creează o matrice:

```
// creeaza un tablou de referinte la tablou unidimensional de int;
// toate elementele sunt initializate cu null și nu sunt create tablouri secundare
int[][] s = new int[4][];

// creeaza un tablou de referinte la tablou unidimensional de int, plus toate tablourile secundare
// toate tablourile secundare vor avea aceeași lungime
int[][] s = new int[4][3];
```

2.2.3.4. Accesarea elementelor și popularea tabloului

Pentru a accesa un element scalar (o valoare) dintr-un tablou multidimensional este necesar un număr de indici egal cu dimensiunea tabloului. Spre exemplu, în cazul unui tablou bidimensional, pentru a accesa o anumita valoare dintr-un tablou secundar este necesar să stim atât despre ce tablou e vorba (asadar indexul în tabloul de bază) cât și poziția elementului în cadrul tabloului secundar.

Referirea la un element scalar din cadrul unui tablou multidimensional se realizează folosind numele tabloului urmat de numarul corespunzător de indecsi:

```
int[][] m = {{1,2,3},{4,5}};
System.out.println(m[1][0]); // afiseaza 4
```

Dacă dorim să referim un întreg tablou component al unuia multidimensional, trebuie să folosim un număr de indici egal cu adâncimea la care se află elementul dorit:

```
// considerand tabloul s din Figura 2, cream o copie a referinței către primul tablou secundar
int[] copie = s[0]; // s[0] are ca tip de date int[]

// pentru un tablou tridimensional t, pentru a accesa unul dintre tablourile unidimensionale este
// nevoie de doi indici, iar pentru cele bidimensionale de unul:
int[] a = t[1][3];
int[][] b = t[0];
```

2.2.3.5. Parcursarea tabloului

Parcursarea unui tablou multidimensional presupune accesarea fiecarui element din fiecare dimensiune. Spre exemplu, pentru un tablou bidimensional trebuie să avem acces la fiecare valoare din fiecare tablou secundar; în cazul unui tridimensional, la fiecare valoare din fiecare tablou ternar din fiecare tablou bidimensional din tabloul primar etc. În practică, orice cuvânt “fiecare” din descrierile de mai sus se materializează în cale o buclă for, rezultând astfel o structură de bucle imbricate.

Exemplu - incrementarea valorilor din tabloul bidimensional de întregi s din Figura 2:

```
// pentru fiecare element al tabloului de bază, care este un tablou unidimensional de int...
for(int a=0; a<s.length; a++) {
    // ...accesăm fiecare element al tabloului secundar curent
    for(int b=0; b<s[a].length; b++) {
        s[a][b]++;
    }
}
```

2.2.4. Tablourile si operatorul de atribuire

2.2.4.1. Tablourile nu se pot copia prin simpla atribuire

Datorita faptului ca tabloul este un obiect si variabila care ii da numele este o referinta, simpla atribuire $s1=s2$, unde $s1$ si $s2$ sunt variabile de tip tablou, nu va avea ca efect copierea informatiei dintr-un tablou in celalalt, ci crearea a doua referinte catre acelasi obiect tablou.

In programul urmator ne vom folosi de faptul ca, daca seincearca afisarea unei variabile de tip tablou, va fi afisat tipul elementelor si hash code-ul obiectului:

```
public class Tablouri {
    public static void main(String[] args) {
        int[] s1={1,2}, s2=new int[]{1,2}, temp;
        // s1 si s2 sunt identice ca si elemente, insa sunt tablouri diferite
        System.out.println("s1 : "+s1);
        System.out.println("s2 : "+s2);
        temp=s2;           //aici nu se face o copie a elementelor lui s2, ci o duplicare a
                           // referintei s2 in temp; avem acum doua referinte catre acelasi tablou
        System.out.println("temp : "+temp);
        /* temp si s2 se refera acum la acelasi tablou; modificand un element al lui s2 se va
         * modifica si elementul corespunzator al lui temp */
        s2[0]=5;
        System.out.println("s1[0] : "+s1[0]);
        System.out.println("s2[0] : "+s2[0]);
        System.out.println("temp[0] : "+temp[0]);
    }
}
```

O solutie pentru copierea de tablouri o reprezinta:

```
System.arraycopy(tablouSursa, pozitieSursa,tablouDestinatie, pozitieDestinatie, lungime);
```

Atentie insa! In cazul tablourilor multidimensionale, `System.arraycopy()` duplica numai prima dimensiune, nu si tablourile secundare!

2.2.4.2. Operatii de atribuire corecte

Cand vine vorba de operatii de atribuire inter-tablouri, trebuie avut intotdeauna grija ca numarul de dimensiuni ale taboului din dreapta sa fie egal cu al tabloului stang. Acest lucru nu este intotdeanua evident; fie urmatoarele variabile tablou:

```
int[] a;
int[][] b;
int[][][] c;
```

Presupunand ca variabilele au fost initializate (s-au creat tablourile catre care pointeaza) si ca toti indecsii folositi sunt valizi, iata cateva exemple de atribuiriri si rezultatul acestora:

```
b = a;           // nu functioneaza; in stanga avem tablou bidimensional iar in dreapta unidimensional
b = c;           // nu functioneaza; in stanga avem tablou bidimensional iar in dreapta tridimensional
b[0] = a;         // ok; in ambele parti avem tablou unidimensional
c[0] = b;         // ok; in ambele parti avem tablou bidimensional
c[0][1] = a;      // ok; in ambele parti avem tablou unidimensional
b = c[2];         // ok; in ambele parti avem tablou bidimensional
c[1][2] = b;      // nu functioneaza; in stanga avem tablou unidimensional iar in dreapta bidimensional
c[2][0][1] = a[0]; // ok, in ambele parti avem scalar (int)
```

Nota: o astfel de atribuire invalida genereaza eroare de compilare.

2.2.5. Clasa Arrays

Clasa mentionata pune la dispozitia programatorului modalitati pentru:

- ordonarea unui tablou – folosind **Arrays.sort()**. Exista variante ale metodei sort() pentru toate tipurile de primitive si pentru obiecte. Poate fi ordonat intregul tablou sau portiuni ale acestuia:
 - **sort(tablou)** - ordoneaza intregul tablou
 - **sort(tablou, pozitieStart, pozitieStop)** - ordoneaza numai elementele aflate intre indecesii *pozitieStart* (inclusiv acesta) si *pozitieStop*, dar **fara pozitieStop!**
- cautare binara in tablou – folosind **Arrays.binarySearch()**. Este necesar ca elementele sa fi fost ordonate in prealabil! Returneaza indexul pe care a fost gasita valoarea, sau o valoare negativa in caz contrar. Forme:
 - **binarySearch(tablou, valoare)** - cauta valoarea specificata in intregul tablou
 - **binarySearch(tablou, indexStart, indexStop, valoare)** - cauta valoarea numai intre indecesii specificati (inclusiv primul, exclusiv ultimul)
- producerea unei copii (posibil partiale) a unui tablou – folosind **Arrays.copyOf()** sau **Arrays.copyOfRange()**:
 - **copyOf(tablouVechi, lungimeNoua)** - creeaza si returneaza un nou tablou de lungimea dorita in care copiaza elementele corespunzatoare din vechiul tablou. Daca noua lungime este inferioara celei vechi, se realizeaza trunchierea taboului original; daca noua lungime este mai mare, vor fi copiate toate elementele vechiului tablou si restul de pozitii se vor initializa cu valoarea default a tipului de date in cauza (util atunci cand un tablou s-a umplut si avem nevoie sa cream unul nou, mai lung)
 - **copyOfRange(tablouVechi, indexStart, indexStop)** - creeaza si returneaza un tablou ce contine un interval de elemente din tabloul original. Sunt copiate elementele incepand cu indexStart insa omitand indexStop!
- verificarea egalitatii intre tablouri – **Arrays.equals()** sau **Arrays.deepEquals()**:
 - **equals(tablou1, tablou2)** - verifica daca cele doua tablouri sunt egale (au aceleasi elemente si in aceeasi ordine)
 - **deepEquals(tablou1, tablou2)** - aceeasi verificare dar efectuata recursiv, pentru tablouri multidimensionale
- umplerea unui tablou sau a unei portiuni de tablou cu o anumita valoare – **Arrays.fill()**:
 - **fill(tablou, valoare)** - atribuie fiecarui element al tabloului valoarea specificata
 - **fill(tablou, indexStart, indexStop, valoare)** - atribuie elementelor aflate intre indexStart si indexStop (inclusiv primul, exclusiv al doilea) valoarea specificata
- obtinerea unei reprezentari text pentru un tablou – **Arrays.toString()** sau **Array.deepToString()**:
 - **toString(tablou)** - returneaza un sir de caractere (String) cu o reprezentare textuala a elementelor tabloului (ex: [1,2,3,4])
 - **deepToString(tablou)** - acelasi rol, dar functioneaza in mod recursiv. Util pentru tablouri multidimensionale
- returnarea continutului tabloului sub forma de colectie – **Arrays.asList(tablou)**. Returneaza continutul tabloului sub forma de colectie List (vezi capitolul dedicat colectiilor Java)

Nota: explicarea acestor tuturor acestor functii/metode nu este posibila in totalitate folosind cunostintele acumulate pana in acest moment; clarificarea deplina va veni odata cu intelegerea conceptelor de clasa, obiect si metoda statica. Pe de alta parte, acest lucru nu ne opreste sa le privim ca pe niste simple functii procedurale, apelate - este drept - cu o sintaxa mai aparte: **Arrays.numeFunctie()** (pasandu-le parametri acolo unde este nevoie).

Atentie! Pentru a putea utiliza clasa Arrays este necesar ca la inceputul fisierului in care ea este utilizata sa existe urmatoarea declaratie:

```
import java.util.Arrays;
```

Justificarea va fi oferita in cadrul lectiei despre pachete si localizarea resurselor.

Exemple:

```
int[] nr = {5,2,4,3,1};
System.out.println(Arrays.toString(nr));           // [5,2,4,3,1]
```

```
Arrays.sort(nr);
System.out.println( Arrays.toString(nr));           // [1,2,3,4,5]
System.out.println( Arrays.binarySearch(nr,4));      // 3
Arrays.fill(nr,2);
System.out.println( Arrays.toString(nr));           // [2,2,2,2,2]
Arrays.fill(nr,1,3,0);
System.out.println( Arrays.toString(nr));           // [2,0,0,2,2]
```

2.3. BIBLIOGRAFIE

- Java Tutorial:
 - Structuri de control al executiei: <http://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html>
 - Tablouri: <http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>
- Clasa Arrays: <http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>