TEHNICI AVANSATE DE PROGRAMARE

LUCRARE DE LABORATOR 1

Prezentarea si operarea cu SDK-urile Sun Microsystems, Elemente de baza ale limbajului Java

I. SCOPUL LUCRĂRII

Lucrarea de față are rolul de a prezenta şi familiariza studentul cu noțiuni şi elemente de bază ale limbajului Java cum ar fi: Setul de caractere, Cuvinte cheie, Cuvinte rezervate, Identificatori, Literali, Separatori, Comentarii, Operatori, Variabile, Expresii, Vectori.

La sfârşitul acestei lucrări, studentul va avea posibilitatea să scrie programe Java în care să utilizeze elementele limbajului menționate mai sus.

II. NOȚIUNI TEORETICE

Prezentarea si operarea cu SDK-urile Sun Microsystems

Observație: Vezi capitolul 1 din curs.

Primul program Java

```
import java.io.*;
class PrimulProgram{
    public static void main(String[ ] arg) {
        System.out.print("Primul program Java");
        System.out.println(". . . dar nu si ultimul!");
    }
}
```

Din secventa de mai sus se poate constata, la o prima privire, ca cel putin elementele de delimitare a propozitiilor sunt foarte asemanatoare cu cele din C: este vorba despre acolade si caracterele **;** de la sfarsitul instructiunilor.
Programul prezentat nu face altceva decat sa afiseze o secventa de caractere pe ecran, dar poate servi la evidentierea unor elemente de baza care apar in orice program Java.
In primul rand trebuie spus ca ***programele Java sunt constituite ca seturi de clase***.

- **NU EXISTA** ceea ce in programele C numim ***date globale***.
- De asemenea, **NU EXISTA** functii de sine statatoare: avem **DOAR** functii (metode) incluse in clase.

Pentru descrierea unei clase se foloseste o constructie sintactica de forma:

class *nume_clasa* ***{***

//continut clasa
}

Apoape orice program, indiferent ca este scris intr-un limbaj procedural sau obiectual, are o **radacina** sau un punct de plecare. Astfel, in programele Pascal avem ceea ce se numeste ***program principal***, iar in C avem functia ***main***.

In programele Java vom avea o **clasa principală (radacină)** care se caracterizeaza prin faptul ca include o functie al carei prototip este:

public static void main(String[ ] arg)

Elementele desemnate prin cuvintele ***public*** si ***static*** vor fi lamurite putin mai tarziu. Deocamdata le folosim asa cum le vedem.
Numele clasei radacina este un identificator dat de programator.
Parametrul functiei ***main*** este de tip ***tablou de elemente String*** (siruri de caractere). Prin intermediul acestui parametru putem referi si utiliza in program eventualele argumente specificate la lansarea in executie a programului.
In ceea ce priveste clauza ***import*** plasata chiar la inceput, ea se refera la faptul ca programul utilizeaza operatii ale unor clase aflate in pachetul numit ***java.io***.
Actiunile executate de programul de mai sus presupun două operatii de afisare: ***print*** si ***println***. Observam modul putin mai ciudat de apel al acestor operatii. Prefixul ***System.out*** care insoteste numele celor două operatii reprezinta numele unui obiect: este vorba despre un obiect predefinit care se utilizeaza atunci cand destinatia unei operatii de scriere este ecranul monitorului. Asa dupa cum vom vedea si in lucrarile urmatoare, deoarece metodele sunt incluse in clase si, majoritatea, si in instantele claselor, apelul presupune precizarea numelui clasei (daca e vorba de metode statice) sau al obiectului "posesor". Cu alte cuvinte, in Java, apelul unei operatii se realizeaza folosind una din notatiile:

nume_obiect.nume_metoda(parametri_actuali)
sau:
nume_clasa.nume_metoda(parametri_actuali)

Pe parcurs vom clarifica diferentele dintre cele două tipuri de notatii.
Metodele ***print/println*** pot primi ca parametru un sir de caractere dat fie sub forma de constanta, asa ca in exemplul nostru, fie sub forma de expresii al caror rezultat este de tip ***String***. Metodele ***print/println*** mai pot primi ca parametru si valori ale tipurilor primitive sau referinte de clase, dar acestea sunt convertite tot la tipul ***String*** inainte de afisare. Diferenta dintre ***print*** si ***println*** este aceea ca ***println*** realizeaza, dupa afisarea textului dat ca parametru, un salt la linie noua. Se precizeaza ca ***println*** poate fi apelata si fara parametri, caz in care executa doar un salt la linie noua:

System.out.println( );

Am afirmat ca parametrul metodei ***main*** din clasa radacina serveste la accesarea eventualelor argumente date in linia de comanda la lansarea in executie a programului. Pentru a ilustra acest aspect, vom considera un program care afiseaza o formula de salut, urmata de numele si prenumele utilizatorului, date ca argumente:

```
import java.io.*;
class Salutare{
   public static void main(String[ ] arg) {
      System.out.println("Te salut, "+arg[0]+" "+arg[1]);
   }
}
```

Dupa ce compilam acest program, il putem lansa in executie de exemplu cu comanda:

java Salutare mai Popescule

Observatii:

- Programului i se dau 2 argumente (***mai*** si ***Popescule***). Acestea sunt interpretate ca primele 2 elemente ale tabloului ***arg*** ce figureaza ca parametru formal al metodei ***main*** (se poate spune ca ***arg*** este un omolog al lui ***_argv*** din programele C). La fel ca si in C, in Java elementele unui tablou se indexeaza incepand cu 0.

- Daca programului nu i se furnizeaza cele 2 argumente, rezultatul este oprirea executiei cu un mesaj prin care se reclama depasirea indicelui de tablou (***IndexOutOfBoundsException***). Este vorba de tabloul ***arg*** care are dimensiunea 0 in cazul lipsei argumentelor.
  De aceea, daca dorim sa fim mai rigurosi, e bine sa prevedem secvente de verificare in program, prin care sa testam daca utilizatorul a furnizat numarul necesar de parametri:

```
import java.io.*;
    class Salutare{
       public static void main(String[ ] arg) {
          if(arg.length<2)
             System.out.println("Nu ati dat parametrii necesari!!");
          else
             System.out.println("Te salut, "+arg[0]+" "+arg[1]);
       }
    }
```

- Expresia ***arg.length*** utilizata in secventa de mai sus ne da numarul de elemente ale tabloului ***arg***. Posibilitatea de a folosi aceasta expresie explica de ce metoda ***main*** nu are nevoie de un al 2-lea parametru, omolog al lui ***_argc*** din programele C.

- O alta observatie care trebuie facuta in contextul acestui exemplu priveste parametrul functiei ***println***. De data acesta am folosit o expresie construita prin aplicarea operatorului + asupra unor operanzi de tip ***String***, al carei rezultat este un sir de caractere obtinut prin concatenarea operanzilor.
  Ceea ce trebuie sa retinem de aici este faptul ca, spre deosebire de functiile ***printf*** din C, care acceptau un numar variabil de parametri, functiile ***print/println*** din Java accepta **UN SINGUR** parametru. Ca urmare, atunci cand dorim sa scriem printr-o singura instructiune mai multe valori trebuie sa le concatenam, spre a forma un singur sir.

Elemente de bază ale limbajului Java

Observație: Vezi capitolul 1 din curs.

1. Variabile automatice şi inițializarea lor

O variabilă automatică este o variabilă locală unei metode (este creată la intrarea în metodă, există numai pe durată execuției metodei).

Variabilele locale nu sunt inițializate de către sistem şi trebuie inițializate explicit înainte de a fi utilizate.

```
class VarLoc{
```

```
  public static void main(String args[]){
        int i;
        i=i+5;
        System.out.println(i);
 }
}
```

În exemplul de mai sus, compilatorul semnalează o eroare la linia 4 *"Variable i might not have been initialized"*.

2. Operatori în limbajul Java

Operatorul modulo %

```
class TestModulo{
  public static void main(String args[]){
        int i=33,j=6;
        System.out.println(i%j);
        int a=-29,b=4;
        System.out.println(a%b);
        float f1=10.7f,f2=4.5f;
        System.out.println(f1%f2);
        double d1=-8.7d,d2=-4.5d;
        System.out.println(d1%d2);
 }
}
```

Exercițiu: Să se verifice printr-un calcul pe hârtie, valorile afişate de program.

Operatorii de deplasare <<, >>, >>>

Operanzii trebuie să fie de un tip întreg, în general **int** sau **long**.

Operandul din partea dreaptă este redus **modulo x**, unde x depinde de tipul rezultatului operației. Tipul este fie **int**, fie **long**, operatorii de tip mai mic fiind supuşi promovării aritmetice. Dacă operatorul din partea stângă este de tip **int** atunci x este 32, iar dacă acesta este de tip **long** atunci x este 64.

Operatorul << deplasează spre stânga şi biți de 0 sunt introduşi pe pozițiile cele mai puțin semnificative. Deplasarea spre stânga cu o poziție corespunde dublării operandul stâng, deplasarea spre stânga cu mai mult de o poziție corespunde înmulțirii operandului stâng cu 2,4,8,16, ş.a.m.d.

Operatorul >> efectuează o deplasare cu semn (sau aritmetică) spre dreapta. Rezultatul are biți de 0 pe cele mai semnificative poziții dacă operandul stâng este pozitiv, şi biți de 1 pe cele mai semnificative poziții dacă operandul stâng este negativ. Rezultatul aproximează împărțirea operandului stâng la 2 ridicat la puterea operandului drept.

Operatorul >>> efectuează o deplasare fără semn (sau logică) spre dreapta. Rezultatul are biți de 0 pe cele mai semnificative poziții şi poate să nu reprezinte întotdeauna o divizare a operandului stâng.
Exemplu:

```
class TestOpShift{
  public static void main(String args[]){
        int i=192;
        System.out.println(i<<2);
        System.out.println(i>>2);
        System.out.println(i<<33);
        long j=-192;
        System.out.println(j<<1);
        System.out.println(j>>2);
        System.out.println(j>>66);
        System.out.println(64>>>4);
        System.out.println(-64>>>4);
```

```
}
}
```

Operatorul condițional ?:

Observație:

Într-o expresie de forma **a= x ? b : c**

- tipurile expresiilor **b** şi **c** trebuie să fie compatibile şi sunt făcute identice prin promovare aritmetică
- tipul expresiei **x** trebuie să fie boolean
- tipul expresiilor **b** şi **c** trebuie să fie compatibile la asignare cu tipul lui **a**
- valoarea asignată lui **a** va fi **b** dacă **x** este adevărat, sau va fi **c** dacă **x** este fals

Operatorii de asignare

Observații:

Operatorii de asignare setează valoarea unei variabile sau expresii la o nouă valoare. Asignarea simplă utilizează =. Operatori ca "+=" şi "*=" au o funcție compusă de calcul şi asignare. Aceşti operatori compuşi au forma generală "*op*=", unde *op* poate oricare dintre operatorii binari non-booleeni. În general, pentru orice expresii compatibile x şi y, expresia *x op=y* este o prescurtare pentru *x=x op y*.

Totuşi trebuie avute în vedere două aspecte. Primul se referă la faptul că expresia x este evaluată exact o dată în cazul primei scrieri, ci nu de 2 ori aşa cum se sugerează în cazul utilizării scrierii expandate. Al doilea aspect se referă la faptul că operatorii de asignare includ un "cast" (conversie) implicit. Să considerăm următoarele situații:

```
byte x=2;
x+=3;
```

Dacă s-ar fi scris codul anterior utilizând scrierea expandată:

```
byte x=2;
x=(byte)(x+3);
```

operația de cast spre tipul **byte** ar fi fost necesară, deoarece rezultatul adunării unui număr întreg este cel puțin **int**. În prima situție, operația de cast este implicită.

3. Ordinea evaluării operanzilor

Se face de la stânga la dreapta. Rezultatul oricărei expresii va fi ca şi cum toți operanzii sunt evaluați de la stânga la dreapta, chiar dacă ordinea executării operațiilor este câteodată diferită (şi din motive de optimizare a calculului). Exemplu:

```
int a[]={4,4};
int b=1;
a[b]=b=0;
```

Evaluarea operanzilor de la stânga la dreapta cere ca **a[b]** să fie evaluat întâi (deci **a[1]**). Apoi este evaluat **b** iar apoi expresia constantă 0. Acum că operanzii au fost evaluați, sunt efectuate operațiile (în ordinea precedenței şi asociativității). Pentru asignări, asociativitatea este de la dreapta la stânga, deci valoarea 0 este asignată întâi variabilei **b** iar apoi elementului din vector **a[1]** .

4. Conversia tipurilor primitive: promovarea aritmetică a operanzilor

Să considerăm următorul fragment de cod:

```
short s=9;
int i=10;
float f=11.1f;
double d=12.2;
if (-s*i > = f/d)
     System.out.println("">>>>");
else
     System.out.println("<<<<");
```

Conversiile de promovare aritmetică sunt toate conversii de la tipuri mici spre tipuri mari, şi sunt efectuate automat de către sistem. Regulile de promovare aritmetică disting între operatorii unari şi cei binari.

Pentru operatorii unari, se aplică 2 reguli, în funcție de tipul operandului:
- dacă operandul este de tip **byte, short**, sau **char**, atunci acesta este convertit la **int** (cu excepția operatorilor ++ , - - , când nu se aplică nici o conversie)
- altfel nu se efectuează nici o conversie

Pentru operatorii binari există 4 reguli, în funcție de tipurile celor doi operanzi:
- dacă unul dintre operanzi este de tip **double**, atunci celălalt operand este convertit la **double**
- altfel dacă unul dintre operanzi este de tip **float**, atunci celălalt operand este convertit la **float**
- altfel dacă unul dintre operanzi este de tip **long**, atunci celălalt operand este convertit la **long**
- altfel ambii operanzi suntconvertiți la **int**

Pentru exemplul anterior, iată cum lucrează sistemul:
1. Variabila **s** de tip **short** este promovată la un **int** si apoi negată
2. Rezultatul de la pasul 1 (un **int**) este înmulțit cu variabila **i** de tip **int**. În acest caz nu este necesară nici o conversie. Evident, rezultatul înmulțirii este un **int**.
3. Înainte de a împărți **f** de tip **float** la **d** de tip double, **f** este promovat la double. Rezultatul împărțirii este de tip **double**.
4. Rezultatul de la pasul 2 (un **int**) trebuie să fie comparat cu rezultatul de la pasul 3 (un **double**), şi este promovat la **double** apoi se efectuează compararea. Rezultatul unei comparări este întotdeauna de tip **boolean**.

1. Vectori în Java

Un vector în Java este o colecție omogenă şi ordonată de primitive, referințe ale obiectelor sau alți vectori. Pentru a crea şi utiliza un vector trebuie urmați trei paşi:
1. declararea
2. construirea
3. inițializarea

Declararea spune compilatorului care este numele vectorului şi de ce tip vor fi elementele sale. De exemplu:

```
int [] ints;
double dubs[];
float [][] floats;
```

Declararea nu specifică şi dimensiunea vectorului, aceasta fiind specificată la executarea programului, când vectorul este alocat prin intermediul cuvântului cheie **new**. Exemple:

```
int [] ints;
ints=new int[25];

 vectorului intr-o singura linie
// declararea si construireaint vec_int[]=new int[25];

// dimensiunea specificata ca o variabila, nu ca un literal
int size=12*23;
```

```
float [] floats;
floats=new float[size];
```

Atunci când un vector este construit, elementele sale sunt automat inițializate la valorile lor implicite (default). Vezi tabelul de mai jos:

Tipul elementului	Valoare inițială	Tipul elementului	Valoare inițială
Byte	0	Float	0.0f
Short	0	Double	0.0d
Int	0	Char	'\u0000'
Long	0L	Boolean	false

Se poate combina declararea, construirea şi inițializarea vectorului într-un singur pas.

```
float [] vec={1.1f,2.3f,7.9f,5.7f,9.2f};
```

Cel mai sigur mod de a ne referi la dimensiunea unui vector este de aplica **.length** numelui vectorului. Exemplu:

```
long vec[];
vec=new long[600];
for(int i=0;i<vec.length;i++)
       Vec[i]=i*i;
```

III. MODUL DE LUCRU

- Clasic:

1. Se editează codul sursă al programului Java folosind un editor de text disponibil (de ex., se poate utiliza Notepad).
2. Se salvează fişierul cu extensia **.java**.
3. Compilarea mini-aplicației Java se va face din linia de comandă:
   javac nume_fişier_sursă.java
   În cazul în care programul conține erori acestea vor fi semnalate şi afişate.
4. Pentru rularea aplicației Java, se lansează interpretorul Java:
   java nume_clasă_care_conține_main

- Se foloseşte utilitarul disponibil în laborator J2SDK Net Beans.

IV. TEMĂ

- Se vor parcurge toate exemplele prezentate în platforma de laborator testându-se practic (acolo unde este cazul).
- Să se răspundă la următoarele întrebări grilă, explicând şi alegerea rezultatului:

1. După executarea fragmentului de cod de mai jos, care sunt valorile variabilelor *x*, *a* şi *b*?

```
int x, a=6,b=7;
x=a++ + b++;
```

```
A. x=15, a=7, b=8
B. x=15, a=6, b=7
C. x=13, a=7, b=8
D. x=13, a=6, b=7
```

2. Care din următoarele expresii sunt legale? (Alegeți una sau mai multe.)

```
A. int x=6; x=!x;
B. int x=6; if (!(x>3)) { }
C. int x=6; x=~x;
```

3. Care dintre următoarele expresii rezultă într-o expresie pozitivă a lui x? (Alegeți una.)

```
A. int x=-1; x=x >>> 5;
B. int x=-1; x=x >>> 32;
C. byte x=-1; x=x >>> 5;
D. int x=-1; x=x >> 5;
```

4. Ce rezultă după rularea următorului cod?

```
class Xor{
   public static void main(String args[]){
       byte b=10; // 00001010 in binar
       byte c=15; // 00001111 in binary
       b=(byte) (b ^ c);
       System.out.println("b contine "+b);
  }
}
```

```
A. Output-ul este: b contine 10
B. Output-ul este: b contine 5
C. Output-ul este: b contine 250
D. Output-ul este: b contine 245
```

5. Ce rezultă după încercarea de a compila şi rula următorul cod?

```
1.class Conditional{
2.   public static void main(String args[]){
3.       int x=4;
4.       System.out.println("valoarea este "+((x > 4) ? 99.99:9));
5.  }
6.}
```

```
A. Output-ul este: valoarea este 99.99
B. Output-ul este: valoarea este 9
C. Output-ul este: valoarea este 9.0
D. Se semnalează eroare de compilare la linia 4.
```

6. Care este output-ul acestui fragment de cod?

```
int x=3; int y=10;
System.out.println(y % x);
```

```
A. 0
B. 1
C. 2
D. 3
```

7. Ce rezultă din următorul fragment de cod?

```
int x=1;
String []names={"Fred","Jim","Sheila"};
names[--x]+=".";
for(int i=0;i<names.length;i++)
    System.out.println(names[i]);
```

A. Output-ul include *Fred.*
B. Output-ul include *Jim.*
C. Output-ul include *Sheila.*
D. Nimic din cele de mai sus.

8. Care linie din următorul cod nu se va compila?

```
1. byte b=5;
2. char c='5';
3. short s=55;
4. int i=555;
5. float f=555.5f;
6. b=s;
7. i=c;
8. if(f>b)
9.   f=i;
```

9. Se va compila următorul cod ?

```
byte b=2;
byte b1=3;
b=b*b1;
```

10. În codul de mai jos, care sunt posibilele tipuri pentru variabila **result**? (Alegeți cel mai complet răspuns adevărat.)

```
byte b=11;
short s=13;
result=b* ++s;
```

A. byte, short, int, long, float, double
B. boolean, byte, short, char, int, long, float, double
C. byte, short, char, int, long, float, double
D. byte, short, char
E. int, long, float, double