

TEHNOLOGII JAVA

PENTRU DEZVOLTAREA APLICAȚIILOR

LUCRARE DE LABORATOR 2

Elemente de bază ale limbajului Java

I. SCOPUL LUCRĂRII

Lucrarea de față are rolul de a prezenta și familiariza studentul cu noțiuni și elemente de bază ale limbajului Java cum ar fi: vectori – construirea și inițializarea acestora, variabile automate și inițializarea acestora, operatori și utilizarea acestora, ordinea evaluării și promovarea aritmetică a operanzilor.

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

II. NOȚIUNI TEORETICE

1. Vectori în Java

Un vector în Java este o colecție omogenă și ordonată de primitive, referințe ale obiectelor sau alți vectori. Pentru a crea și utiliza un vector trebuie urmați trei pași:

1. declararea
2. construirea
3. inițializarea

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

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

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

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

```
// declararea si construirea vectorului intr-o singura linie  
int 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 și 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;
```

2. Variabile automate și 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 și 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*”.

3. Operatori în limbajul Java

Vezi Curs 2, secțiunea 2.7.

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 afișate 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 supuși 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 și biți de 0 sunt introduși 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, ș.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, și 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 și 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** și **c** trebuie să fie compatibile și sunt făcute identice prin promovare aritmetică
- tipul expresiei **x** trebuie să fie boolean
- tipul expresiilor **b** și **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 "+=" și "*=" au o funcție compusă de calcul și asignare. Acești operatori compuși au forma generală "**op**=", unde **op** poate oricare dintre operatorii binari non-booleeni. În general, pentru orice expresii compatibile x și y, expresia **x op=y** este o prescurtare pentru **x=x op y**.

Totuși 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 așa 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 situație, operația de cast este implicită.

4. Ordinea evaluării operanzilor

Se face de la stânga la dreapta. Rezultatul oricărei expresii va fi ca și cum toți operanzii sunt evaluați de la stânga la dreapta, chiar dacă ordinea executării operațiilor este câteodată diferită (și 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 și 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]**.

5. 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, și sunt efectuate automat de către sistem. Regulile de promovare aritmetică disting între operatorii unari și 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 sunt convertiți la **int**

Pentru exemplul anterior, iată cum lucrează sistemul:

1. Variabila **s** de tip **short** este promovată la un **int** și 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**), și este promovat la **double** apoi se efectuează compararea. Rezultatul unei comparații este întotdeauna de tip **boolean**.

III. MODUL DE LUCRU

1. Se editează codul sursă al programului Java folosind un editor de text disponibil (de ex., se poate utiliza Notepad).
2. Se salvează fișierul cu extensia **.java**. Fișierul trebuie salvat la următoarea locație: **c:\JBulider7\jdk1.3.1\bin**
3. Compilarea mini-aplicației Java se va face din linia de comandă. Se poate proceda astfel. Se deschide o fereastră MS-Dos: Start ->Run, se tipărește **command** în căsuța de text și se apasă butonul OK. Printr-o schimbare de directoare și subdirectoare se trece la locația: **c:\JBulider7\jdk1.3.1\bin**. Sau, se lansează Windows Commander. Se trece la locația **c:\JBulider7\jdk1.3.1\bin**. Se deschide o fereastră MS-Dos: Commander ->Run Dos.
4. Pentru compilare, se tipărește la prompter **javac nume_fișier_sursă.java** și se apasă Enter. De ex., dacă fișierul se numește Test.java, se va scrie **javac Test.java**. În cazul în care programul conține erori acestea vor fi semnalate și afișate în fereastra MS-Dos, după care va apare iar prompter-ul. Dacă programul nu conține erori și compilarea se face cu succes, atunci va apare numai prompter-ul.
5. Pentru rularea aplicației Java, se lansează interpretorul Java. Se tipărește la prompter următoarea comandă **java nume_clasă_care_conține_main** și se apasă Enter. De ex., dacă clasa din program care conține metoda *main()* se numește Test, se va scrie **java Test**.

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 și alegerea rezultatului.

1. După executarea fragmentului de cod de mai jos, care sunt valorile variabilelor x , a și 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 și 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

- Se vor testa practic răspunsurile la întrebările exercițiului anterior.