Tipuri de date, Modificatori si Expresii

CONTINUT

Observatie
Java derivând din C++ si implicit cu C. Sintaxa va fi aproximativ identica, lucru care va fi extreme de convenabil pentru cei care sunt familiarizati cu acest limbaj.

Unitati lexicale

Unitatile lexicale sunt elementele de baza cu care se construieste semantica programelor Java. În sirul de cuvinte de intrare, unitatile lexicale sunt separate între ele prin comentarii si spatii. Unitatile lexicale în limbajul Java pot fi:

- Cuvinte cheie
- Identificatori
- Literali
- Separatori
- Operatori

Cuvinte cheie

Cuvintele cheie sunt secvente de caractere ASCII rezervate de limbaj pentru uzul propriu. Cu ajutorul lor, Java îsi defineste unitatile sintactice de baza. Nici un program nu poate sa utilizeze aceste secvente altfel decât în modul în care sunt definite de limbaj. Singura exceptie este aceea ca nu exista nici o restrictionare a aparitiei cuvintelor cheie în siruri de caractere sau comentarii.

Cuvintele cheie ale limbajului Java sunt:

abstract	**for**	**public**
boolean	future	rest
break	generic	**return**
byte	goto	**short**
case	**if**	**static**
cast	**implements**	**super**
catch	**import**	**switch**
char	inner	**synchronized**
class	**instanceof**	**this**
const	**intinterface**	**throw**
continue	**long**	**throws**
default	**native**	transient
do	**new**	**try**
double	**null**	var
else	operator	**void**
extends	outer	**volatile**
final	**package**	**while**
finally	**private**	byvalue
float	**protected**	

Dintre acestea, cele îngrosate sunt efectiv folosite, iar restul sunt rezervate pentru viitoare extensii ale limbajului.

Declaratii si Expresii

O declaratie este cea mai simpla operatie care poate fi realizata în Java. Iata câteva exemple de declaratii:

```
int i = 1;
import java.awt.Font;
System.out.println("Aceasta motocicleta este o "
    + culoare + " " + marca);
m.StareMotor = true;
```

Declaratiile întorc uneori valori, spre exemplu când sunt adunate doua numere sau când este testata o valoare daca este egala cu alta. Acest tip de declaratii poarta numele de expresii.

Spatiile libere, ca si în C nu au nici o importanta. O declaratie poate contineo singura linie sau mai multe linii, compilatorul Java fiind capabil sa citeasca decsaratia fara probleme. Cel mai important lucru care trebuie avut în vedere, în legatura cu declaratiile Java este faptul ca acestea trebuie sa se termine cu punct si virgula (;). În cazul în care lipseste acest semn, programul Java nu va putea fi compilat.

Declaratiile Java permit structuri compuse, blocuri, care pot fi plasate oriunde poate fi plasata o singura declaratie. Declaratiile de tip bloc sunt delimitate de **({ })**.

Variabile si Tipuri de date

Variabilele sunt locatii de memorie în care sunt memorate valorile. Fiecare are un nume, un tip si o valoare. Înainte de a utilize o variabila, aceasta trebuie declarata. Dupa ce aceasta a fost declarata, i se poate atasa o valoare.

Java opereaza cu trei tipuri de variabile: variabile de instanta, variabile de clasa si variabile locale.

Variabilele de instanta sunt utilizate pentru definirea atributelor unor obiecte particulare. Variabilele de clasa sunt similare variabilelor de instanta, cu observatia ca acestea se aplica tuturor claselor de instante, fara a avea valori diferite de la obiect la obiect.

Variabilele locale sunt declarate si utilizate în interiorul definitiei metodei, spre exemplu pentru numararea unor cicluri, unor variabile temporare sau memorarea unor variabile utilizate numai în interiorul definitiei metodei. Odata ce metoda îsi termina executia, respectivele variabile îsi înceteaza existenta. Variabilele locale sunt utilizate pentru stocarea informatiilor necesare unei singure metode, în timp ce variabilele de instanta vor memora datele necesare mai multor metode din cadrul unui obiect.

Prin urmare declararea variabilelor fiind aproximativ identica, diferenta dintre variabilele de clasa, variabilele de instanta si variabilele locale se datoreaza numai duratei de existenta a acestora.

Atentie
Limbajul Java fiind un limbaj orientat pe obiect nu dispune de variabile globale, valide de-a lungul întregului program. Pentru asigurarea comunicarii globale între obuecte vor fi utilizate variabile de instanta si variabile de clasa. La alcatuirea aplicatiei programatorul va trebui sa realizeze programul în termeni specifici programarii orientate pe obiecte, mai ales asupra modului în care obievctele pot comunica unul cu altul.

Declararea Variabilelor

Pentru a utiliza o variabila în Java, va trebui ca aceasta sa fie initial declarata. Declararea unei variabile consta în declararea tipului acesteia si apoi a numelui variabilei:

```
int varstaMea;
String numeleMeu;
boolean plictisit;
```

Definirea variabilelor poate fi realizata oriunde în cadrul definitiei metodei. Localizarea definirii variabilelor este realizata în general la începutul definitiei metodei:

```
public static void main (String args[]) {
    int numar;
    String titlu;
    boolean obosit;
```

```
...
}
```

Modul de aranjare a declararii variabilelor poate fi facut pe o singura linie, relative la variabilele de acelasi tip:

```
int x, y, z;
String preNume, numeFamilie;
```

Totodata fiecare variabila poate fi si initializata:

```
int varstaMea, varstaTa, varstaLui = 20;
String numeleMeu = "Alin";
boolean obosit = true;
int a = 4, b = 5, c = 6;
```

Daca sunt mai multe variabile pe o aceeasi linie cu o singura initializare, aceasta initializare se va aplica numai la ultima variabila din declaratie. Initializarea mai multor variabile prin valori diferite se realizeaza prin separatori de tip virgula.

Variabilele locale trebuie initializate înainte de a fi utilizate, în caz contrar acestea vor genera o eroare de compilare. Pentru a preîntâmpina acest tip de eroare se recomanda initializarea acestor variabile. Variabilele de instanta si cele de clasa nu au astfel de restrictii. (Valorile de initializare depend de tipul de variabila: `null` pentru instantele unri clase, `0` pentru variabilele numerice, `'\0'` pentru caractere si `false` pentru variabile logice.)

Observatii referitoare la numele variabilelor

Numele variabilelor încep cu o litera, o liniuta de subliniere (_) sau semnul dolarului ($). Acestea nu pot începe cu un numar. Dupa primul character, variabila poate contine orice numar sau litera. Simbolurile `%`, `*`, `@` sunt de obicei cuvinte cheie si prin urmare se recomanda evitarea folosirii lor în alcatuirea numelor unor variabile.

Limbajul Java utilizeaza setul de caractere Unicode. Unicode este un set de caractere care contine nu numai setud standard ASCII, dar si cateva mii de alte caractere care reprezinta cel mai international alfabet.

Cifrele Unicode sunt definite în urmatoarele intervale:

Reprezentare Unicode	Caracter ASCII	Explicatie
\u0030-\u0039	0-9	cifre ISO-LATIN-1
\u0660-\u0669		cifre Arabic-Indic
\u06f0-\u06f9		cifre Eastern Arabic-Indic
\u0966-\u096f		cifre Devanagari
\u09e6-\u09ef		cifre Bengali
\u0a66-\ u0a6f		cifre Gurmukhi

\u0ae6-\u0aef		cifre Gujarati
\u0b66-\u0b6f		cifre Oriya
\u0be7-\u0bef		cifre Tamil
\u0c66-\u0c6f		cifre Telugu
\u0ce6-\u0cef		cifre Kannada
\u0d66-\u0d6f		cifre Malayalam
\u0e50-\u0e59		cifre Thai
\u0ed0-\u0ed9		cifre Lao
\u1040-\u1049		cifre Tibetan

Cifrele Unicode.

Un caracter Unicode este o litera daca este în urmatoarele intervale si nu este cifra:

Reprezentare Unicode	Caracter ASCII	Explicatie
\u0024	$	semnul dolar (din motive istorice)
\u0041-\u005a	A-Z	litere majuscule Latin
\u005f	_	underscore (din motive istorice)
\u0061-\u007a	a-z	litere minuscule Latin
\u00c0-\u00d6		diferite litere Latin cu diacritice
\u00d8-\u00f6		diferite litere Latin cu diacritice
\u00f8-\u00ff		diferite litere Latin cu diacritice
\u0100-\u1fff		alte alfabete si simboluri non-CJK
\u3040-\u318f		Hiragana, Katakana, Bopomofo, si Hangul
\u3300-\u337f		cuvinte patratice CJK
\u3400-\u3d2d		simboluri Hangul coreene
\u4e00-\u9fff		Han (Chinez, Japonez, Corean)
\uf900-\ufaff		compatibilitate Han

Literele Unicode.

Dupa cum se stie limbajul Java este un limbaj case sensitive, facând diferenta între numele de variabile scrise cu litere mari si numele de variabila scrisa cu litera mica: litera `X` este diferita de variabila `x`, iar variabila `rose` nu este identica cu variabila `Rose` si nici cu `ROSE`.

Prin conventie variabilele Java au un nume semnificativ destinatiei sale, continând mai multe cuvinte combinate. Regula este ca prima litera din cadrul numelui variabilei sa fie litera mica:

```
Button unButon;
long unNumarRealMare;
boolean dacaAziPloua;
```

Tipuri de variabile

În afara de nume fiecare variabila trebuie sa aiba definit un tip de data careia îi apartine, definind tipul de valoare ce va fi memorata de catre variabila. Tipurile de variabile sunt de trei feluri:

- Unul dintre cele 8 tipuri primitive de date
- Numele unei clase sau al unei instante
- Tablou

Tipuri primitive

Cele 8 tipuri primitive de date manevreaza informatii de tip intreg, numere în virgula mobila, caractere si valori logice sau booleene (`true` sau `false`). Aceste tipuri de date primitive sunt independete de masina, ceea ce înseamna ca dimensiunile lor vor fi aceleasi indifferent de masina pe care ruleaza aplicatia.

Tipul întreg este implementat prin utilizarea a 4 tipuri de date. Toate tipurile au semn si domenii diferite. Explicitarea acestor tipuri de date este realizata în tabelul urmator:

Tipul întreg.

Tip	Dimensiune	Domeniu
`Byte`	8 biti	-128 la 127
`Short`	16 biti	-32 768 la 32 767
`Int`	32 biti	-2 147 483 648 la 2 147 483 647
`Long`	64 biti	-9 223 372 036 854 775 808 la 9 223 372 036 854 775 807

Numerele cu virgula mobila sunt utilizate pentru numerele ce au o parte zecimala. Java lucreaza cu doua tipuri de numere cu virgula mobila: tipul `float` (32 biti, simpla precizie) si `double` (64 biti, dubla precizie).

Tipul octet

Între tipurile întregi, acest tip ocupa un singur octet de memorie, adica opt cifre binare. Într-o variabila de tip octet sunt reprezentate întotdeauna valori cu semn, ca de altfel în toate variabilele de tip întreg definite în limbajul Java. Aceasta conventie simplifica schema de tipuri primitive care, în cazul altor limbaje include separat tipuri întregi cu semn si fara.

Fiind vorba de numere cu semn, este nevoie de o conventie de reprezentare a semnului. Conventia folosita de Java este reprezentarea în complement fata de doi. Aceasta reprezentare este de altfel folosita de majoritatea limbajelor actuale si permite memorarea, pe 8 biti a 256 de numere începând de la `-128` pâna la `127`. Daca aveti nevoie de numere mai mari în valoare absoluta, apelati la alte tipuri întregi.

Valoarea implicita pentru o variabila neinitializata de tip octet este valoarea 0 reprezentata pe un octet.

Iata si câteva exemple de declaratii care folosesc cuvântul Java rezervat **`byte`**:

```
byte octet;
byte eleviPeClasa;
```

În continuare vom folosi interschimbabil denumirea de tip octet cu cea de tip byte.

Tipul întreg scurt

Tipul întreg scurt este similar cu tipul octet dar valorile sunt reprezentate pe doi octeti, adica 16 biti. La fel ca si la tipul octet, valorile

sunt întotdeauna cu semn si se foloseste reprezentarea în complement fata de doi. Valorile de întregi scurti reprezentabile sunt de la `-32768` la `32767` iar valoarea implicita este 0 reprezentat pe doi octeti.

Pentru declararea variabilelor de tip întreg scurt în Java se foloseste cuvântul rezervat **`short`**, ca în exemplele urmatoare:

```
short i, j;
short valoareNuPreaMare;
```

În continuare vom folosi interschimbabil denumirea de tip întreg scurt si cea de tip short.

Tipul întreg

Singura diferenta dintre tipul întreg si tipurile precedente este faptul ca valorile sunt reprezentate pe patru octeti adica 32 biti. Valorile reprezentabile sunt de la `-2147483648` la `2147483647` valoarea implicita fiind 0. Cuvântul rezervat este int ca în:

```
int salariu;
```

În continuare vom folosi interschimbabil denumirea de tip întreg si cea de tip int.

Tipul întreg lung

În fine, pentru cei care vor sa reprezinte numerele întregi cu semn pe 8 octeti, 64 de biti, exista tipul întreg lung. Valorile reprezentabile sunt de `la -9223372036854775808` la `9223372036854775807` iar valoarea implicita este 0L.

Pentru cei care nu au calculatoare care lucreaza pe 64 de biti este bine de precizat faptul ca folosirea acestui tip duce la operatii lente pentru ca nu exista operatii native ale procesorului care sa lucreze cu numere asa de mari.

Declaratia se face cu cuvântul rezervat **`long`**. În continuare vom folosi interschimbabil denumirea de tip întreg lung cu cea de tip long.

Tipuri flotante

Acest tip este folosit pentru reprezentarea numerelor reale sub forma de exponent si cifre semnificative. Reprezentarea se face pe patru octeti, 32 biti, asa cum specifica standardul IEEE 754.

Tipul flotant

Valorile finite reprezentabile într-o variabila de tip flotant sunt de forma:

```
sm2e
```

unde *s* este semnul +1 sau -1, *m* este partea care specifica cifrele reprezentative ale numarului, numita si mantisa, un întreg pozitiv mai mic decât 2^{24} iar *e* este un exponent întreg între -149 si 104.

Valoarea implicita pentru variabilele flotante este 0.0f. Pentru declararea unui numar flotant, Java defineste cuvântul rezervat **`float`**. Declaratiile se fac ca în exemplele urmatoare:

```
float procent;
float noi, ei;
```

În continuare vom folosi interschimbabil denumirea de tip flotant si cea de tip float.

Tipul flotant dublu

Daca valorile reprezentabile în variabile flotante nu sunt destul de precise sau destul de mari, puteti folosi tipul flotant dublu care foloseste opt octeti pentru reprezentare, urmând acelasi standard IEEE 754

Valorile finite reprezentabile cu flotanti dubli sunt de forma:

```
sm2e
```

unde *s* este semnul +1 sau -1, *m* este mantisa, un întreg pozitiv mai mic decât 2^{53} iar *e* este un exponent întreg între -1045 si 1000. Valoarea implicita în acest caz este 0.0d.

Pentru a declara flotanti dubli, Java defineste cuvântul rezervat **double** ca în:

```
double distantaPânaLaLuna;
```

În continuare vom folosi interschimbabil denumirea de tip flotant dublu si cea de tip double.

În afara de valorile definite pâna acum, standardul IEEE defineste câteva valori speciale reprezentabile pe un flotant sau un flotant dublu.

Reali speciali definiti de IEEE

Prima dintre acestea este NaN (Not a Number), valoare care se obtine atunci când efectuam o operatie a carei rezultat nu este definit, de exemplu 0.0 / 0.0.

În plus, standardul IEEE defineste doua valori pe care le putem folosi pe post de infinit pozitiv si negativ. Si aceste valori pot rezulta în urma unor calcule.

Aceste valori sunt definite sub forma de constante si în ierarhia standard Java, mai precis în clasa java.lang.Float si respectiv în java.lang.Double. Numele constantelor este POSITIVE_INFINITY, NEGATIVE_INFINITY, NaN.

În plus, pentru tipurile întregi si întregi lungi si pentru tipurile flotante exista definite clase în ierarhia standard Java care se numesc respectiv java.lang.Integer, java.lang.Long, java.lang.Float si java.lang.Double. În fiecare dintre aceste clase numerice sunt definite doua constante care reprezinta valorile minime si maxime care se pot reprezenta în tipurile respective. Aceste doua constante se numesc în mod uniform MIN_VALUE si MAX_VALUE.

Tipul `char` este utilizat pentru caractere individuale. Deoarece java foloseste setul de caractere Unicode, reprezentarea acestui tip se realizeaza pe 16 biti, fara semn.

Tipul boolean are doua valori `true` si `false`.

Toate tipurile primitive de date sunt scrise cu litera mica

Tipul clasa

În afara de cele 8 tipuri de date primitive, variabilele Java pot fi declarate pentru a retine o instanta a unei clase particulare:

```
String PreNume;
Font basicFont;
OvalShape galbenOval;
```

Fiecare dintre aceste variabile pot contine numele unei instante sau al unei clase sau al oricarei subclase. Utilizarea unui nume de subclasa permite referirea la instantele ce au derivate din respective subclasa, fara a mai fi nevoie de o referire explicita a acestora. Spre exemplu daca din clasa VehiculCuPatruRoti au derivate Autoturisme, Tractoare, Camioane, o referinta la clasa VehiculCuPatruRoti va realize referirea la cele 3 instante amintite anterior.

Technical Note
Java nu dispune de declaratii de tip `typedef`. Pentru crearea unui nou tip, în Java, va trevbui declarata o noua clasa, iar variabilele vor fi declarate în cadrul respectivului tip de clasa.

Asignarea valorilor unor variabile

Odata ce o variabila a fost declarata, acesteia i se poate atribui o valoare prin utilizarea unui operator de asignare `=` :

```
dimensiune = 14;
preaMultaCafea = true;
```

Comentarii

Java ofera trei tipuri de comentarii: doua tipuri pentru comentarii obisnuite, în cadrul codului sursa, iar un tip pentru realizarea unui tip special de documentare realizat prin intermediul comenzii `javadoc`.

Simbolul `/*` si `*/` definesc comentarii de tip multilinii. Întregul text continut între cele doua simboluri este ignorat în clipa compilarii aplicatiei:

```
/* Acest tip de comentariu are valoare numai pentru programator
ajutândul în inserarea unor elemente relativ la versiune,
varianta
data realizarii si orice alte elemente considerate utile de catre
programator.
*/
```

Acest tip de comentariu nu permite implementarea comentariilor în cadrul altor comentarii.

Urmatorul semn `//` poate fi folosit pentru comentariu valid de-a lungul unei linii. Întregul text, de la semnul // la sfârsitul liniei va fi ignorat:

```
int voci = 8; // sunt oare chiar 8 voci?
```

Cel de-al treilea tip de comentariu este cel utilizat pentru generarea mesajelor utilizate în cadrul comenzii javadoc. Acest tip de comentariu începe cu `/**` si se termina cu `*/`. Textul, în cadrul compilarii va fi complet ignorat.

Literali

Literali sunt termeni care implementeaza exact tipul de data ce este introdus de catre utilizator. Acest tip de data este un tip de forma **W**hat **Y**ou **T**ype **I**s **W**hat **Y**ou **G**et . Spre exemplu daca veti tipari cifra 7, atunci veti obtine un întreg egal cu 7. Daca veti tipari litera `'a'`, veti obtine o data de tip character egala cu litera `a`. Literali sunt utilizati pentru a indica valori le simple din cadrul programelor Java.

Definitie
Un literal este o valoare simpla în care ceea ce introduce este ceea ce obtii. Numere, caractere si siruri de caractere sunt exemple de literali.

Literali par a fi un tip de date intuitive, dar sunt cazuri speciale de literali numerici, de tip character, de tip string sau de tip boolean.

Literali de tip numeric

Se disting mai multe tipuri de literali întregi. Un literal mai mare decât tipul `int` este convertit automat la tipul `long`. Convertirea la tipul long a unei valori mai mici se poate realize prin atasarea la sfârsitul numarului a literei l sau L (spre exemplu, `9L` este `long` integerul valorii `9`). Intregii negative sunt precedati de semnul minus -exemplu, `-53`.

Întegii pot fi exprimati sub forma octala sau hexazecimala: un `0` prefix indica folosirea codului octal – exemplu - `0456` sau `0104`. Un suffix `0x` (sau `0X`) indica folosirea codului hexazecimal (`0xAF`, `0XAe4a`).

Literali cu virgula mobila au doua parti: o parte întrega si una zecimala - `4.666666`. Precizia utilizata în descrierea literalului de tip virgula mobila este dubla precizie. Un numar poate fi fortat sa fie exprimat prin literal de tip virgula mobila prin atasarea sufixului `f` (sau `F`) la respectivul numar - `23.156F`.

Pentru descrierea literalilor în virgula mobila poate fi folosita si descrierea numerica cu exponent negative sau pozitiv prin utilizarea literei `e` sau `E` urmata de exponent: `15e15` or `.36E-3`.

Valoarea maxima a unui literal întreg normal este de `2147483647` (2^{31}-1), scrisa în baza 10. În baza 16, cel mai mare literal pozitiv se scrie ca `0x7fffffff` iar în baza 8 ca `017777777777`. Toate trei scrierile reprezinta de fapt aceeasi valoare, doar ca aceasta este exprimata în baze diferite.

Cea mai mica valoare a unui literal întreg normal este -2147483648 (-2^{31}), respectiv `0x80000000` si `020000000000`. Valorile `0xffffffff` si `037777777777` reprezinta amândoua valoarea -1.

Specificarea în sursa a unui literal întreg normal care depaseste aceste limite reprezinta o eroare de compilare. Cu alte cuvinte, daca folosim în sursa numarul: `21474836470` de exemplu, fara sa punem sufixul de numar lung dupa el, compilatorul va genera o eroare la analiza sursei.

Valoarea maxima a unui literal întreg lung este, în baza 10, `9223372036854775807L` (`2`^{63}`-1`). În octal, asta înseamna `07777777777777777777L` iar în baza 16 `0x7fffffffffffffffL`. În mod asemanator, valoarea minima a unui literal întreg lung este `-9223372036854775808L` (`-2`^{63}`-1`), în octal aceasta valoare este `04000000000000000000L` iar în baza 16 este `0x8000000000000000L`.

Valoarea maxima a unui literal flotant normal este `3.40282347e+38f` iar valoarea cea mai mica reprezentabila este `1.40239846e-45f`, ambele reprezentate pe 32 de biti.

Valoarea maxima reprezentabila a unui literal flotant dublu este de `1.79769313486231570e+308` iar valoarea cea mai mica reprezentabila este `4.94065645841246544e-324`, ambele reprezentate pe 64 de biti.

Conversii de extindere a valorii

În aceste conversii valoarea se reprezinta într-o zona mai mare fara sa se piarda nici un fel de informatii. Iata conversiile de extindere pe tipuri primitive:

- **`byte`** la **`short`**, **`int`**, **`long`**, **`float`** sau **`double`**
- **`short`** la **`int`**, **`long`**, **`float`** sau **`double`**
- **`char`** la **`int`**, **`long`**, **`float`** sau **`double`**
- **`int`** la **`long`**, **`float`** sau **`double`**
- **`long`** la **`float`** sau **`double`**
- **`float`** la **`double`**

Sa mai precizam totusi ca, într-o parte din aceste cazuri, putem pierde din precizie. Aceasta situatie apare de exemplu la conversia unui **`long`** într-un **`float`**, caz în care se pierd o parte din cifrele semnificative pastrându-se însa ordinul de marime. De altfel aceasta observatie este evidenta daca tinem cont de faptul ca un **`long`** este reprezentat pe 64 de biti în timp ce un **`float`** este reprezentat doar pe 32 de biti.

Precizia se pierde chiar si în cazul conversiei **`long`** la **`double`** sau **`int`** la **`float`** pentru ca, desi dimensiunea zonei alocata pentru cele doua tipuri este aceeasi, numerele flotante au nevoie de o parte din aceasta zona pentru a reprezenta exponentul.

În aceste situatii, se va produce o rotunjire a numerelor reprezentate.

Conversii de trunchiere a valorii

Conventiile de trunchiere a valorii pot produce pierderi de informatie pentru ca ele convertesc tipuri mai bogate în informatii catre tipuri mai sarace. Conversiile de trunchiere pe tipurile elementare sunt urmatoarele:

- **`byte`** la **`char`**
- **`short`** la **`byte`** sau **`char`**
- **`char`** la **`byte`** sau **`short`**
- **`int`** la **`byte`**, **`short`** sau **`char`**
- **`long`** la **`byte`**, **`short`** **`char`**, sau **`int`**
- **`float`** la **`byte`**, **`short`**, **`char`**, **`int`** sau **`long`**
- **`double`** la **`byte`**, **`short`**, **`char`**, **`int`**, **`long`** sau **`float`**.

În cazul conversiilor de trunchiere la numerele cu semn, este posibil sa se schimbe semnul pentru ca, în timpul conversiei, se îndeparteaza pur si simplu octetii care nu mai încap si poate ramâne primul bit diferit de vechiul prim bit. Copierea se face începând cu octetii mai putin semnificativi iar trunchierea se face la octetii cei mai semnificativi.

Prin *octetii cei mai semnificativi* ne referim la octetii în care sunt reprezentate cifrele cele mai semnificative. *Cifrele cele mai semnificative* sunt cifrele care dau ordinul de marime al numarului. De exemplu, la numarul 123456, cifrele cele mai semnificative sunt primele, adica: 1, 2, etc. La acelasi numar, *cifrele cele mai putin semnificative* sunt ultimele, adica: 6, 5, etc.

Literali logici

Literali de tip boolean sunt implementati prin cuvintele cheie `true` si `false`. Aceste cuvinte cheie pot fi folosite oriunde este nevoie de un test sau singurele valori posibile sunt valorile logice.

Literali de tip caracter

Literali de tip character sunt exprimati printr-un singur character înconjurat de apostrof: `'a'`, `'#'`, `'3'`. Caracterele sunt memorate utilizând setul de caractere de 16-biti Unicode. În tabelul de mai jos sunt listate codurile speciale ale caracterelor neprintabile.

Table 3.2. Character escape codes.

Escape	Meaning
`\n`	Linie noua
`\t`	Tab
`\b`	Backspace
`\r`	Carriage return

`\f`	Formfeed
`\\`	Backslash
`\'`	Apostrof
`\"`	Ghilimele duble
`\ddd`	Octal
`\xdd`	Hexadecimal
`\udddd`	Unicode character

Literali de tip String

O combinatie de carcatere formaeaza un sir de caractere sau mai precis un string. String-urile în Java sunt instante ale clasei `String`. Acestea nu sunt simple siruri de caractere ca în C sau C++. Ele dispun de o serie de caracteristici de tip sir (se poate identifica si testa lungimea sirului, poate fi accesat la nivel de element sirul de caractere). Deoarece sirurile de caractere sunt obiecte , ele dispun de metode ce permit combinarea acestora,testarea lor si modificarea lor extreme de usor.

Sirurile de caractere sunt scrise ca fiind cuprinse între ghilimele:

```
"Acesta este un sir de caractere."
"" //Acesta este un sir gol
```

String-urile pot contine si elemenet de tip newline, tab, si caractere Unicode :

```
"Un sir cu un  \t tab în el"
"În interiorul unui sir poate fi inclus \"un alt sir de
caractere\" ca într-un bloc "
"Acest sir este un sir cu caractere Unicode\u2122"
```

Utilizarea unui string literal în Java conduce la creerea unei instante a clasei `String` având o valoare egala cu valoarea data.

Expresii si operatori

Expresiile sunt cele mai simple forme de declaratii din Java care acopera toate domeniile. Toate expresiile când sunt evaluate întorc o valoare. Datorita acestui fapt se poate atribui unei variabile rezultatul unei expresii Java.

Toate expresiile Java utilizeaza operatori. Acestia sunt simboluri speciale pentru operatii aritmetice, diverse forme de asignare, incrementare si decrementare si operatii logice.

Definitie
Expresiile sunt declaratii care întorc o valoare.
Definitie

> Operatorii sunt simboluri speciale care sunt utilizate în general în cadrul expresiilor.

Operatii aritmetice

Java ofera posibilitatea implementarii a 5 operatii aritmetice elementare:

Table 3.3. Operatorii aritmetici.

Operator	Semnificatie	Exemplu
+	Adunare	`3 + 4`
-	Scadere	`5 - 7`
*	Înmultire	`5 * 5`
?	Împartire	`14 / 7`
%	Modul	`20 % 7`

Fiecare operator are nevoie de doi operanzi, unul pentru fiecare parte a operandului. Operatorul minus (`-`) poate fi folosit cu un singur operand pentru a nega respective valoare.

Împartirea a doi întregi are ca rezultat tot un întreg. Spre exemplu împartirea lui `31 la 9, mai precis 31/9` are ca rezultat `3`.

Modulul (`%`) are ca rezultat restul împartirii celor doi operanzi. Spre exemplu, `31 % 9` are ca rezultat `4`.

Atunci când ambii operanzi sunt de acelasi tip rezultatul este de acelasi tip cu operanzii. Daca unul dintre operanzi este de tip diferit spre exemplu `long`, rezultatul este de tip `long`. Daca un operand este de tip virgula mobila, atunci rezultatul este de tip virgula mobila.

Program expresii aritmetice.

```
 1: class ArithmeticTest {
 2: public static void main (String args[]) {
 3:     short x = 6;
 4:     int y = 4;
 5:     float a = 12.5f;
 6:     float b = 7f;
 7:
 8:     System.out.println("x este " + x + ", y este " + y);
 9:     System.out.println("x + y = " + (x + y));
10:     System.out.println("x - y = " + (x - y));
11:     System.out.println("x / y = " + (x / y));
12:     System.out.println("x % y = " + (x % y));
13:
14:     System.out.println("a este " + a + ", b este " + b);
15:     System.out.println("a / b = " + (a / b));
16: }
17: }
```

```
x este 6, y este 4
x + y = 10
x - y = 2
x / y = 1
x % y = 2
a este 12.5, b este 7
a / b = 1.78571
```

Metoda `System.out.println()` permite afisarea unui mesaj la iesirea standard a sistemului – displayul sau o fereastra speciala sau chiar un alt fisier, în funcsie de setarile realizate. Metoda `System.out.println()` utilizeaza un singur argument - de tip string-b dar prin utilizarea operatorului de concatenare poate realiza un sir complex.

Declaratii - extensie

Asignarea unei valori unei variabile este un caz particular de expresie:

```
x = y = z = 0;
```

În acest exemplu toate variabilele sunt initializate cu valoarea `0`.

Partea dreapta a expresiei este evaluata, înainte de realizarea asignarii. Acest lucru înseamna ca în cazul expresiei `x = x + 2` prima data este adunat la valoarea lui x valoarea `2` iar apoi noua valoare obtinuta este asignata variabilei `x`.

Operatori de asignare.

Expresie	Semnificatie
`x += y`	x = x + y
`x -= y`	x = x – y
`x *= y`	x = x * y
`x /= y`	x = x ?y

Incrementare si decrementare

Structurile de tip `++` si `–` sunt utilizate pentru incrementarea sau decrementarea unei variabile cu o unitate. Spre exemplu `x++` este echivalent cu expresia `x = x + 1`. Similar `x–` realizeaza decrementarea variabilei cu 1.

Operatorii de decrementare sau incrementare pot fi situati înaintea sau dupa variabile, lucru care pentru incrementari sau decrementari simple nu au nici o importanta. În cazul exresiilor complexe aceasta localizare are conotatii diferite.

Iata spre exemplu urmatoarele expresii:

```
y = x++;
y = ++x;
```

În primul caz `y` prea valoarea lui `x` dupa care x este incrementat; în cel de-al doilea caz valoarea lui `x` va fi asignata lui `y` numai dupa ce x a fost incrementat.

Program de exemplificare a prefix-ului si a postfix-ului operatorului de incrementare.

```
 1: class PrePostFixTest {
 2:
 3: public static void main (String args[]) {
 4:     int x = 0;
 5:     int y = 0;
 6:
 7:     System.out.println("x si y sunt " + x + " si " + y );
 8:     x++;
 9:     System.out.println("x++ rezulta in " + x);
10:     ++x;
11:     System.out.println("++x rezulta in " + x);
12:     System.out.println("Se sterge x initializand cu 0.");
13:     x = 0;
14:     System.out.println("------------");
15:     y = x++;
16:     System.out.println("y = x++ (postfix) rezulta in:");
17:     System.out.println("x este " + x);
18:     System.out.println("y este " + y);
19:     System.out.println("------------");
20:
21:     y = ++x;
22:     System.out.println("y = ++x (prefix) rezulta in:");
23:     System.out.println("x este " + x);
24:     System.out.println("y este " + y);
25:     System.out.println("------------");
26:
27: }
28: }
```

```
x si y sunt 0 si 0
x++ rezulta in 1
++x rezulta in 2
Se sterge x initializand cu 0.
------------
y = x++ (postfix) rezulta in:
x este 1
y este 0
------------
y = ++x (prefix) rezulta in:
x este 2
y este 2
------------
```

La începutul programului atât x cât si y utilizând operatorul ++ atât ca `prefix` cât si ca `postfix` ofera aceleasi rezultate prin incrementarea variabilei.

În cadrul expresiei `y = x++`, în care operatorul de incrementare este `postfix` valoarea lui `x` este asignata lui `y` si apoi este incrementata valoarea lui `x`. Deci rezultatul este `: y` ia valoarea originala a lui `x` (`0`), iar aopi `x` este incrementat.

În partea a treia, utilizarea operatorului de incrementare ca `prefix y = ++x` conduce la incrementarea initiala a lui x, urmata de asignarea valorii rezultat variabilei `y`. Deoarece `x` fusese `1` la pasul anterior rezultatul final este `2`, atât pentru valoarea lui `x` cât si pentru valoarea lui `y`.

Comparatii

Java ofera câteva modalitati de comparare a doua valori. Toate aceste operatii întorc o valoarea logica – booleana (`true` sau `false`).

Table 3.5. Operatorii de comparare.

Operator	Semnificatie	Exemplu
==	Egal	x == 3
!=	Diferit	x != 3
<	Mai mic decât	x < 3
>	Mai mare decât	x > 3
<=	Mai mic sau cel mult egal decât	x <= 3
>=	Mai mare sau cel mult egal decât	x >= 3

Operatori logici

Expresiile ce au ca rezultat valori logice pot fi combinate prin utilizarea unor operatori logici ce reprezinta combinatii logice de tip `AND`, `OR`, `XOR`, si `NOT`.

Pentru functia logica `AND` sunt utilizati operatorii `&` sau `&&` operators. Diferenta dintre cei doi operatori se datoreaza faptului ca daca primul `&`, exmineaza fiecare element al celor doi operanzi, cel de-al doilea `&&`, examineaza primul termen, care daca este fals întoarce automat rezultatul fals, expresia celui de-al doilea operand nemaifiind evaluata. Acest al doilea operand poarta numele de operand de scurt circuit.

Expresia `OR` are în mod similar doua forme `|` si `||`. Cea de-a doua expresie , de scurt circuit, daca examinând primul operand identifica valoarea true, rezultatul întors este true, fara a mai evalua cel de-al doilea operand.

Iata toate posibilitatile de combinare pentru operatorii && si || :

```
true && true == true
true && false == false
false && true == false
false && false == false
true || true == true
true || false == true
false || true == true
false || false == false
```

Operatorul `XOR, ^`, are rezultatul true numai daca operanzii sunt diferiti din punct de vedrere logic si false în celalalt caz.

În general, numai `&&` si `||` sunt utilizati în cadrul combinatiilor logice uzuale, operatorii. `&, |,` and `^` fiind utilizati în cadrul operatiilor logice la nivel de bit.

Operatorul `NOT, !` utilizeaza un singur argument, rezultatul fiind negarea valorii logice a variabilei curente; daca `x` este true, `!x` este false.

Observatie

Tabelul de adevar pentru operatia logica *si* (&):

1 & 1 == 1
1 & 0 == 0
0 & 1 == 0
0 & 0 == 0

Daca apelam operatorul **&** pe numerele reprezentate binar:
00101111
01110110
rezultatul este:
00100110
Primul numar reprezinta cifra 47, al doilea 118 iar rezultatul 38, deci:
47 & 118 == 38

Tabela de adevar pentru operatia logica *sau* (|) :

1 | 1 == 1
1 | 0 == 1
0 | 1 == 1
0 | 0 == 0

Tabela de adevar pentru operatia logica *sau exclusiv* (^) :

1 ^ 1 == 0
1 ^ 0 == 1
0 ^ 1 == 1
0 ^ 0 == 0

Operatori pe bit (unari)

Operatorii la nivel de bit sunt utilizati în implementarea functiilor copmplexe ce implica operatii cu întregi. În cele ce urmeaza sunt enumerati principalii operator ice lucreaza cu operanzi de dimensiune 1:

Operatori la nivel de bit.

Operator	Semnificatie
&	`AND` la nivel de bit
|	`OR` la nivel de bit

^	XOR la nivel de bit
<<	Deplasare stânga
>>	Deplasare dreapta
>>>	Deplasare dreapta cu completare cu Zero
~	Complement la nivel de bit
<<=	Deplasare stânga si asignare (x = x << y)
>>=	Deplasare dreapta si asignare (x = x >> y)
>>>=	Deplasare dreapta cu completare cu Zero si asignare (x = x >>> y)
x&=y	AND si asignare (x = x & y)
x|=y	OR si asignare (x = x | y)
x^=y	XOR si asignare (x = x ^ y)

Observatie: Operatori de deplasare stânga, dreapta cu sau fara completare cu Zero: **>>, <<, >>>**

Operatorii de deplasare se pot aplica doar pe valori primitive întregi. Ei reprezinta respectiv operatiile de deplasare cu semn stânga (**<<**) si dreapta (**>>**) si operatia de deplasare fara semn spre dreapta (**>>>**).

Deplasarile cu semn lucreaza la nivel de cifre binare. Cifrele binare din locatia de memorie implicata sunt mutate cu mai multe pozitii spre stânga sau spre dreapta. Pozitia binara care reprezinta semnul ramâne neschimbata. Numarul de pozitii cu care se efectueaza mutarea este dat de al doilea operand. Locatia de memorie în care se executa operatia este locatia în care este memorat primul operand.

Deplasarea cu semn la stânga reprezinta o operatie identica cu înmultirea cu 2 de n ori, unde n este al doilea operand. Deplasarea cu semn la dreapta reprezinta împartirea întreaga. În acest caz, semnul este copiat în mod repetat în locurile ramase goale. Iata câteva exemple:

```
255 << 3 == 2040
// 00000000 11111111 -> 00000111 11111000
255 >> 5 == 7
// 00000000 11111111 -> 00000000 00000111
```

Deplasarea fara semn la dreapta, muta cifrele binare din operand completând spatiul ramas cu zerouri:

```
0xffffffff >>> -1 == 0x00000001
// 1111 1111 1111 1111 1111 1111 1111 1111 -> 0000 0000 0000 0000 0000 0000
0000 0001
0xffffffff >>> -2 == 0x00000003
// 1111 1111 1111 1111 1111 1111 1111 1111 -> 0000 0000 0000 0000 0000 0000
0000 0011
0xffffffff >>> -3 == 0x00000007
```

```
// 1111 1111 1111 1111 1111 1111 1111 1111 -> 0000 0000 0000 0000 0000 0000
0000 0111
0xffffffff >>> 3 == 0x1fffffff
// 1111 1111 1111 1111 1111 1111 1111 1111 -> 0001 1111 1111 1111 1111 1111
1111 1111 0xffffffff >>> 5 == 0x07ffffff
// 1111 1111 1111 1111 1111 1111 1111 1111 -> 0000 0111 1111 1111 1111 1111
1111 1111
```

Succesiunea operatiilor

Operatorii de succesiune determina ordnea în care are loc evaluarea expresiilor.Spre exemplu, în cadrul urmatoarei expresii:

```
y = 6 + 4 / 2
```

are loc evaluarea expresiei e `4 ? 2` urmata de adunarea rezultatului la cifra 6. În general, incrementarea si decrementarea sunt evaluate cu prioritate, urmate de expresiile aritmetice, apoi urmeaza compararile si în final expresiile logice. Expresiile de asignare sunt realizate ultimele.

În tabelul de mai jos este specificata ordinea evaluarii operatorilor dintr-o expresie. Operatorii din primele linii sunt operatori prioritari, în timp ce operatorii din liniile inferioare sunt operatori evaluati dupa evaluarea operatorilor prioritari.

Spre exemplu rezultatul 8 pentru expresia: `y = 6 + 4 / 2` se obtine datorita faptului ca prima este evaluata împartirea si apoi adunarea.

Prioritatea operatorilor

Operator	**Explicatii**
`. [] ()`	Parentezele (`()`) sunt utilizate pentru gruparea expresiilor; punctul (`.`) este utilizat pentru accesul la metodele sau variabilele unui obiect sau clase; parantezele drepte (`[]`) sunt utilizate pentru lucrul cu siruri
`++ -- ! ~ instanceof`	Operatorul `instanceof` are ca rezultat `true` sau `false` daca obiectul este sau nu o instanta a clasei curente sau a unei sublcalse a clasei curente
`new (type)expression`	Operatorul `new` este utilizat pentru creearea unor noi instante ale unei clase; `()` în acest caz este pentru a indica o valoare de alt tip
`* / %`	Înmultire, împartire, modul
`+ -`	Adunare, scadere
`<< >> >>>`	Deplasare dreapta sau stânga la nivel de bit
`< > <= >=`	Comparare la nivel de bit
`== !=`	Egalitate
`&`	`AND`
`^`	`XOR`

`|`	`OR`
`&&`	Logical `AND`
`||`	Logical `OR`
`? :`	`if...then...else`
`= += -= *= /= %= ^=`	Diverse asignari
`&= |= <<= >>= >>>=`	Asignari

Ordinea indicata de tabelul de mai sus poate fi schimbata oricând prin folosirea parantezelor în cadrul respectivei expresii. În cazul existentei mai multor rânduri de parantze se incepe cu evaluarea parantezelor celor mai apropiate de o expresie si se continua cu urmatoarea pereche pâna ce sunt rezolvate toate seturile de paranteze. N acel moment se revine la ordinea specificata de tabelul anterior.

Rezultatul urmatoarei expresii este de aceasta data `5`, deoarece expresia `6 + 4` este evaluata prima, dupa care este realizata împartirea la `2`:

```
y = (6 + 4) / 2
```

Parentezele pot fi utilizate si pentru identificarea unei structuri de xepresii importante pentru programator, chiar daca ordinea operatiilor ar fi condos la acelasi rezultat si fara folosirea parantezelor.

Conversii de atribuire

În cazul valorilor apartinând tipurilor primitive, urmatorul tabel arata conversiile posibile. Pe coloane avem tipul de valoare care se atribuie iar pe linii avem tipurile de variabile la care se atribuie:

	boolean	char	byte	short	int	long	float	double
boolean	Da	Nu	Nu	Nu	Nu	Nu	Nu	Nu
char	Nu	Da	Da	Da	Nu	Nu	Nu	Nu
byte	Nu	Da	Da	Nu	Nu	Nu	Nu	Nu
short	Nu	Da	Da	Da	Nu	Nu	Nu	Nu
int	Nu	Da	Da	Da	Da	Nu	Nu	Nu
long	Nu	Da	Da	Da	Da	Da	Nu	Nu
float	Nu	Da	Da	Da	Da	Da	Da	Nu
double	Nu	Da	Da	Da	Da	Da	Da	Da

Conversiile posibile într-o operatie de atribuire cu tipuri primitive. Coloanele reprezinta tipurile care se atribuie iar liniile reprezinta tipul de variabila catre care se face atribuirea.

Operatii aritmetice cu siruri

Un caz special de utilizare a operatorului de adunare este crearea si concatenarea sirurilor de caractere - strings :

```
System.out.println(nume + " este " + culoare + " interesant");
```

Iesirea generata de o astfel de expresie este un singur sir de caractere în care variabilele (`nume` si `culoare`) sunt inserate în modul dorit în cadrul sirului.

Operatorul + utilizat în legatura cu siruri de caractere sau alte obiecte, realizeaza un singur sir decaractere prin concatenarea tuturor operanzilor. Daca unul dintre operanzi nu este de tip string, acesta este convertit automat la tipul sir de caractere.

Atentie
Un obiect sau tip poate fi convertit la tipul string daca este implementata metoda `toString()`. Toate obiectele au implicit o reprezentare de tip string, dar cele mai multe clase rescriu functia `toString()` pentru a realize o reprezentare mult mai clara.

Operatorul += poate fi utilizat în operatii cu siruri de caractere. Un exemplu în acest sens este expresia:

```
numeleMeu += " N.G.";
```

Aceasta expresie este echivalenta cu:

```
numeleMeu = numeleMeu + " N.G.";
```

Concluzii

Un program Java este realizat din clase si obiecte. Clasele si obiectele au în componenta lor metode si variabile, iar metodele sunt realizate din declaratii si expresii.

În ceel prezentate anterior se poate observa ca variabilele dupa ce au primit un nume pot fi de de mai multe tipuri; literali pentru a creea numere, caractere si siruri, iar operatiile ce se pot realize cu ajutorul variabilelo sunt operatii aritmetice, teste, operatii logice.

În tabelul urmator sunt trecute în revista operatiile elementare ce pot fi realizate în cadrul limbajului Java.

Operatii posibile .

Operatii – symbol	Semnificatie
+	Adunare
-	Scadere
*	Înmultire
/	Împartire
%	Modul
<	Mai mic decât
>	Mai mare decât
<=	Mai mic sau egal decât
>=	Mai mare sau egal decât
==	Egal
??	Diferit
&&	AND logic
||	OR logic
!	NOT logic
&	AND
|	OR
^	XOR
<<	Deplasare stânga
>>	Deplasare dreapta
>>>	Deplasare dreapta cu completare cu Zero
~	Complement
=	Asignare
++	Incrementare
- - - -	Decrementare
+=	Adunare si asignare
-=	Scadere si asignare
*=	Înmultire si asignare
/=	Împartire si asignare
%=	Modul si asignare
&=	AND si asignare
|=	OR si asignare
<<=	Deplasare stânga si asignare
^=	XOR si asignare
>>=	Deplasare dreapta si asignare
>>>=	Deplasare dreapta cu completare cu zero si asignare