Operatori

- atribuirea: `=`
- operatori matematici: `+, -, *, /, %`
  Este permisa notatia prescurtata de forma `lval op= rval (ex: n += 2)`
  Exista operatorii pentru autoincrementare si autodecrementare (post si pre)
  `ex: x++, ++x, n--, --n`
  Observatie: evaluarea expresiilor logice se face prin metoda *scurtcircuitului* (evaluarea se opreste în momentul în care valoarea de adevar a expresiei este sigur determinata)
- operatori logici: `&&(and), ||(or), !(not)`
- operatori relationali: `<, <=, >, <=, ==, !=`
- operatori pe biti: `& (and), |(or), ^(xor), ~(not)`
- operatori de translatie `<<, >>, >>> (shift la dreapta fara semn)`
- operatorul `if-else: expresie_logica ? val_pt_true : val_pt_false ;`
- operatorul , (virgula) folosit pentru evaluarea secventiala a operatiilor `int x=0, y=1, z=2;`
- operatorul + pentru concatenarea sirurilor:
- ```
  String s="abcd"
  int x=100;
  System.out.println(s + " - " + x);
  ```
- operatori pentru conversii (cast) : `(tip_de_data)`
- ```
  int i = 200;
  long l = (long)i; //widening conversion - conversie prin extensie
  long l2 = (long)200;
  int i2 = (int)l2; //narrowing conversion - conversie prin contractie
  ```

Comentarii

In Java exista trei feluri de comentarii:

- Comentarii pe mai multe linii, închise între /* si */.
- Comentarii pe mai multe linii care tin de documentatie, închise între /** si */. Textul dintre cele douã secvente este automat mutat în documentatia aplicatiei de catre generatorul automat de documentatie `javadoc`.
- comentarii pe o singura linie care încep cu //.

Observatii:

1. nu putem sa scriem comentarii în interiorul altor comentarii.
2. nu putem introduce comentarii în interiorul literalilor caracter sau sir de caractere.
3. secventele /* si */ pot sa apara pe o linie dupa secventa // dar îsi pierd semnificatia; la fel se întâmplã cu secventa // în comentarii care încep cu /* sau /**.

Tipuri de date

In Java tipurile de date se împart în doua categorii: **tipuri primitive de date** si **tipuri referinta**. Java porneste de la premiza ca "orice este un obiect". Asadar tipurile de date ar trebui sa fie de fapt definite de clase si toate variabilele ar trebui sa memoreze de fapt instante ale acestor clase (obiecte). In principiu acest lucru este adevarat, însa, pentru usurinta programarii, mai exista si asa numitele **tipurile primitive** de date, care sunt cele uzuale :

- aritmetice
  - întregi: `byte (1 octet), short (2), int (4), long (8)`
  - reale: `float (4 octeti), double (8)`
- caracter : `char (2 octeti)`
- logic : `boolean (true si false)`

In alte limbaje formatul si dimensiunea tipurilor primitive de date folosite într-un program pot depinde de platforma pe care ruleaza programul. In Java acest lucru nu mai este valabil, orice dependenta de o anumita platforma specifica fiind eliminata.
Vectorii, clasele si interfetele sunt **tipuri referinta**. Valoarea unei variabile de acest tip este, în contrast cu tipurile primitive, o referinta (adresa de memorie) catre valoarea sau multimea de valori reprezentata de variabila respectiva.
Exista trei tipuri de date C care nu sunt suportate de limbajul Java. Acestea sunt: `pointer, struct` si `union`. Pointerii au fost eliminati din cauza ca erau o sursa constanta de erori, locul lor fiind luat de tipul referinta, iar struct si union nu îsi mai au rostul atât timp cât tipurile compuse de date sunt formate în Java prin intermediul claselor.

Variabile

Variabilele pot avea ca tip fie un tip primitiv de data, fie o referinta la un obiect.

Declararea variabilelor	`Tip nume_variabila`
Initializarea variabilelor	`Tip nume_variabila = valoare`
Declararea constantelor	`Final Tip nume_variabila`

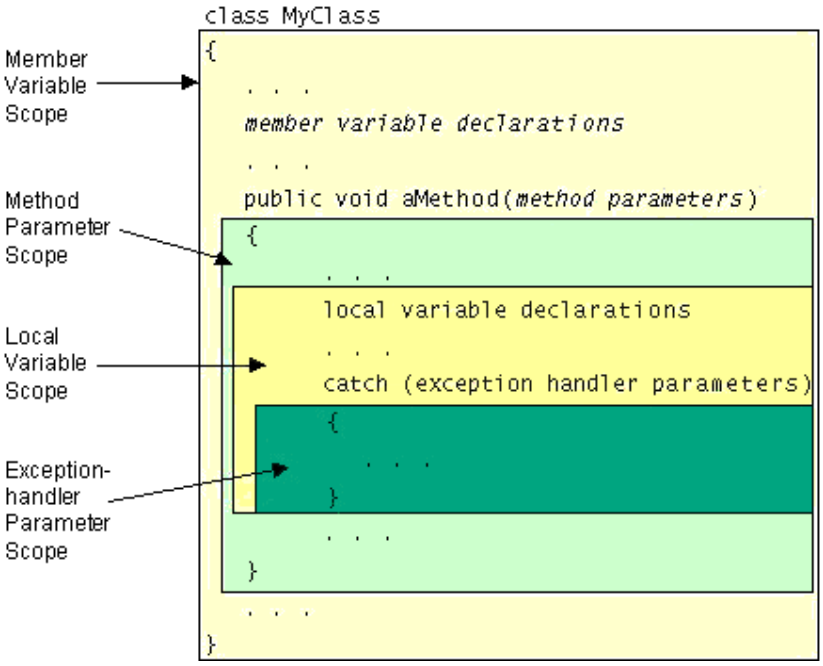
Conventia de notare a variabilelor in Java este data de urmatoarele criterii:

1. variabilele finale (constante) se scriu cu majuscule
2. variabilele normale se scriu astfel : prima litera cu litera mica, daca numele variabilei este format din mai multi atomi lexicali, atunci primele litere ale celorlalti atomi se scriu cu majuscule, de exemplu:

```
3.   final double PI = 3.14;
4.   int valoare = 100;
5.   long numarElemente = 12345678L;
6.   String bauturaMeaPreferata = "apa";
```

In functie de locul în care sunt declarate variabile se împart în urmatoatele categorii:

1. **Variabile membre**, declarate în interiorul unei clase, vizibile pentru toate metodele clasei respective si pentru alte clase în functie de nivelul lor de acces (vezi "Declararea variabilelor membre")
2. **Variabile locale**, declarate într-o metoda sau într-un bloc de cod, vizibile doar în metoda/blocul respectiv
3. **Parametri metodelor**, vizibili doar în metoda respectiva
4. **Parametrii de la tratarea exceptiilor**



Obs: variabilele declarate într-un `for` pentru controlul ciclului, ramân locale corpului ciclului.

```
for(int i=0; i<100; i++) { }
int i; //ok în Java, eroare în C++
```

Obs: Spre deosebire de C++ nu este permisa ascunderea unei variabile :

```
int x=12;
{
        int x=96; //ilegal
}
```

Controlul executiei

Instructiunile Java pentru controlul executiei sunt asemanatoare celor din C.

Instructiuni de decizie	`if-else, switch-case`
Instructiuni de salt	`for, while, do-while`
Instructiuni pt. tratarea exceptiilor	`try-catch-finally, throw`

Alte instructiuni	break, continue, label: , return

Instructiuni de decizie	
if-else	`if (exp_booleana) { /*...*/}` `if (exp_booleana) { /*...*/} else { /*...*/}`

switch-case	`switch (variabila) {`
	`case val1 : /* ... */ break;`
	`case val2 : /* ... */ break;`
	`/*...*/`
	`default : /*...*/`
	`}`

Instructiuni de salt	

for	`for(initializare; exp_booleana; pas_iteratie)`
	`Ex: for(int i=0, j=100 ; i<100 && j>0; i++, j--)`
	`{/* ... /*}`
	Obs: atât la initializare cât si în pasul de iteratie pot fi mai multe instructiuni despartite prin virgula.

while	`while (exp_booleana) {`
	`/*...*/`
	`}`

do-while	`do {`
	`/*...*/`
	`}`
	`while (exp_booleana) ;`

Instructiuni pentru tratarea exceptiilor	
try-catch-finally, throw	(vezi "Tratarea exceptiilor")
Alte instructiuni	
break	Paraseste fortat corpul iteratiei curente
continue	termina fortat iteratia curenta
return	`return [valoare];` Termina o metoda
label:	Defineste o eticheta

Atentie: In Java nu exista **goto**. Se pot însa defini etichete de forma `nume_eticheta:`, folosite în expresii de genul: `break nume_eticheata` sau `continue nume_eticheta`

```
Exemplu:
        i=0;
        eticheta:
        while (i<10) {
                System.out.println("i="+i);
                j=0;
                while (j<10) {
                        j++;
                        if (j==5) continue eticheta;
                        if (j==7) break eticheta;
                        System.out.println("j="+j);
                }
```

```
            i++;
        }
```

Vectori

Crearea unui vector

1. Declararea vectorului

```
            Tip[] numeVector;  sau
            Tip numeVector[];
    Ex:     int[] intregi;
            String adrese[];
```

6. Instantierea
   Se realizeaza prin intermediul operatorului `new` si are ca efect alocarea memoriei pentru vector, mai precis specificarea numarului maxim de elemente pe care îl va avea vectorul;

```
            numeVector = new Tip[dimensiune];
    Ex:     v = new int[10];       //se aloca spatiu pentru 10 intregi
            adrese = new String[100];
```

   Obs: declararea si instantierea unui vector pot fi facute simultan astfel:

```
Tip[] numeVector = new Tip[dimensiune];
```

10. Initializarea (optional)
    Dupa declararea unui vector, acesta poate fi initializat, adica elementele sale pot primi niste valori initiale, evident daca este cazul pentru asa ceva. In acest caz instantierea lipseste, alocarea memoriei facându-se automat în functie de numarul de elemente cu care se initializeaza vectorul.

```
    Ex:     String culori[] = {"Rosu", "Galben", "Verde"};
            int []factorial = {1, 1, 2, 6, 24, 120};
```

Observatii:
Primul indice al unui vector este 0, deci pozitiile unui vector cu n elemente vor fi cuprinse între 0 si n-1
Nu sunt permise constructii de genul **`Tip numeVector[dimensiune]`**, alocarea memoriei facându-se doar prin intermediul opearatorului `new`.

```
Ex:     int v[10];             //ilegal
        int v[] = new int[10]; //corect
```

Vectori multidimensionali
In Java tablourile multidimensionale sunt de fapt vectori de vectori.

```
Ex:     int m[][];             //declararea unei matrici
        m = new int[5][10];    //cu 5 linii, 10 coloane
```

Obs: `m[0], m[1], ..., m[5]` sunt vectori de intregi cu 10 elemente

Dimensiunea unui vector
Cu ajutorul cuvântului cheie **`length`** se poate afla dimensiunea unui vector.

```
        int []a = new int[5];
        a.length are valoarea 5
```

```
int m = new int[5][10];
m[0].length are valoarea 10
```

Copierea vectorilor
Copierea unui vector în alt vector se face cu ajutorul metodei **`System.arraycopy`**:

```
int x[] = {1, 2, 3, 4};
int y[] = new int[4];
System.arraycopy(x,0,y,0,x.length);
```

Vectori cu dimensiune variabila
Implementarea vectorilor cu numar variabil de elemente este oferita de clasa `Vector` din pachetul `java.util`. Un obiect de tip `Vector` contine numai elemente de tip `Object`.

Siruri de caractere

In Java, un sir de caractere poate fi reprezentat printr-un vector format din elemente de tip `char`, un obiect de tip `String` sau un obiect de tip `StringBuffer`.

Declararea unui sir
Daca un sir de caractere este constant atunci el va fi declarat de tipul `String`, altfel va fi declarat cu `StringBuffer`. (vezi "Clasele `String, StringBuffer`") Exemple echivalente de declarare a unui sir:

```
String str = "abc";
char data[] = {'a', 'b', 'c'};
String str = new String(data);
String str = new String("abc");
```

Concatenarea sirurilor
Concatenarea sirurilor de caractere se face prin intermediul operatorului +.

```
String str1 = "abc" + "xyz";
String str2 = "123";
String str3 = str1 + str2;
```

In Java, operatorul de concatenare + este extrem de flexibil în sensul ca permite concatenarea sirurilor cu obiecte de orice tip care au o reprezentare de tip sir de caractere.

```
System.out.print("Vectorul v are" + v.length + " elemente")
```

Folosirea argumentelor de la linia de comanda

O aplicatie Java poate primi oricâte argumente de la linia de comanda în momentul lansarii ei. Aceste argumente sunt utile pentru a permite utilizatorului sa specifice diverse optiuni legate de functionarea aplicatiei sau sa furnizeze anumite date initiale programului.

Atentie: Programele care folosesc argumente de la linia de comanda nu sunt 100% pure Java deoarece unele sisteme de operare cum ar fi Mac OS nu au în mod normal linie de comanda.

Argumentele de la linia de comanda sunt introduse la lansarea unei aplicatii, fiind specificate dupa numele aplicatiei si separate prin spatiu. De exemplu, sa presupunem ca aplicatia `Sort` ordoneaza lexicografic liniile unui fisier si primeste ca argument numele fisierului pe care sa îl sorteze. Pentru a ordona fisierul `"persoane.txt"` lansarea aplicatiei se va face astfel:

```
java Sort persoane.txt
```

Asadar, formatul general pentru lansarea unei aplicatii care primeste argumente de la linia de comanda este:

```
java NumeAplicatie [arg1 arg2 . . . argn]
```

In cazul în care sunt mai multe, argumentele trebuie separate prin spatii iar daca unul dintre argumente contine spatii, atunci el trebuie pus între ghilimele. Evident, o aplicatie poate sa nu primeasca nici un argument sau poate sa ignore argumentele primite de la linia de comanda.

In momentul lansarii unei aplicatii interpretorul parcurge linia de comanda cu care a fost lansata aplicatia si, în cazul în care exista, transmite aplicatiei argumentele specificate sub forma unui vector de siruri. Acesta este primit de aplicatie ca parametru al metodei `main`. Reamintim ca formatul metodei `main` din clasa principala este:

```
public static void main ( String args[])
```

Vectorul primit ca parametru de metoda main va contine toate argumentele transmise programului de la linia de comanda. In cazul apelului `java Sort persoane.txt` vectorul `args` va contine un singur element `args[0]="persoane.txt"`.

Numarul argumentelor primite de un program este dat deci de dimensiunea vectorului `args` si acesta poate fi aflat prin intermediul atributului `length` al vectorilor:

```
numarArgumente = args.length ;
```

Spre deosebire ce C/C++ vectorul primit de metoda `main` nu contine pe prima pozitie numele aplicatiei, întrucât în Java numele aplicatiei este numele clasei principale, adica a clasei în care se gaseste metoda main.

Exemplu: afisarea argumentelor primite la linia de comanda

```
public class Echo {
        public static void main (String[] args) {
        for (int i = 0; i < args.length; i++)
                System.out.println(args[i]);
        }
}
```

Un apel de genul `java Echo Drink Hot Java` va produce urmatorul rezultat:

```
Drink
Hot
Java
```

(aplicatia Echo a primit 3 argumente).

Una apel de genul `java Echo "Drink Hot Java"` va produce urmatorul rezultat:

```
Drink Hot Java
```

(aplicatia Echo a primit un singur argument).

Argumente numerice la linia de comanda

Argumentele de la linia de comanda sunt primite sub forma unui vector de siruri (obiecte de tip `String`). In cazul în care unele dintre acestea reprezinta valori numerice ele vor trebui convertite din siruri în numere. Acest lucru se realizeaza cu

metode de tipul **`parseXXX`** aflate în clasa corespunzatoare tipului în care vrem sa facem conversia: `Integer, Float, Double,` etc.
Sa consideram, de exemplu, ca aplicatia `Power` ridica un numar real la o putere întreaga, argumentele fiind trimise de la linia de comanda:

```
java Power "12.1" "3" //ridica 12.1 la puterea 3
```

Conversia celor doua argumente în numere se va face astfel:

```
double numar;
int putere;
numar = Double.parseDouble(args[0]);
putere = Integer.parseInt(args[1]);
```

Metodele de tipul `parseXXX` pot produce exceptii (erori) de tipul `NumberFormatException` în cazul în care sirul primit ca parametru nu reprezinta un numar de tipul respectiv.