Capitolul 1

Introducere in Java

Ce reprezinta Java?

Java este unul dintre cele mai raspandite limbaje de nivel inalt insa acesta nu este principalul merit al sau. Acest limbaj a revolutionat programarea, in multe sensuri pe care le vom detalia in acest curs. Scopul acestei lucrari este de a prezenta la nivel mediu acest limbaj si de a-l utiliza pentru intelegerea conceptelor de structuri de date.

Limbajul de programare Java este acel limbaj in care se pot scrie aplicatii, applet-uri, servlet-uri. Atunci cand un program Java este compilat, codul sursa va fi convertit in cod te tip byte code si anume un limbaj masina ce este portabil pe orice arhitectura CPU. Acest lucru este posibil datorita existentei unei masini virtuale JVM care faciliteaza interpretarea byte code in limbaj masina specific acelei masini pe care va fi programul va fi rulat. Platforma Java, limbajul Java si masina virtuala Java sunt trei lucruri distincte pe care le vom detalia in cele ce urmeaza.

Platforma Java este multimea de clase Java care exista in orice kit de instalare Java. Aceste clase vor putea fi folosite de orice aplicatie Java care ruleaza pe calculatorul unde acestea au fost instalate. Platforma Java se mai numeste mediul Java (Java enviroment) sau kernelul Java API (Application Programing Interface). O alta denumire a aceste multimi de clase este si cea de framework.

Clasele Java sunt grupate in colectii de clase numite pachete. Utilitatea acestora o vom detalia mai tarziu in acest curs. Pachetele sunt de asemenea organizate dupa rolul/functia lor ca de exemplu: pachete de retele, grafica, manipularea interfetelor cu utilizatorul, securitate, etc.

Limbajul de programare Java este limbajul OO (orientat pe obiecte) asemanator cu C++, foarte puternic si usor de folosit si mai ales de invatat de catre programatori. Este rezultatul multor ani de lucru si inglobeaza un design elegant si functionalitati de ultima ora ceea ce il face destul de popular printre programatori. Versatilitatea, flexibilitatea, eficienta si portabilitatea sunt alte aspecte care propulseaza Java inaintea altora.

Pe langa acestea faptul ca programatorii pot creea programe care pot rula in cadrul unor browsere sau web service-uri, sau ca pot creea aplicatii care sa ruleze pe diferite platforme, sau faptul ca pot creea programe ce sa ruleze pe aproape orice dispozitiv electronic mai nou (mobile, aparate medicale, industriale, la distanta etc), fac din acest limbaj unul foarte puternic.

Masina virtuala Java constituie elementul fundamental Java. Programele Java sunt portabile pe orice sistem de operare, arhitectura hardware care suporta un interpretator Java. Sun, firma care realizeaza diverse kituri VM (Virtual Machine), suporta interpretatoare pentru platforme Solaris, Microsoft si Linux. De asemenea au fost creeate interpretatoare si pentru dispozitive ce au ca sisteme de operare Windows CE sau PalmOS.

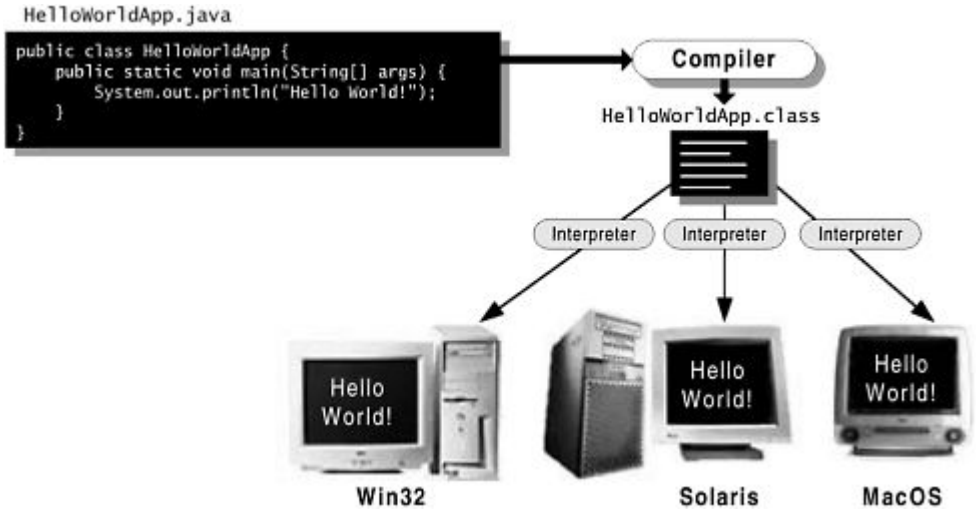

Figura 1. Java poate rula pe orice sistem de operare/arhitectura hardware.

Una din caracteristicile de baza a tehnologiei VM este compilarea *just-in-time (JIT)* unde „byte code”-ul Java este convertit la momentul executiei, in limbaj nativ.

Astfel compilarea are loc doar odata, iar interpretarea ori de cate ori ruleaza programul. Pentru a vizualiza acest lucru in figura 2, avem cele doua evenimente schitate:

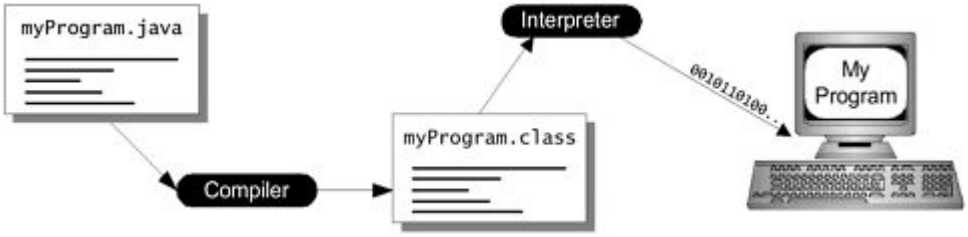

Figura 2. Compilarea si interpretarea unui program Java

Care sunt beneficiile Java?

In cadrul acestei sectiuni vom urmari cateva avantaje ale acestui care incearca sa raspunda la intrebarea fireasca: de ce sa utilizam Java cand avem alte limbaje OOP la dispozitie?

1. *Scris odata va rula oriunde.* Aceasta „lozinca” a firmei Sun este de fapt nucleul conceptual pe care s-a construit platforma Java. Altfel spus odata ce aplicatia a fost scrisa, ea va rula pe orice platforma ce suporta Java, fara a fi nevoie de modificari. Acesta este un avantaj asupra altor limbaje care trebuie rescrise (de cele mai multe ori total) pentru a rula pe alte sisteme de operare.
2. *Securitate.* Platforma permite utilizatorilor sa downloadeze cod prin retea intr-un mediu sigur: codul nesigur nu poate infecta sistemul gazda, nu poate scrie/citi fisiere pe hardisc etc. Aceasta capacitate facea ca Java sa fie unica pana la aparitia altor platforme concurente (.NET).
3. *Programare orientata catre retele.* Alt principu Sun spune ca ”Reteaua este computerul”. Cei care au conceput Java credeau in importanta comunicarii prin retea si au avut in vedere acest fapt: Java faciliteaza folosirea resurselor prin retea si de a creea arhitecturi pe mai multe niveluri.

4. *Programe dinamice.* Programele scrise in Java sunt usor de extins deoarece organizarea este modulara si anume pe clase. Clasele sunt stocate in fisiere separate si incarcate de interpretator ori de cate ori si doar atunci cand este nevoie. Astfel o aplicatie Java apare ca o interactiune intre diverse componente software independente. Aceasta caracteristica este net superioara aplicatiilor ce constau dintr-un cod organizat ca un bloc monolitic.
5. *Performanta.* Masina virtuala Java ruleaza un program interpretand instructiuni portabile byte-code. Aceasta arhitectura inseamna ca programele Java sunt mai lente decat cele C, C++ care sunt compilate folosind cod nativ. Totusi, pentru eficienta, anumite portiuni ale Java, cum ar fi manipularea string-urilor folosesc instructiuni cod nativ. De la versiune la versiune acest neajuns a fost imbunatatit.

Ce poate oferi Java?

Unelte de dezvoltare: aceste unelte reprezinta cam tot de ce are nevoie un programator si anume unelte pentru compilare, rulare, monitorizare, debug, documentare. In principiu se vor folosi *javac* – compilatorul, *java* programul ce ruleaza aplicatiile Java, si *javadoc* pentru documentare.

Application Programming Interface (API): Aceasta reprezinta nucleul de functii Java. Ofera o serie de clase folositoare ce le veti folosi in aplicatii. Acest nucleu este foarte mare, iar pentru a avea o idee despre ce contine avem imaginea de mai jos. In figura 3 sunt prezentate doua concepte JRE (Java Runtime Enviroment) si JDK (Java Development Kit).

JRE ofera librariile, masina virtuala Java si alte componente necesare *rularii* applicatiilor si a applet-urilor. Acest mediu poate fi redistribuit odata cu aplicatiile pentru a le da autonomie.

JDK include JRE si unelte cum ar fi compilatoare sau debuger-e ce sunt utile dezvoltatorilor.

Unelte pentru interfete grafice: Swing si Java 2D sunt folosite pentru crearea de GUI (Graphical User Interfaces) si ofera facilitati pentru un design (si nu numai) frumos al aplicatiilor.

Librarii pentru integrare: De exemplu Java IDL(Interface Definition Language) API, JDBC(Java Database Connectivity), Java Naming and Directory Interface™ ("J.N.D.I.") API, Java Remote Method Invocation etc. Aceste librarii permit accesul la baze de date sau manipularea obiectelor la distanta.

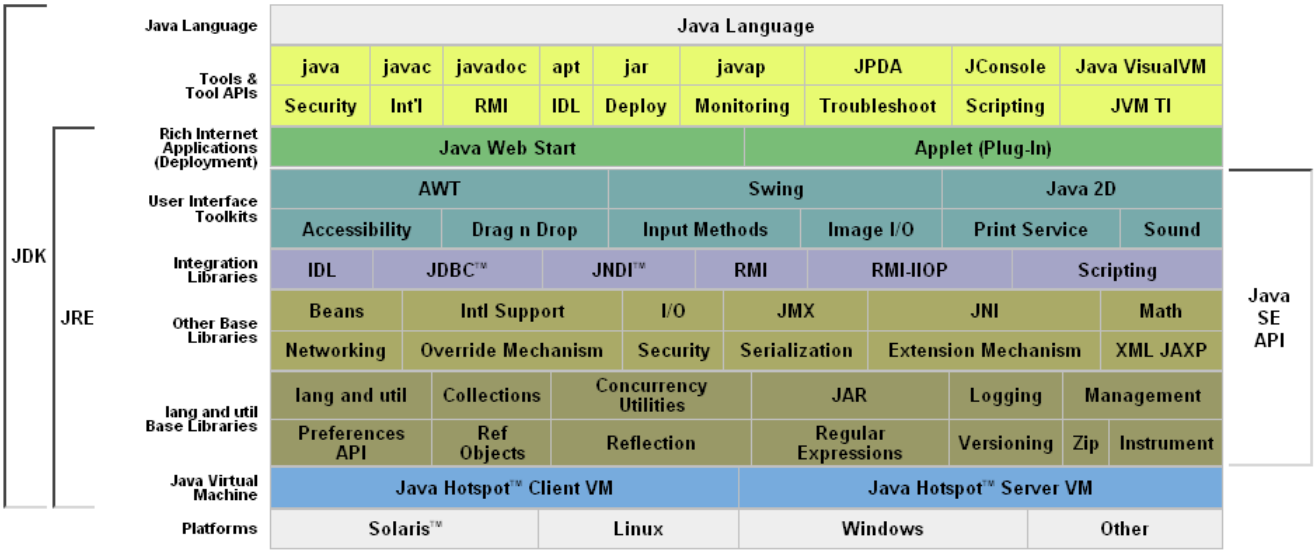

Figura 3. Java Standard Edition

Un prim exemplu

Obtinerea JDK (Java Development Kit)

Inainte de a incepe compilarea sau rularea programelor java, trebuie sa avem acest mediu instalat pe calculatoare. Pentru aceasta trebuie sa instalam JDK de la Sun (exista si alte pachete oferite de alte firme, insa vom folosi pe aceasta pentru ca este gratuit si constituie autoritatea principala a limbajului Java). Acesta poate fi downloadat de pe http://java.sun.com/

Un prim exemplu de program Java

```
/*
      Acesta este primul program
      Ce va fi scris intr-un fisier Example.java
*/
class Example {
      // Orice program Java incepe cu main().
      public static void main(String args[]) {
            System.out.println("Hello Java!");
      }
}
```

Intotdeauna pasii care vor fi urmati in crearea unui program Java vor fi :

1. Scrierea programului.
2. Compilarea acestuia.
3. Rularea programului.

Evident lucrurile se vor complica si pasii se pot detalia foarte mult. In cele ce urmeaza vom descrie pe scurt ce se intampla si cateva indicatii pentru fiecare din cei trei pasi.

Scrierea programului

Pentru editarea unui program Java se pot folosi orice editoare: notepad, wordpad daca lucrati sub Windows, sau joe, nano, vi daca lucrati sub Linux. De asemena se pot folosi IDE(Integrated Development Enviroment) in cazul in care aplicatiile devin complexe.

Pentru majoritatea limbajelor, numele fisierelor ce contin codul sursa, poate fi oricare. Pentru Java *numele fisierului ce contine codul sursa este important*. De exemplu in acest caz ar trebui sa fie *Example.java.* In Java un fisier Java se numeste o *unitate de compilare.* Este un fisier text care contine una sau mai multe definitii de clase. Compilatorul cere ca acest fisier sa aiba extensia *java*. Numele fisierului este Example iar acesta nu este o coincidenta cu numele clasei. In Java totul trebuie sa fie intr-o clasa. Numele clasei publice trebuie sa fie acelasi cu numele fisierului in care acea clasa se afla. Atentie, Java, ca si C, este case sensitive, adica Example este diferit de example sau de eXample.

Compilarea programului

Pentru a compila un program se va executa compilatorul, *javac,* specificand numele fisierului sursa:

C:\>javac Example.java

Compilatorul *javac* creeaza o clasa *Example.class* care contine byte code-ul corespunzator codului sursa din Example.java. Atentie, acest cod nu este executabil direct. El va fi executat de JVM. Pentru a rula programul, va trebui sa utilizam interpretatorul Java, si anume *java:*

C:\>java Example

Atunci cand codul Java este compilat, fiecare clasa individuala este plasata in fisierul numit ca si clasa, folosind extensia *class*. De aceea, este o idee buna sa dam fisierelor sursa acelasi nume ca si clasa pe care o contin. Atunci cand rulam interpretatorul, acesta va cauta fisierul Example care are extensia *.class.* Il va gasi si va executa codul continut in acea clasa.

Detalierea primului program

```
/*
      Acesta este primul program
      Ce va fi scris intr-un fisier Example.java
*/
```

Acesta este un comentariu pe mai multe linii: acesta incepe cu /* si se va termina cu */. Orice caracter dintre aceste doua simboluri va fi ignorat la compilare.

Alt mod de a comenta cod ar fi pe o singura line ca mai jos:

```
int i =0;//Ceea ce urmeaza este un comentariu doar pe aceasta linie
```

Apoi urmeaza :

```
class Example {
```

Aceasta linie foloseste cuvantul *class* marcand inceputul definitie undei noi clase. Clasa este in Java, unitatea de baza pentru incapsulare. Clasa va fi definita in blocul delimitat de „{„ si se va termina cu „}”. Momentan nu vom intra in detalii legate de continutul unei clase, acesta fiind aspectul urmatorului capitol din curs.
Urmatorul rand din program este un comentariu pe o singura linie (cu referire la functia ce va urma).
Urmatorul rand este metoda main sau entry point:

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

Se numeste astfel pentru ca reprezinta functia care este prima executata atunci cand interpretatorul java ruleaza programul. Toate aplicatiile Java incep executia cu apelul functiei *main().*
Cuvantul cheie *public* este un specificator de acces. Cuvintele cheie ale unui limbaj reprezinta acele cuvinte care „alcatuiesc” limbajul si definesc divesele instructiuni sau operatori sau tipuri de data, etc.
Un *specificator de acces* determina modul in care alte clase din program pot accesa membrii unei clase.
Membrul unei clase poate fi considerat o variabila sau o functie de exemplu.
Cuvantul cheie *static* permite ca apelul functiei *main*() sa fie facut fara a fi nevoie de instantierea clasei din care aceasta functie face parte.
Cuvantul cheie *void* specifica tipul de date returnat de functia main(), in cazul acesta nimic, adica spune compilatorului ca functia nu va returna nici o valoare.
Intre parantezele functiei main() se afla lista de parametrii, in cazul de fata *String args[]*.
Aceasta declaratie *String args[]* inseamna colectia de obiecte de tip String(sir de caractere).
Ultimul caracter al acestei linii este „{” si anume faptul ca incepe corpul functiei *main*. Corpul functiei se termina cu „}” asemanator ca si „corpul” clasei.

Urmatoarea parte din cod este:

```
      System.out.println("Hello Java!");
```

Aceasta are ca efect – la rulare – afisarea mesajului „Hello Java!”
Afisarea are loc datorita functiei gata definite, sau predefinite, *println()*. Linia totusi incepe cu System.out...
In momentul acesta vom spune ca *System* este o clasa predefinita care ofera acces la sistem, si *out* este stream-ul de iesire conectat la consola. Practic *System.out* este un obiect care incapsuleaza iesirea la consola. Consola nu este des folosita in programele reale Java, sau in applet-uri, dar este utila la procesul de invatare Java.

Erori de sintaxa, erori de runtime

Atunci cand scrieti un cod sursa, mai ales daca sunteti la inceput este posibil sa omiteti din neatentie sau nestiinta anumite portiuni din cod. Din fericire, daca ceva nu este corect, compilatorul va raporta aceste probleme ca erori de sintaxa.Compilatorul va incerca sa „dea un sens” codului sursa, indiferent in ce consta acesta. De aceea, eroarea raportata nu reflecta de cele mai multe ori cauza problemei!

De exemplu daca omitem caracterul „{„ din

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

La compilare va apare urmatorul mesaj:

```
Example.java: 7 ';' expected

Public static void main(String[] args)

Example.java:10: class, interface or enum expected

}

^

2 errors
```

Evident ceva nu este in regula cu mesajul pentru ca nu lipseste „;” ci „{„. Ideea este ca atunci cand programul contine o eroare de sintaxa, mesajul nu trebuie interpretat cuvant cu cuvant. Trebuie sa ne uitam la contextul in care apare eroarea, la liniile din jurul locului unde este indicata eroarea si sa deducem care este cauza reala a acesteia.

Refacand codul si greseala de dinainte, sa inlocuim afisarea mesajului cu acest cod:

```
System.out.println(args[1]);
```

Ce realizeaza acest cod? Afisarea celui de al doilea argument transmis programului: argumentele sunt un sir care va fi scris la rularea programului. Compilam si rulam:

```
java Example
```

Urmatoarea eroare va apare:

```
Exception in thread „main” java.lang.ArrayIndexOutOfBoundException : 1 at
Example.main<Example.java:8>
```

Acest tip de exceptie apare in momentul rularii programului (de aceea se numeste runtime error) si se datoreaza unor situatii neprevazute de programator. In cazul de fata se presupune ca avem argumente la rulare, cand realitatea este ca sirul de argumente este gol. Accesarea argumentului cu numarul 1 este astfel o greseala! Vom detalia tot intelesul acestei erori la momentul cuvenit.

Bazele limbajului

Inainte de a incepe sa exploram sintaxa Java, vom studia mai intai in mare alcatuirea unui program Java.

Java consta din una sau mai multe unitati de compilare, adica fisiere ce contin clase. Fiecare unitate poate incepe cu un cuvant cheie optional *package*, urmata eventual de declaratii *import*. Acestea, asemanator directivei de *include* din C permit folosirea claselor din alte pachete. Vom discuta mai tarziu despre aceste aspecte.

Apoi urmeaza definirea uneia sau mai multor *clase* sau *interfete*, dar mai nou si a structurilor *enum.*

In cadrul claselor putem defini campuri(variabile), metode sau constructori. Toate acestea se numesc membrii clasei. Majoritatea metodelor vor contine instructiuni ce includ expresii, operatori, tipuri de data etc. Abordarea in continuare va fi exact de la unitatile de baza catre pachete in final. Pachetele contin clase, clasele contin metode. Aceasta relatie de incluziunie este marcata prin punct. Diferenta este ca in cazul pacheteleor „.” Semnifica o relatie de incluziune a directoarelor/folderelor iar in cazul claselor/metodelor incluziunea are lor in acelasi fisier. Vom insista asupra acestui aspect in capitolul 2.

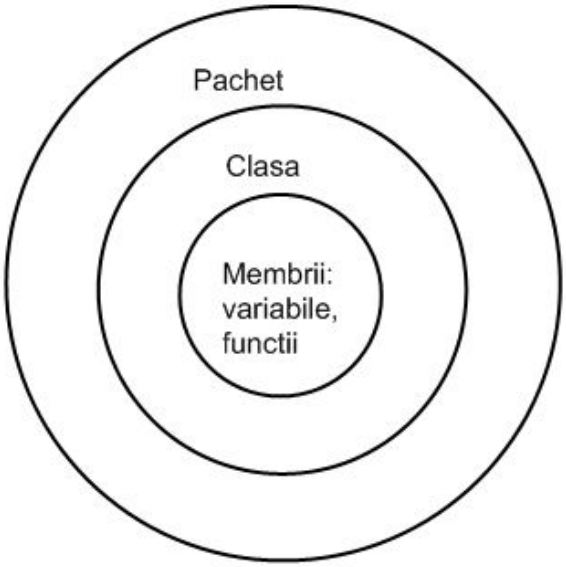

Figura 4. Organizarea programelor Java

Variabile

Variabila este un element de un anumit tip de data identificat printr-un nume, utilizat la stocarea datelor.

Numele variabilei denumit si identificator este compus din o serie de caractere ce incepe cu o litera. Tipul de data al variabilei determina ce valori poate lua acea variabila.

Declaratia va fi de forma : *tip nume*

Pe langa nume si tip, o variabila mai are si un domeniu de vizibilitate. Sectiunea de cod unde acea variabila poate fi folosita se numeste domeniu de vizibilitate al variabilei.

Sa analizam codul de mai jos pentru o intelegere mai buna acelor descrise mai sus:

```
public class Variabile{

      public static void main(String args[])

      // integer

      byte largestByte;

      largestByte = Byte.MAX_VALUE;

      int largestInteger = Integer.MAX_VALUE;

      // real

      float largestFloat = Float.MAX_VALUE;

      double largestDouble = Double.MAX_VALUE;

      // afisarea

      System.out.println("Valoarea maxima byte este " + largestByte);

      System.out.println("Valoarea maxima integer value este " + largestInteger);

      System.out.println("Valoarea maxima float value este " + largestFloat);

      System.out.println("Valoarea maxima double value este " + largestDouble);

      }

}
```

Clasa Variable contine o metoda *main()* si nici o variabila. Da, este corect, nici o variabila, deoarece domeniul in care cele patru variabile sunt vizibile, este cel al functiei main().

Sa luam declaratia variabilei `byte largestByte;`

Prin aceasta instructiune se declara o variabila numita *largestByte* care are tipul de data *byte*. Acest tip de data este o valoare intreaga cu semn cuprinsa intre -128 si 127. Vom detalia in tabelul 1 tipurile primitive de date. Java contine doua categorii de tipuri de date: referinta si primitive. O variabila de tip primitiv poate contine doar o singura valoare conform cu acel tip si avand formatul acelui tip. Evident tipurile referinta sunt mai complexe si putem spune ca inglobeaza si tipuri primitive.

Tabelul 1. Tipurile primitive

	Cuvant cheie	Descriere	Marime
Integer	byte	Intreg de lungime byte	8-bit cu semn
	short	Intreg short	16-bit cu semn
	int	Intreg	32-bit cu semn
	long	Intreg short	64-bit cu semn
Real	float	Real, virgula mobila cu o singura precizie	32-bit format IEEE 754
	double	Real, virgula mobila cu o dubla precizie	64-bit format IEEE 754
Alte tipuri	char	Un singur caracter Unicode	16-bit caracater Unicode
	boolean	Valoarea boolean-a (true sau false)	8-bit/1-bit (8 bits de spatiu, 1 bit de data)

Clasele, sirurile de data si interfetele sunt tipuri de data referinta. Valoarea unei variabile de tip referinta este adresa unui obiect. O referinta este denumita si obiect, sau adresa unei memorii in alte limbaje, insa in Java nu avem posibilitatea de a accesa ca in C zona de memorie direct.

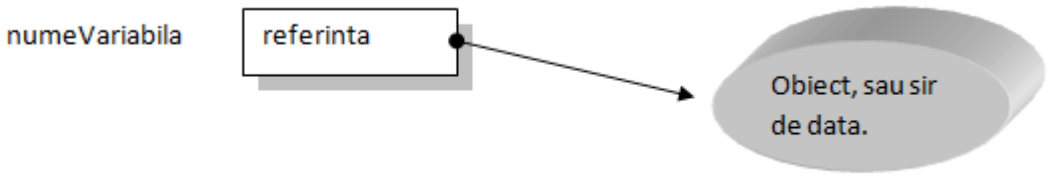

Figura 5. Variabila de tip referinta contine adresa obiectului

Conditii de numire a variabilelor

Pentru ca numele unei variabile sa fie valid trebuie sa indeplineasca urmatoarele conditii:

1. Trebuie sa inceapa cu o litera si sa fie alcatuit din caractere Unicode
2. Nu trebuie sa fie un cuvant cheie.
3. Trebuie sa fie unic in domeniul de vizibilitate.

Domeniul de vizibilitate

Domeniul de vizibilitate al unei variabile este regiunea din program in care se poate face referire la acea variabila. Alt scop al domeniului este de a determina cand sistemul creeaza sau distruge memoria alocata pentru o variabila.

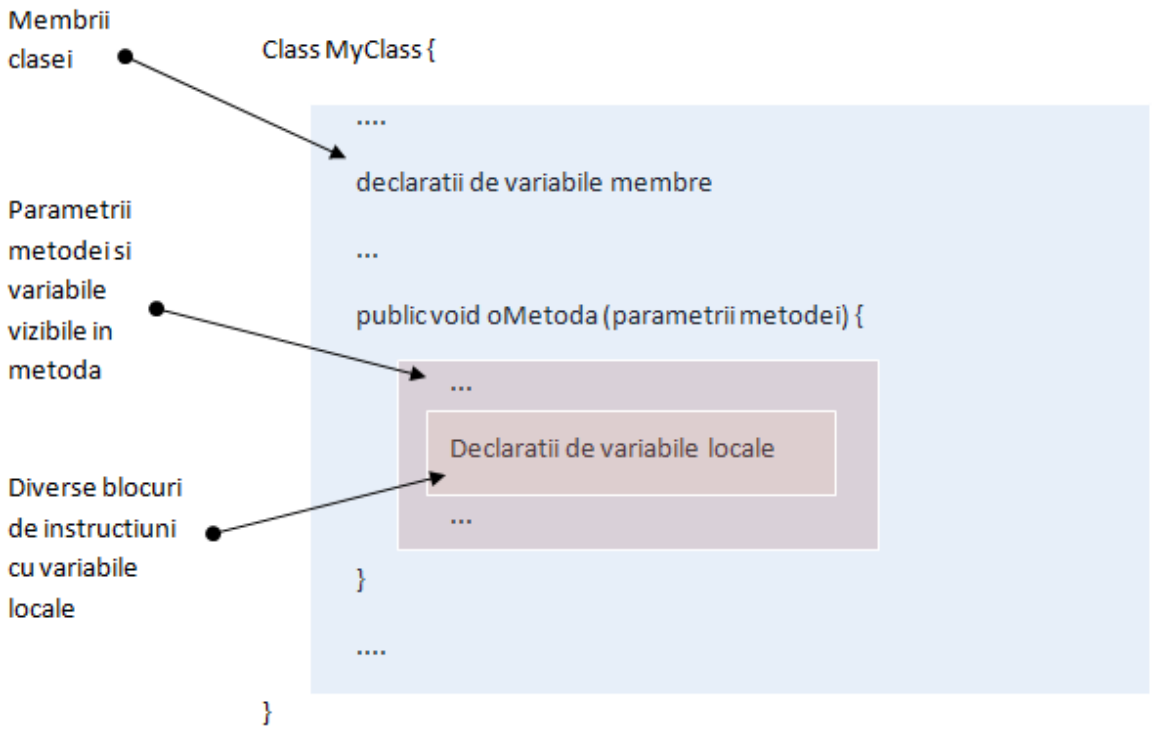

Figura 6. Diverse categorii de domenii de vizibilitate ale variabilelor

Initializarea variabilelor

Variabilele locale si membre trebuie initializate inainte de a fi folosite. Initializarea inseamna atribuirea de valori acestora:

```
int largestInteger = Integer.MAX_VALUE;
```

sau

```
int largestInteger = 23;
```

Cuvantul cheie *final* in cazul variabilelor semnifica faptul ca o variabila nu mai poate fi modificata odata ce a fost initializata, fiind practic o constanta.

```
final int largestInteger = 0;
```

Operatori

Un operator indeplineste o functie, pentru unul doi sau trei operanzi. Un operator care necesita doar un operand este denumit *unar*. De exemplu ++ este operator unar care incrementeaza valoarea operandului cu 1. Un operator care necesita doi operanzi se numeste *binar*. De exemplu = este un operator ce asigneaza o valoare operandului din stanga, valoarea fiind a operandului din dreapta. Un operator cu trei operanzi ar fi ?: si vom vorbi imediat despre el.

Operatorii unari pot avea notatie prefixata ,infixata sau postfixanta:

operator op - notatie prefixata

op operator – notatie postfixata

op1 operator op2 – notatie infixata

In cele ce urmeaza vom vorbi de diversele categorii de operatori exemplificand utilizarea lor.

Operatori aritmetici

Acestia realizeaza operatiunile matematice de baza dupa tabelul de mai jos

Tabelul 2. Operatori aritmetici

Operator	Folosire	Descriere
+	op1 + op2	Adauga op1 and op2; de asemenea concateneaza stringuri
-	op1 - op2	scade op2 din op1
*	op1 * op2	inmulteste op1 cu op2
/	op1 / op2	divide op1 la op2
%	op1 % op2	Calculeaza restul impartirii lui op1 la op2

Pe langa acesti operatori mai exista si operatori unari aritmetici:

Tabelul 3. Operatori aritmetici unari

Operator	Folosire	Descriere
+	+op	Transforma op in int daca este byte, short sau char
-	-op	Aplica negatia aritmetica asupra lui op

Alti operatori aritmetici sunt ++ care incrementeaza cu 1 operandul si -- care decrementeaza cu 1 operandul:

Tabelul 4. Operatori de incrementare/decrementare

Operator	Folosire	Descriere
++	op++	incrementeaza op cu 1; evaluarea lui op inainte de incrementare
++	++op	incrementeaza op cu 1; evaluarea lui op dupa incrementare
--	op--	decrementeaza op cu 1; evaluarea lui op inainte de decrementare
--	--op	decrementeaza op cu 1; evaluarea lui op dupa decrementare

Operatori relationali

Acestia compara doua valori si determina relatia dintre ele.

Tabelul 5. Operatori relationali

Operator	Folosire	Descriere
>	op1 > op2	Returneaza true daca op1 este mai mare decat op2
>=	op1 >= op2	Returneaza true daca op1 este mai mare sau egal cu op2
<	op1 < op2	Returneaza true daca op1 este mai mic decat op2
<=	op1 <= op2	Returneaza true daca op1 este mai mic sau egal cu op2
==	op1 == op2	Returneaza true daca op1 este egal cu op2
!=	op1 != op2	Returneaza true daca op1 este diferit de op2

Operatori conditionali

Acestia returneaza o valoare true sau false evaluand operanzii aferenti.

Tabelul 6. Operatori conditionali

Operator	Folosire	Descriere
&&	op1 && op2	Returneaza true daca op1 and op2 sunt true; evalueaza conditional op2
||	op1 || op2	Returneaza true daca op1 sau op2 este true; evalueaza conditional op2
!	!op	Returneaza true daca op este false
&	op1 & op2	Returneaza true daca op1 si op2 sunt boolean si amandoi sunt true; evalueaza intotdeauna op1 si op2 Daca ambii operanzi sunt numere, efectueaza SI pe biti
|	op1 | op2	Returneaza true daca ambii op1 si op2 sunt de tip boolean, si atat op1 cat si op2 sunt true; evalueaza intotdeauna op1 si op2 Daca ambii operanzi sunt numere, efectueaza SAU pe biti
^	op1 ^ op2	Returneaza true daca op1 si op2 sunt diferiti, adica un XOR pe biti

Operatori de siftare

Tabelul 6. Operatori de siftare

Operator	Folosire	Descriere
<<	op1 << op2	Siftarea bitilor lui op1 la stanga cu o lungime data de op2; umple cu zero bitii din partea dreapta
>>	op1 >> op2	Siftarea bitilor lui op1 la dreapta cu o lungime data de op2; umple cu bitul cel mai semnificativ (de semn) bitii din partea stanga
>>>	op1 >>> op2	Siftarea bitilor lui op1 la dreapta cu o lungime data de op2; umple cu zero bitii din partea stanga

Operatori de asignare

Operatorul = este cel implicit de asignare. De exemplu daca dorim sa adaugam o valoare la o variabila putem scrie

```
i = i+2;
```

De asemenea putem prescurta aceasta operatie astfel:

```
i += 2;
```

Acest lucru este valabil pentru diverse operatii:

Tabelul 7. Operatori de asignare

	Operator	Folosire	Echivalent
Scurtaturi pentru operatorii aritmetici	+=	op1 += op2	op1 = op1 + op2
	-=	op1 -= op2	op1 = op1 - op2
	*=	op1 *= op2	op1 = op1 * op2
	/=	op1 /= op2	op1 = op1 / op2
	%=	op1 %= op2	op1 = op1 % op2
Scurtaturi pentru operatorii pe biti	&=	op1 &= op2	op1 = op1 & op2
	|=	op1 |= op2	op1 = op1 | op2
	^=	op1 ^= op2	op1 = op1 ^ op2
Scurtaturi pentru siftare	<<=	op1 <<= op2	op1 = op1 << op2
	>>=	op1 >>= op2	op1 = op1 >> op2
	>>>=	op1 >>>= op2	op1 = op1 >>> op2

Alti operatori

Vom aminti aici si o lista cu operatorii care nu au fost inclusi in celelalte liste

Tabelul 8. Alti operatori

Operator	Descriere
?:	Scurtatura la o anumita instructiune if-else
[]	Folosit la lucrul cu siruri
.	Pentru accesarea membrilor, a claselor.
(params)	Lista de parametrii (unei metode, sau instructiuni)
(tip)	Operator cast: schimba tipul de data al unei variabile
New	Creeaza un nou obiect sau sir.
Instanceof	Determina daca primul operand este instanta celui de-al doilea operand

Precedenta operatorilor

Cand folosim operatorii in combinatie cu operanzi obtinem expresii. Expresiile ajuta la calcularea si asignarea de valori variabilelor si ajuta la contorul fluxului logic al unui program.

Expresia este o serie de variabile, operatori, apeluri care rezulta in final intr-o singura valoare. Exemple de expresii:

```
x * y * z
x+y/10          //ambigua
(x+y)/100  // clara, asa este recomandat
```

Atunci cand se va face evaluarea expresiilor se va tine cont ce operator va fi prima data evaluat:

Tabelul 9. Precedenta operatorilor

Precedenta cea mai mare	Operatori postfixati	[] . (params) expr ++ expr --
	Operatori unari	++expr --expr +expr -expr ~ !
	Operatori creeare sau cast	new (type)expr
	Operatori de multiplicare	* / %
	Operatori aditivi	+ -
	Operatori de siftare	> >>>
	Operatori relationali	< > <= >= instanceof
	Operatori de egalitate	== !=
	SI pe biti	&
	XOR pe biti	^
	SAU pe biti	|
	SI logic	&&
	SAU logic	||
	Scurtatura if-else	?:
Precedenta cea mai mica	asignare	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Instructiuni de control

Bloc de instructiuni

Un bloc de instructiuni este cuprins intre { si } si poate fi privit ca o singura instructiune. De exemplu:

```
if (i>0)
{
      System.out.println(”i este pozitiv”);
}
else
{
      System.out.println(”i este negativ”);
}
```

Acest cod poate fi scris si asa:

```
if (i>0)
      System.out.println(”i este pozitiv”);
else
      System.out.println(”i este negativ”);
```

Atunci la ce sunt folositoare acoladele?

Daca de exemplu dorim ca atunci cand *i* este mai mare ca zero, nu numai sa afisam un mesaj despre asta dar sa decrementam aceasta valoare (motivatii pot fi multe) atunci trebuie sa facem intr-o instructiune compusa:

```
if (i>0)
{
      System.out.println(”i este pozitiv”);
      i--;
}
else
{
      System.out.println(”i este deja negativ”);
}
```

In cele ce urmeaza vom descrie instructiunile de baza pentru controlul logicii unui program:

Tabelul 10. Instructiuni de control

Tipul de instructiune	Cuvinte cheie
Bucla	while, do-while, for
De decizie	if-else, switch-case
Tratarea erorilor	try-catch-finally, throw
Ramificare	break, continue, label :, return

While

Forma generala a instructiunii *while* este:

```
While (expresie) {

... lista de instructiuni

 }

... instructiuni dupa blocul while
```

Prima data se evalueaza expresia dintre paranteze care trebuie sa returneze o valoare booleana. Daca ea returneaza *true* atunci se executa lista de instructiuni din corpul *while.* Se revine la evaluarea expresiei, daca ea returneaza true se executa a doua oara lista de instructiuni pana cand expresia va returna false. In acel moment se „sare" peste corpul de instructiuni din while la prima instructiune ce urmeaza dupa blocul while.

Exemplu:

```
int i=0;

      while (i<10)

      {

            System.out.println(i++);

      }

      System.out.println("Dupa while");
```

Se vor executa afisarile 0 pana la 9 si la sfarsit mesajul „Dupa while".

Instructiunea *do while* are aproape acelasi efect singura diferenta este ca evaluarea expresiei se va face la sfarsitul blocului de instructiuni, astfel ca acele instructiuni se vor efectua cel putin odata:

```
do {
    … lista de instructiuni
} while (expresie);
```

For

Ca si while permite executarea instructiunilor in bucla. In plus permite iterarea unor variabile intr-un mod compact:

```
for (initializari; conditie de terminare; incrementare) {
       ...lista de instructiuni
}
```

Initializari reprezinta secventa de instructiuni care se executa la intrarea in bucla; se executa doar odata la inceput.

Conditie de terminare reprezinta conditia care determina iesirea din bucla. Incrementare este o instructiune sau un set de instructiuni care este efectuat la fiecare iteratie prin bucla.

Exemplu:

```
for (int i=0,j=1; (i<10 || j<16);i++, j+=2)
      {
            System.out.println(i+" " + j);
      }
System.out.println("Dupa for");
```

Se vor afisa in paralel valorile lui *i* pana la 9 si lui *j* pana la 19.

If – else

Aceasta instructiune permite executarea unor instructiuni doar in anumite cazuri: in acest fel se poate separa logica programului. In cea mai simpla forma instructiunea coditie poate fi scrisa asa:

```
if (expresie) {
... lista instructiuni
}
```

Daca expresia va fi evaluata cu valoarea *true*, atunci lista de instructiuni este executata, altfel se sare peste. Ce se intampla daca insa conditia este *false* si doar in acel caz vrem sa executam alte instructiuni? In acest caz avem instructiunea compusa if else de exemplu:

```
if (response == OK) {
    // code to perform OK action
} else {
    // code to perform Cancel action
}
```

Mai departe, sa cosideram ca avem urmatorul scenariu: avem o variabila care poate lua valori de la 1 la 12 si in functie de valoarea ei de la un moment dat vrem sa afisam luna calendaristica, corespunzatoare ei.

```
int month = 8;
if (month == 1)
      System.out.println("Ianuarie");
else if (month == 2)
      System.out.println("Februarie");
else if (month == 3)
      System.out.println("Martie");
else if (month == 4)
      System.out.println("Aprilie");
else if (month == 5)
      System.out.println("Mai");
else if (month == 6)
      System.out.println("Iunie");
else if (month == 7)
      System.out.println("Iulie");
else if (month == 8)
      System.out.println("August");
else if (month == 9)
      System.out.println("Septembrie");
else if (month == 10)
      System.out.println("Octombrie");
else if (month == 11)
      System.out.println("Noiembrie");
else if (month == 12)
      System.out.println("Decembrie");
```

Problema este ca pentru a evalua care din instructiuni trebuie executate se fac multe verificari, degeaba.Alta instructiune care realizeaza acelasi lucru mai eficient este switch

Switch

```
int month = 8;
 switch (month) {
     case 1:  System.out.println("Ianuarie "); break;
     case 2:  System.out.println("Februarie "); break;
     case 3:  System.out.println("Martie "); break;
     case 4:  System.out.println("Aprilie"); break;
     case 5:  System.out.println("Mai "); break;
     case 6:  System.out.println("Iunie "); break;
     case 7:  System.out.println("Iulie "); break;
     case 8:  System.out.println("August"); break;
     case 9:  System.out.println("Septembrie "); break;
     case 10: System.out.println("Octombrie "); break;
     case 11: System.out.println("Noiembrie "); break;
     case 12: System.out.println("Decembrie "); break;
 }
```

In cazul instructiunii *switch* se va extrage valoarea variabilei sau expresiei dintre paranteze, in cazul de fata *month* si in functie de valoarea acelei expresii se va executa una din ramificatiile *case.* Dupa ce se executa instructiunea din dreptul case, se va executa si instructiunea break, altfel se vor efectua si celelate instructiuni din switch care urmeaza.

Spre exemplu:

```
int month = 8;
        switch (month) {
            case 1:  System.out.println("Ianuarie "); break;
            case 2:  System.out.println("Februarie "); break;
            case 3:  System.out.println("Martie "); break;
            case 4:  System.out.println("Aprilie"); break;
            case 5:  System.out.println("Mai "); break;
            case 6:  System.out.println("Iunie "); break;
            case 7:  System.out.println("Iulie "); break;
            case 8:  System.out.println("August");
            case 9:  System.out.println("Septembrie ");
            case 10: System.out.println("Octombrie ");
            case 11: System.out.println("Noiembrie ");break;
            case 12: System.out.println("Decembrie ");
        }
```

Va afisa August dar si lunile ce urmeaza dupa, pana la intalnirea unui break si anume Noiembrie(inclusiv).