

Tipuri primitive și IO pe intrarea sistem standard

1. Scopul lucrării

Obiectivele acestei sesiuni de laborator sunt:

- Înțelegerea și exersarea modului de construire și de execuție a unui program Java fără a folosi un mediu de programare
- Cunoașterea elementelor unui stil de programare bun
- Înțelegerea și exersarea lucrului cu tipurile primitive și clasee învelitoare
- Folosirea câtorva metode de I/E simple

2. Compilarea unui program Java

Presupunând că se folosește JDK (Java Development Kit) de la Sun Microsystems, pașii sunt următorii:

1. **Creați programul sursă cu un editor de texte** (d.e., Notepad, jEdit, TextPad, ...). Salvați-l într-un fișier cu același nume precum cel al clasei pe care o conține și adăugați-i extensia ".java" (d.e., Salut.java). Una dintre erorile uzuale este folosirea unui alt nume pentru fișier decât cel al clasei. Numele dinainte de "." trebuie să fie identic cu cel al clasei, inclusiv tipul de literă (mare sau mică). Mulți programatori își salvează sursele la fiecare 10 minute – se face repede și ne scapă de pericolul de a re-tasta totul dacă sistemul nu mai funcționează.
2. Deschideți o fereastră de comenzi DOS și navigați (cu **cd**) spre directorul care conține sursa. Aceasta se face ușor dacă aveți nume de directoare scurte și fără spații.
3. **Compilați** programul sursă (**Salut.java** în acest exemplu) folosind comanda:

```
javac Salut.java
```

Aceasta va produce una sau mai multe fișiere ".class", care sunt în formatul obiect (Java byte code) a programelor Java. Puteți produce și un fișier ".exe" din acesta, dar asta nu se face în mod normal.

4. **Rulați** aplicația cu:

```
java Salut
```

Această comandă va încărca fișierul **Salut.class** și toate clasele necesare. Execuția începe cu metoda **main** din clasa Salut. Continuați acest ciclu până când programul funcționează.

Notă. Sursele Java au extensia **.java**. Codul compilat are extensia **.class**.

3.1. Cum se localizează programele

Adesea programele Java simple se compilează și rulează folosind comenzi precum:

```
javac MyProgram.java  
java MyProgram
```

Sau se poate face acest lucru folosind un IDE. În orice caz, Java trebuie să știe atât pentru compilarea cât și pentru execuția programelor unde să găsească clasele de bibliotecă folosite. Java (cel puțin după

versiunea 1) știe cum să-și localizeze clasele proprii, dar trebuie să îi comunicați unde să găsească alte biblioteci care le folosiți. Directoarele standard Microsoft Windows nu sunt incluse în căutare. S-ar putea să trebuiască să specificați o listă de directoare sau fișiere **.jar** unde se pot găsi programele, prin setarea *variabilei de mediu* CLASSPATH.

3.2. Organizarea lucrului

Ori de câte ori începeți un **proiect nou**, creați un **nou director** pentru fișierele sursă. Numele directorului trebuie să fie din litere mici, fără spații sau alte semne de punctuație.

Mai multe clase în programele de dimensiuni mai mari sunt grupate de obicei în *packages* (pachete). După declarația opțională *package*, pot exista instrucțiuni *import*, care vă permit să specificați clase care pot fi referite fără a le califica prin numele pachetului.

Packages sunt directoare / cataloage care conțin clasele Java și constituie o modalitate de grupare a claselor înrudite. Pentru programele mici, este uzual să se omită specificarea pachetului (Java creează ceea ce numește un pachet *default* (implicit) în acest caz.).

O clasă per fișier

Puneți fiecare clasă în propriul său fișier sursă, separat. Fiecare fișier sursă trebuie numit *exact* la fel cu clasa, plus sufixul ".java". Spre exemplu, dacă clasa se numește "Test", fișierul trebuie să fie numit "Test.java" (nu "test.java").

Se pot pune mai multe clase într-un fișier și totul să funcționeze. Dar acest lucru nu este folositor la programe mai mari. Mediile de dezvoltare interactive (cum este NetBeans) cer ca fiecare clasă să fie într-un fișier sursă separat, lucru cerut și de alte medii.

3. Stilul de programare

Cele ce urmează se bazează pe articolul "Good Java Style" de Thornton Rose.

Câteva motive pentru a folosi un stil bun [din "Java Code Conventions," Sun Microsystems]:

- 80% din costurile implicate de un produs pe durata lui de viață sunt merg la întreținere.
- Foarte rar software este întreținut pe întreaga sa durată de viață de către autorii originali.
- Folosirea unui stil bun îmbunătățește capacitatea de a întreține codul produsului.
- Dacă se livrează cu produsul, codul sursă trebuie să fie la fel de bine împachetat, curat și profesional ca și restul produsului.

Scrierea de cod cu un stil bun oferă următoarele beneficii:

- Îmbunătățește lizibilitatea, consistența și omogenitatea codului, ceea ce îl face mai ușor de înțeles și întreținut.
- Face codul mai ușor de trasat și depanat, pentru că este clar și consistent.
- Permite continuarea mai ușoară a dezvoltării din locul unde Dvs. Sau un alt programator s-a oprit, în special după o lungă perioadă de timp.
- Crește beneficiile parcurgerii codului, deoarece participanții se pot focaliza mai mult asupra aspectelor de interes (ce face codul respectiv).

Linii generale de ghidare

Scrierea de cod Java folosind un stil bun nu este dificilă, dar necesită atenție la detalii. Iată câteva linii generale de urmărit:

- Faceți codul clar și ușor de citit.
- Faceți codul consistent.

- Folosiți nume evidente pentru identificatori.
- Organizați-vă logic fișierele și clasele.
- Stocați o clasă pe fișier (aici nu se numără clasele interne - inner classes).
- Folosiți cel mult 80-90 caractere pe linie.
- Folosiți judicios spațiile albe și/sau alți separatori.
- Folosiți spațiile în locul tabulatorilor la indentare (schimbarea dimensiunii tabulatorilor nu va afecta atunci aspectul codului scris) .

Acoladele și indentarea

Stilul de indentare sau plasarea acoladelor ("{" și "}") indentarea asociată codului constituie una dintre celelalte probleme legate de scrierea codului. Există câteva stiluri de indentare comune limbajelor de stil C, cum este Java. Multe favorizează stilul K&R cu acolada deschisă pe linia instrucțiunii a căreia îi aparține logic blocul și acolada închisă la același nivel de indentare cu instrucțiunea respectivă.

Stilul de comentarea a programelor este și el parte a stilului de programare.

3.3. Comentarii în Java

Programele sunt citite și de calculatoare și de oameni. Instrucțiunile le scrieți pentru a spune calculatorului ce să facă. Trebuie însă să scrieți și comentarii pentru a explica oamenilor ce face programul. Desigur, Java nu le înțelege pentru că ele sunt scrise în limbaj natural.

Java ignoră toate comentariile. Totuși, există un program numit **javadoc** care citește anumite tipuri de comentarii și produce documentație în format HTML.

Folosiți spații și linii goale în program. Una dintre cele mai eficiente modalități pentru a face un program lizibil este să puneți spații în punctele cheie. Sunt câteva stiluri pentru a face asta. Chiar mai important este să puneți linii goale în program. Acestea vor separa secțiunile de cod. Ar trebui să fie o linie goală între fiecare dintre grupurile de instrucțiuni care sunt grupate logic.

Există câteva feluri de comentarii:

// comentarii – de o singură linie

După două caractere //, Java ignoră tot ce mai există până la sfârșitul liniei respective. Acesta este cel mai comun tip de comentariu.

```
//--- variabile locale ---
int nArticole; // numărul de articole.
int nInTermen; // numără câte sunt în termenul de garanție.
```

/* ... */ comentarii – de mai multe linii

După caracterele /*, Java va ignora totul până găsește o pereche */. Acest fel de comentariu, pe care îl știți din C, se poate întinde pe mai multe linii și se folosește de obicei pentru a "comenta" secțiuni de cod – comentând, de exemplu, porțiuni de cod în timpul depanării unui program. Spre exemplu,

```
/* Folosiți comentarii pentru a descrie variabilele sau secțiunile de program.
Ele sunt foarte utile tuturor persoanelor care citesc programele:
In primul rând Dvs., apoi profesorilor, apoi șeful etc., dar în primul rând Dvs.!
*/
```

Comentarii javadoc

Comentariile care încep cu /** sunt folosite de programul **javadoc** pentru a produce documentație în formatul HTML pentru program. Documentația Java de la Sun Microsystems este produsă folosind **javadoc**. Este esențial să folosiți acest fel de comentariu pentru programele mari. Vă recomandăm cu tărie să folosiți acest fel de comentarii pentru a promova reutilizarea codului scris de Dvs.

Cele mai bune practici referitor la comentarii:

- Nu scrieți comentarii pentru a documenta lucruri evidente. Presupuneți că cititorul știe Java.
- Fiecare comentariu are potențialul de a crea inconsistențe între ceea ce spune și ce face codul. Una dintre cauzele problemelor din software este că se schimbă codul în timp, dar comentariile nu se actualizează. Pentru a evita această situație, țineți comentariile în apropierea codului pe care îl documentează astfel încât să fie mai ușor de sincronizat.

Numele de identificatori

Alegerea corectă a numelor pentru lucruri este foarte importantă.

Caractere permise

Fiecare nume de identificator este compus din următoarele caractere, începând cu o literă:

- Litere: a-z, A-Z (și alte caractere alfabetice din alte limbi)
- Cifre: 0-9
- Special: _ (subliniere – [underscore])

Nici un nume nu poate fi la fel cu un cuvânt cheie Java. Cuvintele cheie din Java sunt:

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Exemple

apple	Este un nume legal. Toate minuscule implică faptul că este o variabilă sau o metodă .
Apple	Alt nume legal. Majuscula de la început înseamnă ca este o clasă sau o interfață .
APPLE	Alt nume legal. Toate majuscule semnifică o constantă .
opleft	Legal; mai multe cuvinte trebuie scrise în stil "cămilă".
top_left	Mai bun, dar stilul "cămilă" este preferat lui _ în Java.
topLeft	În stil Java bun
top left	ILEGAL – nu se admit spații
import	ILEGAL – este cuvânt Java

Folosirea majusculor, minusculelor și a amestecului de litere

Convențiile de folosire a tipurilor de litere nu este impus de compilatoare, dar este respectat de foarte multă lume. Stilul "*cămilă*" este preferat în practică la identificatorii compusi din mai multe cuvinte.

Numele de interfețe și clase – încep cu literă mare

Numele de interfețe și clase încep cu literă mare și continua cu litere mici. Pentru cuvinte multiple folosiți stilul cămilă. Exemple: Direction, LogicalLayout, DebugGapSpacer.

Nume de metode și de variabile – încep cu literă mică

Minusculele sunt folosite pentru nume de variabile și metode. La cuvinte multiple, folosiți stilul cămilă. Exemple: top, width, topLeft, roomWidth, incomeAfterTaxes.

Constante – doar majuscule, folosiți `_` pentru separarea cuvintelor

Numele constantelor (tipic declarate *static final*) trebuie să fie toate din litere mari. De exemplu, BorderLayout.NORTH. Atunci când o constantă este formată din mai multe cuvinte, folosiți liniuța de subliniere pentru a separa cuvintele. Exemplu: JFrame.EXIT_ON_CLOSE

Numele lizibile sunt mai importante decât mai toate comentariile

Java nu ține seama dacă numele folosite sunt lizibile, dar este extrem de important pentru oameni.

4. Tipuri primitive în Java

3.1. Numere

Există două tipuri generale de numere în Java (și în multe alte limbaje de programare): **întregi** binari și numere în **virgulă mobilă (floating-point)** (uneori numite numere *reale*). Deși aceste numere sunt stocate binar, de obicei se folosesc numere zecimal în programele sursă; compilatorul le traduce în formatul binar corespunzător.

Întregi

Sunt patru feluri de întregi în Java: byte, short, int, long. **Cel mai frecvent este int.** Toți întregii sunt stocați în reprezentarea **cu semn în codul complement față de doi**.

D.p.d.v. tehnic, `char` este un întreg fără semn, deși este folosit aproape exclusiv pentru a stoca caractere. Ca este întreg se datorează în principal rădăcinilor în limbajul C++ ale Java. Nu folosiți `char` pentru întregi decât dacă știți ce faceți.

Clase. Pe lângă tipurile primitive, există două clase folosite pentru întregi.

- **Integer** – Utilă în principal pentru metodele care le oferă și pentru a pune structura de date **Collections** clase.
- **BigInteger** – Folosită acolo unde e nevoie aritmetică cu numere întregi extrem de mari.

Java stochează toți întregii ca numere binare.

Tipul	Dimensiunea		Gama de reprezentare	
	octeți	biți	minimum	Maximum
byte	1	8	-128	+127
short	2	16	-32,768	+32,767
int	4	32	-2,147,483,648	+2,147,483,647
long	8	64	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807

Iată cum se scriu literalii (constantele) **zecimali întregi**.

- Literalii `int` se scriu în notația zecimală obișnuită, d.e. 34 sau -222.
- Literalii `long` se scriu adăugând un L (sau l deși este aproape imposibil de distins un l de cifra 1), d.e. 34L sau -222L.

Nu există posibilitatea de a scrie un octet literal sau un `short`, deși câteodată Java va converti automat un literal **int** la tipul corespunzător.

Literali hexazecimali. Puteți scrie întregi în hexazecimal prin prefixarea numărului hexazecimal cu cifra zero urmată de litera x, "0x" sau "0X".

```
int i;
i = 0x2A; // atribuie numarul zecimal 42 lui i.
```

Operațiile pot produce numere care sunt prea mari pentru a fi stocate într-un `int`. Nu se semnalează nici o eroare, iar rezultatul va fi pur și simplu incorect. Împărțirea prin zero va

genera o excepție la execuție (ArithmeticException). Folosiți BigInteger pentru a preveni depășirea aritmetică.

Virgula mobilă

Numere în virgulă mobilă sunt ca și numerele *reale* din matematică. D.e., 3.14159, -0.000001. Java are două feluri de numere în virgulă mobilă: `float` și `double`, ambele stocate în format IEEE-754. Tipul **implicit** la scrierea unui număr în virgulă mobilă este `double`.

Tip	Dimensiune		Gamă	Precizie
<i>nume</i>	<i>octeți</i>	<i>biți</i>	<i>Valoare aproximativă</i>	<i>în cifre zecimale</i>
<code>float</code>	4	32	+/- 3.4 * 10 ³⁸	6-7
<code>double</code>	8	64	+/- 1.8 * 10 ³⁰⁸	15

Fiindcă există un număr limitat de biți pentru reprezentarea din fiecare tip în virgulă mobilă, unele numere pot fi inexacte, asemănător modului în care sistemul zecimal nu poate reprezenta exact unele numere, cum este 1/3. Cel mai problematic dintre acestea este faptul că 1/10 nu poate fi reprezentat exact în binar.

Literali în virgula mobilă

Există două feluri de notare pentru numerele în virgulă mobilă. Fiecare dintre acestea poate fi urmată de un "F" (sau "f") pentru a face numărul `float` în loc de implicitul `double`.

Notația standard care este o serie de cifre în partea întregă, urmată de punctul zecimal, urmată de o serie de cifre pentru partea fracționară. D.e., 3.14159 este un `double`. Un semn (+ sau -) poate precede numărul.

Notația științifică: literalul în virgulă mobilă standard urmat de litera "E" (sau "e") urmat de un exponent cu baza 10 opțional, care este folosit ca multiplicator (adică spune cum să se deplaseze punctul zecimal). Notația științifică este folosită în general pentru numere foarte mari sau foarte mici.

Științifică	Standard
1.2345e5	123450.0
1.2345e+5	123450.0
1.2345e-5	0.000012345

Infinit și NaN

Nu se generează excepții pentru operațiile în virgula mobilă. În locul unei întreruperi a execuției, rezultatul unei operații poate fi infinit pozitiv, infinit negativ sau NaN (not a number). Împărțirea cu zero sau depășirea pot produce infinit. Scăderea a doi infiniți produce NaN. Folosiți metodele din clasele de împachetare (învelire) (**Float** sau **Double**) pentru a testa aceste valori.

3.2. Convertirea șirurilor la numere

Pentru a converti o valoare șir la un număr (de exemplu pentru a converti valoarea dintr-un câmp String la un int), folosiți metodele din tabelul de mai jos. Presupunând că sunt făcute declarațiile următoare:

```
String s;  
int i;  
long l;  
float f;  
double d;
```

Tip	Exemplu
<code>int</code>	<code>i = Integer.parseInt(s);</code>
<code>long</code>	<code>l = Long.parseLong(s);</code>
<code>float</code>	<code>f = Float.parseFloat(s);</code>
<code>double</code>	<code>d = Double.parseDouble(s);</code>

<code>i == j</code>	egal	<code>6 == 24</code> este false.
<code>i >= j</code>	mai mare sau egal	<code>10 >= 10</code> este true.
<code>i > j</code>	mai mare	<code>10 > 10</code> este false.
<code>i != j</code>	diferit	<code>6 != 24</code> este true.

Operatori logici

Operator	Nume	Semnificație
<code>a && b</code>	and	Rezultatul este true dacă și numai dacă atât <i>a</i> cât și <i>b</i> sunt true.
<code>a b</code>	or	Rezultatul este true dacă fie <i>a</i> fie <i>b</i> este true.
<code>!a</code>	not	true dacă <i>a</i> este false și false dacă <i>a</i> este true.

Alți operatori și metode care întorc valori booleene

Operatorul `instanceof`.

Multe **metode** returnează valori booleene, d.e. `equals` și metode care încep cu "is" (este...). Dacă scrieți propria Dvs. metodă booleană este recomandat să începeți (în engleză) cu "is".

Operatori mai puțin comuni folosiți nerecomandat cu tipul boolean sunt: `&`, `|` și `^` cu operanzi booleeni. Acești operatori se folosesc de obicei pe biți. `||` (sau) și `&&` (și) sunt preferați lui `|` și `&` fiindcă ei sunt operatori *scurtcircuit* care pot opri evaluarea atunci când unul dintre operanzi determină valoarea rezultatului.

Variabile booleene

Puteți declara și testa variabile booleene. Spre exemplu, sortarea prin metoda bulelor repetă operațiile până nu mai apar interschimbări. Exemplu:

```
void bubbleSort(int[] x, int n)
{
    boolean anotherPass; // true dacă un element nu a fost în ordine
    do {
        anotherPass = false; // presupunem că totul este sortat
        for (int i=0; i<n-1; i++) {
            if (x[i] > x[i+1]) {
                int temp = x[i]; x[i] = x[i+1]; x[i+1] = temp; // interschimbă
                anotherPass = true; // ceva nu este ordonat, continuă
            }
        }
    } while (anotherPass);
}
```

Character

Metodele statice ale clasei Character

Metode ale clasei Character		
<i>Clasa Character este folosită mai ales pentru metode statice pentru a testa valori char.</i>		
b =	<code>Character.isDigit(c)</code>	adevărat dacă c este caracter cifră.
b =	<code>Character.isLetter(c)</code>	adevărat dacă c este caracter literă.
b =	<code>Character.isLetterOrDigit(c)</code>	adevărat dacă c este literă sau cifră.
b =	<code>Character.isLowerCase(c)</code>	adevărat dacă c este caracter minuscule.
b =	<code>Character.isUpperCase(c)</code>	adevărat dacă c este caracter majuscul.
b =	<code>Character.isWhitespace(c)</code>	adevărat dacă c este spațiu, tab,
c =	<code>Character.toLowerCase(c)</code>	Versiunea minusculă a lui c.
c =	<code>Character.toUpperCase(c)</code>	Versiunea majusculă a lui c.

Secțiunile ANSI/ASCII și Extended Latin ale Unicode

Unicode încearcă să reprezinte caracterele din toate limbile curente, precum și numeroase simboluri speciale. Cea mai frecventă implementare a Unicode folosește 16 biți, ceea ce permite

65,536 caractere (multe nu au încă atribuit un simbol grafic). Primele 128 coduri sunt identice cu ANSI/ASCII (American National Standards Institute / American Standard Code for Information Interchange). Dintre codurile ASCII, primele 32 sunt coduri de control. Primele 256 coduri sunt la fel cu ISO-8859-1 (Latin-1), care include ASCII. Tabelul de mai jos arată aceasta parte comună a setului de caractere Unicode.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15
32		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	€	q	r	s	t	u	v	w	x	y	z	{		}	~	•
128	€	•	,	f	„	...	†	‡	^	% ₀₀	Š	<	Œ	•	Ž	•
144	•	`	'	“	”	•	-	—	~	™	š	>	œ	•	ž	ÿ
160		i	¢	£	×	¥	!	§	“	©	ª	«	¬		®	¯
176	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
192	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

5. IO simplă

Clasa Java 5' *java.util.Scanner* a simplificat IO pe consolă. Iată un exemplu:

```
// File   : introductiv/IntroScanner.java
// Purpose: Scrie și citește p/de pe consolă.
// Author : Michael Maus
// Date   : 2005-03-29

import java.util.*;

public class IntroScanner
{
    public static void main(String[] args) {
        //... Inițializare
        String nume;           // Declara o variabila pentru nume.
        Scanner in = new Scanner(System.in);

        //... Scrie o explicație și citește intrarea.
        System.out.println("Care e numele tau, prietene?");
        nume = in.nextLine(); // Citește o linie de la consolă.

        //... Display output
        System.out.println("Du-ma la seful vostru, " + nume);

        System.out.println("\nInput an integer value: ");
        int val = in.nextInt();
        System.out.println("S-a citit: " + val );

        float f;
        System.out.println("\nIntroduceti un numar real: ");
        f = in.nextFloat();
        System.out.println("Numarul de tip float citit este: " + f);
    }
}
```

Un rezumat pentru IO consolă.

Caracteristica	Consola
Imports	<code>import java.util.*; // Scanner</code>
Inițializare	<code>// Declară și inițializează un obiect Scanner. Scanner input = new Scanner(System.in);</code>
Citirea unei linii de text	<code>System.out.print("Numele Dvs.: "); String nume; nume = input.nextLine();</code>
Citirea unui întreg	<code>System.out.print("Vârsta Dvs.: "); int virsta = input.nextInt();</code>
Output	<code>System.out.println(rezultat);</code>

Pentru output pe consolă nu sunt necesare clauze imports. Clasa `System` este automat importată (cum sunt toate clasele din `java.lang`). Puteți scrie o linie la consolă folosind metoda `System.out.println()`. Se va tipări argumentul acestei metode. `println` vine din Pascal și este o prescurtare pentru "print line". Există și o metodă `print` care scrie fără linie nouă după tipărire.

6. Mersul lucrării

6.1. Creați o aplicație simplă care să afișeze mesajul "Hello world".

6.2. Studiați documentația Java, pentru detalii despre clasele care împachetează tipurile primitive. <http://docs.oracle.com/javase/7/docs/api/>

Creați un program în care citiți de la consolă valori întregi și numere reale și le afișați apoi pe ecran. Vedeți exemplul clasei `IntroScanner` de la capitolul 5, pagina 9.

6.3. Scrieți scurte programe pentru a testa limitele reprezentărilor. Observați ce se întâmplă dacă:

- adunați o cantitate întreagă la cel mai mare întreg primitiv din fiecare categorie de întregi;
- scădeți o cantitate întreagă din cel mai mare întreg primitiv din fiecare categorie de întregi;
- reprezentați în virgulă mobilă numere cu un număr de cifre zecimale mai mare decât numărul de cifre reprezentabile exact
- adunați sau scădeți cantități din numere care au mai multe cifre zecimale în reprezentarea în baza 10 decât permite reprezentarea în virgulă mobilă

6.4. Problema boabelor de grâu pe o tablă de șah: dacă pe o tablă de șah se așează boabe de grâu astfel încât pe prima căsuță a tablei este un bob de grâu, pe a doua căsuță sunt 2 boabe de grâu, pe a treia căsuță sunt 4 boabe de grâu ș.a.m.d. (numărul de boabe de grâu se dublează la fiecare căsuță) și pentru căsuța i sunt 2^{i-1} , câte boabe de grâu sunt în total pe tabla de șah? Sugestie: folosiți obiecte `BigInteger`.

Exemplu de folosire `BigInteger`:

```
BigInteger nr1, nr2;

nr1 = BigInteger.valueOf(200); //se initializeaza nr1 cu 200
nr2 = BigInteger.valueOf(300); //se initializeaza nr2 cu 300
nr1 = nr1.add(nr2); //la nr1 se aduna nr2
System.out.println("Value for nr1 " + nr1 );
```