Tipuri primitive şi I/E simplă pe consolă

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 clasele învelitoare corespunzătoare
- Folosirea metodelor de I/E simplă pe consolă

2. Compilarea unui program Java

Presupunând că se foloseşte JDK (Java Development Kit) de la Oracle, paşii sunt următorii:

1. **Creaţi programul sursă cu un editor de texte** (d.e., Notepad, jEdit, TextPad, ...). Salvaţi-l într-un fişier cu acelaşi nume precum cel al clasei pe care o conţine şi adăugaţi-i extensia ".java" (d.e., Salut.java). Una dintre erorile uzuale este folosirea unui alt nume pentru fişier decât cel al clasei. Numele dinainte de "." trebuie să fie identic cu cel al clasei, inclusiv tipul de literă (mare sau mică). Mulţi programatori îşi salvează sursele la fiecare 10 minute – se face repede şi ne scapă de pericolul de a re-tasta totul dacă sistemul nu mai funcţionează.
2. Deschideţi o fereastră de comenzi (**cmd**) si navigaţi (cu **cd**) spre directorul care conţine sursa. Aceasta se face uşor dacă aveţi nume de directoare scurte şi fără spaţii.
3. **Compilaţi** programul sursă (**Salut.java** în acest exemplu) folosind comanda:

   javac Salut.java

   Aceasta va produce una sau mai multe fişiere ".class", care sunt în formatul obiect (Java byte code) a programelor Java.

4. **Rulaţi** aplicaţia cu:

   java Salut

   Această comandă va încărca fişierul **Salut.class** şi toate clasele necesare. Execuţia începe cu metoda **main** din clasa Salut. Continuaţi acest ciclu până când programul funcţionează.

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

2.1. Cum se localizează programele

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

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

Sau se poate face acest lucru folosind un IDE. În orice caz, Java trebuie să ştie atât pentru compilarea cât şi pentru execuţia programelor unde să găsească clasele de bibliotecă folosite. Java ştie cum să-şi localizeze clasele proprii, dar trebuie să îi comunicaţi unde să găsească alte biblioteci care le folosiţi. Directoarele standard Microsoft Windows nu sunt incluse în căutare.

S-ar putea să trebuiască să specificaţi o listă de directoare sau fişiere .**jar** unde se pot găsi programele, prin setarea *variabilei de mediu* `CLASSPATH`.

2.2. Organizarea lucrului

Ori de câte ori începeţi un **proiect nou**, creaţi un **nou director** pentru fişierele sursă. Numele directorului trebuie să fie din litere mici, fără spaţii sau alte semne de punctuaţie.
Mai multe clase în programele de dimensiuni mai mari sunt grupate de obicei în *packages* (pachete). După declaraţia opţională *package*, pot exista instrucţiuni *import*, care vă permit să specificaţi clase care pot fi referite fără a le califica prin numele pachetului.
Packages sunt directoare / cataloage care conţin clasele Java şi constituie o modalitate de grupare a claselor înrudite. Pentru programele mici, este uzual să se omită specificarea pachetului: Java creează ceea ce numeşte un pachet *default* (implicit) în acest caz.

O clasă per fişier

Puneţi fiecare clasă în propriul său fişier sursă, separat. Fiecare fişier sursă trebuie numit *exact* la fel cu clasa, plus sufixul ".java". Spre exemplu, dacă clasa se numeşte "Test", fişierul trebuie să fie numit "Test.java" (nu "test.java").
Se pot pune mai multe clase într-un fişier şi totul să funcţioneze. 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 fişier sursă separat, lucru cerut şi 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", Oracle]:

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

Scrierea de cod cu un stil bun oferă următoarele beneficii:

- Îmbunătăţeşte lizibilitatea, consistenţa şi omogenitatea codului, ceea ce îl face mai uşor de înţeles şi întreţinut.
- Face codul mai uşor de trasat şi depanat, pentru că este clar şi consistent.
- Permite continuarea mai uşoară a dezvoltării din locul unde Dvs. sau un alt programator s-a oprit, în special după o lungă perioadă de timp.
- Creşte beneficiile parcurgerii codului, deoarece participanţii 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 atenţie la detalii. Iată câteva linii generale de urmărit:

- Faceţi codul clar şi uşor de citit.
- Faceţi codul consistent.
- Folosiţi nume evidente pentru identificatori.
- Organizaţi-vă logic fişierele şi clasele.

- Stocaţi o clasă pe fişier (aici nu se numără clasele interne - inner classes).
- Folosiţi cel mult 80-90 caractere pe linie.
- Folosiţi judicios spaţiile albe şi/sau alţi separatori.
- Folosiţi spaţiile în locul tabulatorilor la indentare (schimbarea dimensiunii tabulatorilor nu va afecta atunci aspectul codului scris) .

Acoladele şi indentarea

Stilul de indentare sau plasarea acoladelor ("{" şi "}") 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 instrucţiunii căreia îi aparţine logic blocul şi acolada închisă la acelaşi nivel de indentare cu instrucţiunea respectivă.

Stilul de comentare a programelor este şi el parte a stilului de programare.

3.1. Comentarii în Java

Programele sunt citite şi de calculatoare şi de oameni. Instrucţiunile le scrieţi pentru a spune calculatorului ce să facă. Trebuie însă să scrieţi şi comentarii pentru a explica oamenilor ce face programul. Desigur, Java nu le înţelege pentru că ele sunt scrise în limbaj natural.
Java ignoră toate comentariile. Totuşi, există un program numit **javadoc** care citeşte anumite tipuri de comentarii şi produce documentaţie în format HTML.
Folosiţi spatii şi linii goale în program. Una dintre cele mai eficiente modalităţi pentru a face un program lizibil este să puneţi spaţii în punctele cheie. Sunt câteva stiluri pentru a face asta. Chiar mai important este să puneţi linii goale în program. Acestea vor separa secţiunile de cod. Ar trebui să fie o linie goală între fiecare dintre grupurile de instrucţiuni 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 sfârşitul 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 garanţie.
```

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

După caracterele **/*** , Java va ignora totul până găseşte o pereche ***/**. Acest fel de comentariu, se poate întinde pe mai multe linii şi se foloseşte de obicei pentru a "comenta" secţiuni de cod – comentând, de exemplu, porţiuni de cod în timpul depanării unui program. Spre exemplu,

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

Comentarii javadoc

Comentariile care încep cu **/**** sunt folosite de programul **javadoc** pentru a produce documentaţie în formatul HTML pentru program. Documentaţia Java de la Oracle este produsă folosind **javadoc**. Este esenţial să folosiţi acest fel de comentariu pentru programele mari. Vă recomandăm cu tărie să folosiţi acest fel de comentarii pentru a promova reutilizarea codului scris de Dvs.

Cele mai bune practici referitor la comentarii:

- Nu scrieţi comentarii pentru a documenta lucruri evidente. Presupuneţi că cititorul ştie Java.

- Fiecare comentariu are potenţialul de a crea inconsistenţe între ceea ce spune şi 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 situaţie, ţineţi comentariile în aproprierea codului pe care îl documentează astfel încât să fie mai uşor de sincronizat.

3.2. 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 (şi 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 **interfaţă**.
APPLE	Alt nume legal. **Toate majuscule** semnifică o **constantă**.
topleft	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 spaţii
import	ILEGAL – este cuvânt Java

Folosirea majusculelor, minusculelor şi a amestecului de litere

Convenţiile 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 compuşi din mai multe cuvinte.

Numele de interfeţe şi clase – încep cu literă mare

Numele de interfeţe şi clase încep cu literă mare şi continua cu litere mici. Pentru cuvinte multiple folosiţi stilul cămilă. Exemple: Direction, LogicalLayout, DebugGapSpacer.

Nume de metode şi de variabile – încep cu literă mică

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

Constante – doar majuscule, folosiţi _ 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, folosiţi liniuţa de subliniere pentru a separa cuvintele. Exemplu: JFrame.EXIT_ON_CLOSE

Numele lizibile sunt mai importante decât mai toate comentariile

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

4. Tipuri primitive în Java

4.1. Numere

Există două tipuri generale de numere în Java (şi în multe alte limbaje de programare): **întregi** binari şi numere în **virgulă mobilă**(**floating-point**) (uneori numite numere *reale*). Deşi 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**. Toţi întregii sunt stocaţi în reprezentarea **cu semn în codul complement faţă de doi**.
D.p.d.v. tehnic, `char` este un întreg fără semn, deşi este folosit aproape exclusiv pentru a stoca caractere. Că este întreg se datorează în principal rădăcinilor în limbajul C++ ale Java. Nu folosiţi `char` pentru întregi decât dacă ştiţi ce faceţi.

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

- **Integer** – Clasă învelitoare utilă în principal pentru metodele care le oferă şi pentru a pune obiecte care încapsulează valori întregi în clase **Collections**.
- **BigInteger** – Folosită acolo unde e nevoie aritmetică cu numere întregi extrem de mari.

Java stochează toţi întregii ca numere binare.

Tipul	**Dimensiunea**		**Gama de reprezentare**	
numele	octeţi	biţi	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 notaţia zecimală obişnuită, d.e. 34 sau -222.
- Literalii `long` se scriu adăugând un L (sau l deşi 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`, deşi câteodată Java va converti automat un literal **int** la tipul corespunzător.

Literali hexazecimali. Puteţi 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.
```

Operaţiile 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 şi simplu incorect. Împărţirea prin zero va

genera o excepţie la execuţie (ArithmeticException). Folosiţi BigInteger pentru a preveni depăşirea aritmetică.

Virgula mobilă

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

Tip	**Dimensiune**		**Gamă**	**Precizie**
Nume	*octeţi*	*biţi*	*Valoare aproximativă*	*în cifre zecimale*
`float`	4	32	+/- $3.4 * 10^{38}$	6-7
`double`	8	64	+/- $1.8 * 10^{308}$	15

Fiindcă există un număr limitat de biţi 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`.

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

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

Ştiinţifică	**Standard**
1.2345e5	123450.0
1.2345e+5	123450.0
1.2345e-5	0.000012345

Infinit şi NaN

Nu se generează excepţii pentru operaţiile în virgula mobilă. În locul unei întreruperi a execuţiei, rezultatul unei operaţii poate fi infinit pozitiv, infinit negativ sau NaN (not a number). Împărţirea cu zero sau depăşirea pot produce infinit. Scăderea a doi infiniţi produce NaN. Folosiţi metodele din clasele de împachetare (învelire) (**Float** sau **Double**) pentru a testa aceste valori.

4.2. Convertirea şirurilor la numere

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

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

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

Dacă *s* este null sau nu reprezintă un număr valid de tipul respectiv, metodele generează (*throw*=aruncă*)* o excepţie `NumberFormatException`.

Tratarea excepţiilor de tipul NumberFormatException

Puneţi conversiile de numere în interiorul unei instrucţiuni **`try ... catch`** ca să puteţi face ceva dacă nu se introduce ce trebuie. Metoda de conversie va *arunca* o excepţie NumberFormatException la intrare necorespunzătoare. Prindeţi *(catch)* excepţia şi trataţi această eroare. Puneţi-vă conversia în clauza `try`, iar tratarea erorii în clauza `catch`. Iată un exemplu de rutină (în cadrul unei funcții utilitare pentru a obține un întreg folosind un dialog) pe care aţi putea să o scrieţi pentru astfel de situaţii:

```
public static int getInt(String mess) {
    int val;
    while (true) { // itereaza in bucla pana cand se primeste un int valid
        String s = JOptionPane.showInputDialog(null, mess);
        try
        {
          val = Integer.parseInt(s);
          break;  // iese din bucla cu un intreg valid >>>>>>>>>>>>>>>>>>>>>
        }
         catch (NumberFormatException nx)
        {
          JOptionPane.showMessageDialog(null, "Introduce\u0163i un \u00eentreg valid");
        }
    }
    return val;
}//end getInt
```

Întregi ne-zecimali

Convertiţi întregi din alte baze decât 10 folosind aceste două metode. Cel mai adesea numerele sunt în hexazecimal (baza 16) sau în binar (baza 2).

Tip	*Exemplu*
`Int`	*`i`* `= Integer.parseInt(`*`s`*`, `*`radix`*`);`
`long`	*`l`* `= Long.parseLong(`*`s`*`, `*`radix`*`);`

Spre exemplu, pentru a converti un şir care conţine numărul hexazecimal "F7" la un întreg, apelaţi

```
i = Integer.parseInt("F7", 16)
```

Java oferă şi clase pentru precizie arbitrară pentru numerele zecimale: `BigDecimal`.

4.3. Tipul primitiv boolean

Tipul primitiv `boolean` are două valori posibile: `true` şi `false`.
Cele două valori sunt scrise folosind cuvintele rezervate `true` şi `false`.
Instrucţiunile `if`, `for`, `while` şi `do` toate necesită valori booleene. De obicei acestea sunt scrise ca expresii cu evaluare la valori booleene, folosind operatori care produc valori booleene.

Operatori de comparare

Sunt folosiţi pentru a compara valori primitive (mai rar și pentru obiecte).

Operator	Nume	Semnificaţie
`i < j`	mai mic	`6 < 24` este true.
`i <= j`	mai mic sau egal	`6 <= 24` este true.
`i == j`	egal	`6 == 24` este false.
`i >= j`	mai mare sau egal	`10 >= 10` este true.
`i > j`	mai mare	`10 > 10` este false.
`i != j`	diferit	`6 != 24` este true.

Operatori logici

Operator	Nume	Semnificaţie
`a && b`	And	Rezultatul este true dacă şi numai dacă atât *a* cât şi *b* sunt true.
`a || b`	Or	Rezultatul este true dacă fie *a* fie *b* este true.
`!a`	not	true dacă *a* este false şi false dacă *a* este true.

Alţi operatori şi metode care întorc valori booleene

Operatorul `instanceof`.

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

Operatori mai puţin comuni folosiţi nerecomandat cu tipul boolean sunt: &, | şi ^ cu operanzi booleeni. Aceşti operatori se folosesc de obicei pe biţi. Operatorii || (sau) şi && (şi) sunt preferaţi lui | şi & fiindcă ei sunt operatori *scurtcircuit* care pot opri evaluarea atunci când unul dintre operanzi determină valoarea rezultatului.

Variabile booleene

Puteţi declara şi testa variabile booleene. Spre exemplu, sortarea prin metoda bulelor repetă operaţiile 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);
}
```

4.4. Tipul caracter

Metodele statice ale clasei Character

Metode ale clasei Character		
Clasa **Character** este folosită mai ales pentru metodele statice care testază valori char.		
b =	Character.isDigit(c)	adevărat dacă c este caracter cifră.
b =	Character.isLetter(c)	adevărat dacă c este caracter literă.
b =	Character.isLetterOrDigit(c)	adevărat dacă c este literă sau cifră.
b =	Character.isLowerCase(c)	adevărat dacă c este minusculă
b =	Character.isUpperCase(c)	adevărat dacă c este majusculă
b =	Character.isWhitespace(c)	adevărat dacă c este spaţiu, tab, ...
c =	Character.toLowerCase(c)	Versiunea minusculă a lui c.
c =	Character.toUpperCase(c)	Versiunea majusculă a lui c.

Secţiunile ANSI/ASCII şi Extended Latin ale Unicode

Unicode încearcă să reprezinte caracterele din toate limbile curente, precum şi numeroase simboluri speciale. Cea mai frecventă implementare a Unicode foloseşte 16 biţi, 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 această 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	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	•
128	€	•	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ	•	Ž	•
144	•	‘	’	“	”	•	–	—	˜	™	š	›	œ	•	ž	Ÿ
160		¡	¢	£	¤	¥	¦	§	¨	©	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. I/E simplă pe consolă

Clasa *java.util.Scanner* a simplificat I/E pe consolă. Iată un exemplu:

```
// Author : Michael Maus
// Date   : 2005-03-29

import java.util.*;

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

        //... Scrie o explicatie si citeste intrarea
        System.out.println("Care e numele tau, prietene?");
        nume = in.nextLine();      // Citeste o linie de la consola.

        //... Afiseaza linia citita inainte
        System.out.println("Du-ma la seful vostru, " + nume);

        //... Citeste un intreg si apoi in afiseaza
       System.out.println("Introduceti un intreg: ");
       int val = in.nextInt();
       System.out.println("Numarul de tip intreg citit este: " + val );

        //... Citeste un numar real si apoi in afiseaza
       float f;
       System.out.println("Introduceti un numar real: ");
       f = in.nextFloat();
       System.out.println("Numarul de tip float citit este: " + f);

       //... Inchiderea citirii prin intermediul Scanner
       in.close();
   }
}
```

Un rezumat pentru I/E pe consolă

Caracteristica	Consola
Imports	`import java.util.*; // Scanner`
Iniţializare	`// Declară şi iniţializează un obiect Scanner.` `Scanner input = new Scanner(System.in);`

Citirea unei linii de text	`System.out.print("Numele Dvs.: ");`
	`String nume;`
	`nume = input.nextLine();`

Citirea unui întreg	`System.out.print("Vârsta Dvs.: ");` `int virsta = input.nextInt();`
Afișarea unui rezultat	`System.out.println(rezultat);`

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

6. Mersul lucrării

6.1. Fără a folosi mediul de programare Eclipse, compilați și apoi executați un program simplu Java prin intermediul unei ferestre de comenzi (**cmd**) (vezi secțiunea 2).

6.2. Studiaţi documentaţia Java, pentru detalii despre clasele care împachetează tipurile primitive (Integer, Float, Double, Boolean, Character) https://docs.oracle.com/javase/8/docs/api/. Scrieți un program în care să realizați conversia șirurilor la numere utilizând metode statice din clasele care împachetează tipurile primitive (vezi exemplul metodei statice *getInt* din capitolul 4.2). Ce se întâmplă când conversia nu este posibilă?

6.3. Scrieţi un program pentru a testa limitele reprezentărilor tipurilor primitive. Observaţi ce se întâmpla dacă:
- adunaţi o cantitate întreagă la cel mai mare întreg primitiv din fiecare categorie de întregi
- scădeţi o cantitate întreagă din cel mai mic întreg primitiv din fiecare categorie de întregi
- înmulțiți cel mai mare număr real reprezentat în simplă și dublă precizie cu o valoare supraunitară
- reprezentaţi în virgulă mobilă numere cu un număr de cifre zecimale mai mare decât numărul de cifre reprezentabile exact
- adunaţi sau scădeţi cantităţi din numere care au mai multe cifre zecimale în reprezentarea în baza 10 decât permite reprezentarea în virgulă mobilă

6.4. Scrieți un program pentru a testa ce se întâmplă dacă:
- împărțiti un întreg la zero
- împărțiți un număr negativ în virgulă mobilă la zero; apoi unul pozitiv la zero
- scădeți în virgulă mobilă din Infinit un alt Infinit
- înmulțiți în virgulă mobilă Infinit cu -Infinit
- asignați unei valori booleene un întreg
- asignați unei valori întregi un număr în virgulă mobilă

6.5. Scrieți un program în care citiți de la consolă valori întregi şi numere reale şi le afişaţi apoi pe ecran (vezi exemplul clasei IntroScanner din secțiunea 5).

6.6. Problema boabelor de grâu pe o tablă de şah: dacă pe o tablă de şah se aşează boabe de grâu astfel încât pe prima căsuţă a tablei este un bob de grâu, pe a doua căsuţă sunt 2 boabe de gâu, pe a treia căsuţă sunt 4 boabe de grâu ş.a.m.d. (numărul de boabe de grâu se dublează la fiecare căsuţă) şi pentru căsuţa i sunt 2^{i-1}, câte boabe de grâu sunt în total pe tabla de şah? Sugestie: folosiţi obiecte BigInteger pentru a calcula numărul total de boabe de grâu.