Programare orientată pe obiecte

#8 JAVA
Expresii și operatori. Instrucțiuni

https://www.runceanu.ro/adrian/activitate-didactica/

copyright@www.adrian.runceanu.ro

Curs 8

Limbajul JAVA. Expresii și operatori. Instrucțiuni

copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:
   - 5.1. Instrucţiunea break
   - 5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

1. Expresii şi operatori

- O expresie este compusa dintr-o succesiune de operanzi, legati prin operatori.
- Un operand poate fi o constanta, o variabila, un apel de metoda, o expresie incadrata intre paranteze rotunde.
- Operatorii desemneaza operatiile care se executa asupra operanzilor si pot fi grupati pe categorii, in functie de tipul operatiilor realizate.
- Operatorii limbajului Java sunt:
  - unari (se aplica unui singur operand)
  - binari (se aplica asupra a doi operanzi)

copyright@www.adrian.runceanu.ro

A. Operatorii aritmetici

- Operatorii aritmetici sunt:
  - '*' - inmultirea
  - '/' - impartirea
  - '%' - restul impartirii intregi
  - '+' - adunarea
  - '-' - scaderea
- De asemenea este folosit operatorul unar '-' (minus) pentru schimbarea semnului, precum si operatorul unar '+' (plus) (introdus pentru simetrie).

copyright@www.adrian.runceanu.ro

Nota:

1. Operatorul ‘%’ nu poate fi aplicat decat operanzilor intregi.
2. Operatorul ‘/’ poate fi aplicat atat operanzilor intregi, cat si operanzilor reali, dar functioneaza diferit pentru operanzii intregi, fata de operanzii reali.

- Daca cei doi operanzi sunt numere intregi, operandul ‘/’ are ca rezultat catul impartirii intregi (fara parte fractionara).
- Daca cel putin unul dintre cei doi operanzi este un numar real, operandul ‘/’ furnizeaza rezultatul impartirii reale (cu parte fractionara).

copyright@www.adrian.runceanu.ro

De exemplu:

Fie declaratiile de variabile:

int a=5, b=7

float x=3.5

Expresie	Rezultat
b%2	1
a/2	2
x/2	1.75

copyright@www.adrian.runceanu.ro

B. Operatorii de incrementare/decrementare:

- Operatorul de incrementare este '**++**'
- Operatorul de decrementare este '**--**'
- Acesti operatori sunt unari si au ca efect marirea (respectiv micsorarea) valorii operandului cu 1.

Limbajul Java permite doua forme pentru operatorii de incrementare / decrementare:

- forma prefixata (inaintea operandului)
- forma postfixata (dupa operand)

copyright@www.adrian.runceanu.ro

- In cazul cand se foloseste operatorul de incrementare / decrementare in forma prefixata (inaintea operandului), limbajul Java va incrementa / decrementa mai intai valoarea variabilei si apoi va utiliza variabila intr-o alta expresie.

- In cazul cand se foloseste operatorul de incrementare / decrementare in forma postfixata (dupa operand), limbajul Java va utiliza mai intai valoarea variabilei intr-o alta expresie si apoi va efectua operatia de incrementare / decrementare.

copyright@www.adrian.runceanu.ro

- De exemplu, daca valoarea curenta a lui x este 5, atunci:

  - evaluarea expresiei 3 * ++x conduce la rezultatul 18

  - evaluarea expresiei 3 * x++ conduce la rezultatul 15

  - dupa care valoarea lui x va fi in ambele cazuri 6

- *Operatorii de incrementare / decrementare pot fi aplicati operanzilor intregi, operanzilor in virgula mobila si operanzilor de tipul char.*

copyright@www.adrian.runceanu.ro

C. Operatori relaţionali

- Operatorii relationali sunt operatori binari si desemneaza relatia de ordine in care se gasesc cei doi operanzi:

  <, >, <=, >=

- Rezultatul aplicarii unui operator relational este **true** daca cei doi operanzi sunt in relatia indicata de operator, si **false**, altfel.

copyright@www.adrian.runceanu.ro

De exemplu, expresiile logice:

2 > 14 are ca rezultat valoarea **false**,

15 <= 4+21 are ca rezultat valoarea **true**.

copyright@www.adrian.runceanu.ro

- Un alt operator relational este *instanceof* care *testeaza daca un anumit obiect este sau nu instanta a unei anumite clase de obiecte* (adica, apartine unei clase de obiecte).

De exemplu:

```
Object o = new Object( );
String s = new String( );
```

o instanceof Object - are ca rezultat valoarea **true**
s instanceof String - are ca rezultat valoarea **true**
o instanceof String - are ca rezultat valoarea **false**

copyright@www.adrian.runceanu.ro

D. Operatori de egalitate

- Operatorii de egalitate sunt folositi pentru testarea unei egalitati sau inegalitati intre doua valori.
- Sunt operatori binari si arata relatia de egalitate (**==**) sau inegalitate (**!=**).
- Rezultatul aplicarii unui operator de egalitate este **true**, daca cei doi operanzi sunt in relatia indicata de operator si **false** altfel.

De exemplu, expresiile logice:

5 == 2+3 are ca rezultat valoarea **true**

5 != 2+3 are ca rezultat valoarea **false**

copyright@www.adrian.runceanu.ro

E. Operatori logici se aplica asupra unor operanzi de tip *boolean*.

Exista trei operatori logici globali:

- negatia logica (not) reprezentata cu **!**

- conjunctie logica (si) reprezentata cu **&&**

- disjunctie logica (sau) reprezentata cu **||**

copyright@www.adrian.runceanu.ro

! true = false
! false = true

true && true = true
true && false = false
false && true = false
false && false = false

true || true = true
true || false = true
false || true = true
false || false = false

Tabele de evaluare a operatorilor logici pentru expresii:

! (negatie)
&& (si)
||(sau)

copyright@www.adrian.runceanu.ro

Nota:

- O regula importanta este ca operatorii logici **&&** si **||** folosesc *evaluarea booleana partiala (scurcircuitata).*
- Aceasta inseamna ca *daca rezultatul poate fi determinat evaluand prima expresie, a doua expresie nu mai este evaluata*.

✓ De exemplu, in expresia:

```
x != 0 && 1/x != 5
```

✓ Daca x este 0, atunci prima jumatate are valoarea **false**.

✓ Aceasta inseamna ca rezultatul conjunctiei va fi fals, deci a doua expresie nu mai este evaluata.

copyright@www.adrian.runceanu.ro

F. Operatori la nivel de biti

- Operatorii logici pe biti se aplica numai operanzilor intregi (de tipul *byte*, *short*, *int* si *long*) si *au acelasi rezultat ca si operatiile logice pentru expresii* (negatie, conjunctie, disjunctie si disjunctie exclusiva) ***dar bit cu bit***.

- De fapt, operatorii se aplica reprezentarii binare a numerelor implicate

copyright@www.adrian.runceanu.ro

- Operatorii logici pe biti sunt:

Operator	Denumire	Tip
~	Complementare (negatie) pe biti	unar
&	Conjunctia logica (si) pe biti	binar
^	Disjunctie exclusiva (sau exclusiv) pe biti	binar
|	Disjunctie logica (sau) pe biti	binar

copyright@www.adrian.runceanu.ro

H. Operatori de atribuire

- Operatorii de atribuire sunt operatori binari care permit modificarea valorii unei variabile.

Exista:

- un operator de atribuire simplu (**=**)
- 10 operatori de atribuire compusi cu ajutorul operatorului '**=**' si un alt operator binar (aritmetic sau logic pe biti).

copyright@www.adrian.runceanu.ro

O varianta de sintaxa folosita este:

<nume_variabila> = <expresie>

Efectul aplicarii operatorului este:

Se evalueaza <expresie>, apoi se atribuie variabilei <nume_variabila> valoarea expresiei.

Nota: <expresie> poate fi la randul ei o expresie de atribuire, caz in care se realizeaza o atribuire multipla.

copyright@www.adrian.runceanu.ro

- Atunci cand compilatorul intalneste o operatie de atribuire multipla, el atribuie valorile de la dreapta la stanga.

<nume_variabila1> = <nume_variabila2> = … = <nume_variabilan> = <expresie>;

- Se foloseste atunci cand se doreste sa se atribuie aceeasi valoare mai multor variabile.

copyright@www.adrian.runceanu.ro

A doua varianta de sintaxa folosita este:

<nume_variabila> <operator_binar> = <expresie>;

unde:

- <operator_binar> - este din multimea { ***, /, %, +, -, <<, >>, &, |, ^** }.

Efectul aplicarii operatorilor de atribuire compusi este echivalent cu instructiunea:

<nume_variabila> = <nume_variabila> <operator_binar> <expresie>;

copyright@www.adrian.runceanu.ro

I. Operatorul de concatenare (+) de siruri de caractere

este un operator binar folosit pentru alipirea mai multor siruri de caractere.

- La concatenarea sirurilor de caractere, lungimea sirului rezultat este suma lungimii sirurilor care intra in operatie.
- Caracterele din sirul rezultat sunt caracterele din primul sir, urmate de cele dintr-al doilea sir in ordine.
- Daca cel de-al doilea operand este un tip primitiv de data, acesta este convertit la un sir de caractere care sa reprezinte valoarea operandului.

copyright@www.adrian.runceanu.ro

I. Operatorul de concatenare (+) de siruri de caractere

este un operator binar folosit pentru alipirea mai multor siruri de caractere.

De exemplu:

“Acesta este ” + “un sir” este echivalent cu “Acesta este un sir”.

“Variabila a are valoarea ” + 3 este echivalent cu “Variabila are valoarea 3”.

copyright@www.adrian.runceanu.ro

J. Operatorul conversie de tip (sau conversie explicita de tip sau *cast*) *este un operator unar utilizat pentru a genera o variabila temporara de un nou tip*.

Rezultatul unui *cast* este valoarea operandului convertita la noul tip de data exprimat de *cast*.

O conversie explicita de tip (un *cast*) este de forma:

(<tip_nou>) <expresie>

unde:

- <tip_nou> - este noul tip de data al expresiei <expresie> altul decat cel declarat initial sau implicit;
- <expresie> - este o variabila sau o expresie care se doreste a fi convertita la tipul nou.

copyright@www.adrian.runceanu.ro

De exemplu, in secventa de instructiuni:

```
double f = 7.8;
int i = (int)f;
```

valoarea variabilei *f* este convertita la o valoare intreaga si anume 7, si noua valoare este atribuita variabilei *i*.

copyright@www.adrian.runceanu.ro

K. Operatorul conditional ?:

Operatorul conditional examineaza o conditie si returneaza o valoare daca conditia este adevarata si alta daca conditia este falsa.

Sintaxa operatorului conditional este:

(<conditie>) ? <rezultat_adevar> : <rezultat_fals>

unde:

- <conditie> - o expresie de evaluat;
- <rezultat_adevar> -rezultatul returnat daca conditia are valoarea **true**;
- <rezultat_fals> - rezultatul returnat daca conditia are valoarea **false**.

copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:
   - 5.1. Instrucţiunea break
   - 5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

2. Instructiuni simple si blocuri de instructiuni

- O instructiune simpla este o singura instructiune, cum ar fi aceea prin care se atribuie o valoare unei variabile (instructiunea de atribuire) sau se apeleaza o metoda.
- Un bloc de instructiuni (numit si instructiune compusa) este o secventa de instructiuni simple si declaratii de variabile locale.
- Aceste instructiuni se executa in ordinea in care apar in interiorul blocului.
- Sintactic, blocurile de instructiuni sunt delimitate de acolade.
- Blocurile de instructiuni pot fi incluse (imbricate) in cadrul altor blocuri de instructiuni.

copyright@www.adrian.runceanu.ro

Sintaxa unui bloc de instructiuni este:

```
{
 <declaratii_de_variabile_locale>;
  <instructiune_1>;
  <instructiune_2>;
   …
  <declaratii_de_variabile_locale>;
  <instructiune_n>;
}
```

unde:

- <declaratii_de_variabile_locale> - reprezinta instructiuni de declarare a unor **variabile locale**;

o instructiune de declarare poate sa apara oriunde in interiorul unui bloc.

copyright@www.adrian.runceanu.ro

Nota:

Declaratiile de variabile locale care apar intr-un bloc sunt valabile numai in interiorul blocului, din momentul declararii lor pana la sfarsitul blocului.

- **Instructiunea vida** este o instructiune care nu executa nimic.

- Ea este formata numai din **;** si se foloseste atunci cand este obligatoriu sa avem o instructiune, dar nu dorim sa executam nimic in acea instructiune.

copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:
   - 5.1. Instrucţiunea break
   - 5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

3. Structuri fundamentale de control

A. Structuri alternative (de decizie)

3.1. Instructiunea if

Sintaxa instructiunii este:

```
if (<expresie>) <instructiune_1>;
    [else <instructiune_2>];
```

unde:

- <expresie> - specifica expresia de evaluat;
- <instructiune_1>, <instructiune_2> - specifica instructiunile (simple sau compuse) de executat.

copyright@www.adrian.runceanu.ro

Semantica:

- se evalueaza <expresie> si
- daca valoarea expresiei este **true**, se executa <instructiune_1>,
- altfel se executa <instructiune_2>.

Nota:

- Instructiunea **if** poate sa faca parte dintr-o alta instructiune **if** sau **else**, adica instructiunile **if** pot fi incluse (imbricate) in alte instructiuni **if**.

copyright@www.adrian.runceanu.ro

Exemplu

Urmatorul program (denumit arie_triunghi.java) testeaza daca trei numere pot forma laturile unui triunghi si daca da, calculeaza aria triunghiului folosind *formula lui Heron.*

copyright@www.adrian.runceanu.ro

```
import java.util.*;
public class Triunghi
{
    private static Scanner input;
    public static void main(String args[])
    {
        int x, y, z;
        double p, aria;
        System.out.print ("Introduceti cele 3 valori  ");
        input = new Scanner(System.in);
        x = input.nextInt();
        y = input.nextInt();
        z = input.nextInt();
```

copyright@www.adrian.runceanu.ro

```
if (x<=0 || y<=0 || z<=0)
    System.out.println("\nNumerele introduse nu sunt
    laturi ale unui triunghi");
else
    if (x+y<=z || x+z<=y || y+z<=x)
        System.out.println("\nNumerele introduse nu
    sunt laturi ale unui triunghi");
    else
    {
        p = (x+y+z)/2;
        aria = Math.sqrt(p*(p-x)*(p-y)*(p-z));
        System.out.println("\nAria triunghiului = " + aria);
    }
}
}
```

copyright@www.adrian.runceanu.ro

Executia programului pe date de test:
https://www.jdoodle.com/online-java-compiler/

```java
import java.util.*;
public class Triunghi
{
    private static Scanner input;

    public static void main(String args[])
    {
        int x, y, z;
        double p, aria;

        System.out.print ("Introduceti cele 3 valori  ");
        input = new Scanner(System.in);
        x = input.nextInt();

        y = input.nextInt();

        z = input.nextInt();

        if (x<=0 || y<=0 || z<=0)
            System.out.println("\nNumerele introduse nu sunt laturi ale unui triunghi");
        else
            if (x+y<=z || x+z<=y || y+z<=x)
                System.out.println("\nNumerele introduse nu sunt laturi ale unui triunghi");
            else
            {
                p = (x+y+z)/2;
                aria = Math.sqrt(p*(p-x)*(p-y)*(p-z));
                System.out.println("\nAria triunghiului = " + aria);
            }
    }
```

Execute Mode, Version, Inputs & Arguments

JDK 11.0.4

Interactive

Stdin Inputs

3 4 5

CommandLine Arguments

Execute

Result

CPU Time: 0.20 sec(s), Memory: 36764 kilobyte(s)

```
Introduceti cele 3 valori
Aria triunghiului = 6.0
```

copyright@www.adrian.runceanu.ro

Observatie:

Metoda sqrt() face parte din **clasa de obiecte Math** care este implementata in **pachetul java.lang**.

Metoda sqrt() este de tip *double* si are un parametru de tip *double*.

copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:
   - 5.1. Instrucţiunea break
   - 5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

3.2. Instructiunea switch

Sintaxa instructiunii este:

```
switch (<expresie>)
{
  case <constanta_1> : <grup_de_instructiuni_1>;
  case <constanta_2> : <grup_de_instructiuni_2>;
   …
  case <constanta_n> : <grup_de_instructiuni_n>;
  [default: <grup_de_instructiuni_n+1>;]
}
```

copyright@www.adrian.runceanu.ro

unde:

- <expresie> - specifica variabila sau expresia de evaluat;
- <constanta_1>, <constanta_2>, …, <constanta_n> - specifica valorile constantelor cu care se face compararea rezultatului evaluarii expresiei;
- <grup_de_instructiuni_1>, … - o instructiune sau un grup de instructiuni care se executa in cazul in care o alternativa *case* se potriveste.

copyright@www.adrian.runceanu.ro

Semantica:

Se evalueaza <expresie>; se compara succesiv valoarea expresiei cu valorile constantelor <constanta_1>, <constanta_2>, …, <constanta_n> din alternativele *case*:

- ✓ daca se intalneste o constanta din alternativa *case* cu valoarea expresiei, se executa secventa de instructiuni corespunzatoare si toate secventele de instructiuni care urmeaza, pana la intalnirea instructiunii **break** sau pana la intalnirea acoladei inchise (**}**) care marcheaza sfarsitul instructiunii **switch**;
- ✓ daca nici una dintre valorile constantelor din alternativa *case* nu coincide cu valoarea expresiei, se executa secventa de instructiuni din alternativa *default* (alternativa implicita sau prestabilita).

copyright@www.adrian.runceanu.ro

Observatii:

1. Spre deosebire de **if-else**, care permite selectarea unei alternative din maximum doua posibile, **switch** permite selectarea unei alternative din maximum n+1 posibile.

2. In instructiunea **if-else** se executa instructiunea (instructiunile) corespunzatoare valorii expresiei si atat, in timp ce in instructiunea **switch** se executa si toate secventele de instructiuni ale alternativelor *case* urmatoare.

copyright@www.adrian.runceanu.ro

Instructiunea break din switch

Sintaxa instructiunii este:

break;

Semantica: determina iesirea neconditionata din instructiunea **switch**, adica opreste executia secventelor de instructiuni ale alternativelor *case* urmatoare.

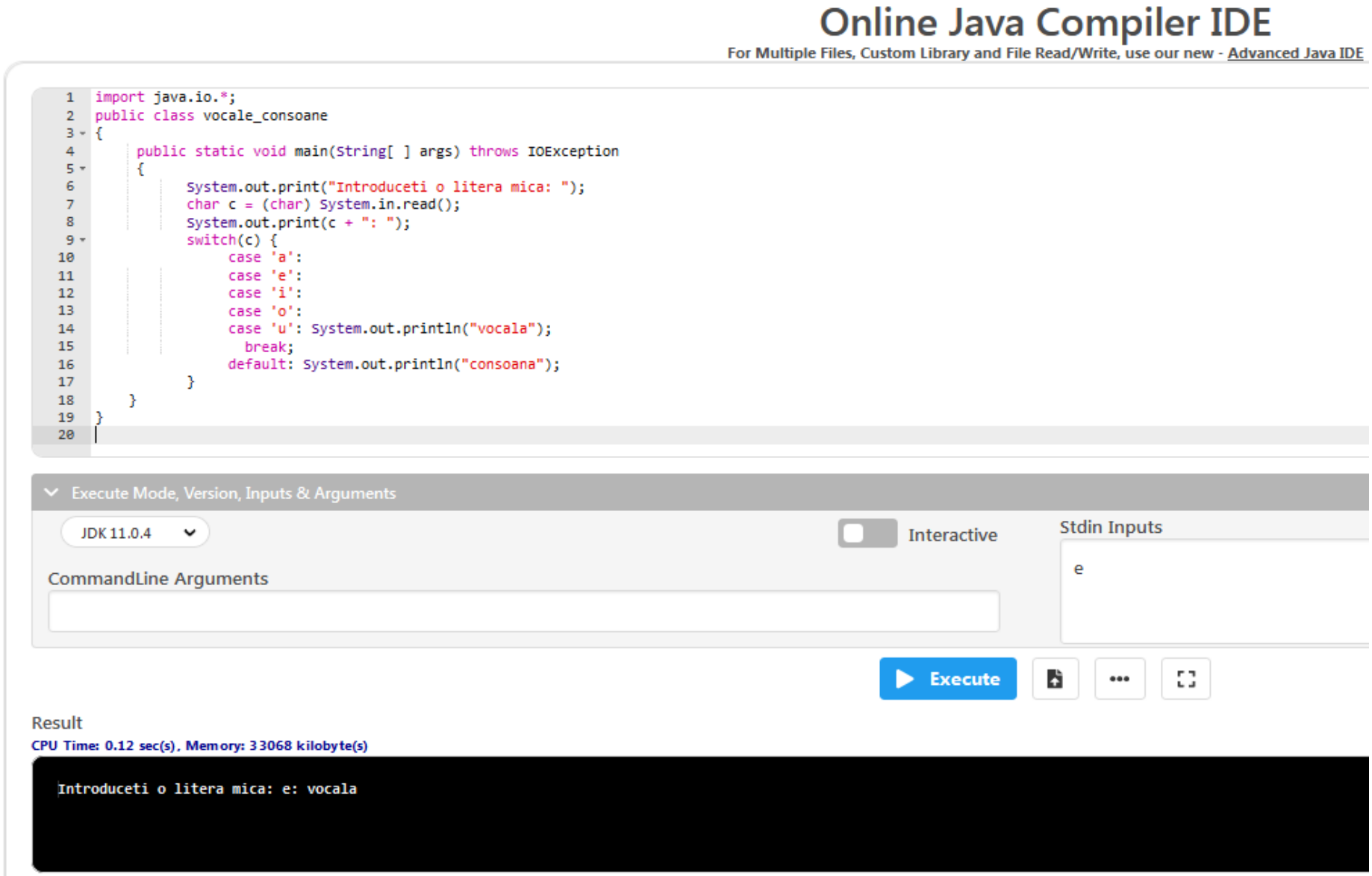
Exemplu urmator (vocale_consoane.java) citeste de la tastatura o litera si determina daca aceasta este o vocala sau o consoana.

copyright@www.adrian.runceanu.ro

```
import java.io.*;
public class vocale_consoane
{
    public static void main(String[ ] args) throws IOException
    {
        System.out.print("Introduceti o litera mica: ");
        char c = (char) System.in.read();
        System.out.print(c + ": ");
        switch(c) {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u': System.out.println("vocala");
            break;
            default: System.out.println("consoana");
          }
    }
}
```

copyright@www.adrian.runceanu.ro

Executia programului pe date de test:
https://www.jdoodle.com/online-java-compiler/



copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:
   - 5.1. Instrucţiunea break
   - 5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

4. Structuri repetitive

4.1. Instructiunea while

Sintaxa instructiunii este:

```
while (<expresie>)
    <instructiune>;
```

unde:

- <expresie> - specifica expresia de testat;

Semantica: se evalueaza <expresie>:

- daca valoarea expresiei este **false** se iese din ciclul **while**;
- daca valoarea expresiei este **true**, se executa instructiunea atita tip cat valoarea expresiei este **true**.

copyright@www.adrian.runceanu.ro

Exemplu:

Sa se verifice daca un numar este *numar perfect* sau nu.

Spunem ca un numar este *numar perfect daca este egal cu suma divizorilor lui, mai putin el insusi*.

Exemplu: numarul 6 este perfect, deoarece este egal cu suma divizorilor sai 1,2,3.

copyright@www.adrian.runceanu.ro

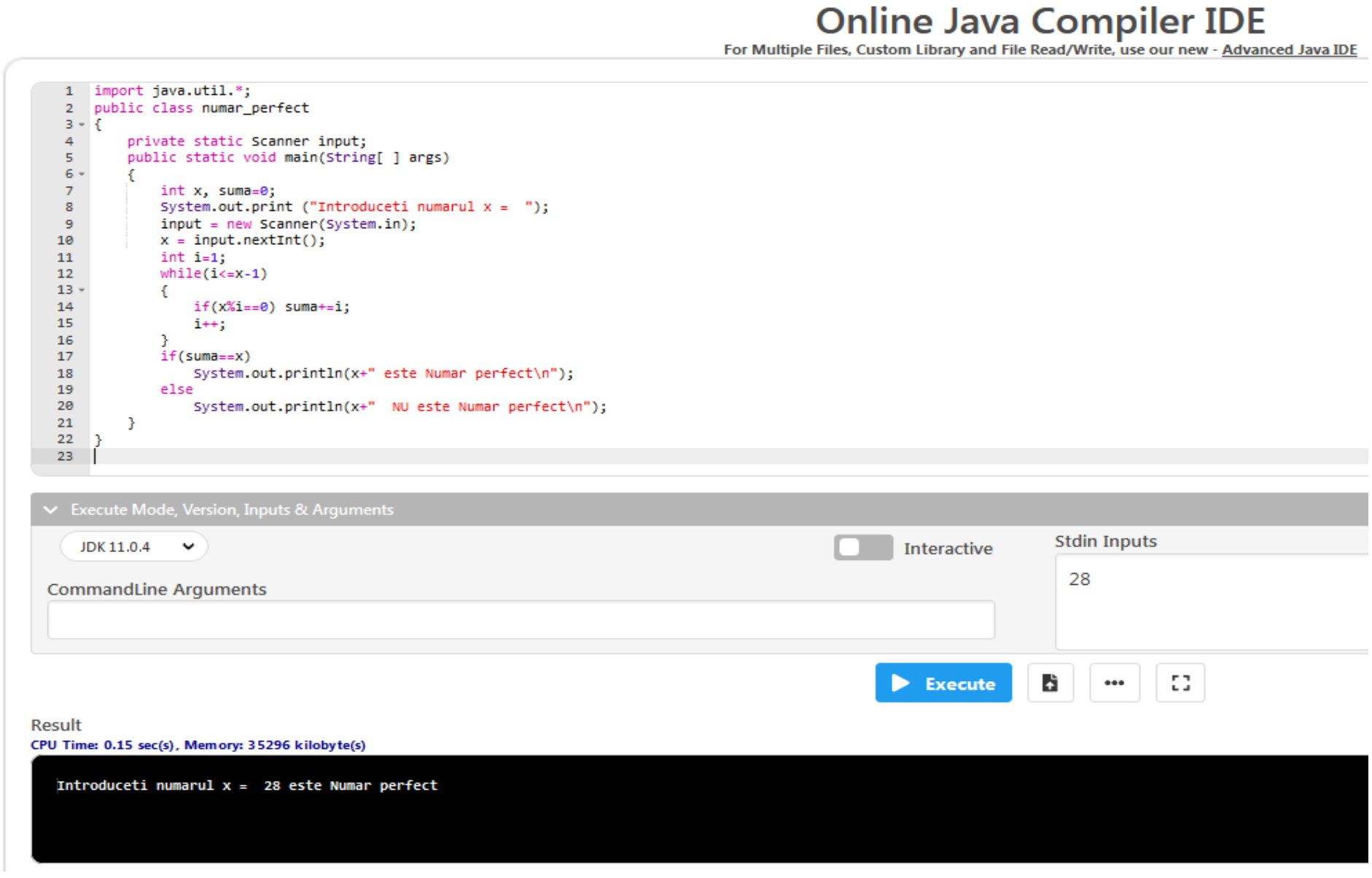
```
import java.util.*;

public class numar_perfect
{
    private static Scanner input;

    public static void main(String[ ] args)
    {
        int x, suma=0;
        System.out.print ("Introduceti numarul x =  ");
        input = new Scanner(System.in);
        x = input.nextInt();
```

copyright@www.adrian.runceanu.ro

```
        int i=1;
        while(i<=x-1)
        {
                if(x%i==0) suma+=i;
                i++;
        }
        if(suma==x)
                System.out.println(x+" este Numar
    perfect\n");
        else
                System.out.println(x+"  NU este Numar
    perfect\n");
}
}
```

copyright@www.adrian.runceanu.ro

Executia programului pe date de test:
https://www.jdoodle.com/online-java-compiler/



copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:
   - 5.1. Instrucţiunea break
   - 5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

4.2. Instructiunea do-while

Sintaxa instructiunii este:

```
do
   <instructiune>;
while (<expresie>);
```

unde:

- <instructiune> - o instructiune simpla de executat;
- <expresie> - specifica expresia de testat (de evaluat);

Semantica: se executa instructiunea si apoi se evalueaza expresia:

- daca valoarea expresiei este **false** se iese din ciclul **do-while**;
- daca valoarea expresiei este **true** se executa instructiunea (din ciclul **do-while**) atata tip cat valoarea expresiei este adevarata.

copyright@www.adrian.runceanu.ro

Exemplu:

Urmatorul program (cifra_control.java) citeste de la tastatura un numar natural x si calculeaza cifra de control a lui x.

Cifra de control a unui numar natural se obtine calculand suma cifrelor numarului, apoi suma cifrelor sumei, s.a.m.d. pana la obtinerea unei singure cifre.

De exemplu, pentru x = 335 calculam suma cifrelor 3+3+5 = 11.

Cum suma nu este formata dintr-o singura cifra, repetam procedeul: 1+1=2.

Deci 2 este cifra de control a lui 335.

copyright@www.adrian.runceanu.ro

```
import java.util.*;
public class cifra_control
{
   private static Scanner input;
   public static void main(String[ ] args)
   {
        int x, suma=0;
        //System.out.print ("Introduceti numarul x =
   ");
        input = new Scanner(System.in);
        x = input.nextInt();
```

copyright@www.adrian.runceanu.ro

```
    do
      {
       while (x !=0)
         {
                  suma+=x%10;
                  x/=10;
         }
      System.out.println("Suma cifrelor numarului: " +
    suma);
      x = suma;
      suma = 0;
    } while (x >9);
    System.out.println("Cifra de control a numarului este: "
    + x);
    }
}
```

copyright@www.adrian.runceanu.ro

Executia programului pe date de test:
https://www.jdoodle.com/online-java-compiler/

copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:

5.1. Instrucţiunea break

5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

4.3. Instructiunea for

Este folosita pentru efectuarea unor prelucrari de un anumit numar de ori.

Sintaxa instructiunii este:

```
for (<valoare_initiala>; <conditie_sfarsit>;
<valoare_increment>)
  <instructiune>;
```

Instructiunea **for** foloseste, de obicei, o variabila denumita variabila de control care indica de cate ori s-a executat instructiunea (<instructiune>) din corpul ciclului.

copyright@www.adrian.runceanu.ro

Instructiunea **for** contine patru sectiuni:

1. sectiunea **<valoare_initiala>** atribuie variabilei de control o valoare initiala, care, de cele mai multe ori, este 0 sau 1;
2. sectiunea **<conditie_sfarsit>** testeaza valoarea variabilei de control pentru a stabili daca programul a executat instructiunea de atatea ori cat s-a dorit;
3. sectiunea **<valoare_increment>** **adauga (scade),** de obicei, valoarea 1 la variabila de control, de fiecare data, dupa ce se executa instructiunea din corpul ciclului; valoarea de incrementare sau decrementare poate fi diferita de 1;
4. sectiunea **<instructiune>** reprezinta instructiunea (sau instructiunile) care se doreste (doresc) a fi repetata (repetate).

copyright@www.adrian.runceanu.ro

- ✓ Pentru a exemplifica instructiunea **for**, programul urmator (vocale_consoane_random.java) creaza 10 litere aleator si determina daca acestea sunt vocale sau consoane.
- ✓ Metoda **Math.random** face parte din clasa **Math** care se gaseste in pachetul **java.lang** si este folosita in program pentru a genera o valoare aleatoare in intervalul [0, 1).
- ✓ Prin inmultirea valorii returnate de aceasta functie cu numarul de litere din alfabet (26 litere) se obtine un numar in intervalul [0, 26).
- ✓ Adunarea cu prima litera (‘a’, care are de fapt valoarea 97, codul ASCII al literei ‘a’) are ca efect transpunerea in intervalul [97, 123).
- ✓ In final se foloseste operatorul de conversie explicita de tip pentru a trunchia numarul la o valoare din multimea 97, 98, …, 122, adica un cod ASCII al unui caracter din alfabetul englez.

copyright@www.adrian.runceanu.ro

```
public class vocale_consoane_random
{
    public static void main(String[] args)
    {
        for (int i = 0; i < 10; i++)
        {
            char c = (char) (Math.random() * 26 + 'a');
            System.out.print(c + ": ");
            switch(c)
            {
              case 'a': case 'e': case 'i': case 'o': case 'u':
                    System.out.println("vocala");
                    break;
              default: System.out.println("consoana");
            }
        }
    }
}
```

copyright@www.adrian.runceanu.ro

Executia programului pe date de test:
https://www.jdoodle.com/online-java-compiler/

copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:
   - 5.1. Instrucţiunea break
   - 5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

5. Instructiunile break si continue

5.1.Instructiunea **break** permite intreruperea instructiunilor care se repeta in corpul ciclului **for**, **while** sau **do-while**.

De obicei, instructiunea **break** apare in cadrul unei instructiuni **if**, ca in exemplul urmator:

```
while (…)
{
      …
      if (conditie)
          {
                  break;
          }
      …
}
```

copyright@www.adrian.runceanu.ro

- ✓ In cazul in care sunt doua cicluri imbricate, instructiunea **break** intrerupe doar ciclul cel mai interior.

- ✓ Instructiunea **break etichetata** este folosita cand sunt mai mult de doua cicluri imbricate.

- ✓ In acest, o anumita instructiune de ciclare este etichetata si instructiunea **break** poate fi aplicata acelei instructiuni de ciclare, indiferent de numarul de cicluri imbricate.

Un exemplu:

```
eticheta:
   while (…)
    {
         while (…)
          {
               …
               if (conditie)
                    {
                      break eticheta;
                    }
           }
    }
```

copyright@www.adrian.runceanu.ro

1. Expresii şi operatori
2. Instrucţiuni simple şi blocuri de instrucţiuni
3. Structuri fundamentale de control:
   - 3.1. Instrucţiunea if
   - 3.2. Instrucţiunea switch
4. Structuri repetitive:
   - 4.1. Instrucţiunea while
   - 4.2. Instrucţiunea do-while
   - 4.3. Instrucţiunea for
5. Structuri de salt:
   - 5.1. Instrucţiunea break
   - 5.2. Instrucţiunea continue

copyright@www.adrian.runceanu.ro

5.2. Instructiunea

continue permite terminarea iteratiei curente din ciclu for, while sau do-while si trecerea la urmatoarea iteratie a ciclului.

- ✓ Se aplica doar ciclului cel mai interior, in cazul ciclurilor imbricate.

- ✓ Urmatorul fragment de cod tipareste primele 50 de numere intregi, cu exceptia celor divizibile cu 10:

```
for (int i=1; i<=50; i++)
{
    if (i%10 == 0)
    {
        continue;
    }
    System.out.println(i);
}
```

copyright@www.adrian.runceanu.ro

Întrebări?