Expresii si instructiuni de control in Java

1. Scopul lucrării

Obiectivele acestei sesiuni de laborator sunt:

- Înțelegerea caracteristicilor specifice ale operatorilor Java și deprinderea utilizării lor corecte
- Utilizării instrucțiunilor de control al fluxului execuției în Java
- Acumularea de experiență nemijlocită în lucrul cu variabile și expresii prin dezvoltarea și executarea de programe de dimensiuni mici

2. Expresii

Expresiile constituie principalul mijloc pentru a crea valori. Expresiile sunt create prin combinarea literalilor (constantelor), a variabilelor, invocărilor de metode şi utilizând operatori. Pentru a obţine ordinea de evaluare corectă se vor folosi paranteze.

2.3.1. Tipuri.

Fiecare variabilă are asociat un *tip*. Sunt prevăzute două categorii de tipuri:

- **Tipuri primitive** : byte, short, char, int, long, float, double, boolean.
- **Tipuri obiect** : Tipurile string şi tablou sunt tipuri predefinite, dar fiecare clasă care este definită creează un nou tip de obiect.

2.3.2. Constante literale

Oferă un mod de a scrie valori aparţinând diferitelor tipuri (3, 3.0, true, 'a', "abc").

2.3.3. Variabile

Fiecare variabilă trebuie declarată ca şi aparţinând unui tip. Variabilele se pot încadra în trei categorii de bază diferite:

- Variabile locale declarate în metode .
- Variabile instanţă (deseori numite câmpuri) declarate în obiecte.
- Variabile clasă (static) declarate în clase.

2.1. Operatori

Operatorii sunt utilizaţi pentru a exprima diferite formule elementare de calcul sau prelucrare a informaţiei şi sunt folosiţi alături de literali, variabile, apeluri de metode şi alte expresii. Operatorii se pot clasifica în următoarele categorii.

- Operatori aritmetici(+, -, *, /, %, ++, --)
- Operatori relaţionali (<, <=, ==, >=, >, !=)
- Operatori booleeni (&&, ||, !, &, !, ^, |, &)
- Operatori la nivel de bit (&, |, ^, ~, <<, >>, >>>)
- Operatorul de concatenare a String-urilor (+)
- Operatori speciali (`instanceof`, ?:)
- Operatori de atribuire (=, +=, -=, *=, ...)

2.2. Ordinea de evaluare

In general, ordinea de evaluare a expresiilor este de la stânga la dreapta; excepţie de la această regulă o fac operatorii de atribuire. Ordinea de evaluare poate fi schimbată utilizând paranteze. A se vedea tabelul de precedenţă al operatorilor.

2.3.4. Operatori relaţionali

Toţi operatorii standard funcţionează pentru *valori de tip primitiv* (`int`, `double`, `char`, ...). Operatorii == (test de egalitate) şi != (test de inegalitate) pot fi utilizaţi pentru a compara referinţe

la un obiect. A se vedea secţiunea Compararea Obiectelor pentru a înţelege cum funcţionează comparaţia *valorilor obiect*.

2.3.5. Operatori

Rezultatul oricărei comparaţii este o valoare de tip *boolean* (`true` sau `false`).

operator	Semnificaţie
<	mai mic
<=	mai mic sau egal
==	Egalitate
>=	mai mare sau egal
>	mai mare
!=	Inegalitate (diferit de)

2.3.6. Erori comune

`0 < x < 100`

Operatorii de comparare nu pot exprima inegalităţi duble. Deşi matematic formula este corectă, sintaxa Java nu permite scrierea ei. Exprimarea corectă în limbajul Java este scrierea a două comparaţii:

```
0<x && x<100
```

= în loc de ==

Utilizarea operatorului de atribuire în loc de test de egalitate va genera o eroare de compilare uşor de corectat.

== pentru valori reprezentate în virgulă flotantă

Deoarece numerele în virgulă mobilă nu oferă totdeauna reprezentarea exactă a valorilor va trebui să utilizaţi >= sau <= în loc de ==. Spre exemplu, deoarece numărul zecimal 0.1 nu poate fi reprezentat exact în binar, expresia (0.1 + 0.1 + 0.1) **nu** va fi egală cu 0.3!

Recomandări pentru programatorii C/C++

Operatorii de comparare ai limbajului Java sunt la prima vedere identici cu cei din C/C++. Diferenţa constă în faptul că tipul rezultatului operaţiei de comparare este boolean. Astfel, eroarea frecventă în limbajul C: utilizarea operatorului = în loc de == este aproape complet eliminată. Limbajul Java nu permite supraîncărcarea operatorului, aspect ce poate fi ignorat de unii programatori obişnuiţi cu filozofia C++.

2.3.7. ==, .equals(), şi compareTo()

Testul de egalitate : o variantă pentru tipuri primitive, patru variante pentru obiecte

Comparaţie	Primitive	Obiecte
`a == b`, `a != b`	Valori egale	Referinţă spre acelaşi obiect
`a.equals(b)`	Nu se aplică	Compară valorile, dacă sunt definite pentru această clasă, dacă sunt definite pentru clasele respective, se comportă ca şi pentru clasele predefinite Java. Dacă nu este definită pentru o clasă (utilizator) se comportă ca şi operatorul ==.
`a.compareTo(b)`	Nu se aplică	Compară valori. Clasa trebuie să aibă implementată interfaţa *Comparable<T>* . Toate clasele Java având relaţia de ordine naturală au implementată această interfaţă (`String, Double, BigInteger,` ...) Face excepţie clasa `BigDecimal` care poate produce rezultate imprevizibile. Obiectele *Comparable* pot fi utilizate de metodele **Collections** *sort()* şi de structurile de date sortate implicit (d.e. `TreeSet, TreeMap`).
`compareTo(a, b)`	Nu se aplică	Compară valori. Este disponibilă numai dacă este implementată interfaţa *Comparator<T>* care nu este specifică pentru clasele Java. De obicei se foloseşte pentru a defini un obiect de comparare care va fi transmis metodelor **Collections** sort() sau structurilor de date ordonate.

Compararea referinţelor la obiecte folosind operatorii == şi !=
Cei doi operatori de comparare pentru referinţe la obiecte sunt testul de egalitate (==) şi testul de neegalitate (!=). Aceşti operatori compară două valori pentru a vedea dacă ele **referă acelaşi obiect**. Cu toate că este o operaţiune foarte rapidă nu întotdeauna produce ceea ce aşteptaţi.
De obicei vreţi să aflaţi dacă obiectele au aceeaşi *valoare*, şi nu dacă două obiecte constituie o *referinţă* spre acelaşi obiect. Spre exemplu,

```
if (name == "Mickey Mouse")   // Aproape sigur o eroare
```

Aici, rezultatul este adevărat(true) dacă `name` este o referinţă către *acelaşi obiect* spre care face referinţă şi "`Mickey Mouse`". In schimb, rezultatul va fi probabil false dacă String-ul din variabila `name` a fost citit din fişierul de intrare sau calculat (prin concatenare sau extragere subşir), chiar dacă `name` are într-adevăr exact aceleaşi caractere memorate în locaţia asociată.
Mai multe clase (d.e., `String`) definesc metoda `equals()` pentru a compara *valori* ale obiectelor.

Compararea valorilor obiectelor utilizând metoda `equals()`

Se va folosi metoda `equals()` pentru a compara valori ale obiectelor. Metoda `equals()` furnizează o valoare de tip boolean. Exemplul de mai sus poate fi corectat scriind:

```
if (name.equals("Mickey Mouse"))  // Compara valori, nu referințe.
```

Alte comparaţii – Interfaţa Comparable<T>

Metoda `equals` precum şi operatorii `==` şi `!=` sunt prevăzuţi pentru testul de egalitate/inegalitate, dar nu oferă o metodă de a testa valori înrudite. Unele clase (d.e. String şi alte clase având relaţia de ordine naturală) au implementată interfaţa *Comparable<T>* , care defineşte metoda `compareTo` . Va trebui să implementaţi *Comparable<T>* în clasele proprii dacă aveţi nevoie să le utilizaţi în metodele Collections.sort() sau Arrays.sort(). Clasa String oferă şi metode de comparaţie care nu fac diferenţă între literele mari şi mici.

Erori frecvente în utilizarea operatorului == în loc de `equals()` pentru obiecte

Dacă doriţi să comparaţi obiecte, trebuie să ştiţi că operatorul == se foloseşte pentru a afla dacă ele sunt *acelaşi obiect*, iar metoda `equals()` pentru a testa dacă ele sunt obiecte diferite, dar au *aceeaşi valoare*. O eroare produsă de această eventuală confuzie este greu de depistat.

2.3.8. Operatori la nivel de bit

Operatorii pe biţi acţionează asupra biţilor valorilor întregi (int şi long). Dacă un operand este mai scurt decât un întreg, el este convertit la int înainte de a efectua operaţiunea.
Este necesar a şti cum se reprezintă valorile întregi în sistemul binar. Spre exemplu numărul zecimal 3 este reprezentat 11 în baza 2 iar reprezentarea binară a lui 5 este 101. Numerele întregi negative sunt reprezentate utilizând codul *complementului faţă de 2*. Spre exemplu, -4 este reprezentat prin următoarea secvenţă de cifre binare : 1111 1111 1111 1111 1111 1111 1111 1100.

Operatorii la nivel de bit

Operator	Nume	Exemplu	Rezultat	Descriere
a & b	şi	3 & 5	1	1 dacă ambii biţi au valoarea 1.
a | b	sau	3 | 5	7	1 dacă cel puţin un bit are valoarea 1.
a ^ b	sau exclusiv	3 ^ 5	6	1 dacă biţii au valori diferite.
~a	complement	~3	-4	Inversează valoarea biţilor.
n << p	Deplasare stânga fără semn	3 <<2	12	Deplasează biţii lui *n* la stânga cu *p* poziţii. Biţi având valoarea zero sunt introduşi în poziţiile de rang nesemnificativ.

n >> p	Deplasare dreapta cu semn	5 >> 2	1	Deplasează biţii lui *n* la dreapta cu *p* poziţii. Dacă n este un număr cu semn reprezentat în forma *complement faţă de 2* , semnul este propagat în poziţiile de rang semnificativ.
n >>> p	Deplasare dreapta fără semn	-4 >>> 28	15	Deplasează biţii lui *n* la dreapta cu *p* poziţii. Biţi având valoarea zero se introduc în poziţiile de rang semnificativ.

Utilizarea împachetării / despachetării

O utilizare frecventă a operatorilor la nivel de bit (deplasări combinate cu *şi* pentru extragere de valori şi deplasări combinate cu *or* pentru adăugare valori) este aceea de a codifica mai multe valori într-o singură valoare de tip int. Procedeul se numeşte împachetare, iar extragerea rezultatului despachetare. Câmpurile de biţi oferă o altă metodă de a face acelaşi lucru. Spre exemplu, să presupunem că avem de a face cu următoarele variabile întregi : `virsta` (domeniu 0-127), `sex` (domeniu 0-1), `inaltime` (domeniu 0-255). Acestea pot fi împachetate/despachetate într-o singură variabilă `short` (întreg reprezentat pe doi octeţi) după cum urmează (sau în variante similare).

```
int   virsta, sex, inaltime;
short info_impachetata;
. . .
// împachetare
info_impachetata = (short)((((virsta << 1) | sex) << 7) | inaltime);
. . .
// despachetare
inaltime = info_impachetata & 0xff;
sex = (info_impachetata >>> 7) & 1;
virsta    = (info_impachetata >>> 8);
```

Utilizarea biţilor fanion (flag)

Anumite funcţii de bibliotecă interpretează valoarea unui întreg ca şi un şir de biţi în care fiecare reprezintă o valoare booleană (true/false) . Această presupunere are avantajul că economiseşte spaţiu de memorare şi în acelaşi timp poate să fie procesată rapid.

Utilizarea deplasărilor: deplasarea la stânga – înmulţire cu 2; deplasarea la dreapta – împărţire cu 2

Pe unele calculatoare mai vechi operaţiunile de deplasare sunt mai rapide decât înmulţirile respectiv împărţirile.

```
y = x << 3;        // Atribuie lui y valoarea 8*x
y = (x << 2) + x;  // Atribuie lui y valoarea 5*x
```

Utilizarea operatorului sau-exclusiv pentru a comuta pe unu/zero

Uneori se foloseşte operatorul *sau-exclusiv* pentru a comuta valorile 1 şi 0.

```
x = x ^ 1;      // sau în stil criptic  x ^= 1;
```

Această instrucţiune plasată într-o buclă va schimba alternativ 0 cu 1.

Utilizare îndoielnică : Schimbare de valori folosind operatorul sau-exclusiv

Iată o secvenţă de cod ciudată. Se foloseşte operatorul *sau-exclusiv* pentru a schimba între ele două valori (x şi y). Aceasta este traducerea în Java a unei secvenţe dintr-un limbaj de asamblare, unde nu se dispunea de spaţiu de memorare pentru o variabilă temporară. Nu se recomandă să utilizaţi o schemă de acest gen, ea trebuie privită ca şi o curiozitate dintr-un muzeu al codurilor bizare.

```
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Nu confundaţi operatorii && şi &

Nu confundaţi operatorul &&, care produce operaţia *şi logic*, cu operatorul &, care este o *transformare de biţi, numită "şi pe biţi".* Acesta din urmă este mai puţin obişnuit . Cu toate că operaţia *şi pe biţi*

poate fi folosită împreună cu operanzi de tip boolean, acest lucru este extrem de rar şi aproape întotdeauna generează o eroare de programare.

2.3. Rezumat pentru expresii

Parantezele () au trei utilizări:

1. Gruparea pentru a controla ordinea de evaluare sau pentru claritate. Exemplu: (a + b) * (c - d)
2. După o metodă pentru a cuprinde parametrii. Exemplu: x = suma(a, b);
3. În jurul unui tip pentru a forma determina o conversie explicită (*cast*). Exemplu: i = (int)x;

Ordinea de evaluare

- Operatorii cu precedenţa mai mare se aplică înaintea celor cu precedenţă mai mică.
- La precedenţă egală, evaluarea este de la stânga la dreapta, cu excepţia operatorilor unari, a atribuirii şi operatorului condiţional.

Abrevieri folosite

i, j – valori întregi (int, long, short, byte, char).
m, n – valori numerice (întregi, double, sau float).
b, c - boolean; x, y – orice tip sau obiect.
s, t - String; a - tablou; o - obiect; co - clasă sau obiect

Precedenţa Operatorilor	
`. [] (argumente) post ++ --` `! ~ op.unari + - pre ++ --` `(tip) new` `* / %` `+ -` `<< >> >>>` `< <= > >= instanceof` `== !=` `&` `^` `|` `&&` `||` `?:` `= += -= etc.`	Memoraţi doar pentru operatorii unari, * / % + - *comparaţii* && || *= asignări* Folosiţi () pentru toate celelalte

Operatori aritmetici	
Rezultatul operaţiilor aritmetice este double dacă oricare dintre operanzi este double, altfel float dacă oricare dintre operanzi este float, altfel long dacă oricare dintre operanzi este long, altfel int.	
++i	Adună 1 la i *înainte* de a folosi valoarea în expresia curentă
– –i	Ca mai sus, dar scade
i++	Adună 1 la i *după ce* s-a folosit valoarea în expresia curentă
i– –	Ca mai sus, dar scade
n + m	Adunare. D.e. 7+5 este 12, 3 + 0.14 este 3.14
n - m	Scădere
n * m	Înmulţire
n / m	Împărţire. *D.e.* 3.0 / 2 este 1.5 , 3 / 2 este 1
n % m	Restul (Mod) după împărţirea lui n prin m. *D.e.* 7 % 3 este 1

Compararea valorilor primitive
Rezultatul tuturor comparaţiilor este boolean (true sau false).
== != < <= > >=

Operatori logici

Operanzii trebuie să fie booleeni. Rezultatul este boolean.	
b && c	Condiţionalul "şi". Este true daca amândoi operanzii sunt true, altfel false. Evaluarea se face cu scurtcircuitare. *D.e.* (false && orice) este false.
b || c	Condiţionalul "sau". Este true dacă oricare dintre operanzi este true, altfel false. Evaluarea se face cu scurtcircuitare. *D.e.* (true || orice) este true.
!b	true dacă b este false, false dacă b este true.
b & c	"Şi" care evaluează întotdeauna ambii operanzi (*nu* scurtcircuit).
b | c	"Sau"care evaluează întotdeauna ambii operanzi (*nu* scurtcircuit).
b ^ c	"Sau-exclusiv" La fel ca b != c

Operatorul condiţional	
b?x:y	dacă b este true, valoarea este x, altfel y. x şi y trebuie să fie de acelaşi tip.

Operatori de asignare (atribuire)	
=	Partea stângă trebuie să fie o lvaloare (vezi C).
+= -= *= …	Toţi operatorii binari, cu excepţia lui && şi || pot fi combinaţi cu o asignare. *D.e.* a += 1 este la fel ca a = a + 1

Operatori pe biţi	
Operatorii pe biţi operează pe biţi sau int. Rezultatul este int.	
i & j	Biţilor li se aplică operatorul "şi". 1 dacă ambii biţi sunt 1. 5 & 3 este 1.
i | j	Biţilor li se aplică "sau". 1 dacă oricare dintre biţi este 1. 5 | 3 este 7.
i ^ j	Biţilor li se aplică "sau-exclusiv". 1 dacă biţii sunt diferiţi. 5 ^ 3 este 6.
~i	Biţii sunt complementaţi (0 -> 1, 1 -> 0)
i << j	Biţii din i sunt deplasaţi cu j biţi spre stânga; sunt inserate zerouri prin partea dreaptă. 5 << 2 este 20.
i >> j	Biţii din i sunt deplasaţi cu j biţi spre dreapta. Se inserează biţi ca bitul de semn prin stânga. 5 >> 2 este 1.
i >>> j	Biţii din i sunt deplasaţi cu j biţi spre dreapta. Prin stânga se inserează zerouri.

Conversii explicite	
Folosiţi conversii explicite atunci când "îngustaţi" gama unei valori. De la cea mai restrânsă la cea mai largă, ordonarea tipurilor primitive este: byte, short, char, int, long, float, double. Obiectele se pot atribui fără precizare explicită a tipului în sus pe ierarhia de moştenire. Este nevoie de precizarea explicită a tipului pentru a merge în jos pe ierarhia de moştenire (downcasting).	
(*t*)x	Face din x o variabilă convertită la tipul t (nu modifică pe x)

Operatori asupra obiectelor	
co.f	Membru. Câmpul f sau metoda unui obiect sa a clasei co.
x instanceof co	true dacă obiectul x este o instanţă a clasei co, sau o instanţa de clasa lui co.
a[i]	Acces la elementele unui tablou.
s + t	Concatenare de obiecte şir dacă unul sau ambii operanzi sunt String.
x == y	True dacă x şi y se referă la acelaşi obiect, altfel fals (chiar dacă valorile obiectelor sunt identice).
x != y	Ca mai sus pentru inegalitate.
comparaţii	Compară **valori** de obiecte cu .equals() sau .compareTo()
x = y	Asignarea copiază *referinţa*, nu obiectul.

4. Controlul fluxului execuției

Cele ce urmează constituie un rezumat al sintaxei instrucţiunilor de control. Fiecare instrucţiune de control este o instrucţiune logică, care de multe ori include un *bloc* de instrucţiuni între acolade {}. Exemplele presupun că blocul conţine mai mult de o instrucţiune.

Indentarea este esenţială. Patru spaţii sunt cel mai frecvent utilizate.

4.1. Selecţia (`if, switch`)

4.1.1. Instrucţiunea if

```
//----- instrucţiune if cu o ramură
   if (expresie)
   {
       instrucţiuni // execută aceste instrucţiuni dacă  expresie se evaluează la
adevărat
   }
//----- instrucţiune if cu două ramuri (pe adevărat şi fals)
   if (expresie)
   {
       instrucţiuni // execută aceste instrucţiuni dacă  expresie se evaluează la
adevărat
   }
   else
   {
       instrucţiuni // execută aceste instrucţiuni dacă  expresie se evaluează la
fals
   }
//----- instrucţiuni if  înlănţuite
   if (expresie1)
   {
       instrucţiuni // execută aceste instrucţiuni dacă  expresie1 este adevărată
   }
   else if (expresie2)
   {
       instrucţiuni // execută aceste instrucţiuni dacă  expresie2 este adevărată
   }
   else if (expresie3)
   {
       instrucţiuni // execută aceste instrucţiuni dacă  expresie3 este adevărată
   . . .
   }
   else
   {
       instrucţiuni // execută aceste instrucţiuni dacă nici o expresie n-a fost
adevărată
   }
```

4.1.2. Instrucţiunea switch

Efectul instrucţiunii **switch** este alegerea unor instrucţiuni de executat în funcţie de valoarea întreagă a unei expresii. Acelaşi efect poate fi obţinut cu o serie de instrucţiuni **if** înlănţuite, dar în anumite cazuri instrucţiunea **switch** este mai uşor de citit, iar unele în compilatoare poate produce cod mai eficient. Instrucţiunea **break** provoacă ieşirea din instrucţiune **switch**. Dacă nu există **break** la sfârşitul unui caz (**case**), execuţia continuă în cazul următor, iar acest lucru este de multe ori o eroare de logică a programului..

```
   switch (expr) {
      case c1:
            instrucţiuni // execută aceste instrucţiuni dacă  expr == c1
            break;
      case c2:
```

```
            instrucţiuni // execută aceste instrucţiuni dacă  expr == c2
            break;
      case c2:
      case c3:
      case c4:         //  cazurile au o tratare comună.
            instrucţiuni // execută aceste instrucţiuni dacă  expr ==  oricare
dintre c-uri
            break;
      . . .
      default:
            instrucţiuni // execută aceste instrucţiuni dacă  expr != oricare
dintre cele de mai sus
   }
```

4.2. Instrucţiuni de ciclare

4.2.1. while

Instrucţiunea **while** testează *expresie*. Dacă *expresie* se evaluează la adevărat, se execută corpul instrucţiunii while. Dacă este falsă, execuţia continuă cu instrucţiunea de după corpul instrucţiunii while. De fiecare dată după executarea corpului, execuţia reîncepe cu testul. Acest proces continuă până când *expresie* devine falsă sau o alta instrucţiune (**break** sau **return**) întrerupe ciclarea.

```
   while (testExpresie) {
       instrucţiuni
   }
```

4.2.2. for

Multe bucle au o iniţializarea înainte de buclă şi un fel de "incrementare" înainte de următoarea parcurgere a ciclului. Bucla **for** reprezintă modalitatea standard pentru combinarea acestor trei părţi.

```
   for (instrucţiuneIniţială; testExpr; instrucţiuneIncrement) {
       instrucţiuni
   }
```

Aceasta este la fel cu (cu excepţia faptului ca instrucţiunea **continue** în bucla **for** va executa şi incrementarea):

```
   instrucţiuneIniţială;
   while (testExpr)
   {
       instrucţiuni
       instrucţiuneIncrement
   }
```

4.2.3. do

Aceasta este cel mai puţin folosită dintre instrucţiunile de ciclare, dar câteodată este necesară o buclă care se execută o dată înainte de test.

```
   do
   {
      instrucţiuni
   }
   while (testExpr);
```

4.2.4. Alte instrucţiuni de control pentru cicluri

Toate instrucţiunile de ciclare pot fi etichetate, astfel încât se poate folosi `break` şi `continue` la orice adâncime de imbricare. Etichetele trebuie să le preceadă folosirea.

```
   break;       //ieşi din bucla sau switch-ul cel mai interior
   break eticheta; //ieşi din bucla etichetată cu eticheta
   continue;    //începe următoarea iteraţie a ciclului
   continue eticheta; //începe următorul ciclu cu eticheta eticheta
```

Plasaţi eticheta urmată de două puncte în faţa buclei.

```
exterioara: for (. . .) {
        . . .
        continue exterioara;
```

4.3. try...catch şi throw

Aici sunt date scheletele doar spre referinţă. Le vom trata în detaliu într-o altă lucrare.

4.3.1. try...catch simple pentru excepţii

```
try
{
    . . . // instrucţiuni care pot arunca excepţii
}
catch (tip-excepţie x)
{
    . . . // instrucţiuni de tratare a excepţiei
}
```

4.3.2. throw

throw *obiect-excepţie*;

5. Mersul lucrării

Scrieţi scurte programe în Java pentru:

5.1. Orice număr natural par mai mare decât 2 se poate scrie ca sumă a două numere prime – conjectura lui Goldbach. Scrieţi un program Java care să verifice această conjectura pentru numere situate între *m* şi *n*.

5.2. Implementati metoda Newton de gasire a radacinii patrate a unui polinom de grad 2 de forma $f(x) = ax^2 + bx + c = 0$. Metoda Newton gaseste o solutie aproximativa a ecuatiei, folosind un algoritm iterativ:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Solutia este gasita atunci cand algoritmul converge ($x_{n+1} - x_n < \varepsilon$)
Pornind de la o valoare initiala x_0, gasisti solutia polinomului, pentru diferite valori ale lui $\varepsilon \in [0.01,\ 0.000001]$. Afisati solutia gasita si numarul de iteratii (n).

5.3. Fiind data o variabila String s, determinati:
- numarul de consoane si vocale
- indicii pentru o vocala introdusa din linia de comanda.

5.4. Calculati şansele de câştig la loterie (6 din 49).

5.5. Simulati extragerea numerelor (pentru generarea numerelor aleatoare folosiţi Math.random)

5.6. Afişaţi în ordine crescătoare/descrescătoare numerele extrase fără a face sortări sau a folosi tablouri. (Sugestie: folosiţi împachetarea/despachetarea/extragerea de biţi stocaţi într-un long)