

[Clasa String](#)

[Scrieri cu format*](#)

[Clasa Scanner](#)

[Exerciții și probleme \(temă\)](#)

Clasa String

În Java șirurile de caractere sunt **obiecte** ale clasei `String` din pachetul `java.lang`. Un literal de tip `String` este o secvență de caractere între ghilimele

```
String s = "Un sir de caractere";
```

Amintim o serie de **metode** utile în lucrul cu șiruri de caractere din clasa `String`

- **Determinarea lungimii unui șir:** `length`

- **Accesarea unui caracter din șir:** `charAt`

Spre exemplu, următoarea secvență de cod afișează caracterele din șirul `s` câte unul pe linie

```
String s="Un sir de caractere";
for(int i=0;i<s.length();i++)
    System.out.println(s.charAt(i));
```

- **Căutarea unui caracter sau a unui șir de caractere într-un șir:** `indexOf`, `lastIndexOf` (returnează poziția, -1 dacă nu apare)

- `indexOf (int ch)`
- `indexOf (int ch, int index)`
- `indexOf (String str)`
- `indexOf (String str, int index)`

```
String s = "maama";
int poz = s.indexOf('a');
while(poz != -1){
    System.out.printf("%d ",poz);
    poz = s.indexOf('a',poz+1);
}
```

- **Accesarea unui subșir dintr-un șir:** `substring`

```
String s1 = new String("abcdefg");
System.out.println(s1.substring(4)); //efg
System.out.println(s1.substring(1,5));
//de la 1 la 4, nu la 5-> bcde
```

- **Egalitatea a două șiruri:**

Operatorul `==` testează dacă două referințe indică același obiect. În particular, dacă `s1` și `s2` sunt două variabile de tip `String`, atunci `s1==s2` este `true` doar dacă `s1` și `s2` sunt referințe către același șir (nu dacă șirurile sunt egale în sens lexicografic).

 **Exemplu** Următoarea secvență de cod

```
String s1 = "a b", s2;
char c1 = 'a', c2 = 'b';
s2 = c1+" "+c2;
System.out.println(s1+"/"+s2);
System.out.println(s1==s2);
```

va afișa


a b/a b

false


Pentru a testa egalitatea se folosește metoda `equals` (sau `equalsIgnoreCase` pentru a nu diferenția literele mari de mici):

```
System.out.println(s1.equals(s2));
```

va afișa `true`

 **Observație:** Metoda `equals` este moștenită din clasa `Object` există metoda `equals`. Orice clasă extinde direct sau indirect `Object`, deci moștenește această metodă. Metoda `equals` a clasei `Object` verifică dacă obiectul referit de parametru este același cu cel curent (ca și `==`). Într-o clasă putem suprascrie această metodă pentru a defini modul în care se face testul de egalitate pentru obiecte aparținând acestei clase.

- **Compararea a două șiruri lexicografic:** se face cu metoda `compareTo` (sau `compareToIgnoreCase`)
- **Concatenarea de șiruri** se face folosind operatorul `+`. Operatorul `+` este foarte flexibil, permițând concatenarea la un obiect de tip `String` a obiectelor de orice alt tip (**se apelează implicit metoda `toString()`** a obiectului respectiv) sau a unei variabile de tip primitiv. Mai mult, dacă oricare dintre operanzi este de tip `String`, operatorul `+` îi convertește și pe ceilalți la tipul `String`. Rezultatul operației de concatenare este un șir nou (complet distinct de cele concatenate).

 **Exemplu:** Ce afișează următoarea secvență de cod? Modificați această secvență pentru a afișa rezultatul corect.

```
System.out.println("1+1="+1+1);
```

Valoarea unui șir de caractere de tip **`String`** nu poate fi modificată după creare (de exemplu nu se poate modifica un caracter, sau un subșir al șirului). Dacă este necesară și modificarea șirului de caractere se pot utiliza clasele `StringBuilder` sau `StringBuffer` (dacă lucrăm cu threaduri) din pachetul `java.lang`.

Scrieri cu format*

Pentru scrieri cu format se poate folosi o formă a metodei `printf` a clasei `PrintStream`

(`System.out` este un obiect de tip `PrintStream`).

```
printf(String format, Object ... args)
```

Parametrul `format` este un șir de caractere care conține cel puțin câte o specificare a formatului pentru fiecare argument care urmează. Această specificație are forma generală

```
%[argument_index$][flags][width][.precision]conversion
```

(parantezele drepte au semnificația de opțional)

Opțiunile din forma generală pot lua următoarele valori:

`conversion` – un caracter ce specifică tipul, de exemplu:

- ‘d’, ‘o’ și ‘x’ se folosesc pentru întregi și arată că reprezentarea valori este în baza 10, 8, respectiv 16 .
- ‘f’, ‘g’ și ‘a’ se folosesc pentru valori reale și arată că reprezentarea va fi în notație zecimală, științifică sau hexazecimală cu exponent.
- ‘c’ pentru caracter; ‘s’ pentru șir de caractere; ‘b’ pentru boolean (afișat `true` sau `false`)

`argument_index` – un întreg reprezentând numărul argumentului care urmează parametrului de format. De exemplu “1\$” se referă la primul argument. Se poate folosi și simbolul ‘<’ în loc de o secvență de tipul `număr$`, indicând faptul că se folosește același argument ca și la specificarea anterioară.

`flags` – este un set de caractere care modifică formatul de ieșire. De exemplu

- ‘+’ forțează scrierea semnului pentru valori numerice
- ‘0’ forțează completarea valorilor numerice cu 0
- ‘-’ arată că argumentul va fi aliniat la stânga

`width` – numărul minim de caractere pe care va fi scrisă ieșirea

`precision` – folosit de exemplu pentru a specifica numărul de zecimale pentru valori reale.

Exemplu

```
class ExpPrintf{
    public static void main(String s[]){
        String s1="ab"; int x=3; float f1=1.2345f,f2=3f;
        System.out.printf("Un intreg %d",x); System.out.println();
        System.out.printf("Un intreg scris pe sase caractere aliniat la
dreapta %6d",x);System.out.println();
        System.out.printf("Numarul PI cu 5 zecimale %.5f\n",Math.PI);
        System.out.printf("Numarul PI cu 5 zecimale scris pe zece
caractere %10.5f",Math.PI); System.out.println();
        System.out.printf("primul numar real %f,al doilea numar %f \nal
doilea numar cu semn %<+f, primul numar cu 2 zecimale %1$.2f\n",f1,f2);
        System.out.printf("Sir aliniat la dreapta %10s, la stanga %1$-10s
sirul %1$s",s1);
    }
}
```

Clasa Scanner

Example

1. În următorul exemplu sunt citite două numere întregi de la tastatură și se afișează suma lor

```
import java.util.Scanner;
class ExpScanner{
    public static void main(String arg[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Introduceti doua numere intregi: ");
        int x=sc.nextInt();
        int y=sc.nextInt();
        System.out.println("Suma lor este "+(x+y));
    }
}
```

2. Să presupunem că dorim să citim un întreg. În caz că valoarea introdusă de utilizator nu reprezintă un întreg, îi permitem utilizatorului să reintroducă valoarea. Utilizatorul va putea reintroduce valoarea de un număr maxim de ori.

```
import java.util.Scanner;
class TestScan{
    static final int NR_MAX=3;
    public static void main(String arg[]){
        int i,k=0;
        Scanner sc=new Scanner(System.in);
        System.out.print("Introduceti un intreg: ");
        i=0;
        while(i<NR_MAX)
            if(sc.hasNextInt()){
                k=sc.nextInt();
                i=NR_MAX;
            }
            else{
                String s=sc.next();
                System.out.print(s+" nu este o valoare intreaga.");
                i++;
                if(i<NR_MAX)
                    System.out.print("Incercati din nou: ");
                else
                    System.exit(1);
            }
        System.out.printf("Ati introdus intregul %d",k);
    }
}
```

3. Având o clasă `Muchie` cu câmpurile reprezentând vârful inițial, vârful final și costul muchiei și un fișier cu următoarea structură: pe prima linie avem numărul de muchii, iar pe următoarele câte trei numere reprezentând vârful inițial, vârful final și costul unei muchii, ca de exemplu:

```
5
2 3 10.5
1 3 4
1 2 17.23
1 4 20
3 4 0.5
```

să se construiască un vector de muchii cu datele citite din fișier.

```
import java.util.Scanner;
import java.util.InputMismatchException;
import java.util.NoSuchElementException;
//import java.util.Locale;
import java.io.FileNotFoundException;

class Muchie{
    int vi,vf;
    double cost;
    Muchie(int vi,int vf, double cost){
        this.vi=vi;
        this.vf=vf;
        this.cost=cost;
    }
    public String toString(){
        return "("+vi+", "+vf+") "+cost;
    }
}

class TestScanF{
    public static void main(String arg[]){
        int i,n;
        Muchie muchii[];
        try{
            Scanner scFisier=new Scanner(new java.io.File("muchii.in"));
            //scFisier.useLocale(Locale.ENGLISH);
            n=scFisier.nextInt();
            muchii=new Muchie[n];
            for(i=0;i<n;i++){
                muchii[i]=new Muchie(scFisier.nextInt(), scFisier.nextInt(),
                                     scFisier.nextDouble());
            }
            scFisier.close();
            for(i=0;i<n;i++){
                System.out.println(muchii[i]);
            }
        } catch (FileNotFoundException fnf){ //obligatoriu
            System.out.println("Fisier inexistent.");
        }
    }
}
```

```

        catch(InputMismatchException im){ //optional
            System.out.println("Date de tip incorect.");
        }
        catch(NoSuchElementException nse){ // optional
            System.out.println("Nu exista informatii despre toate muchiile.");
        }
    }
}

```

Observatii:

1. Nu este obligatoriu să tratăm erorile `InputMismatchException` și `NoSuchElementException` aruncate de metodele `nextInt()` și `nextDouble()`. În acest caz puteam folosi metodele `hasNextInt()` sau `hasNextDouble()` ale clasei `Scanner` dacă vroiam să ne asigurăm ca datele citite sunt corecte.
2. Este obligatoriu să tratăm eroarea `FileNotFoundException`, altfel vom avea eroare la compilare

Exerciții și probleme

Exerciții:

1. Scrieți o aplicație Java care împarte o propoziție dată în cuvinte (cuvintele se pot separa prin spațiu, virgulă sau punct) și afișează cuvintele din propoziție ordonate lexicografic. Pentru a împărți propoziția în cuvinte folosiți:
 - a) clasa `Scanner`
 - b) o metodă utilă din clasa `String`
 Pentru sortare:
 - a) implementați o metodă de sortare la alegere
 - b) folosiți o metodă din clasa `Arrays`
2. Se citesc două șiruri de caractere. Să se determine numărul de apariții ale primului șir în cel de al doilea.
3. Să se creeze o clasă pentru lucru cu matrice triunghiulară care are două câmpuri: un număr întreg `n` și un tablou triunghiular `a` de numere reale (!care va fi alocat triunghiular), precum și două metode:
 - o metodă pentru citirea matricei inferior triunghiulară `a` de dimensiune `n`
 - o metodă de afișare a acestei matrice; matricea se va afișa sub formă triunghiulară, iar elementele matricei se vor afișa cu două zecimale.
 Să se scrie o clasă principală în care să se creeze și să se afișeze o matrice inferior triunghiulară folosind această clasă.

4. Dat fisierul „complex.in” care conține pe prima linie un număr n și pe următoarele n linii câte un număr complex dat prin partea reală și imaginară, creați un tablou de numere complexe (de tipul `Complex` implementat anterior).

a) Afișați numerele din tablou și modulul fiecăruia folosind instrucțiunea `for` pentru colecții

b) Adunați la primul număr din tabloul obținut celelalte numere și afișați numărul rezultat.

Exemplu de fișier de intrare „complex.in” :

```
3
1 4
2.5 7
0 9
```

5. Ce afișează următorul cod? Justificați.

```
class ExpStatic{
    static int nr=1;
    int x=1;
    static void cresteNr(){ nr++; }
    void cresteX(){ x++; } //NU static
    void afis(){ System.out.println(nr+" "+x ); }
    public static void main(String s[]){
        ExpStatic.cresteNr();
        System.out.println(ExpStatic.nr);
        ExpStatic ob1,ob2;
        ob1=new ExpStatic();
        ob2=new ExpStatic();
        ob1.cresteNr();
        ob1.cresteX();
        ob1.afis();
        ob2.afis();
    }
}
```



Probleme (TEMĂ)

1. **3-SUM (3p).** Implementați o soluție cât mai eficientă ($O(n^2)$) pentru problema 3-SUM: **Să se afișeze tripletele (distincte) de elemente ale unui vector de numere întregi care au suma 0** (de preferat fără a folosi alte structuri de date în afară de vectorul citit). Exemplu:

date.in	date.out
8	(-5, 2, 3)
3 1 2 -5 -2 10 7 3	(-5, -2, 7) nu neapărat în această ordine.

2. **Coadă (1p).** Scrieți o clasă pentru o coadă (memorată înlănțuit) cu metode pentru operațiile de bază: adăugare, eliminare, interogarea elementului curent (peek), toString (v. și <https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html> și <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html>); metodele se vor numi ca în interfața Queue din Java. Adăugați în clasă și o metodă pentru accesarea elementului de pe poziția i.

Observație. În laboratoarele următoare vom modifica această clasă astfel încât proiectarea și funcționalitatea ei să devină similară cu cea a colecțiilor deja implementate în Java, ca spre exemplu ArrayDeque. Încercați încă de acum o proiectare cât mai corectă a clasei.

3. **Graf (2p).** Scrieți o clasă pentru graf neorientat memorat prin liste de adiacență, care să conțină o metodă pentru afișarea listelor de adiacență și o metodă pentru parcurgerea în lățime a grafului. **Pentru memorarea listelor de adiacență și pentru coada necesară la parcurgerea în lățime se va folosi clasa implementată la 2.** În fișierul de intrare se vor da numărul de vârfuri, numărul de muchii, muchiile grafului (extremitățile lor) și vârful de start; se vor afișa listele de adiacență și parcurgerea în lățime din vârful de start

date.in	date.out
4 3	1: 3
1 3	2: 3
2 3	3: 1 2 4
3 4	4: 3
1	1 3 2 4

4. **Criptare xor șiruri de caractere XOR-cryption (3p).**

a) În fișierul date.in se găsește scris un text alcătuit din una sau mai multe fraze. Scrieți un program care citește acest text iar apoi citește de la tastatură un cuvânt ce va fi folosit pe post de cheie și care criptează acest text folosind criptarea XOR cu cheia citită. **Criptarea XOR** constă în următoarele: dacă n este lungimea cheii, pentru fiecare al k-lea caracter din text și al (k%n)-lea caracter din cheie se aplica operatorul XOR asupra reprezentării binare a celor două caractere, și, în final, se scrie numărul obținut urmat de spațiu în fișierul de ieșire date.out.

Exemplu: pentru cheia caine

date.in	date.out
Hello World!	43 4 5 2 10 67 54 6 28 9 7 64 73 100 45
How are you?	12 22 73 15 23 6 65 16 1 16 92

Explicații: $H \oplus c = 43$, $e \oplus a = 4$, $l \oplus i = 5$, $l \oplus n = 2$, $o \oplus e = 10$, $\text{ } \oplus c = 67$, $W \oplus a = 54$ etc...

b) Să se efectueze procedeul invers. Se citește din fișierul date.out o serie de numere întregi separate prin spațiu reprezentând o codificare a unui text. De la tastatură se citește un cuvânt cheie. Să se scrie într-un fișier rezultatul obținut prin decriptarea secvenței de numere folosind cheia citită

Exemplu: pentru cheia caine

date.in	date.out
43 4 5 2 10 67 54 6 28 9 7 64 73 100 45 12	Hello World!
22 73 15 23 6 65 16 1 16 92	How are you?