Lucrarea 3

Şiruri de caractere. Compararea obiectelor. Clase înfăşurătoare

Cuprins

Clasa *String*

Clasa *String* este una dintre clasele predefinite, fundamentale, care modelează lucrul cu şiruri de caractere. Modul cel mai simplu de manipulare a obiectelor de tip *String* îl constituie constantele *String*, care sunt şiruri de caractere încadrate de ghilimele. Orice referire a unei asemenea constante în cadrul unui program *Java* are ca efect crearea unui obiect *String* care conţine secvenţa de caractere respectivă.

Clasa *String* are mai mulţi constructori:

- un constructor care are ca parametru o referinţă la *String*. Noul obiect creat va avea acelaşi conţinut ca şi obiectul indicat de parametru:

```
String s1 = "un sir";
String s2 = new String(s1);
```

- un constructor *no-arg*, care crează un şir vid:

```
String svid = new String( );
```

- un constructor care are ca parametru un tablou cu elemente de tip char:

```
char[ ] tc = {'a','b','c'};
String stc = new String(tc);
```

Clasa String oferă un set de metode pentru prelucrarea unor şiruri existente:

```
String toUpperCase();
String toLowerCase();
String substring(int);
String substring(int, int);
```

Metodele clasei String includ şi operaţii care permit consultarea şirurilor de caractere, respectiv, obţinerea de informaţii despre conţinutul lor:

```
int length();
```

```
char charAt(int);
int indexOf(char);
int indexOf(String);
int lastIndexOf(char);
int lastIndexOf(String);
```

O proprietate foarte importantă a metodelor clasei *String* este aceea că ele nu modifică NICIODATĂ obiectul receptor, ci creează obiecte noi în care este inclus şirul de caractere modificat. În Java, operaţia de concatenare se realizează cu ajutorul operatorului „+“ . Va fi creat unu obiect nou, al cărui conţinut va fi şirul rezultat prin concatenare.

```
String s1 = "codul ";
String s2 = "penal";
String s3 = s1 + s2;
```

Operaţia de concatenare suportă următoarele precizări:

- clasa *String* este singura asupra căreia se poate aplica operatorul „+“. Tipurile primitive numerice şi tipul *char* mai au definit acest operator (cu altă semnificaţie);
- într-o expresie de concatenare pot să apară şi valori ale tipurilor primitive,acestea fiind automat convertite la *String*;

```
String s2 = "x = " +8.5;
```

- într-o expresie de concatenare pot să apară ca operanzi şi obiecte ale altor clase decât *String*, cu condiţia ca în clasele respective să fie definită metoda *toString* care să genereze obiecte *String*.

Obiecte *String* şi tablouri de caractere

Deşi, aparent, un şir de caractere şi un tablou de caractere (tipul *char[]*) modelează acelaşi lucru (o secvenţă de caractere), cele două tipuri se comportă diferit. Unui obiect *String* nu i se poate modifica conţinutul, unui tablou de caractere i se poate modifica oricare element. Pentru a obţine caracterul aflat la o anumită poziţie *pos* într-un *String*, *s*, nu putem folosi notaţia *s[pos]* ca pentru tablouri, ci trebuie utilizată metoda *charAt(pos)*. Metoda *toCharArray()* creează un tablou ale cărui elemente sunt caracterele şirului de caractere sursă. Operaţia inversă se realizează cu ajutorul unui constructor al clasei *String*.

Compararea obiectelor

Una dintre operaţiile cele mai frecvente de comparare a obiectelor este testarea egalităţii. În Java această operaţie poate avea două sensuri:

- **testarea identităţii**: în acest caz ceea ce se compară sunt două referinţe de obiect, iar testul presupune a verifica dacă cele două referinţe indică unul şi acelaşi obiect. Testarea identităţii se realizează cu ajutorul operatorului ==

- **testarea echivalenţei**: în acest caz operanzii sunt două obiecte propriu-zise şi testul presupune a verifica dacă cele două obiecte au conţinut asemănător. De regulă, pentru testarea echivalentei se utilizează o metodă denumită *equals()*, operanzii fiind un

obiect receptor şi unul dat ca parametru metodei. Această metodă este definită în clasa *Object* şi este moştenită de toate clasele.

Definiţia metodei *equals()*, aşa cum apare ea în clasa *Object*, pentru majoritatea claselor utilizatorului nu este satisfăcătoare. Ca urmare, programatorul va trebui să-şi definească propriile variante de *equals()*, acolo unde este cazul. Pentru clasa *Punct*, o posibilă implementare a metodei *equals()* ar fi:

```
class Punct{
        private int x,y;
        public Punct(int x, int y){
              this.x = x;
              this.y = y;
        }
        public boolean equals(Punct p) {
             return ((x==p.x) && (y==p.y));
        }
}
class ClientPunct {
        public static void main(String []args ) {
             Punct p1 = new Punct(1,2);
             Punct p2 = new Punct(1,2);

             Punct p3 = p1;
             boolean t1 = (p1==p2); //test de identitate; rezultat: false
             boolean t2 = p1.equals(p2); //test de echivalenta; rezultat: true
             boolean t3 = (p1==p3); //test de identitate; rezultat: true

             System.out.println(t1);
             System.out.println(t2);
             System.out.println(t3);.
        }
}
```

Compararea şirurilor de caractere

Regulile prezentate mai sus se aplică şi asupra clasei *String*. Ca atare, se va utiliza operatorul == pentru a testa identitatea obiectelor *String* şi metoda *equals()* pentru a testa egalitatea şirurilor de caractere conţinute de obiectele String.

```
class oClasa {
        public void oMetoda( ) {
             String s1 = new String("ababu");//se aloca in zona de memorie heap string-ul cu
                                             //valoarea ”ababu” si se returneaza o referinta
                                             //catre string-ul alocat
             String s2 = new String("ababu");");//se aloca in zona de memorie heap string-ul
cu
```

```
                                                    //valoarea ”ababu” si se returneaza o referinta
                                                   //catre string-ul alocat
            String s3 = s1;
            boolean t1 = (s1==s2); //test de identitate;  rezultat: false
            boolean t2 = s1.equals(s2); //test de echivalenta; rezultat: true

            boolean t3 = (s1==s3); //test de identitate; rezultat: true

            String s4 = "ababu"; //Daca in depozitul de string-uri al masinii virtuale se gasete
                                 //stringul ”ababu” se returneaza o referinta catre el. Daca nu
                                 //se intruce stringul ”ababu” in depozitul de stringuri si apoi
                                 //se returneaza o referinta catre el, preluata in s4
            String s5 = "ababu";//s5 va indica spre adresa string-ului ”ababu” care se gaseste
                                //in depozitul de string-uri al masinii virtuale
            boolean t4 = (s4==s5); //test de identitate;  rezultat: true pentru ca cele doua
                                   //string-uri vor indica spre aceeasi adresa

            //. . .
        }
}
```

Se observă că referinţele *s1* şi *s2* indică obiecte distincte, deşi ele au fost iniţializate cu acelaşi şir de caractere.

Pe lângă metoda *equals()*, clasa *String* mai conţine şi alte metode de comparare:

- metoda *equalsIgnoreCase()*, care este similară cu *equals()*, dar la comparare nu diferenţiază literele mari de cele mici.
- metoda *compareTo()*, care primeşte ca parametru o referinţă *String* şi returnează ca rezultat un număr întreg a cărui valoare este:
    - negativă, dacă şirul din obiectul receptor este mai mic decât şirul din obiectul parametru;
    - nulă, dacă cele două şiruri sunt egale;
    - pozitivă dacă şirul din obiectul receptor este mai mare decât şirul din obiectul parametru;

Metoda *compareTo()* poate fi folosită pentru realizarea sortării alfabetice a şirurilor de caractere.

Conversii între tipul *String* şi tipurile primitive

Într-un program *Java*, conversiile între tipul *String* şi tipurile primitive sunt foarte necesare, în primul rând pentru că datele de intrare ale programului sunt receptate în majoritatea cazurilor sub formă de şiruri de caractere, deoarece în *Java* nu există ceea ce în *C* se numea funcţie de citire formatată (*scanf*). În ceea ce priveşte conversiile, în Java există o regulă care spune că întotdeauna tipul spre care se face conversia trebuie să aibă metoda necesară conversiei. În acest sens, clasa *String* este dotată cu metode numite *valueOf()* care asigură conversiile de la tipurile primitive spre *String*.

```
class oClasa {
        public void oMetoda( ){
              int a = 4;
              double b = 13.2;
              boolean c = true;
              String s1 = String.valueOf(a);
              String s2 = String.valueOf(b);
              String s3 = String.valueOf(c);
        //. . .
        }
}
```

valueOf() este o metodă statică, supraîncărcată astfel încât să admită ca parametri valori ale tuturor tipurilor primitive. Efectul metodei este crearea unui obiect *String* al cărui conţinut este obţinut prin conversia la şir de caractere a valorii parametrului. Ca rezultat, metoda returnează o referinţă la obiectul *String* creat. Pentru a realiza conversiile în sens invers, adică de la *String* spre tipurile primitive se utilizează metode ale unor clase speciale, numite clase înfăşurătoare *(wrapper classes)*.

Clase înfăşurătoare

Tuturor tipurilor primitive din limbajul *Java* le corespunde câte o clasă înfăşurătoare. Aceste clase au fost definite din două motive:

- să înglobeze metode de conversie între valori ale tipurilor primitive şi obiecte ale altor clase, precum şi constante speciale legate de tipurile primare;
- să permită crearea de obiecte care să conţină valori ale tipurilor primare, şi care apoi să poată fi utilizate în cadrul unor structuri unde nu pot să apară decât obiecte, nu date simple.

Clasele înfăşurătoare sunt definite în pachetul *java.lang* şi ele sunt:

```
Integer pentru int
Short pentru short
Byte pentru byte
Long pentru long
Float pentru float
Double pentru double
Boolean pentru boolean
Character pentru char
Void pentru void
```

Exceptând clasa Void, restul claselor înfăşurătoare conţin următoarele metode:

- câte un constructor care acceptă ca parametru o valoare a tipului primitiv corespondent.

```
Integer a = new Integer(5);
Double b = new Double(12.13);
```

- câte un constructor care acceptă ca parametru o referinţă *String* (excepţie face clasa *Character* care nu are acest constructor). Se presupune că şirul respectiv conţine o secvenţă compatibilă cu tipul primitiv corespondent.

```
Integer a = new Integer("5");
Boolean b = new Boolean("true");
```

- o metodă statică *valueOf()* care acceptă ca parametru un String şi returnează un obiect al clasei înfăşurătoare, convertind conţinutul şirului; apelul acestei metode este echivalent cu a executa un *new* cu constructorul descris mai înainte.

```
Integer a = Integer.valueOf("5");
Double b = Double.valueOf("-8.12");
```

- o metodă *toString()* care realizează practic conversia de la clasa înfăşurătoare la *String*

De multe ori este necesar ca datele conţinute de obiecte să fie convertite la şiruri de caractere în vederea participării obiectelor respective ca operanzi în expresii de manipulare a şirurilor. De exemplu, funcţiile *print()* / *println()* necesită ca parametru o referinţă *String*. Convertind un obiect la *String*, am putea să afişăm acel obiect cu ajutorul funcţiilor *print()/println()*. Programatorul trebuie să-si definească în clasele sale câte o metodă *toString()*, dacă doreşte să convertească obiectele respective la şiruri.

```
class Punct {
        //. . .
        public String toString( ) {
             String s = "("+String.valueOf(x)+","+String.valueOf(y)+")";
             return s;
        }
}
```

- o metodă *typeValue()*, unde *type* este numele tipului primitiv corespondent; această metodă returnează conţinutul obiectului ca valoare a tipului primitiv.

```
Integer a = new Integer("5");
int x = a.intValue( );
Double b = new Double("3.14");
double y = b.doubleValue( );
```

- o metodă *equals()* care realizează testul echivalenţei obiectelor.
- o metodă *compareTo()* care acceptă ca parametru un obiect al aceleiaşi clase înfăşurătoare ca şi obiectul receptor. Metoda returnează o valoare de tip *int* pozitivă, nulă sau negativă după cum conţinutul obiectului receptor este mai mare, egal sau mai mic decât cel al parametrului.

```
Integer a = new Integer(7);
Integer b = new Integer(10);
int x = b.compareTo(a); //val. x va fi pozitivă
```

Pentru conversiile de la *String* la tipurile primitive se utilizează metode statice ale claselor înfăşurătoare, si anume:

- în cazul claselor care reprezintă tipuri numerice există metodele *parseType()*, unde *Type* este numele tipului primitiv corespondent scris cu prima literă majusculă; metoda acceptă ca parametru şirul care trebuie convertit şi returnează valoarea corespunzătoare, în reprezentarea tipului primitiv respectiv;
- în cazul tipului boolean exista metoda *getBoolean()* care funcţionează similar metodei *parseType()*.

În cazul tipurilor numerice, dacă şirul care se converteşte nu conţine o secvenţă în concordanţă cu tipul primitiv spre care se face conversia, programul va fi întrerupt cu un mesaj de eroare corespunzător (excepţie *NumberFormatException*).

În continuare, se va prezenta un program în care va fi ilustrat conceptul de **introducere interactivă de date**, care presupune că utilizatorul trebuie să introducă anumite date de la tastatură, Pentru aceasta, este necesar ca înaintea operaţiilor de citire, în program să existe operaţii de afişare a unor mesaje de genul: „Introduceţi denumirea:“ sau „Daţi cantitatea:“. Programul considerat realizează completarea unui tablou de numere întregi, citind numerele de la tastatură, apoi afişează pe ecran tabloul ordonat crescător.

```
import java.io.*;

public class Ordonare{
        public static void main (String [] args) throws IOException{
                BufferedReader flux_in=new BufferedReader (new
                                        InputStreamReader(System.in));
                int[] tab = new int[10];
                int i, aux;

                String linie;
                boolean sw = true;

                for(i=0;i<tab.length;i++){
                        System.out.println("Dati elementul de pe pozitia "+(i+1));
                        linie=flux_in.readLine();
                        tab[i]=Integer.parseInt(linie);
                }

                while(sw){
                     sw=false;
                     for(i=o;i<tab.length-1;i++)
                          if(tab[i]>tab[i+1]){
                                aux=tab[i];
                                tab[i]=tab[i+1];
                                tab[i+1]=aux;
                                sw=true;
                          }
                 }
```

```
            System.out.println("Tabloul ordonat este:");
            for(i=0;i<tab.length;i++)
                System.out.println(tab[i]+" ");
        }
}
```

Temă

1. Fişierul *cantec_in.txt* conţine versurile unui cântec la alegere. Să se scrie un program care creează fişierul *cantec_out.txt,* care conţine versurile cântecului original dar în plus în dreptul fiecărui rând sunt afişate numărul de cuvinte de pe rând şi numărul de vocale de pe fiecare rând. În dreptul rândurilor care se încheie cu o grupare de litere aleasă se va pune o steluţă (*). Rândurile pentru care un număr generat aleator este mai mic decât 0.1 vor fi scrise cu majuscule (se vor genera aleator numere între 0 şi 1). Se va defini o clasă Vers, care are o variabilă membră privată un șir de caractere reprezentând versul și se va dezvolta câte un operator pentru fiecare cerință de mai sus (o metodă care returnează numărul de cuvinte, o metodă care returnează numărul de vocale, etc).

2. Să se insereze într-o anumită poziție a unui șir de caractere, un alt șir. Datele vor fi preluate de la tastatură sau din fișier. Să se șteargă o porțiune a unui șir de caractere care începe dintr-o anumită poziție și are un anumit număr de caractere. Se recomandă utilizarea clasei *StringBuilder*.

3. Fișierul *judete_in.txt*, conține lista neordonată a județelor din țară. Să se încarce datele din fișier într-un tablou de *String*-uri și să se ordoneze acest tablou cu ajutorul metodei *sort()* din clasa *Arrays*. Să se returneze pe ce poziție se află în vectorul oronat un județ introdus de la tastatură. Se va utiliza metoda de căutare binară din clasa *Arrays*.

4. Fişierul *in.txt* conţine o listă cu melodiile preferate de un student. Pentru fiecare melodie se reţine: *nume_melodie; nume_artist; an_lansare; numar_vizualizări_youtube*. Să se realizeze un program care:

a) încarcă informaţiile despre melodii în program, într-un vector de obiecte. Se va crea clasa *Melodie* cu variabile membre private, constructor cu parametri, *gettere* pentru accesul variabilelor membre
b) afişează informaţiile melodiilor încărcate
c) afişează clasamentul melodiilor în ordine descrescătoare al numărului de vizualizări pe youtube
d) afişează toate melodiile unui artist