Poo Laboratoare[1]

Contents

Laborator2 **2**

1 Operatii cu şiruri de caractere **2**
1.1 Clasa String . . . 2
1.1.1 Instanţierea unui obiect **String** . . . 2
1.1.2 Câteva operaţii uzuale . . . 2
1.2 Clasa StringBuffer . . . 2
1.2.1 Instanţierea unui obiect **StringBuffer** . . . 2
1.2.2 Câteva operaţii uzuale . . . 3

2 Probleme de laborator **3**
2.1 Problema 1 . . . 3
2.2 Problema 2 . . . 3
2.3 Problema 3 . . . 3
2.4 Problema 4 . . . 4
2.5 Problema 5 . . . 4
2.6 Problema 6 . . . 4
2.7 Problema 7 . . . 4

[1] `http://www.google.ro/`

Laborator2

Programare Orientata pe Obiecte: Laborator 2

1 Operatii cu șiruri de caractere

1.1 Clasa String

În Java, **String** este o clasă elementară care oferă funcțiile de bază pentru manipularea șirurilor de caractere.

Ca utilizare, această clasă se folosește pentru compararea a două șiruri de caractere, extragerea de subșiruri, determinarea lungimii unui șir sau alte operații care vor fi descrise mai jos.

1.1.1 Instanțierea unui obiect String

```
String str = ‘‘text’’;
```

sau

```
String str = new String( ‘‘text’’ );
```

sau

```
char data[] = {’t’, ’e’, ’x’, ’t’}; String str = new String( data );
```

1.1.2 Câteva operații uzuale

- **str.charAt(** *1* **)** - întoarce caracterul de la poziția *1*
- **str.contact(** *str2* **)** - concatenarea de șiruri
- **str.equals(** *str2* **)** - verificarea egalității
- **str.indexOf(** *str2* **)** - determină poziția de început a subșirului *str2* în șirul *str1*
- **str.length()** - determină lungimea șirului

Operatoul + introduce posibilitatea concatenării de șiruri astfel:

```
str = str1 + str2;
```

1.2 Clasa StringBuffer

Este **asemănătoare cu clasa String**, doar că oferă operații mai complexe cu ajutorul cărora se poate modifica conținutul șirului.

1.2.1 Instanțierea unui obiect StringBuffer

```
StringBuffer str = new StringBuffer( ‘‘abc’’ );
```

sau

```
StringBuffer str = new StringBuffer()
```

1.2.2 Câteva operații uzuale

Fiind date

```
String str = ‘‘text’’;
StringBuffer sbuf = new StringBuffer();
```

iată câteva exemple:

- **sbuf.append(** *str* **)** - se adaugă șirul *str* la *sbuf*
- **sbuf.append(** *2* **)** - se adaugă reprezentarea ca șir de caractere a numărului *2*, la *sbuf*
- **sbuf.append(** *2, str* **)** - se inserează șirul *str* în *sbuf* începând de la poziția *2*
- **sbuf.replace(** *1, 3, str* **)** - se înlocuiesc caracterele de la pozițiile *1-3* cu șirul *str*
- **sbuf.delete(** *1, 3* **)** - se șterg caracterele de la pozițiile *1-3*

Pentru mai multe metode din clasele **String** și **StringBuffer** se va consulta API-ul Java din NetBeans sau adresa[2].

2 Probleme de laborator

2.1 Problema 1

(1 punct) Să se scrie un program pentru căutarea tuturor aparițiilor unui șir dat **s1** ca subșir într-un alt șir **s**. Se va afișa numărul de apariții.

```
Exemplu:
String s1 = ‘‘si’’;
String s = ‘‘sir1 si cu sir2 fac un sir3’’;
Rezultat:  4.
```

Se poate folosi metoda **indexOf()** din clasa **String** cu două argumente: șirul căutat și poziția din șirul destinație de unde începe căutarea.

```
Exemplu:
int pos = s.indexOf( s1, p );
```

2.2 Problema 2

1. *(1 punct)* Să se scrie o funcție pentru înlocuirea tuturor aparițiilor unui șir dat **s1** printr-un alt șir **s2** în cadrul unui șir **s**, folosind **numai** metode ale clasei **String**.

Funcția va fi verificată prin apelare în funcția **main()**.

2. *(1 punct)* Să se rescrie apoi funcția folosind metode din clasa **StringBuffer**.

2.3 Problema 3

(1 punct) Să se scrie un program pentru extragerea și afișarea cuvintelor dintr-un text introdus ca un șir constant. Un cuvânt este un șir delimitat prin spații (blancuri) de alte cuvinte. Se pot folosi următoarele metode din clasa **String**:

- **int indexOf(char c, int p)**;
- **String substring(int p1, int p2)**;

[2] http://java.sun.com/j2se/1.4.2/docs/api/

Observație! Căutați aceste funcții în API-ul Java pentru a vedea cum se folosesc.

2.4 Problema 4

(1 punct) Să se scrie un program pentru extragerea și afișarea cuvintelor dintr-un text introdus ca un șir constant. Se va folosi un obiect de tip **StringTokenizer**. Să se adauge un alt text care conține și alți separatori de cuvinte: virgula (','), punct ('.'), punct și virgulă (';'), două puncte (':'). Se va utiliza un alt constructor pentru obiectul analizor (StringTokenizer).

2.5 Problema 5

(1 punct) Să se scrie o funcție iterativă și una recursivă pentru determinarea prefixului comun maxim a două șiruri primite ca argumente.

```
Exemplu:
s1 = ‘‘obiectual’’
s2 = ‘‘obiectiv’’
Rezultat:  ‘‘obiect’’
```

2.6 Problema 6

(2 puncte) Să se scrie o funcție statică pentru ordonarea unui vector de șiruri. Comparați timpul de ordonare al unui vector de 1000 de șiruri cu timpul realizat de metoda **Arrays.sort()** (pachetul **java.util**).

Se va defini și folosi o funcție statică pentru generarea unui vector de șiruri de lungime dată, cu rezultat vector. Header-ul va fi de forma **String [] build(int n)**;

Pentru măsurarea duratei se va folosi metoda statică **long System.currentTimeMillis()** care are ca rezultat ora curentă exprimată în milisecunde (față de ora zero).

Dublați dimensiunea vectorului și observați cum se modifică timpul de sortare.

2.7 Problema 7

(2 puncte) Să se scrie o funcție pentru ordonarea liniilor unei matrici de șiruri după valorile dintr-o coloană dată.

```
Exemplu de folosire:
  String a[][] =
  for( int i = 0; i < 3; i ++ ){ // succesiv dupa coloanele 0,1,2
       String b[][] = sortBy( a, i ); // ordoneaza matricea a cu rezultat in b
       print( b ); // afisare matrice ordonata
  }
```