

CLASA String

Metode și exemple



Crearea unui șir

- | Constructorul implicit crează un șir vid:

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

- | `String str = "abc";`

este echivalent cu:

```
char data[] = {'a', 'b', 'c'};
```

```
String str = new String(data);
```

OBS. Dacă elementele din vectorul `data` sunt modificate după crearea șirului `str`, atunci aceste modificări nu apar în acest șir.

- | Construirea unui șir pe baza altui șir:

```
String str2 = new String(str);
```

Operații cu șiruri

- I Metoda **length()** returnează lungimea unui șir
Ex: `System.out.println("Hello".length());` // afișează 5

- I Operatorul '+' este utilizat pentru concatenarea a două sau mai multe șiruri

Ex: `String myname = "Harry";`
`String str = "My name is" + myname + ".";`

- I Caracterele dintr-un șir pot fi accesate astfel:

`public char charAt(int index);`

Returnează caracterul din poziția index. Domeniul de indexare este de la 0 la `length() - 1`.

Ex. `char ch;`

`ch = "abc".charAt(1);` // `ch = "b"`

Operații cu șiruri

- | **getChars()** - Copiază caracterele din șirul sursă în șirul destinație

public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

- srcBegin – indexul primului caracter din sursă
- srcEnd – indexul ultimului caracter din sursă
- dst – vectorul destinație
- dstBegin – poziția de la care începe copierea în vectorul destinație

- | **equals()** – Compară două șiruri la egalitate

public boolean equals(String s2)

- | **equalsIgnoreCase()**- Compară două șiruri la egalitate fără să țină cont de litere mici sau mari

public boolean equalsIgnoreCase(String s2)

Operații cu șiruri

- I **startsWith()** – Testează dacă șirul începe cu prefixul specificat (la început sau dintr-o anumită poziție)

public boolean startsWith(String prefix)

Ex. "Figure".startsWith("Fig"); // true

public boolean startsWith(String prefix, int toffset)

Ex. "figure".startsWith("gure", 2); // true

- I **endsWith()** - Testează dacă șirul se termină cu sufixul specificat

public boolean endsWith(String suffix)

Ex. "Figure".endsWith("re"); // true

- I **compareTo()** – Compară două șiruri din punct de vedere lexicografic

Rezultatul este

- Negativ, dacă șirul precede șirul primit ca argument
- Zero, dacă șirurile sunt egale
- Pozitiv, dacă șirul urmează șirului primit ca argument

public int compareTo(String anotherString)

public int compareToIgnoreCase(String str)

Operații cu șiruri

- I **indexOf** – Caută prima apariție a unui caracter sau șir într-un alt șir. Returnează -1 dacă nu îl găsește.

```
public int indexOf(int ch)  
public int indexOf(String str)
```

```
Ex. String str = "How was your day today?";  
str.indexOf('o'); // 1  
str.indexOf("was"); //4
```

- I căutare începând de la o poziție specificată:

```
public int indexOf(int ch, int fromIndex)  
public int indexOf(String str, int fromIndex)
```

```
Ex. String str = "How was your day today?";  
str.indexOf('a', 6); //14  
str.indexOf("was", 2); //4
```

- I **lastIndexOf()** - Caută ultima apariție a unui caracter sau șir într-un alt șir, similar lui `indexOf`.

Operații cu șiruri

- I **substring()** – Returnează un nou șir care este un subșir al șirului sursă.

public String substring(int beginIndex)

Ex: "unhappy".substring(2); // returnează "happy"

public String substring(int beginIndex, int endIndex)

Ex: "smiles".substring(1, 5); // returnează "mile"

- I **concat()** – Concatenează șirul specificat la șirul sursă

public String concat(String str)

"to".concat("get").concat("her"); // returnează "together"

- I **replace()**- Returnează un nou șir în care toate aparițiile caracterului oldChar sunt înlocuite cu caracterul newChar.

public String replace(char oldChar, char newChar)

Ex. "mesquite in your cellar".replace('e', 'o');
returnează "mosquito in your collar"

Operații cu șiruri

- | **trim()** - Returnează șirul fără spațiile albe de la început și sfârșit

```
public String trim()
```

Ex. String s = " Hi Mom! ".trim();

S este acum "Hi Mom!"

- | **valueOf()** – Returnează reprezentarea sub formă de șir de caractere a argumentului

```
public static String valueOf(char[] data)
```

```
public static String valueOf(char c)
```

```
public static String valueOf(boolean b)
```

```
public static String valueOf(int i)
```

```
public static String valueOf(long l)
```

```
public static String valueOf(float f)
```

```
public static String valueOf(double d)
```


Operații cu șiruri

- | **toLowerCase():** Convertește toate caracterele la litere mici

```
public String toLowerCase()
```

- | **toUpperCase():** Convertește toate caracterele la litere mari

```
public String toUpperCase()
```

```
Ex: "HELLO THERE".toLowerCase();
```

```
    "hello there".toUpperCase();
```

Operații cu șiruri

public String[] split(String regex)

- împarte șirul în subsiruri folosind un șir de delimitatori (expresie regulată)
- regex – expresia regulată de delimitare
- returnează un vector de String-uri

Exemplu: pentru șirul "boo:and:foo" avem următoarele rezultate:

regex	rezultat
:	{ "boo", "and", "foo" }
o	{ "b", "", ":and:f" }

public String[] split(String regex, int limit)

- Limit: de câte ori se aplică expresia regulată

StringBuffer

- I Este asemănător unui obiect String doar că este modificabil în sensul că există metode pentru modificarea lungimii și a conținutului său.
- I Crearea unui obiect StringBuffer:
 - `StringBuffer()`
 - `StringBuffer(int size)`
 - `StringBuffer(String str)`
- I Principalele operații sunt adăugarea, inserarea și ștergerea.
- I Adăugarea:
 - `StringBuffer append(String str)`
 - `StringBuffer append(int num)`
- I Inserarea se face într-o anumită poziție.
 - `StringBuffer insert(int index, String str)`
 - `StringBuffer append(int index, char ch)`

Operații cu StringBuffer

- | **delete()** - șterge un subșir. Subșirul începe în poziția specificată și se termină în poziția index de final - 1 sau până la sfârșitul șirului dacă nu se specifică poziție de final.

`public StringBuffer delete(int start, int end)`

`public StringBuffer delete(int start)`

- | **replace()** – Înlocuiește un subșir din șirul sursă cu un alt șir

`public StringBuffer replace(int start, int end, String str)`

- | **substring()**

`public String substring(int start)`

- | **reverse()** – reversul șirului (ordine inversă a caracterelor)

`public StringBuffer reverse()`

- | **length()**

`public int length()`

Operații cu StringBuffer

- I **capacity()** – Returnează capacitatea curentă a StringBuffer-ului. Capacitatea este dimensiunea disponibilă pentru adăugarea de noi elemente

```
public int capacity()
```

- I **charAt()**

```
public char charAt(int index)
```

- I **getChars()**

```
public void getChars(int srcBegin, int srcEnd,  
char[] dst, int dstBegin)
```

- I **setLength()** – Setează lungimea pentru obiectul de tip StringBuffer.

```
public void setLength(int newLength)
```

Exemple: StringBuffer

```
StringBuffer sb = new StringBuffer("Hello");  
sb.length(); // 5  
sb.capacity(); // 21 (16 caractere sunt adăugate  
    dacă nu se specifică o dimensiune)  
sb.charAt(1); // e  
sb.setCharAt(1, 'i'); // Hillo  
sb.setLength(2); // Hi  
sb.append("l").append("l"); // Hill  
sb.insert(0, "Big "); // Big Hill  
sb.replace(3, 11, ""); // Big  
sb.reverse(); // giB
```

Clasa StringTokenizer

Utilizată pentru împărțirea în atomi lexicali (tokens):

```
public StringTokenizer(String str, String delim)
public boolean hasMoreTokens()
public String nextToken()
```

Exemplu:

```
StringTokenizer st = new StringTokenizer("this, is a
    test.", ", .");
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

Are următorul rezultat:

```
this
is
a
test
```