

Строки



Класс String

String – это один из самых часто используемых типов данных в Java, поэтому очень важно в совершенстве разобратся в принципах работы со строками!

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

Персистентность строк в Java

Стандартные строки в Java являются **персистентными** (constant, immutable), и после создания строки в ней уже не получится что-либо изменить. Причина – обеспечение целостности текстовых данных при работе приложения в режиме нескольких потоков. Все строковые методы возвращают новые строки, не меняя оригинальные данные.

Способы создания строк

```
String first = "text";
```

```
// this way is equivalent to:
```

```
char[] data = {'t', 'e', 'x', 't'};
```

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

```
System.out.println(first); // text
```

```
System.out.println(second); // text
```

Способы создания строк

```
String s1 = new String();
```

```
// создаётся пустая строка, s1 = "";
```

```
byte[] data = {65, 108, 101, 120};
```

```
String s2 = new String(data); // s2 = "Alex";
```

```
String s3 = new String(s2); // копия s2
```

```
String s4 = s2; // ссылка на s2!
```

Посимвольное чтение строки

```
String name = "Alexander";
```

```
for (int i = 0; i < name.length(); i++) {  
    System.out.print(name.charAt(i) + " - ");  
    // System.out.print(name[i]); // ERROR!!!  
    System.out.print(name.codePointAt(i) + ", ");  
}
```

```
// A - 65, l - 108, e - 101, x - 120, ...
```

Ввод, сравнение строк

```
String login = "Alex";
```

```
String user = new Scanner(System.in).next();
```

```
if (user.compareToIgnoreCase(login) == 0) {  
    System.out.println("Welcome, " + user + "!");  
} // лексикографическое сравнение
```

```
// compareTo – с учётом регистра, для паролей
```

Сравнение строк

```
String login = "Alex";
```

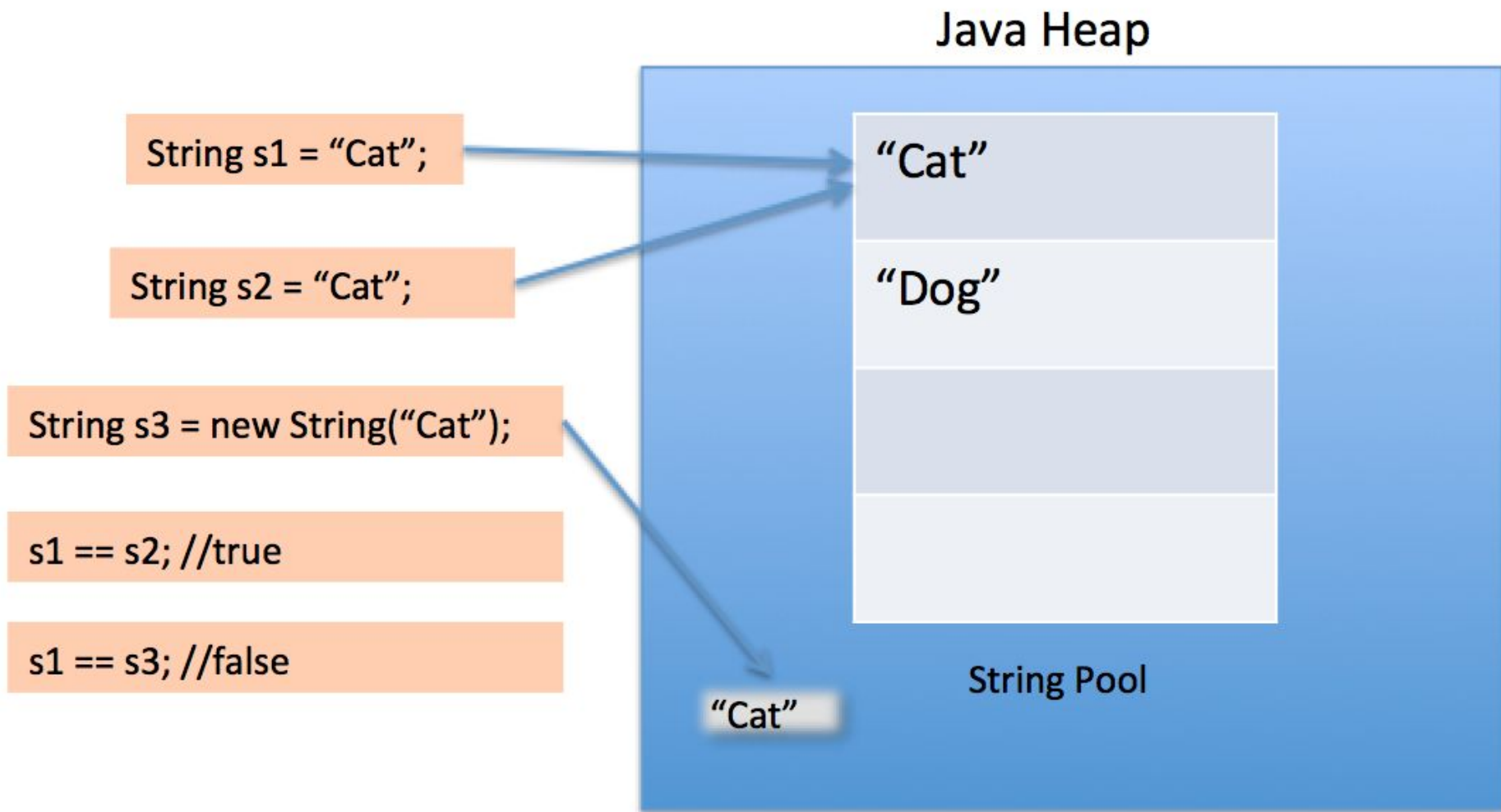
```
String user = new Scanner(System.in).next();
```

```
if (user.equalsIgnoreCase(login)) {  
    System.out.println("Welcome, " + user + "!");  
}
```

```
// equals – с учётом регистра!
```

```
// == никогда не применять!!!
```


Сравнение строк ==



Конкатенация строк

```
String name = "Alex";  
name = name.concat("ander");  
// name = name + "ander";  
// name += "ander";  
System.out.println(name); // Alexander  
// к строкам можно присоединять числа!
```



Преобразования чисел и строк

// число в строку

▲ There are multiple ways:

295

- `String.valueOf(number)` (my preference)
- `"" + number` (I don't know how the compiler handles it, perhaps it is as efficient as the above)
- `Integer.toString(number)`

✓ share improve this answer

edited Feb 21 '11 at 21:12

answered Feb 21 '11 at 20:45

// строку в число

▲

```
int foo = Integer.parseInt("1234");
```

1556

See the [Javadoc](#) for more information.

▼ (If you have it in a `StringBuffer`, you'll need to do `Integer.parseInt(myBuffer.toString());` instead).



Проверка наличия подстроки

```
String login = "Alexander";  
if (login.contains("and") &&  
login.endsWith("er")) {  
    System.out.println("OK");  
}
```

```
// startsWith – строка начинается с  
// isEmpty – проверка на пустоту
```

Поиск позиции подстроки

```
String login = "Alex - the best!";
```

```
System.out.print(login.indexOf('e')); // 2
```

```
System.out.print(login.indexOf('e', 4)); // 9
```

```
System.out.print(login.lastIndexOf('t')); // 14
```

```
System.out.print(login.lastIndexOf('t', 10)); // 7
```

```
System.out.print(login.indexOf('z')); // -1
```

```
System.out.print(login.indexOf("best")); // 11
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	l	e	x		-		t	h	e		b	e	s	t	!

Замена текста, формат

```
String str = "Alex - the best!";  
str = str.replaceAll("Alex", "Vasya");  
System.out.println(str); // Vasya - the best!  
// replaceFirst – замена первого вхождения
```

```
// форматирование:
```

```
String name = "Alex";  
int age = 27;  
String s = String.format("My name is %s, I am %d.", name, age);
```

Массив СИМВОЛОВ, подстрока

```
String str = "Alexander";
```

```
char[] data = str.toCharArray();
```

```
System.out.println(Arrays.toString(data));
```

```
// [A, l, e, x, a, n, d, e, r]
```

```
System.out.println(str.substring(4, 7));
```

```
// 4 – begin index, 7 – end index ("and")
```

Смена регистра

```
String str = "Alexander";
```

```
str = str.toLowerCase();
```

```
System.out.println(str); // alexander
```

```
str = str.toUpperCase();
```

```
System.out.println(str); // ALEXANDER
```


Trim, split

```
String str = "  A l e x  ";  
System.out.println(str); // __A_l_e_x__
```

```
str = str.trim();  
System.out.println(str); // A_l_e_x
```

```
String[] tokens = str.split(" ");  
System.out.println(Arrays.toString(tokens));  
// [A, l, e, x]
```

Класс StringBuffer

Класс **String** представляет собой неизменяемые последовательности символов постоянной длины, и частое использование объектов класса занимает много места в памяти.

Класс **StringBuffer** представляет расширяемые и доступные для изменений последовательности символов, позволяя вставлять символы и подстроки в существующую строку и в любом месте. Данный класс гораздо экономичнее в плане потребления памяти, и к тому же позволяет обращаться к строке из разных потоков.

Способы создания StringBuffer

<https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html>

```
StringBuffer sb = new StringBuffer();  
System.out.println(sb.capacity()); // 16
```

```
StringBuffer sb2 = new StringBuffer(50);  
System.out.println(sb2.capacity()); // 50
```

```
StringBuffer sb3 = new StringBuffer("Alex");  
System.out.println(sb3.capacity()); // 20
```

Некоторые методы StringBuffer

```
StringBuffer sb = new StringBuffer("Alex");  
//sb += "ander"; // error!  
sb.append("ander"); // 13 перегрузок!  
System.out.println(sb.charAt(5)); // n  
sb.setCharAt(3, 'X');  
System.out.println(sb); // AleXander  
sb.insert(0, "Mr. ");  
System.out.println(sb); // Mr. AleXander  
sb.reverse();  
System.out.println(sb); // rednaXeIA .rM
```

Некоторые методы StringBuffer

```
StringBuffer sb = new StringBuffer("Alexander");  
sb.delete(4, sb.length());  
System.out.println(sb); // Alex  
sb.deleteCharAt(0);  
System.out.println(sb); // lex  
sb.replace(1, 3, "ada sedan");  
System.out.println(sb); // lada sedan  
System.out.println(sb.substring(5)); // sedan  
  
// indexOf, trimToSize
```

Класс `StringBuilder`

Класс **`StringBuilder`** идентичен классу **`StringBuffer`** и даже обладает чуть большей производительностью, однако, он не синхронизирован, поэтому его нельзя эффективно использовать в тех случаях, когда к изменяемой строке обращаются несколько потоков.

Практика

- Ввести с клавиатуры строку текста, а затем один символ. Показать на экран индексы и количество совпадений (ищем вхождения символа в строку).