

STRINGS

Класс `String` нельзя расширить

Класс `String` не может быть наследован.

```
public final class String extends Object  
    implements Serializable, Comparable,  
    CharSequence
```

Доступ к элементам строки можно осуществить с помощью метода `charAt(int index)`

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

нумерация символов строки начинается с 0.

Замечание. В отличие от массивов, длина которых содержится в *поле* `length`, в классе `String` поле `length` отсутствует, длину строки (количество входящих в нее символов) возвращает метод `length()`.

```
public int length()
```

Перегрузка операции «+» для строк

В Java отсутствует перегрузка операций в смысле переопределения действия операций, но существует в смысле применения одного знака операции к разным типам данных.

Если в операции сложения участвует строка, то второй операнд всегда приводится к строке.

При конкатенации (сложении) строк будет создан новый объект типа String, т.к. строки - неизменяемые объекты.

Замечание. Нельзя изменить объект типа String, но можно изменить значение объектной переменной, которая *ссылается* на объект, заставив ее ссылаться на новый объект String.

```
String s = "asdf";  
s = "1234";  
s = "ABCD";
```

Получение строкового представления значений примитивных типов

Для того чтобы получить строковое представление значения примитивного типа следует воспользоваться статическим методом `valueOf`, который возвращает соответствующую строку.

Данный метод перегружен таким образом, что может принимать значения любых примитивных типов в качестве параметра.

Кроме этого метод может принимать в качестве параметра массив символов (`char`), который будет преобразован в строку.

```
String s = String.valueOf(0xFF); // s = "255"
```

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

```
s = String.valueOf(ch); // s = "abc"
```

```
s = String.valueOf(-34.4); // s = "-34.4"
```

Сложение объектных переменных, ссылающихся на null, со строкой

Если строка сложить с объектом, который ссылается на null, то вместо него будет подставлена строка "null".

```
String s = null;  
s = s + "34"; // s = "null34"
```

```
Integer x = null;  
s = "abc" + x; // s = "abcnull"
```


Строковые литералы (строки-константы)

Строковые литералы (строки-константы) представляют из себя совокупность символов заключенных в двойные кавычки и являются объектными переменными.

Два строковых литерала, состоящих из одного и того же набора символов ссылаются на один и тот же объект-строку.

```
String s1 = "abc";  
String s2 = "abc";  
boolean flag = (s1 == s2); // flag = true
```

Пул строк

JVM поддерживает пул строк.

Все строковые литералы при загрузке класса будут занесены в пул строк. Можно занести строку в пул с помощью метода `String#intern`.

Метод `intern`, просматривает пул строк и пытается найти, используя метод `equals`, строку, эквивалентную данной.

Если такая строка найдена, то возвращает на нее ссылку.

Если не найдена, то метод пулирует исходную строку и возвращает ссылку `this`.

Вариант	Строка в пуле?	Есть в пуле строка, равная по equals данной строке?	Результат выполнения метода intern
1	да	да, всегда есть - this	возвращает this
2	нет	да, например XXXX	возвращает ссылку на XXXX
3	нет	нет	пулирует this и возвращает this

Сравнение строк посимвольно

Для сравнения строк посимвольно следует использовать метод `equals` объекта `String`.

```
public boolean equals(Object object)
```

Для сравнения строк без учета регистра символов, из которых состоит строка, следует использовать метод `equalsIgnoreCase` класса `String`.

```
public boolean equalsIgnoreCase(String anotherString)
```

Метод `equals` возвращает значение *true* если строка, на которой вызывается метод содержит в точности такую же последовательность символов, как и строка, которая передается методу в качестве аргумента. В противном случае метод вернет значение *false*.

```
boolean flag;  
String s1 = "abc", s2 = "ab";  
flag = s1.equals("ab" + "c"); // flag = true  
flag = s1.equals(s2 + "c"); // flag = true  
flag = s1.equals(new String("a") + "bc"); // flag = true
```

Замечание. Метод `equals` содержит каждый класс, т.к. он определен в классе `Object`. В классе `Object` метод определен таким образом, что результат `x.equals(y)` эквивалентен `x == y`.

Метод `substring`

Класс `String` содержит перегруженный метод **`substring`**, который возвращает подстроку, определяемую значениями индексов начала и конца подстроки, передаваемых в метод в качестве параметров.

```
// возвр. часть строки, начиная с beginIndex:  
public String substring(int beginIndex)
```

```
// возвр. часть строки от beginIndex до endIndex-1:  
public String substring(int beginIndex, int endIndex)
```

Замечание. Метод `substring` выбрасывает исключение `java.lang.IndexOutOfBoundsException` в трех случаях:

- 1) `beginIndex < 0`;
- 2) `beginIndex > endIndex`;
- 3) `endIndex` больше чем длина строки

Лексикографическое сравнение строк

Класс `String` содержит метод `compareTo`, который позволяет лексикографически сравнивать строку, на которой вызывается этот метод, со строкой, которая передается методу в качестве параметра.

```
public int compareTo(String another)
```


Метод возвращает следующие значения:

0, если строка `this` совпадает со строкой `another`;

отрицательное целое число, если `this` лексикографически *предшествует* строке `another`;

положительное целое число, если `this` лексикографически *следует* за строкой `another`.

В случае несовпадения строк результатом является целое число (со знаком), которое определяется следующим образом в зависимости от двух случаев:

1) строки *отличаются в какой то позиции* символом, пусть k – первая слева такая позиция, тогда возвращаемое значение равно **`this.charAt(k)-another.charAt(k);`**

2) строки *имеют разную длину* таким образом, что одна строка является *подстрокой* другой, тогда возвращаемое значение равно **`this.length()-another.length()`**.

```
int k;
```

```
String s1 = "ABC";
```

```
String s2 = "ABE";
```

```
String s3 = "ABCDEF";
```

```
k = s1.compareTo(s2); // k = 'C' - 'E' = -2
```

```
k = s1.compareTo(s3); // k = 3 - 6 = -3
```

Замечание. Метод, который возвращает значение (т. е. не является `void` методом) можно вызывать без присваивания этого значения некоторой переменной. Для некоторых методов такой вызов лишен какого либо смысла, для других же просто игнорирует возвращаемое значение, которое является второстепенным результатом работы метода.

```
String s = "abc";  
int k = s.compareTo("abf"); // k = -3  
  
s.compareTo("abf");  
// метод возвращает значение -3,  
// которое никак не используется
```

Метод concat

Метод `concat` класса `String` *возвращает* строку `this` объединенную со строкой `str`.

```
public String concat(String str)
```

Метод выбрасывает исключение `java.lang.NullPointerException` если `str = null`.

Замечание. Вызов метода `concat` не меняет строку.

```
String s1 = "123";
```

```
String s2 = "567";
```

```
s1.concat(s2);
```

```
System.out.print(s1); // будет выведено 123
```

Метод `replace`

Метод `replace` *возвращает* строку `this` заменяя в ней все вхождения символа `oldChar` на `newChar`.

```
public String replace(char oldChar, char newChar)
```

Замечание. Вызов метода `replace` не меняет строку.

```
String s = "123";  
s1.replace('1', 'A');  
System.out.print(s); // будет выведено 123
```

Копирующий конструктор

```
String s = ...;  
String substr = s.substring(1, 2);
```

Объекты **s** и **substr** внутри себя содержат ссылку на один и тот же массив символов.

Т.о., подстрока из, например, двух символов может содержать внутри себя массив размером на порядки большим, чем количество символов в строке.

```
String substr = new String(s.substring(1, 2));
```

После такого вызова внутренние массивы **s** и **substr** "отвязаны" друг от друга и длина массива у **substr** - 2.