

Объектно-ориентированное программирование на языке Java

Часть 4. Строки в Java

Smykodub Tatyana, NUoS
tetiana.smykodub@nuos.edu.ua
<http://www.berkut.mk.ua>



Класс String

- Во многих языках программирования символьная строка (или просто строка) - это массив символов, **НО** в Java строки - это объекты.

Объекты типа String создаются таким же образом, как и объекты других типов. Для этой цели используется конструктор:

```
String str = new String ("Hello");
```



Класс String

- Каждая строка, создаваемая с помощью оператора `new` или с помощью литерала (заключенная в двойные апострофы), является экземпляром класса `String`.
- Особенностью объекта класса `String` является то, что его значение не может быть изменено после создания объекта при помощи какого-либо метода класса, так как любое изменение строки приводит к созданию нового объекта.
- Класс `String` поддерживает несколько конструкторов, например: `String()`, `String(String str)`, `String(byte[] asciiChar)`, `String(char[] unicodeChar)`, `String(StringBuffer sbuf)`, `String(StringBuilder sbuild)` и др.
- Когда Java встречает литерал, заключенный в двойные кавычки, автоматически создается объект-литерал типа `String`, на который можно установить ссылку. Таким образом, объект класса `String` можно создать, присвоив ссылке на класс значение существующего литерала или с помощью оператора `new` и конструктора, например:

```
String s1 = "oracle.com";
```

```
String s2 = new String("oracle.com");
```

Методы для работы со строками:

- **String concat(String s)** или «+» — слияние строк;
- **boolean equals(Object ob)** и **equalsIgnoreCase(String s)** — сравнение строк с учетом и без учета регистра соответственно;
- **int compareTo(String s)** и **compareToIgnoreCase(String s)** — лексикографическое сравнение строк с учетом и без учета их регистра. Метод осуществляет вычитание кодов первых различных символов вызывающей и передаваемой строки в метод строк и возвращает целое значение. Метод возвращает значение нуль в случае, когда `equals()` возвращает значение `true`;
- **boolean contentEquals(StringBuffer ob)** — сравнение строки и содержимого объекта типа `StringBuffer`;
- **String substring(int n, int m)** — извлечение из строки подстроки длины `m-n`, начиная с позиции `n`. Нумерация символов в строке начинается с нуля;
- **String substring(int n)** — извлечение из строки подстроки, начиная с позиции `n`;
- **int length()** — определение длины строки;

Методы для работы со строками:

- **int indexOf(char ch)** — определение позиции символа в строке;
- **static String valueOf(значение)** — преобразование переменной базового типа к строке;
- **String toUpperCase()/toLowerCase()** — преобразование всех символов вызывающей строки в верхний/нижний регистр;
- **String replace(char c1, char c2)** — замена в строке всех вхождений первого символа вторым символом;
- **String intern()** — заносит строку в «пул» литералов и возвращает ее объектную ссылку;
- **String trim()** — удаление всех пробелов в начале и конце строки;
- **char charAt(int position)** — возвращение символа из указанной позиции (нумерация с нуля);
- **boolean isEmpty()** — возвращает true, если длина строки равна 0;
- **char[] getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)** — извлечение символов строки в массив символов;

Методы для работы со строками:

- **static String format(String format, Object... args), format(Locale l, String format, Object... args)** — генерирует форматированную строку, полученную с использованием формата, интернационализации и др.;
- **String[] split(String regex), String[] split(String regex, int limit)** — поиск вхождения в строку заданного регулярного выражения (разделителя) и деление исходной строки в соответствии с этим на массив строк. Во всех случаях вызова методов, изменяющих строку, создается новый объект типа String.
- Эффективной демонстрацией работы методов класса служит преобразование строки в массив объектов типа String и их сортировка в алфавитном порядке.

Методы для работы со строками:

```
public class SortArray {
    public static void main(String[] args) {
        String names = " Alena Alice alina albina Anastasya ALLA Arina ";
        names = names.trim(); // удаление пробелов по краям строки
        // разбиение строки на подстроки, где "пробел" — разделитель
        String a[] = names.split(" "); // массив содержит элементы нулевой длины
        for (int j = 0; j < a.length-1; j++) {
            for(int i = j + 1; i < a.length; i++) {
                if(a[i].compareToIgnoreCase(a[j]) < 0) {
                    String temp = a[j];
                    a[j] = a[i];
                    a[i] = temp;
                } } }
        for (int j = 0; j < a.length-1; j++) {
            if (!a[j].isEmpty()) {
                System.out.print(a[j] + " ");
            } } }
    }
}
```

Методы для работы со строками:

Вызов метода `trim()` обеспечивает удаление всех начальных и конечных символов пробелов.

Метод `compareToIgnoreCase()` выполняет лексикографическое сравнение строк между собой по правилам Unicode.

Оператор `if(!a[j].isEmpty())` не позволяет выводить пустые строки.



Методы для работы со строками:

При использовании методов класса `String`, изменяющих строку, создается новый обновленный объект класса `String`.

Сохранить произведенные изменения экземпляра класса `String` можно только с применением оператора присваивания, т. е. установкой ссылки на вновь созданный объект.

В следующем примере будет подтвержден тезис о неизменяемости экземпляра типа `String`.



Методы для работы со строками:

```
package RefString;
public class RefString {
    public static void changeStr(String s) {
        s = s.concat(" Certified"); // создается новая строка
        // или s.concat(" Certified");
        // или s += " Certified";    }
    public static void main(String[ ] args) {
        String str = new String("Java");
        changeStr(str);
        System.out.print(str);
    }
}
```

В результате будет выведена строка:

Java

Поскольку объект был передан по ссылке, любое изменение объекта в методе должно сохраняться и для исходного объекта, так как обе ссылки равноправны. Это не происходит по той причине, что вызов метода `concat(String s)` приводит к созданию нового объекта.

Методи для роботи со строками:

```
package by.bsu.strings;  
public class EqualStrings {  
    public static void main(String[] args) {  
        String s1 = "Java";  
        String s2 = "Java";  
        String s3 = new String("Java");  
        String s4 = new String(s1);  
        System.out.println(s1 + "==" + s2 + " : " + (s1 == s2)); // true  
        System.out.println(s3 + "==" + s4 + " : " + (s3 == s4)); // false  
        System.out.println(s1 + "==" + s3 + " : " + (s1 == s3)); // false  
        System.out.println(s1 + " equals " + s2 + " : " + s1.equals(s2)); // true  
        System.out.println(s1 + " equals " + s3 + " : " + s1.equals(s3)); // true  
    }  
}
```

В результате, например, будет выведено:

Java==Java : true

Java==Java : false

Java==Java : false

Java equals Java : true

Java equals Java : true

Методы для работы со строками:

Несмотря на то, что одинаковые по значению строковые объекты расположены в различных участках памяти, значения их кодов совпадают.

Так как в Java все ссылки хранятся в стеке, а объекты — в куче, то при создании объекта `s2` сначала создается ссылка, а затем этой ссылке устанавливается в соответствие объект. В данной ситуации `s2` ассоциируется с уже существующим литералом, так как объект `s1` уже сделал ссылку на этот литерал.

При создании `s3` происходит вызов конструктора, т. е. выделение памяти происходит раньше инициализации, и в этом случае в куче создается новый объект.

Регулярные выражения

- Регулярные выражения позволяют задать шаблон, которому должна соответствовать строка или подстрока.
- Некоторые методы класса String принимают регулярные выражения и используют их для выполнения операций над строками.

Однако не любая строка может быть скомпилирована в регулярное выражение, а только та, которая соответствует правилам написания регулярного выражения – синтаксису, определенному в спецификации языка. Для написания регулярного выражения используются буквенные и цифровые символы, а также метасимволы – символы, имеющие специальное значение в синтаксисе регулярных выражений. Например:

Регулярные выражения

```
String [] s="Java example 1234 else, space ".split(" |, ");
```

Разделители: [пробел] и [,пробел]

```
String [] s="Java | example | 1234 else ".split("\\|");
```

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

Разделитель: [|]

В функция **split** принимает регулярное выражение, а символ | (pipe) используется в регулярных выражениях для задания условия "или", то этот символ нужно экранировать.

Дополнительная информация:

- <https://javarush.ru/groups/posts/regulyarnye-vyrazheniya-v-java>
- <https://metanit.com/java/tutorial/7.4.php>

Классы StringBuilder и StringBuffer

Классы StringBuilder и StringBuffer являются «близнецами» и по своему предназначению близки к классу String,

Содержимое и размеры объектов классов StringBuilder и StringBuffer можно изменять.

Основным отличием StringBuilder от StringBuffer является потокобезопасность последнего. Более высокая скорость обработки класса StringBuilder. Его следует применять, если не существует вероятности использования объекта в конкурирующих потоках.

С помощью соответствующих методов и конструкторов объекты классов StringBuffer, StringBuilder и String можно преобразовывать друг в друга.

Конструктор класса StringBuffer (также как и StringBuilder) может принимать в качестве параметра объект String или неотрицательный размер буфера.

Класс StringBuffer

Объекты этого класса можно преобразовать в объект класса String методом **toString()** или с помощью конструктора класса String.

void setLength(int n) — установка размера буфера;

void ensureCapacity(int minimum) — установка гарантированного минимального размера буфера;

int capacity() — возвращение текущего размера буфера;

StringBuffer append(параметры) — добавление к содержимому объекта строкового представления аргумента, который может быть символом, значением базового типа, массивом и строкой;

Класс StringBuffer

StringBuffer insert(параметры) — вставка символа, объекта или строки в указанную позицию;

StringBuffer deleteCharAt(int index) — удаление символа;

StringBuffer delete(int start, int end) — удаление подстроки;

StringBuffer reverse() — обращение содержимого объекта.

В классе присутствуют также методы, аналогичные методам класса String,

такие как **replace()**, **substring()**, **charAt()**, **length()**, **getChars()**, **indexOf()** и др.

Класс StringBuffer

```
package by.bsu.strings;

public class DemoStringBuffer {

    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        System.out.println("длина —> " + sb.length());
        System.out.println("размер —>" + sb.capacity());
        // sb = "Java"; // ошибка, только для класса String
        sb.append("Java");
        System.out.println("строка —> " + sb);
        System.out.println("длина —> " + sb.length());
        System.out.println("размер —> " + sb.capacity());
        sb.append("Internationalization");
        System.out.println("строка —> " + sb);
        System.out.println("длина —> " + sb.length());
        System.out.println("размер —> " + sb.capacity());
        System.out.println("реверс —> " + sb.reverse());
    }
}
```

Класс StringBuffer

длина —> 0

размер —> 16

строка —> Java

длина —> 4

размер —> 16

строка —> JavaInternationalization

длина —> 24

размер —> 34

реверс —> noitazilanoitanretnlaval

При создании объекта StringBuffer конструктор по умолчанию автоматически резервирует некоторый объем памяти (16 символов), что в дальнейшем позволяет быстро менять содержимое объекта, оставаясь в границах участка памяти, выделенного под объект. Размер резервируемой памяти при необходимости можно указывать в конструкторе. Если длина строки StringBuffer после изменения превышает его размер, то емкость объекта автоматически увеличивается, оставляя при этом некоторый резерв для дальнейших изменений. Методом `reverse()` можно быстро изменить порядок символов в объекте.

Класс StringBuffer

Если метод, вызываемый объектом `StringBuilder`, производит изменения в его содержимом, то это не приводит к созданию нового объекта, как в случае объекта `String`, а изменяет текущий объект `StringBuilder`.

```
package by.bsu.strings;

public class RefStringBuilder {
    public static void changeStr(StringBuilder s) {
        s.append(" Certified");
    }

    public static void main(String[] args) {
        StringBuilder str = new StringBuilder("Oracle");
        changeStr(str);
        System.out.println(str);
    }
}
```

В результате выполнения этого кода будет выведена строка:

Oracle Certified

Класс StringBuffer

Для классов `StringBuffer` и `StringBuilder` не переопределены методы `equals()` и `hashCode()`, т. е. сравнить содержимое двух объектов невозможно, следовательно хэш-коды всех объектов этого типа вычисляются так же, как и для класса `Object`. При идентичном содержимом у двух экземпляров, размеры буфера каждого могут отличаться, поэтому сравнение на эквивалентность объектов представляется неоднозначным.

```
package by.bsu.strings;

public class EqualsStringBuffer {
    public static void main(String[ ] args) {
        StringBuffer sb1 = new StringBuffer();
        StringBuffer sb2 = new StringBuffer(48);
        sb1.append("Java");
        sb2.append("Java");
        System.out.print(sb1.equals(sb2));
        System.out.print(sb1.hashCode() == sb2.hashCode());
    }
}
```

Класс StringBuffer

Результатом выполнения данной программы будет дважды выведенное значение

false.

Сравнить содержимое можно следующим образом:

`sb1.toString().contentEquals(sb2)`



Новые фиши Java SE 11:

java.lang.String

- `boolean isBlank ()`: возвращает `true`, если строка пуста или содержит только пробелы, иначе `false`.
- `Stream lines()`: возвращает поток строк, извлеченных из этой строки, разделенных терминаторами строк.
- `String repeat (int)`: возвращает строку, значение которой представляет собой конкатенацию этой строки, повторяющуюся *int* раз.
- `String strip ()`: Возвращает строку, из которой удалены все пробелы, которые находятся до первого символа, не являющегося пробелом, или после последнего.

Новые фиши Java SE 11: `java.lang.String`

- `String stripLeading ()`: Возвращает строку, из которой удалены все пробелы, которые находятся до первого символа, не являющегося пробелом.
- `String stripTrailing ()`: Возвращает строку, из которой удалены все пробелы, которые находятся после последнего символа, не являющегося пробелом.
- Что-то похожее на `strip()` уже делал метод `trim ()`, только вот под пробелами эти методы понимают разные вещи. В случае `trim()` отсекаются только пробелы, а в `strip()` — ещё и спецсимволы, вроде табуляции.

Новые фичи Java SE 11: `java.lang.StringBuffer`

`java.lang.StringBuffer`

`java.lang.StringBuilder`

Оба этих класса содержат новый метод `compareTo()`, который принимает `StringBuffer/StringBuilder` и возвращает `int`. Метод лексического сравнения аналогичен новому методу `compareTo()` `CharSequence`.