Строки Java

Строки

это **объекты**, хранящие в себе упорядоченная последовательно
символов.

В Java строки являются объектами **класса java.lang.String**.
До java 9 строки хранятся как массив char[], каждый символ занимает 2
Кодировка UTF-16

Начиная с 9 версии (компактные строки) строки хранятся как массив by
кодировке LATIN-1 (если символы занимают по 1му байту) или UTF-16 (е
символы занимают 2 байта). При этом класс String хранит информацию
кодировке.

Объекты класса String нельзя изменить,
можно только создать новую строку на основе старой

Строки

1. Одинаковые строковые литералы всегда ссылаются на
   и тот же экземпляр класса String .
2. Экземпляры класса String , вычисленные во время
   выполнения, создаются заново, автоматически в пуле н
   участвуют и потому различны.
3. Строковые литералы в константных выражениях
   вычисляются на этапе компиляции и затем расцениваю
   как обычные литералы.

Создание строк

Для создания новой строки можно использовать один из конструкторов класса String, либо напрямую присвоить строку в двойных кавычках:

```
String str1 = "string1";    // Объявляем и создаем строку.
System.out.println(str1); // string1

String str2 = new String("string2");   // Строки это объекты, строк
                                       быть создана при помощи ключевого слова n
System.out.println(str2); // string2
```

Создание строк

Когда компилятор видит **строковый литерал**, например "Small
в коде, он создаёт **объект класса java.lang.String**.

Строки **можно создавать с помощью форматирования**:

```
String s1 = String.format("String with int: %02d, float: %.2f", 1,
System.out.println(s1); // String with int: 01, float: 1,50
```

Тоже самое можно сразу вывести в консоль:

```
System.out.printf("String with int: %02d, float: %.2f", 1, 1.5F); /
                                                    int: 01, float: 1,50
```

Методы строк

Многие методы строки принимают в качестве параметра или возвращают в качестве результата индекс. Индексация начинае нуля. Первый char (имеется в виду char code unit, а не символ) в строке имеет индекс 0, а последний имеет индекс length() — 1. I переданный в параметр индекс выходит за пределы строки, то генерируют исключение java.lang.IndexOutOfBoundsException.

Некоторые методы принимают в качестве параметров начальны индекс и конечный индекс. В этом случае начальный индекс включается в отрезок, а конечный индекс исключается

Сравнение строк

Класс String не имеет специальной поддержки для == , поэтом
сравнения строк нужно использовать специальные методы.

1. Метод **.equals()**

```
String srt1 = "string";
String srt2 = new String("string");
String srt3 = "String";
System.out.println(srt1.equals(srt2));   // true
System.out.println(srt1.equals(srt3));   // false
```

Сравнение строк

```
String srt1 = "string";
String srt2 = new String("string");
String srt3 = "String";
```

Сравнение без учета регистра - метод **.equalsIgnoreCase()**

```
System.out.println(srt1.equalsIgnoreCase(srt3)); // true
```

Сравнение начала строки и конца - методы **.startsWith()** и **.end**

```
System.out.println(srt2.startsWith("ST")); // false
System.out.println(srt2.endsWith("g")); // true
```

Сравнение строк

```
String srt1 = "string";
String srt2 = new String("string");
String srt3 = "str";
String srt4 = "String string";
```

compareToIgnoreCase() - лексикографическое сравнение, игнорирующее разницу в регистрах. букв

Для определения порядка строк используется метод **compareT**

Если строки одинаковы, то результат будет 0

```
System.out.println(srt1.compareTo(srt2)); // 0
```

Результат будет положительным, если вызывающая строка боль
строки, переданной в качестве параметра

```
System.out.println(srt2.compareTo(srt3)); // 3
```

Результат будет отрицательным, если вызывающая строка мень
строки, переданной в качестве параметра;

```
System.out.println(srt4.compareTo(srt1)); // -32
```

Сортировка строк

```
String srt1 = "string";
String srt2 = new String("String string");
String srt3 = "str";

String[] strArr = {srt1, srt2, srt3};

Arrays.sort(strArr); // Сортируем массив со строками

System.out.println(Arrays.toString(strArr)); // Отсортировали
                                              алфавиту
```

Методы работы со строками

Взятие подстроки

.substring(int beginIndex)

.substring(int beginIndex, int endIndex)

Возвращает подстроку, начинающуюся с beginIndex (включител
до endIndex (исключительно) или конца строки.

Преобразование в массив

. split(String regex) - разбивает строку на массив строк по регу
выражению.

Поиск подстроки

.contains(String s) - ищет подстроку s в строке

Методы работы со строками

trim() - возвращает строку, в которой убраны пробелы в начале
конце строки.
replaceAll(String regex, String replacement) - возвращает стр
которой все вхождения подстрок согласно регулярному выраже
заменены на replacement.
replace(CharSequence target, CharSequence replacement) -
возвращает строку, в которой все вхождения последовательнос
символов target заменены на replacement.
join(CharSequence delimiter, CharSequence... elements) - об
несколько CharSequence в одну строку, используя в качестве
разделителя delimiter. (Java 8)

Классы StringBuffer и StringBuilder

Класс **StringBuffer** представляет расширяемые и доступные дл
изменений последовательности символов, позволяя вставлять
символы и подстроки в существующую строку и в любом месте.

Класс **StringBuilder** идентичен классу StringBuffer и обладает б
производительностью. Однако он не синхронизирован, поэтому
не нужно использовать в тех случаях, когда к изменяемой строк
обращаются несколько потоков.

Конкатенация строк

Для **конкатенации** (склеивания) строк можно использовать оператор + также методы классов StringBuffer / StringBuilder.

```
String str1 = "Begin";
for (int i = 0; i < 1000; i++) {
    // Каждый append
    будет создавать новый объект String
    и продолжать работу с ним
    str1 += " Промежуток " + i + " ";
}
str1 += " Окончание ";
System.out.println(str1);
```

String += можно использовать для конкатенации нескольких строк,
но не для конкатенации сложных строк

Конкатенация строк

Для **конкатенации** (склеивания) строк можно использовать оператор + также методы классов StringBuffer / StringBuilder.

```
String str2 = "Begin";
StringBuilder sb = new StringBuilder();
sb.append(str2);
for (int i = 0; i < 1000; i++) {
    // добавляет к предыдущей
    sb.append(" Промежуток ").append(i).append(" ");
}
sb.append(" Окончание ");
//  StringBuilder создает строку только при вызове метода toString()
System.out.println(sb.toString());
```

для конкатенации сложных строк (особенно в циклах) лучше использовать StringBuilder