Строки. Класс String.

Любая строка в языке Java является объектом класса String. Класс String представляет набор символов(массив char) и используется для неизменяемых (immutable) строк. Это касается в том числе и неименованных констант. Поэтому вполне допустимой является запись вида:

char **x** = **"Hello World".**charAt(5);

Метод charAt получает символ, находящийся в строке на указанной позиции. В нашем примере в переменную **x** попадет значение ' ', то есть пробел, так как нумерация начинается с нуля.

Как уже было сказано, если строка не должна меняться, можно использовать класс **String**, если в строку необходимо часто вносить изменения, следует использовать класс **StringBuffer**.

Большинство методов изменяющих строки возвращает новую строку.

Пример:

String **s** = "Test String";

String s1 = **s**.substring(5);

Метод substring отрезает фрагмент строки начиная с указанной позиции. В результате выполнения этих операций строка в переменной **s** не изменится, а в **s1** попадет строка "String".

Можно занести в переменную **s** результат работы метода:

String **s** = "Test String";

s = **s**.substring(5);

Но все равно надо помнить, что исходная строка не изменится, а в переменную будет занесен новый объект.

Также всегда следует учитывать, что строки не являются примитивными типами и их нельзя сравнивать обычными операторами сравнения, такими как **==**. Если надо сравнить две строки, следует воспользоваться методом **equals** или **compareTo**.

Класс String имеет следующие основные методы:

- char **charAt**(int index) – возвращает символ, находящийся в указанной позиции;
- **compareTo**(String anotherString) – сравнивает строку с другой строкой (сравнивается именно текст содержащийся в строках);
- int **compareToIgnoreCase**(String str) – то же самое, но без учета регистра;
- String **concat**(String str) – возвращает объект строки, содержащий сумму данной строки, и переданной как аргумент;
- boolean **contentEquals**(StringBuffer sb) – сравнивает содержимое объекта String и StringBuffer;

- boolean **endsWith**(String suffix) – проверяет, завершается ли строка заданным суффиксом (совпадает ли конец строки с переданной);
- boolean **equals**(Object anObject) – сравнивает строку с объектом boolean;
- **equalsIgnoreCase**(String anotherString) – сравнивает строки игнорируя регистр;
- byte[] **getBytes**()- возвращает строку в виде массива байтов (в качестве входного параметра указывается название требуемой кодировки);
- int **indexOf**(int ch) – ищет в строке переданный символ и возвращает позицию первого совпадения. Может иметь два параметра, тогда вторым указывается номер, с которого надо начинать поиск. Вместо символа может быть также строка;
- int **lastIndexOf**(int ch) – аналогично предыдущему, но поиск выполняется с конца;
- int **length**()- возвращает длину строки;
- String **replace**(char oldChar, char newChar) – возвращает строку, где все символы, совпадающие с первым, заменены вторым;
- boolean **startsWith**(String prefix) – проверяет начинается ли строка с данного префикса (совпадает ли начало строки с переданной);
- String **substring**(int beginIndex) – возвращает строку, содержащую фрагмент данной строки. Может иметь два параметра: номер первого символа и номер последнего символа (второй может отсутствовать);
- char[] **toCharArray**() – возвращает массив символов, содержащий данную строку;
- String **toLowerCase**() – возвращает строку, содержащую копию данной, приведенную к нижнему регистру;
- String **trim**() – возвращает строку с удаленными начальными и конечными пробелами;
- 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 n) – возвращает строку, значение которой представляет собой конкатенацию этой строки, повторяющуюся n раз;

- String **strip**() – возвращает строку, из которой удалены все пробелы, которые находятся до первого символа, не являющегося пробелом, или после последнего;
- String **stripLeading**() – возвращает строку, из которой удалены все пробелы, которые находятся до первого символа, не являющегося пробелом;
- String **stripTrainling**() – возвращает строку, из которой удалены все пробелы, которые находятся после последнего символа, не являющегося пробелом.

Примеры работы со строками

Задача: ввести строку и подсчитать количество запятых в ней.

Данная задачаможет быть решена двумя способами: перебором всех элементов строки, либо с использованием метода поиска.

Перебор всех элементов строки будет выглядеть следующим образом:

```
String str = "Тестовая, строка, с несколькими,, запятыми";
int n = 0;
char symbol;
for (int i=0;i < str.length(); i++){
        symbol = str.charAt(i);
                if (symbol == ',') {
                        n++;
                }
}
System.out.println("У нас есть " + n + " запятых");
```

Сначала создается строка str, **n** – количество запятых в строке, так как изначально запятые могут отсутствовать, начальным значением будет 0. С помощью charAt получаем каждый элемент строки и сравниваем его с запятой, увеличивая n, если обнаружено совпадение.

Подсчет с помощью поиска будет выглядеть следующим образом:

```
String str = "Тестовая, строка, с несколькими,, запятыми";
int n = 0;
int p = 0;
while (p != -1) {
        p = str.indexOf(',',p);
        if (p != -1) {
                p++;
                n++;
        }
}
System.out.println("У нас есть " + n + " запятых");
```

Подсчет заканчивается, если indexOf вернула -1, если возвращено другое число – значит, что в тексте найдена запятая, в этом случае увеличивается счетчик **n** и увеличивается p для того, чтобы следующий поиск выполнялся после найденного знака.

Работа со строками String, StringBuffer и StringBuilder

Конструкторы StringBuffer:

- StringBuffer();
- StringBuffer(String str).

Конструкторы StringBuilder:

- StringBuilder ();
- StringBuilder(String str).

Базовым классом для работы со строковыми данными является **String**, но его использование не всегда оправдано. Связано это в первую очередь с тем, что строка, содержащаяся в объекте **String**, не может меняться. При изменении содержимого строки создается новый объект. В ряде случаев это может привести к большим потерям в производительности. В частности, операция вида: **str += " добавление строки";** приводит к тому, что создается новый объект, и содержимое обеих исходных строк в него копируется. Если подобные операции используются в больших количествах или в цикле, то это может привести экспоненциальному падению производительности операций.

Для решения этой проблемы следует использовать объект типа **StringBuffer** или **StringBuilder**. Оба эти объекта позволяют менять содержимое находящихся в них строк. С их использованием операция примет вид:

strBuilder.append(" добавление строки");

В этом случае не создается новый объект и копируется только добавляемая строка.

Сходства и отличия StringBuffer и StringBuilder

Оба класса имеют одинаковый набор методов и используются в сходных ситуациях, за одним исключением. **StringBuilder** не рассчитан на использование в многопоточных приложениях и может приводить к ошибкам, если используется в нескольких потоках одновременно. С другой стороны, отсутствие синхронизации увеличивает скорость его работы.

Таким образом, если вы уверены, что ваш объект будет использоваться только в одном потоке, желательно использовать **StringBuilder**, в противном случае больше подходит **StringBuffer**.