Практическая работа №7

Работа со строками

Строка представляет собой последовательность символов. Для работы со строками в Java определен класс String, который предоставляет ряд методов для манипуляции строками. Физически объект String представляет собой ссылку на область в памяти, в которой размещены символы.

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

```
String str1 = "Java";
String str2 = new String(); // пустая строка
String str3 = new String(new char[] {'h', 'e', 'l', 'l', 'o'});
System.out.println(str1); // Java
System.out.println(str2); //
System.out.println(str3); // hello
```

При работе со строками важно понимать, что объект String является неизменяемым (immutable). Это означает, что при любых операциях над строкой, которые изменяют эту строку, фактически будет создаваться новая строка.

Поскольку строка рассматривается как набор символов, то мы можем применить метод length() для нахождения длины строки или длины набора символов:

```
String str = "Java";
System.out.println(str.length()); // 4
```

А с помощью метода toCharArray() можно обратно преобразовать строку в массив символов:

```
String str = new String(new char[] {'h', 'e', 'l', 'l', 'o'});
char[] helloArray = str.toCharArray();
```

Строка может быть пустой – ей можно присвоить пустые кавычки или удалить из строки все символы. В этом случае длина строки, возвращаемая методом length(), равна 0:

```
String s = "";   // строка не содержит символов
System.out.println(s.length()); // 0
```

Класс String имеет специальный метод, который позволяет проверить строку на пустоту – isEmpty(). Если строка пуста, он возвращает true:

```
String s = "";   // строка не содержит символов
if (s.isEmpty()) {
    System.out.println("String is empty");
}
```

Переменная String может не указывать на какой-либо объект и иметь значение null:

```
String s = null;   // строка не указывает на объект
if(s == null) {
    System.out.println("String is null");
}
```

Значение null не эквивалентно пустой строке. Например, в следующем случае мы столкнемся с ошибкой выполнения:

```
String s = null;   // строка не указывает на объект
if (s.length() == 0) {
    System.out.println("String is empty"); // ошибка
}
```

Так как переменная s не указывает на объект String, мы не можем обращаться к методам объекта String. Чтобы избежать подобных ошибок, можно предварительно проверять строку на null:

```
String s = null;   // строка не указывает на объект
if (s!=null && s.length() == 0) {
    System.out.println("String is empty"); // ошибка отсутствует
}
```

Основные операции со строками раскрывается через методы класса String, среди которых можно выделить следующие:

- concat() – объединяет строки;
- valueOf() – преобразует объект в строковый вид;
- join() – соединяет строки с учетом разделителя;
- compare() – сравнивает две строки;
- charAt() – возвращает символ строки по индексу;
- getChars() – возвращает группу символов;
- equals() – сравнивает строки с учетом регистра;
- equalsIgnoreCase() – сравнивает строки без учета регистра;
- regionMatches() – сравнивает подстроки в строках;
- indexOf() – находит индекс первого вхождения подстроки в строку;
- lastIndexOf() – находит индекс последнего вхождения подстроки в строку;
- startsWith() – определяет, начинается ли строка с подстроки;
- endsWith() – определяет, заканчивается ли строка на определенную подстроку;
- replace() – заменяет в строке одну подстроку на другую;
- trim() – удаляет начальные и конечные пробелы;

- substring() – возвращает подстроку, начиная с определенного индекса до конца или до определенного индекса;
- toLowerCase() – переводит все символы строки в нижний регистр;
- toUpperCase() – переводит все символы строки в верхний регистр.

Для объединения строк можно использовать операцию сложения строк:

```
String str1 = "Hello,";
String str2 = "world!";
String str3 = str1 + " " + str2;
System.out.println(str3); // Hello,world!
```

При этом, если в операции сложения строк используется нестроковый объект, например, число, то этот объект преобразуется к строке:

```
String str3 = "Год " + 2015;
```

Фактически при сложении строк с нестроковыми объектами будет вызываться метод valueOf() класса String. Данный метод имеет множество перегрузок и преобразует практически все типы данных к строке.

Другой способ объединения строк представляет метод concat():

```
String str1 = "Hello,";
String str2 = "world!";
str2 = str1.concat(str2); // Hello,world!
```

Метод concat() принимает строку, с которой надо объединить вызывающую строку, и возвращает соединенную строку.

Еще один метод объединения - метод join() позволяет объединить строки с учетом разделителя. Например, выше две строки сливались в одно слово "Hello,world!", но в идеале две подстроки должны быть разделены пробелом:

```
String str1 = "Hello,";
String str2 = "world!";
String str3 = String.join(" ", str1, str2); // Hello, world!
```

Первым параметром идет разделитель, которым будут разделяться подстроки в общей строке, а все последующие параметры передают через запятую произвольный набор объединяемых подстрок - в данном случае две строки, хотя их может быть и больше.

Для извлечения символов по индексу в классе String определен метод charAt(). Он принимает индекс, по которому надо получить символов, и возвращает извлеченный символ. Как и в массивах, индексация в строках начинается с нуля:

```
String str = "Java";
char c = str.charAt(2);
System.out.println(c); // v
```

Если надо извлечь сразу группу символов или подстроку, то можно использовать метод getChars(). Он принимает следующие параметры:

- srcBegin: индекс в строке, с которого начинается извлечение символов;
- srcEnd: индекс в строке, до которого идет извлечение символов;
- dst: массив символов, в который будут извлекаться символы;
- dstBegin: индекс в массиве dst, с которого надо добавлять извлеченные из строки символы:

```
String str = "Hello, world!";
int start = 6;
int end = 12;
char[] dst=new char[end - start];
str.getChars(start, end, dst, 0);
System.out.println(dst); // world
```

В отличие от сравнения данных примитивных типов, для сравнения строк не применяется знак равенства ==. Для сравнения строк с учетом регистра используются метод equals(), без учета регистра – метод equalsIgnoreCase(). Оба метода в качестве параметра принимают строку, с которой будет осуществляться сравнение:

```
String str1 = "Java";
String str2 = "java";
System.out.println(str1.equals(str2)); // false
System.out.println(str1.equalsIgnoreCase(str2)); // true
```

Еще один специальный метод regionMatches() сравнивает отдельные подстроки в рамках двух строк. Он имеет следующие формы:

- boolean regionMatches(int toffset, String other, int oofset, int len);
- boolean regionMatches(boolean ignoreCase, int toffset, String other, int oofset, int len).

Метод принимает следующие параметры:

- ignoreCase – определяет, игнорировать регистр символов при сравнении. Если параметр имеет значение true, регистр игнорируется;
- toffset – индекс в вызывающей строке, с которого начнется сравнение;
- other – строка, с которой сравнивается вызывающая;
- offset – индекс в сравниваемой строке, с которого начнется сравнение;
- len – количество сравниваемых символов в обеих строках.

```
String str1 = "Hello, world!";
String str2 = "I work";
boolean result = str1.regionMatches(7, str2, 2, 3);
System.out.println(result); // true
```

В данном случае метод сравнивает 3 символа с 7-го индекса первой строки ("wor") и 3 символа со 2-го индекса второй строки ("wor"). Так как эти подстроки одинаковы, то возвращается true.

Еще одна пара методов compareTo() и compareToIgnoreCase() также позволяет сравнить две строки, но при этом узнать, больше ли одна строка, чем другая, или нет. Если возвращаемое значение положительное, то первая строка больше второй, если отрицательное – вторая больше первой, если рано нулю – строки равны. Для определения, больше или меньше одна строка, чем другая, используется лексикографический порядок. Например, строка "А" меньше, чем строка "Б", так как символ 'А' в алфавите стоит перед символом 'Б'. Если первые символы строк равны, то в расчет берутся следующие символы:

```
String str1 = "hello";
String str2 = "world";
String str3 = "hell";
System.out.println(str1.compareTo(str2)); // -15 - str1 меньше, чем strt2
System.out.println(str1.compareTo(str3)); // 1 - str1 больше, чем str3
```

Метод indexOf() находит индекс первого вхождения подстроки в строку, а метод lastIndexOf() - индекс последнего вхождения. Если подстрока не будет найдена, то оба метода возвращают -1:

```
String str = "Hello world";
int index1 = str.indexOf('l'); // 2
int index2 = str.indexOf("wo"); //6
int index3 = str.lastIndexOf('l'); //9
```

Метод startsWith() позволяют определить начинается ли строка с определенной подстроки, а метод endsWith() позволяет определить заканчивается строка на определенную подстроку:

```
String str = "myfile.exe";
boolean start = str.startsWith("my"); //true
boolean end = str.endsWith("exe"); //true
```

Метод replace() позволяет заменить в строке одну последовательность символов на другую:

```
String str = "Hello, world!";
String replStr = str.replace("Hello", "Bye"); // Bye, world!
```

Метод trim() позволяет удалить начальные и конечные пробелы:

```
String str = "  hello, world!  ";
str = str.trim(); // hello, world!
```

Метод substring() возвращает подстроку, начиная с определенного индекса до конца или до определенного индекса:

```
String str = "Hello world";
String substr1 = str.substring(6); // world
String substr2 = str.substring(0, 4); //Hell
```

Метод toLowerCase() переводит все символы строки в нижний регистр, а метод toUpperCase() - в верхний:

```
String str = "Hello, World!";
System.out.println(str.toLowerCase()); // hello, world!
System.out.println(str.toUpperCase()); // HELLO, WORLD!
```

Метод split() позволяет разбить строку на подстроки по определенному разделителю. Разделитель - какой-нибудь символ или набор символов передается в качестве параметра в метод. Например, разобьем текст на отдельные слова:

```
String text = "our work is never over";
String[] words = text.split(" ");
for(String word : words){
    System.out.println(word);
}
```

В данном случае строка будет разделяться по пробелу. Консольный вывод:

```
our
work
is
never
over
```

Практические задания

1. Дано ошибочно написанное слово «алигортм». Путем перемещения его букв получить слово «алгоритм».
2. Дано предложение. Определить количество вхождений в него заданного символа.
3. Вывести на экран только те буквы заданного слова, которые встречаются в нём только один раз.
4. Палиндромом называется слово или фраза, которая читается одинаково как с начала, так и с конца. Проверить, является ли палиндромом:

   а) заданное слово;
   б) заданное предложение (без учёта пробелов).
5. Дана последовательность слов. Проверить, правильно ли в ней записаны буквосочетания «жи» и «ши». Если есть ошибки, подсчитать их количество и исправить.